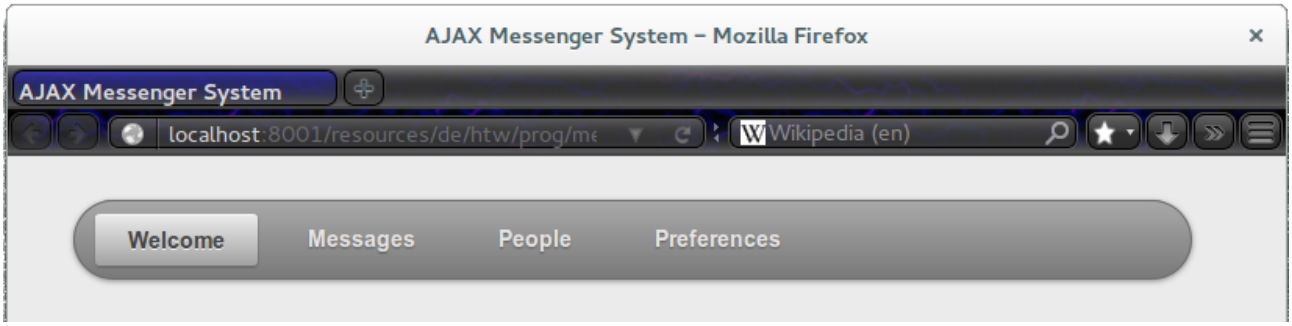
Master Programming

Übung messenger-web

Diese Übung adressiert den Umgang mit JavaScript/ECMAScript und Web 2.0 (AJAX).

Importiert aus dem Share-Laufwerk das Projekt **`messenger-web`**. Alle zu erstellenden Ressourcen sollen in dessen Ordner `src/WEB-INF` beheimatet sein.

Grundlagen



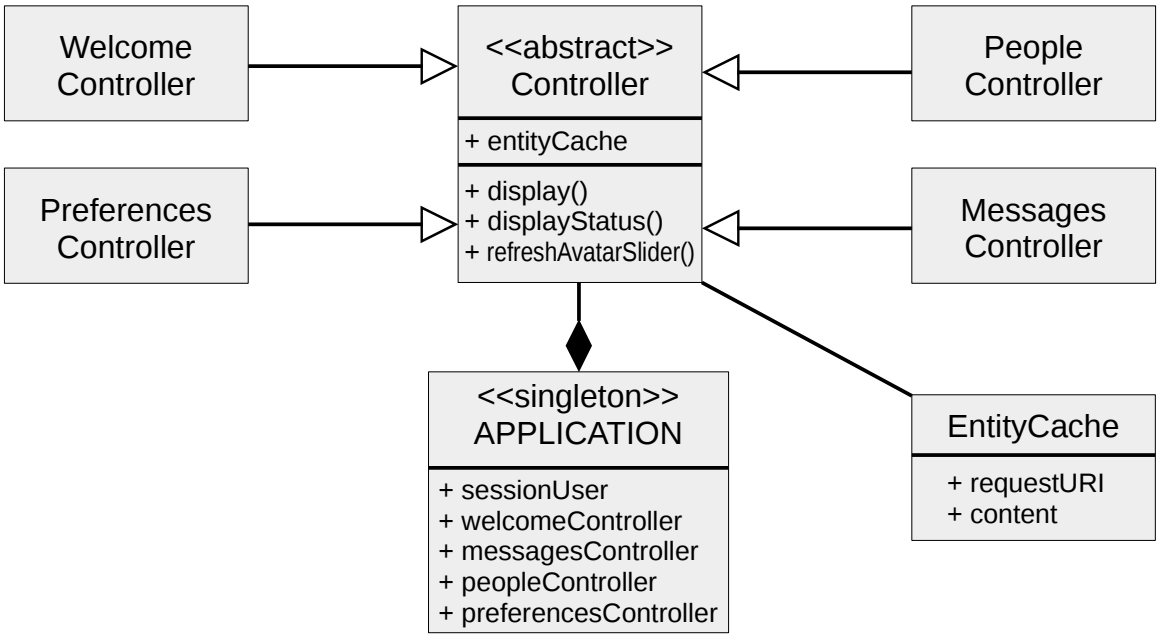
Diese Web 2.0 Applikation stellt eine minimalistische Alternative zu Facebook© dar, in dem angemeldete Benutzer Nachrichten an sich selbst und alle anderen Benutzer versenden welche an ihnen Interesse signalisiert haben. Dazu realisiert die Applikation eine Benutzeranmeldung (**Welcome**), eine Möglichkeit Benutzerdaten einzugeben (**Preferences**), eine Möglichkeit Interesse an anderen Benutzern zu signalisieren (**People**), und schließlich eine Möglichkeit die Nachrichten einzusehen (**Messages**).

