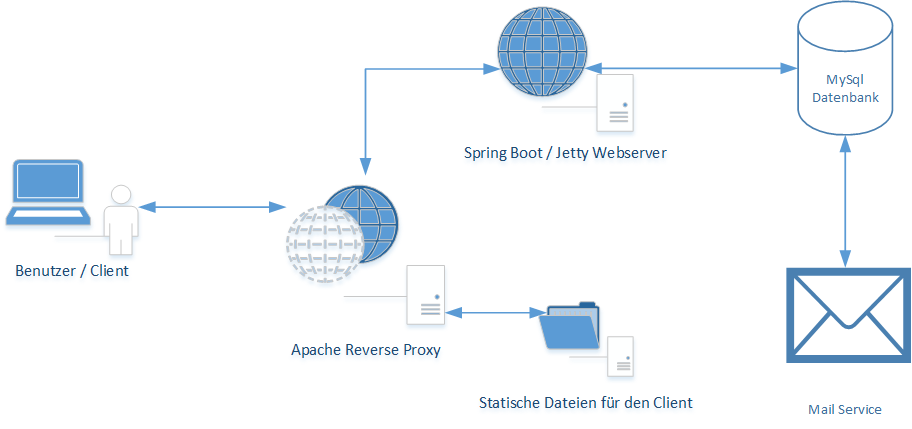
CSSR Dokumentation

# Grobarchitektur

Die Applikation besteht aus verschiedenen Teilen, welche im Folgenden dargestellt sind:



Das Grundprinzip hinter dieser doch ziemlich vielschichtigen Architektur ist simpel: Separation of Concerns. Die Architektur besteht aus mehreren kleinen, relativ unabhängigen Teilen, welche mindestens theoretisch auch gut auf mehrere Server verteilt werden könnten. Wir haben bewusst vermieden, möglichst viel in Spring Boot zu integrieren.

## Reverse Proxy

Der Reverse Proxy ermöglich das Mapping eines Ports (443) auf mehrere Applikationen, die in beliebigen Sprachen geschrieben sein können. Ausserdem hat ein Reverse Proxy den Vorteil, dass auch das Logging oder das Forcieren von TLS über mehrere Applikationen geschehen kann.

In der Praxis können selten mehrere Ports für mehrere Applikationen geöffnet werden, ein Reverse Proxy ist dann in aller Regel die einfachste Lösung.

Eine weitere denkbare Webanwendung auf dem Server wäre beispielsweise ein weiterer Webserver, der für mobile Applikationen optimiert ist und auch auf die Datenbank zugreift.

Wenn die Applikation auf Windows deployt werden müsste, so wäre es gut möglich, den Apache Reverse Proxy durch einen IIS Reverse Proxy zu ersetzen.

Als nette Nebenwirkung ist auf Grund des Reverse Proxys bei dieser Applikation kein CORS nötig.

## Spring Boot / Jetty

Hier geschieht die Business Logik. Der Server ist self-hosting, da er in Zusammenspiel mit dem Reverse Proxy möglichst einfach sein soll. Wir verwenden Jetty, weil dieser für solche Szenarien optimiert ist.

## Statische Dateien für den Client

Hierbei handelt es sich um eine Single Page Application, welche in unserem Fall direkt von Apache ausgeliefert werden. Denkbar wäre hier auch der Einsatz eines spezialisierten Webservers.

## MySql Datenbank

Wir haben uns aus mehreren Gründen dafür entschieden, eine MySql Datenbank zu verwenden:

* Als Standalone Datenbank können mehrere Applikationen auf die Datenbank zugreifen
* MySql ist gratis & Open Source
* MySql enthält alle von uns benötigten Features (CRUD, Unique Constraints, Primary Keys)

Auch wenn wir MySql verwendet haben, ist die Applikation selbst nicht direkt davon abhängig, dass das tatsächlich eine MySql Datenbank besteht, da via JPA Queries auf die Datenbank zugegriffen wird.

Auf unserem Testserver verwenden wir MariaDB, einen Fork von MySql, der mittlerweile bei Ubuntu Standard ist.

