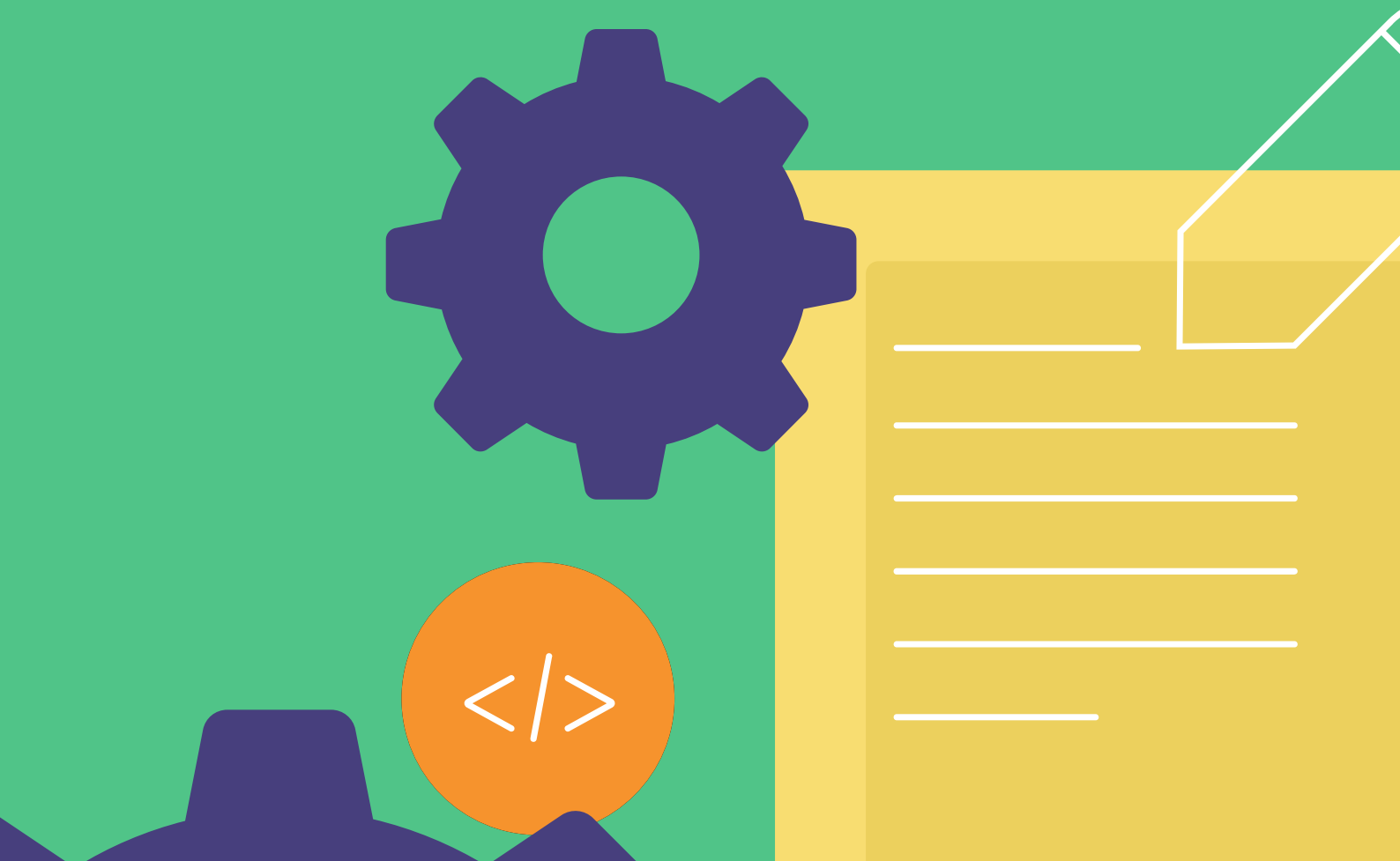
FORMCENTRIC

Entwicklerhandbuch

Version 7.1.0



Formcentric für FirstSpirit: Entwicklerhandbuch

Breite Str. 61, 22767 Hamburg
Deutschland

Einschränkung der Gewährleistung

Inhaltliche Änderungen des Handbuchs und der Software behalten wir uns ohne Ankündigung vor. Es wird keine Haftung für die Richtigkeit des Inhalts des Handbuchs oder Schäden, die sich aus dem Gebrauch der Software ergeben, übernommen.

Warenzeichen

Innerhalb dieses Handbuchs wird auf Warenzeichen Bezug genommen, die nicht explizit als solche ausgewiesen sind. Aus dem Fehlen einer Kennzeichnung kann nicht geschlossen werden, dass ein Name frei von Rechten Dritter ist.

1. Einleitung

Dieses Handbuch beschreibt, wie die Formularmanagererweiterung Formcentric installiert, konfiguriert und erweitert wird. Es richtet sich an Administratoren und Entwickler. Für das vollständige Verständnis des Textes benötigen Sie Kenntnisse im Bereich der Administration und Bedienung von FirstSpirit sowie im Bereich der Java-Softwareentwicklung.

Kapitel 4, *Installation und Konfiguration* : beschreibt die Schritte, die Sie bei der Installation und Konfiguration von Formcentric ausführen müssen. In der Installationsanleitung wird davon ausgegangen, dass Sie Formcentric Analytics einsetzen. Sollte dies nicht der Fall sein, so können Sie alle Punkte, die Formcentric Analytics betreffen, auslassen.

Kapitel 5, *Erweiterung des FirstSpirit-Projekts* : beschreibt die Ergänzungen und Modifikationen, die Sie innerhalb eines FirstSpirit-Projekts vornehmen.

Kapitel 6, *Programmierung und Anpassung* : zeigt Ihnen, wie Sie Formcentric um zusätzliche Funktionen erweitern können.

1.1. Begriffsdefinitionen

In diesem Handbuch werden folgende Begriffe verwendet:

Begriff	**Beschreibung**
Redakteur	Person, die Formulare erstellt und bearbeitet.
Benutzer	Person, die ein Formular ausfüllt.
Formular	Im Browser dargestelltes HTML-Webformular.
Formularelemente	Alle Elemente, aus denen ein Formular zusammengesetzt wird (Eingabefelder, Auswahllisten, Checkboxen, et cetera).
Formulareditor	Eingabekomponente im FirstSpirit SiteArchitect oder ContentCreator, mit der Formulare angelegt und bearbeitet werden können.
Formulardaten	Die vom Benutzer in das Formular eingegebenen Daten.
Editor	FirstSpirit SiteArchitect oder ContentCreator
Formcentric-Client	React-Anwendung für die Darstellung der Formulare im Browser.

2. Übersicht

Auf der Redaktionsseite erweitert Formcentric das FirstSpirit-System um eine Eingabekomponente, mit der Redakteure dynamische Webformulare erstellen und bearbeiten können.

Für die Darstellung der Formulare und die Verarbeitung der abgesendeten Formulardaten stehen zwei zusätzliche Webapp-Module zur Verfügung. Dabei handelt es sich zum einen um eine klassische Webanwendung mit serverseitiger Erzeugung der HTML-Ausgabe (im Folgenden *Spring-MVC-Webanwendung* genannt) und zum anderen um eine moderne Headless-Anwendung mit clientseitigem Rendering. Beide Webanwendungen enthalten verschiedene Spring-Controller für die Verarbeitung der Daten. Ein Formular-Controller validiert die empfangenen Daten und reicht sie an spezifische Actions weiter, die die abschließende Verarbeitung durchführen. Auf diese Weise können verschiedene Backend-Systeme wie Mail-Server, Formcentric Analytics oder Datenbanken angebunden werden.

Die in Formcentric enthaltene Analytics-Komponente ermöglicht es, die abgesendeten Formulardaten zu speichern und auszuwerten. Formcentric Analytics besteht aus zwei globalen Webapp-Modulen. Das Backend-Webapp-Modul ist für das Speichern der Daten in einer relationalen Datenbank zuständig. Hierfür stellt es eine REST-Schnittstelle zur Verfügung, über die Clients mit dem Backend kommunizieren können. Neben den reinen Formulardaten speichert das Backend auch die Formular-Sessions, sofern dies für das jeweilige Formular aktiviert ist.

Das Reporting-Webapp-Modul ist eine moderne Single-Page-Anwendung, mit der die im Backend gespeicherten Formulardaten angezeigt, gelöscht und exportiert werden können.

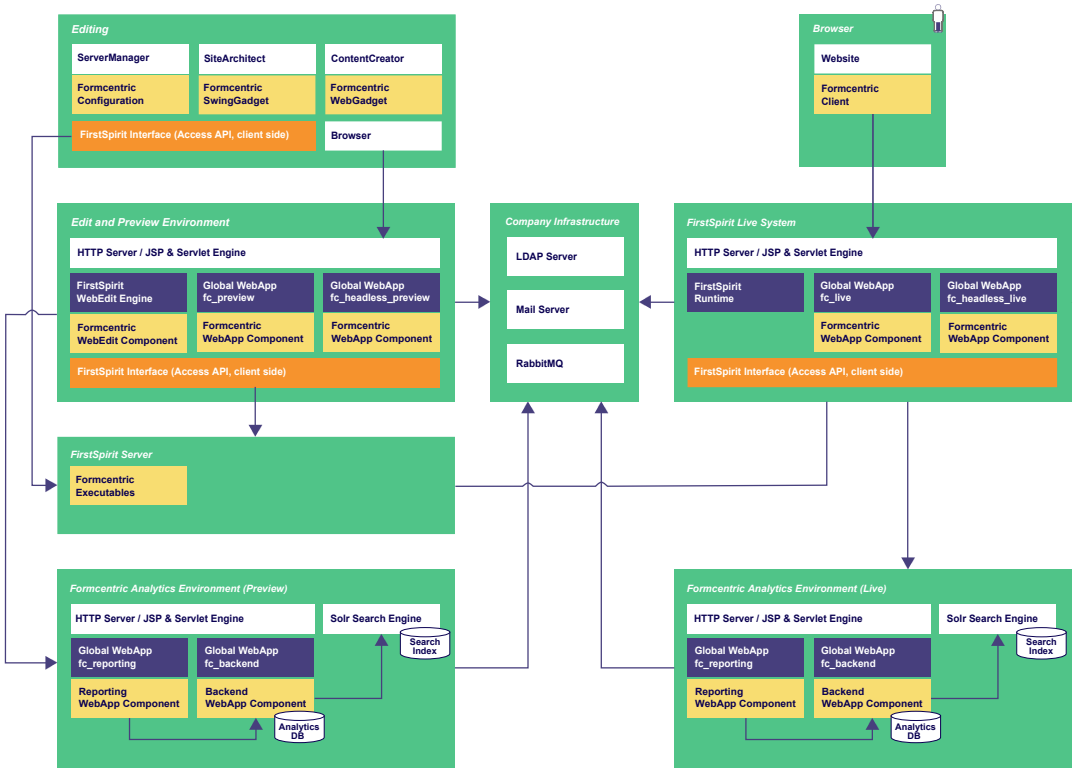


Abbildung 2.1. Architekturübersicht

3. Systemvoraussetzungen

Um Formcentric 7.1.0 nutzen zu können, müssen Sie FirstSpirit 5.2.201209 oder höher einsetzen.

Formcentric benötigt das Javascript-Framework „jQuery“ ab Version 1.12.4.

Für den Fall, dass die PDF-Action die benötigten PDF-Vorlagendokumente aus einem lokalen Verzeichnis lädt, wird auf dem FirstSpirit-Server das externe Programm *rsync* benötigt. Weitere Informationen zur Installation und Konfiguration von „rsync“ im Zusammenhang mit FirstSpirit finden Sie im Kapitel 10 der FirstSpirit „Dokumentation für Administratoren“.

3.1. Java

Formcentric kann mit den folgenden Java-Versionen betrieben werden. Die Ausführung muss innerhalb eines Servlet-Containers, der die Spezifikationen JSP 2.3 und Java Servlet 3.1 unterstützt, erfolgen.

Java	**Status**
OpenJDK 17	supported *(recommended)*

Die Anforderungen an die Java-Version sind immer auch abhängig von der eingesetzten FirstSpirit-Version. Ab FirstSpirit 2023-11 wird mindestens Java 17 benötigt.

3.2. FirstSpirit

Sie können Formcentric mit den folgenden FirstSpirit Versionen verwenden. Für den Fall, dass Sie eine andere Version betreiben, kann eine Anpassung der FirstSpirit Runtime Java-Library erforderlich sein (siehe Abschnitt 4.1, „Installation des Moduls Formcentric“). Dies ist in jedem Fall notwendig, wenn Sie eine neuere Version einsetzen.

FirstSpirit	**Status**
FirstSpirit 2402.11 - 2404.08 (Isolated Mode)	supported *(recommended)*

4. Installation und Konfiguration

Installieren und konfigurieren Sie Formcentric über die *Server- und Projektkonfiguration* von FirstSpirit.

4.1. Installation des Moduls Formcentric

Wählen Sie dazu im Bereich *Server-Eigenschaften* den Menüeintrag *Module* aus. Klicken Sie anschließend auf den Button *Installieren*. Dieser öffnet einen Dateiauswahldialog, in dem Sie die zu installierende Archivdatei *formcentric-7.1.0.fsm* auswählen.

Das Formcentric-Modul enthält einen Dienst für die Abfrage von Lizenzinformationen. Dieser Dienst wird für die Installation und den Betrieb von Formcentric benötigt. Nach dem Laden der Archivdatei werden Sie daher vom System gefragt, ob Sie die im Modul enthaltenen Dienste automatisch starten möchten. Beantworten Sie diese Frage mit *Ja*.

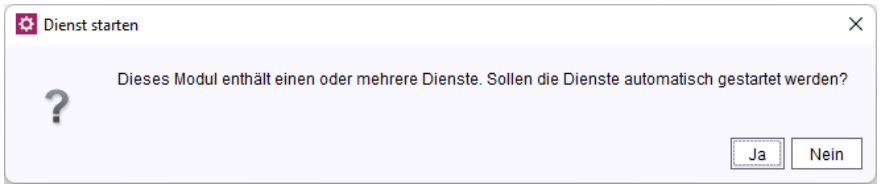


Abbildung 4.1. Dienste automatisch starten

Sobald Sie die Datei erfolgreich installiert haben, zeigt Ihnen das System das Modul *Formcentric (I)* mit den darin enthaltenen Komponenten an.

Wählen Sie nun den neuen Eintrag *Formcentric (I)* aus, klicken Sie auf *Konfigurieren*, aktivieren Sie die Checkbox *Alle Rechte* und bestätigen Sie Ihre Änderung.

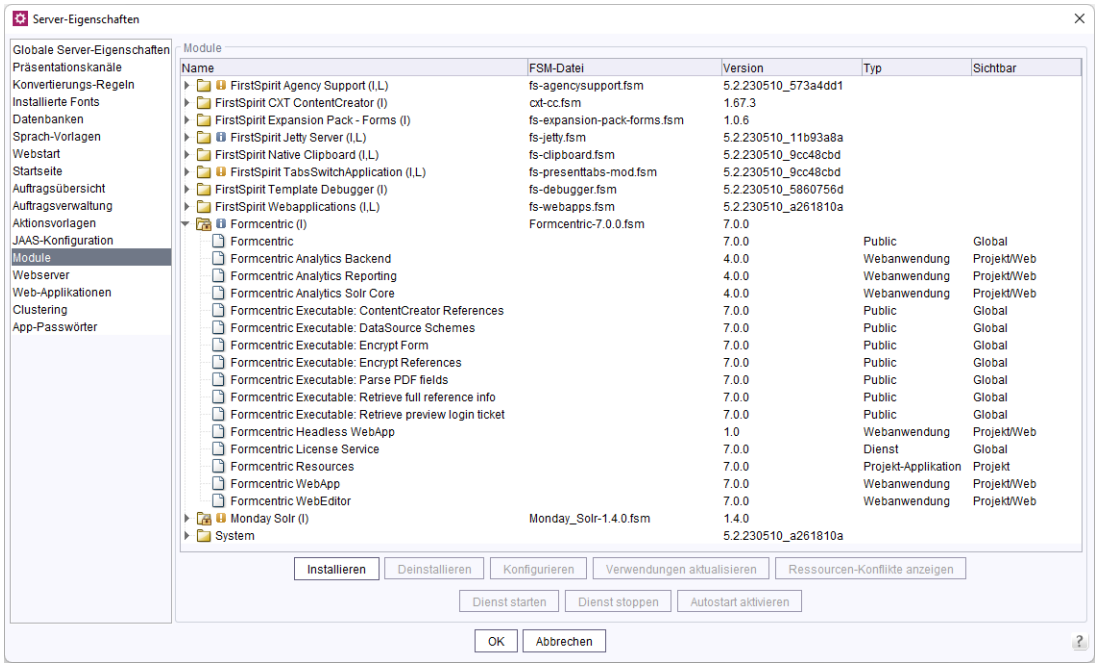


Abbildung 4.2. Liste der Module in den Server-Eigenschaften

Die Berechtigungen des Moduls werden erst wirksam, nachdem Sie den FirstSpirit-Server neu gestartet haben. Setzen Sie die Installation im Anschluss an den Neustart mit dem nächsten Punkt fort.

4.2. Lizenzdatei

Durch Doppelklick auf den Eintrag *Formcentric License Service* in der Modulübersicht öffnet sich ein Konfigurationsdialog. Laden Sie hier die Lizenzdatei, die Sie von Formcentric erhalten haben, hoch. Klicken Sie dafür auf den Button *Lizenzdatei importieren* und wählen Sie im darauf folgenden Dateiauswahldialog die Lizenzdatei aus.

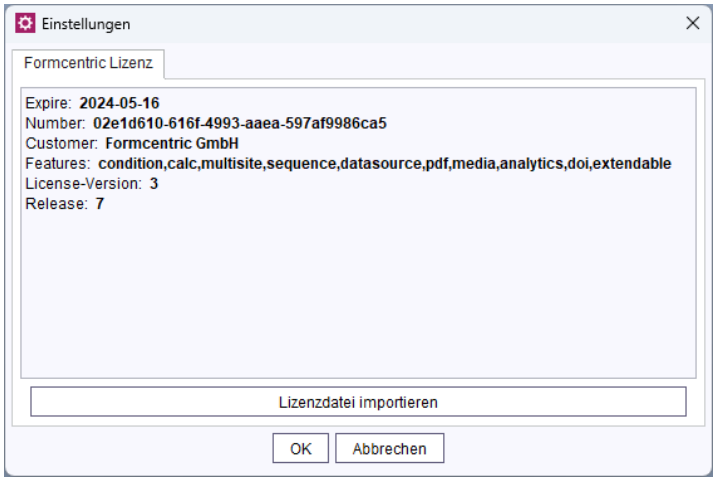


Abbildung 4.3. Konfiguration der Lizenzdatei

Schließen Sie den Konfigurationsdialog durch Klick auf *OK*.

Damit die Lizenzdatei auch in den Formcentric Webapplikationen verfügbar ist, müssen Sie diese nach jedem Update der Lizenz aktualisieren.

Die Lizenzdatei ist in der Regel 12 Monate gültig. In den letzten 30 Tagen vor Ablauf der Lizenz wird alle sechs Stunden eine Warnung im Log der Formcentric Webapplikation ausgegeben. **Eine darüber hinausgehende Warnung erfolgt nicht.** Ab dem Zeitpunkt, an dem die Lizenz abgelaufen ist, werden keine Formularaktionen mehr ausgeführt. Die Formulare werden jedoch weiterhin korrekt auf der Webseite dargestellt.

4.3. Installation des Formcentric Web-Editors

Für die Erstellung und Bearbeitung von Formularen im FirstSpirit ContentCreator stellt Formcentric eine ContentCreator-Erweiterung zur Verfügung.

Zur Installation der Erweiterung wechseln Sie in den Bereich *Webapplikationen*.

Fügen Sie dort der vorhandenen Webapplikation *ContenCreator* den Formcentric Web-Editor hinzu, indem Sie auf *Hinzufügen* klicken. In dem folgenden Auswahldialog werden Ihnen alle verfügbaren Web-Komponenten angezeigt. Wählen Sie den Eintrag *Formcentric WebEditor* aus.

Schließen Sie den Auswahldialog durch Klick auf *OK*.

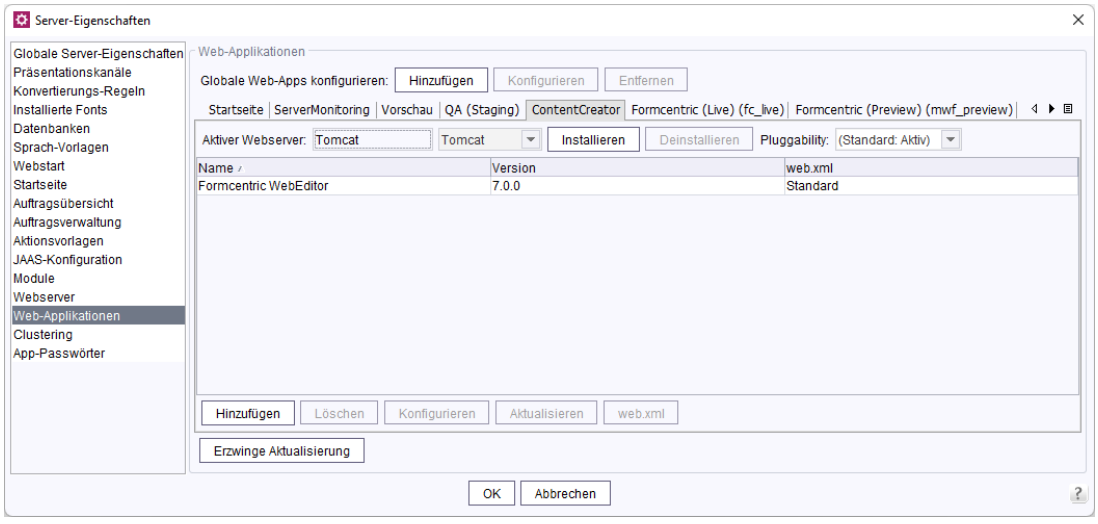


Abbildung 4.4. Installation des Formcentric Web-Editors

Aktualisieren Sie anschließend die ContentCreator Webapplikation auf dem Web-Server.

4.4. Formcentric Webapplikationen

Für die Darstellung und Verarbeitung der Formulare stellt Formcentric zwei unterschiedliche WebApp-Module zur Verfügung. Bei dem einen Modul handelt es sich um

eine Spring-MVC-Webanwendung, während das andere eine Headless-Webanwendung darstellt.

Bei der Spring-MVC-Webanwendung von Formcentric erfolgt die Datenverarbeitung und das Rendering der Formulare serverseitig auf Basis von Spring-MVC sowie JSP-/Freemarker-Templates. Die Headless-Webanwendung auf der anderen Seite, verwendet eine moderne, dezentralisierte Architektur, die das Backend (Datenverarbeitung) und das Frontend (UI-Rendering) trennt. Diese Anwendung nutzt Spring-Boot für das Backend und React für das Frontend.

Die Konfiguration der beiden Web-Anwendungen ist mit Ausnahme der unter Abschnitt „Register Datei-Upload“ beschriebenen Punkte, identisch. Im Folgenden wird die Konfiguration daher am Beispiel der Headless-Anwendung beschrieben.

Üblicherweise werden für die Vorschau und die Live-Seite getrennte Instanzen der Webanwendungen installiert. Aus diesem Grund werden bei der ersten Installation des Moduls automatisch vier globale Webapplikationen angelegt: *fc_preview_headless* und *fc_live_headless* für die Headless-Anwendung und *fc_preview* und *fc_live* für die Spring-MVC-Webanwendung. Für den Fall, dass Sie davon abweichende Namen verwenden möchten, führen Sie die im Abschnitt 4.4.1 aufgeführten Schritte für die betreffende Webanwendung manuell aus.

4.4.1. Installation

Legen Sie eine neue globale Webapplikation an. Diese wird für die Darstellung der Formulare innerhalb der Webseite benötigt.

Eine neue Webapplikation erstellen Sie, indem Sie im Bereich *Web-Applikationen* auf den Button *Hinzufügen* klicken.

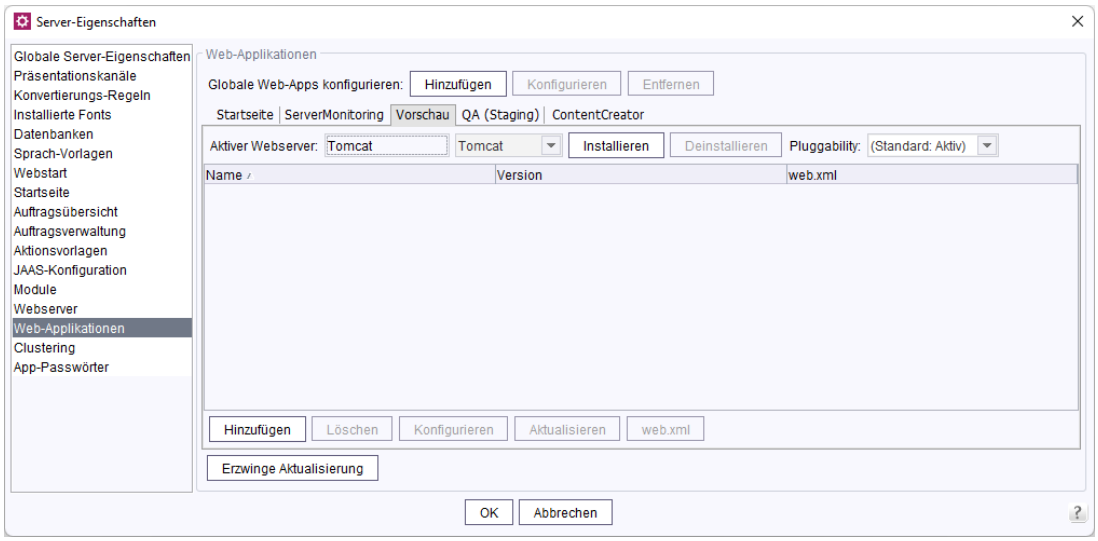


Abbildung 4.5. Globale Webapplikation hinzufügen

In dem folgenden Dialog geben Sie die ID, den Namen und den Kontext der Webapplikation ein. Schließen Sie den Dialog durch Klick auf *OK*.

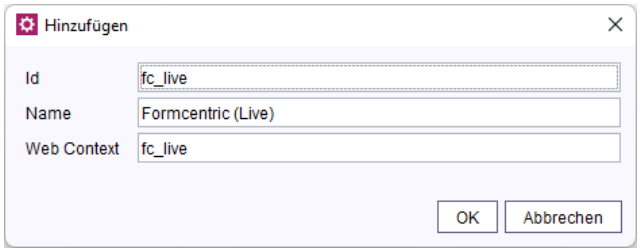


Abbildung 4.6. Neue Webapplikation hinzufügen

Üblicherweise werden folgende IDs verwendet: *fc_preview_headless* für die Preview-Anwendung und *fc_live_headless* für die Live-Anwendung. Wenn Sie hiervon abweichende IDs vergeben, müssen Sie diese auch im Skript *formcentric_headless_url* anpassen (siehe Abschnitt 5.2, „Skript formcentric_headless_url“).

Im nächsten Schritt fügen Sie der soeben angelegten Webapplikation das Formcentric Webapp-Modul hinzu. Klicken Sie auf *Hinzufügen* und wählen Sie *Formcentric Headless WebApp* aus.

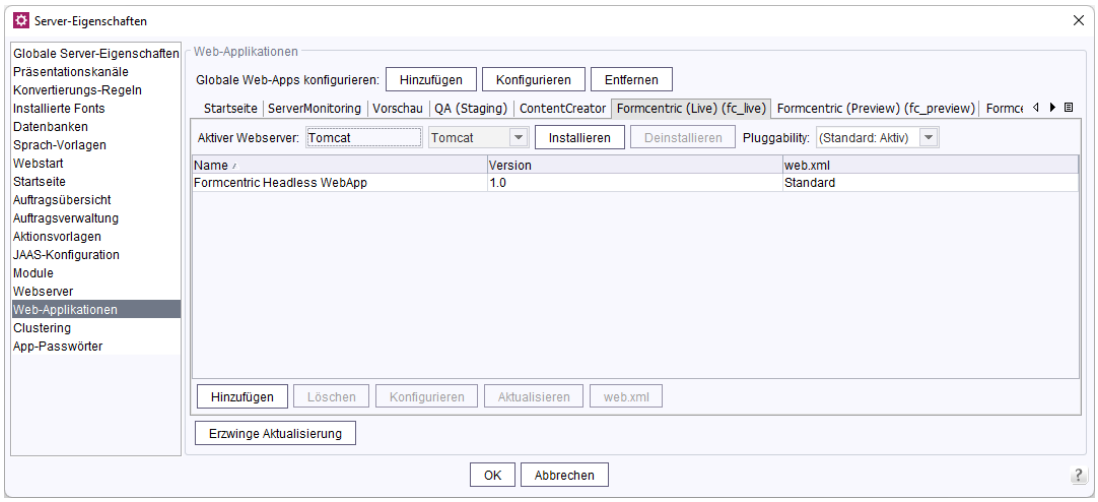


Abbildung 4.7. Globale Webapplikation mit installierter Web-Komponente

4.4.2. Konfiguration

Durch Doppelklick auf die Web-Komponente *Formcentric Headless WebApp* öffnet sich der Konfigurationsdialog, der mehrere Register umfasst.

Register Mail-Server

Im Register *Mail-Server* konfigurieren Sie den Mail-Server, über den die Formularerweiterung E-Mails versendet.

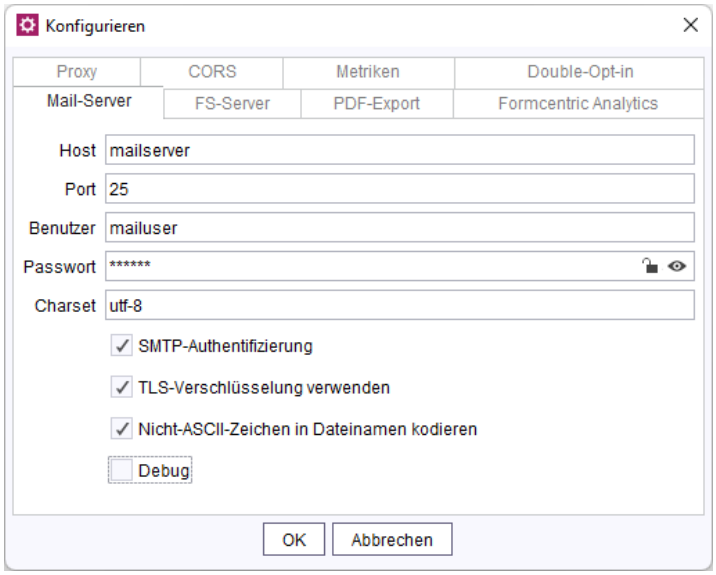


Abbildung 4.8. Konfiguration der Webapplikation im Register „Mail-Server“

Host: Name oder IP-Adresse des SMTP-Mail-Servers.

Port: Portnummer des Mail-Servers.

Benutzer: Der Benutzername für die Anmeldung am Mail-Server.

Passwort: Das Passwort für die Anmeldung am Mail-Server.

Charset: Das Character-Encoding, mit dem die E-Mails versendet werden (beispielsweise utf-8, iso-8859-15, et cetera).

SMTP-Authentication: Durch die Aktivierung dieser Checkbox legen Sie fest, dass die Anmeldedaten (Benutzer, Passwort) für die Anmeldung am SMTP-Server verwendet werden.

TLS-Verschlüsselung verwenden: Hier können Sie konfigurieren, ob START-TLS verwendet werden soll.

Nicht-ASCII-Zeichen in Dateinamen kodieren: Aktivieren Sie diese Checkbox um alle Nicht-ASCII-Zeichen im Dateinamen zu kodieren. Beachten Sie, dass diese Kodierung gegen die MIME-Spezifikation verstößt, aber für das Zusammenspiel mit einigen E-Mail-Clients, die diese Konvention verwenden, nützlich ist. Der Standardwert ist true.

Debug: Durch die Aktivierung dieser Checkbox legen Sie fest, dass für jede versendete E-Mail ausführliche Statusinformationen ins Log geschrieben werden. Hierzu gehören z. B. der gesamte Inhalt, Meta-Informationen und Header, welche persönliche Daten enthalten können.

Register PDF-Export

Im Register *PDF-Export* konfigurieren Sie globale Einstellungen der PDF-Action.

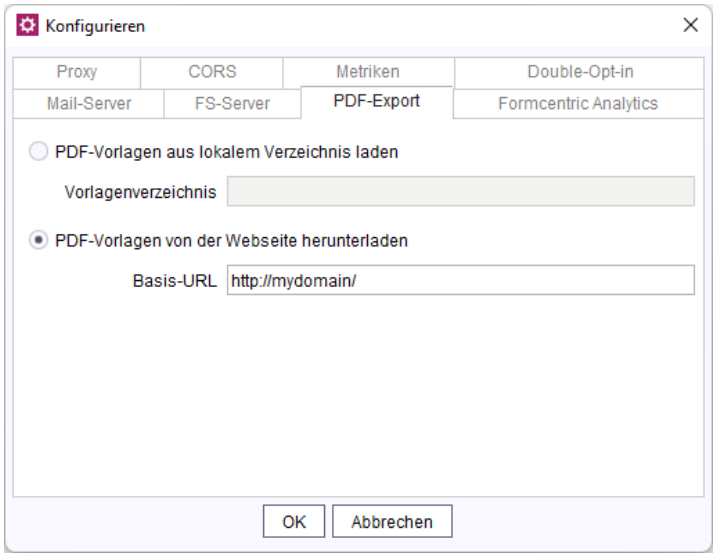


Abbildung 4.9. Konfiguration der Webapplikation im Register „PDF-Export“

PDF-Vorlagen aus lokalem Verzeichnis laden: Ab Version 5.5.0 von Formcentric werden die PDF-Vorlagendokumente standardmäßig direkt aus der Preview- bzw. Live-Webapp geladen. Selektieren Sie diese Option, wenn Sie die Vorlagendokumente stattdessen wie zuvor aus einem lokalen Verzeichnis laden möchten. In diesem Fall müssen Sie die Vorlagen zuvor veröffentlichen.

Vorlagenverzeichnis: In diesem Feld hinterlegen Sie den Pfad des Verzeichnisses, in das die PDF-Vorlagendokumente bei der Veröffentlichung kopiert werden (siehe dazu auch Abschnitt 4.9, „Konfiguration der Veröffentlichungsaufgaben“). Das Verzeichnis muss von der Webapplikation aus erreichbar sein. Zudem benötigt der User, mit dem der Webserver gestartet wurde, Leserechte in dem angegebenen Verzeichnis.

Das Vorlagenverzeichnis können Sie sowohl relativ zum Webapp-Verzeichnis (beispielsweise *WEB-INF/pdf*) als auch absolut (beispielsweise */var/pdf* bzw. *c:/var/pdf*) angeben. Wenn das Verzeichnis nicht existiert, wird es beim Start der Anwendung angelegt. In diesem Fall benötigt der User zusätzlich Schreibrechte im übergeordneten Verzeichnis.

PDF-Vorlagen von der Webseite herunterladen: Wählen Sie diese Option, wenn die PDF-Vorlagendokumente aus der Preview- bzw. Live-Webapp geladen werden sollen. In diesem Fall müssen die Dokumente wie die übrigen Inhalte in den entsprechenden Webauftritt veröffentlicht werden.

Basis-URL: Geben Sie hier die externe Adresse ein, unter der Ihr Webauftritt von außen zu erreichen ist.

Register FS-Server

Im Register *FS-Server* tragen Sie die Verbindungsdaten zu Ihrem FirstSpirit-Server ein. Diese Angaben werden für die Actions *Datenquelle* und *Medienverwaltung* benötigt.

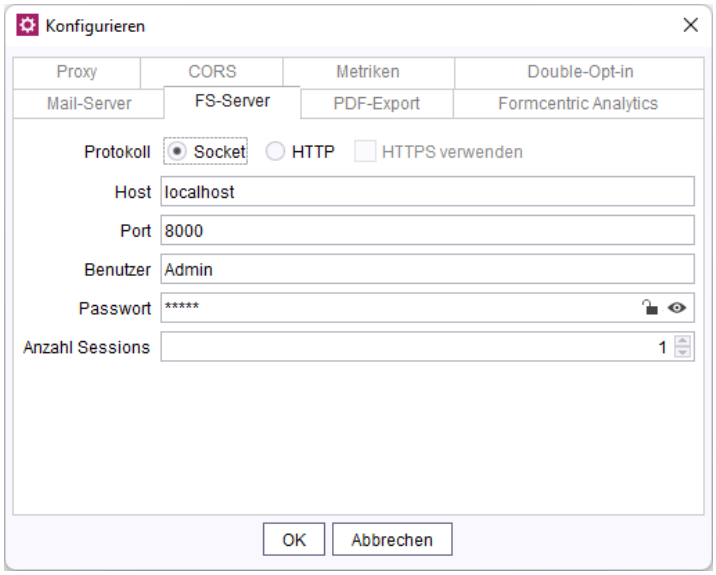


Abbildung 4.10. Konfiguration der Webapplikation im Register „FS-Server“

Protokoll: Wählen Sie hier das Übertragungsprotokoll (HTTP oder Socket) für die Kommunikation mit dem FistSpirit-Server aus.

HTTPS verwenden: Bei Verwendung von HTTP als Übertragungsprotokoll legen Sie durch die Auswahl dieser Checkbox fest, dass die Kommunikation mit dem Server verschlüsselt wird.

Host: Name oder IP-Adresse des FirstSpirit-Servers.

Port: Portnummer des FirstSpirit-Servers.

Benutzer: Der Benutzername für die Anmeldung am FirstSpirit-Server.

Passwort: Das Passwort für die Anmeldung am FirstSpirit-Server.

Anzahl Sessions: Die maximale Anzahl gleichzeitiger Sessions, die zum FirstSpirit-Server geöffnet werden dürfen.

Um die Verbindung zum FirstSpirit-Server herzustellen, benötigt das System den Hostnamen, von dem aus die Webapp auf den Server zugreifen kann. Zudem muss die Webapp in der Lage sein, eine Verbindung zu dem Port des FirstSpirit-Servers aufzubauen. Es kann unter Umständen notwendig sein, den Port in einer verwendeten Firewall freizuschalten.

Die maximale Anzahl an Sessions ist von Ihrer FirstSpirit-Lizenz abhängig. Beachten Sie, dass die in der Registerkarte angegebene Anzahl von Sessions unter Umständen Ihren Redakteuren nicht mehr zur Verfügung steht.

Register Formcentric Analytics

Im Register *Formcentric Analytics* konfigurieren Sie die Verbindungsparameter zum Formcentric Analytics-Backend.