Die Applikation ist strikt nach dem **Model-View-Controller** Paradigma entworfen. Dabei werden REST-Services als Model verwendet, JavaScript (vanilla) für Controller, und ein HTML-Dokument als View-DOM. Die Datei `broker.css` definiert dazu ein einfaches Web-Design; die Datei `messenger.html` verwendet dieses, und gliedert das **body**-Element der Seite mittels folgender **HTML5**-Elemente:

- **`header`**: Enthält das Menü der Seite (siehe unten)
- **`main`**: Enthält die aktuelle View der Seite, welche ausschließlich ein oder mehrere **section**-Elemente beinhaltet. Die Abschnitte sind dabei stets mit class-Attributen versehen um die Formatierung mittels CSS zu erleichtern.
- **`footer`**: Enthält Information über den HTTP Status des letzten REST-Aufrufs (für Fehlermeldungen)

Daneben sind im **head**-Element (nicht header!) der Seite einige **`template`**-Elemente definiert. Diese dienen für die JavaScript-Controller als Vorlagen für HTML-Fragmente, welche bei Bedarf geklont und in das **main**-Element eingebaut werden, z.B. zum Wechseln der aktiven View.

Klassendiagramm der wichtigsten JavaScript-Objekte



Der **Application**-Singleton stellt den Einstiegspunkt in die Applikation dar, und definiert u.A. den *Load-Event Handler* der aufgerufen wird sobald die Seite vollständig geladen ist. Er erzeugt zuvor bereits die vier Controller, einen für jeden Menüpunkt im Hauptmenü. Zudem veranlasst er initial eine Anzeige durch den Welcome-Controller (für Login), und verwaltet den aktuell authentifizierten Session-User .

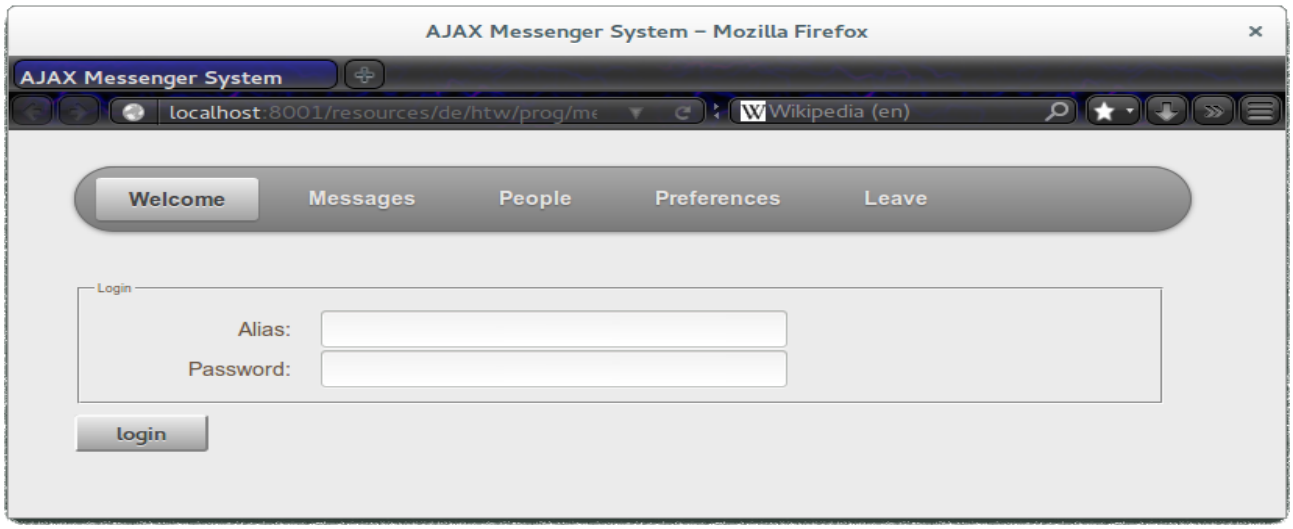
Die **Controller** haben die Aufgabe das **main**-Element der Seite zu initialisieren - siehe Methode `display()`. Zudem reagieren die Controller auf Events in der gesamten Seite, z.B. Button gedrückt, Drag&Drop eines Bildes, Menüpunkt gewählt, usw.). Die Controller haben dazu Zugriff auf einen gemeinsamen Entity-Cache. Die Controller für die Welcome- und Preferences-Views sind als Beispiele bereits vorhanden, während die Controller für die People- und Messages-Views noch zu erstellen sind.

