

# Programmieraufgabe

Anwendungssysteme

Prof. Dr. S. Tai,  
Dominik Ernst, Jacob Eberhardt

SS 2018

Sebastian Führ

Stefan Yovchev

Nikolay Nikolov

383112

394552

394538

<https://github.com/AnwendungSystemeTUBerlin/TeamDirectory>

# Quellcode TeamDirectory

# ApplicationInit

Diese Klasse wird beim Starten des Servers erstellt und kann aufgrund der Singleton Annotation auch kein zweites mal instanziiert werden.

Wenn die Instanz erstellt wurde, wird die Methode `ensureUsersExist()` aufgrund der Annotation `@PostConstruct` aufgerufen. Diese erstellt User Entitäten und Comment Entitäten, welche von nun an in der Datenbank gespeichert sind, bis diese das nächste mal heruntergefahren wird.

```
12 @Singleton
13 @Startup
14 public class ApplicationInit {
15     @PersistenceContext
16     EntityManager em;
17
18     @PostConstruct
19     public void ensureUsersExist() {
20         //Stefan
21         User stefan = new User("Unbekannt", "Stefan");
22         stefan.setUniversity("TU Berlin");
23         stefan.setStudyCourse("Wirtschaftsinformatik");
24         stefan.setAge(0);
25         em.persist(stefan);
26
27         //Sebastian Führ
28         User sebastian = new User("Führ", "Sebastian");
29         sebastian.setUniversity("TU Berlin");
30         sebastian.setStudyCourse("Wirtschaftsinformatik");
31         sebastian.setAge(19);
32         em.persist(sebastian);
33
34         Comment newComment = new Comment(stefan.getId(), sebastian.getId(), "Das Wetter ist schön.");
35         newComment.setDate("13.06.2018");
36         newComment.setTime("10:15");
37         em.persist(newComment);
38
39         newComment = new Comment(sebastian.getId(), stefan.getId(), "Das Wetter ist schlecht.");
40         newComment.setDate("15.06.2018");
41         newComment.setTime("18:43");
42         em.persist(newComment);
43     }
44 }
```

# Entität

## User

Beim Erstellen einer Instanz von User, wird automatisch eine ID generiert. Falls der User jedoch bereits in der Datenbank gespeichert wurde, ist es durch den dritten Konstruktor möglich, die ID der in der Datenbank gespeicherten Entität zu übernehmen und sie der erstellten Instanz zuzuweisen.

# Entitybeans

```
10 @Entity
11 @Table(name = "USERTABLE")
12 public class User implements Serializable {
13     private static final long serialVersionUID = -577432887255987479L;
14     @Id
15     @GeneratedValue
16     int id;
17     String name,
18         surname,
19         university,
20         studyCourse;
21     int userAge;
22
23     public User() {
24         super();
25     }
26
27     public User(String name, String surname) {
28         this.name = name;
29         this.surname = surname;
30     }
31
32     public User(String name, String surname, int userId) {
33         this.name = name;
34         this.surname = surname;
35         this.id = userId;
36     }
```

# Entität

## Comment

Die Entität `Comment` wurde angelegt, um die von Nutzern erstellten Kommentare in der Datenbank speichern zu können. Bei der Erstellung einer Instanz von `Comment` wird die Systemzeit mithilfe der Methode `initDateAndTime()` erfasst und in einem in Deutschland üblichen Format als String gespeichert.

Wir haben uns aus Gründen der Skalierbarkeit dafür entschieden, das Datum und die Uhrzeit in zwei verschiedenen Strings zu speichern. So kann bei Bedarf einer der beiden Strings direkt ausgelesen werden, ohne ihn erst mithilfe einer Methode wie z.B. `split()` auftrennen zu müssen.

```
14 @Entity
15 @Table(name = "COMMENTSTABLE")
16 public class Comment implements Serializable {
17     private static final long serialVersionUID = 384032570455381853L;
18     @Id
19     @GeneratedValue
20     int commentID;
21     int receiverID;
22     String date,
23         time,
24         content;
25     int posterID;
26
27     public Comment() {
28         super();
29     }
30
31     public Comment(int receiverID, int posterID, String content) {
32         this.receiverID = receiverID;
33         this.posterID = posterID;
34         this.content = content;
35         initDateAndTime();
36     }
```

```
43     public void initDateAndTime() {
44         Calendar calender = GregorianCalendar.getInstance();
45         Date dateCal = calender.getTime();
46         SimpleDateFormat formatter = new SimpleDateFormat("dd.MM.yyyy");
47         date = formatter.format(dateCal);
48         formatter = new SimpleDateFormat("HH:mm");
49         time = formatter.format(dateCal);
50     }
```

## UserManagement

Die Methode `getUsers()` gibt alle in der Datenbank vorhandenen User als Liste zurück.

`getUser(int userID)` durchsucht die Datenbank nach einem User mit der übergebenen ID und gibt diesen zurück, sobald er gefunden wurde.

```
30 import java.util.List;
12
13 /**
14  * EJB -- Stateless Session Bean
15  *
16  */
17 @Stateless
18 public class UserManagement {
19
20     @PersistenceContext
21     EntityManager em;
22
23     @EJB
24     ApplicationInit userInit;
25
26     /**
27      * Returns all Users which are currently saved in the database.
28      *
29      * @return List of User
30      */
31     public List<User> getUsers() {
32         return em.createQuery("SELECT u FROM User u", User.class).getResultList();
33     }
34
35     /**
36      * Returns a specific User with the given ID.
37      *
38      * @param userID
39      * @return User
40      */
41     public User getUser(int userID) {
42         try {
43             return em.createQuery(String.format("SELECT u FROM User u WHERE u.id = %d", userID), User.class).getSingleResult();
44         } catch (NoResultException e) {
45             return null;
46         }
47     }
48 }
```

# UserManagement

Die abgebildete Methode durchsucht die Datenbank nach einem User mit dem übergebenen Namen (`name`) und Vornamen (`surname`). Sobald ein Benutzer mit passenden Parametern gefunden ist, wird dieser zurückgegeben.

Wird keine passende User Entität gefunden, wird ein neuer User mit entsprechendem Namen und Vornamen erstellt, seine anderen Parameter auf -1 (`age`) gesetzt oder leer gelassen (`university`, `studyCourse`) und schließlich zurückgegeben.

```
48
49 /**
50  * Validates if there exists a User in the database with the given name
51  * and surname and returns this User. If not, creates a new User with
52  * the given parameters and sets the other attributes university,
53  * studyCourse and age to "", "", and -1.
54  *
55  * @param name
56  * @param surname
57  * @return User
58  */
59 public User getUser(String name, String surname) {
60     try {
61         return em.createQuery(String.format("SELECT u FROM User u WHERE u.name = \"%s\" AND u.surname = \"%s\"", name, surname), User.class).getSingleResult();
62     } catch (NoResultException e) {
63         User user = new User(name, surname);
64         user.setAge(-1);
65         user.setUniversity("");
66         user.setStudyCourse("");
67
68         em.persist(user);
69
70         return user;
71     }
72 }
73
74
```

## CommentsManagement

Diese *Stateless Session Bean* stellt mehrere Methoden zur Verwaltung der in der Datenbank gespeicherten oder zu speichernden Kommentare bereit.