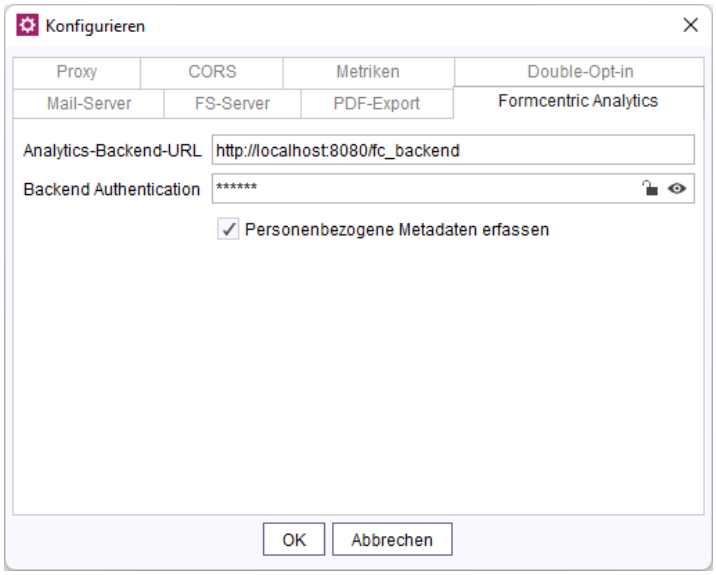


Abbildung 4.11. Konfiguration der Webapplikation im Register „Formcentric Analytics“

Analytics-Backend-URL: URL, unter der das Formcentric Analytics-Backend erreichbar ist.

Backend Authentication: Tragen Sie hier das Secret für die Anmeldung am Analytics-Backend ein. Dieses muss mit dem *Backend-Client-Secret* übereinstimmen, dass Sie bei der Konfiguration des Analytics-Backends vergeben haben (siehe „Register Sicherheit“).

Personenbezogene Metadaten erfassen: Wenn Sie diese Checkbox aktivieren, werden neben den eingegebenen Formulardaten immer auch Informationen über den verwendeten Browser (*User-Agent*), die eingestellte Browser-Sprache (*Language*) und die zuvor besuchte Seite (*Referer*) gespeichert. Hierbei handelt es sich um Metadaten, die unter Umständen dazu verwendet werden können, voneinander unabhängige Datensätze einer Person zuzuordnen.

Anstatt des Client-Secrets kann auch ein vorgenerierter *Access-Token* verwendet werden. Hierfür ist jedoch eine Anpassung in der Spring-Konfiguration notwendig (siehe „formcentric-analytics.xml“).

Register Proxy

PDF-Vorlagendokumente werden standardmäßig direkt aus der Preview- bzw. Live-Webapp geladen. Sollte der Zugriff auf die Webapplikationen über einen Web-Proxy erfolgen, so haben Sie im Register *Proxy* die Möglichkeit, diesen Proxy zu konfigurieren.

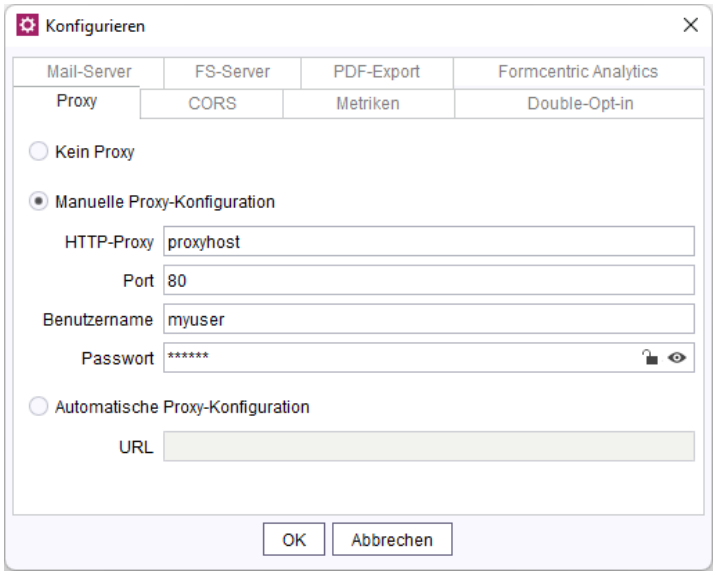


Abbildung 4.12. Konfiguration eines Web-Proxies im Register „Proxy“

Kein Proxy: Aktivieren Sie diese Einstellung, wenn Sie keinen Proxy verwenden möchten.

Manuelle Proxy-Konfiguration: Wählen Sie diese Einstellung, wenn Sie die Proxy-Einstellungen manuell eingeben möchten.

HTTP-Proxy: Hostname des Proxy-Servers, der verwendet werden soll.

Port: Port des Proxy-Servers

Benutzername: Username für die Anmeldung am Proxy-Server.

Passwort: Passwort für die Anmeldung am Proxy-Server.

Automatische Proxy-Konfiguration: Wählen Sie diese Option, wenn Sie anstelle der manuellen Proxy-Konfiguration eine Proxy-Konfigurationsdatei (.pac) verwenden möchten.

URL: Geben Sie hier die Adresse (URL) ein, von der die Konfigurationsdatei geladen werden soll.

Register CORS

Wie bereits ausgeführt, werden die Formulare durch eine eigenständige Webapp-Komponente erzeugt und dynamisch in die umgebene Webseite eingebettet. Der Aufruf der Webapp erfolgt hierbei als asynchroner Javascript Request (AJAX).

Für den Fall, dass die Formular-Webapp unter einer eigenen Domain (z. B. http://forms.mydomain.com) betrieben wird, handelt es sich bei den Formularaufrufen um sog. Cross-Origin-Requests. Diese sind normalerweise durch die Same-Origin-Policy (SOP) untersagt und werden daher von den aktuellen Browsern blockiert. Diese Einschränkungen können durch das Setzen von Access-Control-Headern für bestimmte Clients aufgehoben werden.

In Register *CORS* haben Sie die Möglichkeit, diese Access-Control-Header entsprechend Ihrer Anforderungen zu konfigurieren.

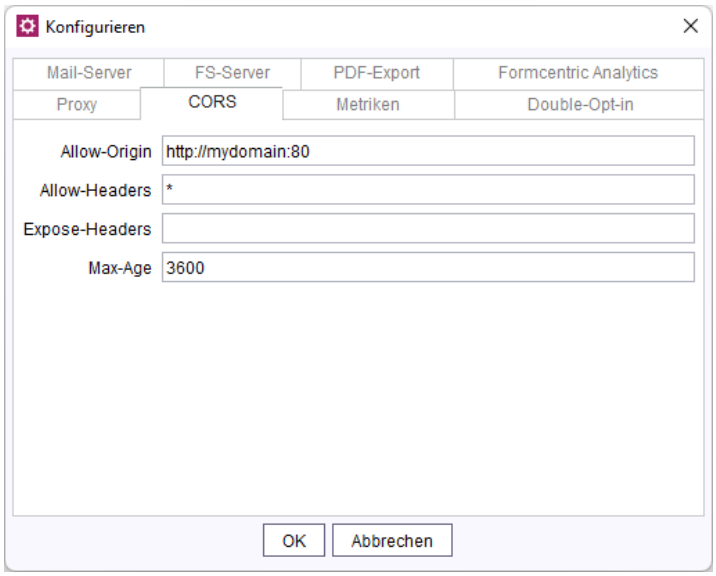


Abbildung 4.13. Konfiguration Cross-Origin Resource Sharing (CORS)

Allow-Origin: Geben Sie hier eine kommaseparierte Liste der Domains an, von denen aus auf die Formular-Webapp zugegriffen werden soll (z. B. www.mydomain.com, www.another-domain.com). Durch die Angabe eines Sterns „*“ legen Sie fest, dass von jeder Domain aus zugegriffen werden darf. Dies ist die Standardeinstellung.

Allow-Headers: Mit diesem Parameter legen Sie fest, welche HTTP-Header über Domain-Grenzen hinweg übertragen werden dürfen. Durch die Angabe eines Sterns „*“ legen Sie fest, dass alle HTTP-Header erlaubt sind. Dies ist die Standardeinstellung.

Expose-Headers: Mit diesem Parameter legen Sie fest, welche HTTP-Header in der Server-Antwort enthalten sein dürfen. Die HTTP-Header *X-Redirect-Location* und *X-Redirect-Delay* werden vom Formcentric jQuery-Plugin ausgewertet und sind daher immer freigegeben.

Max-Age: Mit diesem Parameter legen Sie fest, wie lange die Informationen aus einer Preflight-Anfrage gültig sind.

Wenn Sie die Formular-Webapp wie beschrieben unter einer separaten Domain betreiben, werden von der Anwendung automatisch vollqualifizierte URLs in der HTML-Ausgabe erzeugt. Als Base-URL wird dabei die vom Browser aufgerufene Base-URL verwendet, z. B. *https://forms.mydomain.com/*.

Erfolgt der Aufruf der Formular-Anwendung über einen vorgeschalteten Load-Balancer oder Reverse-Proxy, erhält der HTTP-Request nur Informationen über die Verbindung vom Load-Balancer zur Anwendung. Um die ursprünglich vom aufrufenden Browser verwendeten Verbindungsdaten wie Host, Port etc. an die Anwendung weiterzuleiten, muss der Load-Balancer zusätzliche HTTP-Request-

Header setzen. Diese werden von der Anwendung u. a. für die Generierung der voll-qualifizierten URLs benötigt.

Folgende Header müssen hinzugefügt werden:

HTTP-Request-Header	Beschreibung
X-Forwarded-Proto	Protokoll (HTTP oder HTTPS), das der Browser für die Verbindung zu Ihrem Load-Balancer verwendet hat.
X-Forwarded-Host	Host, den der Browser für die Verbindung mit dem Load-Balancer verwendet hat.
X-Forwarded-Port	Port, den der Browser für die Verbindung mit dem Load-Balancer verwendet hat.
X-Forwarded-For	IP-Adresse des aufrufenden Browsers

Register Metriken

Zur Überwachung betriebsrelevanter Systemwerte wie Speicherverbrauch oder Prozessorauslastung der Live-WebApp stellt Ihnen Formcentric verschiedene Metriken zur Verfügung. Auf diese können Sie über die URL */fc_live/servlet/secure/health* zugreifen. Die Metriken werden standardmäßig in einem Textformat ausgeliefert, welches von dem Monitoring System *Prometheus* verarbeitet werden kann.

Unter der URL */fc_live/servlet/secure/usage* erhalten Sie Metriken über die Verwendung der Formulare.

Der Zugriff auf die Metriken ist durch ein Login (Basic Authentication) abgesichert. Die zugehörigen Login-Daten können Sie im Register *Metriken* festlegen.

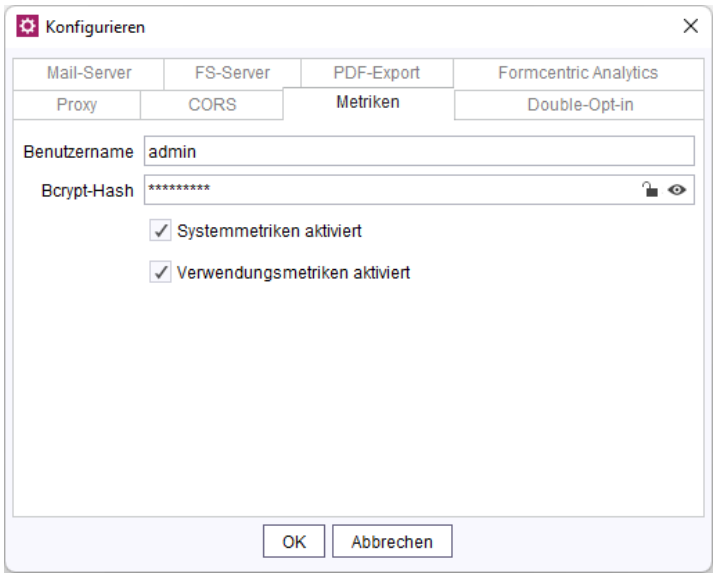


Abbildung 4.14. Konfiguration der Zugangsdaten für die Anzeige der Metriken

Benutzer: Name des Benutzers, der Zugriff auf die Metriken erhalten soll.

Bcrypt-Hash: Geben Sie hier den bcrypt-Hash Ihres Passworts ein. Zur Erzeugung eines bcrypt-Hashes können Sie beispielsweise den Online-Dienst https://bcrypt-generator.com verwenden.

Systemmetriken aktiviert: Aktiviert oder deaktiviert den Endpunkt für Systemmetriken.

Verwendungsmetriken aktiviert: Aktiviert oder deaktiviert den Endpunkt für Nutzungsmetriken.

Register Double-Opt-in

Im Rahmen eines Double-Opt-in Vorgangs erhält der Anwender eine Mail mit einem Bestätigungslink. Der Link verweist auf die Webseite, auf der er das Formular ausgefüllt und abgesendet hat. Um zu verhindern, dass der User durch Manipulation dieses Links auf eine Phishing- oder Spam-Seite umgeleitet werden kann, muss die angegebene URL überprüft werden. Die Überprüfung erfolgt anhand der in diesem Dialog konfigurierten URL-Pattern.

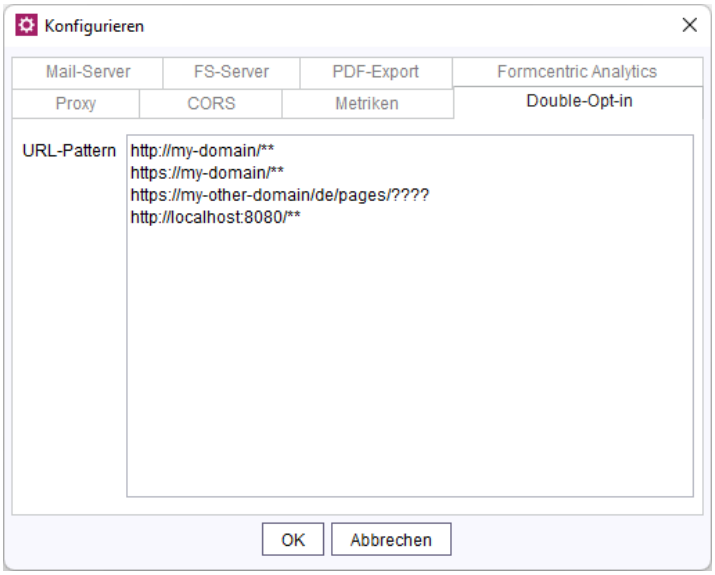


Abbildung 4.15. Konfiguration der URL-Pattern für das Double-opt-in

URL-Pattern: Geben Sie in diesem Eingabefeld ein URL-Pattern für jede Webseite an, in die ein Formcentric-Formular eingebettet werden kann. Innerhalb der URL-Pattern können Sie folgende Wildcards verwenden:

1. **?** entspricht einem Zeichen
2. * entspricht einem oder mehreren Zeichen
3. ** entspricht einem oder mehreren Verzeichnissen im Pfad

Register Datei-Upload

Dateien, die ein Benutzer in einem Formular hochlädt, werden bis zum Abschluss des Formularvorgangs auf dem Server zwischengespeichert. Dieser Konfigurationsdialog

ermöglicht es Ihnen, den Speicherort und die maximale zulässige Größe für Datei-Uploads festzulegen.

Als temporären Zwischenspeicher können Sie entweder ein lokales Verzeichnis oder MinIO, einen S3-kompatiblen Objektspeicherdienst, nutzen. MinIO ist speziell für die Speicherung großer, unstrukturierter Daten konzipiert und für eine hohe I/O-Last optimiert. MinIO bietet zudem starke Sicherheitsmaßnahmen, einschließlich serverseitiger und clientseitiger Verschlüsselung, und kann sowohl On-Premises als auch in der Cloud betrieben werden. Der Einsatz von MinIO ist besonders für Installationen mit mehreren Instanzen der Headless-Webanwendung zu empfehlen.

Dieser Konfigurationsdialog steht Ihnen nur bei der Konfiguration der Headless-Webanwendung zur Verfügung.

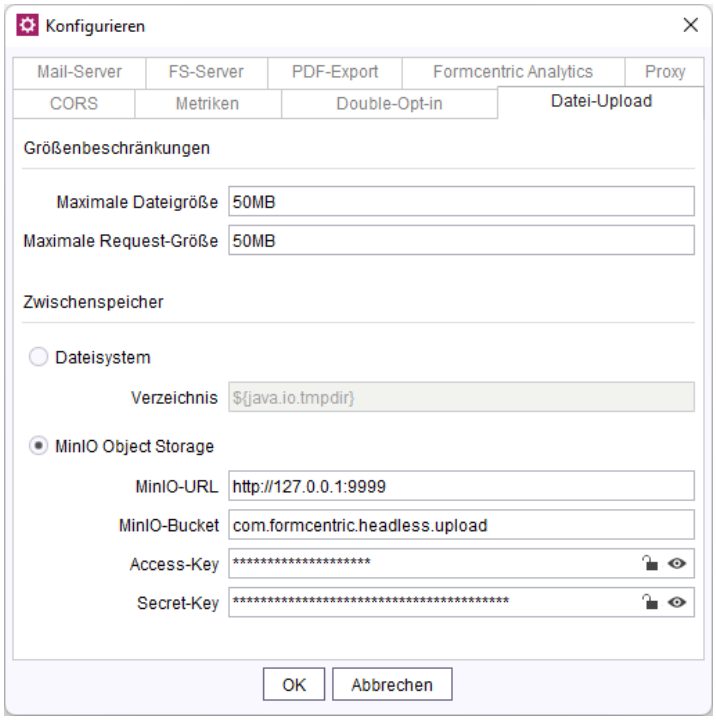


Abbildung 4.16. Konfiguration von File-Uploads

Maximal Dateigröße: Dieser Parameter legt die maximale Größe fest, die eine Datei haben kann, um hochgeladen zu werden. Dies dient als Sicherheitsmaßnahme, um zu verhindern, dass zu große Dateien den Server überlasten. Diese Einstellung stellt eine technische Begrenzung dar, die auch von redaktionellen Vorgaben nicht überschrieben werden kann.

Bei der Spring-MVC-Webanwendung kann die Maximal Dateigröße in der *web.xml* Datei festgelegt werden.

Maximale Request-Größe: Dieser Parameter legt die maximale Größe fest, die eine HTTP-Anfrage haben kann. Er schließt die Größe der hochgeladenen Datei(en) sowie

alle zugehörigen Metadaten und Header ein. Wie bei der maximalen Dateigröße dient auch dieser Parameter dazu, die Belastung des Servers zu begrenzen und die Performance zu verbessern.

Dateisystem: Dieser Parameter legt fest, dass die hochgeladenen Dateien im lokalen Dateisystem oder auf einem verbundenen Netzlaufwerk zwischengespeichert werden.

Verzeichnis: Sofern Sie *Dateisystem* als Zwischenspeicher ausgewählt haben, müssen Sie hier den genauen Speicherort angeben. Dies sollte ein Pfad zu einem Verzeichnis sein, zu dem die Webanwendung Zugang und Schreibrechte hat. In diesem Feld können Sie zudem auch die Umgebungsvariable *${java.io.tempdir}* angeben, wenn Sie das temporäre Verzeichnis der Java-VM verwenden möchten.

MinIO Object Storage: Dieser Parameter aktiviert die Nutzung von MinIO als Zwischenspeicher. Wenn aktiviert, müssen die restlichen MinIO-bezogenen Parameter entsprechend konfiguriert werden.

MinIO-URL: Dieser Parameter legt die URL fest, unter der Ihr MinIO-Speicherdienst erreichbar ist. Sie sollte das Protokoll, den Hostnamen und ggf. den Port enthalten.

MinIO-Bucket: Der Parameter *MinIO-Bucket* bezeichnet den Namen des spezifischen Containers in Ihrem MinIO-Speicher, in dem die hochgeladenen Dateien abgelegt werden. Ein *Bucket* in MinIO entspricht im Prinzip einem Ordner auf einem herkömmlichen Dateisystem.

Access-Key: Der *Access-Key* ist ein Teil Ihrer Zugangsdaten für den MinIO-Speicher. Er wird zusammen mit dem *Secret-Key* zur Authentifizierung bei MinIO verwendet. Der Access-Key sollte sorgfältig verwahrt und nicht öffentlich zugänglich gemacht werden.

Secret-Key: Der *Secret-Key* ist das Gegenstück zum *Access-Key* und wird zusammen mit diesem zur Authentifizierung bei MinIO verwendet. Wie beim Access-Key ist auch der Secret-Key streng vertraulich und sollte nicht öffentlich zugänglich gemacht werden.

Wiederholen Sie die Schritte in den Abschnitten 4.4.1 und 4.4.2 für die Preview Webapplikation.

4.5. Analytics Backend Webapplikation

4.5.1. Installation

Legen Sie eine weitere globale Webapplikation an. Diese wird zum Speichern und Verarbeiten Ihrer Formulare mit Formcentric Analytics benötigt.

Geben Sie die ID, den Namen und den Kontext der Webapplikation ein. Üblicherweise wird die ID *fc_backend* für die Analytics-Backend-Anwendung verwendet. Schließen Sie den Dialog durch Klick auf *OK*.

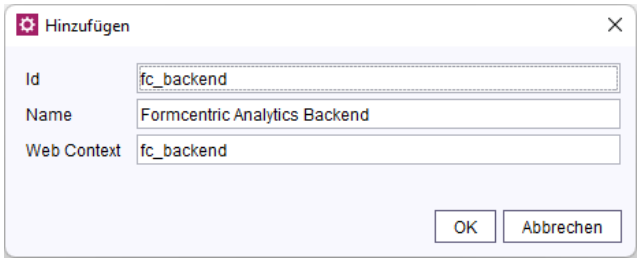


Abbildung 4.17. Neue Webapplikation hinzufügen

Fügen Sie der soeben angelegten Webapplikation eine Web-Komponente hinzu, indem Sie auf *Hinzufügen* klicken. In dem folgenden Auswahldialog werden Ihnen alle verfügbaren Web-Komponenten angezeigt. Wählen Sie den Eintrag *Formcentric Analytics Backend*.

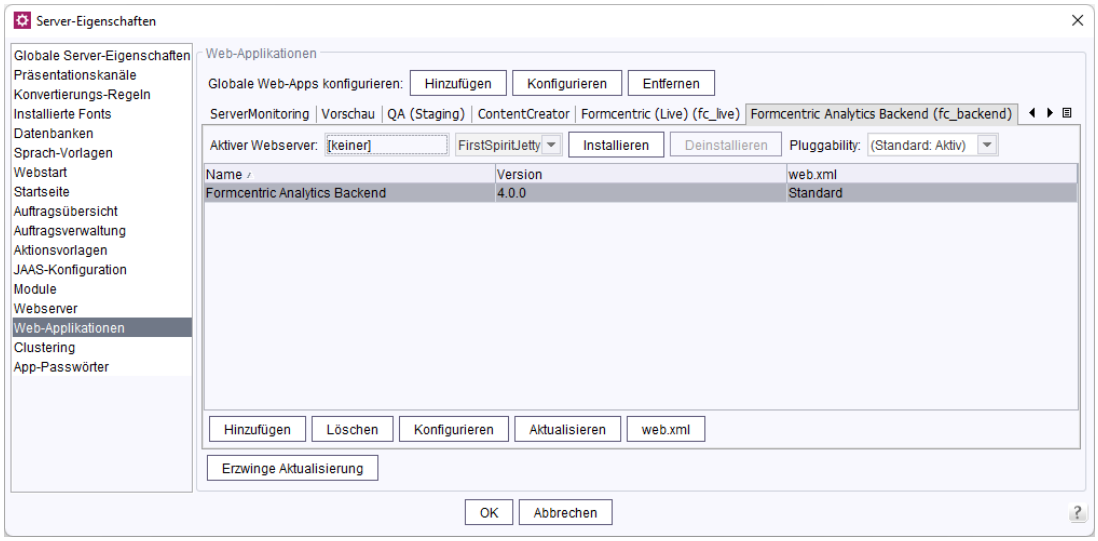


Abbildung 4.18. Globale Webapplikation mit installierter Web-Komponente

4.5.2. Konfiguration

Durch Doppelklick auf die Web-Komponente *Formcentric Analytics Backend* öffnet sich der Konfigurationsdialog, der mehrere Register umfasst.

Register Allgemeines

Im Register Allgemeines konfigurieren Sie allgemeine Einstellungen.

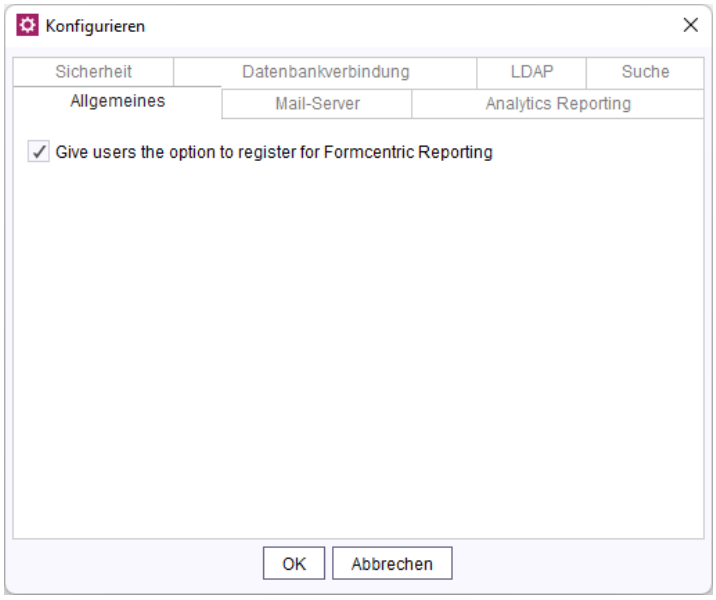


Abbildung 4.19. Konfiguration der Analytics-Backend-Webapplikation im Register „Allgemeines“

Benutzern die Möglichkeit geben, sich für das Reporting zu registrieren: Durch Auswahl dieser Checkbox legen Sie fest, dass im Login-Dialog der Formcentric Reporting-Oberfläche einen Link auf den Registrierungsdialog angezeigt wird. In dem Registrierungsdialog können Benutzer Zugang zu Formcentric Analytics beantragen (siehe dazu auch Kap. 2. „Anmeldung und Registrierung” im Benutzerhandbuch von Formcentric Analytics).

Register Mail-Server

Im Register Mail-Server konfigurieren Sie den Mail-Server, über den das Analytics-Backend E-Mails an die Nutzer versendet.

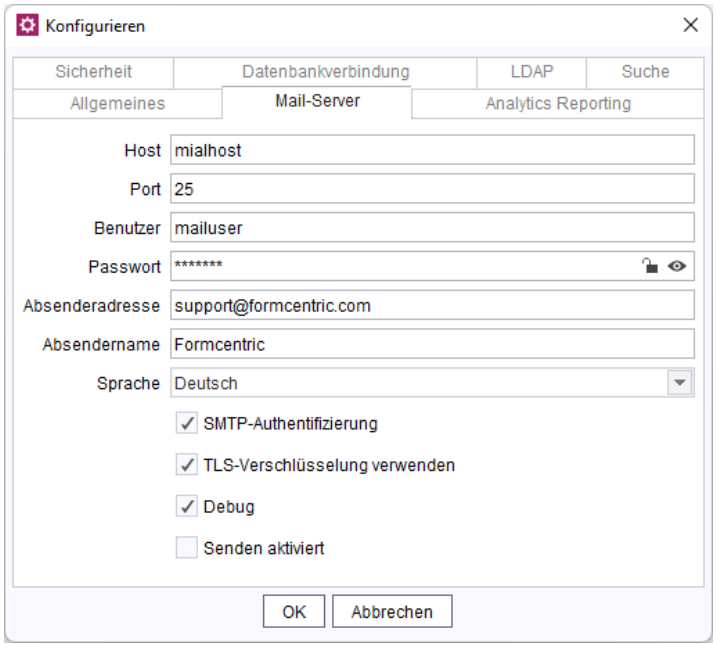


Abbildung 4.20. Konfiguration der Analytics-Backend-Webapplikation im Register „Mail-Server“

Host: Name oder IP-Adresse des SMTP-Mail-Servers.

Port: Portnummer des Mail-Servers.

Benutzer: Benutzer, mit dem sich Formcentric Analytics am SMTP-Server authentifiziert.

Passwort: Passwort für den SMTP-Benutzer.

Absenderadresse: Hier können Sie konfigurieren, von welcher E-Mail-Adresse Formcentric Analytics E-Mails verschickt.

Absendername: Über diese Property vergeben Sie den Namen, der als Absender für alle E-Mails von Formcentric Analytics angezeigt werden soll.

Sprache: Geben Sie an, in welcher Sprache die E-Mails verschickt werden sollen.

SMTP-Authentifizierung: Aktivieren / Deaktivieren, ob sich der User mit dem AUTH-Befehl anmelden soll.

TLS-Verschlüsselung verwenden: Hier können Sie konfigurieren, ob START-TLS verwendet werden soll.

Debug: Aktivieren / Deaktivieren von zusätzlichen Debugging-Informationen während des Mailversands. Hierzu gehören z. B. der gesamte Inhalt, Meta-Informationen und Header, welche persönliche Daten enthalten können.

Senden aktiviert: Hier können Sie das Versenden von E-Mails aus Formcentric Analytics aktivieren / deaktivieren.

Register Analytics Reporting

Im Register *Analyitics Reporting* konfigurieren Sie die externe URL, über die die Reporting-Oberfläche aufgerufen werden kann.

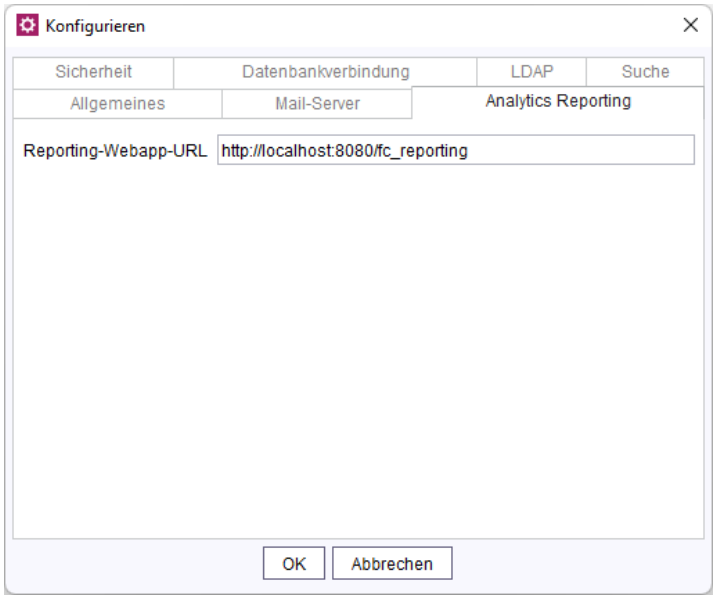


Abbildung 4.21. Konfiguration der Analytics-Backend-Webapplikation im Register „Analytics Reporting“

Reporting-Webapp-URL: Externe URL, unter der die Reporting-Webapplikation erreichbar ist.

Register Sicherheit

Im Register Sicherheit tragen Sie die Zugangsdaten für die Anmeldung am Analytics-Backend ein.

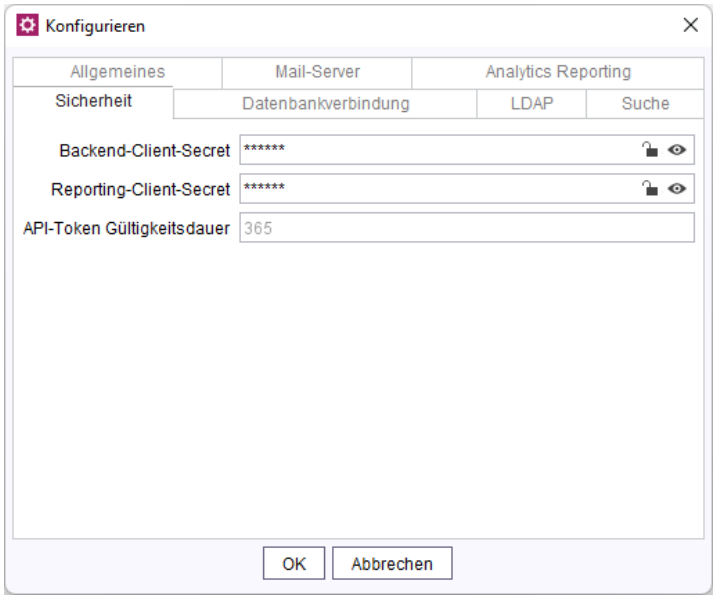


Abbildung 4.22. Konfiguration der Analytics-Backend-Webapplikation im Register „Sicherheit“

Backend-Client-Secret: Vergeben Sie hier ein beliebiges Secret. Dieses wird von der Formcentric Webapplikation verwendet, um *Access-Token* für das Analytics-Backend zu erzeugen und sollte möglichst sicher gewählt werden.

Reporting-Client-Secret: Vergeben Sie hier ein beliebiges Passwort. Dieses wird für die Autorisierung der Reporting-Anwendung am Analytics-Backend verwendet.

API-Token Gültigkeitsdauer: Vergeben Sie hier die Anzahl der Tage, die ein REST API-Token gültig ist.

Register Datenbankverbindung

Im Register Datenbankverbindung konfigurieren Sie die Datenbankverbindung der Analytics-Backend-Datenbank. Die erforderlichen Datenbanktabellen und Indizes werden beim Start der Anwendung automatisch angelegt oder aktualisiert.

Formcentric verwendet für den Zugriff auf die Backend-Datenbank das Java Database Connectivity API (JDBC). Die benötigten JDBC-Treiber sind nicht im FSM-Modul von Formcentric enthalten. Kopieren Sie daher den JDBC-Treiber für das eingesetzte Datenbanksystem auf den Webserver, sodass er im Class-path der Anwendung gefunden wird. Eine ausführliche Anleitung zur Installation und Konfiguration der Backend-Webapplikation finden Sie im Installationshandbuch *formcentric_backend_install_de.pdf*.

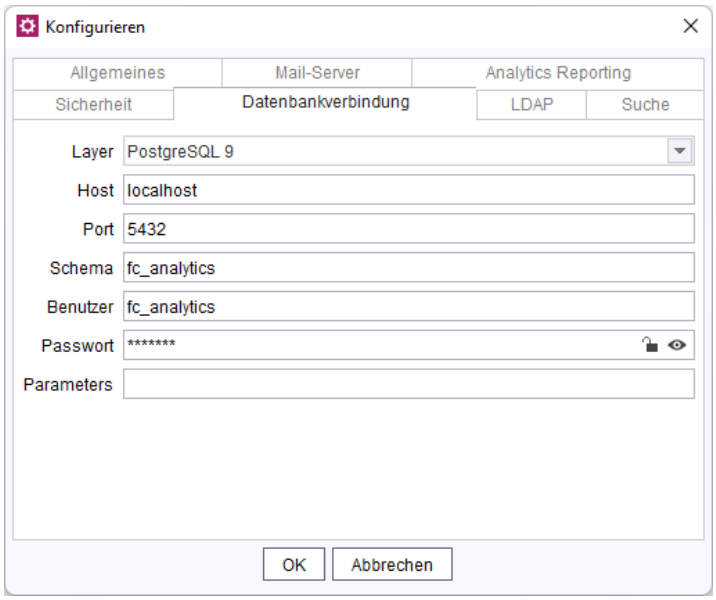


Abbildung 4.23. Konfiguration der Analytics-Backend-Webapplikation im Register „Datenbankverbindung“

Layer: Wählen Sie hier den passenden Treiber für Ihre Datenbank aus.

Host: Geben Sie den Namen oder die IP-Adresse Ihres Datenbank-Servers an.

Port: Geben Sie den Port an, auf dem sich der Client mit Ihrem Datenbank-Server verbinden soll.

Schema: Tragen Sie hier das zu verwendende Datenbank-Schema ein.

Benutzer: Hier hinterlegen Sie den Benutzernamen, mit dem sich das Formcentric Analytics-Backend an Ihrer Datenbank anmeldet.

Passwort: Tragen Sie hier das Passwort für den Benutzer ein.

Parameter: In diesem Feld haben Sie die Möglichkeit, zusätzliche datenbankspezifische Verbindungsparameter anzugeben. Diese werden so wie Sie sie eingetragen haben, an die Verbindungs-URL angefügt. Der Wert muss mit dem Trennzeichen beginnen, das von dem eingesetzten Datenbanksystem verwendet wird (z. B. ? *ssl=true*).

Register LDAP

Neben der Verwendung der internen Benutzerverwaltung von Formcentric Analytics haben Sie auch die Möglichkeit, ein externes Benutzerverzeichnis wie LDAP oder Active Directory anzubinden. Im Register LDAP konfigurieren Sie die Verbindungs- und Filtereinstellungen für Ihr Benutzerverzeichnis.

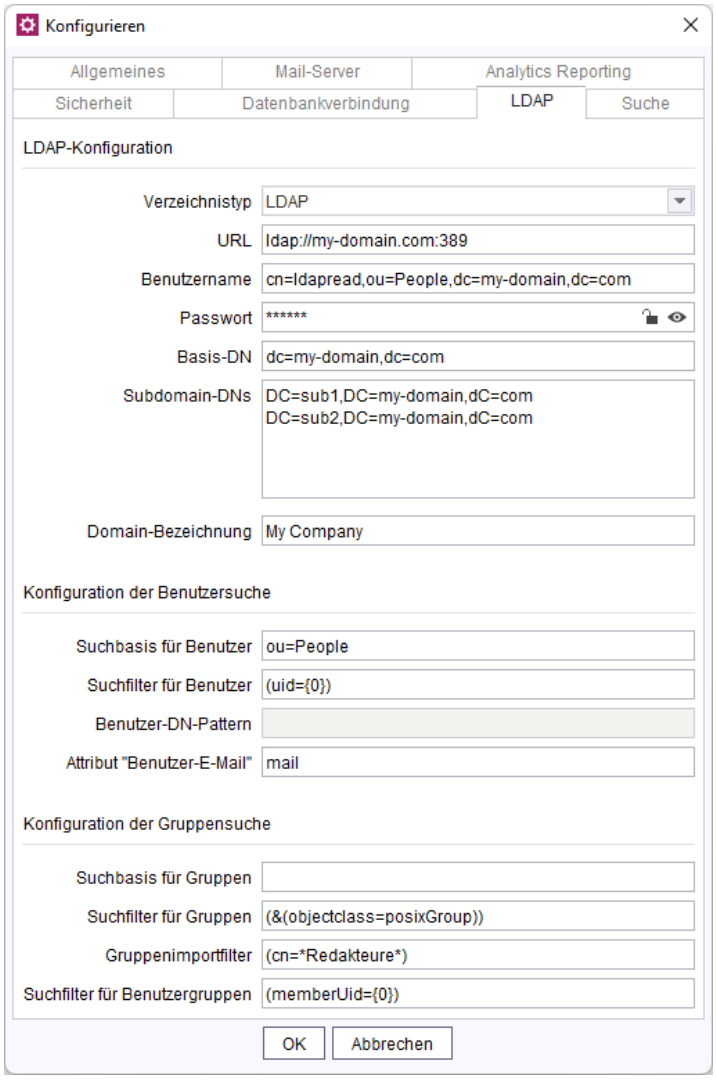


Abbildung 4.24. Konfiguration der Analytics-Backend-Webapplikation im Register „LDAP“

Verzeichnistyp: Wählen Sie hier den Typ des externen Benutzerverzeichnisses aus, das Sie anbinden möchten. Formcentric unterstützt LDAP und Active Directory.