Die Methode **`display()`** wird in jedem Controller-Subtyp erweitert: Die aus dem abstrakten Controller-Typ geerbte Fassung bedient das Hauptmenü und leert das main-Element der Seite. Das Überprüfen von Authorisierung, oder das Befüllen der View mit REST-basierter Information ist dagegen Aufgabe der abgeleiteten Controller.

Die Controller können außerdem die geerbte Methode **`displayStatus()`** nutzen um den Status von REST-Aufrufen an den Benutzer zu melden. Die ebenfalls geerbte Methode **`refreshAvatarSlider()`** kann zudem genutzt werden um die Avatar-Slider der People- und Message-Views mit Inhalt zu befüllen.

Der **Entity-Cache** nutzt REST-Aufrufe gegen den Entity-Service um eine gegebene Entity-Identität asynchron in die zugehörige Entität umzuwandeln – siehe Methode `resolve()`. Diese werden gecached und sparen so bei Mehrfachzugriff überflüssige REST-Aufrufe ein.

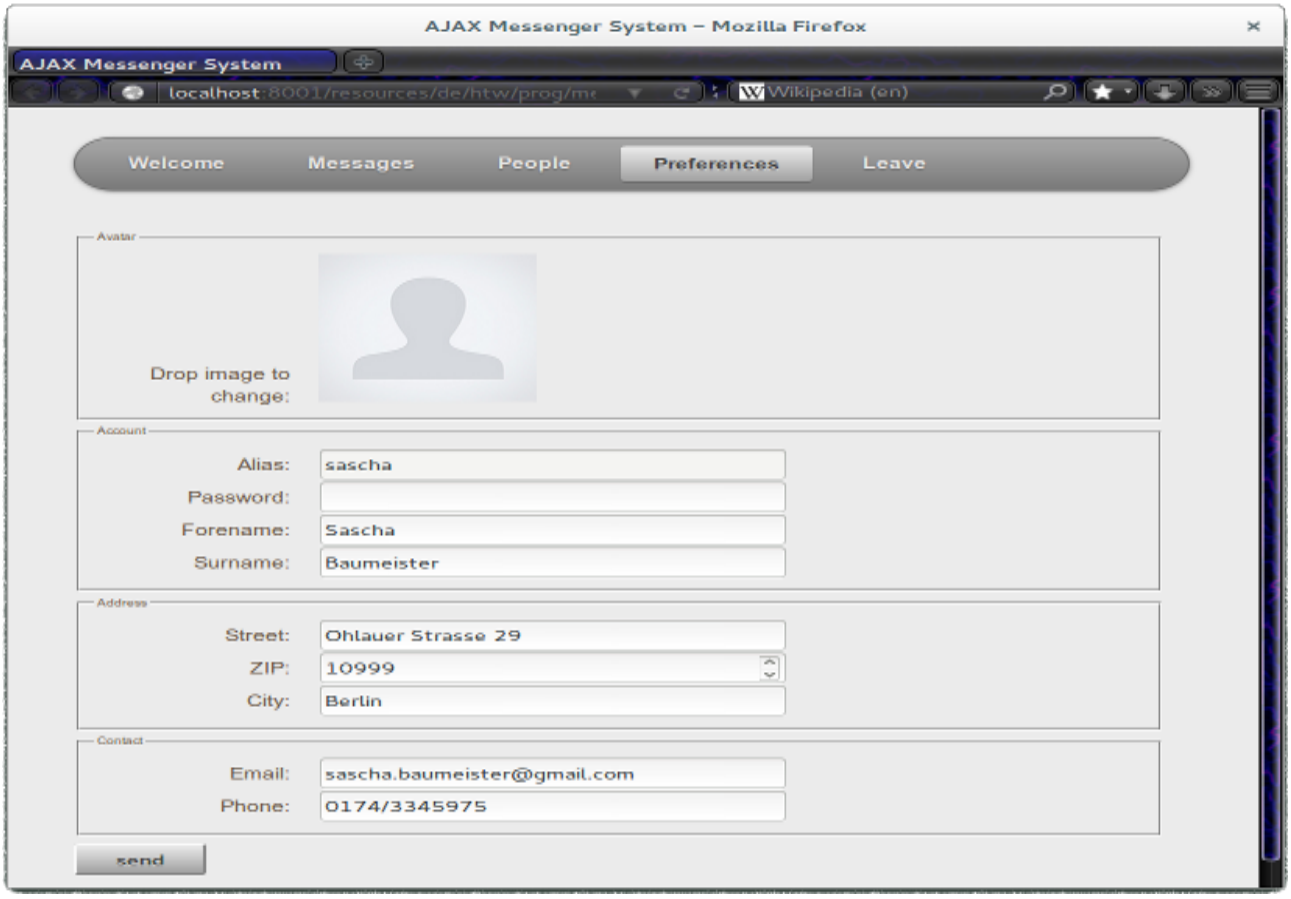
Welcome-View & -Controller



Der Typ `WelcomeController` ist bereits vorhanden, und dient zur Steuerung der Welcome-View:

- Datei `welcome-controller.js`, `Namespace „de.sb.messenger“`
- Templates: `login-template`
- Die Instanz-Methode **`display()`** stellt diese View dar, und registriert die Methode `login()` als Callback für den Einlogknopf.
- Die Instanz-Methode **`login()`** liest diese View aus und führt einen Benutzer-Login aus. Dieser wird mittels REST-Service Aufruf von *`GET /services/persons/requester`* durchgeführt. Der HTTP-Status dieses Aufrufs wird im footer-Element dargestellt. Bei Erfolg initiiert der Controller einen Wechsel in die Preferences-View.

Preferences-View & -Controller



Der Typ `PreferencesController` ist ebenfalls bereits vorhanden, und dient zur Steuerung der Preferences-View:

- Datei `preferences-controller.js`, `Namespace „de.sb.messenger“`
- Templates: `preferences-template`
- Die Instanz-Methode **`display()`** stellt diese View dar, und registriert die Methode `persist()` als Callback für den Sende-Knopf.
- Die Instanz-Methode **`persist()`** liest diese View aus und persistiert die aktuellen Benutzerdaten sowie bei Drag&Drop den aktualisierten Benutzer-Avatar mittels REST-Service Aufruf von *`PUT /services/persons`*. Der HTTP-Status dieses Aufrufs wird im footer-Element dargestellt.

