Projekt YouSong

# Spring

# Application Properties

*# Server konfigurieren für HTTP***server.address**=**127.0.0.1  
server.port** = **8443  
server.ssl.key-store** = **classpath:keystore.jks  
server.ssl.key-store-password** = **123456  
server.ssl.key-alias** = **sew  
server.ssl.key-password** = **123456  
server.http2.enabled** = **true***# Basispfad für Requests an das REST-API***spring.data.rest.base-path**=**/api***# Verzeichnis, Name und Zugangsdaten der Datenbank***spring.datasource.url**=**jdbc:h2:./db/datenbank;AUTO\_SERVER=TRUE  
spring.datasource.username**=**sa  
spring.datasource.password**=  
  
*# Datenbank bei jedem Start neu erzeugen? (none, validate, update, create, create-drop)  
#spring.jpa.hibernate.ddl-auto=validate***spring.jpa.hibernate.ddl-auto**=**create***# src/main/resources/data.sql bei jedem Start ausführen? (always, never)  
#spring.datasource.initialization-mode=never***spring.datasource.initialization-mode**=*always  
  
# SQL-Anweisungen protokollieren, die auf der Datenbank ausgeführt werden***spring.jpa.show-sql**=**true**

# Interface UserInfo

Muss 1 zu 1 so kopiert und übernommen werden. Wurde überrschrieben, da ein Fehler in der Vorlage vorhanden war

*//*

*// Source code recreated from a .class file by IntelliJ IDEA*

*// (powered by Fernflower decompiler)*

*//*

**package** at.rennweg.htl.sew.autoconfig;

**import** **com.fasterxml.jackson.annotation.JsonIgnore**;

**import** **java.util.Collection**;

**import** **java.util.Collections**;

**import** **org.springframework.security.core.GrantedAuthority**;

**import** **org.springframework.security.core.authority.SimpleGrantedAuthority**;

**import** **org.springframework.security.core.userdetails.UserDetails**;

**public** **interface** **UserInfo** **extends** UserDetails {

Long getId();

String getName();

@JsonIgnore

String getPassword();

@JsonIgnore

**default** String getRole() {

**return** "ROLE\_USER";

}

@JsonIgnore

**default** Collection<? **extends** GrantedAuthority> getAuthorities() {

**return** Collections.singleton(**new** SimpleGrantedAuthority(**this**.getRole()));

}

@JsonIgnore

**default** String getUsername() {

**return** **this**.getName();

}

@JsonIgnore

**default** boolean isAccountNonExpired() {

**return** **true**;

}

@JsonIgnore

**default** boolean isAccountNonLocked() {

**return** **true**;

}

@JsonIgnore

**default** boolean isCredentialsNonExpired() {

**return** **true**;

}

@JsonIgnore

**default** boolean isEnabled() {

**return** **true**;

}

}

# Fachklasse Album

Stellt ein Album in der Datenbank dar.

**package** server.models;

**import** **javax.persistence.Cacheable**;

**import** **javax.persistence.Entity**;

**import** **javax.persistence.ManyToMany**;

**import** **javax.validation.constraints.NotBlank**;

**import** **java.util.Set**;

@Entity

@Cacheable

**public** **class** **Album** **extends** Persistent {

@NotBlank

**private** String title;

@ManyToMany

**private** Set<Song> songs;

**public** String getTitle() {

**return** title;

}

**public** void setTitle(String title) {

**this**.title = title;

}

**public** Set<Song> getSongs() {

**return** songs;

}

**public** void setSongs(Set<Song> songs) {

**this**.songs = songs;

}

}

# Fachklasse Artist

Stellt einen Artist in der Datenbank dar.

**package** server.models;

**import** **javax.persistence.\***;

**import** **javax.validation.constraints.NotBlank**;

**import** **java.util.Set**;

@Entity

@Cacheable

**public** **class** **Artist** **extends** Persistent {

@NotBlank(message = "Name der Interpretin fehlt")

@Column(unique = **true**)

**private** String name;

*// Beim Löschen eines Interpreten auch alle seine Songs löschen*

@OneToMany(mappedBy = "artist", cascade = CascadeType.ALL)

**private** Set<Song> songs;

**public** String getName() {

**return** name;

}

**public** void setName(String name) {

**this**.name = name;

}

**public** Set<Song> getSongs() {

**return** songs;

}

**public** void setSongs(Set<Song> songs) {

**this**.songs = songs;

}

}

# Projection ArtistName

Projection der Klasse Artist, welche nur den Namen ausgibt.

**package** server.models;

**import** **org.springframework.data.rest.core.config.Projection**;

@Projection(name = "artistName", types = Artist.class)

**public** **interface** **ArtistName** {

String getName();

}

# Superklasse Persistent

Jede Fachklasse vererbt von Persistent. In Persistent wird die id, sowie die Versionierung mittels eTag umgesetzt.

**package** server.models;

**import** **javax.persistence.\***;

@MappedSuperclass

**public** **class** **Persistent** {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

**private** Long id; *// Primärschlüssel, immer benötigt*

@Version

**private** long version;

**public** Long getId() {

**return** id;

}

**public** long getETag() { **return** version; }

}

# Fachklasse Song

Diese umfangreiche Fachklasse stellt einen Song in der Datenbank dar.

**package** server.models;

**import** **com.fasterxml.jackson.annotation.JsonIgnore**;

**import** **com.fasterxml.jackson.annotation.JsonProperty**;

**import** **org.springframework.data.annotation.CreatedBy**;

**import** **org.springframework.data.jpa.domain.support.AuditingEntityListener**;

**import** **javax.persistence.\***;

**import** **javax.validation.constraints.Min**;

**import** **javax.validation.constraints.NotBlank**;

**import** **java.util.Set**;

@Entity

@Cacheable

@Table(uniqueConstraints =

@UniqueConstraint(columnNames = { "title", "artist\_id" })

)

@EntityListeners(AuditingEntityListener.class)

**public** **class** **Song** **extends** Persistent {

@NotBlank(message = "Es muss ein Titel angegeben werden")

**private** String title;

@ManyToOne(optional = **false**)

**private** Artist artist;

@ManyToOne(optional = **false**)

@CreatedBy

@JsonIgnore

**private** User uploader;

@ManyToMany(mappedBy = "songs")

**private** Set<Album> alben;

@Lob

**private** String cover;

@Lob

**private** String audio;

@Min(value = 1, message = "Die Spieldauer muss positiv sein")

**private** Integer duration;

**public** String getTitle() {

**return** title;

}

**public** void setTitle(String title) {

**this**.title = title;

}

**public** Artist getArtist() {

**return** artist;

}

**public** void setArtist(Artist artist) {

**this**.artist = artist;

}

@JsonProperty

**public** User getUploader() {

**return** uploader;

}

@JsonIgnore

**public** void setUploader(User uploader) {

**this**.uploader = uploader;

}

**public** Set<Album> getAlben() {

**return** alben;

}

**public** void setAlben(Set<Album> alben) {

**this**.alben = alben;

}

**public** String getCover() {

**return** cover;

}

**public** void setCover(String cover) {

**this**.cover = cover;

}

**public** String getAudio() {

**return** audio;

}

**public** void setAudio(String audio) {

**this**.audio = audio;

}

**public** Integer getDuration() {

**return** duration;

}

**public** void setDuration(Integer duration) {

**this**.duration = duration;

}

}

# Projektion SongAudio

Gibt nur die Audiospur eines Songs zurück.

