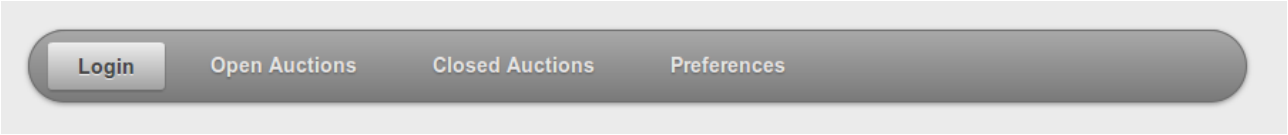
Master Programming

Übung broker-web

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

Importiert aus dem Share-Laufwerk das Projekt **`broker-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 eBay© dar, in dem angemeldete Benutzer zeit-limitierte Auktionen erstellen und auf solche bieten können; im Gegensatz zu eBay werden hier jedoch versteckte Gebote hinterlegt. Dies führt dazu dass die Gebote noch angehoben oder abgesenkt werden können solange die zugehörigen Auktion noch nicht geschlossen ist. Wer am Ende das höchste Gebot abgegeben hat gewinnt die Auktion. Nachdem eine Aktion geschlossen ist werden die Kontaktdaten und das Gebot des Gewinners dem Ersteller der Auktion sichtbar gemacht, und vice versa.

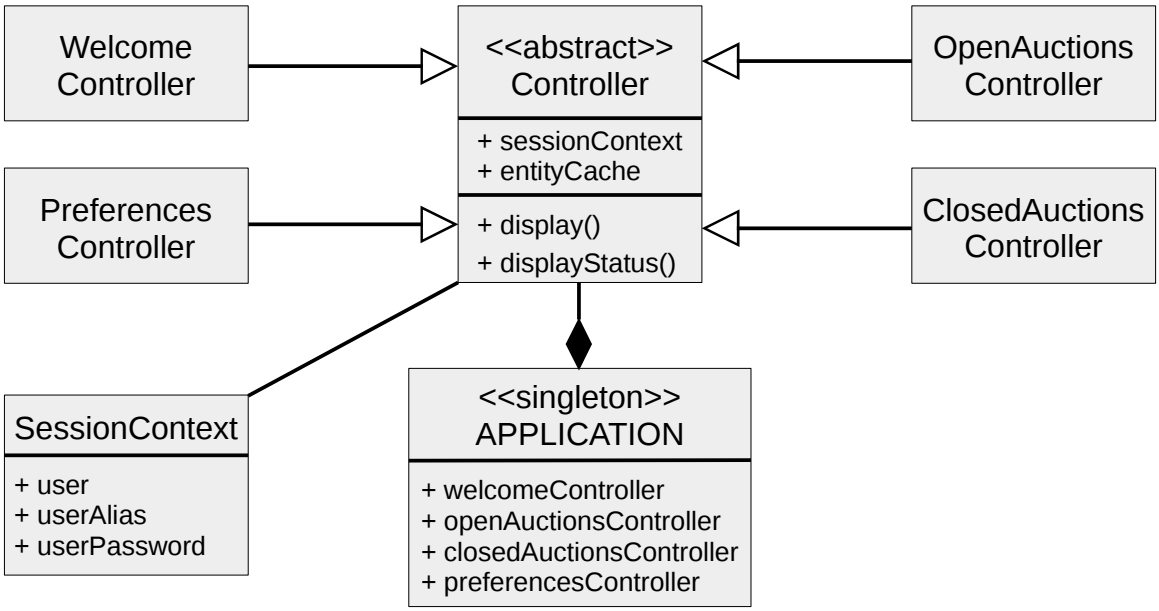
Dazu realisiert die Applikation eine Benutzeranmeldung (**Welcome**), eine Möglichkeit Benutzerdaten einzugeben (**Preferences**), eine Möglichkeit geschlossene Auktionen einzusehen welche entweder vom Benutzer erzeugt oder geboten wurden (**Closed Auctions**), und schließlich eine Möglichkeit die offenen Auktionen einzusehen, auf diese zu bieten, oder eigene Auktionen zu erstellen bzw. zu ändern (**Open Auctions**).