Aufgabe A: Setup des Containers

Benennt folgende Dateien im Projekt-Ordner `src/WEB-INF/js` um:

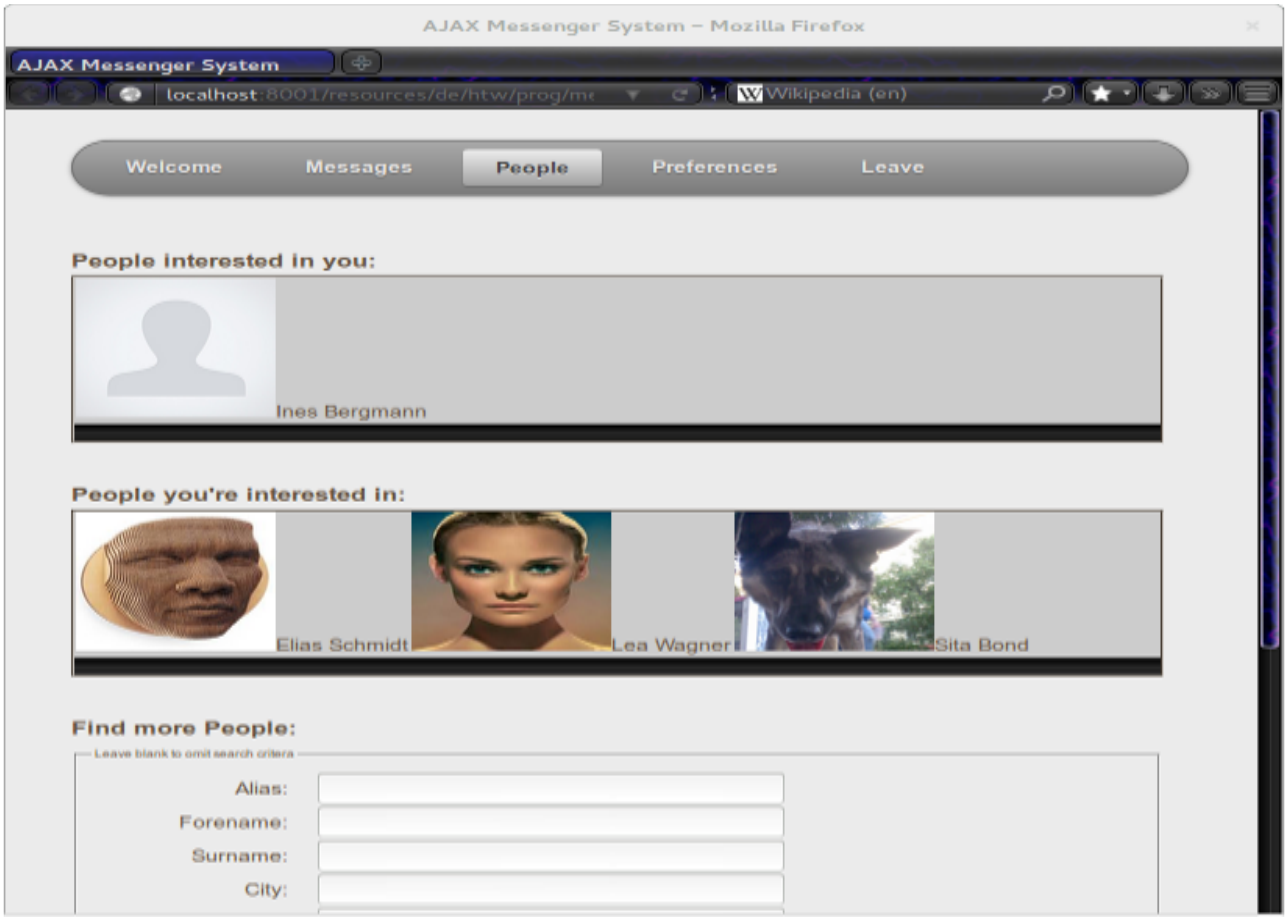
- `people-controller-js-skeleton` nach `people-controller.js`
- `messages-controller.js-skeleton` nach `messages-controller.js`