## Mail Service

Wir haben ein eigenes kleines Programm geschrieben, welches die Mails von der Datenbank holt, sendet und das Resultat speichert. Dieses Tool könnte theoretisch einfach durch eine Standardsoftware ersetzt werden, auf Grund beschränkter Ressourcen haben wir dies allerdings nicht ausprobiert.

# Qualitätssicherung

Um die Qualität des Tools zu garantieren haben wir mehrere Methoden angewendet:

* Java Linter (Checkstyle)
* Linter für TypeScript (TsLint)
* Unit Testing für den Java Webserver

Die Checkstyle Regeln basieren auf jenen von Google und erzwingen einheitliche Codestandards und Kommentare. Der TypeScript Linters ist ähnlich aufgebaut, basiert allerdings auf TsLint von Microsoft.

Das Unit Testing wurde gegen das Java Spring Boot Backend gemacht, weil dieses hauptsächlich die Logik enthält und entsprechend getestet werden kann und muss. Die Tests machen HTTP Requests zu allen Anforderungen. Die Unit Tests werden gegen eine vollfunktionale In-Memory Datenbank gemacht, und zwar gegen eine H2 Datenbank. Hier können wir davon profitieren, dass wir JPL Queries geschrieben haben und nicht native MySql Queries. MySql unterstützt leider keine In-Memory Version.

# Rest Schnittstelle

Die erwähnte Java REST Schnittstelle wurde wo immer möglich am «Stateless» Prinzip gehalten. Eine Ausnahme gibt es allerdings, dazu später mehr.

Wir haben im wesentlichen folgende Ressource und Methoden definiert:

|  |  |  |
| --- | --- | --- |
| Pfad | Methode | Beschreibung |
| /presentation | GET | Holt eine Liste aller Vorträge |
| /presentation | POST | Fügt einen neuen Vortrag ein |
| /presentation/{id} | GET | Holt einen einzelnen Vortrag |
| /presentation/{id} | PUT | Updated einen Vortrag |
| /presentation/{presentationId}/file | GET | Hole alle Dateien eines Vortrags |
| /presentation/{presentationId}/file/{id} | GET | Holt den Inhalt einer Datei oder leitet zu einem Link weiter, falls die Datei verlinkt ist |
| /presentation/{presentationId}/file/{id} | DELETE | Löscht die Datei |
| /presentation/{presentationId}/file/binary | POST | Fügt eine binare Datei hinzu |
| /presentation/{presentationId}/file/link | POST | Fügt eine verlinkte Datei hinzu |
| /presentation/{id}/invitation/template | GET | Holt das Template für eine Einladung zu einem Vortrag |
| /presentation/{id}/invitation/send | POST | Sendet eine Einladung zu einem Vortrag |
| /presentation/{presentationId}/subscription | GET | Holt alle Anmeldungen zu einem Vortrag |
| /presentation/{presentationId}/subscription/{id} | DELETE | Löscht eine Anmeldung |
| /presentation/{presentationId}/subscription/{id} | PUT | Ändert eine Anmeldung |
| /presentation/{presentationId}/subscription | POST | Fügt eine Anmeldung hinzu |
| /admin/user | POST | Fügt einen neuen Externen Benutzer hinzu |
| /admin/user | PUT | Ändert Metadaten eines externen Benutzers |
| /user | GET | Holt alle User |
| /user/me | GET | Holt den eingeloggten User |
| /user/me/tempToken | GET | Holt ein temporäres Token für den User, um eine Datei herunterladen zu können |
| /user/me/password | PUT | Setzt ein neues Passwort |
| /user/me/me/sendResetPassword | POST | Sendet ein Mail für das Passwort |
| /user/login | POST | Holt ein neues Token für den User |

Die /user/me Pfade sind genau genommen nicht wirklich Stateless, ist doch das «me» auf den aktuellen User bezogen. Allerdings ist es aus Applikationssicht nötig, solche Requests machen zu können, immerhin sind die Pfade durch das «me» sehr explizit gekennzeichnet.