**package** server.models;

**import** **org.springframework.data.rest.core.config.Projection**;

@Projection(name = "nurAudio", types = Song.class)

**public** **interface** **SongAudio** {

String getAudio();

}

# Projektion SongOhneAudio

Gibt nur die Infos eines Songs, ohne Audiospur zurück.

**package** server.models;

**import** **org.springframework.data.rest.core.config.Projection**;

**import** **java.util.Set**;

@Projection(name = "ohneAudio", types = Song.class)

**public** **interface** **SongOhneAudio** {

String getTitle();

Artist getArtist();

User getUploader();

Set<Album> getAlben();

String getCover();

Integer getDuration();

}

# Fachklasse User

Stellt die Entität User in der Datenbank dar.

**package** server.models;

**import** **at.rennweg.htl.sew.autoconfig.UserInfo**;

**import** **com.fasterxml.jackson.annotation.JsonIgnore**;

**import** **org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder**;

**import** **javax.persistence.Cacheable**;

**import** **javax.persistence.Column**;

**import** **javax.persistence.Entity**;

**import** **javax.validation.constraints.NotBlank**;

@Entity

@Cacheable

**public** **class** **User** **extends** Persistent **implements** UserInfo {

@NotBlank

@Column(updatable = **false**)

**private** String name;

@Column(unique = **true**, updatable = **false**)

@JsonIgnore

**private** String nameLC;

@NotBlank

@JsonIgnore

**private** String passwort;

**public** String getName() {

**return** name;

}

@Override

**public** String getPassword() {

**return** passwort;

}

**public** void setName(String name) {

**this**.name = name;

**this**.nameLC = (name != **null**) ? name.toLowerCase() : **null**;

}

**public** String getNameLC() {

**return** nameLC;

}

**public** void setNameLC(String nameLC) {

**this**.nameLC = nameLC;

}

**public** String getPasswort() {

**return** passwort;

}

**public** void setPasswort(String passwort) {

**this**.passwort = passwort;

}

**public** void setKlartextPw(String pw) {

passwort = **new** BCryptPasswordEncoder(12)

.encode(pw);

}

}

# Projection UserName

Gibt nur den Namen des Users zurück

**package** server.models;

**import** **org.springframework.data.rest.core.config.Projection**;

@Projection(types = User.class)

**public** **interface** **UserName** {

String getName();

}

# Repository ArtistRepository

Stellt die Fachklasse Artist als PagingAndSortingRepository in der REST API zur Verfügung.

**package** server.repositories;

**import** **org.springframework.data.repository.PagingAndSortingRepository**;

**import** **org.springframework.data.rest.core.annotation.RepositoryRestResource**;

**import** **server.models.Artist**;

**import** **server.models.ArtistName**;

@RepositoryRestResource(excerptProjection = ArtistName.class)

**public** **interface** **ArtistRepository** **extends** PagingAndSortingRepository<Artist, Long> {

}

# Repository SongRepository

Stellt die Songs über die REST API mit Einschränkungen zur Verfügung.

**package** server.repositories;

**import** **org.springframework.data.domain.Page**;

**import** **org.springframework.data.domain.Pageable**;

**import** **org.springframework.data.repository.PagingAndSortingRepository**;

**import** **org.springframework.data.repository.query.Param**;

**import** **org.springframework.data.rest.core.annotation.RepositoryRestResource**;

**import** **org.springframework.security.access.prepost.PreAuthorize**;

**import** **server.models.Song**;

**import** **server.models.SongOhneAudio**;

**import** **java.util.Optional**;

@RepositoryRestResource(excerptProjection = SongOhneAudio.class)

**public** **interface** **SongRepository**

**extends** PagingAndSortingRepository<Song, Long> {

*/\*\**

*\* Nur angemeldete User dürfen Songs sehen.*

*\*/*

@Override

@PreAuthorize("isAuthenticated()")

Page<Song> findAll(Pageable pageable);

*/\*\**

*\* Nur angemeldete User dürfen Songs sehen.*

*\*/*

@Override

@PreAuthorize("isAuthenticated()")

Optional<Song> findById(Long aLong);

*/\*\**

*\* Nur angemeldete User dürfen Songs sehen.*

*\*/*

@PreAuthorize("isAuthenticated()")

Page<Song> findByTitleContainingIgnoreCase(

@Param("titleText") String t,

Pageable p

);

*/\*\**

*\* Nur angemeldete User dürfen Songs sehen.*

*\*/*

@PreAuthorize("isAuthenticated()")

Page<Song> findByArtistNameContainingIgnoreCaseAndTitleContainingIgnoreCaseOrderByTitle(

@Param("artistText") String a,

@Param("titleText") String t,

Pageable p

);

*/\*\**

*\* Jede Userin darf nur die eigenen Songs löschen.*

*\*/*

@Override

@PreAuthorize("isAuthenticated() and #s.uploader.id == principal.id")

void delete(@Param("s") Song song);

*/\*\**

*\* Jede Userin darf nur die eigenen Songs bearbeiten.*

*\*/*

@Override

@PreAuthorize("isAuthenticated() and (#s.uploader == null or #s.uploader.id == principal.id)")

<S **extends** Song> S save(@Param("s") S song);

}

# Service UserInfoService

Service, der den UserDetailService implementiert.

**package** server.repositories;

**import** **org.springframework.beans.factory.annotation.Autowired**;

**import** **org.springframework.security.core.userdetails.UserDetails**;

**import** **org.springframework.security.core.userdetails.UserDetailsService**;

**import** **org.springframework.security.core.userdetails.UsernameNotFoundException**;

**import** **org.springframework.stereotype.Service**;

**import** **server.models.User**;

@Service

**public** **class** **UserInfoService** **implements** UserDetailsService {

@Autowired *// Dependency Injection: BenutzerRepository-Referenz in der Variablen speichern*

**private** UserRepository benutzerRepository;

*// Spring Security sucht bei der Authentifizierung mit dieser Methode die Benutzer-@Entity*

@Override

**public** UserDetails loadUserByUsername(String name) **throws** UsernameNotFoundException {

User benutzer = benutzerRepository

.findByNameIgnoreCase(name);

**if** (benutzer != **null**) {

**return** benutzer;

}

**else** {

**throw** **new** UsernameNotFoundException(name);

}

}

}

# Repository UserRepository

Repository stellt die User über die REST API zur Verfügung.

**package** server.repositories;

**import** **org.springframework.data.repository.PagingAndSortingRepository**;

**import** **org.springframework.data.rest.core.annotation.RepositoryRestResource**;

**import** **org.springframework.data.rest.core.annotation.RestResource**;

**import** **server.models.User**;

**import** **server.models.UserName**;

@RepositoryRestResource(excerptProjection = UserName.class)

**public** **interface** **UserRepository** **extends** PagingAndSortingRepository<User, Long> {

@RestResource(exported = **false**)

User findByNameIgnoreCase(String name);

}

# RestController Me

**package** server;

**import** **at.rennweg.htl.sew.autoconfig.UserInfo**;

**import** **com.fasterxml.jackson.annotation.JsonProperty**;

**import** **org.springframework.boot.autoconfigure.condition.ConditionalOnBean**;

**import** **org.springframework.data.rest.webmvc.BasePathAwareController**;

**import** **org.springframework.security.access.prepost.PreAuthorize**;

**import** **org.springframework.security.core.Authentication**;

**import** **org.springframework.security.core.context.SecurityContextHolder**;

**import** **org.springframework.security.core.userdetails.UserDetailsService**;

**import** **org.springframework.web.bind.annotation.RequestMapping**;

**import** **org.springframework.web.bind.annotation.RequestMethod**;

**import** **org.springframework.web.bind.annotation.RestController**;

@RestController

@BasePathAwareController

@ConditionalOnBean({UserDetailsService.class})

**public** **class** **Me** {

**public** Me() {

}

@RequestMapping(

value = {"/me"},

method = {RequestMethod.GET}

)

@PreAuthorize("isAuthenticated()")

**public** Me.MeInfo me() {

Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

**if** (authentication != **null** && authentication.getPrincipal() **instanceof** UserInfo) {

**final** UserInfo userInfo = (UserInfo)authentication.getPrincipal();

**return** **new** Me.MeInfo() {

**public** Long getId() {

**return** userInfo.getId();

}

**public** String getName() {

**return** userInfo.getName();

}

**public** String getRole() {

**return** userInfo.getRole();

}

};

} **else** {

**return** **null**;

}

}

**private** **interface** **MeInfo** {

@JsonProperty

Long getId();

@JsonProperty

String getName();

@JsonProperty

String getRole();

}

}

# AngularJS

# Komponente Artist

Stellt die Zeile eines Artist dar.

