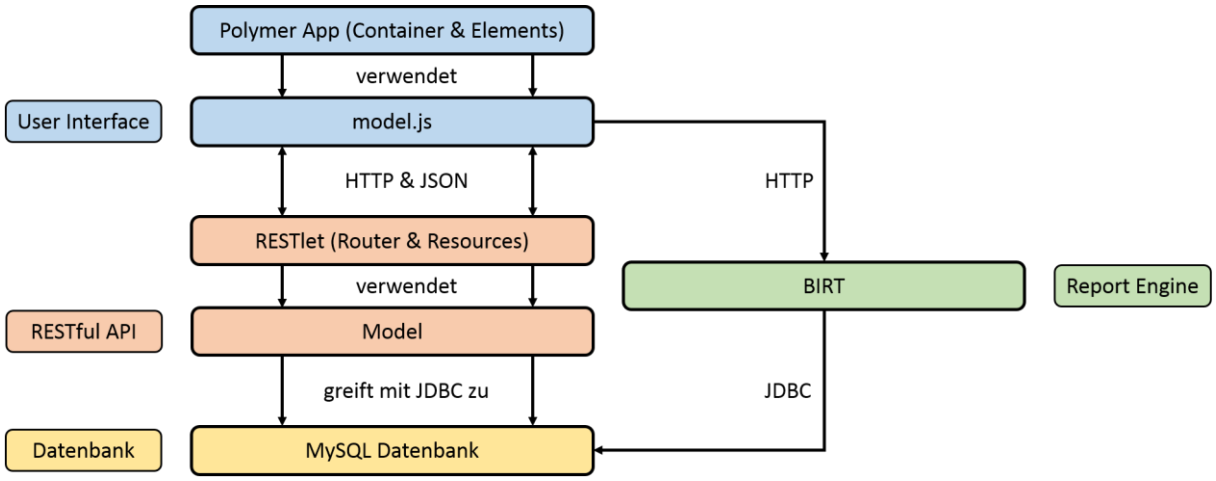
Architekturskizzierung VVS

Bildliche Darstellung:



Grundsätzliche Beschreibung:

Die Architektur des VVS ist in vier Ebenen untergliedert. Die unterste dieser Ebenen ist die Datenbank. Hier werden alle Daten, die für das VVS benötigt werden mit Hilfe einer MySQL Datenbank gespeichert.

Auf der zweiten Ebene wird eine RESTful API zur Verfügung gestellt, die auf Basis des Java Frameworks RESTlet aufgebaut ist und eine Java Web Applikation ist. Die API verwaltet die Daten der Datenbank und stellt ein Model bereit, welches die Funktionalität der einzelnen Bestandteile des VVS implementiert.

Neben der RESTful API wird auf einer dritten Ebene mit Hilfe des Java Frameworks BIRT eine Report Engine zur Verfügung gestellt, die zur Erstellung von PDF Dateien genutzt wird.

Die vierte Ebene stellt das User Interface dar, welches auf Polymer, einer Javascript Bibliothek, basiert. Dabei gibt es eine Trennung zwischen der Kommunikation mit der API Schnittstelle und der Implementierung der SPA (Single Page Application) mit ihrer Containerseite und den einzelnen Seitenelementen.

Datenbank:

Als Datenbank wird eine MySQL Datenbank ab Version 5.6.5 benötigt. Standardmäßig wird der User root mit dem Passwort root verwendet. Dies kann jedoch geändert werden, führt dann aber zu Änderungen auf der Ebene der RESTful API und der Report Engine (siehe entsprechende Abschnitte). Auf der Datenbank muss die SQL-Datei vvs.sql (Verzeichnis: /stuff/vvs.sql) importiert werden. Liegt eine SQL-Datei mit migrierten Daten vor (z.B. vvs_data.sql, im Verzeichnis /stuff/vvs_data.sql [nicht im GitHub Repository verfügbar]) kann diese ebenfalls importiert werden.

RESTful API:

Die Ebene der RESTful API basiert zum größten Teil auf dem Java Framework RESTlet und kann als Java Web Applikation in einem Java Applikationsserver deployt werden. Es wird hierbei empfohlen Tomcat in der Version 7 einzusetzen. Die Webapp sollte unter dem Kontext „VVS“ verfügbar gemacht

werden. Im Folgenden soll die weitere Architektur der RESTful API anhand der verwendeten Java Pakete erläutert werden

Database (de.dhbw.vvs.database)

Die Klassen dieses Pakets abstrahieren die Datenbankverbindung. In der Datei DatabaseConnection.java können Anpassungen der Parameter vorgenommen werden, die zur Verbindung mit der Datenbank genutzt werden (z.B. MySQL-User und -Passwort)

Model (de.dhbw.vvs.model)

In diesem Paket werden alle Klassen gespeichert, die das Modell des VVS beschreiben. Hierzu gehören zum Beispiel Studiengangsleiter, Kurs oder Dozent. Die Struktur der einzelnen Klassen wird für die Serialisierung und Deserialisierung der Objekte nach und von JSON verwendet. Für Konvertierung zwischen JSON und den Model Objekten wird die Bibliothek GSON verwendet.

Innerhalb der einzelnen Model-Klassen wird entsprechende Logik implementiert. Dazu gehören Methoden zum Anlegen, Bearbeiten und Löschen der Objekte, aber auch fachliche Logik wie die Bestimmung von zu planenden Vorlesungen oder etwa die Bestimmung der Dashboard Kennzahlen.