Das «me» ist im Übrigen an Facebooks Graph API angelehnt, welches ebenfalls einen ähnlichen Mechanismus kennt: <https://developers.facebook.com/docs/graph-api/using-graph-api#reading> . Auch das Google Plus API kennt einen «me» Request: <https://www.googleapis.com/plus/v1/people/me>

# Sicherheit

Anstatt eines klassischen Cookies wurde ein [Json Web Token](https://jwt.io) verwendet, welches signiert ist, allerdings nicht verschlüsselt. Das JWT Token wird im localStorage gespeichert, ist also domänenspezifisch abgesichert, so dass nur Applikationen derselben Domain zugreifen können.

Weil das Token signiert ist, muss kein Datenbankrequest gemacht werden, um es zu validieren. Weil es aber nicht verschlüsselt ist, dürfen keine sensitiven Informationen darin gespeichert werden. In diesem Fall speichern wir die Emailadressen und die Tolle im Token.

Das Tool hält sich nicht komplett an den Standard, weil der JWT Header weggeschnitten wird, um die Sicherheit zu verbessern. (<https://dev.to/neilmadden/7-best-practices-for-json-web-tokens>, Tipp 4).

# Rollenkonzept

Es gibt drei verschiedene Rollen mit speziellen Berechtigungen:

* Admin: Der Admin darf alle Aktionen ausführen, insbesondere auch Benutzerrollen editieren
* Koordinator: Darf Präsentation erstellen & Benutzer hinzufügen, aber darf die Benutzer nicht einer Rolle zuordnen
* SGL: Die Studiengangleitung darf Einladungen versenden

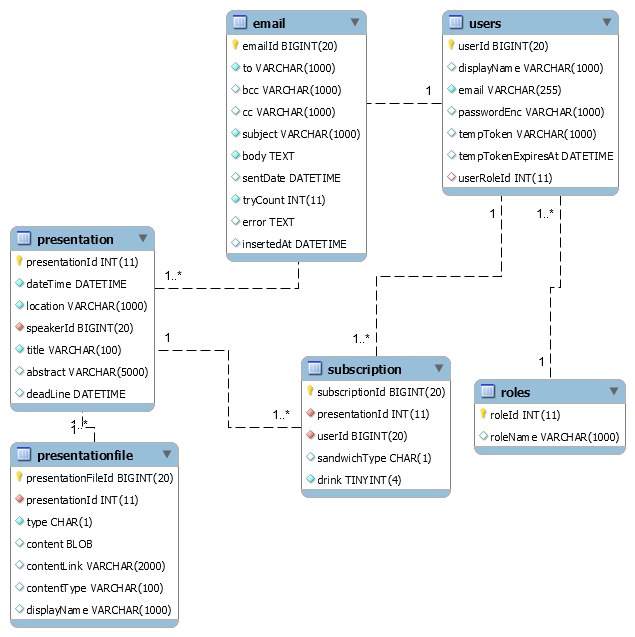
Der Referent ist technisch gesehen keine Rolle, darf aber seine Präsentationen bearbeiten.

# Client Technologien

Um eine schnelle, einfache, aber auch skalierbare Architektur auf der Clientseite zu ermöglichen, setzen wir folgende Technologien ein:

* React als GUI Framework, in Kombination mit Bootstrap
* TypeScript als Programmiersprache, damit das Typsystem bereits beim Kompilieren überprüft wird. Damit erhalten wir eine gewisse Qualität beim Quellcode und vor allem beim Refactoring sehr grosse Vorteile gegenüber reinem JavaScript.
* CSS verwenden wir auf Grund der kleinen Grösse der Applikation ohne Preprocessor
* Webpack bündelt die Applikation in eine einzige Datei, was für diese Grösse der Applikation absolut gerechtfertigt ist.