Die Methode `saveComment ()` speichert die übergebene Kommentar-Entität in der Datenbank.

```
15 @Stateless
16 public class CommentsManagement {
17
18     @PersistenceContext
19     EntityManager em;
20
21     * Returns all Comments which are currently saved in the database.
22     public List<Comment> getComments() {
23         return em.createQuery("SELECT c FROM Comment c", Comment.class).getResultList();
24     }
25
26     * Creates and persists a new comment in the database
27     public void saveComment(Comment comment) {
28         Comment c = new Comment(comment.getReceiverID(), comment.getPosterID(), comment.getContent());
29
30         em.persist(c);
31     }
32 }
```



# CommentsManagement

Die Methode `getCommentsByPosterID()` gibt alle in der Datenbank vorhandenen Kommentare als Liste zurück, welche von dem User mit der übergebenen `posterID` gepostet wurden.

Äquivalent dazu funktioniert die Methode `getCommentsByReceiverID()` für die Kommentare, welche an einen User gerichtet wurden.

```
42- /**
43-  * Returns a list of Comments from the database which where posted by the User
44-  * with the given attribute posterID.
45-  *
46-  * @param posterID
47-  * @return List of Comments.
48-  */
49- public List<Comment> getCommentsByPosterID(int posterID) {
50-     try {
51-         return em.createQuery(String.format("SELECT c, u.name FROM Comment c, User u WHERE c.posterID = %d", posterID), Comment.class).getResultList();
52-     } catch (NoResultException e) {
53-         return null;
54-     }
55- }
56-
57- /**
58-  * Returns a list of Comments from the database which where posted to the
59-  * User with the given attribute receiverID.
60-  *
61-  * @param posterID
62-  * @return List of Comments.
63-  */
64- public List<Comment> getCommentsByReceiverID(int receiverID) {
65-     try {
66-         return em.createQuery(String.format("SELECT c, u FROM Comment c, User u WHERE c.receiverID = %d AND u.id = c.posterID", receiverID), Comment.class).getResultList();
67-     } catch (NoResultException e) {
68-         return null;
69-     }
70- }
71- }
72- }
```

# Quellcode Web

# UserService

Hier können durch HTTP-Nachrichten bestimmte Methoden ausgeführt werden.

Die erste `getUserAsJson()` gibt den User mithilfe der `UserManagement`-Klasse aus der Datenbank zurück, wobei wir Query Parameter benutzt haben um den Vor- und Nachnamen an den Server zu schicken.

Die zweite Methode mit gleichem Namen gibt den Benutzer auch mithilfe von `UserManagement` zurück, aber sucht diesen anhand der `userID` aus der Datenbank heraus.

```
17 @Path("/user")
18 public class UserService {
19
20     @EJB
21     private UserManagement userMgmt;
22
23     public UserService() {
24     }
25
26     * Retrieves representation of an user with the specific name and surname. If
27
28     @GET
29     @Produces("application/json")
30     public String getUserAsJson(@QueryParam("name") String name,
31                                @QueryParam("surname") String surname) {
32         Jsonb jsonb = JsonbBuilder.create();
33         if (name != null || surname != null) {
34             return jsonb.toJson(userMgmt.getUser(name, surname));
35         } else {
36             List<User> users = userMgmt.getUsers();
37             return jsonb.toJson(users);
38         }
39     }
40
41     @GET
42     @Path("/{id}")
43     @Produces("application/json")
44     public String getUserAsJson(@PathParam("id") int userID) {
45         Jsonb jsonb = JsonbBuilder.create();
46         return jsonb.toJson(userMgmt.getUser(userID));
47     }
48 }
49
50
51
52
53 }
```

# CommentsService

Diese Klasse funktioniert ähnlich wie UserService. Statt einem Kommentar gibt die Methode `getCommentAsJson()` jedoch eine Kommentar aus der Datenbank zurück und nutzt dafür die Klasse `CommentsManagement`. Auch hier haben wir Query Parameter benutzt. Durch diese wird entschieden, ob die Kommentare für einen Nutzer zurück gegeben werden, die er verfasst hat, oder diejenigen, welche an ihn gerichtet wurden.

```
20 @Path("/comments")
21 public class CommentsService {
22
23     @EJB
24     private CommentsManagement commentsMgmt;
25
26     public CommentsService() {
27     }
28
29     * Retrieves all Comments which are saved in the database for a specific user id.
30
31     @GET
32     @Produces("application/json")
33     public String getCommentsAsJson(@QueryParam("posterID") int posterID,
34                                     @QueryParam("receiverID") int receiverID) {
35
36         Jsonb jsonb = JsonbBuilder.create();
37         if (posterID != 0) {
38             return jsonb.toJson(commentsMgmt.getCommentsByPosterID(posterID));
39         } else if (receiverID != 0) {
40             return jsonb.toJson(commentsMgmt.getCommentsByReceiverID(receiverID));
41         } else {
42             return "";
43         }
44     }
45 }
```

## CommentsService

Die Methode `postCommentAsJson()` speichert den übergebene Kommentar mithilfe der Methode `saveComment()` aus dem `CommentsManager` in der Datenbank. Dazu wird der übergebene Kommentar mithilfe eines `JSON-Readers` in ein `JSON` umgewandelt und dessen Attribute dann einzeln ausgelesen und direkt bei der Initialisierung des Kommentars übergeben.

```
52  @POST
53  @Produces("application/json")
54  public String postCommentsAsJson(String comment) {
55      JsonReader jsonr = Json.createReader(new StringReader(comment));
56      JsonObject jsonObject = jsonr.readObject();
57
58      Comment newComment = new Comment(jsonObject.getInt("receiverID"), jsonObject.getInt("posterID"), jsonObject.getString("content"));
59
60      commentsMgmt.saveComment(newComment);
61
62      Jsonb jsonb = JsonbBuilder.create();
63      return jsonb.toJson(newComment);
64  }
```