Startet in Eclipse die Java-Klasse **`de.sb.messenger.ApplicationContainer`** des Projekts. Diese Klasse dient als HTTP-Container für die Messenger-Anwendung und stellt dazu REST- und HTML-Ressourcen aus derselben Quelle (i.e. gleiche IP Socket-Adresse) zur Verfügung:

- Runtime-Profilname „**MessengerApplicationContainer**“, Laufzeit-Argument „**8001**“, VM-Argument „`-javaagent:/<path>/eclipselink.jar`“ (wobei <path> mit dem auf Eurem System gültigen Pfad zu ersetzen ist).
- Basis-URL: **`http://localhost:8001/`**
- Die REST-Services sind dabei unter dem Kontext-Pfad **`/services`** verfügbar
- Die HTML-Ressourcen (HTML-Datei, CSS, JavaScript-Dateien, Bilder, usw.) sind unter dem Kontext-Pfad **`/resources/WEB-INF`** verfügbar.

WEB-INF ist (neben META-INF) ein reservierter Name, und kann daher nicht für Java-Pakete verwendet werden. Dies erlaubt dem Container zur Laufzeit das Auslesen von Web-Ressourcen innerhalb des Java-Projekts, ohne dass diese in künstlichen Java-Paketen gehalten werden müssten; letztere könnten sonst in der Folge leicht mit JavaScript Namespaces verwechselt werden. Zudem erlaubt es optional (nach Konfiguration) den sinnvollen Einsatz der Eclipse Web-View.

Aufgabe B: People-View & -Controller



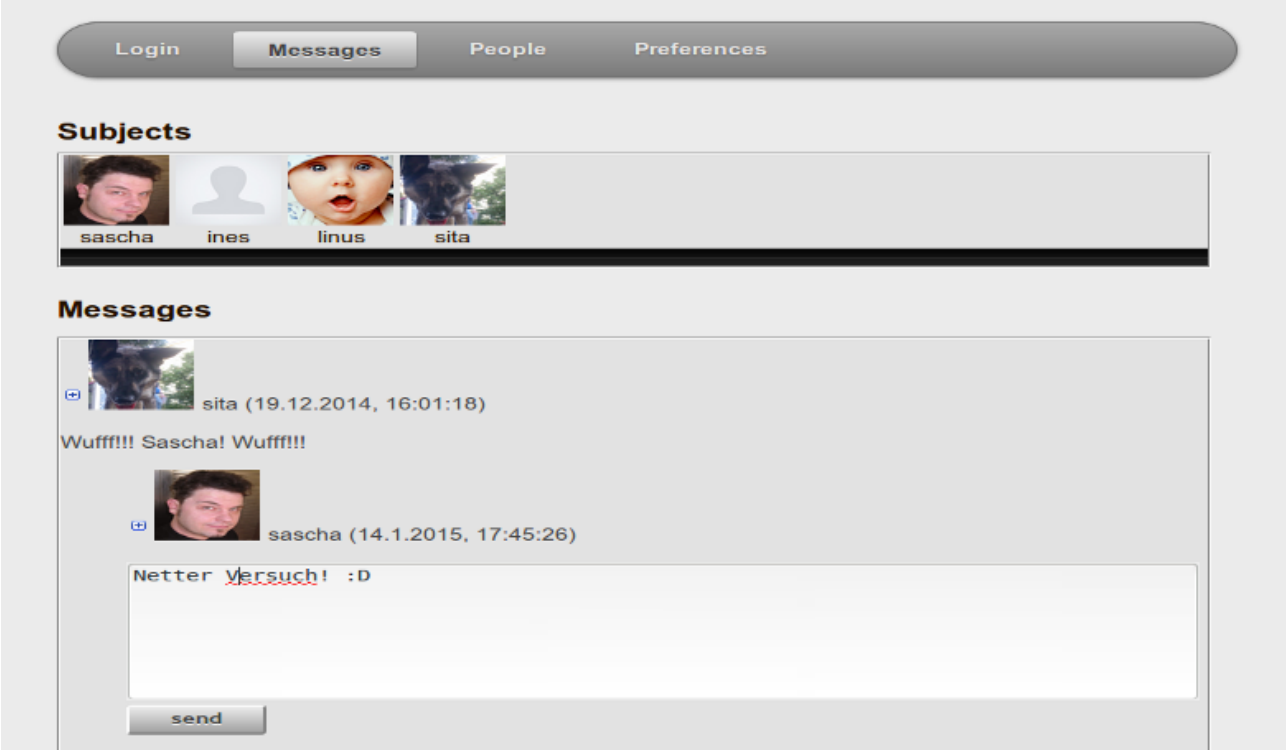
Der Typ `PeopleController` ist bereits teilweise implementiert, und dient zur Steuerung der People-View:

- Datei `people-controller.js`, `Namespace „de.sb.messenger“`
- Templates:
  - `people-observing-template`
  - `people-observed-template`
  - `candidates-template`
- Die Instanz-Methode **`display()`** stellt diese View dar, und registriert die Methode `query()` als Callback für das Drücken des Sende-Knopfs. Zudem registriert sie die Methode `toggleMonitorTarget()` als Callback für das Klicken auf einen der Benutzer-Avatare.

Definiert die fehlenden Teile dieses Controllers:

- Die Instanz-Methode **`query()`** soll den Inhalt der Kriterien-Felder auslesen, einen Kriterien-Query mittels geeignetem REST-Service Aufruf durchführen, und das Ergebnis im dritten Avatar-Slider der View anzeigen.
- Die Instanz-Methode **`toggleObservation(personIdentity)`** soll die jeweilige Person mittels REST-Service Aufruf zu den durch den Benutzer beobachteten Personen hinzufügen bzw. von diesen entfernen, je nachdem ob diese Person bereits in dieser Relationsmenge zu finden ist oder nicht.

Aufgabe C: Messages-View & -Controller



Der Typ `MessagesController` ist ebenfalls bereits teilweise implementiert, und dient zur Steuerung der Messages-View:

- Datei `messages-controller.js`, `Namespace „de.sb.messenger“`
- Templates:
  - `subjects-template`
  - `messages-template`
  - `message-output-template`
  - `message-input-template`
- Die Instanz-Methode **`display()`** stellt diese View teilweise dar, und registriert die Methode `displayMessageEditor()` als Callback für das Klicken auf einen der Benutzer-Avatare im Avatar-Slider, zur Erzeugung einer Message mit der gewählten Person als **Subject**. Des Weiteren wird die Methode `displayRootMessages()` aufgerufen um die Darstellung der Seite zu vervollständigen.

Definiert die fehlenden Teile dieses Controllers:

- Die Instanz-Methode **`displayRootMessages()`** soll diejenigen Messages untereinander anzeigen welche als **Subject** (sic!) entweder den aktuellen Benutzer oder eine von diesem beobachtete Person haben. Dazu müssen die Messages mittels multipler REST-Aufrufe abgefragt, und zudem nach absteigendem Erzeugungsdatum sortiert werden. Bei Klick auf das Plus-Symbol neben dem jeweiligen Benutzer-Avatar sollen alle Messages eingeblendet werden welche die gewählte Message (sic!) als Subject haben – setzt dazu die Klasse „expanded“ im `li`-Element der Message. Bei

Klick auf einen der Avatare soll die Methode `displayMessageEditor()` aufgerufen werden um eine Nachricht mit der gewählten Message als Subject zu erzeugen.

- Die Instanz-Methode **`displayMessageEditor(parentElement, subjectIdentity)`** soll einen Message-Editor auf der gewählten Hierarchieebene einblenden. Die Message soll dabei das gegebene Subject, und den aktuellen Benutzer als Autor erhalten. Nach Klick auf deren Sende-Knopf soll die Nachricht mittels REST-Aufruf gespeichert, und die Hierarchieebene frisch geladen werden.