## Template Artist

<**div class="artist" flex="none"  
 layout="row" layout-align="start center" layout-padding**>  
  
 <**div flex="20"**>{{ $ctrl.**artist**.**name** }}</**div**>  
  
 <**div flex**></**div**>  
  
 <**div flex="10"**>  
 <**md-button class="md-fab md-mini"  
 ng-click="$ctrl.loeschen({ artist: $ctrl.artist })"**>  
 <**md-icon md-font-icon="material-icons"**>delete</**md-icon**>  
 </**md-button**>  
 </**div**>  
</**div**>

## Controller Artist

**"use strict"**;  
  
***app***.**component**(**"artist"**, {  
 **templateUrl**: **"components/artist.html"**,  
 **bindings**: {  
 **artist**: **"<"**,  
 **loeschen**: **"&"** }  
});

# Komponente ArtistListe

Listet alle Artists in einer Liste auf.

## Template ArtistListe

<**div class="artist-liste liste" layout="column" layout-fill**>  
 <**md-toolbar**>  
 <**div class="md-toolbar-tools"**>  
 <**h3**>Interpreten</**h3**>  
  
 <**http-aktiv**>  
 <**md-progress-circular md-mode="indeterminate"  
 md-diameter="30px"  
 class="md-accent"**>  
 </**md-progress-circular**>  
 </**http-aktiv**>  
  
 <**seiten-nav seite="$ctrl.seite"  
 seite-laden="$ctrl.seiteLaden(seite)"  
 ng-show="$ctrl.seite.page.totalPages"**>  
 </**seiten-nav**>  
  
 <**div flex**></**div**>  
  
 <**md-button ui-sref="song-liste"**>  
 Songs  
 </**md-button**>  
  
 <**div flex**></**div**>  
  
 <**auth**></**auth**>  
 </**div**>  
 </**md-toolbar**>  
  
 *<!-- Keine Interpreten -->* <**h3 ng-hide="$ctrl.seite.page.totalElements"**>Keine Interpreten gefunden</**h3**>  
  
 *<!-- Header -->* <**div class="header" layout="row" layout-align="start center" layout-padding  
 ng-show="$ctrl.seite.page.totalElements"**>  
 <**div flex="20"**>Interpret</**div**>  
 <**div flex**></**div**>  
 <**div flex="10"**></**div**>  
 </**div**>  
  
 *<!-- Interpreten-Liste -->* <**md-content flex layout="column"  
 ng-show\_="$ctrl.page.totalElements"**>  
 <**artist class="item"  
 ng-repeat="*artist* in $ctrl.seite.entities"  
 artist="*artist*"  
 loeschen="$ctrl.loeschen(*artist*)"**>  
 </**artist**>  
 </**md-content**>  
</**div**>

## Controller ArtistListe

**"use strict"**;  
  
***app***.**component**(**"artistListe"**, {  
 **templateUrl**: **"components/artist-liste.html"**,  
 **controller**: **"ArtistListeController"**,  
 **bindings**: {}  
});  
  
  
***app***.**config**(**function** ($stateProvider, $urlRouterProvider) {  
 $stateProvider.state({  
 **name**: **"artist-liste"**,  
 **url**: **"/artist-liste"**,  
 **component**: **"artistListe"** });  
});  
  
  
***app***.controller(**"ArtistListeController"**, **function** ($log, RestService, Artist) {  
  
 $log.**debug**(**"ArtistListeController()"**);  
  
  
 **this**.seiteLaden = (seitenNr) => {  
 $log.**debug**(**"ArtistListeController.seiteLaden()"**, seitenNr);  
  
 RestService.seiteLaden(Artist, seitenNr, { **size**: 7 })  
 .then(seite => {  
 $log.**debug**(**"ArtistListeController.seiteLaden() OK"**, seite);  
  
 **this**.**seite** = seite;  
 });  
 };  
  
  
 **this**.loeschen = (artist) => {  
 $log.**debug**(**"ArtistListeController.loeschen()"**, artist);  
  
 RestService  
 .loeschen(artist)  
 .then(() => {  
 $log.**debug**(**"ArtistListeController.loeschen() erfolgt"**, artist);  
  
 **this**.seiteLaden(**this**.**seite**.**page**.*number*);  
 });  
 };  
  
  
 **this**.seiteLaden(0);  
   
});

# Komponente Auth

Login- und Logout Formular

## Template Auth

<**style**>  
 .**auth md-input-container** {  
 **width**: 5**em**;  
 **margin**: 12**px** 0 16**px** 1**em**;  
 }  
  
 .**auth md-input-container** .**md-errors-spacer** {  
 **min-height**: 0 **!important**;  
 }  
  
 .**auth md-input-container input**,  
 .**auth md-input-container input**::**placeholder** {  
 **font-size**: 14**px**;  
 **color**: **rgba**(255, 255, 255, 0.87);  
 **border-color**: **rgba**(255, 255, 255, 0.87);  
 }  
</**style**>  
  
  
<**div class="auth"**>  
 <**form name="$ctrl.loginForm" layout="row" layout-align="start center" ng-hide="$ctrl.AuthService.username"**>  
 <**md-input-container md-no-float**>  
 <**input type="text" name="username" placeholder="Benutzer" ng-model="$ctrl.username" required**>  
 </**md-input-container**>  
  
 <**md-input-container md-no-float**>  
 <**input type="password" name="password" placeholder="Passwort" ng-model="$ctrl.password" required**>  
 </**md-input-container**>  
  
 <**md-button ng-click="$ctrl.login()" ng-disabled="$ctrl.loginForm.$invalid"**>  
 Anmelden  
 </**md-button**>  
 </**form**>  
  
 <**div layout="row" layout-align="start center" ng-show="$ctrl.AuthService.username"**>  
 <**md-button ng-click="$ctrl.logout()"**>  
 {{ $ctrl.**AuthService**.**username** }} abmelden  
 </**md-button**>  
 </**div**>  
</**div**>

## Controller Auth

**"use strict"**;  
  
***app***.**component**(**"auth"**, {  
 **templateUrl**: **"components/auth.html"**,  
 **controller**: **"AuthController"**,  
 **bindings**: {  
 **loginLogout**: **"&"** }  
});  
  
  
***app***.controller(**"AuthController"**, **function** ($log, AuthService) {  
  
 $log.**debug**(**"AuthController()"**);  
  
 *// Nur zum Testen* **this**.**username** = **"adria"**;  
 **this**.**password** = **"123"**;  
  
   
 **this**.**AuthService** = AuthService;  
  
 **this**.login = () => {  
 AuthService  
 .login(**this**.**username**, **this**.**password**)  
 .then(() => {  
 **delete this**.**username**;  
 **this**.**loginLogout**();  
 })  
 .finally(() => {  
 **delete this**.**password**;  
 });  
 };  
  
 **this**.logout = () => {  
 AuthService  
 .logout()  
 .then(() => {  
 **this**.loginForm.$setUntouched();  
 **this**.loginForm.$setPristine();  
 **this**.**loginLogout**();  
 });  
 };  
});

# Komponente FileContentChooser

Shadow Element für einen Upload / File Upload Button.