# Frontend

## Projektstruktur

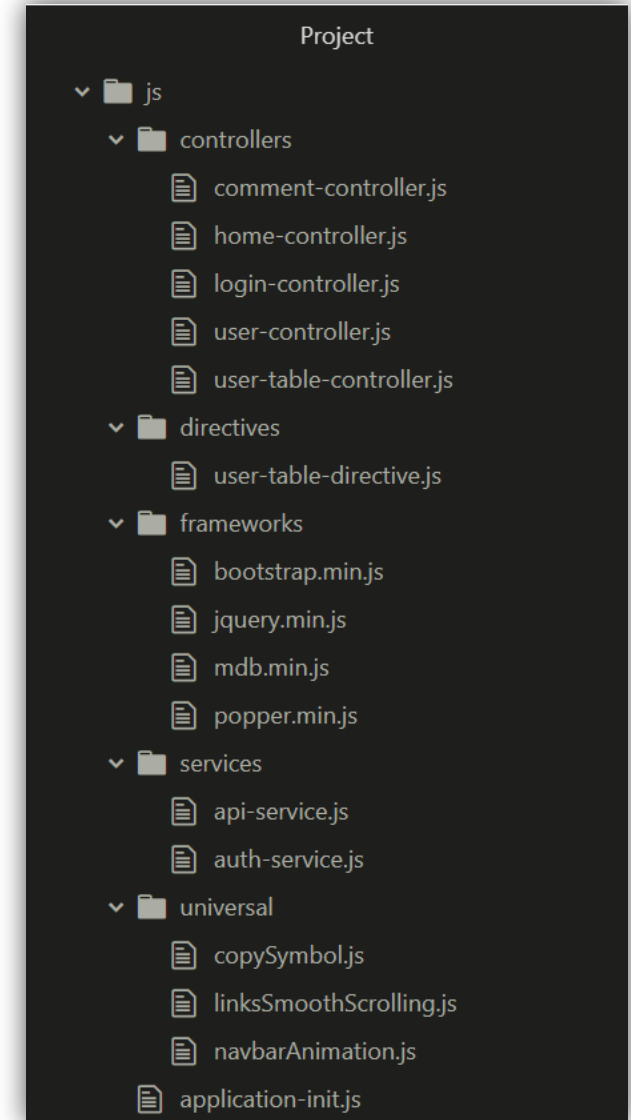
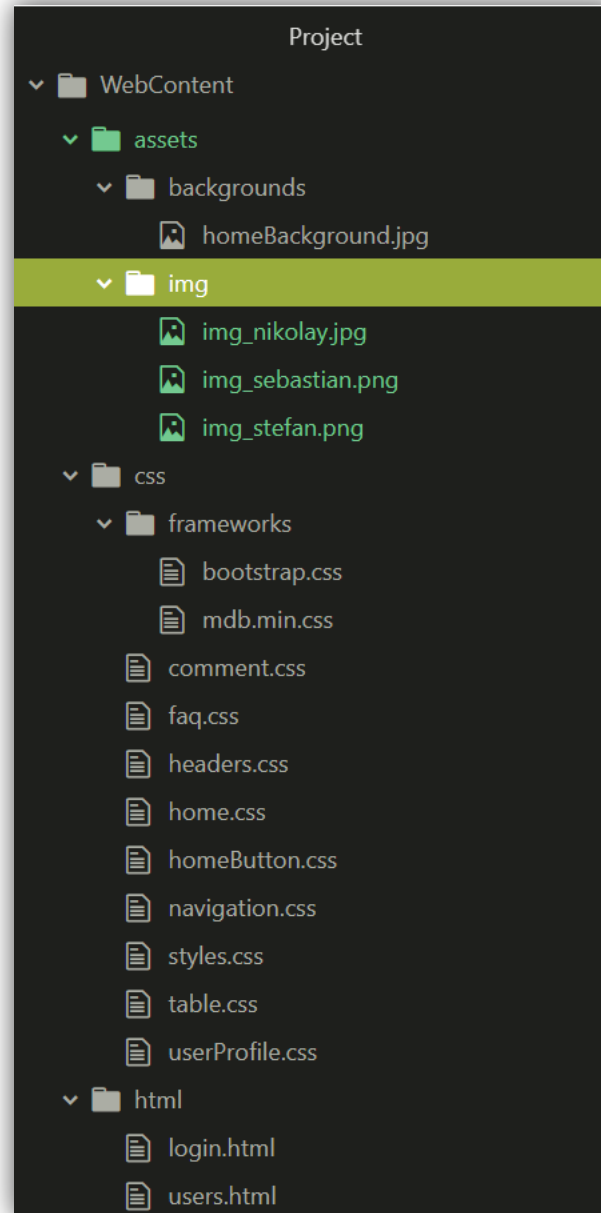
Hier beschreiben wir die Ordnerstruktur des Frontends.

Es gibt 5 Hauptordner:

- `assets` – Hier befinden sich alle Fotos, die die Webseite nutzt (Hintergründe (backgrounds) zum Beispiel)
- `css` – Hier befinden sich alle CSS Dateien
- `html` – Hier befinden sich alle html Dateien (exklusive der `index.html`)
- `js` – Hier befinden sich alle JavaScript Dateien

Außerdem gibt es eine `index.html` Datei, in der sich die Landing Page befindet.

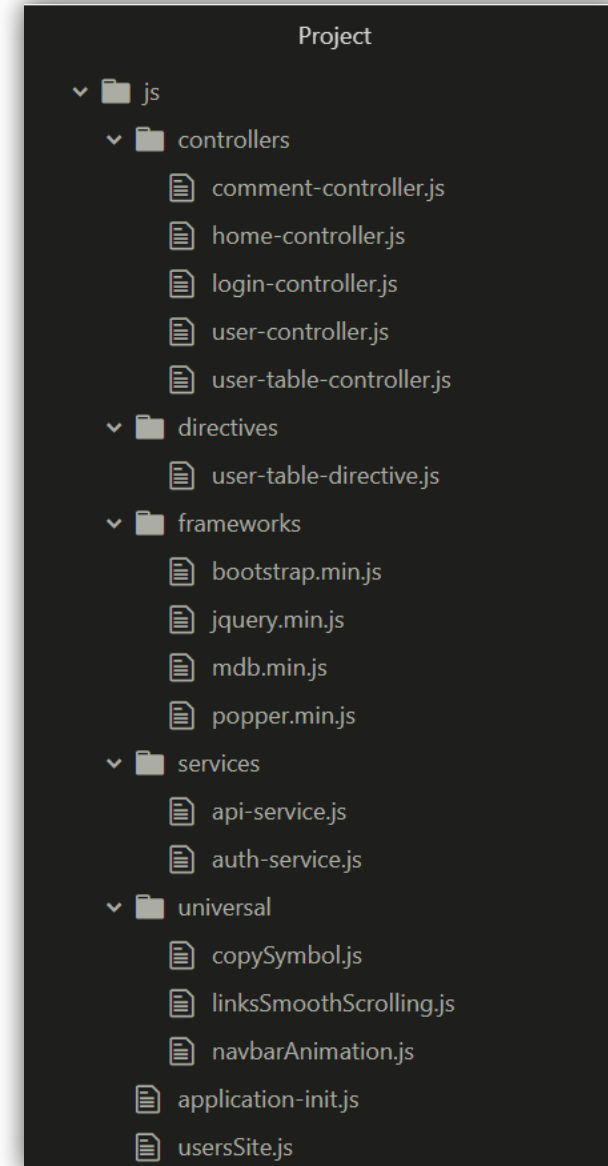
# Client-side



# Frontend JavaScript

Das Frontend JavaScript der Webseite gestaltet sich folgendermaßen:

- `controllers` – Hier befindet sich der JavaScript Code, welcher für die Kontrolle der wichtigsten Komponenten (Benutzer, Benutzertabelle, Kommentare) verantwortlich ist.
- `directives` – Hier befindet sich der JavaScript Code, der das UI der Webseite verändert.
- `frameworks` – Hier befinden sich alle Frameworks und Libraries, die für das Frontend benutzt werden.
- `services` – Hier befindet sich der JavaScript Code, welcher für die Kommunikation zwischen dem Client und dem Server und die Authentifizierung des Benutzers verantwortlich ist.
- `universal` – Hier befindet sich der JavaScript Code, der den in unterschiedlichsten Seiten der Webseite genutzt wird (Navigation, Smooth Link Scrolling, usw.)



## Frontend JavaScript

Außerdem gibt es noch eine JavaScript Datei:

- application-init.js –

Die prüft, ob ein Benutzer sich eingeloggt hat. Wenn nicht, wird er auf die Login-Seite umgeleitet.

```
application-init.js
1  (function init () {
2      let currentUser = AuthService.getCurrentUser();
3
4      if (!currentUser) {
5          window.location.href = 'http://localhost:8080/TeamDirectoryWeb/html/login.html';
6      }
7  }());
```



# Frontend Frameworks und Libraries

Wir benutzen einige Frameworks und Libraries, um die Webseite schöner zu machen (sie manipulieren nur das Aussehen der Webseite und nicht die Logik):

- Bootstrap – <https://getbootstrap.com/>
- MDBBootstrap – <https://mdbbootstrap.com/>