URL: URL des LDAP-Servers.

Benutzername: Nutzername für den Zugriff auf den LDAP-Server

Passwort: Passwort für den Zugriff auf den LDAP-Server

Basis-DN: Der Distinguished Name (DN) des Basisverzeichnisses, in dem die Benutzer- und Gruppenobjekte enthalten sind (z. B. *dc=mydomain,dc=com*).

Subdomain-DNs: Optionale Angabe von Subdomains, die weitere Benutzer- und Gruppenobjekte enthalten (z. B. *dc=subdomain1,dc=my-domain,dc=com*). Fügen Sie für jede weitere Subdomain eine neue Zeile mit dem voll qualifizierten DN der Subdomain ein.

Domain-Bezeichnung: Lesbare Bezeichnung der Domain, die dem Anwender bei der Anmeldung vorgeschlagen wird.

Suchbasis für Benutzer: Angabe eines Objekts im Verzeichnisbaum, unterhalb dessen die Benutzersuche durchgeführt werden soll (z. B. *ou=people* für LDAP oder *cn=users* für Active Directory.

Suchfilter für Benutzer: Angabe eines LDAP-Suchfilters, mit dem unterhalb der angegebenen Suchbasis nach Benutzern gesucht wird (z. B. *(uid={0})* für LDAP oder *(samaccountname={0})* für Active Directory).

Der Platzhalter *{0}* wird bei der Ausführung der Suche durch den eingegebenen Nutzernamen ersetzt.

Eine allgemeine Beschreibung der Syntax von Suchfiltern finden Sie unter folgender Adresse: http://www.faqs.org/rfcs/rfc2254.html.

Benutzer-DN-Pattern: Pattern, mit dessen Hilfe der Distinguished Name (DN) eines Benutzers erzeugt werden kann (z. B. *uid={0},ou=people*).

Benutzer-E-Mail: Name des Attributfelds des Benutzer-Objekts, in dem die E-Mail-Adresse zu finden ist. Die E-Mail-Adresse wird in der Analytics Datenbank gespeichert.

Suchbasis für Gruppen: Angabe eines Objektes im Verzeichnisbaum, unterhalb dessen die Gruppensuche durchgeführt werden soll (z. B. *DC=company,DC=com* für LDAP oder *cn=users* für Active Directory).

Suchfilter für Gruppen: Angabe eines LDAP-Suchfilters mit dem unterhalb der angegebenen Suchbasis nach Gruppen gesucht wird (z. B. *(&(objectclass=groupOfUniqueNames))* für LDAP oder *(&(objectclass=group)))* für Active Directory).

Gruppenimportfilter: Alle Gruppen, für die bestimmte Berechtigungen innerhalb von Formcentric Analytics vergeben werden sollen, müssen zuvor in das interne Benutzerverzeichnis von Formcentric Analytics importiert werden. Im Feld *Gruppenimportfilter* haben Sie die Möglichkeit, einen LDAP-Suchfilter anzugeben, mit dem die zu importierenden LDAP-Gruppen aus der Liste der verfügbaren Gruppen ausgewählt werden (z. B. *(cn=*AnalyticsUsers*)*). Wenn Sie in diesem Feld nichts angeben,

werden alle Gruppen importiert, die durch den im Feld *Suchfilter für Gruppen* angegeben LDAP-Suchfilter ermittelt werden.

Suchfilter für Benutzergruppen: Mithilfe dieses LDAP-Suchfilters werden die Gruppen eines Benutzers ermittelt, in denen er Mitglied ist. Diesen Parameter müssen Sie nur angeben, wenn Sie als Verzeichnistyp *LDAP* ausgewählt haben (z. B. *(memberUid={0})*). Der Platzhalter *{0}* wird bei der Ausführung der Suche durch den eingegebenen Nutzernamen ersetzt.

Wenn Sie die Analytics-Backend-Webapplikation auf dem *FirstSpirit Jetty-Server* installieren möchten, müssen Sie die *web.xml* der Webapplikation um den Context-Parameter *webAppRootKey* erweitern. Geben Sie als Wert die ID der Webapplikation ein.

```
<context-param>
    <param-name>webAppRootKey</param-name>
    <param-value>mwf_backend</param-value>
</context-param>
```

Register Suche

Für die formularübergreifende Volltextsuche wird die Enterprise Suchmaschine *Solr* benötigt. Üblicherweise wird die Solr-Suchmaschine als eigenständige Anwendung betrieben. Eine Anleitung zur Installation und Konfiguration von Solr finden Sie auf der Produktseite http://lucene.apache.org/solr/.

Um die externe Solr-Instanz mit Formcentric Analytics verwenden zu können, müssen Sie zunächst über die Administrationsoberfläche von Solr einen neuen Core mit dem Namen *mwfanalytics* anlegen.

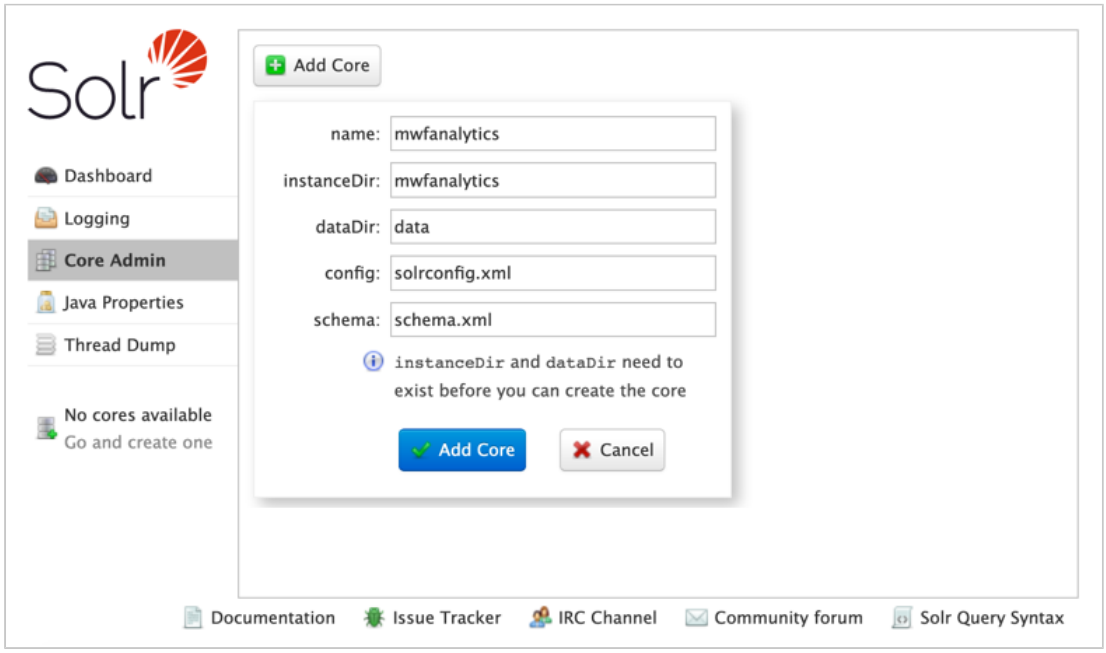


Abbildung 4.25. Core Administration in der Solr-Administrationsoberfläche

Die hierfür benötigten Konfigurationsdateien *solrconfig.xml* und *schema.xml* finden Sie in der Archivdatei *webforms-backend-solr-config-4.3.1.tar*.

Konfigurieren Sie anschließend die Solr-Suchmaschine im Register *Suche*.

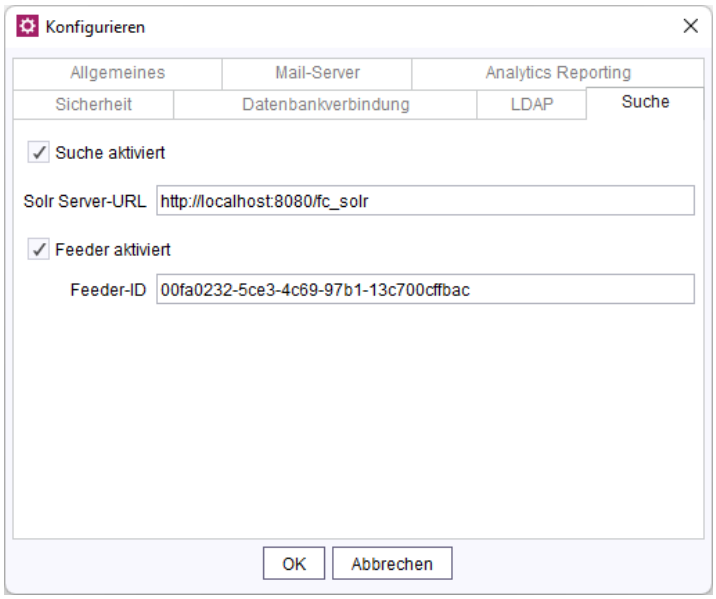


Abbildung 4.26. Konfiguration der Solr-Suchmaschine im Register „Suche“

Suche aktiviert: Aktiviert oder deaktiviert die formularübergreifende Volltextsuche.

Solr Server-URL: Tragen Sie hier die URL des Solr-Servers ein.

Feeder aktiviert: Aktiviert oder deaktiviert den Feeder, mit dessen Hilfe der Suchindex befüllt wird. Für den Fall, dass Sie mehrere Instanzen der Backendanwendung betreiben, ist es in der Regel ausreichend, wenn nur eine Backend-Anwendung den Suchindex befüllt.

Feeder-ID: Vergeben Sie hier eine eindeutige ID für den Feeder. Dabei kann es sich um eine beliebige Zeichenkette handeln. Bei gleichzeitigem Betrieb mehrerer Backend-Anwendungen müssen sich die Feeder-IDs der verschiedenen Backend-Anwendungen voneinander unterscheiden.

Alternativ zur Verwendung einer externen Solr-Anwendung haben Sie auch die Möglichkeit, Solr als globale FirstSpirit-Webapp zu installieren. Formcentric stellt hierfür ein separates FirstSpirit-Modul zur Verfügung. Eine Anleitung zur Installation dieses Moduls finden Sie in Abschnitt 4.7, „Solr Webapplikation“.

In diesem Fall müssen Sie im Parameter *Solr Server-URL* die URL der globalen Webapp eintragen.

Wiederholen Sie die Schritte in den Abschnitten 4.5.1 und 4.5.2, wenn Sie für die Preview- und Live-Umgebung getrennte Instanzen des Analytics-Backends verwenden möchten.

4.6. Analytics Reporting Webapplikation

4.6.1. Installation

Legen Sie eine weitere globale Webapplikation für die Reporting-Anwendung von Formcentric Analytics an.

Geben Sie die ID, den Namen und den Kontext der Webapplikation ein. Üblicherweise wird die ID *fc_reporting* für Reporting-Webapplikation verwendet. Schließen Sie den Dialog durch Klick auf *OK*.

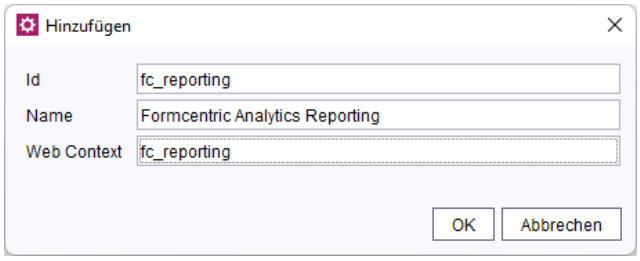


Abbildung 4.27. Reporting-Webapplikation hinzufügen

Fügen Sie der soeben angelegten Webapplikation eine Web-Komponente hinzu, indem Sie auf *Hinzufügen* klicken. In dem folgenden Auswahldialog werden Ihnen alle verfügbaren Web-Komponenten angezeigt. Wählen Sie nun den Eintrag *MFormcentric Analytics Reporting* aus.

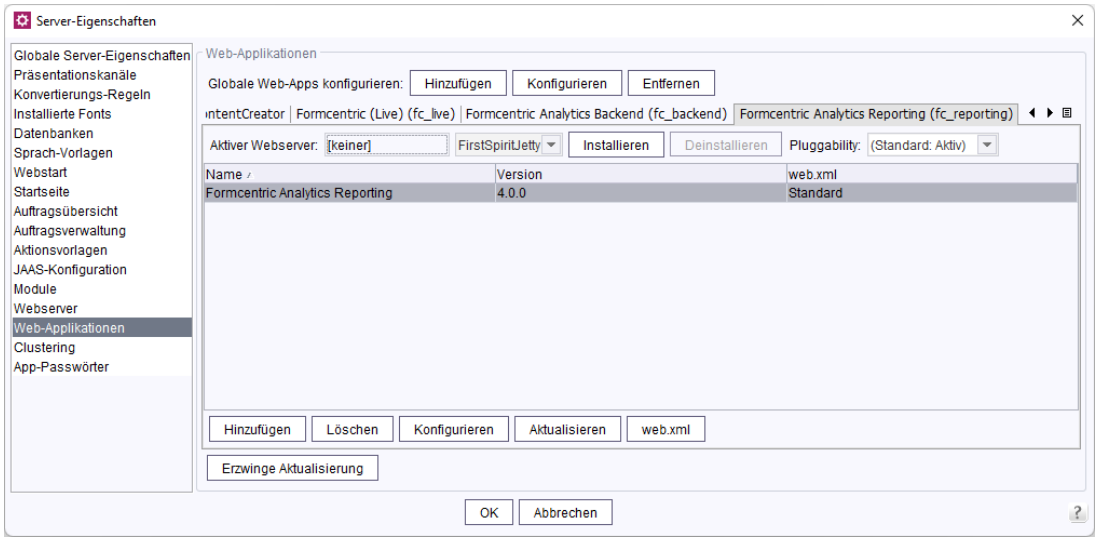


Abbildung 4.28. Globale Webapplikation mit installierter Web-Komponente

4.6.2. Konfiguration

Durch Doppelklick auf die Web-Komponente *Formcentric Analytics Reporting* öffnet sich der Konfigurationsdialog.

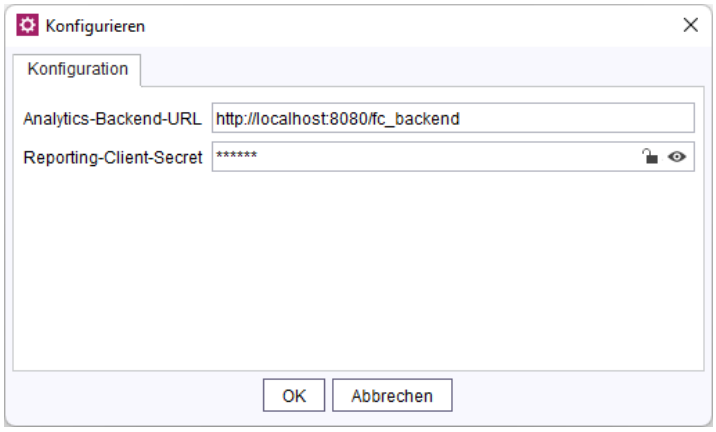


Abbildung 4.29. Konfiguration der Webapplikation im Register „Konfiguration“

Analytics Backend URL: URL zum Aufrufen des Formcentric Analytics-Backend. Wenn Sie in ihrer Umgebung für Live und Preview ein jeweils ein eigenes Backend betreiben, achten Sie darauf, dass Sie hier die passende URL angeben.

Reporting-Client-Secret: Tragen Sie hier das Passwort für die Anmeldung am Analytics-Backend ein. Dieses muss mit dem *Reporting-Client-Secret* übereinstimmen, das Sie bei der Konfiguration des Analytic-Backends vergeben haben (siehe „Register Sicherheit“).

Wenn Sie die Reporting-Anwendung auf dem *FirstSpirit Jetty-Server* installieren möchten, müssen Sie die *web.xml* der Webapplikation um den Context-Parameter *webAppRootKey* erweitern. Geben Sie als Wert die ID der Webapplikation ein.

```
<context-param>
    <param-name>webAppRootKey</param-name>
    <param-value>fc_reporting</param-value>
</context-param>
```

Wiederholen Sie die Schritte in den Abschnitten 4.6.1 und 4.6.2, wenn Sie für die Preview- und Live-Umgebung getrennte Instanzen der Reporting-Anwendung verwenden möchten.

4.7. Solr Webapplikation

Für Test- und Entwicklungszwecke stellt Formcentric die Solr-Suchmaschine als separates FirstSpirit-Modul zur Verfügung. Dieses können Sie von unserem FTP-Server herunterladen.

Installieren Sie sie die Moduldatei *solr-module-1.4.0.fsm* über die Server- und Projektkonfiguration von FirstSpirit.

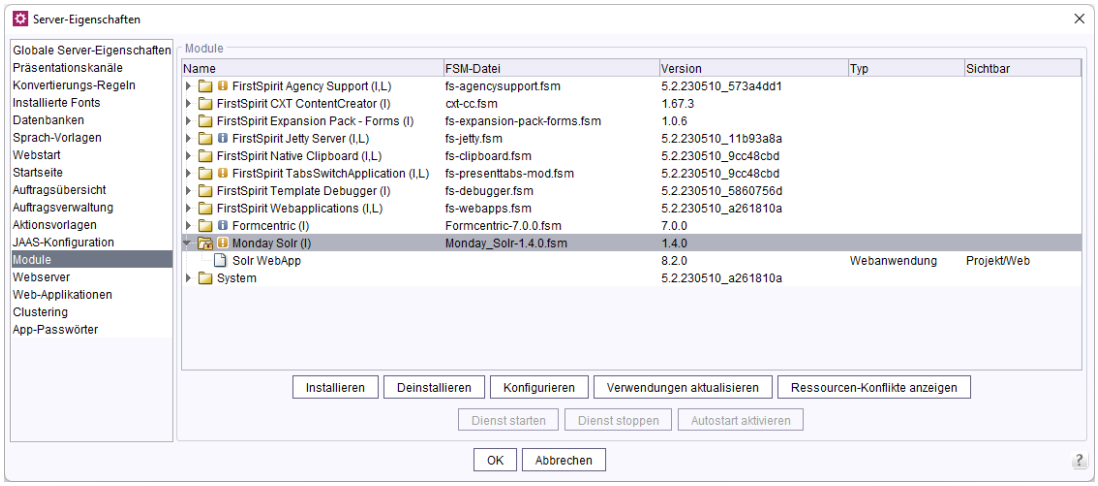


Abbildung 4.30. Modulübersicht

Legen Sie anschließend eine weitere globale Webapplikation an.

Geben Sie die ID, den Namen und den Kontext der Webapplikation ein. Üblicherweise wird die ID *fc_solr* für die Solr-Anwendung verwendet. Schließen Sie den Dialog durch Klick auf *OK*.

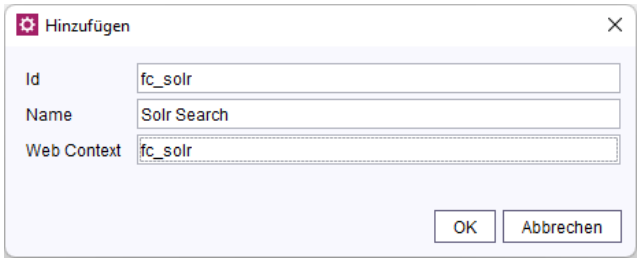


Abbildung 4.31. Neue Webapplikation hinzufügen

Fügen Sie der soeben angelegten Webapplikation folgende Web-Komponenten hinzu, indem Sie auf *Hinzufügen* klicken. In dem folgenden Auswahldialog werden Ihnen alle verfügbaren Web-Komponenten angezeigt. Wählen Sie die Einträge *Solr WebApp* und *Monday Webforms Analytics Solr Core* aus.

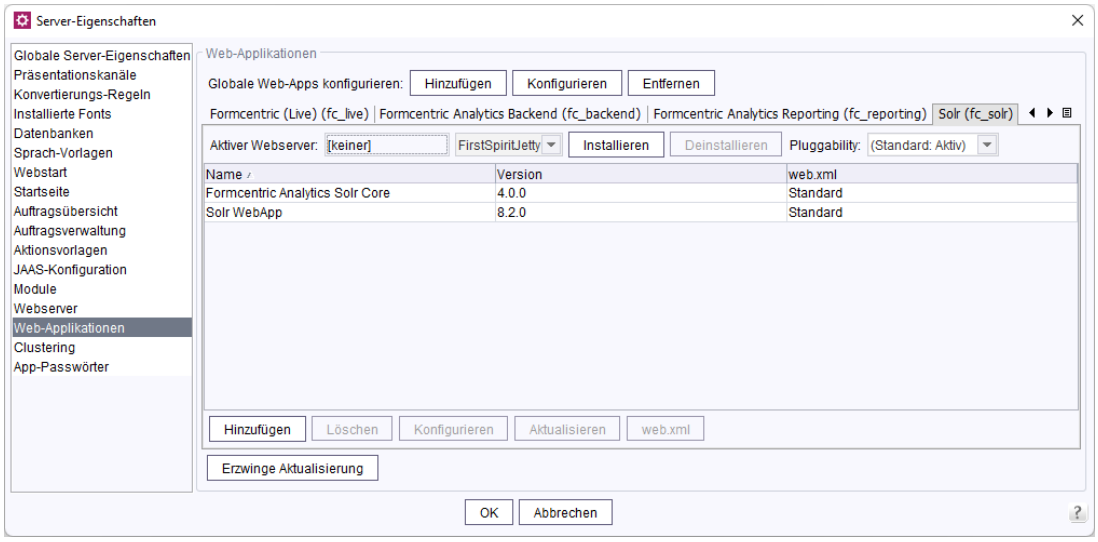


Abbildung 4.32. Solr-Webapplikation mit installierten Web-Komponenten

Installieren Sie die Webapplikation auf dem Web-Server.

4.8. Installation der Projektkomponente

Für die Integration der Formularerweiterung in ein neues oder bestehendes Projekt benötigen Sie zusätzliche Ressourcen (Javascripte, Absatzvorlagen und CSS). Diese sind in einer separaten Projektkomponente zusammengefasst, die Sie Ihrem Projekt mithilfe der Projektkonfiguration hinzufügen können.

Öffnen Sie dazu die Projektkonfiguration durch Doppelklick auf das entsprechende Projekt in der Projektübersicht und wählen Sie im linken Menü den Punkt *Projekt-Komponenten*. Mit einem Klick auf den Button *Hinzufügen* öffnet sich ein Auswahldialog, in dem alle Projektkomponenten angezeigt werden, die auf dem Server installiert sind. Wählen Sie den Eintrag *Formcentric Resources*.

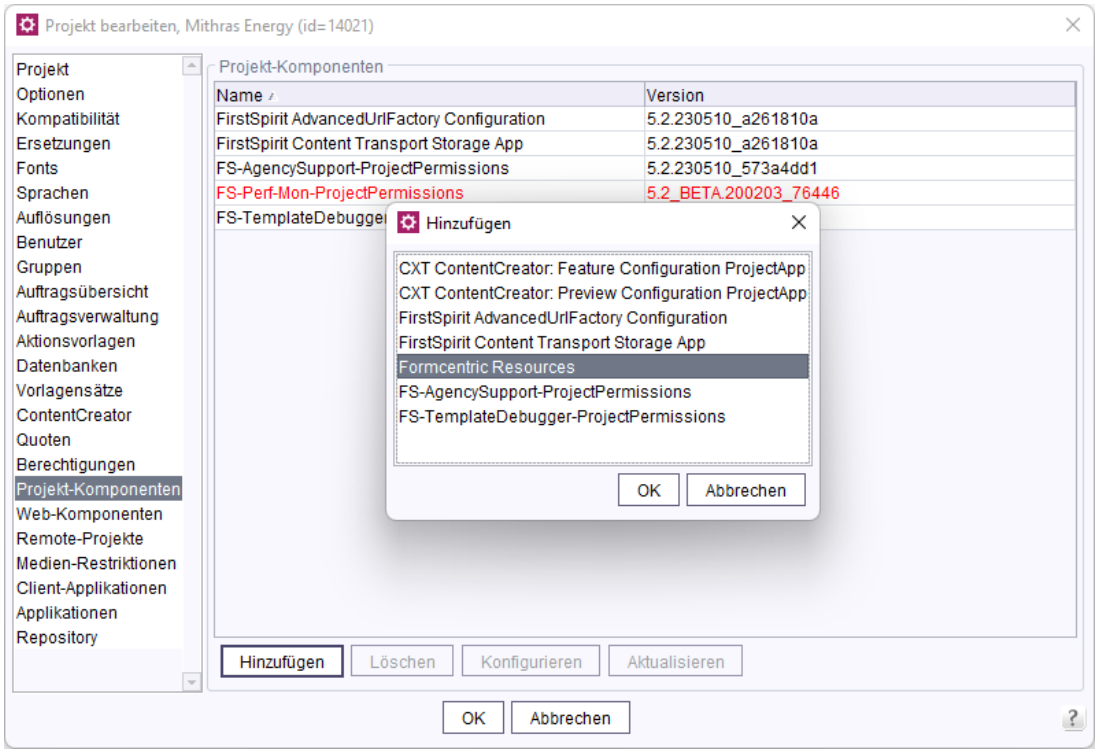


Abbildung 4.33. Projekt-Komponenten in den Projekteigenschaften

Nach der erfolgreichen Installation ist das Projekt nun um folgende Elemente erweitert:

- **Absatzvorlagen:** In den Absatzvorlagen des Projekts sind die neuen Absatzvorlagen *Formular* und *Headless-Formular* angelegt.

  Die enthaltenen Absatzvorlagen verwenden den Standard HTML-Ausgabekanal um das Formular in die Seite einzubetten. Wenn Ihr Projekt diesen Ausgabekanal nicht verwendet, müssen Sie die im Abschnitt 5.1.3, „Register Internet (HTML)“ beschriebenen Anpassungen in dem von Ihnen verwendeten Ausgabekanal manuell vornehmen.

- **Skripte:** Der Bereich *Skripte* in der Vorlagenverwaltung ist nun um den Ordner *Formcentric* erweitert, der verschiedene Generierungsskripte enthält.

- **Medien:** Die Medienverwaltung enthält nun den Ordner *Formcentric*. Dieser enthält die benötigten Javascripte und Style-Sheets in originaler und minimierter Form.

Sollten Sie die Projektkomponente wieder aus dem Projekt entfernen, werden automatisch auch die aufgeführten Ressourcen entfernt. Alle Änderungen, die Sie nach der Installation des Moduls daran vorgenommen haben, gehen dabei verloren!

Bei einem Update des Moduls auf eine neuere Version aktualisiert sich auch die Projektkomponente automatisch. Dabei ersetzt das System zwangs-

läufig die Absatzvorlagen (siehe Abschnitt 5.1, „Absatzvorlage“) durch neue Versionen. Dies führt dazu, dass Sie bestehende Formulare nach der Aktualisierung nicht mehr bearbeiten können, da das System die zugehörige Absatzvorlage nicht mehr findet. Um dieses Problem zu vermeiden, sollten Sie mit einer Kopien der in der Projektkomponente enthaltenen Absatzvorlagen arbeiten.

Hinweis: Sie können Formulare, deren Absatzvorlage gelöscht wurde, mit dem Skript *fix_ReferenceNotFoundException* reparieren.

4.9. Konfiguration der Veröffentlichungsaufgaben

Die nachfolgend beschriebene Konfiguration der Veröffentlichungsaufgaben müssen Sie nur durchführen, wenn die PDF-Vorlagendokumente aus einem lokalen Verzeichnis innerhalb der Formcentric Webapp geladen werden sollen (siehe dazu auch „Register PDF-Export“).

Für die Funktion der PDF-Action ist es notwendig, dass Sie Ihre PDF-Vorlagen in die Formcentric Webapplikation veröffentlichen. Die Veröffentlichung ist sowohl für die produktive Webapplikation als auch für die Vorschau erforderlich. Für die Veröffentlichung wird das externe Programm *rsync* verwendet.

Informationen zur Installation und Konfiguration von *rsync* im Zusammenhang mit FirstSpirit finden Sie im Kapitel 10 der FirstSpirit „Dokumentation für Administratoren“.

Für die Veröffentlichung der PDF-Vorlagen wird Ihnen ein Beanshell-Skript zur Verfügung gestellt, das Sie zusammen mit Formcentric erhalten.

Öffnen Sie die Projektkonfiguration durch Doppelklick auf das entsprechende Projekt in der Projektübersicht und wählen Sie im linken Menü den Punkt *Aktionsvorlagen*. Mit einem Klick auf den Button *Hinzufügen* öffnet sich ein Auswahldialog, in dem Ihnen verschiedene Aktivitäten angezeigt werden. Wählen Sie den Eintrag *Skript ausführen*.

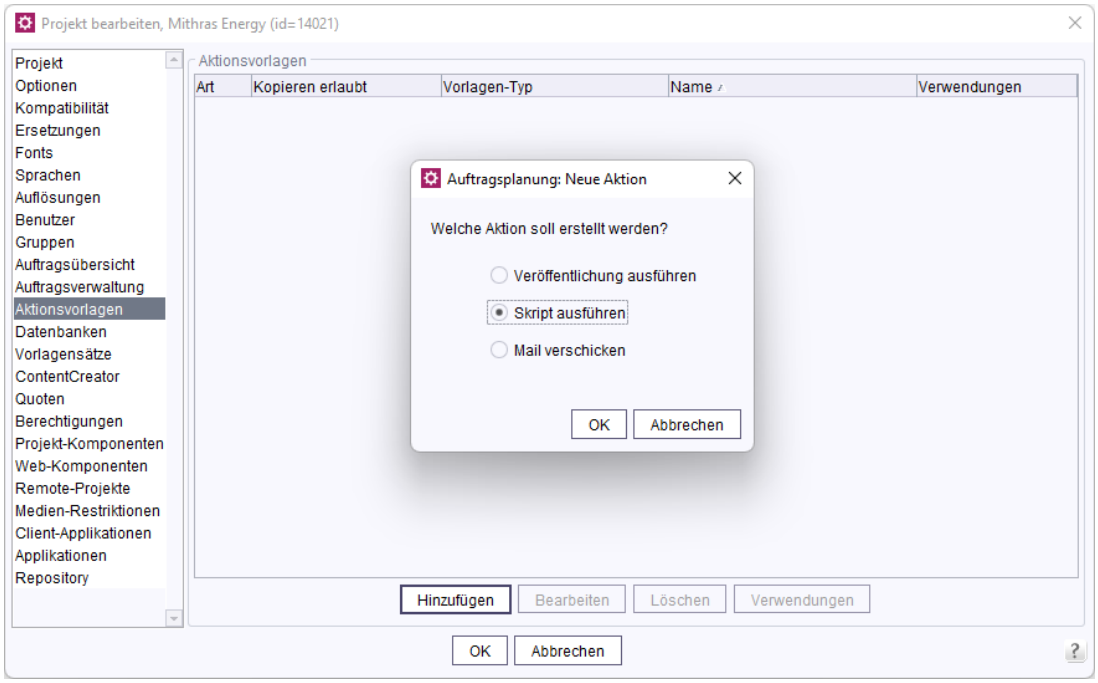


Abbildung 4.34. Auftragsvorlage im ServerManager anlegen

Im darauf folgenden Dialog *Skript* vergeben Sie den Namen „Formcentric RSync Deployment“.

Sie können einen abweichenden Namen verwenden, müssen dann aber das Skript im JavaClient anpassen.

Kopieren Sie den Inhalt des mitgelieferten Skripts (*ext-rsync-ssh.bsh*) in das darunterliegende Eingabefeld.

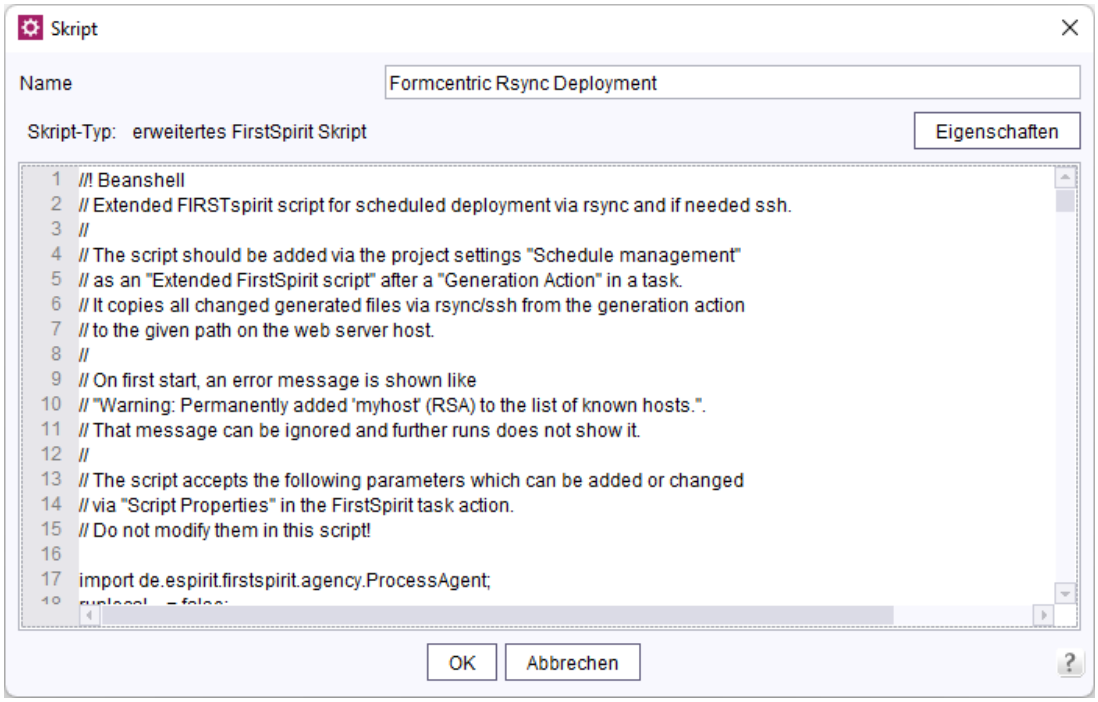


Abbildung 4.35. Deploymentskript der PDF-Action

Konfigurieren Sie anschließend die nachfolgend aufgeführten Parameter Ihres Preview- oder Auslieferungsservers, wie im Kapitel 10.5 der FirstSpirit „Dokumentation für Administratoren“ beschrieben.

Parameter	Beschreibung
rsync	Pfad zum Programm rsync (nur notwendig auf Windows-Systemen). Beispiel: *c:\cygwin\bin\rsync.exe*
runlocal	Mit diesem Parameter legen Sie fest, ob die Veröffentlichung in ein lokales Verzeichnis (*true*) - oder auf einen entfernten Server (*false*) erfolgen soll.
subfolder	Pfad zu einem Verzeichnis in der Medienverwaltung, in dem sich die PDF-Vorlagen befinden. Beispiel: *media/pdf*
webpath	Pfad zum Verzeichnis, in dem die PDF-Vorlagen bei der Veröffentlichung abgelegt werden sollen. Beispiel: */opt/firstspirit5/web/fc_preview/WEB-INF/pdf* **Achten Sie darauf, dass Sie dasselbe Verzeichnis wie in der Modulkonfiguration für die PDF-Action angeben.** Der Standardwert ist *WEB-INF/pdf*.

Die nachfolgenden Parameter müssen Sie nur angeben, wenn die Veröffentlichung auf einem entfernten Server (*runlocal=false*) erfolgen soll.

Parameter	Beschreibung
webuser	Benutzername, mit dem sich auf einem entfernten Server angemeldet werden soll.
webhost	Host-Name oder IP-Adresse des entfernten Servers.
ssh	Pfad zum Programm ssh. Beispiel: *C:\cygwin\bin\ssh.exe*
privkey	Auf Windows-Systemen können Sie hier den vollständigen Pfad zur SSH-Schlüsseldatei des im Parameter *webuser* konfigurierten Benutzers angeben. Beispiel: *c:\User\fs5\.ssh\id_rsa*

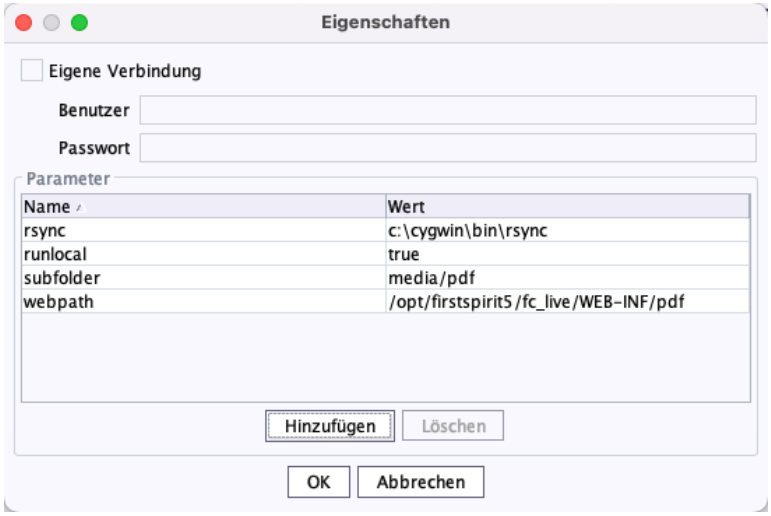


Abbildung 4.36. Konfigurationsbeispiel des Deploymentscripts der PDF-Action auf einem Windows-Server

Speichern Sie abschließend die Auftragsvorlage.

Verwenden Sie die neue Auftragsvorlage, um einen Veröffentlichungsauftrag anzulegen, mit dem die Redaktion die PDF-Vorlagen in die Preview-Webapp (*fc_preview*) übertragen kann (siehe dazu auch Kapitel 3.4.3 im Formcentric Benutzerhandbuch).

Damit bei jedem Deployment auch die PDF-Vorlagen übertragen werden, sollten Sie die Auftragsvorlage in alle relevanten Veröffentlichungsaufträge einbinden. Nähere Informationen zu Aufträgen und der Auftragsverwaltung von FirstSpirit finden Sie im Kapitel 7.5 der FirstSpirit „Dokumentation für Administratoren“.