**"use strict"**;  
  
*/\*\*  
 \* <file-content-chooser> bietet Fähigkeiten, die über <input type="file"> hinausgehen.  
 \*  
 \* Er verwendet das innere HTML-Markup für seine Darstellung und speichert die Inhalte  
 \* der ausgewählten Dateien als Data-URL oder Array von Data-URLs in einem "ng-model".  
 \*  
 \* Dateien, die nicht gelesen werden konnten, werden als "error"-Objekt gespeichert.  
 \*  
 \* Verwendung:  
 \* <file-content-chooser ...>inneres HTML-Markup</file-content-chooser>  
 \*  
 \* Attribute (optional, wenn nicht anders angegeben):  
 \* ng-model (erforderlich) Controller-Property, in dem der Inhalt der ausgewählte(n)  
 \* Datei(en) als Data-URL bzw. als Array von Data-URLs zu speichern ist  
 \* ng-change AngularJS-Ausdruck, der bei Änderungen von "ng-model" ausgewertet wird  
 \* ng-disabled AngularJS-Ausdruck, der die Komponente sperrt; ändert die Darstellung des  
 \* inneren HTML nicht, d.h. dort sind evtl. zusätzliche Maßnahmen nötig.  
 \* name Name dieses Elements in einem Formular. Wird benötigt, wenn "required"  
 \* angegeben ist.  
 \* accept Lässt nur Dateien mit einer dieser MIME-Types auswählen  
 \* multiple Falls angegeben, kann man mehrere Dateien auswählen, und "ng-model" wird  
 \* an eine "FileList" gebunden (auch wenn nur eine einzige Datei ausgewählt ist);  
 \* andernfalls wird "ng-model" an ein einzelnes "File"-Objekt gebunden  
 \* required validiert im umgebenden Formular, dass eine Datei ausgewählt wurde. Benötigt  
 \* zusätzlich das "name"-Attribut.  
 \*/****app***.**component**(**"fileContentChooser"**, {  
 **template**: **`  
 <ng-transclude ng-click="$ctrl.open()" style="cursor: pointer;">file-content-chooser</ng-transclude>  
 <input type="file" style="display: none;">  
 `**,  
 **transclude**: **true**,  
 **require**: {  
 **ngModel**: **""** },  
 **bindings**: {  
 **ngDisabled**: **"<"**,  
 **name**: **"@"**,  
 **accept**: **"@"**,  
 **multiple**: **"@"**,  
 **required**: **"@"** },  
 **controller**: **"FileContentChooserController"**});  
  
  
***app***.controller(**"FileContentChooserController"**, **function**($element) {  
  
 **var** fileInput = $element.find(**"input"**);  
  
  
 **this**.$onInit = () => {  
 *// Attribute dem input-Element zuweisen* ***Object***.keys(**this**)  
 .filter(k => angular.*isString*(**this**[k]))  
 .forEach(k => fileInput.attr(k, **this**[k]));  
 };  
  
  
 *// Klicks an das input-Element weiterleiten* **this**.open = () => !**this**.ngDisabled && fileInput.**click**();  
  
  
 *// Auf Dateiauswahl reagieren* fileInput.**on**(**"change"**, () => {  
 **let** files = fileInput[0].**files**;  
 **if** (files **instanceof *FileList***) {  
 **var** promises = [];  
  
 *// Für jede Datei ein Promise auf die Beendigung des Lesens erzeugen* **for** (**let** i = 0; i < files.**length**; i++) {  
 promises.push(  
 **new *Promise***(**function**(resolve, reject) {  
 **let** reader = **new *FileReader***();  
 reader.**onload** = () => { files[i].**content** = reader.**result**; resolve(); };  
 reader.**onerror** = () => { files[i].**error** = reader.**error**; reject(); };  
 reader.readAsDataURL(files[i]);  
 })  
 );  
 }  
  
 *// ng-model erst aktualisieren, wenn alle Dateien gelesen wurden* ***Promise***.all(promises)  
 .finally(() => {  
 **let** viewValue;  
 **if** (angular.*isString*(**this**.**multiple**)) {  
 *// Aus der FileList ein Array von Data-URLs erzeugen* viewValue = [];  
 files.forEach(f => viewValue.push(f.**content** || f.**error**));  
 } **else** {  
 *// Einzelnen Data-URL übernehmen* viewValue = files[0].**content** || files[0].**error**;  
 }  
 **this**.**ngModel**.$setViewValue(viewValue);  
 });  
  
 } **else** {  
 **this**.**ngModel**.$setViewValue(**undefined**);  
 }  
 });  
  
});

# Komponente HTTPAktiv

Zeigt an, wenn der http-Service derzeit genutzt wird.

**"use strict"**;  
  
*/\*\*  
 \* HTTP-Aktivitätsanzeige: zeigt das innere HTML (oder "Beschäftigt...") an,  
 \* solange es unbeantwortete HTTP-Requests gibt.  
 \* Erfasst sowohl $http-Requests als auch das Laden von Komponenten.  
 \*  
 \* Verwendung:  
 \* <http-aktiv>inneres HTML-Markup</http-aktiv>  
 \*/****app***.**component**(**"httpAktiv"**, {  
 **template**: **"<div ng-show='$ctrl.aktiv()'><ng-transclude>Beschäftigt...</ng-transclude></div>"**,  
 **transclude**: **true**,  
 controller: **function**($http) {  
 **this**.aktiv = () => $http.**pendingRequests**.**length** > 0;  
 }  
});

# Komponente SeitenNav

Pagination Komponente.

## Template SeitenNav

<**md-button class="md-icon-button"  
 ng-disable="$ctrl.seite.istErste"  
 ng-click="$ctrl.seiteLaden({ seite: $ctrl.seite.erste })"**>  
 <**md-icon md-font-icon="material-icons"**>first\_page</**md-icon**>  
</**md-button**>  
  
<**md-button class="md-icon-button"  
 ng-disabled="$ctrl.seite.istErste"  
 ng-click="$ctrl.seiteLaden({ seite: $ctrl.seite.vorige })"**>  
 <**md-icon md-font-icon="material-icons"**>chevron\_left</**md-icon**>  
</**md-button**>  
  
<**span**>Seite {{ $ctrl.**seite**.**page**.*number* + 1 }} / {{ $ctrl.**seite**.**page**.totalPages }}</**span**>  
  
<**md-button class="md-icon-button"  
 ng-click="$ctrl.seiteLaden({ seite: $ctrl.seite.naechste })"  
 ng-disabled="$ctrl.seite.istLetzte"**>  
 <**md-icon md-font-icon="material-icons"**>chevron\_right</**md-icon**>  
</**md-button**>  
  
<**md-button class="md-icon-button"  
 ng-click="$ctrl.seiteLaden({ seite: $ctrl.seite.letzte })"**>  
 <**md-icon md-font-icon="material-icons"**>last\_page</**md-icon**>  
</**md-button**>

## Controller SeitenNav

**"use strict"**;  
  
***app***.**component**(**"seitenNav"**, {  
 **templateUrl**: **"components/seiten-nav.html"**,  
 **bindings**: {  
 **seite**: **"<"**,  
 **seiteLaden**: **"&"** }  
});

# Komponente Song

Zeigt einen einzelnen Song in Zeilenform an.