Client-side

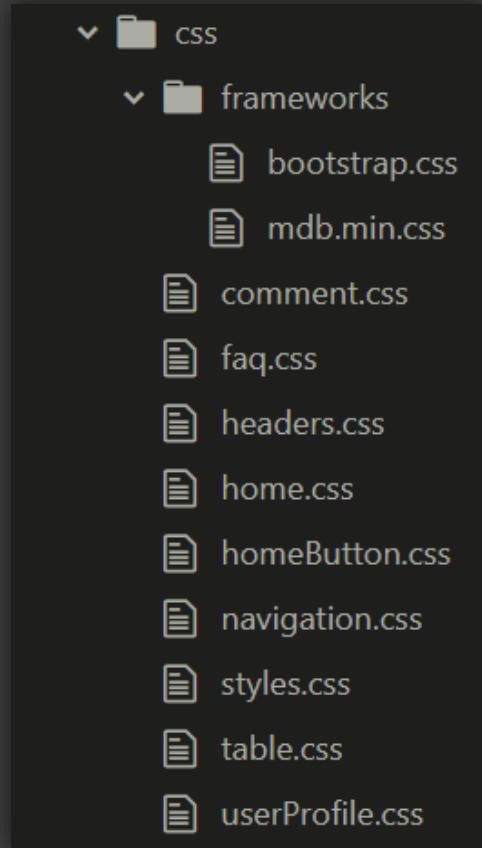


# Frontend CSS

## Client-side

Wir benutzen CSS, um die Webseite schöner zu machen. Wir haben die CSS Dateien folgendermaßen aufgeteilt:

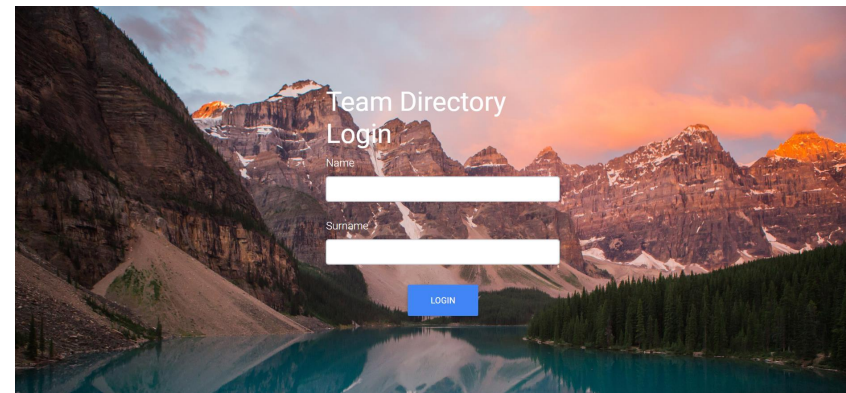
- `home.css` – CSS für die Home Section in der `index.html` (Landing Page)
- `homeButton.css` – CSS für den Button in der Home Section. Dieser leitet den Benutzer auf die Signup/Login Page weiter
- `aboutSection.css` – CSS für die About Section in der `index.html` (Landing Page)
- `faq.css` – CSS für die FAQ Section in der `index.html` (Landing Page)
- `contact.css` – CSS für die Contact Section in der `index.html` (Landing Page)
- `formValidation.css` – CSS für das *Contact form* in der Contact Section in der `index.html` (Landing Page), welches das Aussehen des Formulars nach der Validierung verändert
- `comment.css` – CSS für den Kommentar in der Datei `users.html` (Users and Profiles Page)
- `table.css` – CSS für die Tabelle, in der alle Users aus der Datenbank aufgerufen werden
- `userProfile.css` – CSS für das Benutzerprofil in `users.html` (Users and Profiles Page)
- `headers.css` – CSS für h1, h2, h3, h4 und h5 Überschriften
- `navigation.css` – CSS für die Navigationsleiste
- `socialButtons.css` – CSS für die Buttons für die sozialen Medien im Footer
- `styles.css` – CSS, welches von allen Seiten der Webseite benutzt wird und nicht konkret ist
- `bootstrap.css` – Bootstrap CSS (Framework)
- `mdb.min.css` – Minified MDBBootstrap CSS (Framework)



# Die Webseite

Die Webseite besteht aus 3 Seiten:

1. **Landing Page** – Wenn man zum ersten Mal die Webseite besucht, wird diese Seite aufgerufen. Das ist der so genannte Einstiegspunkt der Webseite.
2. **Signup/Login Page** – Wenn man die Webseite vollständig nutzen will (und nicht nur die Landing Page), muss man sich anmelden. Der Benutzer kann auch ein neues Konto erstellen, falls er noch keines hat.
3. **Users and Profiles Page** – Nachdem man sich angemeldet hat, kann man nach anderen Benutzern suchen und Kommentare schreiben.



Team Directory						Home	About	FAQ	Contact
Search									
First Name	Surname	Age	Study Course	University	Id				
John	Doe	20	Wirtschaftsinformatik	TU Berlin	123				
Ivan	Doe	19	Chemie	TU Berlin	321				

## Client-side



## Die Landing Page



Foto von Jonathan Daniels auf Unsplash - <https://unsplash.com/photos/kxgMWWVbvcw>

## Die Signup-/Login-Seite

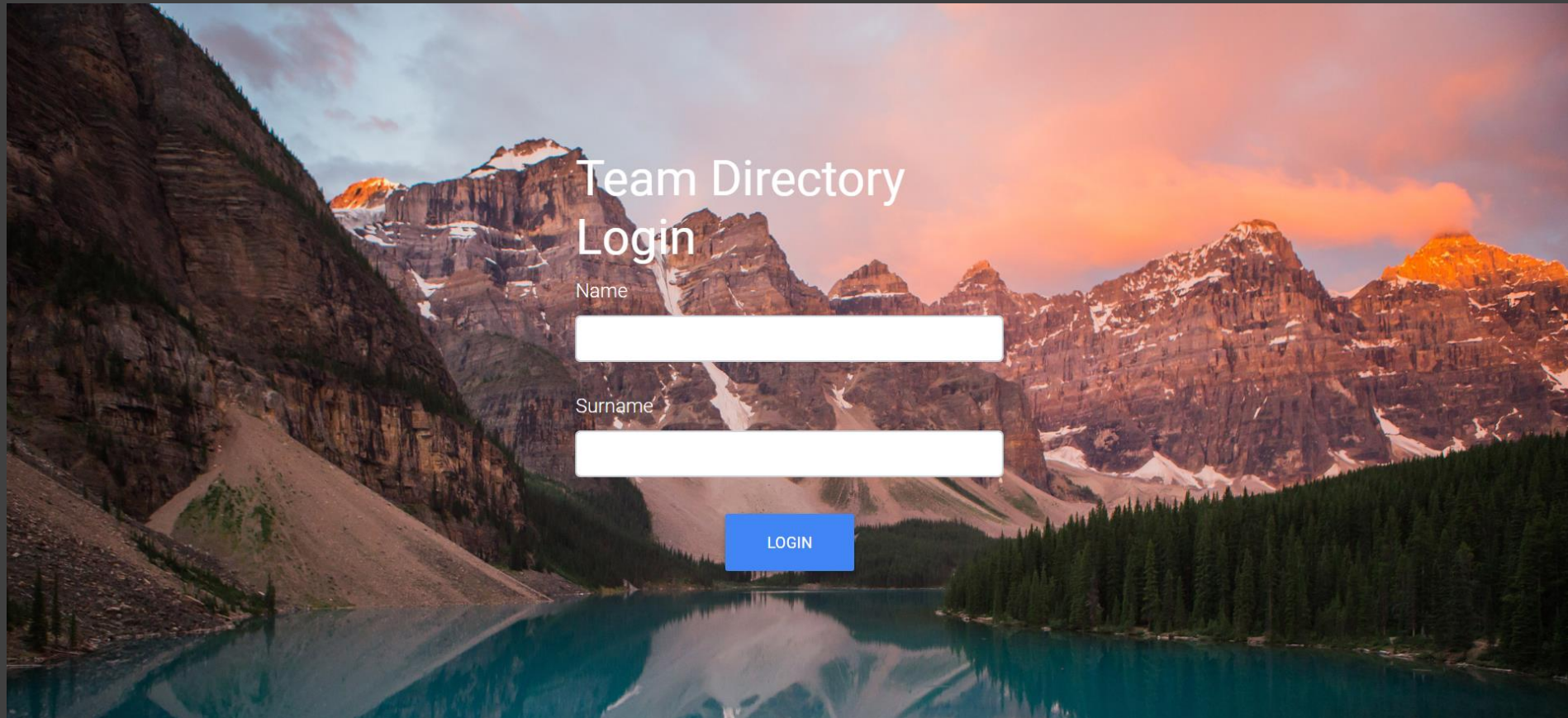






Foto von garrett parker auf Unsplash - <https://unsplash.com/photos/DIkF4-dbCOU>

# Die Benutzer- und Profilseite

## Team Directory

Search

Click any User to show his profile information.