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 `stylesheet.css` definiert dazu ein einfaches Web-Design; die Datei `broker.html` verwendet dieses, und gliedert das **body**-Element der Seite mittels folgender **HTML5**-Elemente:

- **`header`**: Enthält das Menü der Seite
- **`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



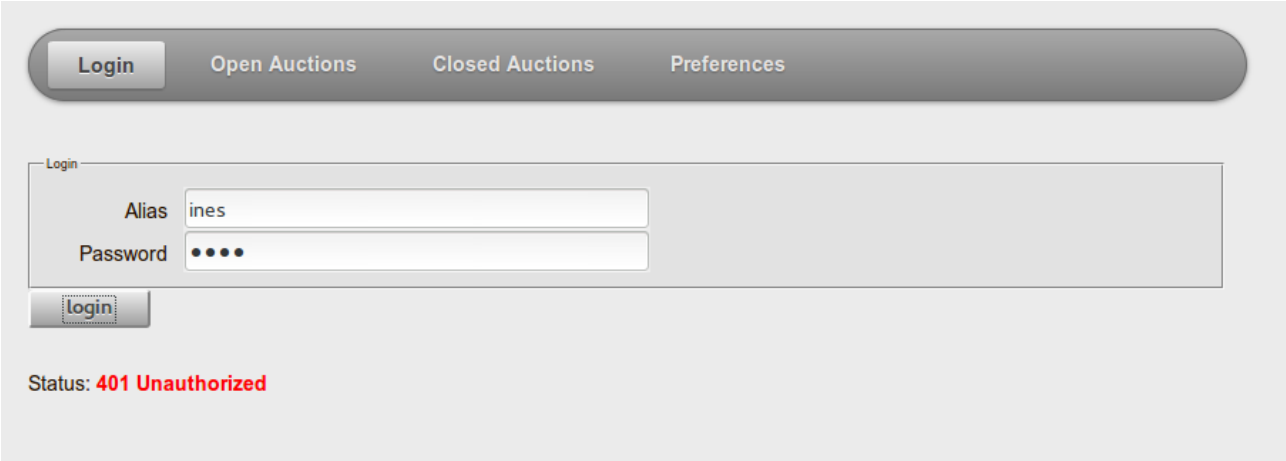
Der **Application**-Singleton stellt den Einstiegspunkt in die Applikation dar, und definiert u.A. den *Load-Event Handler* der aufgerufen wird sobald die Seite vollständig geladen ist. Er erzeugt zuvor bereits die vier Controller, einen für jeden Menüpunkt im Hauptmenü. Zudem veranlasst er initial eine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

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 Session-Context. Die Controller für die Welcome- und Preferences-Views sind als Beispiele bereits vorhanden, während die Controller für die `OpenAuctions`- und `ClosedAuctions`-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 zudem die ererbte Methode **`displayStatus()`** nutzen um den Status von REST-Aufrufen an den Benutzer zu melden.

Der **Session-Context** verwaltet das Person-Objekt des aktuellen Benutzers, sowie sein Passwort. Es adaptiert diese Information um als ganzes für REST-Aufrufe mit HTML Authorisierung (siehe `util.js`) verwendet werden zu können.