## Template Song

<**div class="song" flex="none"  
 layout="row" layout-align="start center" layout-padding**>  
  
 <**div flex="20"**>{{ $ctrl.**song**.**artist**.**name** }}</**div**>  
 <**div flex="10"**>  
 <**img ng-src="{{ $ctrl.song.cover }}"**>  
 </**div**>  
 <**div flex**>{{ $ctrl.**song**.**title** }}</**div**>  
 <**div class="center" flex="10"**>{{ $ctrl.**song**.**duration** \* 1000 | **date**:'mm:ss':'UTC' }}</**div**>  
 <**div flex="10"**>{{ $ctrl.**song**.uploader.**name** }}</**div**>  
 <**ng-transclude flex="10"**></**ng-transclude**>  
 <**div flex="10"**>  
 <**md-button class="md-fab md-mini md-primary"  
 ng-disabled="!$ctrl.auth.isAuthenticated($ctrl.song.uploader.name)"  
 ui-sref="song-editor({ song: $ctrl.song })"**>  
 <**md-icon md-font-icon="material-icons"**>edit</**md-icon**>  
 </**md-button**>  
  
 <**md-button class="md-fab md-mini"  
 ng-disabled="!$ctrl.auth.isAuthenticated($ctrl.song.uploader.name)"  
 ng-click="$ctrl.loeschen({ song: $ctrl.song })"**>  
 <**md-icon md-font-icon="material-icons"**>delete</**md-icon**>  
 </**md-button**>  
 </**div**>  
</**div**>

## Controller Song

**"use strict"**;  
  
***app***.**component**(**"song"**, {  
 **templateUrl**: **"components/song.html"**,  
 **controller**: **"SongController"**,  
 **bindings**: {  
 **song**: **"<"**,  
 **loeschen**: **"&"** },  
 **transclude**: **true**});  
  
***app***.controller(**"SongController"**, **function**($log, AuthService) {  
  
 **this**.**auth** = AuthService;  
  
});

# Komponente SongEditor

Frontend für die Editor-Ansicht eines Songs.

## Template SongEditor

<**style**>  
 .**song-editor img** {  
 **border**: 1**px solid #9e9e9e**;  
 }  
</**style**>  
  
  
<**div class="song-editor"**>  
 <**md-toolbar**>  
 <**div class="md-toolbar-tools" layout="row"**>  
 <**md-button class="md-icon-button"  
 ui-sref="song-liste"**>  
 <**md-icon md-font-icon="material-icons"**>arrow\_back</**md-icon**>  
 </**md-button**>  
  
 <**div flex**></**div**>  
  
 <**md-button class="md-icon-button"  
 ng-click="$ctrl.speichern()"  
 ng-disabled="$ctrl.formular.$invalid"**>  
 <**md-icon md-font-icon="material-icons"**>check</**md-icon**>  
 </**md-button**>  
 </**div**>  
 </**md-toolbar**>  
  
 <**form name="$ctrl.formular" layout="column" layout-padding**>  
 <**md-input-container**>  
 <**label**>Interpret</**label**>  
 *<!--  
 Achtung: bei 'trackBy' muss man ein 'unique'-Property der Entity angeben,  
 damit zu Beginn der richtige Eintrag ausgewählt ist  
 -->* <**md-select name="artist"  
 ng-model="$ctrl.song.artist"  
 ng-model-options="{ trackBy: '$value.name' }"  
 required**>  
 <**md-option ng-repeat="*a* in $ctrl.artists" ng-value="*a*"**>{{ ***a***.**name** }}</**md-option**>  
 </**md-select**>  
 <**div ng-messages="$ctrl.formular.artist.$error"**>  
 <**div ng-message="required"**>Wählen Sie einen Interpreten</**div**>  
 </**div**>  
 </**md-input-container**>  
  
 <**md-input-container**>  
 <**label**>Titel</**label**>  
 <**input type="text"  
 name="title"  
 ng-model="$ctrl.song.title"  
 required**>  
 <**div ng-messages="$ctrl.formular.title.$error"**>  
 <**div ng-message="required"**>Ein Songtitel wird benötigt</**div**>  
 </**div**>  
 </**md-input-container**>  
  
 <**div layout="row" layout-align="start center"**>  
 <**file-content-chooser ng-model="$ctrl.song.cover"  
 accept="image/\*"**>  
 <**md-button class="md-raised md-primary"**>  
 Coverbild festlegen  
 </**md-button**>  
 </**file-content-chooser**>  
  
 <**img ng-src="{{ $ctrl.song.cover }}"**>  
 </**div**>  
  
 <**div layout="row" layout-align="start center"**>  
 <**file-content-chooser ng-model="$ctrl.song.audio"  
 accept="audio/\*"**>  
 <**md-button class="md-raised md-primary"**>  
 Song hochladen  
 </**md-button**>  
 </**file-content-chooser**>  
  
 <**span**>Bytes: {{ $ctrl.**song**.**audio**.**length** | **number**:0 }}</**span**>  
 </**div**>  
  
 <**md-input-container**>  
 <**label**>Spieldauer</**label**>  
 <**input type="number" name="duration" ng-model="$ctrl.song.duration" min="1"**>  
 <**div ng-messages="$ctrl.formular.duration.$error"**>  
 <**div ng-message="min"**>Die Spieldauer muss positiv sein</**div**>  
 </**div**>  
 </**md-input-container**>  
 </**form**>  
</**div**>

## Controller SongEditor

**"use strict"**;  
  
***app***.**component**(**"songEditor"**, {  
 **templateUrl**: **"components/song-editor.html"**,  
 **controller**: **"SongEditorController"**,  
 **bindings**: {}  
});  
  
  
***app***.**config**(**function** ($stateProvider, $urlRouterProvider) {  
 $stateProvider.state({  
 **name**: **"song-editor"**,  
 **params**: { **song**: **null** },  
 **component**: **"songEditor"** });  
});  
  
  
***app***.controller(**"SongEditorController"**, **function** ($state, $stateParams, $log, RestService, Artist, Song) {  
  
 $log.**debug**(**"SongEditorController()"**, $stateParams.**song**);  
  
 *// Verfügbare Interpreten laden* RestService  
 .seiteLaden(Artist, 0, { **size**: 9999 })  
 .then(seite => {  
 $log.**debug**(**"SongEditorController.seiteLaden() OK"**, seite);  
 **this**.**artists** = seite.**entities**;  
 });  
  
 *// \_Veränderbare\_ Song-Kopie erzeugen* **this**.**song** = ***Object***.assign({}, $stateParams.**song**);  
  
  
 **this**.speichern = () => {  
 $log.**debug**(**"SongEditorController.speichern()"**, **this**.**song**);  
  
 RestService  
 .speichern(**new** Song(**this**.**song**))  
 .then(song => {  
 $log.**debug**(**"SongEditorController.speichern() OK"**, song);  
  
 $state.go(**"song-liste"**);  
 });  
 };  
  
});

# Komponente SongListe

Frontend für die Listenansicht der Songs.