# Datenbankmodell



# Java Klassendiagramme

## Grundsätzliches

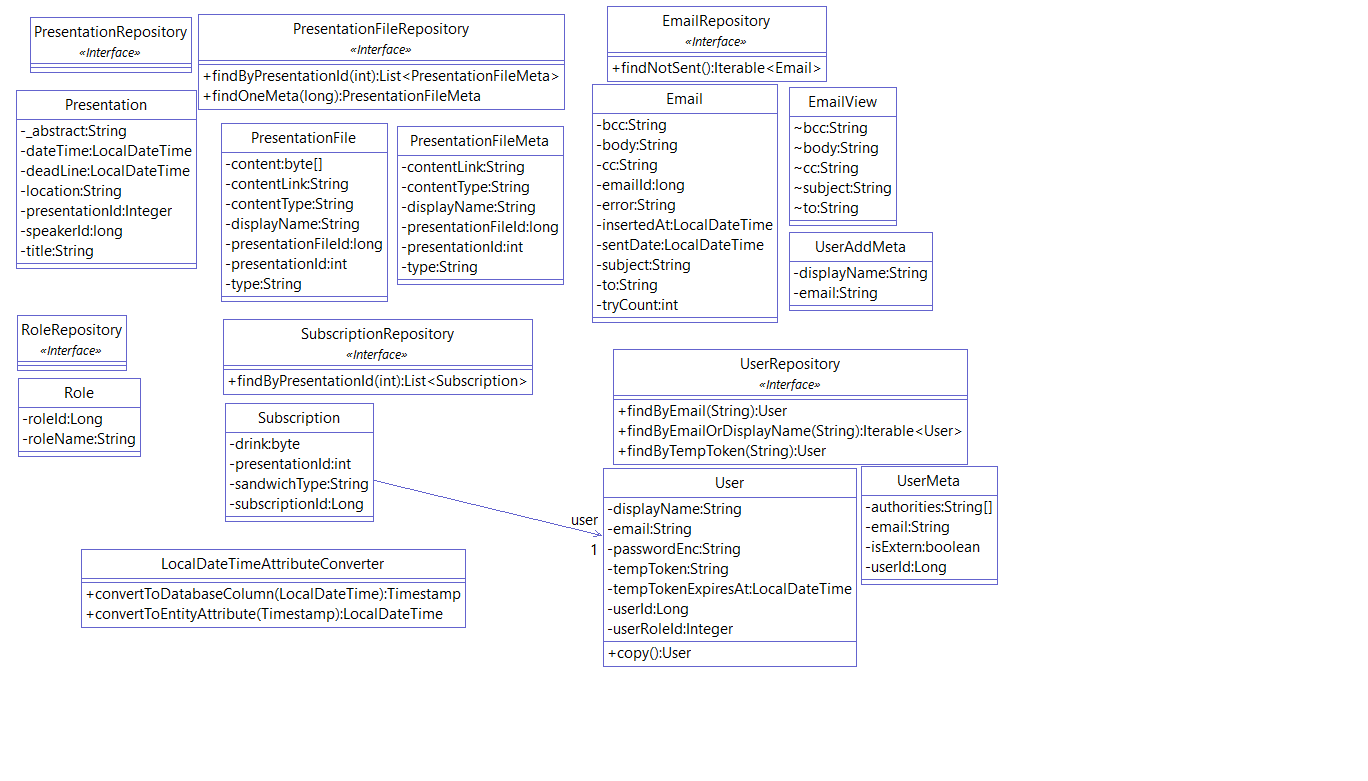
Der Java Code ist in drei Maven Module aufgeteilt:

* domain
* mailer
* webserver

Die Klassendiagramme sind jeweils für ein Modul.

## domain

Hier findet sich der Datenzugriff, also hauptsächlich die JPA Klassen:



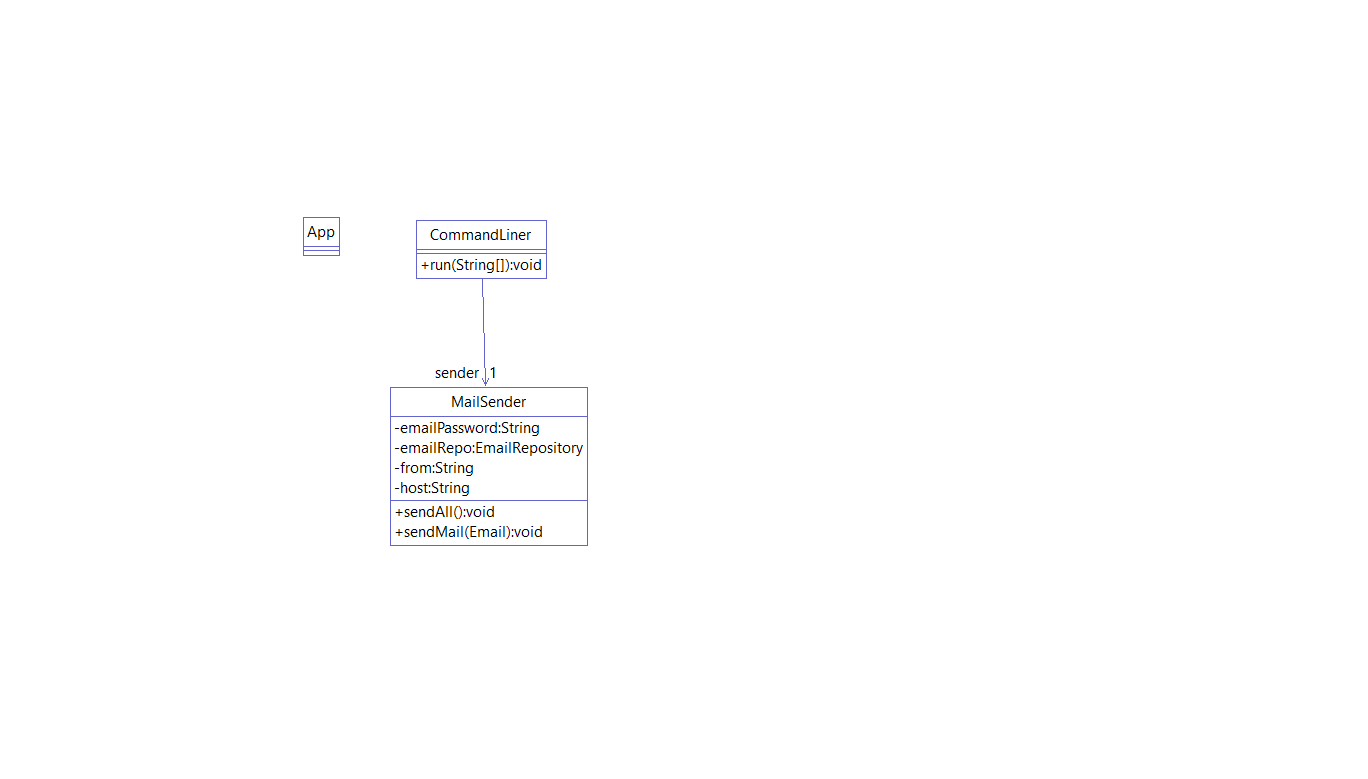
Es gibt auch mehrere Klassen, die einfach vom webserver benötigt werden und die einige Properties weniger enthalten. Beispielsweise ist UserMeta eine Untermenge von User, enthält aber nur Properties, die für den User relevant sind (spricht eine Repräsentation ohne Passwörter etc).

Die Repositories liegen im package ch.fhnw.cssr.domain.repository, der LocalDateTimeAttributeConverter im package ch.fhnw.cssr.domain.jpa, der Rest im package ch.fhnw.cssr.domain. Der LocalDateTimeAttributeConverter ist eine Hilfsklasse, die es ermöglicht, die Java 8 Date / Time Klassen zu verwenden.