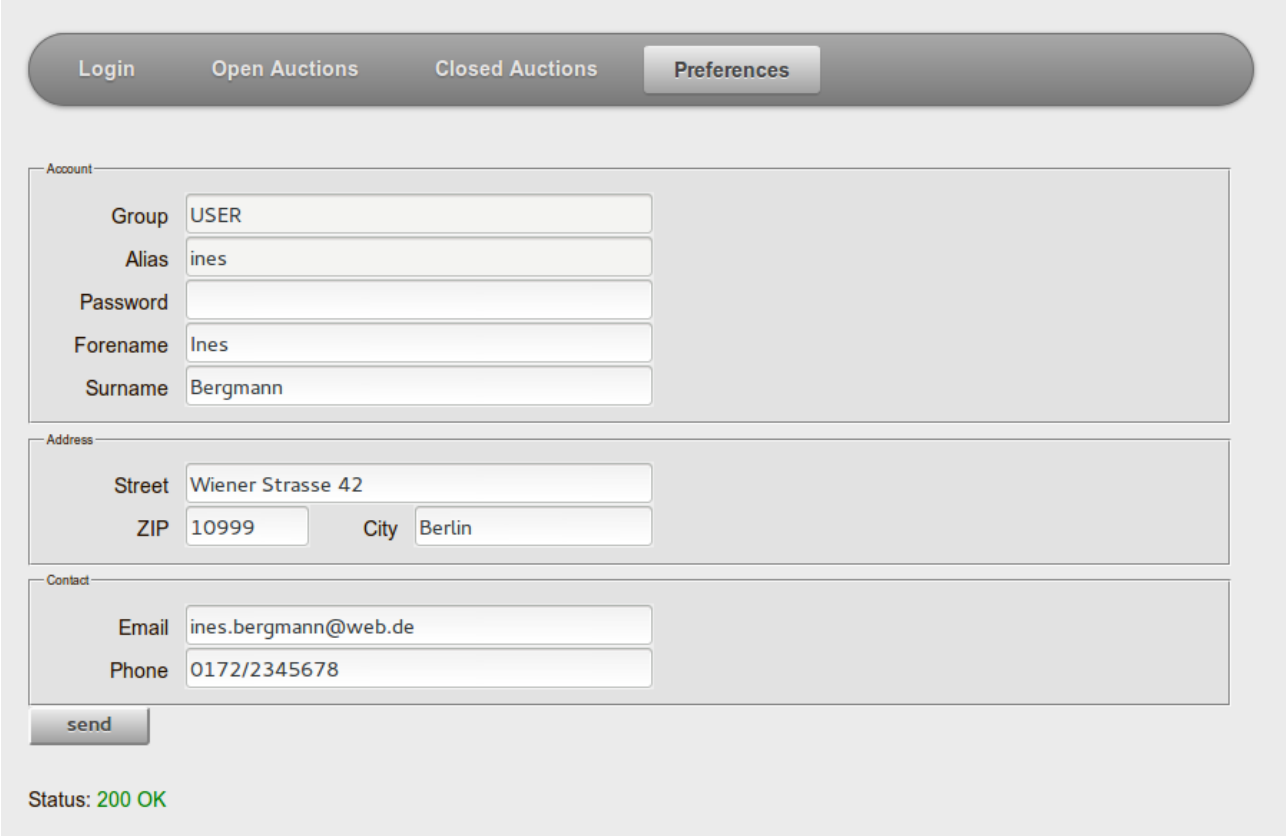
Welcome-View & -Controller



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

- Datei `welcome-controller.js`, `Namespace „de.sb.broker“`
- 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.broker“`
- 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

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

- Runtime-Profilname „**BrokerApplicationContainer**“, Laufzeit-Argument „**8001**“
- 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: Closed-Auctions-View & -Controller

Erstellt den Typ `ClosedAuctionsController` zur Steuerung der Closed-Auctions-View:

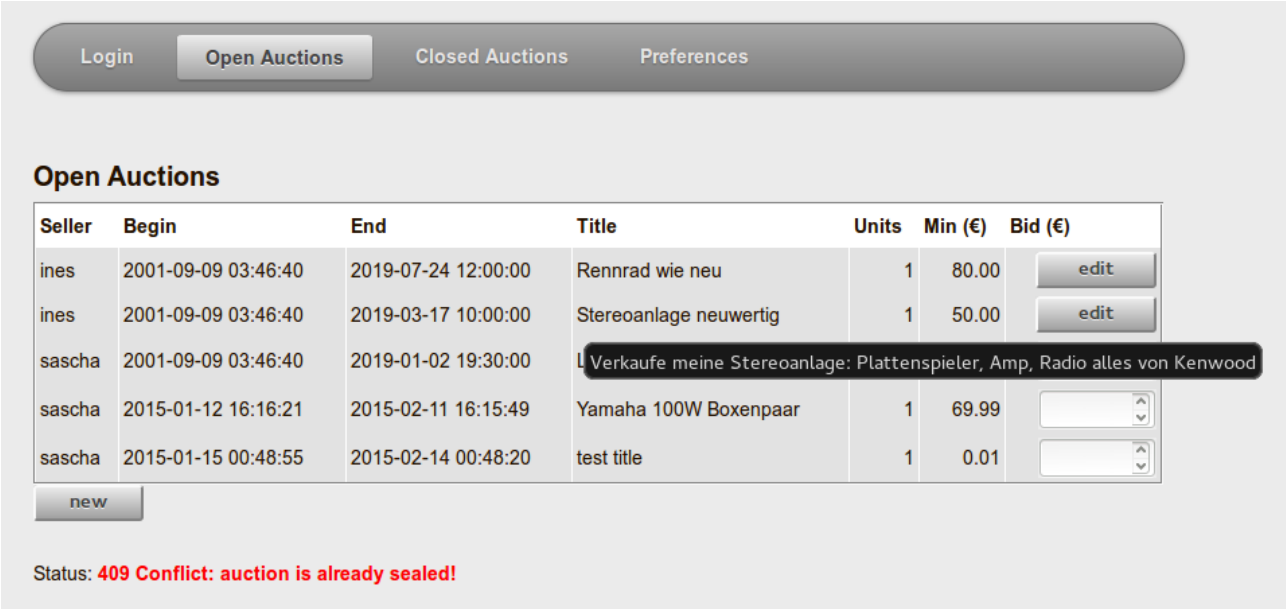
- Neue Datei `closed-auctions-controller.js`, `Namespace „de.sb.broker“`
- Templates: `closed-auctions-template`, `closed-bids-template`