## Template SongListe

<**div class="song-liste liste" layout="column" layout-fill**>  
 <**md-toolbar**>  
 <**div class="md-toolbar-tools"**>  
 <**h3**>Songs</**h3**>  
  
 <**http-aktiv**>  
 <**md-progress-circular md-mode="indeterminate"  
 md-diameter="30px"  
 class="md-accent"**>  
 </**md-progress-circular**>  
 </**http-aktiv**>  
  
 <**seiten-nav seite="$ctrl.seite"  
 seite-laden="$ctrl.seiteLaden(seite)"  
 ng-show="$ctrl.seite.page.totalPages"**>  
 </**seiten-nav**>  
  
 <**div flex**></**div**>  
  
 <**md-button ui-sref="artist-liste"**>  
 Interpreten  
 </**md-button**>  
  
 <**div flex**></**div**>  
  
 <**auth login-logout="$ctrl.loginLogout()"**></**auth**>  
 </**div**>  
 </**md-toolbar**>  
  
 <**div layout="row" layout-margin**>  
 <**md-input-container**>  
 <**label**>Nach Interpreten filtern</**label**>  
 <**input type="text"  
 ng-model="$ctrl.artistText"  
 ng-model-options="{ debounce: 300 }"  
 ng-change="$ctrl.seiteLaden(0)"**>  
 </**md-input-container**>  
  
 <**md-input-container**>  
 <**label**>Nach Titeln filtern</**label**>  
 <**input type="text"  
 ng-model="$ctrl.titleText"  
 ng-model-options="{ debounce: 300 }"  
 ng-change="$ctrl.seiteLaden(0)"**>  
 </**md-input-container**>  
 </**div**>  
  
 *<!-- Keine Songs -->* <**h3 ng-hide="$ctrl.seite.page.totalElements"**>Keine Songs gefunden</**h3**>  
  
 *<!-- Header -->* <**div class="header" layout="row" layout-align="start center" layout-padding  
 ng-show="$ctrl.seite.page.totalElements"**>  
 <**div flex="20"**>Interpret</**div**>  
 <**div class="center" flex="10"**>Cover</**div**>  
 <**div flex**>Titel</**div**>  
 <**div class="center" flex="10"**>Spieldauer</**div**>  
 <**div class="center" flex="10"**>Uploader</**div**>  
 <**div flex="10"**></**div**>  
 <**div flex="10"**>  
 <**md-button class="md-fab md-mini md-primary"  
 ui-sref="song-editor"**>  
 <**md-icon md-font-icon="material-icons"**>add</**md-icon**>  
 </**md-button**>  
 </**div**>  
 </**div**>  
  
 *<!-- Song-Liste -->* <**md-content flex layout="column"  
 ng-show\_="$ctrl.page.totalElements"**>  
 <**song class="item"  
 ng-repeat="*song* in $ctrl.seite.entities"  
 song="*song*"  
 loeschen="$ctrl.loeschen(*song*)"**>  
  
 *<!-- Abspielen/Anhalten-Buttons -->* <**md-button class="md-fab md-mini md-primary"  
 ng-hide="$ctrl.nowPlaying === *song*"  
 ng-click="$ctrl.abspielen(*song*)"**>  
 <**md-icon md-font-icon="material-icons"**>play\_arrow</**md-icon**>  
 </**md-button**>  
  
 <**md-button class="md-fab md-mini md-warn"  
 ng-show="$ctrl.nowPlaying === *song*"  
 ng-click="$ctrl.abspielenBeenden()"**>  
 <**md-icon md-font-icon="material-icons"**>stop</**md-icon**>  
 </**md-button**>  
 </**song**>  
 </**md-content**>  
</**div**>

## Controller SongListe

**"use strict"**;  
  
app.component(**"songListe"**, {  
 templateUrl: **"components/song-liste.html"**,  
 controller: **"SongListeController"**,  
 bindings: {}  
});  
  
  
app.config(**function** ($stateProvider, $urlRouterProvider) {  
 $stateProvider.state({  
 name: **"song-liste"**,  
 url: **"/song-liste"**,  
 component: **"songListe"** });  
  
 $urlRouterProvider.otherwise(**"/song-liste"**);  
});  
  
  
app.controller(**"SongListeController"**, **function** ($http, $timeout, $log, Song, RestService, AuthService) {  
  
 $log.debug(**"SongListeController()"**);  
  
 **let** player;  
  
 **this**.titleText = **""**;  
 **this**.artistText = **""**;  
  
  
 **this**.seiteLaden = (seitenNr) => {  
 $log.debug(**"SongListeController.seiteLaden()"**, seitenNr);  
  
 *// Den gerade laufenden Song beenden, wenn nötig* **this**.abspielenBeenden();  
  
 RestService.seiteLaden(  
 Song,  
 seitenNr,  
 {  
 size: 11,  
 projection: **"ohneAudio"**,  
 titleText: **this**.titleText,  
 artistText: **this**.artistText,  
 },  
 **"findByArtistNameContainingIgnoreCaseAndTitleContainingIgnoreCaseOrderByTitle"** )  
 .then(seite => {  
 $log.debug(**"SongListeController.seiteLaden() OK"**, seite);  
  
 **this**.seite = seite;  
 });  
 };  
  
  
 **this**.loeschen = (song) => {  
 $log.debug(**"SongListeController.loeschen()"**, song);  
  
 RestService  
 .loeschen(song)  
 .then(() => {  
 $log.debug(**"SongListeController.loeschen() erfolgt"**, song);  
  
 **this**.seiteLaden(**this**.seite.page.number);  
 });  
 };  
  
  
 **this**.abspielen = (song) => {  
 $log.debug(**"SongListeController.abspielen()"**, song);  
  
 *// Audiotrack für diesen Song schon geladen?* **if** (!song.audio) {  
 RestService  
 .laden(Song, song.\_links.self.href, { projection: **"nurAudio"** })  
 .then(s => {  
 $log.debug(**"SongListeController.abspielen() geladen"**, s);  
  
 **if** (s.audio) {  
 *// Audiodatei speichern und das Ganze nochmals versuchen* song.audio = s.audio;  
 **this**.abspielen(song);  
 }  
 });  
  
 } **else** {  
 $log.debug(**"SongListeController.abspielen() gestartet"**, song);  
  
 *// Den gerade laufenden Song beenden, wenn nötig* **this**.abspielenBeenden();  
  
 *// Song, der nun abgespielt wird (steuert Sichtbarkeit der Buttons)* **this**.nowPlaying = song;  
  
 *// Audioplayer erzeugen und Song abspielen* player = **new *Audio***(song.**audio**);  
  
 *// Wenn ein Song bis zum Ende abgespielt wurde, dann so behandeln  
 // wie beim manuellen Beenden des Abspielens* player.addEventListener(**"ended"**, () => {  
 **this**.abspielenBeenden();  
  
 *// AngularJS "aufwecken", damit das GUI aktualisiert wird* $timeout();  
 });  
 player.play();  
 }  
 };  
  
  
 **this**.abspielenBeenden = () => {  
 **if** (player) {  
 *// Player anhalten und entsorgen* player.pause();  
 player = **null**;  
  
 *// Sichtbarkeit der Buttons steuern* **delete this**.nowPlaying;  
 }  
 };  
  
  
 *// Abspielen beenden, sobald diese Komponente nicht mehr sichtbar ist* **this**.$onDestroy = () => **this**.abspielenBeenden();  
  
  
 **this**.loginLogout = () => {  
 **if** (AuthService.username) {  
 **this**.seiteLaden(0);  
  
 } **else** {  
 **this**.abspielenBeenden();  
 **delete this**.seite;  
 }  
 };  
  
  
 **this**.loginLogout();  
  
});

# Model Artist

Stellt die Fachklasse Artist in Angular dar. Wird mit den IntelliJ-Dateivorlagen erstellt.