4.10. Verschlüsselung von Passwörtern

In der Grundkonfiguration werden Zugangsdaten zu Datenbanken, Mail-Servern etc. im Klartext in verschiedenen Konfigurationsdateien gespeichert. Bei einem eventuellen Einbruch in den Server können so gültige Zugangsdaten entwendet werden. Aus diesem Grund haben Sie die Möglichkeit, Passwörter auch verschlüsselt abzu-

legen. Die Passwörter werden in diesem Fall erst beim Start der Anwendung mit dem hinterlegten Verschlüsselungspasswort entschlüsselt. Das zur Verschlüsselung verwendete Passwort müssen Sie vor dem Start der Formcentric Webapplikationen in einer Umgebungsvariable speichern. Standardmäßig verwendet Formcentric hierfür die Umgebungsvariable *MWF_ENCRYPTION_PASSWORD*.

```
export MWF_ENCRYPTION_PASSWORD=my-encryption-password
```

In der Formcentric Konfigurationsoberfläche haben Sie an jedem Passworteingabefeld die Möglichkeit, das eingegebene Passwort zu verschlüsseln.

Abbildung 4.37. Passwortfeld mit Verschlüsselungsfunktion

Klicken Sie hierfür auf das Schloss-Symbol innerhalb des Feldes. Dies öffnet einen Dialog, in dem Sie ein Verschlüsselungspasswort eingeben können, mit dem das Passwort verschlüsselt wird. Bitte beachten Sie, dass Sie für alle zu verschlüsselnden Passwörter das auf dem System hinterlegte Verschlüsselungspasswort verwenden müssen. Eine Prüfung, ob das in dem Dialog eingegebene Passwort mit dem hinterlegten übereinstimmt, findet nicht statt.

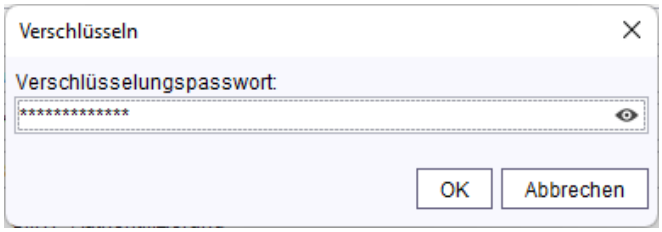


Abbildung 4.38. Dialog zum Ver- und Entschlüsseln eines Passworts

Wenn Sie nicht die FirstSpirit Administrationsoberfläche verwenden, um Formcentric zu installieren, haben Sie alternativ auch die Möglichkeit Passwörter mithilfe eines Komandozeilenprogramms zu ver- und entschlüsseln. Die so verschlüsselten Passwörter müssen Sie anschließend manuell in die entsprechende Konfiguration eintragen.

Laden Sie das Programm aus dem Formcentric Maven-Repository herunter, indem Sie folgende Befehlszeile in der Konsole ausführen. Die hierfür erforderlichen Zugangsdaten erhalten Sie über unseren Helpdesk (support@formcentric.com).

```
mvn org.apache.maven.plugins:maven-dependency-plugin:3.0.2:copy \
-Dartifact=com.monday.webforms:encryption-cli:1.0:jar \
-DoutputDirectory=.
```

Um ein Passwort zu verschlüsseln, geben Sie in der Kommandozeile folgenden Befehl ein:

```
java -jar encryption-cli-1.0.jar \
```

```
  -p '<encryption-password>' -e '<password>'
```

Bitte beachten Sie, dass die Parameter in einfachen Anführungszeichen angegeben werden müssen. Folgende Kommandozeilenparameter können Sie beim Start angeben:

Parameter	**Beschreibung**
-p encryption-password	Passwort, das zur Ver- oder Entschlüsselung verwendet wird.
-d	Passwort entschlüsseln
-e	Passwort verschlüsseln
-?	Hilfe anzeigen

5. Erweiterung des FirstSpirit-Projekts

Nachdem Sie die Projektkomponente installiert haben, müssen Sie einige manuelle Änderungen an den importierten Ressourcen und an bestehenden Seitenvorlagen vornehmen.

5.1. Absatzvorlage

Das System bildet Formulare auf Content-Ebene durch eine Absatzvorlage ab. Diese können Sie wie nachfolgend beschrieben an Ihre konkreten Anforderungen anpassen. Im Lieferumfang von Formcentric sind bereits zwei Absatzvorlagen enthalten, wovon die Absatzvorlage *Formular-Headless* für die Verwendung mit der Headless-Webanwendung – und die Absatzvorlage *Formular* für die Verwendung mit der Spring-MVC-Webanwendung vorgesehen ist. Beide Absatzvorlagen unterscheiden sich dabei standardmäßig nur im Register *Internet (HTML)*.

5.1.1. Register Eigenschaften

Im Register *Eigenschaften* haben Sie unter anderem die Möglichkeit, Vorgabewerte für neue Formulare festzulegen. Mit Klick auf den Button *Vorgabewerte* öffnen Sie den Formulareditor in einem neuen Fenster. Darin können Sie für jede Projektsprache ein Formular anlegen, das vom System künftig als Vorlage für neuangelegte Formulare verwendet wird.

5.1.2. Register Formular

Verwenden Sie für Formulare die Eingabekomponente *FORMCENTRIC_FORMEDITOR*. Konfigurieren Sie diese im Register *Formular*. Sie finden eine ausführliche Beschreibung der Komponente in den Benutzerhandbüchern (*formcentric_contencreator_de.pdf* und *formcentric_sitearchitect_de.pdf*).

```
<?xml version="1.0" encoding="UTF-8"?>
<FORMCENTRIC_FORMEDITOR name="form" expandOnStartup="yes" hFill="yes"
    useLanguages="yes">
    <DATASOURCES>
        <DATASOURCE name="Countries" type="comboBox">
            <LANGINFOS>
                <LANGINFO lang="*" label="English country names"/>
                <LANGINFO lang="DE" label="Englische Ländernamen"/>
            </LANGINFOS>
        </DATASOURCE>
        <DATASOURCE name="Laender" type="comboBox">
            <LANGINFOS>
                <LANGINFO lang="*" label="German country names"/>
                <LANGINFO lang="DE" label="Deutsche Ländernamen"/>
            </LANGINFOS>
        </DATASOURCE>
        <DATASOURCE name="Countries" type="inputField">
            <LANGINFOS>
                <LANGINFO lang="*" label="English country names"/>
```

```
                <LANGINFO lang="DE" label="Englische Ländernamen"/>
            </LANGINFOS>
        </DATASOURCE>
        <DATASOURCE name="Laender" type="inputField">
            <LANGINFOS>
                <LANGINFO lang="*" label="German country names"/>
                <LANGINFO lang="DE" label="Deutsche Ländernamen"/>
            </LANGINFOS>
        </DATASOURCE>
        ...
    </DATASOURCES>
    <FILE_TYPES>
        <FILE_TYPE name="bmp">
            <LANGINFOS>
                <LANGINFO lang="*" label="Windows-Bitmap (*.bmp)"/>
            </LANGINFOS>
        </FILE_TYPE>
        <FILE_TYPE name="gif">
            <LANGINFOS>
                <LANGINFO lang="*" label="CompuServe-Bitmap (*.gif)"/>
            </LANGINFOS>
        </FILE_TYPE>
        <FILE_TYPE name="jpeg">
            <LANGINFOS>
                <LANGINFO lang="*" label="JPEG Image (*.jpeg)"/>
                <LANGINFO lang="DE" label="JPEG Bild (*.jpeg)"/>
            </LANGINFOS>
        </FILE_TYPE>
        ...
    </FILE_TYPES>
    <FORM_LAYOUTS>
        <FORM_LAYOUT name="mwf-separator">
            <LANGINFOS>
                <LANGINFO lang="*" label="Horitzontal seperator"/>
                <LANGINFO lang="DE" label="Horizontale Trennlinie"/>
            </LANGINFOS>
        </FORM_LAYOUT>
    </FORM_LAYOUTS>
    <FORM_VARIABLES>
        <FORM_VARIABLE name="date"/>
        <FORM_VARIABLE name="time"/>
        <FORM_VARIABLE name="url"/>
        <FORM_VARIABLE name="language"/>
        <FORM_VARIABLE name="ip"/>
        <FORM_VARIABLE name="userAgent"/>
        <FORM_VARIABLE name="referer"/>
    </FORM_VARIABLES>
    <INPUT_STYLE_CLASSES>
        <INPUT_STYLE_CLASS name="mwf-s" type="inputField">
            <LANGINFOS>
                <LANGINFO lang="*" label="Small width (mwf-s)"/>
                <LANGINFO lang="DE" label="Kleine Breite (mwf-s)"/>
            </LANGINFOS>
        </INPUT_STYLE_CLASS>
        <INPUT_STYLE_CLASS name="mwf-m" type="inputField">
            <LANGINFOS>
```

```
                <LANGINFO lang="*" label="Medium width (mwf-m)"/>
                <LANGINFO lang="DE" label="Mittlere Breite (mwf-m)"/>
            </LANGINFOS>
        </INPUT_STYLE_CLASS>
        <INPUT_STYLE_CLASS name="mwf-l" type="inputField">
            <LANGINFOS>
                <LANGINFO lang="*" label="Large width (mwf-l)"/>
                <LANGINFO lang="DE" label="Große Breite (mwf-l)"/>
            </LANGINFOS>
        </INPUT_STYLE_CLASS>
        <INPUT_STYLE_CLASS name="mwf-s" type="checkBoxGroup">
            <LANGINFOS>
                <LANGINFO lang="*" label="Small width (mwf-s)"/>
                <LANGINFO lang="DE" label="Kleine Breite (mwf-s)"/>
            </LANGINFOS>
        </INPUT_STYLE_CLASS>
        <INPUT_STYLE_CLASS name="mwf-m" type="checkBoxGroup">
            <LANGINFOS>
                <LANGINFO lang="*" label="Medium width (mwf-m)"/>
                <LANGINFO lang="DE" label="Mittlere Breite (mwf-m)"/>
            </LANGINFOS>
        </INPUT_STYLE_CLASS>
        ...
    </INPUT_STYLE_CLASSES>
    <LANGINFOS>
        <LANGINFO lang="*" label="Form"/>
        <LANGINFO lang="DE" label="Formular"/>
    </LANGINFOS>
    <MAIL_ACTION>
        <MAIL_FORMATS>
            <MAIL_FORMAT name="text">
                <LANGINFOS>
                    <LANGINFO lang="*" label="Plain Text"/>
                    <LANGINFO lang="DE" label="Text" />
                </LANGINFOS>
            </MAIL_FORMAT>
            <MAIL_FORMAT name="html">
                <LANGINFOS>
                    <LANGINFO lang="*" label="HTML"/>
                    <LANGINFO lang="DE" label="HTML"/>
                </LANGINFOS>
            </MAIL_FORMAT>
            <MAIL_FORMAT name="freetext">
                <LANGINFOS>
                    <LANGINFO lang="*" label="Freemarker (Plain Text)"/>
                    <LANGINFO lang="DE" label="Freemarker (Text)"/>
                </LANGINFOS>
            </MAIL_FORMAT>
            <MAIL_FORMAT name="freehtml">
                <LANGINFOS>
                    <LANGINFO lang="*" label="Freemarker (HTML)"/>
                    <LANGINFO lang="DE" label="Freemarker (HTML)"/>
                </LANGINFOS>
            </MAIL_FORMAT>
        </MAIL_FORMATS>
    </MAIL_ACTION>
```

```
    <PDF_ACTION templateFolder="pdf,pdf2"/>
    <PHONE_NUMBER_TYPES>
        <PHONE_NUMBER_TYPE name="FIXED_LINE">
            <LANGINFOS>
                <LANGINFO lang="*" label="Fixed-line"/>
                <LANGINFO lang="DE" label="Festnetz"/>
            </LANGINFOS>
        </PHONE_NUMBER_TYPE>
        <PHONE_NUMBER_TYPE name="MOBILE">
            <LANGINFOS>
                <LANGINFO lang="*" label="Mobile"/>
                <LANGINFO lang="DE" label="Mobil"/>
            </LANGINFOS>
        </PHONE_NUMBER_TYPE>
        <PHONE_NUMBER_TYPE name="FIXED_LINE_OR_MOBILE">
            <LANGINFOS>
                <LANGINFO lang="*" label="Fixed-line or Mobile"/>
                <LANGINFO lang="DE" label="Festnetz oder Mobil"/>
            </LANGINFOS>
        </PHONE_NUMBER_TYPE>
        <PHONE_NUMBER_TYPE name="TOLL_FREE">
            <LANGINFOS>
                <LANGINFO lang="*" label="Toll-free"/>
                <LANGINFO lang="DE" label="Gebührenfrei"/>
            </LANGINFOS>
        </PHONE_NUMBER_TYPE>
        <PHONE_NUMBER_TYPE name="PREMIUM_RATE">
            <LANGINFOS>
                <LANGINFO lang="*" label="Premium Rate"/>
                <LANGINFO lang="DE" label="Premium-Tarif"/>
            </LANGINFOS>
        </PHONE_NUMBER_TYPE>
        <PHONE_NUMBER_TYPE name="SHARED_COST">
            <LANGINFOS>
                <LANGINFO lang="*" label="Shared Cost"/>
                <LANGINFO lang="DE" label="geteilte Kosten"/>
            </LANGINFOS>
        </PHONE_NUMBER_TYPE>
        <PHONE_NUMBER_TYPE name="VOIP">
            <LANGINFOS>
                <LANGINFO lang="*" label="Voice over IP"/>
                <LANGINFO lang="DE" label="Voice-over-IP"/>
            </LANGINFOS>
        </PHONE_NUMBER_TYPE>
        <PHONE_NUMBER_TYPE name="PERSONAL_NUMBER">
            <LANGINFOS>
                <LANGINFO lang="*" label="Personal Numbers"/>
                <LANGINFO lang="DE" label="Persönliche Nummer"/>
            </LANGINFOS>
        </PHONE_NUMBER_TYPE>
        ...
    </PHONE_NUMBER_TYPES>
    <REDIRECT_ACTION httpsOnly="no" supportDynamicUrls="yes">
        <ALLOWED_HOSTS>
            <ALLOWED_HOST name="*"/>
        </ALLOWED_HOSTS>
```

```
    </REDIRECT_ACTION>
    <REGEX_PATTERNS>
      <REGEX_PATTERN name="^[0-9]*$">
        <LANGINFOS>
          <LANGINFO lang="*" label="Ciphers"/>
          <LANGINFO lang="DE" label="Nummern"/>
        </LANGINFOS>
      </REGEX_PATTERN>
      <REGEX_PATTERN name="^[0-9]{3,7}$">
        <LANGINFOS>
          <LANGINFO lang="*" label="3-7 ciphers"/>
          <LANGINFO lang="DE" label="3-7 Nummern"/>
        </LANGINFOS>
      </REGEX_PATTERN>
      <WEBEDIT_MACROS>
          <WEBEDIT_MACRO>{"name":"File upload", ... }</WEBEDIT_MACRO>
      </WEBEDIT_MACROS>

      ...
    </REGEX_PATTERNS>
</FORMCENTRIC_FORMEDITOR>
```

Der nachfolgende Abschnitt beschreibt die verschiedenen Tags und deren Attribute, die Sie bei der Konfiguration des Formulareditors verwenden können.

FORMCENTRIC_FORMEDITOR

Eingabekomponente für Formulare (Formulareditor)

Attribut	**Beschreibung**
name	Über das Attribut *name* geben Sie dem Formulareditor einen Variablennamen. Diesen können Sie anschließend in den Vorlagen nutzen, um die verschlüsselte Text-Repräsentation mit Hilfe von $CMS_VALUE()$ auszugeben (siehe Abschnitt 5.1.3, „Register Internet (HTML)“).
expandOnStartup	Mit dem Parameter *expandOnStartup* legen Sie fest, ob die Formularelemente in der Baumansicht beim Öffnen des Editors oder Speichern des Formulars ausgeklappt werden oder nicht.
excludeElements	Mit dem Parameter *excludeElements* können Sie Formular-elementtypen im Formulareditor komplett ausblenden. Geben Sie als Wert eine kommaseparierte Liste mit den Namen der auszublendenden Elementtypen an. Soll beispielsweise die Verwendung von Captchas und Seitenumbrüchen unterbunden werden, so geben Sie den Wert *captcha,pageBreak* an. Folgende Elementtypen stehen Ihnen zur Verfügung: *button, checkBoxGroup, comboBox, condition, fileUpload, hiddenField, shortText, emailField, numberField, dateField, phoneField, inputField, layout, fieldSet, pageBreak, paragraph,*

Attribut	Beschreibung
	passwordField, radioGroup, calculatedValue, pageCondition, summary, textArea, captcha, sequenceAction, mailAction, dataSourceAction, datastoreAction, redirectAction, mediaStore-Action, pdfAction, webhookAction
hFill	Der Formulareditor wird mit einer vordefinierten Breite angezeigt. Wollen Sie die volle zur Verfügung stehende Anzeigebreite ausnutzen, geben Sie den Parameter *hFill* mit dem Wert *YES* an. Aufgrund des größeren Platzbedarfs dieser Eingabekomponente wird die Einstellung *YES* empfohlen.
useLanguages	Mit dem Parameter *useLanguages* legen Sie fest, ob der Formulareditor für verschiedene Sprachen unterschiedliche Formulardefinitionen speichern soll oder nicht (mehrsprachige Pflege).

Darüber hinaus können Sie die Standardparameter *hidden, label, noBreak, preset, convertEntities* und *allowEmpty* angeben. Eine ausführliche Beschreibung dieser Parameter finden Sie in der Dokumentation von FirstSpirit.

FORM_VARIABLES

Mithilfe des Tags *FORM_VARIABLES* definieren Sie Variablen, die ein Redakteur anschließend bei der Vorbelegung von Eingabefeldern im Feld *Vorbelegung* des ein- und mehrzeiligen Texteingabefeldes verwenden kann. Dadurch kann er beispielsweise ein Eingabefeld mit dem aktuellen Datum (Variable *date*) vorbelegen.

Standardmäßig können Sie die Variablen *date, time, serverDate, serverTime, client-Date, clientTime, timezone, url, language, ip, userAgent* und *referer* verwenden. Sie können darüber hinaus jedoch auch eigene Variablen definieren. Siehe dazu auch Abschnitt 6.5.7, „Variable zur Vorbelegung von Formularfeldern hinzufügen“.

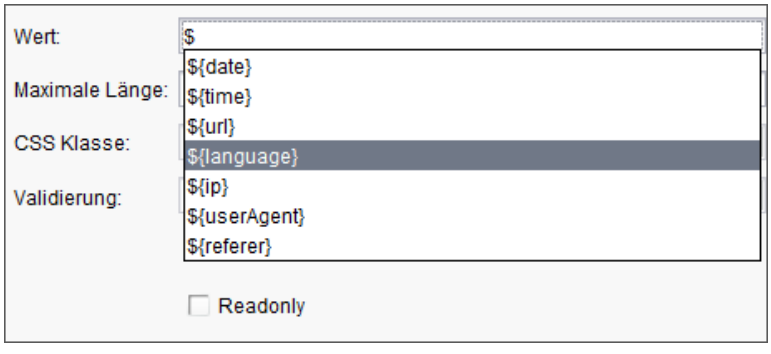


Abbildung 5.1. Vorschlagsliste mit Variablen beim einzeiligen Textfeld

FORM_VARIABLE

Mit dem Tag *FORM_VARIABLE* geben Sie den Namen einer Formularvariable an.

Attribut	Beschreibung
name	Name der Formularvariablen.

FORM_LAYOUTS

Das Tag *FORM_LAYOUTS* ermöglicht die Definition einer Liste von Layout-Namen, die der Redakteur später über das Layout-Element auswählen kann.

FORM_LAYOUT

Mit dem Tag *FORM_LAYOUT* geben Sie den Namen eines Layouts an.

Attribut	Beschreibung
name	Name des Layouts.

Eine optionale sprachabhängige Beschriftung des Layouts kann durch Angabe eines untergeordneten *<LANGINFOS>* Tags festgelegt werden (siehe „LANGINFOS“).

FIELD_WIDTHS

Das Tag *FIELD_WIDTHS* ermöglicht Ihnen, eine Liste von CSS-Klassen zu definieren, aus der der Redakteur später bei der Anlage eines Formularelements auswählen kann, um die Breite des Formularelements festzulegen.

Im Standardlieferumfang sind bereits die CSS-Klassen *mwf-s, mwf-m* sowie *mwf-l* enthalten, mit deren Hilfe die Breite von Eingabeelementen auf der Formularseite beeinflusst werden kann.

FIELD_WIDTH

Attribut	Beschreibung
name	Name der CSS-Klasse.
type	Formularelementtyp, auf dem die CSS-Klasse für den Redakteur auswählbar sein soll. Standardmäßig können Sie CSS-Klassen auf folgenden Elementtypen konfigurieren: *inputField, textArea, passwordField, checkBoxGroup, comboBox, paragraph, radioGroup, shortText, emailField, numberField, dateField, phoneField, fileUpload, pageBreak*.

Eine optionale sprachabhängige Beschriftung der CSS-Klasse kann durch Angabe eines untergeordneten *<LANGINFOS>* Tags festgelegt werden (siehe „LANGINFOS“).

INPUT_STYLE_CLASSES

Das Tag *INPUT_STYLE_CLASSES* ermöglicht Ihnen, eine Liste von CSS-Klassen zu definieren, aus der der Redakteur später bei der Anlage eines Formularelements auswählen kann.

INPUT_STYLE_CLASS

Attribut	Beschreibung
name	Name der CSS-Klasse.
type	Formularelementtyp, auf dem die CSS-Klasse für den Redakteur auswählbar sein soll. Standardmäßig können Sie CSS-Klassen auf folgenden Elementtypen konfigurieren: *form, inputField, textArea, passwordField, checkBoxGroup, comboBox, paragraph, radioGroup, shortText, emailField, numberField, dateField, phoneField, summary, fieldSet fileUpload, pageBreak*.

Eine optionale sprachabhängige Beschriftung der CSS-Klasse kann durch Angabe eines untergeordneten *<LANGINFOS>* Tags festgelegt werden (siehe „LANGINFOS“).

DATASOURCES

Mit dem Tag *DATASOURCES* können Sie dynamische Datenquellen (REST-Services) definieren, die der Redakteur anschließend auswählen kann, um ein Formularelement mit Werten zu füllen (siehe Abschnitt 6.5.8, „Implementierung eines REST-Services“).

DATASOURCE

Attribut	Beschreibung
name	Name des REST-Services, so wie Sie ihn in das Service-Mapping des REST-Controllers eingetragen haben.
type	Formularelementtyp, auf dem der Redakteur die Datenquelle auswählen kann. Standardmäßig können Sie Datenquellen nur auf Auswahllisten vom Typ *comboBox, radioGroup, checkBoxGroup, inputField* und *hiddenField* konfigurieren.

Eine optionale sprachabhängige Beschriftung der Datenquelle kann durch Angabe eines untergeordneten *<LANGINFOS>* Tags festgelegt werden (siehe „LANGINFOS“).

FILE_TYPES

Mit dem Tag *FILE_TYPES* definieren Sie Dateitypen, die der Redakteur auf dem Datei-Validator auswählen kann.

FILE_TYPE

Attribut	Beschreibung
name	Name des Dateityps, so wie er in das MIME-Type-Mapping des Datei-Validators eingetragen wurde. Standardmäßig können Sie folgende Dateitypen konfigurieren: *bmp, gif, jpeg, png, wmf, tif, mp3, wav, mp4, avi, mpg, wma, mov, asf, wmv, zip, gz, tar, gtar, 7z, rar, arj, bz, bz2, doc, ppt, pptx, xls, xlsx, docx, mpx, wps, pdf, rtf, flv, txt, dvi, xml, js, css, xhtml, html, svg, psd, rpm, indd, inx*

Eine optionale sprachabhängige Beschriftung des Dateityps kann durch Angabe eines untergeordneten *<LANGINFOS>* Tags festgelegt werden (siehe „LANGINFOS“).

PHONE_NUMBER_TYPES

Mit dem Tag *PHONE_NUMBER_TYPES* definieren Sie Telefonnummertypen, die der Redakteur auf dem Telefonnummer-Validator auswählen kann.

PHONE_NUMBER_TYPE

Attribut	Beschreibung
name	Name des Telefonnummertypen, so wie er in das MIME-Type-Mapping des Telefonnummer-Validators eingetragen wurde. Standardmäßig können Sie folgende Telefonnummertypen konfigurieren: *FIXED_LINE, MOBILE, FIXED_LINE_OR_MOBILE, TOLL_FREE, PREMIUM_RATE, SHARED_COST, VOIP, PERSONAL_NUMBER, UAN, VOICEMAIL, UNKNOWN*

Eine optionale sprachabhängige Beschriftung des Telefonnummertyps kann durch Angabe eines untergeordneten *<LANGINFOS>* Tags festgelegt werden (siehe „LANGINFOS“).

REGEX_PATTERNS

Mit dem Tag *REGEX_PATTERNS* definieren Sie reguläre Ausdrücke, die der Redakteur auf dem Reguläre Ausdrücke-Validator auswählen kann.

REGEX_PATTERN

Attribut	Beschreibung
name	Der reguläre Ausdruck welchen Sie hiermit in die Liste der Beispiele im Reguläre Ausdrücke-Validator hinzufügen.

Eine optionale sprachabhängige Beschriftung des regulären Ausdrucks kann durch Angabe eines untergeordneten *<LANGINFOS>* Tags festgelegt werden (siehe „LANGINFOS“).

MAIL_ACTION

Mit diesem Tag konfigurieren Sie den Redaktionsdialog der Mail-Action.

MAIL_FORMATS

Das Tag *MAIL_FORMATS* ermöglicht Ihnen die Definition einer Liste von Formatbezeichnern, die der Redakteur später innerhalb der Mail-Action auswählen kann, um die Formatierung der vom System versendeten E-Mail festzulegen. Wenn Sie keine Formatbezeichner angeben, so werden standardmäßig die Formate *html* und *text* zur Auswahl angeboten.

MAIL_FORMAT

Attribut	Beschreibung
name	Name des E-Mail-Formats, so wie Sie ihn in das *bodyRendererMapping* der Mail-Action eingetragen haben (siehe dazu auch „formcentric-actions.xml“).

Eine optionale sprachabhängige Beschriftung des E-Mail-Formats kann durch Angabe eines untergeordneten *<LANGINFOS>* Tags festgelegt werden (siehe „LANGINFOS“).

PDF_ACTION

Mit diesem Tag konfigurieren Sie den Redaktionsdialog der PDF-Action.

Attribut	Beschreibung
templateFolder	Alle PDF-Dokumente, die als Vorlage für die PDF-Action verwendet werden sollen, müssen in einem zentralen Ordner der Medienverwaltung abgelegt werden. Geben Sie hier den Referenznamen eines Ordners oder Komma separiert für mehrere Ordner an. Im Redaktionsdialog der PDF-Action ist die Auswahl auf PDF-Dokumente auf diese Ordner beschränkt.

REDIRECT_ACTION

Mit diesem Tag konfigurieren Sie den Redaktionsdialog der Weiterleitungs-Action.

Attribut	Beschreibung
httpsOnly	Mit dem Parameter *httpsOnly* legen Sie fest, dass nur sichere URLs als Weiterleitungsziel eingegeben werden dürfen.

Attribut	Beschreibung
supportDynamicUrls	Mit dem Parameter *supportDynamicUrls* legen Sie fest, ob die Weiterleitungs-URL Variablen der Form *${feld-name}* enthalten darf oder nicht.

ALLOWED_HOSTS

Mit diesem Tag haben Sie die Möglichkeit, die möglichen Weiterleitungsziele auf bestimmte Adressen (Hosts) einzuschränken.

ALLOWED_HOST

Angabe eines erlaubten Hosts.

Möchten Sie beispielsweise sicherstellen, dass nur Webseiten Ihres Unternehmens als Weiterleitungsziel angegeben werden dürfen, so tragen Sie hier den Domain-Namen Ihres Unternehmens ein. Dabei können Sie Platzhalter wie *.ihre-domain.de verwenden, um beispielsweise auch alle Subdomains zuzulassen.

Attribut	Beschreibung
name	Name eines zulässigen Hosts.

LANGINFOS

Mit diesem Tag können Sie für das Attribut *name* des übergeordneten Tags sprachabhängige Labels für die Anzeige in der Redaktionsoberfläche angeben. Dies gilt für die Tags *DATASOURCE, FILE_TYPE, INPUT_STYLE_CLASS, MAIL_FORMAT, FORM_LAYOUT, PHONE_NUMBER_TYPE* und *REGEX_PATTERN*.

LANGINFO

Mit diesem Tag wird das Label für eine konkrete Sprache festgelegt.

Attribut	Beschreibung
lang	Sprachkürzel der Anzeigesprache (z. B. *DE* für Deutsch, *EN* für Englisch).
label	Text des Labels in der angegebenen Sprache
description	Mit dem Parameter *description* können Sie eine optionale Beschreibung angeben, die zur Anzeige eines Tooltipps (Mouse-Over) genutzt wird.

5.1.3. Register Internet (HTML)

Das System stellt Formulare grundsätzlich dynamisch dar. So können beispielsweise einzelne Formularseiten oder Formularfelder – abhängig von den Eingaben des Benutzers – ein- oder ausgeblendet werden. Aus diesem Grund wird die HTML-Ausgabe des Formulars entweder serverseitig durch die Webanwendug oder browserseitig durch eine React-Anwendung (Formcentric-Client) erzeugt und dynamisch

in die jeweilige Seite eingebettet. Beim serverseitigen Ansatz erfolgt der Aufruf der WebApp-Komponente als asynchroner Javascript Request (AJAX). Die Absatzvorlage erzeugt dabei nur das Javascript zum Einbetten des AJAX-Response. Beim browserseitigen Ansatz hingegen wird der Formcentric-Client direkt im Browser ausgeführt. Die Absatzvorlage erzeugt in diesem Fall das Javascript zum Einbetten des Clients.

Bei Verwendung der Spring-MVC-Webanwendung müssen die benötigten Javascript-Libraries und CSS-Styles in der Seitenvorlage (siehe Abschnitt 5.6, „Seitenvorlage“) geladen werden. Bei Verwendung der Headless-Webanwendung werden diese in der Absatzvorlage referenziert und vom Formcentric-Client geladen.

Das folgende Beispiel zeigt das Ausgabe-Template für die Verwendung mit der Spring-MVC-Webanwendung:

```
$CMS_RENDER(script: "formcentric_encrypted_form",
            form: form.XML)$
$CMS_RENDER(script: "formcentric_encrypted_refs",
            ids: form.internalReferences)$
$CMS_RENDER(script: "formcentric_login_ticket")$

<!-- Formcentric -->
<div class="clearfix module"$CMS_VALUE(editorId())$>
  <div class="mwf-form">
    <div id="ajaxreplace$CMS_VALUE(form.uid)$">
      <script type="text/javascript">
        jQuery(function() {
          jQuery.mwfAjaxReplace({
            uid: '$CMS_VALUE(form.uid)$',
            selector: '#ajaxreplace$CMS_VALUE(form.uid)$',
            url: '$CMS_RENDER(script:"formcentric_url")$/servlet/form',
            appendUrlVars: true,
            data: {
              _view: 'webform',
              _fd: '$CMS_VALUE(fc_encryptedForm)$',
              _refs: '$CMS_VALUE(fc_encryptedRefs)$',
              _lang: '$CMS_VALUE(form.lang())$',
              _ticket: '$CMS_VALUE(fc_loginTicket)$'
            }
          });
        });
      </script>
    </div>
  </div>
</div>
```

Bei jedem Formular-Submit wird der DIV-Tag mit der ID *ajaxreplace<Formular-Id>* durch den AJAX-Response ersetzt.

Für die Headless-Webanwendung sieht das Ausgabe-Template wir nachfolgend dargestellt aus:

```
$CMS_RENDER(script: "formcentric_encrypted_form",
            form: form.XML)$
```

```
$CMS_RENDER(script: "formcentric_encrypted_refs",
            ids: form.internalReferences)$
$CMS_RENDER(script: "formcentric_login_ticket")$

<!-- Formcentric -->
<div class="clearfix module"$CMS_VALUE(editorId())$>
   <div data-fc-id="$CMS_VALUE(form.uid)$"
        data-fc-formapp-url="$CMS_REF(media:"formapp")$"
        data-fc-theme-url="$CMS_REF(media:"formcentric_headless_css")$"
        data-fc-template-url="$CMS_REF(media:"formcentric_templates_js")$"
        data-fc-theme-variable-url="$CMS_REF(media:"formcentric_json")$"
        data-fc-form-definition="$CMS_VALUE(fc_encryptedForm)$"
        data-fc-refs="$CMS_VALUE(fc_encryptedRefs)$"
        data-fc-params='{"ticket": "$CMS_VALUE(fc_loginTicket)$"}'
        data-fc-data-url='$CMS_RENDER(script:"formcentric_url")$'
      ></div>
</div>
```

Die Konfiguration des Formcentric-Clients erfolgt über data-Attribute:

data-fc-id: Angabe der Formular-ID.

data-fc-formapp-url: Angabe der URL der Javascript-Datei des Formcentric-Clients.

data-fc-theme-url: Die URL zu den CSS-Styles der Anwendung.

data-fc-template-url: Die URL zu der Javascript-Datei der Formular-Templates.

data-fc-theme-variable-url: Die URL zu der JSON-Datei mit dem Theme-Variablen.

data-fc-form-definition: Die verschlüsselte Formulardefinition.

data-fc-refs: Die verschlüsselten Referenzen des Formulars.

data-fc-params: Angabe zusätzliche Parameter, die an die Headless-Anwendung übertragen werden als JSON-Objekt. Für die Anmeldung der Headless-Anwendung an die Vorschau wird hier standardmäßig ein Login-Ticket übergeben.

data-fc-data-url: Die URL der Headless-Webanwendung.

5.2. Skript formcentric_headless_url

Das System verwendet das Generierungsskript *formcentric_headless_url* innerhalb der Absatzvorlage, um die Kontextpfade der globalen Headless-Webapplikationen auszugeben (siehe Abschnitt 4.1, „Installation des Moduls Formcentric“).

Achten Sie darauf, dass die Werte der Skriptvariablen *previewWebAppId* und *liveWebAppId* die IDs der globalen Webapplikationen enthalten. Passen Sie diese gegebenenfalls an.

```
//!Beanshell
previewWebAppId = "fc_headless_preview";
liveWebAppId = "fc_headless_live";
```

```
agent = context.requestSpecialist(
    de.espirit.firstspirit.agency.LegacyModuleAgent.TYPE);

url = context.isPreview() ? agent.getGlobalWebAppUrl(previewWebAppId) :
    agent.getGlobalWebAppUrl(liveWebAppId);

return url;
```

Für die Ausgabe der Kontextpfade der Spring-MVC-Webanwendung verwenden Sie bitte das Generierungsskript *formcentric_url*, welches analog aufgebaut ist.

5.3. Skript formcentric_encrypted_form

Die Formulardefinition muss vor dem Übertragen an die Formcentric Webapp verschlüsselt werden. Das Skript *formcentric_encrypted_form* ruft ein Executable auf, das die Formulardefinition verschlüsselt und in der Variablen *fc_encryptedForm* im Seitenkontext speichert. Beim Aufruf des Skripts muss die unverschlüsselte Formulardefinition im Skriptparameter *form* übergeben werden.

```
$CMS_RENDER(script: "formcentric_encrypted_form", form: form.XML)$
```

5.4. Skript formcentric_encrypted_refs

Verweise innerhalb der Formulardefinition auf FirstSpirit Objekte, wie z. B. Bilder oder PDF-Vorlagendokumente müssen analog zur Formulardefinition in verschlüsselter Form an die Formcentric Webapp übertragen werden. Hierfür steht Ihnen das Skript *formcentric_encrypted_refs* zur Verfügung. Das Skript ruft ein Executable auf, das die externen URLs zu den FirstSpirit-Objekten ermittelt und diese verschlüsselt. Die verschlüsselten URLs werden in der Variablen *fc_encryptedRefs* im Seitenkontext gespeichert. Beim Aufruf des Skripts müssen alle internen Verweise im Parameter *ids* an das Skript übergeben werden.

Über den optionalen Parameter *resolutions* können Sie eine kommaseparierte Liste von Auflösungen (Resolutions) angeben, für die die Bild-URLs erzeugt werden sollen. Wenn Sie den Parameter *resolutions* nicht angeben, werden die Bild-URLs für alle Auflösungen erzeugt.

```
$CMS_RENDER(script: "formcentric_encrypted_refs", ids: form.allReferences,
    resolutions: "47x47, 60x40")$
```

5.5. Skript formcentric_login_ticket

Für den Zugriff auf Inhalte der Preview-Webapp wird ein FirstSpirit Login-Ticket benötigt. Dieses wird beim Aufruf der Formcentric Webapp zusammen mit der

verschlüsselten Formulardefinition und den verschlüsselten internen Verweisen übertragen. Das Skript *formcentric_login_ticket* erzeugt ein Login-Ticket und speichert es in der Variablen *fc_loginTicket* im Seitenkontext. Wenn es sich nicht um den Preview-Context handelt, enthält die Variable einen leeren String.

```
$CMS_RENDER(script:"formcentric_login_ticket")$
```

5.6. Seitenvorlage

Für eine korrekte Anzeige der Formulare werden zusätzliche Javascripte und CSS-Styles benötigt. Diese wurden bei der Installation der Monday Formcentric Projektkomponente importiert.

Bei Verwendung der Spring-MVC-Webanwendung erweitern Sie die Ausgabe des HTML-Headers in Ihren Seitenvorlagen um das importierte CSS (*formcentric_css*) und die importierten Javascripte. Bitte achten Sie darauf, die Javascripte in der dargestellten Reihenfolge zu laden, da diese teilweise aufeinander aufbauen.

Alternativ können Sie auch die minifizierten Versionen der Skripte (*_min) verwenden.

```
...
<link rel="stylesheet" type="text/css"
    href="$CMS_REF(media:"formcentric_flex_css_min")$" />
...
$CMS_FOR(_script, [
    "json2_js_min",
    "jquery_3_6_0_js_min",
    "select2_4_0_13_js_min",
    "jquery_autocomplete_js_min",
    "jquery_format_1_3_js_min",
    "jquery_ui_widget_1_13_2_js_min",
    "load_image_all_js_min",
    "canvas_to_blob_js_min",
    "jquery_xdr_transport_js_min",
    "jquery_iframe_transport_js_min",
    "jquery_fileupload_10_31_0_js_min",
    "jquery_fileupload_process_10_31_0_js_min",
    "jquery_fileupload_image_10_31_0_js_min",
    "jquery_formcentric_1_9_js_min"
])$

<script type="text/javascript" src="$CMS_REF(media:_script)$"></script>
$CMS_END_FOR$
...
```

Bitte beachten Sie, dass die tatsächlichen Referenznamen von den hier angegebenen Namen abweichen können. FirstSpirit erweitert den Referenznamen eines Inhalts beim Import um eine fortlaufende Nummer, wenn bereits ein Inhalt dieses Namens existiert. Verwenden Sie in diesem Fall den erweiterten Referenznamen.