First Name	Surname	Age	Study Course	University	Id	
Stefan	Yovchev	20	Wirtschaftsinformatik	TU Berlin	1	
Bolz	Victor	19	Wirtschaftsinformatik	TU Berlin	4	
Nikolay	Nikolov	19	Wirtschaftsinformatik	TU Berlin	3	
Sebastian	Führ	19	Wirtschaftsinformatik	TU Berlin	2	



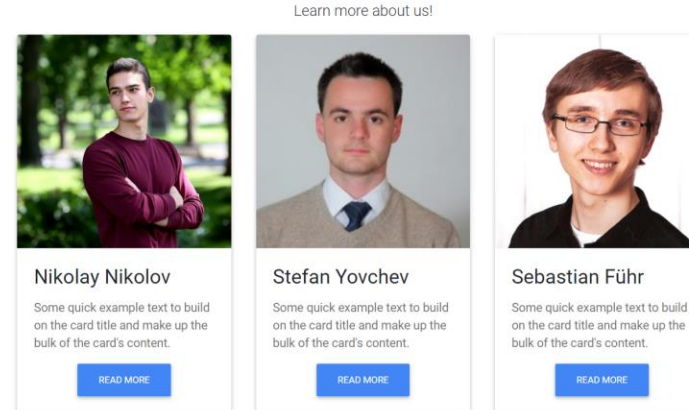
# Die Landing Page

Die Landing Page bestehen aus 4 Abschnitten:

1. **Home** – Hier befindet sich der Button zur Registrierungs- bzw. Login-Seite. Für die Animation des homeButtons, haben wir Code aus codepen benutzt –  
<https://codepen.io/wintr/pen/wWoRVW>  
<https://codepen.io/vpdemo/pen/WbMNJv>
2. **About us** – Umfasst die wichtigsten Informationen über die drei Schöpfer der Webseite – Stefan, Sebastian und Nikolay
3. **Contact us** - Ein Kontaktformular, welches man sehr leicht ausfüllen kann um es dann an uns zu schicken. Das Formular wird Client-seitig validiert (keine Implementation auf Server)



## About Section



## Contact us!

You have questions? Contact us and we will help you!

Name  Email

Subject

Message

+499999999999  
teamdirectory@aws.com

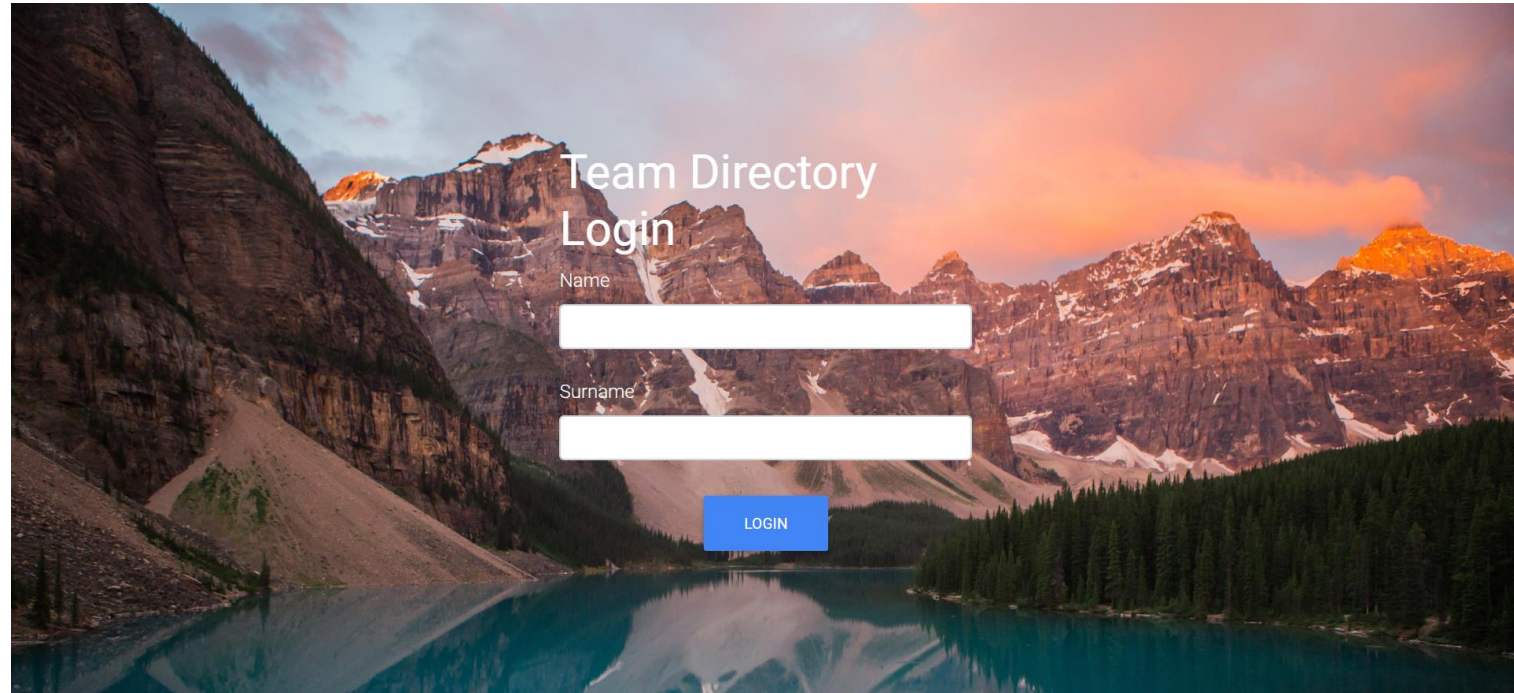
Client-side

# Die Registrierungs- bzw.

## Login-Seite

Hier loggt sich der Benutzer ein, indem er seine Name bzw. Nachname eingibt. Seine Daten werden an Server geschickt. Falls so ein Benutzer existiert, wird sein Profil an Client zurückgeschickt. Wenn nicht wird einen neuen Benutzer mit den oben beschriebenen (Folie ) Standartwerten erstellt. Im beiden Fällen wird die Benutzerinformation im localStorage gespeichert

## Client-side





```
1 const LoginController = (() => {
2   return {
3     init() {
4       $(document).ready(() => {
5         (function initDOM () {
6           $('#login-btn').on('click', event => {
7             let name = $('#name').val(),
8                 surname = $('#surname').val();
9
10            if (!name || !surname) {
11              return;
12            }
13
14            $(event.target).prop('disabled', true);
15
16            AuthService.login(name, surname);
17          });
18        })();
19      });
20    }
21  };
22 })();
```

Der LoginController behandelt die Validierungslogik und übergibt die Daten an den AuthService.