**"use strict"**;  
  
***app***.**factory**(**"Artist"**, **function** () {  
  
 **function** *Artist*(template, modifier) {  
  
 *// Schreibgeschützte Properties und ihre Defaultwerte* **let** properties = {  
 **name**: **undefined**,  
 **\_links**: {}  
 };  
  
 ***Object***.assign(**this**, properties, template, modifier);  
  
 *// Properties schreibschützen* ***Object***.keys(properties).forEach(k => ***Object***.defineProperty(**this**, k, {**writable**: **false**}));  
  
 *// Liefert eine neue Instanz dieses Objekts mit den angegebenen Änderungen* **this**.variante = modifier => **new** *Artist*(**this**, modifier);  
 }  
  
 *// Relativer Pfad im REST-API  
 Artist*.**path** = **"artists"**;  
  
 **return** *Artist*;  
});

# Model Seite

Stellt eine Seite eines beliebigen Objekts in Angular dar.

**"use strict"**;  
  
***app***.**factory**(**"Seite"**, **function** () {  
  
 **function** *Seite*(konstruktor, data, modifier) {  
  
 *// Schreibgeschützte Properties und ihre Defaultwerte* **let** properties = {  
 **page**: {},  
 **entities**: [],  
 };  
  
 ***Object***.assign(**this**, properties, data, modifier, { **\_embedded**: **undefined** });  
  
 *// Anonyme Objekte in den gewünschten Datentyp umwandeln* **this**.**entities** = data[konstruktor.**path**]  
 .map(obj => **new** konstruktor(obj));  
  
 *// Hilfsvariable erzeugen* **this**.**vorige** = **this**.**page**.*number* - 1;  
 **this**.**naechste** = **this**.**page**.*number* + 1;  
 **this**.**erste** = 0;  
 **this**.**letzte** = **this**.**page**.totalPages-1;  
  
 **this**.**istErste** = **this**.**page**.*number* <= **this**.**erste**;  
 **this**.**istLetzte** = **this**.**page**.*number* >= **this**.**letzte**;  
  
 *// Properties schreibschützen* ***Object***.keys(properties).forEach(k => ***Object***.defineProperty(**this**, k, {**writable**: **false**}));  
  
 *// Liefert eine neue Instanz dieses Objekts mit den angegebenen Änderungen* **this**.variante = modifier => **new** *Seite*(**this**, modifier);  
 }  
  
 **return** *Seite*;  
});

# Model Song

Stellt die Fachklasse Song in Angular dar.

**"use strict"**;  
  
***app***.**factory**(**"Song"**, **function** (Artist) {  
  
 **function** *Song*(template, modifier) {  
  
 *// Schreibgeschützte Properties und ihre Defaultwerte* **let** properties = {  
 **title**: **undefined**,  
 **artist**: **undefined**,  
 **cover**: **undefined**,  
 **duration**: **null**,  
 **\_links**: {}  
 };  
  
 ***Object***.assign(**this**, properties, template, modifier);  
  
 *// Aus dem anonymen Objekt ein Artist-Objekt machen* **this**.**artist** = **new** Artist(**this**.**artist**);  
  
 *// Properties schreibschützen* ***Object***.keys(properties).forEach(k => ***Object***.defineProperty(**this**, k, {**writable**: **false**}));  
  
 *// Liefert eine neue Instanz dieses Objekts mit den angegebenen Änderungen* **this**.variante = modifier => **new** *Song*(**this**, modifier);  
 }  
  
 *// Relativer Pfad im REST-API  
 Song*.**path** = **"songs"**;  
  
 **return** *Song*;  
});

# Service Auth

Kümmert sich um den Authentifizierungs-Prozess in Angular.

**"use strict"**;  
  
*/\*\*  
 \* Verwaltet An- und Abmeldungen von Benutzern dieser App am Server.  
 \*  
 \* Siehe die Methoden AuthService.login(...), AuthService.logout()  
 \* sowie die Variable AuthService.username.  
 \*/****app***.**service**(**"AuthService"**, **function**($http, $log) {  
  
 $log.**debug**(**"AuthService()"**);  
  
 **const** LOGIN = **"/login"**,  
 LOGOUT = **"/logout"**,  
 TOKEN\_HEADER = **"x-auth-token"**;  
  
  
 *// Name des angemeldeten Benutzers, oder undefined, wenn kein Benutzer angemeldet ist* **this**.**username** = **undefined**;  
  
  
 */\*\*  
 \* Setzt die Authentifizierungsdaten für einen Benutzer oder löscht sie,  
 \* wenn keine Argumente angegeben sind.  
 \*/* **let** self = **this**;  
 **function** *authentifizieren*(username, token) {  
 self.**username** = username;  
 $http.**defaults**.**headers**.**common**[TOKEN\_HEADER] = token;  
 }  
  
  
 */\*\*  
 \* Versucht den angegebenen Benutzer am Server anzumelden und liefert  
 \* ein Promise auf die Beendigung des Vorgangs.  
 \*/* **this**.login = (username, password) => {  
 **return** $http  
 .post(LOGIN, {}, { **params**: { **username**: username, **password**: password } })  
 .then(response => {  
 $log.**debug**(**`AuthService.login("**${username}**"): OK, Token =** ${response.headers(TOKEN\_HEADER)}**`**);  
  
 *authentifizieren*(username, response.headers(TOKEN\_HEADER));  
 **return *Promise***.resolve(response);  
 })  
 .catch(response => {  
 $log.**error**(**"AuthService.login(): Fehler"**, response);  
  
 *authentifizieren*();  
 **return *Promise***.reject(response);  
 });  
 };  
  
  
 */\*\*  
 \* Meldet den gerade angemeldeten Benutzer wieder vom Server ab,  
 \* liefert ein Promise auf die Beendigung des Vorgangs.  
 \*/* **this**.logout = () => {  
 **return** $http  
 .post(LOGOUT)  
 .then(response => {  
 $log.**debug**(**"AuthService.logout(): OK"**);  
 **return *Promise***.resolve(response);  
 })  
 .catch(response => {  
 $log.**error**(**"AuthService.logout(): Fehler"**, response);  
 **return *Promise***.reject(response);  
 })  
 .finally(() => {  
 *authentifizieren*();  
 });  
 };  
  
  
 */\*\*  
 \* Liefert true, wenn der angegebene User gerade angemeldet ist.  
 \*/* **this**.isAuthenticated = (username) => {  
 **return this**.**username** && username && username.toLowerCase() === **this**.**username**.toLowerCase();  
 };  
  
});

# Service RestService

Kümmert sich um jegliche REST API Abfragen in der Applikation.