## Mailer

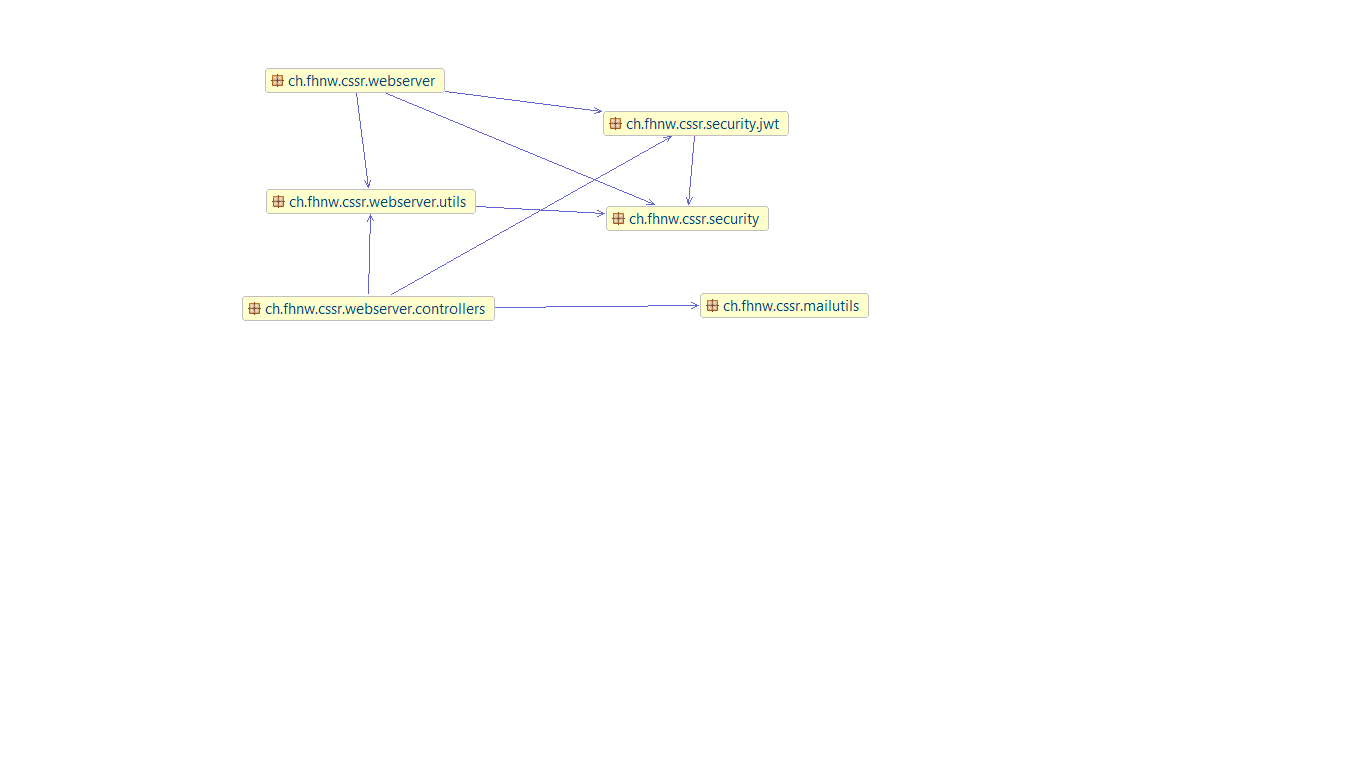
Das Klassendiagramm des Mailtools ist sehr bescheiden, macht es doch nicht viel mehr als einfach die Emails abzuarbeiten. Die App Klasse ist für die SpringBootConfiguration verantwortlich und ist der Einstiegspunkt der Applikation. Der CommandLiner ist die Klasse für Spring Boot, welche tatsächlich Code ausführt und der MailSender enthält die eigentliche Logik.

Die package ist hier ch.fhnw.cssr.mailer



## Webserver

Der Webserver Teil enthält doch einige Packages, der Übersicht halber sind hier die Dependencies:



Im folgenden Klassendiagramm sind die packages folgendermassen sortiert (jeweils mit Abstand getrennt):

* ch.fhnw.cssr.mailutils: Enthält nur die Klasse EmailTemplate für das Aufbereiten von formatierten Emails
* ch.fhnw.cssr.security: Enthält Klassen für Spring Boot, welche das Laden des Users betreffen
* ch.fhnw.cssr.security.jwt: Enthält Klassen für JWT Authorisierung
* ch.fhnw.cssr.webserver: Enthält die grundsätzliche Spring Boot Configuration
* ch.fhnw.cssr.webserver.controllers: Enthält die REST Controller
* ch.fhnw.cssr.webserver.utils: Hilfsklassen, vor allem für das User Handling