Der beschäftigt sich mit alles, was mit Authentifizierung verbunden ist.





```
1 const AuthService = (() => {
2   return {
3     login(name, surname) {
4       ApiService.User.getUser().byName(name, surname).done(user => {
5         localStorage.setItem('user', JSON.stringify(user));
6
7         window.location.href = '/TeamDirectoryWeb';
8       });
9     },
10    getCurrentUser() {
11      return JSON.parse(localStorage.getItem('user'));
12    },
13    removeCurrentUser() {
14      localStorage.removeItem('user');
15    }
16  };
17 })();
```

# Die Benutzer-/Profilseite




Die Benutzer- und Profilseiten sind die wichtigsten Seiten unserer Webseite. Hier entsteht die wichtigste Kommunikation zwischen dem Client und dem Server. Zugang zu dieser Seite haben nur registrierte Benutzer, die sich bereits angemeldet haben.

Folgenden Komponenten sind zu finden:

- Benutzer Suchen** – Man kann nach anderen registrierten Benutzern suchen, die Resultate ordnen, u.ä. Wenn man auf einen Benutzer klickt, dann wird das Konto (Profil) dieses Benutzers aufgerufen und im Browser angezeigt.
- Benutzer Profile** – besteht aus zwei Teilen:
  - Benutzer Daten:  
Hier werden die wichtigste Daten des aufgerufenen Benutzer angezeigt.
  - Kommentare:  
Hier werden alle Kommentare auf dem Konto dieses Benutzers angezeigt. Außerdem kann der angemeldeten Benutzer neue Kommentare schreiben.


Team Directory						
Search						
Click any User to show his profile information.						
First Name	Surname	Age	Study Course	University	Id	
Stefan	Yovchev	20	Wirtschaftsinformatik	TU Berlin	1	
Bolz	Victor	19	Wirtschaftsinformatik	TU Berlin	4	
Nikolay	Nikolov	19	Wirtschaftsinformatik	TU Berlin	3	
Sebastian	Führ	19	Wirtschaftsinformatik	TU Berlin	2	

Team Directory






Nikolay	Nikolov	19	Wirtschaftsinformatik	TU Berlin	3	
Bolz	Victor	19	Wirtschaftsinformatik	TU Berlin	4	
Stefan	Yovchev	20	Wirtschaftsinformatik	TU Berlin	1	

Current User

This is the user you have selected!




**Sebastian Führ**

Age: 19  
Study course: Wirtschaftsinformatik  
University: TU Berlin

1 comment/s



**Author: Stefan Yovchev**  
Date: 15.06.2018  
Time: 18:43

Das Wetter ist schlecht.

Your comment

### Benutzer Suchen

- a) Wenn man diese Seite unserer Webseite besucht, wird eine Anfrage (Request) an den Server gesendet. Der Client braucht alle Benutzer die schon registriert sind und sich in der Datenbank befinden.
- b) Wenn die Antwort vom Server ankommt, wird das JSON durch Client-seitiges JavaScript gelesen und die Daten jedes Benutzers werden zu einer Tabelle hinzugefügt.

Auf den nächsten Seiten werden alle Schritte vollständig gezeigt!

# Benutzer vom Server abrufen

```

1  user-table-controller.js
2
3  4. const UserTableController = (() => {
4      const UserRowTemplate = ({ name, surname, age, studyCourse, university, id }) => `
5          <tr class="user-row">
6              <td class="user-data">${name}</td>
7              <td class="user-data">${surname}</td>
8              <td class="user-data">${age}</td>
9              <td class="user-data">${studyCourse}</td>
10             <td class="user-data">${university}</td>
11             <td class="user-data">${id}</td>
12             <td class="copySymbol" onmouseout="outFunc()"><span class="tooltip">Copy</span><i class="fas fa-copy fa-lg"></i></td>
13         </tr>
14     `;
15
16     return {
17         init() {
18             $(document).ready(() => {
19                 (function initDOM () {
20                     $('body').scrollspy({
21                         target: ".navbar",
22                         offset: 300
23                     });
24
25                     let navbarHeight = $("#navbar").css("height").split("px")[0];
26
27                     $("#usersSection").css("margin-top", (navbarHeight * 1.5) + "px");
28                     $("#navbar").css("background-color", "rgba(0,123,255, 0.95)");
29
30                     setupCopySymbol();
31                     linkSmoothScrolling();
32                     navbarAnimation();
33                 })();
34
35                 (function initVM () {
36                     ApiService.User.getUsers().done(users => {
37                         users.forEach(user => {
38                             $('#theTable').append([user].map(UserRowTemplate).join(''));
39                         });
40                     });
41                 })();
42             });
43         }
44     };
45 }

```

```

1  api-service.js
2
3  const ApiService = (() => {
4      const api = 'http://localhost:8080/TeamDirectoryWeb/api'
5
6      3. function $request (type, url, data) {
7          if (!data) {
8              data = JSON.stringify(data);
9          }
10
11          return $.ajax({
12              type,
13              dataType: 'json',
14              contentType: 'application/json',
15              url,
16              data
17          });
18
19          2. return {
20              User: {
21                  getUsers() {
22                      let url = `${api}/user`;
23
24                      return $request('GET', url, null);
25                  },
26                  getUser() {
27                      return {
28                          byId(id) {
29                              let url = `${api}/user/${id}`;
30
31                              return $request('GET', url, null);
32                          },
33                          byName(name, surname) {
34                              let url = `${api}/user?name=${name}&surname=${surname}`;
35                              window.alert("api-service: "+JSON.stringify($request('GET', url, null)));
36                              return $request('GET', url, null);
37                          }
38                      }
39                  }
40              }
41          };
42      }
43  })();

```

# Benutzer Suchen

Man kann diese Daten dann  
sortieren, wenn man auf die  
Kopfzeilen der Tabelle klickt.

Wir haben Code von der  
Website w3schools benutzt -  
[https://www.w3schools.com/howto/howto\\_js\\_sort\\_table.asp](https://www.w3schools.com/howto/howto_js_sort_table.asp)

```
user-table-directive.js
1  const UserTableDirective = () => {
2    return {
3      sortTable(n) {
4        let table, rows, switching, i, x, y, shouldSwitch, dir, switchcount = 0;
5        table = document.getElementById("databaseTable");
6        switching = true;
7        // Set the sorting direction to ascending:
8        dir = "asc";
9        /* Make a loop that will continue until
10         no switching has been done: */
11        while (switching) {
12          // Start by saying: no switching is done:
13          switching = false;
14          rows = table.getElementsByTagName("TR");
15          /* Loop through all table rows (except the
16           first, which contains table headers): */
17          for (i = 1; i < (rows.length - 1); i++) {
18            // Start by saying there should be no switching:
19            shouldSwitch = false;
20            /* Get the two elements you want to compare,
21             one from current row and one from the next: */
22            x = rows[i].getElementsByTagName("TD")[n];
23            y = rows[i + 1].getElementsByTagName("TD")[n];
24            /* Check if the two rows should switch place,
25             based on the direction, asc or desc: */
26            if (dir == "asc") {
27              if (x.innerHTML.toLowerCase() > y.innerHTML.toLowerCase()) {
28                // If so, mark as a switch and break the loop:
29                shouldSwitch = true;
30                break;
31              }
32            } else if (dir == "desc") {
33              if (x.innerHTML.toLowerCase() < y.innerHTML.toLowerCase()) {
34                // If so, mark as a switch and break the loop:
35                shouldSwitch = true;
36                break;
37              }
38            }
39          }
40        }
41        if (shouldSwitch) {
42          /* If a switch has been marked, make the switch
43             and mark that a switch has been done: */
44          rows[i].parentNode.insertBefore(rows[i + 1], rows[i]);
45          switching = true;
46          // Each time a switch is done, increase this count by 1:
47          switchcount++;
48        } else {
49          /* If no switching has been done AND the direction is "asc",
50             set the direction to "desc" and run the while loop again. */
51          if (switchcount == 0 && dir == "asc") {
52            dir = "desc";
53            switching = true;
54          }
55        }
56      },
57    },
58  };
59 }
```