Für die Headless-Webanwendung muss in der Seitenvorlage lediglich ein Javascript geladen werden. Alle weiteren benötigten Javascripte und CSS-Styles werden in der zugehörigen Absatzvorlage *Headless-Formualar* referenziert und bei Aufruf eines Formulars durch den Formcentric-Client geladen (siehe Abschnitt 5.1.3, „Register Internet (HTML)“).

```
...
<script type="text/javascript"
        src="$CMS_REF(media:"formcentric")$" defer></script>
```

Die Angabe des *defer*-Attributs gewährleistet, dass das Javascript asynchron geladen wird und seine Ausführung erst nach dem vollständigen Parsen der Seite stattfindet.

5.7. Themes

Für die Gestaltung der Headless-Formulare bietet Formcentric eine Reihe von Themes, die individuell an Ihre Bedürfnisse angepasst werden können. Jedes Theme setzt sich aus einer Vorlagen-Datei, einer CSS-Datei und einer JSON-Datei zusammen. In der JSON-Datei sind spezifische CSS-Parameter enthalten, mit denen Sie das Erscheinungsbild des Themes Ihren Anforderungen entsprechend modifizieren können. Für weiterführende Anpassungen haben Sie die Möglichkeit den Quellcode aus dem Formcentric npm-Repository über folgende URL herunterzuladen:

https://maven.monday-consulting.com:443/artifactory/formcentric-npm/@formcentric/client/-/@formcentric/client-1.0.7.tgz

5.8. CSS

Formcentric beinhaltet bereits ein einfaches CSS, das Sie als Basis für die Gestaltung der Formulare in Verbindung mit den verwenden können. Die JSP-Templates der Spring-MVC-Anwendung verwenden die darin definierten CSS-Klassen. Das nachfolgende Beispiel zeigt Ihnen die generierte HTML-Repräsentation eines einfachen Kontaktformulars (Auszug).

```
<div id="ajaxreplaceDE2790">
  <form accept-charset="utf-8" class="mwf-form" data-mwf-id="DE187126"
        enctype="multipart/form-data" id="commandDE187126"
        method="post" onsubmit="return false;">

    <div class="mwf-layout mwf-layout--default">
      <div class="mwf-textinput" data-mwf-container="fccc48273e2eeb">
        <input id="fccc48273e2eeb" type="text" name="email"
               class="mwf-textinput__input" data-mwf-id="fccc48273e2eeb" />
        <label class="mwf-textinput__label" for="fccc48273e2eeb">Email</label>
      </div>

      <div class="mwf-textinput" data-mwf-container="fc313d3971298a">
```

```
        <input id="fc313d3971298a" name="message" type="text"
               class="mwf-textinput__input" data-mwf-id="fc313d3971298a" />
        <label class="mwf-textinput__label" for="fc313d3971298a">Message</label>
      </div>
    </div>
  </form>
</div>
```

Das System gibt das Formular als eine Abfolge von *div*-Containern aus, wobei jedes Formularelement in ein separates *div*-Element eingebettet ist. Die verschiedenen HTML-Elemente sind mit speziellen CSS-Klassen ausgezeichnet, sodass Sie sowohl das gesamte Formular als auch einzelne Bestandteile wie Eingabefelder, Labels und Fehlermeldungen mittels CSS gestaltet können. Um Überschneidungen mit anderen CSS-Klassen im Projekt zu vermeiden, beginnen alle Klassennamen mit dem Präfix *mwf*.

6. Programmierung und Anpassung

In diesem Abschnitt finden Sie aus Entwicklerperspektive beschrieben, wie Sie Formcentric erweitern oder verändern können. Sie benötigen dazu neben allgemeinen Java- und XML-Kenntnissen grundlegende Kenntnisse des Spring-Frameworks sowie des Maven Build-Systems.

6.1. Entwicklungs-Workspace

Um Ihnen einen sehr schnellen Einstieg in die Entwicklung von Erweiterungen zu ermöglichen, stellt Monday einen vorbereiteten Entwicklungs-Workspace zur Verfügung. Hierbei handelt es sich um einen Maven-Workspace, der alle in diesem Handbuch beschriebenen Beispiele beinhaltet.

Der Workspace umfasst neun Maven-Artefakte:

Artefaktverzeichnis	**Beschreibung**
/formcentric-admin-customizations	Erweiterungen der Server und Projektkonfiguration.
/formcentric-editor-customizations	Erweiterungen der Eingabekomponente
/formcentric-module-customizations	Modul-Descriptor und FirstSpirit-Projektressourcen
/formcentric-webapp-customizations	Erweiterungen der Spring-MVC-Webanwendung
/formcentric-webapp-lib-customizations	Java-Klassen für Erweiterungen der Spring-MVC-Webanwendung
/formcentric-headless-webapp-customizations	Erweiterungen des Headless-Servers (WAR Deployment)
/formcentric-headless-server-customizations	Erweiterungen des Headless-Servers (Embedded Web Server Deployment)
/formcentric-headless-firstspirit-customizations	Java-Klassen für Erweiterungen der Headless-Webanwendung
/formcentric-webedit-customizations	Erweiterungen der Web-Editor-Komponente
/formcentric-webedit-lib-customizations	Java-Klassen der erweiterten Web-Editor-Komponente

Alle benötigten JAR-Archive (Dependencies) werden beim Bauen des Workspace vom Monday Maven-Server (http://maven.monday-consulting.com) heruntergeladen. Die dafür notwendigen Zugangsdaten müssen Sie vorher in der Maven-Konfiguration *settings.xml* eintragen. Die Konfigurationsdatei finden Sie im Root-Verzeichnis des Entwicklungs-Workspace.

Um Ihre persönlichen Zugangsdaten zu erhalten, wenden Sie sich bitte an unseren Helpdesk (support@formcentric.com).

```
...
<servers>
  <server>
    <id>maven.monday-consulting.com</id>
    <username>my-username</username>
    <password>my-password</password>
  </server>
</servers>
...
```

Passen Sie anschließend die in der Root-POM (*pom.xml*) angegebene FirstSpirit-Version an die bei Ihnen eingesetzte Version an.

```
<cms.version>5.2.190105</cms.version>
```

Zum Bauen des Workspace führen Sie im Verzeichnis *<formcentric-firstspirit-workspace>* den folgenden Befehl über die Kommandozeile aus:

```
mvn -s ./settings.xml install
```

Das fertige FirstSpirit-Modularchiv wird im Verzeichnis *<formcentric-firstspirit-workspace>/formcentric-module-customizations/target* erstellt.

6.2. Monday Maven-Plugin

Jedes FirstSpirit-Modul enthält einen speziellen Modul-Descriptor (*module.xml*), der beschreibt, welche Komponenten und Dateien in dem entsprechenden Modul enthalten sind. Bei Formcentric sind dies neben Konfigurationen und Ausgabe-Templates vor allem JAR-Archive. Um die Erstellung des Modul-Descriptors zu vereinfachen, verwendet Formcentric ein spezielles Maven-Plugin, mit dem der Modul-Descriptor generiert wird. Das Plugin wertet die in den Maven-Konfigurationen definierten Dependencies aus und erstellt daraus die *<resource>*-Einträge im Modul-Descriptor.

Das Plugin wird in der Maven-Konfiguration (*pom.xml*) des Artefakts *formcentric-module-customizations* konfiguriert.

Folgende Parameter müssen Sie im *<configuration>*-Tag des Plugins angeben:

Parameter	**Beschreibung**
configXml	Verweis auf die Konfigurationsdatei für das Maven-Plugin
prototypeXml	Verweis auf die Datei, die als Vorlage für den zu erstellenden Modul-Descriptor dient.

```
<plugin>
```

```
  <groupId>com.monday-consulting.maven.plugins</groupId>
  <artifactId>fsm-maven-plugin</artifactId>
  <configuration>
    <configXml>${basedir}/target/extra-resources/fsm-plugin.xml</configXml>
    <prototypeXml>${basedir}/target/extra-resources/
        prototype.module.xml</prototypeXml>
    <targetXml>${basedir}/target/extra-resources/module.xml</targetXml>
  </configuration>
  <executions>
    <execution>
      <id>dependencyToXML</id>
      <phase>package</phase>
      <goals>
        <goal>dependencyToXML</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Als Basis für die Erzeugung des FirstSpirit Modul-Descriptors wird die Vorlage *prototype.module.xml* verwendet, die im Verzeichnis */formcentric-module-customizations/src/non-packaged-resources* abgelegt ist. In dieser Datei werden verschiedene Variablen verwendet, die beim Bauen des Workspace ersetzt werden. Geänderte Klassennamen etc. müssen darin konfiguriert werden.

```
<module>
  <name>Formcentric</name>
  <version>${project.version}</version>
  <description>Formcentric Form Editor</description>
  <vendor>Formcentric GmbH</vendor>
  <components>
    <public>
      <name>FORMCENTRIC_FORMEDITOR</name>
      <description>Formcentric editor component</description>
      <class>de.espirit.firstspirit.module.GadgetSpecification</class>
      <configuration>
        <gom>com.formcentric.examples.gom.CustomGomFormEditor</gom>
        <factory>com.formcentric.editor.
            gadgets.FormEditorSwingGadgetFactory</factory>
        <value>com.formcentric.editor.
            gadgets.FormValueEngineerFactory</value>
        <scope data="yes" content="yes" link="yes"/>
      </configuration>
      <resources>
        <resource>files/</resource>
        <!-- Variable, die durch das Maven-Plugin ersetzt wird. -->
        <dependencies>formcentric-editor-resources</dependencies>
      </resources>
    </public>

    <web-app scopes="global">
      <name>Formcentric WebApp</name>
      <description>Formcentric FIRSTspirit integration.</description>
      <configurable>com.formcentric.examples.
          admin.CustomWebAppConfiguration</configurable>
```

```
      <class>com.formcentric.admin.webapp.FormcentricWebApp</class>
      <web-xml>web.xml</web-xml>
      <resources>
        <!-- Variable, die durch das Maven-Plugin ersetzt wird. -->
        <dependencies>formcentric-admin-resources</dependencies>
      </resources>
      <web-resources>
        ..
        <!-- Variable, die durch das Maven-Plugin ersetzt wird. -->
        <dependencies>formcentric-webapp-resources</dependencies>
      </web-resources>
    </web-app>

    <project-app>
      <name>Formcentric Resources</name>
      <description>Formcentric project resources</description>
      <class>com.formcentric.admin.project.FormcentricProjectApp</class>
      <resources>
        <resource>files/</resource>
        <!-- Variable, die durch das Maven-Plugin ersetzt wird. -->
        <dependencies>formcentric-admin-resources</dependencies>
      </resources>
    </project-app>

  </components>
</module>
```

In der Konfigurationsdatei *fsm-plugin.xml* erfolgt die Definition der *<dependencies>*-Variablen mit den dazu gehörenden Abhängigkeiten.

```
<fsm-maven-plugin>
  <modules>
    <module>
      <id>change.this.now.fs5:formcentric-admin-customizations:jar</id>
      <dependencyTagValueInXml>formcentric-admin-resources
          </dependencyTagValueInXml>
      <firstSpiritScope>module</firstSpiritScope>
      <firstSpiritMode>isolated</firstSpiritMode>
    </module>
    <module>
      <id>change.this.now.fs5:formcentric-editor-customizations:jar</id>
      <dependencyTagValueInXml>formcentric-editor-resources
          </dependencyTagValueInXml>
      <firstSpiritScope>module</firstSpiritScope>
      <firstSpiritMode>isolated</firstSpiritMode>
    </module>
    <module>
      <id>change.this.now.fs5:formcentric-webapp-customizations:war</id>
      <dependencyTagValueInXml>formcentric-webapp-resources
          </dependencyTagValueInXml>
      <firstSpiritScope>module</firstSpiritScope>
    </module>
  </modules>
</fsm-maven-plugin>
```

Element	Beschreibung
<fsm-maven-plugin>	Root-Element.
<scopes>	Optionale Angabe der Maven-Scopes, die bei der Ermittlung der abhängigen Maven-Artefakte berücksichtigt werden sollen. Jeder Scope ist einzeln anzugeben. Ohne die Angabe von Scopes greifen die Standard-Scopes *compile* und *runtime*.
<scope>	Bezeichnung eines zu berücksichtigenden Maven-Scopes.
<modules>	Container-Element für die Definition von Artefakt-Dependencies.
<module>	Angabe eines Artefakts, dessen Dependencies in den Modul-Descriptor geschrieben werden sollen.
<id>	Der Maven-Identifier des Artefakts. Die Angabe muss in der bei Maven gebräuchlichen Notation *<groupId>:<artefactId>:<type>* erfolgen.
<dependencyTagValueInXml>	Name der *<dependencies>*-Variable, die in die Vorlage des Modul-Descriptors eingetragen wird.
<firstSpiritScope>	Angabe des FirstSpirit-Gültigkeitsbereichs (siehe dazu auch Kap. 2.5.1.2, Seite 13 im Entwicklerhandbuch „MDEVDE_FirstSpirit_ModulDeveloperDoc.pdf“).

6.3. Erweiterung Eingabekomponente im SiteManager

Der Monday Formulareditor basiert auf einem allgemeinen Framework, mit dem beliebige Swing-basierte XML-Editoren entwickelt werden können. Innerhalb des Frameworks wird ein XML-Dokument durch eine hierarchische Struktur von *NodeModel* Objekten repräsentiert. Jedes *NodeModel*-Objekt entspricht einem oder mehreren Elementen der XML-Struktur. Das Framework stellt Methoden zur Anzeige, Auswahl und Bearbeitung der einzelnen *NodeModel*-Objekte zur Verfügung.

Jedem *NodeModel* ist ein *NodeType*-Objekt zugeordnet. Dieses enthält Informationen darüber, wie das zugehörige *NodeModel* initialisiert, dargestellt, bearbeitet und validiert werden soll. Zudem können Sie auf jedem *NodeType* -Objekt festlegen, welche anderen XML-Elemente unterhalb des Models angeordnet werden dürfen.

Die benötigten *NodeType*-Objekte werden durch die *EditorSetup*-Klasse erzeugt. Nutzen Sie diese somit als zentralen Ansatzpunkt für die Erweiterung des Formulareditors.

Im Formular-Framework kapseln Actions die eigentliche Businesslogik der Formulardatenverarbeitung. Im folgenden Teil finden Sie am Beispiel einer Action gezeigt, wie Sie den Formulareditor um zusätzliche Elemente erweitern können.

6.3.1. Entwicklung eines NodeEditorPane

Für jeden Elementtyp benötigt das System einen angepassten Editor, der von der Klasse *NodeEditorPane* abgeleitet werden muss. Der Node-Editor dient dazu, ein NodeModel des entsprechenden Typs zu bearbeiten. Das nachfolgende Code-Beispiel zeigt Ihnen die Implementierung eines NodeEditorPanes.

```
public class CustomActionEditorPane extends TabbedBaseActionEditorPane {

  public static final String PROP_CUSTOM = "customProperty";
  protected JPanel propertiesPanel;
  protected JTextField customPropertyField;
  protected ActionModel model;  // extends NodeModel

  PropertiesBundle custombundle =
      PropertiesBundle.getInstance("com/custom/forms/formeditor",
            CustomActionEditorPane.class.getClassLoader());

  protected JTextField getCustomPropertyField() {
    if (customPropertyField == null) {
      customPropertyField = new JTextField();
      customPropertyField.addFocusListener(focusListener);
      customPropertyField.addKeyListener(new KeyAdapter() {
        public void keyReleased(final KeyEvent e) {
          model.setProperty(PROP_CUSTOM, customPropertyField.getText());
        });
    }
    return customPropertyField;
  }

  protected JPanel getPropertiesPanel() {
    if (propertiesPanel != null) {
        return propertiesPanel;
    }

    propertiesPanel = new JPanel();

    GridBagDesigner layout =
        new GridBagDesigner(custombundle, propertiesPanel);

    layout.row().labelTop("propLabel").expand()
        .add(getCustomPropertyField());

    layout.fill();

    return propertiesPanel;
  }

  public void setEditable(final boolean editable) {
    if (customPropertyField!= null) {
      customPropertyField.setEditable(editable);
    }
  }

  public void setModel(NodeModel model) {
```

```
    this.model = (ActionModel) model;
    update();
  }

  public void update() {
    getCustomPropertyField().setText(
        (String) model.getProperty(PROP_CUSTOM));
  }
}
```

Das oben dargestellte Beispiel zeigt Ihnen einen Node-Editor, der ein Textfeld beinhaltet, in dem der Wert des Properties *customProperty* eingegeben werden kann. Das Textfeld und das dazugehörige Label sind in einem GridBagLayout auf dem zugrunde liegenden JPanel angeordnet. Zum Zweck der Internationalisierung des Editors wird der Text des Labels aus einem sprachabhängigen Ressource-Bundle gelesen.

Texteingaben werden durch einen anonymen KeyListener an das NodeModel übertragen.

6.3.2. Erweiterung der EditorSetup-Klasse

Innerhalb der *init*-Methode der *EditorSetup*-Klasse werden die *NodeType*-Objekte des Formulareditors erzeugt. Indem Sie diese Methode überschreiben, können Sie ein zusätzliches *NodeType*-Objekt für die neue *CustomAction* erzeugen und registrieren:

```
@Override
public void init() {

  super.init();

  PropertiesBundle custombundle =
      PropertiesBundle.getInstance("com/custom/forms/formeditor",
            this.getClass().getClassLoader());

  // neuen Typ erzeugen
  NodeType customActiontype = createCustomActionType(custombundle);

  // neuen Typ registrieren
  registerType(customActiontype);

  // als mögliches Unterelement des Form-Elements registrieren
  NodeType formNodeType = getNodeType(FORM.TYPE.FORM);
  formNodeType.addAllowedChildType(customActiontype);
}
```

Die *NodeType*-Objekte der einzelnen Formularelemente werden typischerweise in separaten Factory-Methoden erzeugt. Wie Sie diese implementieren, zeigt Ihnen das folgende Beispiel:

```
protected NodeType createCustomActionType(final EditorBundle bundle)  {
```

```
  final String CUSTOM_ACTION = "customAction";

  NodeType action = new NodeType(CUSTOM_ACTION, ActionModel.class,
      CustomActionEditorPane.class);

  action.setTitle(bundle.getString(CUSTOM_ACTION + "Label"));

  action.setIcon(bundle.getImageIcon(CUSTOM_ACTION + "Icon"));

  action.setToolbarIcon(bundle.getImageIcon(CUSTOM_ACTION + "LargeIcon"));

  action.setValidator(new CustomActionValidator());

  action.setInitializer(new CustomActionInitializer());

  action.setButtonGroup(1);

  return action;
}
```

Im oben dargestellten Beispiel wird zunächst eine neue *NodeType*-Instanz mit dem Namen *customAction* erzeugt. Der Name dient in der weiteren Verarbeitung dazu, diesen Typ eindeutig zu identifizieren. So wird beispielsweise in der Webapplikation die zugehörige Action-Implementierung anhand dieses Namens ermittelt. Typ-Namen dürfen Sie daher nur einmal verwenden. Anschließend wird ein spezieller *NodeValidator* und ein *NodeInitializer* definiert.

Wenn Sie die Initialisierung oder Validierung eines bestehenden Formularelements ändern wollen, reicht es in der Regel aus, wenn Sie die dazugehörige Factory-Methode überschreiben, um einen anderen Validator oder Initializer zu setzen.

6.3.3. Erweiterung des GUI-Object-Models des Formulareditors

Die EditorSetup-Klasse wird durch das GUI-Object-Model (GOM) des Formulareditors instanziiert. Indem Sie die *createEditorSetup*-Methode der Klasse *com.formcentric.editor.gadgets.FormEditorSwingGadgetFactory* überschreiben, wird statt der bestehenden die neue EditorSetup-Klasse erzeugt.

```
public class CustomFormEditorSwingGadgetFactory extends
        FormEditorSwingGadgetFactory {
  @Override
  public FormEditorSetup createEditorSetup(GomFormEditor gomFormEditor,
        SpecialistsBroker broker, Language lang) {
    return new CustomFormEditorSetup(gomFormEditor, broker, lang));
  }
}
```

Konfigurieren Sie die geänderte Klasse *CustomGomFormEditor* im Modul-Deskriptor *prototype.module.xml* im Element *configuration* der Eingabekomponente.

```
<name>MONDAY_FORMEDITOR</name>
<description>Monday Webforms editor component</description>
<class>de.espirit.firstspirit.module.GadgetSpecification</class>
<configuration>
  <factory>com.custom.webforms.CustomFormEditorSwingGadgetFactory</factory>
  ...
<configuration>
```

6.4. Erweiterung der ContentCreator-Webapplikation

Bei der Formcentric ContentCreator-Integration handelt es sich um eine Single-Page-Anwendung, die auf dem JavaScript Framework *React* basiert. Die Benutzeroberfläche des Formulareditors wird dabei browser-seitig aus den vom Server übertragenen JSON-Daten erzeugt. Der Aufbau der Oberfläche wird deklarativ über verschiedene JavaScript-Konfigurationsdateien festgelegt. Dieser Ansatz ermöglicht es Ihnen, auf einfache Weise Änderungen und Erweiterungen an der Redaktionsoberfläche vorzunehmen.

Zentraler Ansatzpunkt für die Anpassung der Redaktionsoberfläche sind die JavaScript-Konfigurationsdateien, die im Entwicklungs-Workspace im Modul *formcentric-webedit-customizations* im Verzeichnis */src/main/webapp/WEB-INF/formcentric_editor/gadget/formeditor/config/editor* abgelegt sind. Alle nachfolgend beschriebenen Anpassungen erfolgen in den darin enthaltenen Dateien.

Die verfügbaren Formularelemente mit ihren Eigenschaften werden in Form von JSON-Objekten beschrieben. Die React-Anwendung erzeugt daraus die Redaktionsoberfläche. Das nachfolgende Beispiel zeigt einen Auszug der Konfiguration für das Formularelement *textArea*.

```
{
    icon: 'textarea',
    type: 'textArea',
    properties: {
        general: [
            {
                title: 'name',
                type: 'text',
                properties: {
                  required: true
                }
            },
            {
                title: 'label',
                type: 'text'
            },
            {
                title: 'hint',
                type: 'text'
            },
            {
                title: 'value',
```

```
                type: 'wysiwyg'
            },

            ...
        ]
    }
}
```

6.4.1. Neues Formularelement hinzufügen

Erweitern Sie den Formulareditor um ein neues Formularelement, in dem Sie die Konfiguration *fields_custom.js* erweitern. Möchten Sie den Editor beispielsweise um das Formularelement *termsCheckbox* mit den Eigenschaften *name, text* und *link* erweitern, so fügen Sie dem JavaScript-Array in der Konfiguration *fields_custom.js* folgende Objektdefinition hinzu.

```
[
    {
        icon: 'termscheckbox',
        type: 'termsCheckbox',
        properties: {
            general: [
                {
                    title: 'name',
                    type: 'text',
                    properties: {
                        required: true
                    }
                },
                {
                    title: 'text',
                    type: 'wysiwyg',
                    properties: {
                        required: true
                    }
                },
                {
                    title: 'link',
                    type: 'reference',
                    properties: {
                        refType: 'pageref',
                        FS_refType: 'pageref'
                    }
                },
            ]
        },
        specialProperties: {
            condition: {
                conditionable: false,
                operators: {}
            }
        }
    }
]
```

Hinweis: Das äußere JavaScript-Array ist bereits vorhanden und muss lediglich um das Konfigurationsobjekt erweitert werden.

Die nachfolgende Tabelle beschreibt die möglichen Attribute, die eine Felddefinition auf erster Ebene besitzen kann.

Attribut	**Beschreibung**
icon	Typ: *String* Name des zu ladenden Icons. Der angegebene Name muss mit dem Dateinamen des Icons ohne Dateiendung übereinstimmen.
type	Typ: *String* Name des Formularelements

properties	Typ: *Object*
	
	Definiert die Eigenschaften eines Feldes, die auf der rechten Seite unter *Feldeigenschaften* im ContentCreator bearbeitet werden können. Die Objekteigenschaften von *properties* stellen dabei einzelne Editor-Tabs dar. Der folgende JSON-Ausschnitt konfiguriert zwei Tabs mit den Namen *general* und *special*, mit insgesamt drei Eigenschaften: *name, label, hint*.
	
	`properties: {`
	`  general: [`
	`  {`
	`  title: 'name',`
	`  type: 'text'`
	`  },`
	`  {`
	`  title: 'label',`
	`  type: 'text'`
	`  }`
	`  ],`
	`  special: [`
	`  {`
	`  title: 'hint',`
	`  type: 'text'`
	`  }`
	`  ]`
	`}`
	
	Damit das Feld bei der weiteren Verarbeitung eindeutig identifiziert werden kann, wird die Eigenschaft *titel* mit dem Wert *name* im Array *general* benötigt.
	
	Eine Liste aller möglichen Eigenschaftstypen finden sie unter Abschnitt 6.4.5, „Eingabeelemente für Elementeigenschaften“

specialProperties	Typ: *Object*

Attribut	Beschreibung
	Im Attribut *specialProperties* konfigurieren Sie Eigenschaften, die vom Editor für interne Funktionen ausgewertet werden. Der folgende JSON-Ausschnitt definiert die Verwendungsweise des Feldes innerhalb einer Condition. `specialProperties: { condition: { conditionable: true, operators: { startswith: { values: [], freeField: true, useChildren: false }, endswith: { values: [], freeField: true, useChildren: false }, contains: { values: [], freeField: true, useChildren: false } } } }` Mit der Angabe von *conditionable: true* legen Sie fest, dass das Feld in einer Bedingung angewählt werden kann. Die in der Bedingung für diesen Formularelementtyp auswählbaren Operatoren legen Sie im Object *operators* fest. Eine Operator-Definition folgt dabei immer dem Schema *<operator-name>: {values: [], freeField: true, useChildren: false}*. Der Name des Operators wird zudem auch als Übersetzungs-ID für die Internationalisierung der Benutzeroberfläche verwendet (siehe Abschnitt 6.4.7, „Internationalisierung der Benutzeroberfläche“). Im Attribut *values* haben Sie die Möglichkeit, ein String-Array mit Werten anzugeben, die vom Redakteur bei der Definition einer Condition ausgewählt werden können. Durch Angabe des Attributs *freeField: true* ermöglichen Sie es Redakteuren, benutzerdefinierte Werte einzugeben. Diese Option wird zum Beispiel für Vergleichsoperatoren benötigt, bei denen Redakteure eigene Vergleichswerte angeben müssen.

Attribut	Beschreibung
	Wenn es sich bei dem neuen Feldtyp um einen Listentyp mit vorgegebenen Optionen handelt, haben Sie im Attribut *useChildren* die Möglichkeit festzulegen, dass die Optionen als Wert der Bedingung ausgewählt werden können.

6.4.2. Neuen Validator hinzufügen

Um einem Eingabefeld einen neuen Validator hinzuzufügen, erweitern Sie die *format*-Eigenschaft des zugehörigen Eingabeelements.

Das nachfolgende Beispiel zeigt die Konfiguration des E-Mail-Validators für das einzeilige Textfeld (*inputField*).

```
{
    title: 'format',
    type: 'dropdown_format',
    properties: {
        options: {
            email: {
                enabled: true,
                fields: {
                    errormessage: {
                        title: 'errormessage',
                        type: 'text'
                    }
                }
            }
        }
    }
}
```

Der angegebene Attributname (im Beispiel *email*) muss dem externen Namen des Validators entsprechen. Der Name wird auch für die Internationalisierung der Oberfläche verwendet. In der Übersetzungsdatei wird mit der Übersetzungs-ID `<validator-name>Validator` nach einer Beschriftung für den Validator gesucht.

Sie können unter *fields* die gewünschten Felder des Validators definieren. Die verfügbaren Feldtypen sind in der Tabelle unter Abschnitt 6.4.5, „Eingabeelemente für Elementeigenschaften“ aufgeführt.

6.4.3. Neue Aktion hinzufügen

Erweitern Sie den Formulareditor um eine neue Aktion, in dem Sie die Konfiguration *actions_custom.js* erweitern.

Möchten Sie den Editor beispielsweise um die Aktion *simpleMailAction* mit den Eigenschaften *to, subject, body* und *note* erweitern, so fügen Sie dem JSON-Array in der Konfiguration *actions_custom.js* folgende Objektdefinition hinzu.

```
[
    {
```

```
        icon: 'simplemailaction',
        type: 'simpleMailAction',
        properties: {
            general: [
                {
                    title: 'to',
                    type: 'text',
                    properties: {
                        required: true
                    }
                },

                {
                    title: 'subject',
                    type: 'text',
                    properties: {
                        required: true
                    }
                },
                {
                    title: 'body',
                    type: 'wysiwyg'
                },
                {
                    title: 'note',
                    type: 'wysiwyg'
                },
            ]
        },
        specialProperties: {
            condition: {
                conditionable: false,
                operators: {}
            }
        }
    }
]
```

Die nachfolgende Tabelle beschreibt die möglichen Attribute, die eine Aktionsdefinition auf erster Ebene besitzen kann.

Attribut	**Beschreibung**
icon	Typ: *String* Name des zu ladenden Icons. Der angegebene Name muss mit dem Dateinamen des Icons ohne Dateiendung übereinstimmen.
type	Typ: *String* Name des Feldtyps
properties	Typ: Object Beschreibt die Eigenschaften einer Aktion die auf der rechten Seite unter *Eigenschaften* im ContentCreator bearbeitet

Attribut	Beschreibung
	werden können. Die Objekt-Eigenschaften von *properties* stellen dabei einzelne Tabs da. Der folgende JSON-Ausschnitt erzeugt zwei Tabs mit den Namen *general* und *special*, mit insgesamt drei Eigenschaften *to, subject* und *hint*. `properties: { general: [ { title: 'to', type: 'text', properties: { required: true } }, { title: 'subject', type: 'text', properties: { required: true } } ], special: [ { title: 'hint', type: 'text' } ] }` Eine Liste aller möglichen Eigenschaftstypen finden sie unter Abschnitt 6.4.5, „Eingabeelemente für Elementeigenschaften“
specialProperties	Typ: *Object* Im Attribut *specialProperties* konfigurieren Sie Eigenschaften, die vom Editor für interne Funktionen ausgewertet werden. `specialProperties: { maxCount: 1 }` Mit `maxCount: <anzahl>` legen Sie fest, wie oft die Aktion innerhalb eines Formulars verwendet werden kann.

6.4.4. Neue Elementeigenschaften hinzufügen

Die Definition von Elementeigenschaften erfolgt unterhalb des Attributs *properties* der übergeordneten Formularelementdefinition (siehe Abschnitt 6.4.1, „Neues Formularelement hinzufügen“). Durch Angabe eines JSON-Objekts mit folgender Struktur fügen Sie dem Formularelement (Formularfeld, Aktion oder Validator) eine neue Eigenschaft hinzu.

```
{
    title: '<attribute-name>',
    type: '<field-type>',
    value: 'DefaultValue',
    properties: {
      required: true
    }
}
```

Die nachfolgende Tabelle beschreibt die Attribute des Konfigurationsobjekts.

Attribut	**Beschreibung**
title	Name, unter dem die Feldeigenschaft in der Formulardefinition gespeichert wird.
type	Typ der Eigenschaft. Die möglichen Typen werden in der folgenden Liste erklärt.
value	Optionale Angabe eines Vorbelegungswertes
properties	Weitere typspezifische Konfigurationsmöglichkeiten
properties.required	Leget fest, ob es sich bei der Eigenschaft um ein Pflichtfeld handelt.

6.4.5. Eingabeelemente für Elementeigenschaften

In der folgenden Tabelle finden Sie die Konfigurationsobjekte für die Eingabeelemente der verfügbaren Elementeigenschaften beschrieben. Diese können Sie bei der Definition der verschiedenen Formularelementeigenschaften verwenden. Bitte beachten Sie, dass einige Typen nicht für alle Formularelemente verwendet werden können.

Typ	**Beschreibung**
text	Freitextfeld Verwendung: alle Formularelemente `{ title: 'label', type: 'text' }`
number	Zahlenfeld das ausschließlich numerische Eingaben zulässt. Verwendung: alle Formularelemente `{ title: 'maxlength', type: 'number', properties: { min: 0, max: null } }`

Typ	Beschreibung
	`}` Unterstützt auch die wissenschaftliche Schreibweise (z. B. *10e6*). Durch die Angaben *properties.min* und *properties.max* können Sie Grenzen definieren. Mit *properties.min: null* heben Sie eine vorhandene Begrenzung auf.

date	Element für die Auswahl eines Datums.
	
	Verwendung: alle Formularelemente
	
	`{`
	`  title: 'from',`
	`  type: 'date'`
	`}`
	
	Mit *value* können sie eine Vorbelegung definieren. Folgt dem Standard JavaScript-Datumsformat.

checkbox	Checkbox
	
	Verwendung: alle Formularelemente
	
	`{`
	`  title: 'requiredField',`
	`  type: 'checkbox'`
	`}`

dropdown	Liste mit festen Einträgen, aus der der Redakteur einen Eintrag auswählen kann.
	
	Verwendung: alle Formularelemente
	
	`{`
	`  title: 'pattern',`
	`  type: 'dropdown',`
	`  properties: {`
	`  options: ['dd.MM.yyyy', 'yyyy-MM-dd']`
	`  }`
	`}`
	
	Unter `properties.options` können Sie die Auswahlmöglichkeiten als String-Array angeben.
	
	`properties: {`
	`  options: [`
	`  {text: 'Value 1', value: 'one'},`
	`  {text: 'Example Two', value: two}`
	`  ]`
	`}`

Typ	Beschreibung
	Optionen können auch als Objekte mit *value* (Wert) und *text* (Anzeigename) definiert werden.
dropdown_format	Auswahlliste für Feldvalidatoren Verwendung: Eingabeelemente (*inputField, passwordField,* etc.) Bei der Auswahl werden die Eigenschaften des ausgewählten Validators unterhalb der Auswahlliste eingeblendet. Das folgende Beispiel zeigt die Definition des E-Mail-Validators. `{ title: 'format', type: 'dropdown_format', properties: { options: { email: { enabled: true, fields: { errormessage: { title: 'errormessage', type: 'text' } } } } } }` Unter `properties.options` können Sie die auswählbaren Validatoren festlegen. Unter `properties.options["<validator-name>"].fields` können Sie die Validator-Eigenschaften pflegen. Durch die `properties.options["<validator-name>"].enabled=true` oder `properties.options["<validator-name>"].enabled=false` aktivieren bzw. deaktivieren Sie den Validator, so dass er nicht mehr auswählbar ist.
syntax	Mehrzeiliges JavaScript-Eingabefeld mit Syntax-Highlighting Verwendung: alle Formularelemente `{ title: 'script', type: 'syntax', value: 'function calculate() {};' }`

Typ	Beschreibung
condition	Eingabeelement für die Bearbeitung von Bedingungen. Verwendung: *condition* `{ title: 'conditionContent', type: 'condition', properties: { conditional_fields: [], condition_conjunction: 'true', condition: [] } }`
wysiwyg	Mehrzeiliges Texteingabefeld, das die Angabe von Formatierungen ermöglicht. Verwendung: alle Formularelemente `{ title: 'value', type: 'wysiwyg' }`
element	Ermöglicht die Auswahl von anderen Elementen desselben Formulars. Verwendung: alle Formularelemente `{ title: 'elements', type: 'element' }`
dataSource	Eingabeelement für die variable Parameterliste einer Datenquelle Verwendung: inputField, comboBox, radioGroup, checkboxGroup, hiddenField `{ title: 'datasource', type: 'dataSource', properties: { datasource_params: [] } }`
reference	Element für die Auswahl einer FirstSpirit-Referenz (öffnet den Auswahldialog von FirstSpirit). Verwendung: alle Formularelemente

Typ	Beschreibung

	<pre>{
	    title: 'pictureUrl',
	    type: 'reference',
	    properties: {
	        FS_refType: 'picture'
	    }
	}</pre>
	Im Property *properties.FS_refType* haben Sie die Möglichkeit, die Auswahl auf einen bestimmten Inhaltstyp (Page, Picture, Folder etc.) einzuschränken. Folgende Angaben sind an dieser Stelle möglich: *pageref, picture, file*

multi_dropdown	Liste mit festen Einträgen, aus der der Redakteur mehrere Einträge auswählen kann.
	
	Verwendung: alle Formularelemente
	
	<pre>{
	    title: 'numberType',
	    type: 'multi_dropdown',
	    properties: {
	        configuration_name: 'phoneNumberTypes'
	    }
	}</pre>
	Im Attribut *properties.options* können Sie die Auswahlmöglichkeiten (als *Strings*) definieren
	
	Alternativ können die Werte auch aus der Absatzvorlage bezogen werden. Geben Sie in diesem Fall im Attribut *configuration_name* den Namen des entsprechenden GOM-Elements an.
	
	Der Parameter *configuration_name* legt fest, unter welchem Namen der Wert in der Formulardefinition gespeichert wird.

regEx_dropdown	Auswahlliste für reguläre Ausdrücke
	
	Verwendung: regex
	
	<pre>{
	    title: 'mailPattern',
	    type: 'regEx_dropdown',
	    properties: {
	        options: [
	            {
	                text: '^[+]{0,1}[0-9\\s-/]*$',
	                label: 'phone'
	            },
	            {
	                text: '^[a-zA-ZÁ-ÿöäüÖÄÜß\\s-]*$',</pre>

Typ	Beschreibung

	<pre>label: 'characters'
	}
	]
	}
	}</pre>

mediastore_mapping	Element für die Zuordnung von Data-Upload-Feldern zu Verzeichnissen im FirstSpirit Media-Store.
	Verwendung: *mediaStoreAction*
	<pre>{
	title: 'mediastore_mapping',
	type: 'mediastore_mapping',
	value: '[]'
	}</pre>

field_mapping	Element für die Auswahl einer PDF-Vorlage (öffnet einen Auswahldialog von FirstSpirit).
	Verwendung: *pdfAction*
	Anschließend kann den Formularfeldern ein PDF-Feld aus der Vorlage zugeordnet werden. Wenn es sich bei den Feldern um Auswahllisten handelt, können deren Optionen ausgewählt und einander zugeordnet werden.
	Verwendung: pdfAction
	<pre>{
	title: 'field_mapping',
	type: 'field_mapping',
	value: '[]'
	}</pre>

datasource_mapping	Mit diesem Eingabeelement können Formularfelder den Spalten einer FirstSpirit-Datenquelle zugeordnet werden.
	Verwendung: *dataSourceAction*
	<pre>{
	title: 'datasource_mapping',
	type: 'datasource_mapping',
	value: '[]'
	}</pre>

custom_mapping	Mit diesem Eingabeelement können eigene Zuordnungstabellen gestaltet werden.
	<pre>{
	title: 'custom_table',
	type: 'custom_mapping',
	properties: {</pre>

Typ	Beschreibung
	`mapping: [{ type: 'dropdown', name: 'field', placeholder: 'custom_table.field', selectableFieldTypes: ['inputField', 'radioGroup'] }, { type: 'dropdown', name: 'option', placeholder: 'custom_table.option', connectedField: 'field' }, { type: 'text', name: 'otherValue', placeholder: 'custom_table.otherValue' }, { type: 'dropdown', name: 'attribut', loadRemoteData: 'FS_ServiceField', loadRemoteDataOptions: [{ name: 'connectedField', key: 'task', value: 'someDropdown' }, { name: 'connectedMapField_type', key: 'type', value: 'field' }, { name: 'connectedMapField', key: 'value', value: 'field' },], placeholder: 'custom_table.attribut', }, { type: 'dropdown', name: 'attributoption', placeholder: 'custom_table.attributoption', loadRemoteData: 'FS_ServiceField', connectedField: 'attribut', },], }, },`

Typ	Beschreibung
	Im Attribut *properties.mapping* können Sie die Spalten definieren. Mit dem Key *type*, entscheiden Sie, ob es sich um eine Auswahlliste (dropdown) oder um ein Eingabefeld (text) handelt. Der Key *name*, setzt den Key für den Export, der jeweiligen Felder in einer Zeile. Mit dem Key *placeholder*, können Sie den Platzhalter, für das Feld definieren. Mit dem Key *loadRemoteData*, können Sie wie bei einem Dropdown-Feld, entscheiden, ob die Optionen von einem FirstSpirit Service kommen sollen. Falls die Optionen aus einem FirstSpirit Service kommen, können Sie mit dem Key *loadRemoteDataOptions* zusätzliche Attribute mitgeben, wie zum Beispiel Werte von anderen Feldern auf dem Element oder aus dem Mapping selbst. Das folgende Beispiel erstellt dieses Object, um es beim *loadRemoteData* Aufruf mitzugeben. `{task: <ValueOfFieldsomeDropdown>, type: <TypeVonFormElementAusgewähltInField>, value: '<ValueVonMappingDropdownField>'}` Falls Sie in einem Dropdown vorhandene Formularfelder auswählen möchten, können Sie diese über den Key *selectableFieldTypes* angeben. Möchten Sie auf verschachtelte Optionen eines Formularfelds oder von den Optionen aus auf *loadRemoteData* zugreifen, können Sie mit *connectedField* und der Angabe des Namens eines Mapping-Feldes auf diese zugreifen und zur Auswahl bereitstellen.

6.4.6. Bestehende Formularelemente anpassen

Um ein bestehendes Formularelement anzupassen, kopieren Sie dessen vollständige Elementdefinition aus der zugehörigen Standardkonfiguration (*fields_default.js* bzw. *actions_default.js*) in die entsprechende Konfigurationsdatei (*fields_custom.js* bzw. *actions_custom.js*).

Ändern oder ergänzen Sie anschließend wie oben beschrieben die Elementeigenschaften entsprechend Ihrer Anforderungen.

Hinweis: Änderungen an den Standardkonfigurationen im Entwicklungs-Workspace haben keine Auswirkungen auf den Formulareditor.

6.4.7. Internationalisierung der Benutzeroberfläche

Zur Internationalisierung der Benutzeroberfläche werden die sprachabhängigen Beschriftungen aus zentralen Sprachdateien gelesen. Formcentric unterstützt standardmäßig die Sprachen Deutsch und Englisch.

Um die Beschriftungen bestehender oder neuer Formularelemente anzupassen bzw. hinzuzufügen, erweitern bzw. ändern Sie die Sprachdateien *formeditor_de.json* und *formeditor_en.json*, die Sie im Entwicklungs-Workspace finden.

Jede Beschriftung ist mit einer eindeutigen Übersetzungs-ID in den Sprachdateien gespeichert. Die Übersetzungs-IDs der Elementeigenschaften setzen sich dabei typischerweise aus dem internen Elementnamen und dem Namen der jeweiligen Eigenschaft zusammen. Für die Eigenschaft *placeholder* des Passwortfelds sieht der Eintrag wie folgt aus:

```
"passwordField.placeholder": "Platzhalter"
```

Die Beschriftung einer neuen Elementeigenschaft können Sie hinzufügen, in dem Sie in jeder Sprachdatei einen entsprechenden Eintrag hinzufügen.

6.5. Erweiterung der Spring-MVC-Webapplikation

6.5.1. Spring-Konfigurationen

Bei dem WebApp-Modul handelt es sich um eine Spring-basierte Webapplikation. Die meisten Funktionen sind in speziellen Beans gekapselt, die durch den Dependency-Injection-Mechanismus von Spring instanziiert und initialisiert werden. Sie können die Anwendung um neue oder geänderte Funktionalitäten ergänzen, indem Sie einzelne Beans austauschen. Dabei reicht es meist aus, wenn Sie einzelne Methoden der bestehenden Beans überschreiben.

Die Spring-MVC-Webapplikation beinhaltet standardmäßig die nachfolgend beschriebenen Konfigurationen, die im Verzeichnis */WEB-INF/spring* der Webapplikation abgelegt sind.

formcentric-application.xml

Hier werden die verschiedenen Spring-XML-Dateien aggregiert und der *PropertySourcesPlaceholderConfigurer* konfiguriert. Sie bietet einen schnellen Überblick über alle relevanten Spring-Dateien, die für die Webapplikation benötigt werden.

formcentric-actions.xml

Konfiguriert die benötigten Actions. Tragen Sie die hier aufgeführten Action-Beans zusätzlich in das Action-Mapping (siehe unten) ein. Die benötigten Properties werden aus mehreren Property-Dateien ausgelesen und entsprechend gesetzt. Wenn Sie Werte ändern möchten, müssen Sie diese Dateien entsprechend anpassen.