**"use strict"**;  
  
***app***.**service**(**"RestService"**, **function** ($mdToast, $http, $log, Seite) {  
  
 $log.**debug**(**"RestService()"**);  
  
 **const** API\_PFAD = **"/api/"** */\*\*  
 \* Liefert ein Promise auf eine Seite von Entities der  
 \* angegebenen Type. Existiert die Seite nicht, so wird  
 \* die letzte vorhandene Seite geliefert.  
 \*  
 \* Alle \_embedded-Objekte werden durch ihre Inhalte ersetzt.  
 \*  
 \* Argumente (optional, wenn nicht anders angegeben):  
 \* konstruktor (erforderlich) Factoryfunktion für geladene  
 \* Objekte, liefert auch ihren Pfad im REST-API  
 \* seitenNr (erforderlich) Nummer der zu ladenden Seite  
 \* (erste Seite === 0)  
 \* parameter Namen und Werte der Request-Parameter als  
 \* Objekt  
 \* query Name der serverseitigen Query-Methode, falls  
 \* eine solche verwendet werden soll  
 \*/* **this**.seiteLaden = (konstruktor, seitenNr, parameter, query) => {  
 $log.**debug**(**`RestService.seiteLaden("**${konstruktor.**path**}**",** ${seitenNr}**,** ${parameter}**)`**);  
  
 *// REST-Pfad und Query-Parameter vorbereiten* **let** pfad = query  
 ? **`**${API\_PFAD}${konstruktor.**path**}**/search/**${query}**`** : **`**${API\_PFAD}${konstruktor.**path**}**`**,  
 params = angular.*extend*({ **page**: seitenNr }, parameter);  
  
 **return** $http  
 .get(pfad, { **params**: params })  
 .then(response => {  
 $log.**debug**(**"RestService.seiteLaden() OK"**, response);  
  
 *// Seitennummer im zulässigen Bereich, oder keine Seiten?* **if** (response.**data**.**page**.*number* < response.**data**.**page**.totalPages || !response.**data**.**page**.totalElements) {  
 *// OK, Seite erzeugen und zurückgeben* **return new** Seite(konstruktor, response.**data**);  
  
 } **else** {  
 *// Letzte vorhandene Seite ausliefern* **return this**.seiteLaden(  
 konstruktor,  
 response.**data**.**page**.totalPages-1,  
 params,  
 query);  
 }  
 })  
 .catch(*fehlerBehandeln*);  
 };  
  
  
 */\*\*  
 \* Lädt über das REST-API vom Server eine einzelne Entity der  
 \* angegebenen Type und liefert ein Promise auf das geladene Objekt.  
 \*  
 \* Argumente (optional, wenn nicht anders angegeben):  
 \* konstruktor (erforderlich) Factoryfunktion für geladene  
 \* Objekte, liefert auch ihren Pfad im REST-API  
 \* url (erforderlich) Link zur Entity  
 \* parameter Namen und Werte der Request-Parameter als  
 \* Objekt  
 \*/* **this**.laden = (konstruktor, url, parameter) => {  
 $log.**debug**(**`RestService.laden("**${konstruktor.**path**}**",** ${url}**)`**);  
  
 **return** $http  
 .get(url, { **params**: parameter })  
 .then(response => {  
 $log.**debug**(**`RestService.laden() OK`**, response);  
  
 response.**data** = **new** konstruktor(response.**data**);  
 **return** response.**data**;  
 })  
 .catch(*fehlerBehandeln*);  
 };  
  
  
 */\*\*  
 \* Löscht die angegebene Entity von Server und liefert ein  
 \* Promise auf den Erfolg.  
 \*/* **this**.loeschen = (entity) => {  
 $log.**debug**(**"RestService.loeschen()"**, entity);  
  
 *// Stammt die Entity vom Server, oder wurde sie lokal erzeugt?* **if** (entity.**\_links** && entity.**\_links**.**self**) {  
 **return** $http  
 .delete(entity.**\_links**.**self**.**href**)  
 .catch(*fehlerBehandeln*);  
  
 } **else** {  
 *// Entity stammt nicht vom Server und kann dort nicht gelöscht werden  
 fehlerBehandeln*({ **status**: 404, **statusText**: **"Not found"**, **data**: {} });  
 **return *Promise***.reject();  
 }  
 };  
  
  
 */\*\*  
 \* Aktualisiert oder erzeugt die angegebene Entity auf dem Server,  
 \* je nachdem, ob sie bereits vom Server stammt oder nicht. Liefert  
 \* ein Promise auf die gespeicherte Entity.  
 \*/* **this**.speichern = (entity) => {  
 *// Stammt die Entity vom Server, oder wurde sie lokal erzeugt?* **if** (entity.**\_links** && entity.**\_links**.**self**) {  
 *// Entity wurde schon einmal vom Server geladen, aktualisieren* $log.**debug**(**"RestService.speichern(): update"**, entity);  
  
 **return** $http  
 .patch(  
 entity.**\_links**.**self**.**href**,  
 entity,  
 { **headers**: { **"If-Match"**: entity.**etag** } })  
 .then(response => {  
 $log.**debug**(**"RestService.speichern(): update OK"**, response);  
  
 *// Aktualisierten Satz in eine Entity umwandeln* **return new** entity.**constructor**(response.**data**);  
 })  
 .catch(*fehlerBehandeln*);  
  
 } **else** {  
 *// Entity wurde noch nie auf dem Server gespeichert, erzeugen* $log.**debug**(**"RestService.speichern(): insert"**, entity);  
  
 **return** $http  
 .post(**`**${API\_PFAD}${entity.**constructor**.**path**}**`**, entity)  
 .then(response => {  
 $log.**debug**(**"RestService.speichern(): insert OK"**, response);  
  
 *// Neuen Satz in eine Entity umwandeln* **return new** entity.**constructor**(response.**data**);  
 })  
 .catch(*fehlerBehandeln*);  
 }  
 };  
  
  
 */\*\*  
 \* Zeigt den Fehlercode in einem Toast und liefert ein  
 \* zurückgewiesenes Promise.  
 \*/* **function** *fehlerBehandeln*(response) {  
 $log.**error**(**"RestService::fehlerBehandeln()"**, response);  
  
 $mdToast.showSimple(**`Fehler** ${response.**status**}**`**);  
 **return *Promise***.reject();  
 }  
  
  
 */\*\*  
 \* Ersetzt in den Response-Daten rekursiv alle \_embedded-Objekte durch ihre Inhalte.  
 \*/* **function** *embeddedAufloesen*(obj) {  
 *// Liegt ein Array vor?* **if** (angular.*isArray*(obj)) {  
 *// \_embedded-Objekte in den Arrayelementen auflösen* obj.forEach(*embeddedAufloesen*);  
  
 *// Liegt ein Objekt mit einem \_embedded-Property vor?* } **else if** (angular.*isObject*(obj) && obj.**\_embedded**) {  
 *// Inhalte von \_embedded direkt im Objekt platzieren* ***Object***.keys(obj.**\_embedded**).forEach(k => {  
 obj[k] = obj.**\_embedded**[k];  
 *embeddedAufloesen*(obj[k]);  
 })  
 **delete** obj.**\_embedded**;  
 }  
  
 **return** obj;  
 }  
  
  
 */\*\*  
 \* Ersetzt in den Request-Daten alle Entities durch ihre self-Links.  
 \*/* **function** *entitiesVerlinken*(obj) {  
 *// Liegt ein Array vor?* **if** (angular.*isArray*(obj)) {  
 *// In Arrayelementen ersetzen* obj.forEach(*entitiesVerlinken*);  
  
 *// Liegt ein Objekt vor?* } **else if** (angular.*isObject*(obj)) {  
 *// Verlinkte Objekte suchen und durch ihre self-Links ersetzen* ***Object***.keys(obj).forEach(k => {  
 **if** (obj[k] && obj[k].**\_links** && obj[k].**\_links**.**self**) {  
 *// Objekt durch self-Link ersetzen und Templates aus dem Link entfernen* obj[k] = obj[k].**\_links**.**self**.**href**.replace(/\{.\*\}$/, **""**);  
 }  
 });  
 }  
  
 **return** obj;  
 }  
  
  
 *// embeddedAufloesen() automatisch zuerst nach jeder Response anwenden* $http.**defaults**.**transformResponse**.**push**(*embeddedAufloesen*);  
  
 *// entitiesVerlinken() automatisch zuletzt vor jedem Request auf eine Request-Kopie anwenden* $http.**defaults**.**transformRequest**.unshift(requestData => {  
 **return** *entitiesVerlinken*(angular.*copy*(requestData));  
 });  
  
});