Utility (de.dhbw.vvs.utility)

Das Utility Paket beinhaltet nützliche Klassen, die im gesamten Projekt wiederverwendet werden. Dazu gehören zum einen vorkonfigurierte Methoden zur Verwendung der GSON Bibliothek (JSONify.java) oder verschiedene Utility Methoden zur Konvertierung von Date und Timestamp Objekten oder zur Verifizierung bestimmter logischer Datentypen, wie E-Mail Adressen und Telefonnummern.

Application (de.dhbw.vvs.application)

In diesem Paket werden zum einen die Exceptions, die innerhalb des Projekts verwendet werden verwaltet. Dabei werden im Enum ExceptionStatus.java alle Fehlertypen des Projekts aufgelistet. Diese sind auch nach außen transparent und stellen gleichzeitig die Fehlermeldungen der RESTful API dar. Innerhalb der Datei VVSApplication.java wird das Mapping zwischen der URIs der RESTful API und den Restlet Ressourcen hergestellt.

Resources (de.dhbw.vvs.resources)

In diesem Paket finden sich alle Ressourcen wieder. Jeder URI der RESTful API ist eine Ressource zugeordnet. Innerhalb der Ressource können die HTTP Methoden GET, POST, PUT, DELETE mit entsprechender Funktionalität versehen werden. Die Ressourcen stellen die Schnittstelle zwischen RESTful API und dem Model her. Eingehende HTTP Requests werden interpretiert, abgebarbeitet, indem die entsprechenden Methoden der Model Objekte aufgerufen werden, und aus den Resultaten wird ein entsprechender HTTP Response generiert. Alle HTTP Requests und nahezu alle HTTP Responses basieren auf JSON. Eine genaue Beschreibung der RESTful API findet sich in der Datei API Dokumentation.xlsx (Verzeichnis: /stuff/API Dokumentation.xlsx)

Report Engine:

Die Report Engine Ebene verwendet das Java Framework BIRT zur Generierung von PDF Reports. Im Verzeichnis /stuff/birt-viewer.war befindet sich die Ebene als Webapp. Im Root-Ordner dieses Archives finden sich die Report-Definitionen (Dateiendung .rptdesign). Diese werden zur Generierung der PDFs verwendet. Die .war Datei muss ebenfalls im Java Applikationsserver unter dem Kontext „birt-viewer“ deployt werden. Werden innerhalb der Datenbank andere Usernamen oder ein anderes Passwort verwendet, muss entsprechendes innerhalb der Report-Definition angepasst werden.

User Interface:

Die Dateien des User-Interfaces sind ebenfalls Bestandteil der im Abschnitt RESTful API beschriebenen Webapp (Verzeichnis: /WebContent/app). Das User Interface wurde in Javascript mit Hilfe der Bibliothek Polymer implementiert. Polymer basiert auf dem Prinzip der Webcomponents und implementiert das Designparadigma Material Design von Google. Alle Dateien der Polymer Bibliothek befinden sich im Verzeichnis /WebContent/app/polymer.

Im Verzeichnis /WebContent/app/js befindet sich die Datei model.js. Diese implementiert die Schnittstelle zur Ebene der RESTful API. Darin werden zum einen Template Objekte aufgeführt, die die gleiche Struktur wie die Model Objekte der RESTful API aufweisen. Zusätzlich werden alle Funktionen der API als Javascript Funktionen zur Verfügung gestellt.

Im Verzeichnis /WebContent/app/html befindet sich die Datei home.html. Diese ist der generelle Eintrittspunkt der VVS SPA (Single Page Application). Im Zusammenspiel mit der Datei home.js (Verzeichnis: /WebContent/app/js/home.js) kümmert sie sich um den Lebenszyklus der Applikation. Dabei ist sie in erster Linie für das URL-Hash Handling und die Anzeige der Seiten zuständig. Zusätzlich verwaltet sie die Daten, die zwischen den einzelnen Seiten ausgetauscht werden und vollzieht das User Management. Der letztendliche Login wird innerhalb der Dateien login.html (Verzeichnis: /WebContent/login.html) und login.js (Verzeichnis: /WebContent/app/js/login.js) implementiert.

Die einzelnen Seiten sind als Webcomponents implementiert und befinden sich im Verzeichnis /WebContent/app/html/elements. Der Name des Elements stimmt mit dem URL-Hash überein, der für die entsprechende Seite verwendet wird. Die einzelnen Seiten verwalten ihren Lebenszyklus vollkommen selbstständig und machen großen Gebrauch der Funktionalitäten, die Polymer zur Verfügung stellt.

Im Verzeichnis /WebContent/app/css befinden sich die einzelnen CSS Dateien, die für das Layout der Seiten und des SPA Containers verwendet werden. Die CSS Definitionen des SPA Containers werden in der Datei home.css vorgenommen. Die Datei elements.css wird innerhalb der einzelnen Seiten verwendet. Ausnahme dabei sind alle Popup Elemente. Diese verwenden die CSS Definitionen der Datei overlays.css

GitHub Repository:

Der komplette Code des VVS befindet sich in folgendem GitHub Repository: https://github.com/MarcNBecker/VVS. Alle Dateipfade sind relativ zum Ordner VVS des Repositorys angegeben. Auf der Seite des GitHub Repositories befindet sich zusätzlich eine Kurzanleitung zum Deployment des VVS.