```
<bean name="mailAction" class="com.formcentric.actions.mail.MailAction">
```

```
  <property name="mailer" ref="mailer" />
  <property name="successView" value="success" />

  <!-- Mapping von Formatbezeichnern auf MailBodyRenderer. -->
  <property name="bodyRendererMapping">
    <map>
      <entry key="html" value-ref="htmlBodyRenderer"/>
      <entry key="text" value-ref="textBodyRenderer"/>
      <entry key="freetext" value-ref="freeTextBodyRenderer"/>
      <entry key="freehtml" value-ref="freeHtmlBodyRenderer"/>/>
    </map>
  </property>
</bean>
...
```

formcentric-captcha.xml

Für die Erzeugung der Captchas wird das Open Source Framework JCaptcha verwendet. Bei dieser Konfiguration handelt es sich um eine JCaptcha Standardkonfiguration, mit der sich das Aussehen und Verhalten der Captchas beeinflussen lässt. Eine ausführliche Beschreibung der Konfigurationsmöglichkeiten finden Sie auf der Projekt-Homepage https://jcaptcha.atlassian.net/wiki/display/general/Home.

formcentric-services.xml

Konfiguriert die REST-Services. Tragen Sie die hier aufgeführten *RestService*-Beans zusätzlich in das Service-Mapping des REST-Controllers ein (siehe „formcentric-controllers.xml“).

```
<bean id="deCountriesRestService"
    class="com.formcentric.rest.CountriesRestService">
  <property name="lang" value="de"/>
</bean>

<bean id="enCountriesRestService"
    class="com.formcentric.rest.CountriesRestService">
  <property name="lang" value="en"/>
</bean>
```

formcentric-controllers.xml

Konfiguriert den Formular-Controller, den REST-Controller sowie die *FormCommandBeanFactory*, mit der das *FormCommandBean* erzeugt wird. Das *FormCommandBean* ruft die konfigurierten Initializer, Validatoren und Actions auf und erzeugt das Form-Model.

Neue Actions, Validatoren und Formularelemente konfigurieren Sie auf der CommandBean-Factory:

```
<bean id="defaultFormCommandBeanFactory"
  class="com.formcentric.logicbeans.DefaultFormCommandBeanFactory">

  <property name="formElementClassMapping">
```

```
    <map>
      <entry key="inputField" value="java.lang.String"/>
      <entry key="passwordField" value="java.lang.String"/>
      <entry key="hiddenField" value="java.lang.String"/>
      <entry key="textArea" value="java.lang.String"/>
      <entry key="radioGroup" value="java.lang.String[]"/>
      <entry key="comboBox" value="java.lang.String[]"/>
      <entry key="checkBoxGroup" value="java.lang.String[]"/>
      <entry key="fileUpload" value="com.formcentric.model.FileHolder"/>
      <entry key="captcha" value="java.lang.String"/>
    </map>
  </property>

  <property name="validatorMapping">
    <map>
      <entry key="notempty" value-ref="notemptyValidator"/>
      <entry key="jcaptcha" value-ref="captchaValidator"/>
      <entry key="email" value-ref="emailValidator"/>
      <entry key="date" value-ref="dateValidator"/>
      <entry key="number" value-ref="numberValidator"/>
      <entry key="javascript" value-ref="javascriptValidator"/>
      <entry key="regex" value-ref="regexValidator"/>
      <entry key="length" value-ref="lengthValidator"/>
      <entry key="zipcode" value-ref="zipCodeValidator"/>
      <entry key="phone" value-ref="phoneValidator"/>
      <entry key="password" value-ref="passwordValidator"/>
      <entry key="bic" value-ref="bicValidator"/>
      <entry key="iban" value-ref="ibanValidator"/>
      <entry key="file" value-ref="fileValidator"/>
      <entry key="equal" value-ref="equalValidator"/>
    </map>
  </property>

  <property name="actionMapping">
    <map>
      <entry key="mailAction" value-ref="mailAction"/>
      <entry key="compositeAction" value-ref="compositeAction"/>
      <entry key="dataSourceAction" value-ref="dataSourceAction"/>
      <entry key="mediaStoreAction" value-ref="mediaStoreAction"/>
      <entry key="pdfAction" value-ref="pdfAction"/>
      <entry key="datastoreAction" value-ref="datastoreAction"/>
      <entry key="redirectAction" value-ref="redirectAction"/>
    </map>
  </property>
</bean>

<bean id="formController"
  class="com.formcentric.controllers.FormController">

  <property name="prefix" value="/form" />
  <!-- form controller specific part -->
  <property name="formCommandBeanFactory"
            ref="defaultFormCommandBeanFactory" />
  <property name="bindOnNewForm" value="true" />
  <property name="commandNamePrefix" value="command" />
  <property name="restoreInitialPage" value="true" />
```

```
  <property name="trimSpaces" value="false" />
  <property name="supportedMethods" value="POST" />
</bean>

<bean id="restController"
  class="com.formcentric.controllers.RestController">

  <property name="prefix" value="/rest" />
  <property name="commandNamePrefix" value="command" />

  <!-- Service-Mapping -->
  <property name="restServiceMapping">
    <map>
      <entry key="Länder" value-ref="deCountriesRestService"/>
      <entry key="Countries" value-ref="enCountriesRestService"/>
    </map>
  </property>
</bean>
```

formcentric-resourcebundle.xml

Fügt dem bestehenden *messageSource*-Bean das Formcentric Ressource-Bundle hinzu.

```
<bean id="messageSource"
  class="org.springframework.context.support.ResourceBundleMessageSource">
  <property name="basenames">
    <list>
      <value>com.formcentric.resourcebundle.messages</value>
    </list>
  </property>
</bean>
```

Einzelne Formcentric Messages können über ein Projekt eigenes Ressource-Bundle überschrieben werden. Dabei empfiehlt es sich, dieses in der Liste der *basenames* in einer Position vor dem Formcentric Ressource-Bundle zu platzieren. Weitere Informationen darüber finden Sie in der Spring JavaDoc-Konfiguration nachlesen.

formcentric-views.xml

Erzeugt zwei neue Spring-*ViewResolver* für die Ermittlung der JSP-Views und der Bean-Views.

```
<bean id="jspViewResolver"
  class="com.formcentric.view.JspViewResolver">

  <property name="prefix" value="/WEB-INF/templates/" />
  <property name="viewNames" value="webform,default,optin,exit,success,
      pdf,error,methodNotSupported,uploadSizeExceeded,sessionExpired,
      invalidLicense"/>
</bean>
```

```
<bean id="jsonView" class="com.formcentric.view.JsonView"/>

<bean id="fileDownloadView"
      class="com.formcentric.view.FileDownloadView"/>

<bean id="fileInfoView" class="com.formcentric.view.FileInfoView"/>

<bean id="thumbnailView" class="com.formcentric.view.ThumbnailView">
  <property name="thumbnailWidth" value="90"/>
  <property name="thumbnailHeight" value="90"/>
  <property name="crop" value="true"/>
</bean>

<bean id="beanViewResolver"
      class="com.formcentric.view.BeanViewResolver">
  <property name="order" value="1"/>
  <property name="views">
    <map>
      <entry key="json" value-ref="jsonView"/>
      <entry key="file" value-ref="fileDownloadView"/>
      <entry key="fileInfo" value-ref="fileInfoView"/>
      <entry key="thumbnail" value-ref="thumbnailView"/>
      <entry key="captcha" value-ref="captchaView"/>
    </map>
  </property>
</bean>
...
```

formcentric-mail.xml

In dieser Konfiguration legen Sie die Verbindungseinstellungen zum Mail-Server fest. Das konfigurierte *mailer*-Bean wird von der Mail-Action verwendet. Wenn Sie die grundlegenden Properties der *mailer*-Bean anpassen möchten, finden Sie alle benötigten Werte in der *formcentric-mail.properties*-Datei.

```
<bean id="mailer" class="com.formcentric.mail.SpringMailer">
    <constructor-arg ref="mailProperties"/>
</bean>
```

Die Konfigurationseinstellungen des Mailer-Beans werden mit Hilfe eines *PropertiesFactoryBean* aus einer separaten Property-Datei gelesen, die von der Server- und Projektkonfiguration erzeugt wird (siehe Abschnitt 4.4.2, „Konfiguration“).

formcentric-network.xml

Konfiguriert den HttpClient, mit dessen Hilfe benötigte Web-Ressourcen wie beispielsweise die PDF-Vorlagendokumente heruntergeladen werden. Umgebungsabhängige Konfigurationsparameter finden Sie in der Datei *formcentric-network.properties*.

```
<!-- Proxy configuration -->
<bean id="proxy" class="com.formcentric.http.ProxyConfig">
    <property name="host" value="${proxy.host}"/>
    <property name="port" value="${proxy.port}"/>
    ...
</bean>

<!-- HttpClient configuration -->
<bean id="httpClient" class="com.formcentric.http.HttpClientFacade">
    <!-- Browser identification string -->
    <property name="userAgent" value="Mozilla/5.0 Firefox/26.0"/>

    <!-- Proxy configuration -->
    <property name="proxyConfig" ref="proxy" />
    ...
</bean>
```

formcentric-connection.xml

In dieser Konfiguration legen Sie die Verbindungseinstellungen zum FirstSpirit-Server fest. Das konfigurierte *ConnectionFactory*-Bean wird von der FirstSpirit-Action verwendet. Die Factory kann über Properties aus der zugehörigen *formcentric-connection.properties*-Datei angepasst werden.

```
<bean id="firstSpiritConnection"
    class="com.formcentric.firstspirit.DisposableConnection" >
    <property name="protocol" value="${connection.transport.protocol}" />
    <property name="host" value="${connection.host}" />
    <property name="port" value="${connection.port}" />
    <property name="user" value="${connection.user}" />
    <property name="password" value="${connection.password}" />
    <property name="useHttps" value="${connection.https:false}"/>
</bean>
```

Die Konfigurationseinstellungen des ConnectionFactory-Beans werden mit Hilfe eines *PropertyPlaceholderConfigurer* aus einer separaten Property-Datei gelesen, die von der Server- und Projektkonfiguration erzeugt wird (siehe Abschnitt 4.4.2, „Konfiguration“).

formcentric-analytics.xml

In dieser Konfiguration legen Sie die Verbindungseinstellungen zum Formcentric Backend fest. Das konfigurierte *BackendApiClient*-Bean wird von der Analytics-Action, dem *BackendFormStateStore* und der *TrackingCommandBean* verwendet.

Für die Authentifizierung gegenüber dem Formcentric Backend wird ein *Access-Token* benötigt. Wenn dieses Token automatisch zur Laufzeit generiert werden soll, können Sie den *ClientSecretCredentialsAuthProvider* nutzen. Dieser benötigt das Client-Secret, dass in der Konfiguration des Backends vergeben wurde. Alternativ können Sie das Token manuell erzeugen und den *ApplicationAuthProvider*

verwenden. Nähere Informationen zur Generierung eines Access-Tokens finden Sie im Installationshandbuch des Formcentric Backend.

Sie können die zugehörigen Properties in der *formcentric-analytics.properties*-Datei individuell anpassen.

```
<!--
  METHOD 1: Use a pre-generated token.
-->
<bean id="analyticsAuthenticator"
    class="com.formcentric.backend.api.auth.ApplicationAuthProvider">
  <constructor-arg index="0" value="${analytics.apiAuthentication}" />
</bean>

<!--
  METHOD 2: Pass the full client credentials and request a token at runtime.
-->
<bean id="analyticsAuthenticator" class="com.formcentric.backend
    .api.auth.ClientSecretCredentialsAuthProvider">
  <constructor-arg index="0" value="${analytics.backendUrl}" />
  <constructor-arg index="1" value="${analytics.apiAuthentication}" />
</bean>

<bean id="analyticsApiBuilder"
    class="com.formcentric.backend.api.ApiClientBuilder">
  <constructor-arg index="0" value="${analytics.backendUrl}" />
  <constructor-arg index="1" ref="analyticsAuthenticator" />
</bean>

<bean id="fcBackendApiClient"
    factory-bean="analyticsApiBuilder" factory-method="backendApiClient" />
```

6.5.2. Verwendung ohne Formcentric Analytics

Die Spring-Konfigurationen *formcentric-actions.xml*, *formcentric-controllers.xml* und *formcentric-analytics.xml* beinhalten Komponenten, die nur bei der Verwendung von Formcentric Analytics genutzt werden können. Wenn Sie Formcentric ohne Analytics verwenden möchten, müssen Sie folgende Beans und Bean-Referenzen aus den aufgelisteten Spring-Konfigurationen entfernt werden:

formcentric-actions.xml: *datastoreAction*

formcentric-controllers.xml: *formStateStore (BackendFormStateStore)*, *defaultTrackingCommandBean*, *trackingController*

formcentric-analytics.xml: Sämtliche Beans in dieser Datei werden nicht benötigt.

6.5.3. Formcentric Lizenzdatei

Die Datei *formcentric-license.xml* konfiguriert den LicenseLoader von Formcentric. Über die zugehörige *formcentric-license.properties* kann der Pfad zur Lizenzdatei angegeben werden.

Beispiel (Linux / Unix): */path/to/formcentric-license*

Beispiel (Windows): *C:/path/to/formcentric-license*

Pfade, die nicht mit einem / beginnen, werden relativ zur Webapp aufgelöst.

6.5.4. Web-Security

Zur Abwehr von Cross Site Scripting (XSS) Angriffen und Cross Site Request Forgery Angriffen (XSRF) beinhaltet Formcentric einen Security-Servlet-Filter. Dieser entfernt unzulässige HTML-Tags, CSS und Skripte aus den übertragenen Formulardaten. Zusätzlich prüft der Filter, ob die Formulardaten einen gültigen XSRF-Token enthalten.

Für die Abwehr von XSRF-Angriffen kann jedem Formular ein zusätzlicher XSRF-Token als Hidden-Parameter hinzugefügt und zusammen mit den übrigen Formulardaten an die Webapplikation übertragen werden. Der Security-Filter prüft, ob der übertragene Token mit dem in der User-Session hinterlegten Token übereinstimmt. In diesem Fall wird der Request an die Webapplikation weitergeleitet. Andernfalls wird eine 401-Fehlermeldung an den aufrufenden Client zurückgegeben und der Aufruf im Log der Webapplikation im Log-Level *warn* mit den folgenden Informationen protokolliert:

- aufgerufene URL
- übertragene Formulardaten (POST-Parameter)
- IP-Adresse des aufrufenden Clients
- vollqualifizierter Name des aufrufenden Clients oder des letzten Proxies

Das nachfolgende Beispiel zeigt Ihnen, wie Sie den XSRF-Token in das Ausgabe-Template des Formulardokuments einfügen können:

```
<%@ taglib prefix="form"
    uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="fcs"
    uri="http://www.formcentric.com/web-security-1.0" %>
<%@ taglib prefix="c"
    uri="http://java.sun.com/jsp/jstl/core" %>

<c:url value="/servlet/form/${self.uid}?view=ajax" var="ajaxUrl"/>
<form:errors path="command${self.uid}" cssClass="error" />

<div id="ajaxreplace${self.uid}" class="ajax_box">

  <form:form name="${self.uid}" commandName="command${self.uid}"
             enctype="multipart/form-data">

  <!-- include XSRF token&#8211;->
  <fcs:xsrfToken/>
```

```
  ...
  </form:form>
</div>
```

Zusätzlich zum Formular-Template muss der XSRF-Token in alle URLs eingefügt werden, die auf einen Formcentric-Controller verweisen. Aktuell sind dies der *Form*-, *FileUpload*- und *RestController*.

Das folgende Beispiel zeigt Ihnen, wie Sie den XRSF-Token im Template *fileUpload.jsp* als URL-Parameter in die Upload-URL einfügen können.

```
<%@ taglib prefix="form"
    uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="fcs"
    uri="http://www.formcentric.com/web-security-1.0" %>
<%@ taglib prefix="c"
    uri="http://java.sun.com/jsp/jstl/core" %>

<!-- Upload-URL zusammenbauen und in Variable speichern -->
<c:url value="/servlet/upload" var="uploadUrl">
   <c:param name="_uid" value="${form.uid}"/>
   <c:param name="_lang" value="${form.lang}"/>
   <fcs:xsrfTokenParam method="POST"/>
</c:url>

<c:set var="id" value="${self.id}_${form.uid}"/>

<div class="mwf-upload"
     data-mwf-fileupload='{
        "url":"${uploadUrl}",
        "id":"${id}",
        "name":"${self.name}",
        "autoUpload": ${self.properties['auto_upload']},
        "labels": ${rowLabels},
        "previewMaxWidth": "90",
        "previewMaxHeight": "90"
     }'>

...
```

Die Upload-URL wird mithilfe des JSP-Tags *<c:url>* erzeugt. Verwenden Sie den *<mwf:xsrfTokenParam>*-Tag um der URL einen XSRF-Token-Parameter hinzuzufügen. Dieser kann wie der JSP-Tag *<c:param>* in Verbindung mit dem *<c:url>*-Tag verwendet werden (siehe „fcs:xsrfTokenParam“).

Den Security-Filter konfigurieren Sie im Deployment-Descriptor (web.xml) der Webapplikation. In den Mapping-Regeln müssen Sie alle Formcentric-Controller angeben, für die der Filter verwendet werden soll.

```
<filter>
  <filter-name>webSecurityFilter</filter-name>
  <filter-class>com.formcentric.security.
      SecurityServletFilter</filter-class>
  <init-param>
```

```
    <param-name>xsrfPrevention</param-name>
    <param-value>true</param-value>
  </init-param>
  <init-param>
    <param-name>xsrfMethods</param-name>
    <param-value>POST</param-value>
  </init-param>
  <init-param>
    <param-name>xsrfSessionBased</param-name>
    <param-value>true</param-value>
  </init-param>
  <init-param>
    <param-name>xsrfTokenName</param-name>
    <param-value>_mwfToken</param-value>
  </init-param>
  <init-param>
    <param-name>xssPrevention</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>webSecurityFilter</filter-name>
  <url-pattern>/servlet/form/*</url-pattern>
</filter-mapping>

<filter-mapping>
  <filter-name>webSecurityFilter</filter-name>
  <url-pattern>/servlet/rest</url-pattern>
</filter-mapping>

<filter-mapping>
  <filter-name>webSecurityFilter</filter-name>
  <url-pattern>/servlet/upload</url-pattern>
</filter-mapping>
```

Der Servlet-Filter verfügt über folgende Konfigurationseinstellungen.

Parameter	**Beschreibung**
xsrfPrevention	Aktiviert die XSRF-Absicherung (true, false).
xsrfMethods	Kommaseparierte Liste der HTTP-Methoden (GET, POST), die mit einem XSRF-Token abgesichert werden sollen.
xsrfSessionBased	Mit diesem Parameter legen Sie fest, ob der XSRF-Token bei jedem Seiten-Reload (true) , oder einmal pro User-Session (false) erneuert werden soll.
xsrfTokenName	Basisname des Request-Parameters, in dem der aktuelle XSRF-Token übertragen wird. Standardmäßig wird der Name *com.formcentric.XSRFToken* verwendet. Der Basisname wird automatisch um die ID des zugehörigen Formulars ergänzt.

Parameter	Beschreibung
xssPrevention	Aktiviert die XSS-Absicherung (true, false)

Der Security-Servlet-Filter darf nur einmal pro Webapplikation konfiguriert werden.

6.5.5. Speichern des Formularstatus

Standardmäßig werden alle von einem Benutzer eingegebenen Daten in der User-Session auf dem Server gespeichert. Bei umfangreichen Formularen kann es jedoch passieren, dass die Session abläuft, bevor der Anwender das Formular zu Ende ausgefüllt und abgesendet hat. In diesem Fall gehen die in der Session gespeicherten Daten verloren.

Formcentric bietet daher zusätzlich die Option, die eingegebenen Daten längerfristig zu speichern. Dies ermöglicht es Benutzern, die Formulareingabe zu unterbrechen und zu einem späteren Zeitpunkt fortzusetzen.

Um diese Funktion zu aktivieren, konfigurieren Sie in der Spring-Konfiguration *form-centric-controllers.xml* einen *FormStateStore*.

```
<bean id="defaultFormCommandBeanFactory"
  class="com.formcentric.logicbeans.DefaultFormCommandBeanFactory">
  ...
  <property name="formStateStore" ref="formStateStore"/>
</bean>
```

Formcentric stellt zwei verschiedene Store-Implementierungen zur Verfügung:

BackendFormStateStore

Diese Implementierung speichert die Formulardaten in der Datenbank des Analytics-Backends. Für jeden Datensatz wird eine eindeutige ID generiert und sowohl in der Datenbank als auch in einem Cookie gespeichert. Die Lebensdauer, die Domain und der Pfad des Cookies können in der Spring-Konfiguration angegeben werden.

```
<bean id="formStateStore"
  class="com.formcentric.store.BackendFormStateStore">

  <property name="cookiePath" value="/" />
  <property name="cookieDomain" value="my-domain.com" />
  <property name="cookieMaxAge" value="604800" />
  <property name="backendClient" ref="backendClient" />
</bean>
```

FileFormStateStore

Diese Implementierung speichert die Formulardaten in einer verschlüsselten Datei auf dem Server. Der zugehörige Dateiname wird in einem Cookie gespeichert. Das

Verzeichnis sowie das Verschlüsselungspasswort, die Lebensdauer, die Domain und der Pfad des Cookies können in der Spring-Konfiguration angegeben werden.

```
<bean id="formStateStore"
  class="com.formcentric.store.FileFormStateStore">

  <property name="cookiePath" value="/" />
  <property name="cookieDomain" value="my-domain.com" />
  <property name="cookieMaxAge" value="604800" />
  <property name="secret" value="change-this-now" />
  <property name="storageDir" value="/var/webforms" />
</bean>
```

Der User, mit dem der Application-Server gestartet wurde, muss Schreibrechte in diesem Verzeichnis besitzen. Wenn nichts anderes angegeben ist, wird das in der Systemvariable *java.io.tmpdir* konfigurierte Verzeichnis verwendet. In einer Cluster-Umgebung muss das Verzeichnis zudem von allen Instanzen der Webanwendung erreichbar sein.

6.5.6. Implementierung einer Action

Wie bereits beschrieben, wird die Business-Logik der Formulardatenverarbeitung in der Webanwendung in Actions gekapselt. Konkret handelt es sich dabei um Klassen, die das Interface *com.formcentric.actions.Action* implementieren. Zusätzliche Business-Beans können einer Action über Spring injiziert werden. Auf diese Weise können beispielsweise Data-Access-Objekte (DAO) für den Zugriff auf externe Datenbanken zur Verfügung gestellt werden. Die Actions werden dem Formular-Controller beim Start der Anwendung über Spring injiziert.

Variable Action-Parameter, die bei der Anlage des Formulars vom Redakteur eingegeben werden müssen (beispielsweise die Zieladresse der Mail-Action), werden in der Properties-Map des ActionNode-Beans an die Action-Implementierung übergeben.

Das nachfolgende Beispiel zeigt Ihnen, wie Sie die im Abschnitt 6.3.1, „Entwicklung eines NodeEditorPane“ beschriebene *CustomAction* implementieren und konfigurieren können.

```
public class CustomAction extends BaseAction<WebForm> {

  public static final String PROP_CUSTOM = "customProperty";

  @Override
  public ModelAndView execute(ExecutionContext<WebForm> context, Map<String,
    Object> formData) throws Exception {

    WebForm formDefinition = context.getFormDefinition();
    ActionNode action = context.getAction();

    String customParam = action.getPropertyAsString(PROP_CUSTOM);
```

```
    // Business logic
    ...

    // ModelAndView
    ModelAndView mv = new ModelAndView("success");
    mv.addObject(Constants.ATTRIBUTE_SELF, formDefinition);
    return mv;
  }

  @Override
  public boolean isExecutable(ExecutionContext<WebForm> context,
     Map<String, Object> formData) throws Exception {
    return true;
  }
}
```

Die Action erhält beim Aufruf der *execute*-Methode alle zur Verfügung stehenden Daten. Neben den eigentlichen Formulardaten werden im *ExecutionContext* die Formulardefinition, die Action-Definition, die Formularvariablen (siehe Abschnitt 6.5.7, „Variable zur Vorbelegung von Formularfeldern hinzufügen“) sowie das Request-Objekt übergeben. Die *parameters*-Map enthält nur die Werte der sichtbaren Formularelemente. Durch Aufruf der Methode *getRawFormData()* auf dem *ExecutionContext*-Bean kann auf alle Formulardaten zugegriffen werden.

Die *execute*-Methode muss ein Objekt vom Typ *ModelAndView* zurückgeben. Dieses wird für die Darstellung der Ergebnisseite verwendet, die dem Benutzer nach dem Absenden der Daten angezeigt wird.

In der Regel wird das *ModelAndView*-Objekt mit dem Formular-Bean und einem speziellen View (beispielsweise *success*) erzeugt.

Darüber hinaus besteht aber auch die Möglichkeit, den Request auf eine andere Seite weiterzuleiten. In diesem Fall kann das *ModelAndView*-Objekt wie folgt erzeugt werden:

```
ModelAndView mv = ControllerUtils.redirectTo(renderBean, viewName);
```

Das Objekt *renderBean* und der View-Name können dabei von der speziellen Business-Logik der Action-Implementierung erzeugt werden.

Bei manchen Anwendungsfällen werden Fehler in den Eingabedaten erst bei der Verarbeitung durch das angebundene Backend entdeckt. In diesem Fall soll dem Benutzer nicht die Ergebnisseite, sondern erneut das Formular mit einer Fehlermeldung angezeigt werden. Um dies zu erreichen, muss die Action – analog zu den Validatoren – einen Fehler auf dem im *ExecutionContext* übergebenen *Errors*-Bean erstellen.

```
public ModelAndView execute(ExecutionContext<WebForm> context, Map<String,
Object> formData) throws Exception {
   ...
   context.getErrors().rejectValue("username", DUPLICATE_USER_ERROR,
      "This username is already in use.");
```

```
   ModelAndView mv = new ModelAndView("success");
   mv.addObject("self", formDefinition);
   return mv;
```

Tragen Sie die Action in die Spring-Konfiguration *formcentric-actions.xml* ein:

```
<bean name="customAction" class="com.custom.forms.web.CustomAction">

   <!-- benötigte Properties -->
   ...

</bean>
```

Tragen Sie sie zusätzlich in der Spring-Konfiguration *formcentric-controllers.xml* in das Action-Mapping ein:

```
<bean id="formCommandBeanFactory"
  class="com.formcentric.logicbeans.DefaultFormCommandBeanFactory">

  <!-- Mapping von Action-Namen auf Action-Implementierungen -->
  <property name="actionMapping">
    <map>
      <entry key="mailAction" value-ref="mailAction"/>
      <entry key="customAction" value-ref="customAction"/>
    </map>
  </property>
  ...

</bean>
```

6.5.7. Variable zur Vorbelegung von Formularfeldern hinzufügen

Für die Vorbelegung von Formularfeldern stehen dem Redakteur eine Reihe vordefinierter Variablen zur Verfügung. Standardmäßig kann er die Variablen *date, time, language, ip, remoteUser, principal, userAgent* und *referer* verwenden.

Für die Bereitstellung eigener Variablen müssen Sie die Methode *getVariables()* auf dem FormCommandBean überschreiben.

```
public class CustomFormCommandBean extends DefaultFormCommandBean {

  @Override
  protected Map<String, Object> getVariables(HttpServletRequest request,
      WebForm formDefinition) {

    Map<String, Object> variables =
        super.getVariables(request, formDefinition);

    // eigene Variablen hier einfügen
    ...

    return variables;
  }
```

```
}
```

Zur Instanziierung des neuen *CustomFormCommandBeans* ist es notwendig, dass Sie auch die *FormCommandBean*-Factory überschreiben und diese in die Konfiguration *formcentric-controllers.xml* eintragen.

```
public class CustomCommandBeanFactory extends DefaultFormCommandBeanFactory {

  @Override
  public CustomFormCommandBean createBeanFor(WebForm formDefinition) {

     CustomFormCommandBean commandBean = new CustomFormCommandBean();
     initCommandBean(commandBean, formDefinition);

     return commandBean;
  }
}
```

Tauschen Sie die *DefaultFormCommandBeanFactory* in der Spring-Konfiguration *formcentric-controllers.xml* gegen die *CustomCommandBeanFactory* aus:

```
<bean id="formCommandBeanFactory"
   class="com.custom.forms.web.CustomCommandBeanFactory"
   ...
```

Die Formularfelder werden einmalig beim ersten Aufruf des Formulars mit den definierten Vorbelegungswerten initialisiert. Dabei werden auch die Variablen durch ihre Werte ersetzt. Anschließende Änderungen der Variablenwerte werden daher nicht in ein bereits initialisiertes Formular übernommen.

6.5.8. Implementierung eines REST-Services

Formcentric beinhaltet eine REST-Schnittstelle, die Sie dazu verwenden können, Auswahllisten oder Eingabefelder zur Laufzeit mit Daten aus externen Systemen zu befüllen. Dabei kann es sich um statische, dynamische oder benutzerindividuelle Daten handeln. Alle spezifischen Funktionen der Schnittstelle sind in Klassen vom Typ *com.formcentric.rest.RestService* gekapselt. Durch die Implementierung eines eigenen REST-Services können Sie die Schnittstelle um zusätzliche Funktionen erweitern. Das nachfolgende Beispiel zeigt Ihnen einen REST-Service, der eine Map mit statischen Key/Value-Paaren erzeugt.

```
public class CustomRestService extends BaseRestService {

  @Override
  public Object invoke(ServiceContext<WebForm> context, Map<String,
    Object> remoteFormData, Map<String, Object> localFormData) {

    String myCustomParam =
        context.getConfigParameterMap().get("myCustomParam");
```

```
    ...

    HashMap<String, String> data = new HashMap<String, String>();

    // Map füllen
    data.put("key1", "value1");
    data.put("key2", "value2");
    data.put("key3", "value3");

    return data;
  }
}
```

Bei Aufruf der *invoke*-Methode werden dem RestService neben dem ServiceContext auch die bereits abgesendeten Benutzereingaben (Parameter *remoteFormData*) und die noch nicht abgesendeten Benutzereingaben (Parameter *localFormData*) übergeben. Damit sind Sie in der Lage, direkt auf die Eingaben des Benutzers zu reagieren, unabhängig davon, ob, diese bereits abgesendet wurden oder nicht.

Über den *ServiceContext* haben Sie zudem Zugriff auf die Formulardefinition, das Eingabeelement, die Konfigurationsparameter des RestService und das Request-Object.

Tragen Sie den REST-Service in die Spring-Konfiguration *formcentric-services.xml* ein:

```
<bean name="customRestService"
   class="com.custom.forms.web.CustomRestService">

   <!– benötigte Properties -->
   ...