## Benutzer Suchen

Man kann auch nach Benutzern suchen. Das Suchen ist dynamisch und die Resultate werden nach jedem Tastendruck angezeigt. Wir haben dafür Code von der Website





stackoverflow benutzt -  
<https://stackoverflow.com/a/40358801/9698409>

```
user-table-directive.js
58 searchTable() {
59     let emptySearch = true;
60
61     $("#emptySearch").css("display", "none");
62
63     let input, filter, found, table, tr, td, i, j;
64
65     input = document.getElementById("searchInput");
66     filter = input.value.toUpperCase();
67     table = document.getElementById("databaseTable");
68     tr = table.getElementsByTagName("tr");
69
70     for (i = 1; i < tr.length; i++) {
71         td = tr[i].getElementsByTagName("td");
72
73         for (j = 0; j < td.length; j++) {
74             if (td[j].innerHTML.toUpperCase().indexOf(filter) > -1) {
75                 found = true;
76                 emptySearch = false;
77             }
78         }
79
80         if (found) {
81             tr[i].style.display = "";
82             found = false;
83         } else {
84             tr[i].style.display = "none";
85         }
86     }
87
88     if (emptySearch) {
89         $("#emptySearch").css("display", "block");
90         console.log("Nothing found!");
91     }
92 }
93 }
94 })();
```

# Benutzer Suchen

Man kann auch den Namen jedes Benutzers sehr leicht und schnell kopieren – Das wurde noch nicht vollständig implementiert, aber es könnte sehr leicht mit etwas mehr Zeit umgesetzt werden. Wir haben auch hier Code von w3schools benutzt -

[https://www.w3schools.com/css/css\\_tooltip.asp](https://www.w3schools.com/css/css_tooltip.asp)

Team Directory						
Search						
Click any User to show his profile information.						
First Name	Surname	Age	Study Course	University	Id	
Stefan	Yovchev	20	Wirtschaftsinformatik	TU Berlin	1	
Sebastian	Führ	19	Wirtschaftsinformatik	TU Berlin	2	
Nikolay	Nikolov	19	Wirtschaftsinformatik	TU Berlin	3	
Bolz	Victor	19	Wirtschaftsinformatik	TU Berlin	4	

```

table.css
25 .copySymbol{
26   color: rgba(0, 123, 255, 0.95);
27   position: relative;
28   cursor: pointer;
29   margin: auto;
30 }
31
32 /* Tooltip text */
33 .tooltip {
34   visibility: hidden;
35   width: 50px;
36   background-color: grey;
37   color: white;
38   text-align: center;
39   padding: 5px 0;
40   border-radius: 6px;
41   opacity: 1;
42
43   /* Position the tooltip text - see examples below! */
44   position: absolute;
45   z-index: 9999;
46   bottom: 90%;
47   left: 50%;
48   margin-left: -25px;
49 }

```

```

50 .tooltip::after {
51   content: "";
52   position: absolute;
53   top: 100%;
54   left: 50%;
55   margin-left: -5px;
56   border-width: 5px;
57   border-style: solid;
58   opacity: 1;
59   border-color: grey transparent transparent transparent;
60 }
61 .copySymbol:hover .tooltip {
62   visibility: visible;
63 }
64 .copySymbol:hover i{
65   transform: scale(1.5);
66   transition: transform 0.25s ease-out;
67 }

```

## Benutzerprofile

- a) Wenn man in der Tabelle auf einen bestimmten Benutzer klickt, dann wird eine Anfrage (Request) an den Server geschickt, damit die Daten dieses Benutzers gesendet werden.
- b) Wenn die Antwort vom Server ankommt, wird das JSON durch Client-seitiges JavaScript gelesen und die Daten dieses Benutzers in dem Abschnitt „Benutzer Profile“ angezeigt. Von dem Server werden sowohl die **Benutzerdaten**, als auch die **Kommentare** auf dem Konto dieses Benutzers aufgerufen. Außerdem kann der in Moment angemeldeten Benutzer neue Kommentare schreiben.

Auf den nächsten Seiten werden alle Schritte vollständig gezeigt!



# Benutzerprofile

## user-controller.js

```

1  const UserController = (() => {
2    const labels = {
3      'age': 'Age: ',
4      'studyCourse': 'Study course: ',
5      'university': 'University: '
6    };
7
8    return {
9      init() {
10        $(document).ready(() => {
11          (function initDOM () {
12            1.  $('body').on('click', 'table tr.user-row, .commentAuthor, .commentAuthorPhoto', event => {
13              let id;
14
15              let $target = $(event.target);
16
17              switch ($target.attr('class')) {
18                case 'user-data':
19                  console.log($target);
20                  id = id = $target.parent().children()[5].innerHTML;
21                  break;
22                case 'commentAuthor':
23                  id = $target.parent().parent().parent().attr('id').split("-")[0];
24                  break;
25                case 'commentAuthorPhoto':
26                  id = $target.parent().parent().parent().attr('id').split("-")[0];
27                  break;
28              }
29
30              2.  ApiService.User.getUser().byId(id).done(user => {
31                $("#currentUserId").text(user.id);
32                $("#currentUserName").text(`${user.name} ${user.surname}`);
33                $("#currentUserAge").text(`${labels.age} ${user.age}`);
34                $("#currentUserStudyCourse").text(`${labels.studyCourse} ${user.studyCourse}`);
35                $("#currentUserUniversity").text(`${labels.university} ${user.university}`);
36                $("#currentUserPhoto").attr("src", `../img/${user.imgPath}`);
37
38                $("#currentUserSection").css("display", "block");
39
40                5.  CommentController.init(id);
41              });
42
43              $('html,body').animate({ scrollTop: $("#currentUserSection").offset().top }, 'slow');
44            });
45          })();
46        });
47      }
48    };

```

## api-service.js

```

1  const ApiService = (() => {
2    const api = 'http://localhost:8080/TeamDirectoryWeb/api'
3
4.  function $request (type, url, data) {
5    if (!data) {
6      data = JSON.stringify(data);
7    }
8
9    return $.ajax({
10      type,
11      dataType: 'json',
12      contentType: 'application/json',
13      url,
14      data
15    });
16
17    return {
18      User: {
19        getUsers() {
20          let url = `${api}/user`;
21
22          return $request('GET', url, null);
23        },
24        getUser() {
25          3.  return {
26            byId(id) {
27              let url = `${api}/user/${id}`;
28
29              return $request('GET', url, null);
30            },
31            byName(name, surname) {
32              let url = `${api}/user?name=${name}&surname=${surname}`;
33              window.alert("api-service: "+JSON.stringify($request('GET', url, null)));
34              return $request('GET', url, null);
35            }
36          }
37        }
38      }
39    };

```

## Benutzerdaten

Der erste Teil des Benutzerprofils sind die Benutzer Daten:

- **Name** (Vor- und Nachname)
- **Bild**
- **Alter**
- **Studiengang**
- **Universität**

Es gibt auch Links zu verschiedenen sozialen Medien, aber diese sind nicht in dem Server implementiert.