Definiert die fehlenden Teile dieses Controllers:

- Der **Konstruktor** soll so definiert werden dass der Typ von de.sb.broker.Controller erbt. Die beim Aufruf des Super-Konstruktors zu übergebende Ordinale ist dabei 2.
- Die Instanz-Methode **`display()`** stellt diese View dar indem sie die geerbte Version dieser Methode aufruft, und zudem zwei REST-Aufrufe von `GET /services/auctions` durchführt um die Daten für die beiden Tabellen zu erhalten. Nutzt dazu die Möglichkeit Euch von dieser REST-Methode Tupel von Entitäten liefern zu lassen. Zeigt zudem den höheren der beiden HTTP Codes an die aus diesen beiden asynchronen REST-Aufrufen resultieren. Teilt zudem die Aufgaben geeignet auf mehrere Methoden auf um die Übersichtlichkeit des Codes zu wahren.

Testet Eure Implementierung mit den Usern sascha (Passwort sascha) und ines (Passwort ines), da für diese bereits abgelaufene Auktionen bzw. Gebote definiert sind.

Aufgabe C: Open-Auctions-View & -Controller



Erstellt den Typ `OpenAuctionsController` zur Steuerung der OpenAuctions-View:

- Neue Datei `open-auctions-controller.js`, `Namespace „de.sb.broker“`
- Templates: `open-auctions-template`, `auctions-input-template`

Definiert die fehlenden Teile dieses Controllers:

- Der **Konstruktor** soll so definiert werden dass der Typ von de.sb.broker.Controller erbt. Die beim Aufruf des Super-Konstruktors zu übergebende Ordinale ist dabei 1.
- Die Instanz-Methode **`display()`** stellt diese View dar indem sie die geerbte Version dieser Methode aufruft, und zudem einen REST-Aufruf von `GET /services/auctions` durchführt um die Daten für die Tabelle zu erhalten. Nutzt dazu die Möglichkeit Euch von dieser REST-Methode Tupel von Entitäten liefern zu lassen. Zeigt zudem den HTTP Code an der aus diesem asynchronen REST-Aufruf resultiert. Achtet darauf in der letzten Spalte bei eigenen Auktionen einen Knopf zu erzeugen, während bei fremden Auktionen ein Eingabefeld für ein Gebot eingesetzt werden soll. Initialisiert zudem die Callback-Methoden für das Klicken der Knöpfe dieser View geeignet.
- Beim Klicken auf den new-Knopf soll unter der Tabelle ein Auktions-Editor eingeblendet werden welcher als HTML-Template bereits vordefiniert ist. Die Auktion soll dabei sofort beginnen, und 30 Tage andauern. Beim Klicken eines der edit-Knöpfe soll der gleiche Editor erscheinen, nur diesmal mit den Daten der bestehenden Auktion initialisiert. Beim Drücken des Sende-Knopfes im Editor soll die Auktion mittels REST-Aufruf von `PUT /services/auctions` persistiert, und zudem der Auktions-Editor wieder aus der View entfernt werden. Im Fehlerfall sollen die Daten in der Tabelle zurückgesetzt werden; achtet dabei darauf dass dieses nicht den HTTP Code der Auktions-Persistierung überschreibt. Dies ist besonders wichtig bei Fehler 409 da dieser anzeigt dass eine Auktion nicht mehr modifiziert werden kann weil bereits Gebote für sie existieren, oder weil sie in der Zwischenzeit abgelaufen ist.