</bean>
```

Tragen Sie ihn zusätzlich in der Spring-Konfiguration *formcentric-controllers.xml* in das Service-Mapping des REST-Controllers ein:

```
<bean id="restController"
    class="com.formcentric.controllers.RestController">
        <property name="prefix" value="/rest" />
        <property name="commandNamePrefix" value="command" />

        <property name="restServiceMapping">
            <map>
                <entry key="Example" value-ref="exampleRestService"/>
                ...
            </map>
        </property>
</bean>
```

Der Zugriff auf den Service erfolgt über die URL:

```
<context-path>/servlet/rest?_service=Example&_uid=<Dokument-ID>&
                     _input=<Input-Name>
```

Als Antwort des Aufrufs wird folgender JSON-String zurückgegeben:

```
[
  {
    "k":"key1",
    "v":"value1",
    "i":"mwf6aab0bb24033",
    "h":"8d0c3e13950d86c1a7383f066105f78c"
  },
  {
    "k":"key2",
    "v":"value2",
    "i":"mwf06a7a0930d37",
    "h":"d22d445101243a5f616cfd64c765e399"
  },
  {
    "k":"key3",
    "v":"value3",
    "i":"mwf1674ffb0a121",
    "h":"c0ad1fa77bb1b79ca757ee1ffce9f416"
  }
]
```

Um Manipulationen an den übertragenen JSON-Daten zu verhindern, werden die einzelnen Key/Value-Paare durch einen zusätzlichen HASH-Wert abgesichert, der beim Absenden des Formulars serverseitig validiert wird.

Aufgrund dieser Absicherung können keine externen REST-Services aufgerufen werden, da deren Daten nicht die erforderlichen HASH-Werte enthalten. Für den Zugriff auf externe Services können Sie jedoch einen eigenen Proxy-REST-Service implementieren, der seinerseits auf den externen REST-Service zugreift.

Ab Version 2.3 von Formcentric erfolgt der Aufruf der REST-Service innerhalb der JSP-Templates, über das HTML-Attribut *data-mwf-datasource*. Im Attributwert müssen Sie ein JSON-Objekt angeben, das die URL des REST-Services, die Verwendungsart (*checkbox, radio, selection, suggestion* oder *hidden*) sowie ggf. weitere Parameter enthält.

Standardmäßig können Sie bei folgenden Eingabeelementen einen REST-Service angeben:

inputField:

```
<c:url value="/servlet/rest" var="restUrl">
   <c:param name="_service" value="${self.properties['datasource']}"/>
   <c:param name="_uid" value="${form.uid}"/>
   <c:param name="_input" value="${self.name}"/>
   <fcs:xsrfTokenParam/>
</c:url>

<c:set var="params" value="${self.properties['datasource_params']}"/>

<form:input data-mwf-id="${self.id}"
            data-mwf-datasource='{
```

```
                "type" : "suggestion",
                "url" : "${restUrl}",
                "data" : {},
                "params" : ${params}
            }' ... />
```

hiddenField:

```
<c:url value="/servlet/rest" var="restUrl">
   <c:param name="_service" value="${self.properties['datasource']}"/>
   <c:param name="_uid" value="${form.uid}"/>
   <c:param name="_input" value="${self.name}"/>
   <fcs:xsrfTokenParam/>
</c:url>

<c:set var="params" value="${self.properties['datasource_params']}"/>

<form:hidden path="${self.name}" data-mwf-id="${self.id}"
            data-mwf-datasource='{
                "type" : "hidden",
                "url" : "${restUrl}",
                "data" : {},
                "params" : ${params}
            }' ... />
```

comboBox:

```
<![CDATA[<c:url value="/servlet/rest" var="restUrl">
   <c:param name="_service" value="${input.properties['datasource']}"/>
   <c:param name="_uid" value="${form.uid}"/>
   <c:param name="_input" value="${input.name}"/>
   <fcs:xsrfTokenParam/>
</c:url>

<fc:valueOut var="userValue" name="${input.name}"/>
<c:set var="strUserValue" value="${fn:join(userValue,',')}"/>
<c:set var="params" value="${input.properties['datasource_params']}"/>

<form:select data-mwf-id="${self.id}"
             data-mwf-datasource='{
                "type" : "selection",
                "url" : "${restUrl}",
                "preselected" : "${strUserValue}",
                "data" : {},
                "params" : ${params}
             }' ... >
   ...
</form:select>
```

checkBoxGroup:

```
<c:url value="/servlet/rest" var="restUrl">
   <c:param name="_service" value="${input.properties['datasource']}"/>
   <c:param name="_uid" value="${form.uid}"/>
   <c:param name="_input" value="${input.name}"/>
```

```
   <fcs:xsrfTokenParam/>
</c:url>

<fc:valueOut var="userValue" name="${input.name}"/>
<c:set var="strUserValue" value="${fn:join(userValue,',')}"/>
<c:set var="params" value="${input.properties['datasource_params']}"/>

<fieldset data-mwf-id="${self.id}"
          data-mwf-datasource='{
             "name" : "${input.name}",
             "type" : "checkbox",
             "url" : "${restUrl}",
             "preselected" : "${strUserValue}",
             "data" : {},
             "params" : ${params}
          }' ... >
   ...
</fieldset>
```

radioGroup:

```
<c:url value="/servlet/rest" var="restUrl">
   <c:param name="_service" value="${self.properties['datasource']}"/>
   <c:param name="_uid" value="${form.uid}"/>
   <c:param name="_input" value="${self.name}"/>
   <fcs:xsrfTokenParam/>
</c:url>

<fc:valueOut var="userValue" name="${self.name}"/>
<c:set var="strUserValue" value="${fn:join(userValue,',')}"/>
<c:set var="params" value="${self.properties['datasource_params']}"/>

<fieldset data-mwf-id="${self.id}"
          data-mwf-datasource='{
             "name" : "${self.name}",
             "type" : "radio",
             "url" : "${restUrl}",
             "preselected" : "${strUserValue}",
             "data" : {},
             "params" : ${params}
          }' ... >
   ...
</fieldset>
```

Da innerhalb des JSON-Strings das doppelte Anführungszeichen verwendet wird, müssen Sie für das HTML-Attribut das einfache Anführungszeichen verwenden.

Wie zuvor beschrieben, stehen Ihnen innerhalb des RestServices die abgesendeten und die noch nicht abgesendeten Formulareingaben zur Verfügung. Dies können Sie beispielsweise dafür verwenden, einen RestService zu implementieren, der zu einer vom Benutzer eingegebenen Postleitzahl passende Standorte in einer Auswahlliste zurückgibt.

In diesem Beispiel ist es sinnvoll, die Auswahlliste automatisch zu aktualisieren, wenn der Anwender die Postleitzahl ändert, da zu der geänderten Postleitzahl evtl. andere Standorte gehören. Hierfür steht der Parameter *dependsOn* zur Verfügung. Dieser kann in der Redaktionsoberfläche in die Parameterliste eines RestService eingetragen werden (siehe dazu auch Kap. 3.2.5 im Benutzerhandbuch). Als Wert müssen dabei die Namen der Eingabeelemente angegeben werden, von denen das Ergebnis des ausgewählten RestService abhängt. Jede Änderung in einem der angegebenen Eingabeelemente führt dazu, dass der RestService erneut aufgerufen wird.

6.5.9. Template-Entwicklung

Die Ausgabe der Formulare und Formularelemente erfolgt durch FreeMarker- oder JSP Templates innerhalb der Webapplikation.

Formcentric ist standardmäßig so konfiguriert, dass die JSP-Templates verwendet werden. Um die WebApp auf Freemarker-Templates umzustellen, müssen Sie in der Spring-Konfiguration *formcentric-views.xml* anstelle der Konfiguration *formcentric-views-jsp.xml* die Konfiguration *formcentric-views-freemarker.xml* importieren.

Alle Formularelemente werden auf der Datenebene (Model) durch ein Objekt vom Typ *com.formcentric.model.xml.InputNode* repräsentiert. Auf die Eigenschaften *id, type, name, label, value, parent* und *children* können Sie über entsprechende Getter-Methoden auf dem *InputNode*-Bean zugreifen. Der Zugriff auf alle weiteren Eigenschaften erfolgt über die Properties-Map des InputNode-Beans.

Die nachfolgende Tabelle zeigt Ihnen alle Formularelementtypen und deren Eigenschaften. Die in eckigen Klammern dargestellten Properties müssen aus der Properties-Map gelesen werden.

Element	**Eigenschaften**
form	name, [style_class, form_style_class, next_label, submit_label, cancel_label, script, description, save_state, save_statistics, doi_to, doi_note, doi_message, doi_from, doi_sender, doi_subject, doi_enabled, doi_format, doi_condition, doi_condition_conjunction]
inputField	name, label, value, [hint, placeholder, field_width, style_class, readonly, maxlength, datasource, datasource_params]
shortText	name, label, value, [hint, placeholder, field_width, style_class, readonly, maxlength]
emailField	name, label, value, [hint, placeholder, field_width, style_class, readonly]
numberField	name, label, value, [hint, placeholder, field_width, style_class, readonly, maxlength]
dateField	name, label, value, [hint, placeholder, field_width, style_class, readonly, maxlength]

Element	Eigenschaften
phoneField	name, label, value, [hint, placeholder, field_width, style_class, readonly, maxlength]
textArea	name, label, value, [hint, placeholder, field_width, style_class, readonly, maxlength, rows, cols]
passwordField	name, label, [hint, placeholder, field_width, style_class]
button	name, label, [hint, field_width, style_class, onclick]
checkBoxGroup	name, label, children, [pictureUrl, pictureFileName, hint, field_width, style_class, datasource, dynamic, datasource_params]
comboBox	name, label, value, children, [pictureUrl, pictureFile-Name, hint, field_width, style_class, datasource, dynamic, datasource_params]
picture	name, [pictureUrl, pictureFileName, alt, field_width, style_class]
pageBreak	name, label, [style_class, condition, next_label, back_label, script]
paragraph	name, value, [bold, italic, field_width, style_class]
captcha	name, label, [field_width, hint]
radioGroup	name, label, children, [pictureUrl, pictureFileName, hint, field_width, style_class, datasource, dynamic, datasource_params]
summary	label, [style_class, elements, hide_empty_fields]
hiddenField	name, value, [datasource, datasource_params]
fileUpload	name, [multiple, auto_upload, hint, field_width, style_class]
condition	[condition, condition_conjunction, conditional_fields]
pageCondition	[condition, condition_conjunction, next_page, script]
layout	label, [layout]
fieldSet	name, label, [style_class]
calculatedValue	name, label, [script, visible, clientside, style_class]
mailAction	[subject, to, cc, bcc, from, sender, body, format, note, replyto, send_hidden_fields, condition, condition_execute, condition_conjunction, redirect_url, hide_empty_fields]
dataSourceAction	[note, schema, table, datasource_mapping, commit_message, release, condition, condition_execute, condition_conjunction]
mediaStoreAction	[note, mediastore_mapping, commit_message, release, condi-tion, condition_execute, condition_conjunction]

Element	Eigenschaften
pdfAction	[note, pdfFileName, pdfPath, pdfUid, pdfUrl, field_mapping, linktext, readonly, condition, condition_execute, condition_conjunction]
datastoreAction	[note, condition, condition_execute, condition_conjunction]
redirectAction	[note, condition, condition_execute, condition_conjunction, url, content, delay]
webhookAction	[note, condition, condition_execute, condition_conjunction, url, fields, url_parameters, custom_headers, content_type]
sequenceAction	-

Neben den oben beschriebenen InputNode-Beans übergibt das System im Request weitere Objekte an die Formular-Templates. Die nachfolgende Tabelle zeigt Ihnen eine Übersicht aller übergebenen Objekte.

Parametername	Typ	Beschreibung
self	com.formcentric.model.WebForm	Bean des aktuellen Formulardokuments
input	com.formcentric.model.xml.InputNode	Bean des aktuellen Formularelements
pageElements	java.util.List	Liste mit den Elementen der aktuellen Formularseite
pageCount	java.lang.Integer	Anzahl der Formularseiten
form	com.formcentric.model.WebForm	Formulardefinition
currentPage	java.lang.Integer	Seitennummer der aktuellen Formularseite
currentPageNode	com.formcentric.model.xml.InputNode	Bean der aktuellen Formularseite
formdata	java.util.Map	Map mit den vom Anwender eingegebenen Formulardaten

FreeMarker Templates

Die Freemarker-Templates sind im Verzeichnis */WEB-INF/templates/ftl* abgelegt. Neben den übergeordneten Formular-Templates ist jedem Formularelementtyp ein eigenes Template zugeordnet. Das nachfolgende Beispiel zeigt Ihnen das Template *textArea.ftl* des mehrzeiligen Texteingabefeldes:

```
<li data-mwf-container="${self.id}"
    class="mwf-field ${self.properties['style_class']!""}">
    <label class="mwf-label" for="${self.id}">${self.label!""}
        <#if self.required><span class="mwf-required">*</span></#if>
    </label>
    <div class="mwf-input">

        <@spring.bind mwf.bind(self)/>
        <#assign hasErrors=spring.status.error />
        <textarea id="${self.id}"
              class="mwf-text ${self.properties['style_class']!""}"
              name="${spring.status.expression!""}"
              ${self.properties['readonly']?boolean?then("readonly", "")}
              maxlength="${self.properties['maxlength']!""}"
              spellcheck="true"
              placeholder="${self.properties['placeholder']!""}"
              rows="${self.properties['rows']!""}"
              cols="${self.properties['cols']!""}"
              data-mwf-id="${self.id}">${spring.status.value!""}</textarea>
        <#if self.properties['hint']?has_content>
            <div class="mwf-hint">
                <@fc.markdown value=(self.properties['hint']!"")?string />
            </div>
        </#if>
        <@spring.showErrors separator="<p>" classOrStyle="mwf-error"/>
    </div>
</li>
```

Freemarker Funktionen und Makros

Formcentric stellt Ihnen eine Freemarker Library zur Verfügung, die spezialisierte Funktionen für die Darstellung der Formulare beinhaltet.

Um die Funktionen verwenden zu können, fügen Sie folgende Anweisung in die Freemarker-Templates ein:

```
<#import "/lib/formcentric.com/webforms.ftl" as mwf>
```

Nachfolgend finden Sie die in der Library enthaltenen Funktionen beschrieben.

mwf.forEachPageElement

Listen-Funktion, die die Elemente der aktuellen Seite beinhaltet.

```
forEachPageElement(boolean layoutFacets, boolean removeEmptyFacets,
    final String exclude, final String include)
```

Parameter	Beschreibung
layoutFacets	Falls dieser Wert *true* ist, werden die Elemente in layoutFacets aufgeteilt. (optional)
removeEmptyFacets	Legt fest, ob leere Layouts bei Erstellung der Liste ignoriert werden sollen. (optional)

Parameter	Beschreibung
	Standardwert: *false*
exclude	Kommaseparierte Liste der Elementtypen, die bei der Erstellung der Liste ignoriert werden. Erfolgt hier keine Angabe, so werden alle Elementtypen – mit Ausnahme der exkludierten Typen – berücksichtigt. (optional)
include	Kommaseparierte Liste der Elementtypen, die bei der Erstellung der Liste berücksichtigt werden. (optional)

```
<#list mwf.forEachPageElement(true, false, "condition", "") as layout>
    <ul class="${layout.properties['layout']!""}">

        <#if layout_index == 0 && currentPage == 0 && self.label?has_content>
            <li class="mwf-field"><h3>${self.label!""}</h3></li>
        </#if>

        <#list layout.items as input>
            <@fc.include self=input view=input.type />
        </#list>
    </ul>
</#list>
```

mwf.forEachPage

Listen-Funktion, die eine Liste der gesammelten Seiten zurückliefert.

```
forEachPage(final boolean compact)
```

Parameter	Beschreibung
compact	Legt fest, dass Formularseiten mit gleichem Titel zusammengefasst werden. (optional) **Standardwert:** *false*

mwf.include

Makro, welches eine Bean mit einem bestimmten Template inkludiert.

```
<@fc.include self=input view=input.type />
```

Attribut	Beschreibung
self	Bean die inkludiert wird.
view	Name des zu inkludierenden Templates.
params	(optional) Ein extended hash mit welchem Sie weitere Parameter an das inkludierte Template übergeben können.

mwf.url

Funktion, mit der absolute URLs auf Formcentric Controller erzeugt werden können.

Wenn im Parameter *baseUrl* nichts angegeben ist, verweist die erzeugte URL auf die formcentric Webapp, in der das FreeMarker-Template aufgerufen wurde.

Diese Funktion ist hilfreich, wenn Sie die Formcentric Webapplikation auf einem anderen Host als die umgebene Webseite installieren möchten, da Sie in diesem Fall keine relativen URLs verwenden können.

Parameter	**Beschreibung**
base	Angabe des Formcentric-Controllers.
params	Ein erweiterter HASH mit welchem Sie weitere Parameter an die URL übergeben können (optional=.
baseUrl	Voll qualifizierter Hostname (optional), Beispiel: *https://your.domain.com:8000*

```
<#assign restUrl=fc.url("/servlet/rest", {
        "_uid": form.uid,
        "_input": self.name,
        "_service": self.properties['datasource'],
        fc.xsrfTokenName(form.uid): fc.xsrfTokenValue(form.uid)
}) />
```

mwf.responseHeader

Makro, mittels dessen Sie einen Header in der Response setzen können.

```
<@fc.responseHeader name="Content-Type"
    value="text/html; charset=UTF-8"/>
```

mwf.summary

Funktion, die eine Liste aller Elemente als *com.formcentric.model.InputBean* des Formulars zurückliefert. Dabei können auch die vom Benutzer eingegebenen Daten abgefragt werden.

Folgende Properties können darauf abgefragt werden:

name	**Name des Formularelements**
label	Beschriftung des Formularelements.
type	Typ des Formularelements.
object	Value-Bean des Formularelements.
value	String-Repräsentation des Value-Beans.
valueLabels	String-Array mit den Labels der Optionen, die in der Auswahl (*comboBox, radioGroup, checkboxGroup*) selektiert wurden. Wenn es sich bei dem zugehörigen Eingabeelement nicht um eine Auswahl handelt, so wird der Wert des Elements in dem Array zurückgegeben.

name	Name des Formularelements
page	Nummer der Seite, auf der sich das Element befindet.
pageLabel	Beschriftung der Seite, auf der sich das Element befindet.
layout	Name des Layouts, in dem sich das Element befindet.
input	*InputNode* des Elements.

```
summary(InputNode self, String elements,
    final String include, final String exclude,
    final boolean hideEmptyFields, final String excludeIfEmpty)
```

Parameter	Beschreibung
self	*InputNode* eines Formularelements. Wenn dieser Wert gesetzt ist, so wird die Iteration bei dem angegebenen Element abgebrochen.
elements	Kommaseparierte Liste mit den Namen der Formularelemente, die in der Zusammenfassung angezeigt werden sollen. Wenn dieses Attribut gefüllt ist, wird das Attribut *self* ignoriert. (optional)
include	Kommaseparierte Liste der Elementtypen, die bei der Iteration berücksichtigt werden. Erfolgt hier keine Angabe, so werden alle Elementtypen – mit Ausnahme der exkludierten Typen – berücksichtigt. (optional)
exclude	Kommaseparierte Liste der Elementtypen, die bei der Iteration ignoriert werden. (optional) **Standardwert:** *button, hiddenField, condition, pageCondition, pagebreak, captcha, passwordField*
hideEmptyFields	Legt fest, dass alle leeren Felder ignoriert werden. **Standardwert:** *false*
excludeEmptyFields	Legt fest, dass leere Felder ignoriert werden. (optional) **Standardwert:** *false*

```
<#list mwf.summary(self, self.getPropertyAsString('elements'),
        self.getPropertyAsBoolean('hide_empty_fields', false)) as item>
    <tr>
        <#if item.input.type == "paragraph">
            <td colspan="2">
                <@fc.markdown>${item.input.value}</@fc.markdown>
            </td>
        <#else>
            <td>${item.label?has_content?then(item.label, item.name!"")}</td>
            <td>${(item.valueLabels![])?join(", ")}</td>
        </#if>
```

```
    </tr>
</#list>
```

mwf.captcha

Template, mit dem Sie ein Captcha-Bild erzeugen können.

Attribut	Beschreibung
url	URL des Captcha-Servlets.
id	Die ID des Captcha-InputNodes.
linkClass	CSS-Klasse(n) die auf den Link um das Captcha-Bild angewandt wird. (optional) **Standardwert:** *""*
imgClass	CSS-Klasse(n) die auf das Captcha-Bild angewandt wird. (optional) **Standardwert:** *""*
title	Das title-Attribut für das Captcha-Bild. (optional) **Standardwert:** *""*
alt	Das alt-Attribut für das Captcha-Bild. (optional) **Standardwert:** *Captcha*

```
<#assign captchaUrl=mwf.url("/servlet/captcha/captcha.jpg")!"" />
<@fc.captcha url=captchaUrl id=self.id linkClass="css-class__link"
    imgClass="css-class__img" title="A title" alt="Captcha" />
```

mwf.ifCaptcha

Boolean Funktion, die auswertet, ob das Captcha *name* **nicht** korrekt eingegeben wurde.

Parameter	Beschreibung
name	Name des Captcha-Elements.

```
<#if fc.ifCaptcha(self.name!"")>
    <#assign captchaUrl=mwf.url("/servlet/captcha/captcha.jpg")!"" />
    <@fc.captcha url=captchaUrl id=self.id />
</#if>
```

mwf.getStandardButton

Funktion, die den über das Attribut *buttonType* bestimmten StandardButton des Formulars liefert.

Parameter	Beschreibung
buttonType	Typ des Standard-Buttons. Er kann die folgenden Werte annehmen: _next Button, der auf die nächste Seite des Formulars leitet. _back Button, der auf die vorherige Seite des Formulars leitet. _cancel Button, der die Bearbeitung des Formulars abbricht. _finish Button, der das Formular abschickt. _exit Button, mit dem das Formular verlassen werden kann.

```
<#assign finishButton=mwf.getStandardButton("_finish") />
<li data-mwf-container="${finishButton.id}" class="mwf-button mwf-next">
    <input type="button" value="${submitLabel}"
        data-mwf-submit='{"type":"finish",
            "query": "navigationId=${cmpage.navigation.contentId}"}'/>
</li>
```

mwf.valueOut

Funktion, mit der der aktuelle Wert eines Formularfeldes ausgegeben werden kann.

```
valueOut(String name, boolean preferLabel)
```

Parameter	Beschreibung
name	Name des Formularfeldes
preferLabel	Legt fest, dass anstelle des Werts das Label des Werts ausgegeben werden soll. (optional)

```
${fc.valueOut(self.name!"", true)!""}
```

mwf.conditions

Funktion, mit der die Javascript-Definitionen der Bedingungselemente erzeugt werden können. Platzieren Sie die Funktion am Ende des Formular-Templates.

```
<#assign conditions=mwf.conditions() />
```

mwf.calculatedValues

Funktion, die die JSON-Definitionen der *Berechneten Werte* erzeugt.

```
<#assign calculatedValues=mwf.calculatedValues() />
```

mwf.markdown

Makro, mit dem ein Wert mit Markdown interpretiert ausgegeben werden kann. Dieses Makro kann sowohl mit einem übergebenen Wert (s. Parameter *value*) oder mit einem body umgehen.

Parameter	Beschreibung
value	Wert der mit Markdown interpretiert werden soll. (optional)
inline	Legt fest, ob das Ausgabe-HTML auf Inline Elemente beschränkt werden soll. (optional) **Standardwert:** *false*

```
<@fc.markdown value=self.value!"" inline=false />
<#-- or -->
<@fc.markdown inline=false>${self.value!""}</@fc.markdown>
```

mwf.vars

Makro, mit dem Variablen aus dem Formularkontext in der Ausgabe ersetzt werden.

Parameter	Beschreibung
map	Map mit den Formulardaten

```
<@fc.vars map=formdata>${action.properties['note']!""}</@fc.vars>
```

mwf.bind

Funktion, die den Pfad zurückgibt, an den der Node gebunden ist.

Parameter	Beschreibung
node	InputNode, dessen Pfad gesucht wird.

```
<@spring.bind mwf.bind(self) />
```

mwf.encodeUrl

Funktion, die die übergebene URL in UTF-8 encodiert.

Parameter	Beschreibung
url	URL, die codiert werden soll.

mwf.hasValidator

Funktion, die überprüft, ob der im Parameter *name* festgelegte Validator in dem Input-Node *node* vorhanden ist.

Parameter	Beschreibung
node	InputNode, der untersucht werden soll.
name	Name des Validators

mwf.validatorByName

Funktion, die den im Parameter *name* festgelegten Validator des InputNode *node* zurückgibt.

Parameter	Beschreibung
node	InputNode, der untersucht werden soll.
name	Name des Validators

mwf.elementByName

Funktion, die den im Parameter *name* festgelegten InputNode des Formular *form* zurückgibt.

Parameter	Beschreibung
form	Formular, das den InputNode enthält.
name	Name des Elements

Security Library

Für die Erzeugung und Ausgabe von XSRF-Token (siehe Abschnitt 6.5.4, „Web-Security“) beinhaltet Formcentric eine Security-Library. Die Security-Library wird bei der Integration der Freemarker Funktionen mit eingebunden.

Nachfolgend beschriebene Freemarker Funktionen sind darin enthalten.

mwf.xsrfToken

Makro, das ein verstecktes Formularfeld mit einem XSRF-Token erzeugt.

```
<@fc.xsrfToken />
```

mwf.xsrfTokenName

Funktion, die aus der Formular-ID einen xsrfTokenName erzeugt.

Parameter	Beschreibung
formId	ID des Formulars, für das der Token erzeugt werden soll. Falls dieser Parameter leer ist, wird die im Request mitgeschickte Formular-ID verwendet. (optional)

```
<#assign restUrl=mwf.url("/servlet/rest", {"_uid": form.uid, ...,
```

```
  "tokenName": fc.xsrfTokenName(), "tokenValue": fc.xsrfTokenValue()})!""/>
```

mwf.xsrfTokenValue

Funktion, die aus der Formular-ID einen xsrfTokenValue erzeugt.

Parameter	Beschreibung
formId	ID des Formulars, für das der Token erzeugt werden soll. Falls dieser Parameter leer ist, wird die im Request mitgeschickte Formular-ID verwendet. (optional)

```
<#assign restUrl=mwf.url("/servlet/rest", {"_uid": form.uid, ...,
  "tokenName": fc.xsrfTokenName(), "tokenValue": fc.xsrfTokenValue()})!""/>
```

JSP Templates

Die JSP-Templates sind im Verzeichnis */WEB-INF/templates/jsp* abgelegt. Neben den übergeordneten Formular-Templates ist jedem Formularelementtyp ein eigenes Template zugeordnet. Das nachfolgende Beispiel zeigt Ihnen das Template *textArea.jsp* des mehrzeiligen Texteingabefeldes:

```
<%@ page contentType="text/html;charset=UTF-8" language="java"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<jsp:useBean id="self" type="com.formcentric.model.xml.InputNode"
     scope="request"/>
<jsp:useBean id="form" type="com.formcentric.model.WebForm"
     scope="request"/>

<c:set var="id" value="${self.id}_${form.uid}"/>
<mwf:hasErrors var="hasErrors" path="${self.name}"/>
<li data-mwf-container="${id}" class="mwf-field">
    <label class="mwf-label" for="${id}">
            <c:out value="${self.label}"/>
            <c:if test="${self.required}">
                <span class="mwf-required">*</span>
            </c:if>
    </label>
    <div class="mwf-input">
        <form:textarea id="${id}" path="${self.name}"
                       cssClass="mwf-text ${self.properties['field_width']}"
                       readonly="${self.properties['readonly']}"
                       maxlength="${self.properties['maxlength']}"
                       spellcheck="true"
                       rows="${self.properties['rows']}"
                       cols="${self.properties['cols']}"
                       data-mwf-id="${id}"
                       placeholder="${self.properties['placeholder']}"/>
        <c:if test="${not empty self.properties['hint']}">
            <p class="mwf-hint">
                <small><c:out value="${self.properties['hint']}"/></small>
```

```
            </p>
        </c:if>
        <form:errors path="${self.name}" cssClass="mwf-error" element="p"/>
    </div>
</li>
```

Taglib webfoms-1.0

Formcentric stellt Ihnen eine Tag-Library zur Verfügung, die spezialisierte Tags für die Darstellung der Formulare beinhaltet.

Um die Taglib verwenden zu können, fügen Sie folgende Anweisung in die JSP-Templates ein:

```
<%@taglib prefix="fc" uri="http://www.monday-consulting.com/webfoms-1.0"%>
```

Nachfolgend finden Sie die in der Library enthaltenen Tags beschrieben.

mwf:forEachPageElement

Iterator-Tag, mit dem Sie über die einzelnen Elemente einer Formularseite iterieren können. Das aktuelle Formularelement wird dabei in der Laufvariablen übergeben.

Attribut	**Beschreibung**
var	Name der Laufvariablen, die das aktuelle Formularelement beinhaltet.
varStatus	Iterationsstatus
exclude	Kommaseparierte Liste der Elementtypen, die bei der Iteration ignoriert werden. Erfolgt hier keine Angabe, so werden alle Elementtypen – mit Ausnahme der exkludierten Typen – berücksichtigt.
include	Kommaseparierte Liste der Elementtypen, die bei der Iteration berücksichtigt werden.
layoutFacets	Legt fest, ob die Formularelemente nach Layouts gruppiert ausgegeben werden sollen. In diesem Fall beinhaltet die Laufvariable das aktuelle Layout. Dabei handelt es sich um ein Bean vom Typ *PageIterateTag.LayoutFacet*. Folgende Properties können darauf abgefragt werden: layout — Layoutbezeichnung items — Liste mit den Formularelementen, die diesem Layout zugeordnet sind. **Standardwert:** *false*
removeEmptyFacets	Legt fest, ob leere Layouts bei der Iteration ignoriert werden sollen. **Standardwert:** *false*

```
<mwf:forEachPageElement var="input"
    exclude="pageBreak,calculatedValue,condition">
    <mwf:include self="${input}" view="${input.type}"/>
</mwf:forEachPageElement>
```

```
<mwf:forEachPageElement var="layout" varStatus="s" layoutFacets="true"
   exclude="pageBreak,calculatedValue,condition">

  <c:choose>

    <%-- einspaltiges Layout --%>
    <c:when test="${empty layout.layout or layout.layout eq '1'}">
      <fieldset class="onecolumn">
           <c:forEach items="${layout.items}" var="input" varStatus="i">
              <mwf:include self="${input}" view="${input.type}"/>
           </c:forEach>
      </fieldset>
    </c:when>

    <%-- mehrspaltiges Layout --%>
    <c:when test="${not empty layout.layout and layout.layout eq '2'}">
      <fieldset class="twocolumn">
           <c:forEach items="${layout.items}" var="input" varStatus="i">
              <mwf:include self="${input}" view="${input.type}"/>
           </c:forEach>
      </fieldset>
    </c:when>

  </c:choose>
</mwf:forEachPageElement>
```

mwf:forEachPage

Iterator-Tag, mit dem Sie über die Seiten eines Formulars iterieren können. In der Laufvariablen werden der Seitentitel sowie weitere Statusinformationen übergeben.

Attribut	**Beschreibung**
var	Name der Laufvariablen, die ein Bean vom Typ *ForEachPageTag.Item* enthält. Folgende Properties können darauf abgefragt werden: label — Beschriftung der Formularseite index — Seitennummer selected — Gibt an, ob die zugehörige Formularseite gerade angezeigt wird. finished — Gibt an, ob die zugehörige Formularseite bereits ausgefüllt wurde.
varStatus	Iterations-Status.
compact	Legt fest, dass Formularseiten mit gleichem Titel zusammengefasst werden.

Attribut	Beschreibung
	Standardwert: *false*

```
<mwf:forEachPage var="page" varStatus="status">
  <c:set var="class" value="${entry.selected ? 'selected' : ''}"/>
  <span class="${class}">
    <c:out value="${status.index}"/>. <c:out value="${page.label}"/>
  </span>
</mwf:forEachPage>
```

mwf:summary

Tag, mit dem Sie über alle Elemente eines Formulars iterieren können. Neben den Elementeigenschaften *Name, Typ* und *Label* wird auch der vom Benutzer eingegebene Wert in der Laufvariablen übergeben.

Attribut	Beschreibung
var	Name der Laufvariablen, die ein Bean vom Typ *com.formcentric.model.InputBean* enthält. Folgende Properties können darauf abgefragt werden: name — Name des Formularelements label — Beschriftung des Formularelements type — Typ des Formularelements object — Value-Bean des Formularelements value — String-Repräsentation des Value-Beans valueLabels — String-Array mit den Labels der Optionen, die in der Auswahl (*comboBox, radioGroup, checkboxGroup*) selektiert wurden. Wenn es sich bei dem zugehörigen Eingabeelement nicht um eine Auswahl handelt, so wird der Wert des Elements in dem Array zurückgegeben. page — Nummer der Seite, auf der sich das Element befindet. pageLabel — Beschriftung der Seite, auf der sich das Element befindet. layout — Name des Layouts, in dem sich das Element befindet. input — *InputNode* des Elements.
varStatus	Iterations-Status.
elements	Kommaseparierte Liste mit den Namen der Formularelemente, die in der Zusammenfassung angezeigt werden sollen. Wenn dieses Attribut gefüllt ist, wird das Attribut *self* ignoriert.

Attribut	Beschreibung
self	*InputNode* eines Formularelements. Wenn dieser Wert gesetzt ist, so wird die Iteration bei dem angegebenen Element abgebrochen.
include	Kommaseparierte Liste der Elementtypen, die bei der Iteration berücksichtigt werden. Erfolgt hier keine Angabe, so werden alle Elementtypen – mit Ausnahme der exkludierten Typen – berücksichtigt.
exclude	Kommaseparierte Liste der Elementtypen, die bei der Iteration ignoriert werden. **Standardwert:** *button, hiddenField, condition, pageCondition, pagebreak, captcha, passwordField*
hideEmptyFields	Legt fest, dass alle leeren Felder ignoriert werden. **Standardwert:** *false*
excludeIfEmpty	Geben Sie hier die Typen der Formularfelder an, die ignoriert werden sollen, wenn sie leer sind.

```
<table>
  <fc:summary exclude="hiddenField" var="item"
              elements="${self.properties['elements']}"
              hideEmptyFields="${self.properties['hide_empty_fields']}">
    <tr>
      <td><c:out value="${empty item.label ? item.name : item.label}"/></td>
      <td><c:out value="${fn:join(item.valueLabels, ', ')}"/></td>
    </tr>
  </fc:summary>
</table>
```

mwf:captcha

Tag, mit dem Sie ein Captcha-Bild erzeugen können.

Attribut	Beschreibung
url	URL des Captcha-Servlets
alt	Alternativtext des Captcha-Images

```
<mwf:captcha url="/servlet/captcha/captcha.jpg" path="${self.name}"
  title="Click for new Captcha"/>
```

mwf:captchaLink

Tag, mit dem Sie einen Reload-Link auf dem eingebetteten Captcha-Bild erzeugen können. Der erzeugte Link verfügt über einen *onClick* Event-Handler, der das Captcha-Bild beim Klick des Benutzers durch ein neues ersetzt.

Attribut	Beschreibung
url	URL des Captcha-Servlets

```
<mwf:captchaLink url="/servlet/captcha/captcha.jpg" path="${self.name}">
  <mwf:captcha url="/servlet/captcha/captcha.jpg" path="${self.name}"
        title="Click for new Captcha"/>
</mwf:captchaLink>
```

mwf:ifCaptcha

Conditional-Tag, dessen Body **nicht** ausgeführt wird, wenn das im Attribut *name* angegebene Captcha korrekt eingegeben wurde.

Attribut	Beschreibung
name	Name des Captcha-Elements
var	Name der Variable, in der gespeichert wird, ob das Captcha korrekt eingegeben wurde.

```
<mwf:ifCaptcha name="${self.name}">
  <mwf:captchaLink url="/jcaptcha/captcha.jpg" path="${self.name}">
    <mwf:captcha url="/jcaptcha/captcha.jpg" path="${self.name}"
          title="Click for new Captcha"/>
  </mwf:captchaLink>
</mwf:ifCaptcha>
```

mwf:getStandardButton

Tag, mit dem auf die Standard-Buttons des Formulars zugegriffen werden kann.

Attribut	Beschreibung
var	Name der Variable, in der der Button gespeichert wird.
buttonType	Typ des Standard-Buttons. Er kann die folgenden Werte annehmen: _next Button, der auf die nächste Seite des Formulars leitet. _back Button, der auf die vorherige Seite des Formulars leitet. _cancel Button, der die Bearbeitung des Formulars abbricht. _finish Button, der das Formular abschickt. _exit Button, mit dem das Formular verlassen werden kann.

```
<fc:getStandardButton buttonType="_finish" var="finishButton" />
<li data-mwf-container="${finishButton.id}" class="mwf-button mwf-next">
    <input type="button" value="${fn:escapeXml(submitLabel)}"
```

```
      data-mwf-id="${finishButton.id}" data-mwf-submit='{"type":"finish"}'/>
</li>
```

mwf:forEachCondition

Tag, mit dem Sie über die einzelnen Regeln einer Bedingung iterieren können. Bei der Iteration werden das Input-Element, der Operator und der Vergleichswert in der Laufvariablen zur Verfügung gestellt.

Attribut	Beschreibung
var	Name der Laufvariablen, die ein Bean vom Typ *ForEachConditionTag.Rule* enthält. Folgende Properties können darauf abgefragt werden: input Eingabeelement auf das sich die Bedingung bezieht. operator Logischer Operator der Bedingung value Vergleichswert der Bedingung
self	Condition-Element
varStatus	Iterations-Status

```
<mwf:forEachCondition self="${self}" var="cond">
   <c:out value="${cond.input.name}"/>
   <c:out value="${cond.operator}"/>
   <c:out value="${cond.value}"/>
</mwf:forEachCondition>
```

mwf:valueOut

Tag, mit dem der aktuelle Wert eines Formularfeldes ausgegeben werden kann.

Attribut	Beschreibung
name	Name des Formularfeldes
var	Name der Variablen, in der der Wert des Feldes gespeichert werden soll.
preferLabel	Legt fest, dass anstelle des Werts das Label des Werts ausgegeben werden soll.
escapeXml	Ein boolsches Flag, das angibt, ob die Standard-XML-Entities, wie beispielsweise „<“ oder „&“ in ihre Entity-Codes umgewandelt werden sollen.

```
<mwf:valueOut name="${input.name}"/>
```

mwf:conditions

Tag, der die JSON-Definitionen der Bedingungselemente erzeugt.

Attribut	Beschreibung
var	Name der Variablen, in der die JSON-Daten gespeichert werden sollen.

```
<mwf:conditions var="conditions"/>
```

mwf:calculatedValues

Tag, der die JSON-Definitionen der *Berechneten Werte* erzeugt.

Attribut	Beschreibung
var	Name der Variablen, in der die JSON-Daten gespeichert werden sollen.

```
<mwf:calculatedValues var="calculatedValues"/>
```

mwf:markdown

Tag, der den im Body übergebenen Text nach Markdown-Auszeichnungen durchsucht und diese in HTML umwandelt.

Attribut	Beschreibung
inline	Legt fest, ob das Ausgabe-HTML auf Inline Elemente beschränkt werden soll. (optional) **Standardwert:** *false*

```
<mwf:markdown inline="false">**Dieser Text wird
    fettgedruckt ausgegeben.**</mwf:markdown>
```

mwf:hasGlobalBindErrors

Mit diesem Tag kann überprüft werden, ob bei der Validierung des im *name*-Attribut angegebenen Command-Beans globale Fehler ermittelt wurden. Der innerhalb des Tags enthaltene HTML-Code wird nur ausgegeben, wenn die Validierung globale Fehler ergeben hat. Die ermittelten Fehler werden in der Variable *errors* vom Typ *org.springframework.validation.Errors* im Request-Scope für die weitere Verarbeitung zur Verfügung gestellt.

Attribut	Beschreibung
name	Name des Command-Beans

```
<mwf:hasGlobalBindErrors name="command${self.uid}">
```

```
    <ul>
        <li class="mwf-error">
            <form:errors cssClass="mwf-error" element="p" />
        </li>
    </ul>
</mwf:hasGlobalBindErrors>
```

mwf:vars

Tag, der den im Body übergebenen Text nach Variablen der Form *${name-der-variable}* durchsucht, und diese durch ihre entsprechenden Werte ersetzt.

Attribut	Beschreibung
map	Map mit den Variablenwerten (Key, Value). **Hinweis:** Im Page-Scope wird standardmäßig eine Map mit den Formularvariablen (*formVariables*) und eine Map mit den Formularwerten (*formdata*) übergeben.
var	Name der Page-Scope Variablen, in der der gefilterte Body-Text gespeichert werden soll.

```
<fc:vars map="${formdata}">
  <fc:markdown><c:out value="${action.properties['note']}"/></fc:markdown>
</fc:vars>
```

Durch die Angabe dynamischer Tag-Attribute haben Sie die Möglichkeit, die Liste der Variablen zu erweitern.

```
<fc:vars map="${formdata}" my-variable="${any-page-scope-variable}">
  <p>${my-variable}</p>
</fc:vars>
```

mwf:url

Tag, mit dem absolute URLs auf Formcentric-Controller erzeugt werden können.

Wenn im Attribut *baseUrl* nichts angegeben ist, verweist die erzeugte URL auf die Formcentric Webapp, in der das JSP-Template aufgerufen wurde.

Dieser Tag ist hilfreich, wenn Sie die Formcentric Webapplikation auf einem anderen Host als die umgebene Webseite installieren möchten, da Sie in diesem Fall keine relativen URLs verwenden können.

Attribut	Beschreibung
value	Angabe des Formcentric Controllers
var	Name der Variablen, in der die erzeugte URL gespeichert werden soll.

Attribut	Beschreibung
scope	Scope, in dem die erzeugte URL gespeichert wird. Wenn Sie hier nichts angeben, wird der Page-Scope verwendet.
context	Namen einer lokalen Webapplikation

```
<fc:url value="/servlet/rest" var="restUrl">
    <fc:param name="_service" value="${self.properties['datasource']}"/>
    <fc:param name="_uid" value="${form.uid}"/>
    <fc:param name="_input" value="${self.name}"/>
    <fcs:xsrfTokenParam />
</fc:url>
```

Taglib web-security-1.0

Für die Erzeugung und Ausgabe von XSRF-Token steht Ihnen eine weitere Tag-Library zur Verfügung. Um die Taglib verwenden zu können, fügen Sie folgende Anweisung in die JSP-Templates ein:

```
<%@ taglib prefix="fcs"
    uri="http://www.formcentric.com/web-security-1.0"%>
```

Nachfolgend finden Sie die in der Library enthaltenen Tags beschrieben.

fcs:xsrfToken

Tag, der ein Hidden-Field mit einem XSRF-Token erzeugt.

Attribut	Beschreibung
formId	ID des Formulars, für den der XSRF-Token erzeugt werden soll. Wenn hier nichts angegeben wird, so wird die ID des im Request-Attribut *from* übergebenen Formulars verwendet.
varName	Name der Variablen, in der der XSRF-Token gespeichert werden soll. Wenn dieses Attribut gefüllt ist, wird kein Hidden-Field erzeugt.

```
<fcs:xsrfToken/>
```

fcs:xsrfTokenParam

Dieser Tag kann in Verbindung mit dem *<c:url>*-Tag verwendet werden, um der URL einen XSRF-Token als Parameter hinzuzufügen.

Attribut	Beschreibung
formId	ID des Formulars, für den der XSRF-Token erzeugt werden soll. Wenn hier nichts angegeben wird, so wird die ID des im Request-Attribut *from* übergebenen Formulars verwendet.

Attribut	Beschreibung
method	Angabe der Übertragungsmethode (GET, POST) mit der die zugehörige URL aufgerufen wird. **Standardwert:** *GET*

```
<c:url value="/servlet/upload" var="uploadUrl">
   <c:param name="_uid" value="${form.uid}"/>
   <c:param name="_lang" value="${form.lang}"/>
   <fcs:xsrfTokenParam method="POST"/>
</c:url>
```

6.5.10. JavaScript

Formcentric beinhaltet standardmäßig die nachfolgend beschriebenen JavaScript-Dateien. Diese sind im Entwicklungs-Workspace sowohl im Verzeichnis */formcentric-webapp-customizations/src/main/webapp/js*, als auch in der Export-Datei */formcentric-module-customizations/resources_export.zip* abgelegt.

jQuery-File-Upload

Für den Upload von Dateien verwendet Formcentric das Blueimp jQuery-File-Upload Plugin. Je nach verwendetem Browser werden die Dateien per AJAX oder in einem versteckten Iframe übertragen. Das Plugin umfasst folgende JavaScript-Dateien.

- jquery_ui_widget_1_13_2.js
- jquery_iframe_transport.js
- jquery_xdr_transport.js
- load-image-all-min.js
- canvas-to-blob.min.js
- jquery_fileupload_10_31_0.js
- jquery_fileupload_process_10_31_0.js
- jquery_fileupload_image_10_31_0.js

Da die JavaScripte teilweise voneinander abhängen, müssen sie in der hier angegebenen Reihenfolge geladen werden.

jquery-autocomplete.js

Dieses JavaScript enthält ein JQuery-Plugin, mit dem Eingabefelder mit einer Vorschlagsfunktion ausgestattet werden können. Die Vorschlagswerte werden asynchron von dem angegebenen REST-Service geladen.

jquery-format-1.3.js

Dieses JavaScript enthält ein jQuery-Plugin, das die Formatierung und Analyse von Daten und Zahlen ermöglicht. Dabei handelt es sich um eine JavaScript-Alternative der Java-Klassen *SimpleDateFormat* und *NumberFormat*.

json2.js

Formcentric verwendet das native *JSON*-Objekt moderner Browser, um JSON-Daten zu parsen und zu schreiben. Bei älteren Browsern, die das *JSON*-Objekt nicht unterstützen, wird das Objekt durch dieses JavaScript bereitgestellt.

Select2

Formcentric nutzt das jQuery-Plugin Select2, um ein anpassbares Auswahlfeld zu bieten, das Funktionen wie Suche, Markierung, benutzerdefinierte Optionen und viele weitere unterstützt. Nachdem das Plugin vom Browser geladen wurde, erscheinen alle Auswahllisten automatisch als Select2-Auswahllisten. Select2 bietet zahlreiche spezifische Konfigurationsparameter, die Sie im JSP- oder Freemarker-Template als JSON-Objekt im Data-Attribut *data-mwf-select* innerhalb des *<form:select>*-Tags angegeben können. Weitere Informationen zum Select2-Plugin finden Sie unter https://select2.org.

```
<form:select id="${self.id}"

    <!-- Select2 config -->
    data-mwf-select='{
        "placeholder": "${placeholder}",
        "width": "100%",
        "tags": "${customInput}"
    }'

    ...
```

jquery-formcentric-1.9.js

Dieses JavaScript enthält ein jQuery-Plugin, das die von Formcentric benötigten JavaScript-Funktionen bereitstellt. Eine Instanz des Plugins können Sie wie nachfolgend dargestellt im JSP-Template *webforms.jsp* erzeugen.

```
<mwf:calculatedValues var="calculatedValues"/>
<mwf:conditions var="conditions"/>

<script type="text/javascript">
    jQuery('#command${self.uid}').webforms({
        trackingUrl: "${trackingUrl}",
        calculatedValues: ${calculatedValues},
        conditions: ${conditions}
    });
</script>
```

Beim Aufruf der Plugin-Methode *webforms* haben Sie die Möglichkeit, ein Konfigurationsobjekt zu übergeben, das die nachfolgenden Optionen enthalten kann.

Option	**Beschreibung**
conditions	Typ: *JSON* JSON-Definition der clientseitig auszuwertenden Bedingungen.
calculatedValues	Typ: *JSON* JSON-Definition der *Berechneten Werte* die clientseitig berechnet werden.
appendUrlVars	Typ: *Boolean* Mit diesem Parameter legen Sie fest, dass die URL-Parameter der umgebenden Seite in den AJAX-Request an den Formular-Controller übernommen werden.
trimSpaces	Typ: *Boolean* Mit diesem Parameter legen Sie fest, dass führende und nachfolgende Leerzeichen aus den Formulardaten entfernt werden.
createOption	Typ: *Function($form, entry, selected)* Funktion, die ein Option-Element einer dynamischen Auswahlliste (*comboBox*) erzeugt.
createRadio	Typ: *Function($form, name, entry, checked)* Funktion, die einen Radio-Button einer dynamischen Einfachauswahl (*radioGroup*) erzeugt.
createCheckBox	Typ: *Function($form, name, entry, checked)* Funktion, die einen Checkbox-Button einer dynamischen Mehrfachauswahl (*checkBoxGroup*) erzeugt.
createUploadFileRow	Typ: *Function(Object $form, Object attr, file)* Diese Funktion erzeugt einen neuen Eintrag in der Dateiliste des FileUpload-Elements bevor die Datei hochgeladen wurde.
createDownloadFileRow	Typ: *Function(Object $form, Object attr, file)* Diese Funktion erzeugt einen neuen Eintrag in der Dateiliste des FileUpload-Elements nachdem die Datei hochgeladen wurde.
updateCalculatedValue	Typ: *Function($form, id, value)* Funktion, die die Darstellung eines *calculatedValues* aktualisiert, wenn dieser neu berechnet wurde.

Option	Beschreibung
updateFormValue	Typ: *Function($form, $elem, name, l)* Diese Funktion aktualisiert den Wert eines Formularfelds in der Zusammenfassung, wenn dieser vom Anwender eingegeben oder verändert wurde. Wenn es sich bei dem Formularfeld um eine Auswahl handelt, werden die Labels der ausgewählten Optionen im Parameter *l* übergeben. Anderenfalls wird der eingegebene Text übergeben.
onFillDropdown	Typ: *Function(Object $form, Object $elem))* Callback-Funktion, die aufgerufen wird, nachdem eine dynamsiche Dropdown-Liste gefüllt wurde.
onFillSelection	Typ: *Function(Object $orm, Object $elem))* Callback-Funktion, die aufgerufen wird, nachdem eine dynamsiche Auswahlliste gefüllt wurde.
onInit	Typ: *Function(Object $form)* Callback-Funktion, die aufgerufen wird, nachdem das jQuery-Plugin initialisiert wurde. Nutzen Sie diese Funktion, um eigene Initialisierungen durchzuführen.
onSubmit	Typ: *Function(Object $form, String url, String query)* Callback-Funktion, die aufgerufen wird, wenn das Formular abgesendet wird. Nur wenn die Funktion den boolschen Wert *true* zurückgibt, wird das Formular abgesendet.
onSuccess	Typ: *Function(Object $form, Object data, String status, Object jqXHR)* Callback-Funktion, die aufgerufen wird, nachdem das Formular erfolgreich abgesendet wurde.
onAjaxError	Typ: *Function(Object jqXHR, String status, String error)* Funktion, die aufgerufen wird, wenn bei einem AJAX-Request ein Fehler aufgetreten ist. Standardmäßig erzeugt diese Funktion einen Eintrag im Fehlerprotokoll des Browsers.
onRedirect	Typ: *Function(Object $form, Object data, String status, Object jqXHR)* Die *onRedirect*-Funktion ist eine Callback-Funktion, die aufgerufen wird, wenn die Serverantwort beim Absenden des Formulars den Header *X-Redirect-Location-Header* enthält.

Option	Beschreibung
	Wird von der *onRedirect*-Funktion der boolsche Wert *true* zurückgegeben, signalisiert dies, dass der Redirect bereits innerhalb der *onRedirect*-Callback-Funktion verarbeitet wurde. Gibt die Funktion hingegen *false* zurück oder erfolgt keine Rückgabe, wird der Redirect automatisch an die in der Serverantwort spezifizierte URL (X-Redirect-Location-Header) durchgeführt.
operations.visible	Type: *Function(Object $form, Object field)* Funktion, mit der Eingabefelder eingeblendet werden.
operations.hidden	Type: *Function(Object $form, Object field)* Funktion, mit der Eingabefelder ausgeblendet werden.
operations.writeable	Type: *Function(Object $form, Object field)* Funktion, mit der Eingabefelder von schreibgeschützt auf änderbar gesetzt werden.
operations.readonly	Type: *Function(Object $form, Object field)* Funktion, mit der Eingabefelder von änderbar auf schreibgeschützt gesetzt werden.
operations.enabled	Type: *Function(Object $form, Object field)* Funktion, mit der Eingabefelder von deaktiviert auf aktiviert gesetzt werden.
operations.disabled	Type: *Function(Object $form, Object field)* Funktion, mit der Eingabefelder von aktiviert auf deaktiviert gesetzt werden.
operations.optional	Type: *Function(Object $form, Object field)* Funktion, mit der Eingabefelder als optional gekennzeichnet werden.
operations.mandatory	Type: *Function(Object $form, Object field)* Funktion, mit der Eingabefelder als Pflichtfelder gekennzeichnet werden.

Die Standardimplementierungen der aufgeführten JavaScript-Funktionen finden Sie im JavaScript *jquery-webforms-1.9.js*.

Im folgenden Beispiel werden die Funktion *operations.mandatory* durch eine angepasste Version ersetzt.

```
<fc:calculatedValues var="calculatedValues"/>
<fc:conditions var="conditions"/>

<script type="text/javascript">
```

```
    $('#command${form.uid}').webforms({
        "calculatedValues" : ${calculatedValues},
        "conditions" : ${conditions},
        "operations": {
            "mandatory": function ($form, field) {
                var $label = $form.find('label[for="' + field.input + '"]');
                $label.children('em').remove();
                $label.append('<em>*</em>', '');
            }
        }
    });
</script>
```

Event-Referenz

Das Formcentric jQuery-Plugin stellt eine Reihe von Events zur Verfügung, die es Ihnen ermöglichen, auf bestimmte Ereignisse zu reagieren. Die zugehörigen Event-Handler müssen auf dem *document* Object registriert werden.

Event-abhängige Detailinformationen wie beispielsweise das zugehörige Formularelement werden im Event-Object *event.details* an den Event-Handler übergeben.

```
document.addEventListener("mwf-fill-selection",
    function(event) {
        console.log(event.detail.$form);
        console.log(event.detail.$elem);
    }
);
```

Die nachfolgende Tabelle beschreibt die Ereignisse, auf die Sie hören und programmgesteuert reagieren können.

Event-Name	Detailinformationen	Wird abgesendet wenn
mwf-ajax-finished	*$dest, $content*	die Funktion *mwfAjaxReplace* erfolgreich ausgeführt wurde.
mwf-ajax-error	*$dest, jqXHR, textStatus, errorThrown*	der asynchrone Aufruf (AJAX-Call) fehlschlägt.
mwf-fill-dropdown	$form, $elem	eine Auswahlliste durch eine Datenquelle befüllt wurde.
mwf-fill-selection	$form, $elem	eine Einfach- oder Mehrfachauswahl durch eine Datenquelle befüllt wurde.
mwf-fill-hidden	$form, $elem	ein verstecktes Feld durch eine Datenquelle befüllt wurde.
mwf-suggestion-selected	$form, $elem, id, selection, params	bei einem Eingabefeld ein Vorschlag ausgewählt wurde.
mwf-value-changed	$form, $elem, name, value	sich der Wert eines Formularfelds ändert.

6.6. Erweiterung der Headless-Webapplikation

Bei der Formcentric Headless-Webanwendung handelt es sich um eine Spring-Boot Applikation, die eine REST-Schnittstelle mit verschiedenen Endpunkten für die Formularverarbeitung bereitstellt. Für die clientseitige Anbindung der Headless-Anwendung steht Ihnen ein vorgefertigter React-Client zur Verfügung, den Sie ebenfalls an Ihre Anforderungen anpassen können (siehe auch Abschnitt 6.7, „Formcentric-Client“).

Der folgende Abschnitt beschreibt, wie die Headless-Anwendung funktional erweitert werden kann. Bitte beachten Sie, dass die Namen der Framework-Klassen teilweise mit denen der Spring-MVC-Webanwendung übereinstimmen, aber unterhalb des Package *com.formcentric.headless.rest* liegen.

6.6.1. Implementierung einer Action

Analog zur Spring-MVC-Webanwendung verwendet auch die Headless-Anwendung Actions, in denen die Business-Logik der Formulardatenverarbeitung gekapselt ist. Diese Klassen implementieren das Interface *com.formcentric.headless.actions.Action*. Durch die Entwicklung einer Custom-Action sind Sie in der Lage beliebige Backend-Systeme anzubinden. Da es sich bei den Actions um Spring-Beans handelt, können Konfigurationsparameter über standard Spring-Mechanismen an die Action übergeben werden. Das nachfolgende Beispiel zeigt Ihnen, wie Sie eine *CustomAction* implementieren und konfigurieren können.

```
import com.formcentric.headless.actions.*;

public class CustomAction extends BaseAction {

  public static final String PROP_CUSTOM = "anyCustomActionPropertyName";

  @Override
  public ActionResult execute(ExecutionContext context, Map<String,
     Object> formData) throws ActionException {

    WebForm formDefinition = context.getFormDefinition();
    ActionNode action = context.getAction();

    String customParam = action.getPropertyAsString(PROP_CUSTOM);

      // Business-Logic
      ...

    ActionResult actionResult = new ActionResult();
    actionResult.setView("success");
    return actionResult;
  }

  @Override
  public boolean isExecutable(ExecutionContext context,
     Map<String, Object> formValues) throws ActionException;
```

```
    return true;
  }

  public String name() {
    return "customAction";
  }
}
```

Um eigene Actions in der Headless-Anwendung zu instanziieren und zu konfigurieren, erstellen Sie eine Konfigurationsklasse, wie im nachfolgenden Code-Beispie dargestellt. Markieren Sie die Konfigurationsklasse mit der Annotation *@Configuration*. In Ihrer Konfigurationsklasse definieren Sie eine Methode, die ein Objekt Ihrer Action-Klasse instanziiert. Diese Methode muss mit *@Bean* annotiert werden, um kenntlich zu machen, dass diese Methode eine Bean-Definition bereitstellt.

Falls Ihre Action-Klasse Abhängigkeiten zu anderen Beans hat, injizieren Sie diese über die Methodenparameter. Nutzen Sie die Annotation *@Value*, um Konfigurationswerte zu injizieren, oder übergeben Sie andere Beans direkt als Parameter.

```
package com.example.myapp.config;

import com.example.myapp.actions.MyCustomAction;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class MyActionsConfiguration {

    @Bean
    public MyCustomAction myCustomAction() {
        // Configuration and initialization of the MyCustomAction
        return new MyCustomAction();
    }

    // Additional bean definitions can be added here
}
```

Wenn Sie Ihre Konfigurationsklasse im Basis-Package com.formcentric.headless oder einem Unter-Package platzieren, wird Spring Boot sie automatisch finden und laden. Sollten Sie davon abweichende Package-Namen verwenden, müssen Sie den Parameter scanBasePackage der *@SpringBootApplication*-Annotation in der Hauptanwendungsklasse erweitern.

Im Entwicklungs-Workspace ist bereits die Anwendungsklasse *CustomHeadlessWebApplication* angelegt, die Sie für diesen Zweck anpassen können.

6.6.2. Variable zur Vorbelegung von Formularfeldern hinzufügen

Neben den vordefinierten Variablen haben Sie auch die Möglichkeit eigene Variablen zur Vorbelegung von Formularfeldern hinzuzufügen. Registrieren Sie hierzu ein Spring-Bean vom Typ *VariablesService* im Application-Context der Headless-Anwendung.

```
import com.formcentric.headless.services.VariablesService;
import com.formcentric.headless.model.WebForm;
import org.springframework.stereotype.Service;
import jakarta.servlet.http.HttpServletRequest;

@Service
public class CustomVariablesService implements VariablesService {
  @Override
  public final Map<String, Object> getVariables(HttpServletRequest request,
      WebForm formDefinition) {

    Map<String, Object> vars = new HashMap<>();

    // Add custom variables to the variables Map
    vars.put("custom_var", "custom_value");
    return vars;
  }
}
```

Da es sich bei dem *VariablesService* um ein Spring-Bean handelt, können Sie bei der Erzeugung der Variablen auch auf externe System oder Dienste zugreifen.

In manchen Anwendungsszenarien ist es erforderlich, Formularfelder mit Werten aus der Webseite oder Client-Anwendung vorzubelegen, in die das Formular eingebettet ist. Für diesen Anwendungsfall ist es ausreichend, die Variablen im Data-Attribut *data-fc-vars* des *div*-Tags anzugeben, an den das Formular gebunden ist.

```
<div
    data-fc-id="1249010"
    ...
    data-fc-vars='{"custom_var":"custom_value"}'
></div>
```

6.6.3. Implementierung eines REST-Services

Die in Abschnitt Abschnitt 6.5.8, „Implementierung eines REST-Services“ beschriebenen REST-Services stehen Ihnen grundsätzlich auch bei Einsatz der Headless-Anwendung zur Verfügung. Das nachfolgende Beispiel zeigt Ihnen einen REST-Service, der eine Map mit statischen Key/Value-Paaren erzeugt.

```
package com.formcentric.headless.examples.rest;

import com.formcentric.headless.rest.BaseRestService;
import com.formcentric.headless.rest.ServiceContext;
import org.jetbrains.annotations.NotNull;
import org.springframework.stereotype.Component;

import java.util.HashMap;
import java.util.Map;

@Component
public class CustomRestService extends BaseRestService {
```

```
  @Override
  public Object invoke(ServiceContext context, Map<String, Object> formData) {

    HashMap<String, String> data = new HashMap<>();

    // fill the map
    data.put("key1", "value1");
    data.put("key2", "value2");
    data.put("key3", "value3");

    return data;
  }

  @NotNull
  @Override
  public String name() {
    return "customRestService";
  }
}
```

Bei Aufruf der *invoke*-Methode werden dem RestService neben dem ServiceContext auch die aktuellen Benutzereingaben (Parameter *formData*) übergeben. Damit sind Sie in der Lage, direkt auf die Eingaben des Benutzers zu reagieren.

Über den *ServiceContext* haben Sie zudem Zugriff auf die Formulardefinition, das Eingabeelement, die Konfigurationsparameter des RestService und das Request-Object.

Alle Formularelemente, zu denen auch die REST-Services gehören, müssen über einen eindeutigen Namen verfügen, mit dem Sie innerhalb der Formulardefinition referenziert werden können. Der Name wird beim Start der Anwendung durch Aufruf der Methode *name()* ermittelt.

Durch die *@Component*-Annotation wird Ihr REST-Service automatisch durch Spring instanziiert und unter dem festgelegten Namen registriert.

6.7. Formcentric-Client

Zur Darstellung von Formcentric-Formularen im Browser ist das NPM-Modul *@formcentric/client* (https://www.npmjs.com/package/@formcentric/client) erforderlich. Dies gilt sowohl für Projekte, die auf reinem HTML und JavaScript basieren als auch für Projekte, die Frontend-Frameworks oder Frontend-Libraries verwenden.

Das installierte Package enthält verschiedene Varianten von Modulen für unterschiedliche Anwendungen. Die Installation der benötigten Dateien erfolgt über NPM, wobei dieses keine eigenen Abhängigkeiten aufweist und auch ohne Bundler verwendet werden kann.

Für die Installation führen Sie folgendes Kommando aus:

```
npm install @formcentric/client
```

oder alternativ

```
pnpm install @formcentric/client
```

Damit ein Formular korrekt angezeigt werden kann, müssen folgende Dinge vorhanden sein:

1. Ein *div*-Tag mit einem *fc-id* Data-Attribut, in das das Formular gerendert werden soll.

2. Ein geladenes Theme, bestehend aus CSS, Templates und CSS-Custom-Properties, sofern diese in der CSS-Datei benutzt werden.

3. Die *formapp.js* muss zugänglich sein, um als Script-Tag eingebunden zu werden.

4. *formcentric_component_style.css* muss zugänglich sein, um als Link-Tag eingebunden zu werden, sofern interne Komponenten wie Datepicker oder FileUploader verwendet werden.

```
<head>
    {...}
    <link rel="stylesheet" href="/example-url/formcentric_component_style.css"/>
</head>
<div
    data-fc-id="1249010"
    data-fc-formapp-url="/example-url/formapp.js"
    data-fc-theme-url="/example-url/formcentric.css"
    data-fc-template-url="/example-url/formcentric_templates.js"
    data-fc-theme-variable-url="/example-url/formcentric.json"
    data-fc-form-definition="K82AClxH1YpNGtKt ... ffUuAm4OyEQsC9"
    data-fc-refs="ffUuAm4OyEQsC9 ... 2AClxH1YpNGtKt"
    data-fc-vars='{}'
    data-fc-params='{"}'
    data-fc-data-url='https://example-url-to-formcentric-headless-server.com'
></div>
```

```
<script
    src="./formcentric.js"
    defer
></script>
```

6.7.1. Theme

Um sicherzustellen, dass das Formular korrekt angezeigt wird, muss das Theme-CSS geladen werden. Dies kann durch die Verwendung eines Link-Tags im HTML-Head und die Verwendung von Custom-Properties erreicht werden. Wenn vorhanden, müssen diese im HTML-Code gesetzt sein.

Jedes Eingabefeld hat sein eigenes Template, das angepasst werden kann. Diese Templates werden in einer JS-Datei namens *formcentric_templates.js* auf dem *Window*-Objekt definiert. Dadurch werden sie später beim Rendern des Formulars

gefunden. Die Templates sind zwingend erforderlich, um das Formular korrekt darzustellen (siehe Abschnitt 6.7.3, „Templates“).

6.7.2. Initialisierung

Um den Client zu starten, kann entweder dass Skript *formcentric.js* geladen werden, oder nachdem dieses geladen wurde zu einem späteren Zeitpunkt *window.formcentric.initFormcentric()* aufgerufen werden. Über das Data-Attribut *fc-data-url* können Sie die URL für den Zugriff auf den Formcentric Headless Server konfigurieren.

6.7.3. Templates

Template bestehen immer aus einer Funktion, deren Rückgabewert vom Formcentric-Client für das Rendern von HTML-Code verwendet wird. Hierbei übergibt der Formcentric-Client zwei Parameter (html und props) an eine Template-Funktion. Die genaue Struktur und die verwendeten Parameter hängen von der spezifischen Verwendung des Templates ab.

html: Ein Template-Literal-Tag, welcher verwendet wird, um HTML-Code zu rendern. Dieser Parameter ermöglicht das Einbetten von HTML im JavaScript-Code des Templates.

props: Ein Objekt, das die Eigenschaften des Formularfelds enthält. Diese bestehen aus vom Formcentric-Client berechneten Werten und aus über den Editor eingestellten Felddaten. Die spezifischen Eigenschaften variieren je nach Formularfeld.

Das fertige HTML wird durch die Kombination von statischem HTML-Code und den Werten der *props*-Eigenschaften erstellt. Dies kann durch Zusammenführung von Strings oder Verwendung von Funktionen zum Rendern von HTML erfolgen. Das hieraus entstehende HTML wird dann als Rückgabewert der Template-Funktion an den Formcentric-Client zurückgegeben, welcher es dann in den DOM rendert.

Alle Template Funktionen können auch asynchron als Promise vom Formcentric-Client verarbeitet werden.

Template Properties

Template Properties werden vom Formcentric-Client als Parameter (*props*) an die entsprechende Template-Funktion übergeben. Diese enthalten alle nötigen Informationen für die Anzeige und das Verhalten der jeweiligen Formularfelder.

Folgende Properties werden an die Templates übergeben:

Property	**Beschreibung**
key	Die eindeutige ID des Elements
oninput	(event: InputEvent) => void

Property	Beschreibung
	Aktualisiert den Wert des Felds
onfocus	(event: FocusEvent) => void Gibt gegebenenfalls vorhandene Validierungsfehler des Felds zurück
onclick	Eine Funktion, die über den Editor definierte *onClick* Funktionen evaluiert
fieldSuccess	Ein boolescher Wert, der angibt, ob das Element erfolgreich validiert wurde und keine Fehler enthält.
fieldError	Ein Objekt, das Informationen über einen Fehler im Element enthält.
properties	Ein Objekt, das die Eigenschaften des Feld-Elements enthält
components	Ein Objekt, das Komponenten für bestimmte Feldtypen enthält. Diese Komponenten werden verwendet, um das Element im Template zu rendern. Es enthält Komponenten für *captcha, fileUploader, comboBox, suggestions, hint, datePicker und markdown*
fieldSetFields	Ein Array, welcher die Felder eines *FieldSets* enthält
layoutFields	Ein Array, welcher die Felder eines *Layouts* enthält
summaryFields	Ein Array welches die Informationen von Feldern, die in einem *SummaryField* festgelegt wurden, enthält
fieldEmptyText	Ein Standardtext, der angezeigt wird, wenn ein *SummaryField* keine Werte enthält
contentMarkup	Eine Content-Komponente, welche von einer über das *form*-Div festgelegten Funktion zurückgegeben wird
hasService	Ein boolescher Wert, der angibt, ob das Element einen REST-Service hat
setRESTParams	(params: Record<string,any>) => void: Eine Funktion mit der die Parameter für den REST-Service des Elements festgelegt werden können.

Element: In den Properties sind außerdem alle Informationen über das darzustellende Element enhalten.

```
interface fcElement {

id: string // ID des Elements

name: string // Technischer Name des Elements
```

```
type: fcFieldTypes // Elementtyp

fieldSetId?: string // ID des FieldSets, in dem das Element enthalten ist

layoutId?: string // ID des Layout Elements, in dem das Element enthalten ist

label?: string // Label des Elements

value?: string | string[] // Wert des Elements

validators?: fcElementValidator[] // Validatoren

children?: {

    id: string

    type: fcFieldTypes

    name: string

    label?: string

    value?: string | string[]

    checked?: boolean

    properties?: fcProperties

    validators?: fcElementValidator[]

}[]

properties?: fcProperties // Properties des Elements (siehe Properties)

}
```

Feldtypen:

```
type fcFieldTypes =

| 'error'

| 'success'

| 'formHeader'

| 'formFooter'

| 'inputField'

| 'button'

| 'form'

| 'layout'
```

```
| 'condition'

| 'passwordField'

| 'textArea'

| 'radioGroup'

| 'comboBox'

| 'checkBoxGroup'

| 'fileUpload'

| 'calculatedValue'

| 'hiddenField'

| 'paragraph'

| 'summary'

| 'dateField'

| 'numberField'

| 'emailField'

| 'phoneField'

| 'shortText'

| 'captcha'

| 'content'

| 'option'

| 'fieldSet'
```

Validatoren:

```
interface fcElementValidator {

    id: string

    name: string

    properties?: {

    errormessage?: string

    from?: string

    to?: string
```

```
    days_from?: string

    days_to?: string

    pattern?: string

    max_files?: string

    max_size?: string

    file_types?: string
}
```

Properties: In *props.properties* sind alle HTML Attribute und Feld-Properties aus der Formulardefinition enthalten. Diese werden vom Formcentric-Client zum Beispiel durch Bedingungen berechnet oder vom Redakteur an dem entsprechenden Formularfeld eingestellt. Die nachfolgende Tabelle zeigt eine Übersicht der möglichen Properties:

Property	**Beschreibung**
hint	Ein optionaler Hinweistext, der dem Benutzer weitere Informationen oder Anweisungen gibt.
placeholder	Ein optionaler Platzhaltertext, der in einem Eingabefeld angezeigt wird, wenn kein Wert eingegeben wurde.
selected	Ein boolescher Wert, der angibt, ob das Element standardmäßig ausgewählt ist.
errormessage	Eine optionale Fehlermeldung, die angezeigt wird, wenn das Element ungültig ist.
multiple	Ein boolescher Wert, der angibt, ob mehrere Werte für dieses Element ausgewählt werden können.
auto_upload	Ein boolescher Wert, der angibt, ob eine automatische Hochladefunktion aktiviert ist.
datasource	Eine Zeichenkette, die eine Datenquelle angibt.
datasource_params	Eine Zeichenkette, die Parameter für die Datenquelle angibt.
dynamic	Ein boolescher Wert, der angibt, ob das Element dynamisch ist und seine Eigenschaften zur Laufzeit geändert werden können.
visible	Ein boolescher Wert, der angibt, ob das Element sichtbar sein soll.
hidden	Ein boolescher Wert, der angibt, ob das Element ausgeblendet werden soll.
writable	Ein boolescher Wert, der angibt, ob das Element beschreibbar sein soll.

Property	Beschreibung
readonly	Ein boolescher Wert, der angibt, ob das Element schreibgeschützt sein soll.
optional	Ein boolescher Wert, der angibt, ob das Element optional ist.
mandatory	Ein boolescher Wert, der angibt, ob das Element obligatorisch ist.
disabled	Ein boolescher Wert, der angibt, ob das Element deaktiviert sein soll.
enabled	Ein boolescher Wert, der angibt, ob das Element aktiviert sein soll.
type	Eine Zeichenkette, die den Typ des Elements angibt.

Components

In *props.components* stehen interne Komponenten zur Auswahl, welche die Arbeit mit einzelnen Formularfeldern vereinfachen sollen, wenn kein Bedarf nach Anpassung der Funktionalität dieser Felder besteht. Folgende Komponenten sind vorhanden:

Property	Beschreibung
captcha	Lädt Captcha-Bilder vom Headless-Server
combobox	Stellt Auswahllisten dar
datePicker	Stellt einen Date Picker dar
fileUploader	Stellt einen Upload-Dialog dar
hint	Stellt Hinweise dar
markdown	Stellt Markdown- als HTML-Elemente dar
suggestions	Stellt Vorschläge von REST-Services als Auswahlliste unter Eingabefeldern dar

Folgende Properties können an die oben genannten Komponenten weitergegeben werden:

captcha:

Property	Beschreibung
buttonText	Eine optionale Zeichenfolge für den Refresh-Button der Captcha Komponente. Wird diese nicht angegeben, dann enthält der Button ein Icon.

combobox:

Property	Beschreibung
alle	Alle Properties aus dem Parameter des Templates müssen übergeben werden.

datepicker:

Property	Beschreibung
alle	Alle Properties aus dem Parameter des Templates müssen übergeben werden.

fileUploader:

Property	Beschreibung
trigger	Die Klasse oder ID des Trigger-Elements
inline	Ein optionaler boolescher Wert, der angibt, ob das Element in einer Inline-Darstellung angezeigt werden soll.

hint:

Property	Beschreibung
alle	Alle Properties aus dem Parameter des Templates müssen übergeben werden.

markdown:

Property	Beschreibung
markdown	Das *markdown* Property nimmt stringifiziertes Markdown als Wert an.

suggestions:

Property	Beschreibung
Alle	Alle Properties aus dem Parameter des Templates müssen übergeben werden.

Die Komponenten werden innerhalb des HTML-Template-Literal-Tags in den Templates ausgeführt:

```
${ props.components.captcha( {…} ) }
```

Anpassung und Erweiterung von Templates

Um die Templates zu erweitern, können Sie die HTML-Elemente und Klassen innerhalb der Vorlagen anpassen. Dies ermöglicht es Ihnen, das Erscheinungsbild und das Verhalten der Komponenten anzupassen.

Sie können Klassen hinzufügen oder entfernen, um das Styling anzupassen, oder zusätzliche HTML-Elemente einfügen, um zusätzliche Funktionalitäten bereitzustellen.

Darüber hinaus können die Templates auch asynchron ausgeführt werden, was bedeutet, dass Sie Daten abrufen und anzeigen können, die nicht direkt vom Formcentric-Client an die Templates übergeben werden. Dies eröffnet Möglichkeiten zur Integration mit APIs oder anderen externen Datenquellen.

Um asynchrone Daten in den Templates zu verwenden, können Sie JavaScript-Funktionen wie *fetch* verwenden, um Daten von einem Server abzurufen. Sie können dann die erhaltenen Daten in den Vorlagen anzeigen, indem Sie die entsprechenden Variablen oder Platzhalter verwenden.

Erweiterungsmöglichkeiten:

1. Unterstützung für eine oder mehrere benutzerdefinierte CSS-Klassen: Es können optionale CSS-Klassen hinzugefügt werden, um das Styling des Eingabeelements anzupassen. Dazu kann eine Custom-Klasse in der *className*-Definition verwendet werden:

```
input className="customClass" />
```

2. Anpassung des Markups: Es können ohne weiteres neue Markup-Elemente hinzugefügt oder bereits vorhandene Elemente angepasst werden:

```
inputField: (html, props) => html`<div className="fc-field
        ${props?.properties?.hidden ? 'fc-hiddenField' : ''}
        ${props.properties?.hint ? 'fc-field--has-hint' : ''}
        ${props?.fieldError ? 'fc-field--has-error' : ''}
        ${props?.fieldSuccess ? 'fc-field--is-valid' : ''}">

    ${customFunction(html, props)}

    <div className="fc-textinput">
        <div className="fc-textinput__input">
            <input
                id=${props.id}
                name=${props.name}
                value=${props.value}
                oninput=${props.oninput}
                onfocus=${props.onfocus}
                onblur=${props.onblur}
                type=${props.properties.type || 'text'}
                autocomplete=${props.properties?.autocomplete}
                maxlength=${props.properties?.maxlength}
                disabled=${props.properties?.disabled}
```

```
                placeholder=${props.properties?.placeholder || ''}
                readonly=${props.properties?.readonly}

                ${...customProperties}
            />

            ${props.components.suggestions(props)}
        </div>

        ${label(html, props)} ${hint(html, props)} ${error(html, props)}
    </div>
</div>
```

6.7.4. Spezielle Integrationsszenarien

In den meisten Anwendungsfällen wird das *@formcentric/client*-Skript einfach geladen und ausgeführt. Es gibt jedoch besondere Szenarien, wie etwa bei Single Page Applications (SPA), in denen das Skript erst ausgeführt werden muss, wenn der DOM-Baum vollständig aufgebaut ist. In solchen Fällen bietet sich ein dynamischer Import des Skripts an, der genau dann ausgeführt wird, wenn der virtuelle DOM vollständig aufgebaut wurde. Dieser Zeitpunkt kann je nach SPA-Framework variieren.

```
function App() {
    const ref = useRef(null);

    const formDef = "TGU5Kmxe4svPaahc2aSm-4PHzoKWWtvC ... D-ZwC6MPQRWA==";

    useEffect(() => {
        if (!ref) return;

        import("@formcentric/client/dist/formcentric");
    }, [ref]);

    return (
            <div
                ref={ref}
                data-fc-id="<<id>>"
                data-fc-form-definition={formDef}
            ></div>
    );
}
```

Falls kein dynamischer Import möglich ist, kann nach dem Laden des Skripts die Funktion *initFormcentric* des *window.formcentric*-Objekts aufgerufen werden.

```
window.formcentric.initFormcentric()
```

6.7.5. Troubleshooting

Kontrollieren Sie das Browser-Log. Wenn dort keine Ausgaben des Clients zu finden sind, wurde das Skript *formcentric.js* nicht geladen bzw. ausgeführt.

Der Hinweis, dass kein Form-*div* gefunden wurde kann zwei Ursachen haben:

1. Es wurde kein *div*-Tag mit dem Data-Attribut *fc-id* gefunden

2. Das Skript *formcentric.js* wurde ohne Angabe des *defer*-Attributs geladen.

Wenn kein Formular angezeigt wird, obwohl ein Form-*div* gefunden wurde, kann folgende Ursachen haben:

1. Es wurde kein *div*-Tag mit dem Data-Attribut *fc-id* gefunden

2. Das Skript *formapp.js* wurde nicht geladen.

Interne Komponenten wie Datepicker oder FileUploader werden ohne Styling gerendert:

1. *@formcentric/client/dist/formcentric_component_style.css*wurde nicht in die Seite eingebunden.

Anpassung des Log-Levels

Das Log-Level des Formcentric-Clients kann bei Bedarf auch um Warnungen und Informationen erweitert werden, indem folgende Einträge im Local storage der Browser-Devtools für die einbindende Seite vorgenommen werden.

Schlüssel	**Wert**
FC-logLevel-[WRAPPER]	Info
FC-logLevel-[FORMAPP]	Info