### Current User

This is the user you have selected!



```
user-controller.js
30 ApiService.User.getUser().byId(id).done(user => {
31   $("#currentUserId").text(user.id);
32   $("#currentUserName").text(`${user.name} ${user.surname}`);
33   $("#currentUserAge").text(`${labels.age} ${user.age}`);
34   $("#currentUserStudyCourse").text(`${labels.studyCourse} ${user.studyCourse}`);
35   $("#currentUserUniversity").text(`${labels.university} ${user.university}`);
36   $("#currentUserPhoto").attr("src", `../img/${user.imgPath}`);
37
38   $("#currentUserSection").css("display", "block");
```

# Benutzerkommentare

Dieser Abschnitt besteht aus zwei Teilen – **“Kommentare”** und **“Neue Kommentare schreiben”**.

Die Kommentare werden vom Server als JSON empfangen. Mit Hilfe von JavaScript wird das JSON gelesen und die Kommentare der HTML hinzugefügt.

Jeder Kommentar besteht aus:

- Bild
- Name (Vor- und Nachname)
- Datum
- Zeit
- Inhalt

## Client-side

```
user-controller.js
1  const UserController = (() => {
2    const labels = {
3      'age': 'Age: ',
4      'studyCourse': 'Study course: ',
5      'university': 'University: '
6    };
7
8    return {
9      init() {
10        $(document).ready(() => {
11          (function initDOM () {
12            $('body').on('click', 'table tr.user-row, .commentAuthor, .commentAuthorPhoto', event => {
13              let id;
14
15              let $target = $(event.target);
16
17              switch ($target.attr('class')) {
18                case 'user-data':
19                  console.log($target);
20                  id = id = $target.parent().children()[5].innerHTML;
21                  break;
22                case 'commentAuthor':
23                  id = $target.parent().parent().parent().attr('id').split("-")[0];
24                  break;
25                case 'commentAuthorPhoto':
26                  id = $target.parent().attr('id').split("-")[0];
27                  break;
28              }
29
30              ApiService.User.getUser().byId(id).done(user => {
31                $("#currentUserId").text(user.id);
32                $("#currentUserName").text(`${user.name} ${user.surname}`);
33                $("#currentUserAge").text(`${labels.age} ${user.age}`);
34                $("#currentUserStudyCourse").text(`${labels.studyCourse} ${user.studyCourse}`);
35                $("#currentUserUniversity").text(`${labels.university} ${user.university}`);
36                $("#currentUserPhoto").attr("src", `../img/${user.imgPath}`);
37
38                $("#currentUserSection").css("display", "block");
39
40                CommentController.init(id);
41              });
42            });
43          });
44        });
45      }
46    };
47  })();
```

1.

1 comment/s



**Author: Stefan Yovchev**

Date: 15.06.2018  
Time: 18:43

Das Wetter ist schlecht.

# Benutzerkommentare

```

comment-controller.js
1  const CommentController = (() => {
2    const CommentTemplate = ({ author, date, content, time, id, posterID, receiverID }) => `
3      <div class="media d-block d-md-flex mt-4" id= "${id}">
4        
5        <div class="media-body text-center text-md-left ml-md-3 ml-0">
6          <h5 class="font-weight-bold mt-0">
7            <p class="commentAuthor">
8              Author: ${author}
9            </p>
10           <p class="commentDate">
11             Date: ${date}
12           </p>
13           <p class="commentTime">
14             Time: ${time}
15           </p>
16         </h5>
17         <p class="commentText">
18           ${content}
19         </p>
20       </div>
21     </div>
22   `;
23
24   return {
25     init(id) {
26       (function initVM () {
27         let commentsNumber = 0;
28
29         $('#userComments').text("");
30
31         ApiService.Comment.getComments().byReceiverId(id).done(comments => {
32           console.log(comments);
33           commentsNumber = comments.length;
34           $('#commentsHeader').text(commentsNumber + " comment/s");
35
36           comments.forEach(comment => {
37             $('#userComments').append([comment].map(CommentTemplate).join(''));
38           });
39         });
40       })();

```

```

api-service.js
40
41   Comment: {
42     postComment(comment) {
43       let url = `${api}/comments`;
44       return $request('POST', url, comment);
45     },
46     getComments() {
47       let url = `${api}/comments`;
48
49       return {
50         byReceiverId(receiverId) {
51           url += `?receiverID=${receiverId}`;
52
53           return $request('GET', url, null);
54         },
55         byPosterId(posterId) {
56           url += `?posterID=${posterId}`;
57
58           return $request('GET', url, null);
59         }
60       };
61     };
62   };
63   })();

```

```

api-service.js
1  const ApiService = (() => {
2    const api = 'http://localhost:8080/TeamDirectoryWeb/api'
3
4    function $request (type, url, data) {
5      if (!!data) {
6        data = JSON.stringify(data);
7      }
8
9      return $.ajax({
10        type,
11        dataType: 'json',
12        contentType: 'application/json',
13        url,
14        data
15      });
16    }

```

# Neue Kommentare schreiben

Man kann auch neue Kommentare schreiben.

1. Das passiert, indem man den Inhalt des Kommentares in ein Textfeld hinein schreibt und dann auf den Button „Comment“ klickt.
2. Eine Anfrage (Request) wird an den Server geschickt und falls der Kommentar vom Server validiert wird, wird er in der Datenbank gespeichert.
3. Dann wird der Kommentar als Antwort (Response) vom Server an den Client gesendet.
4. Das Kommentar wird durch JavaScript angezeigt.

## Client-side

```
comment-controller.js
42 (function initDOM () {
43   $('#writeCommentButton').on('click', event => {
44     let user = AuthService.getCurrentUser(),
45     content = $('#writeCommentText').val();
46     posterID = user.id;
47     receiverID = +($('#currentUserID').text());
48
49     // Check if content is empty
50     if (!content) {
51       return;
52     }
53
54     $(event.target).prop('disabled', true);
55
56     ApiService.Comment.postComment({ content, posterID, receiverID }).done(comment => {
57       $('#userComments').append([comment].map(CommentTemplate).join(''));
58
59       $(event.target).prop('disabled', false);
60     });
61   });
62 })();
```

```
api-service.js
40 Comment: {
41   postComment(comment) {
42     let url = `${api}/comments`;
43     return $request('POST', url, comment);
44   },
45 }
```

Your comment



# Neue Kommentare schreiben

5. `api-service.js`

```

1 const ApiService = (() => {
2   const api = 'http://localhost:8080/TeamDirectoryWeb/api'
3
4   function $request (type, url, data) {
5     if (!data) {
6       data = JSON.stringify(data);
7     }
8
9     return $.ajax({
10      type,
11      dataType: 'json',
12      contentType: 'application/json',
13      url,
14      data
15    });
16  }
17

```

6. `comment-controller.js`

```

41
42 (function initDOM () {
43   $('#writeCommentButton').on('click', event => {
44     let user = AuthService.getCurrentUser();
45     content = $('#writeCommentText').val();
46     posterID = user.id;
47     receiverID = +($('#currentUserId').text());
48
49     // Check if content is empty
50     if (!content) {
51       return;
52     }
53
54     $(event.target).prop('disabled', true);
55
56     ApiService.Comment.postComment({ content, posterID, receiverID }).done(comment => {
57       $('#userComments').append([comment].map(CommentTemplate).join(''));
58
59       $(event.target).prop('disabled', false);
60     });
61   });
62 })();

```


7. `comment-controller.js`

```

1 const CommentController = (() => {
2   const CommentTemplate = ({ author, date, content, time, id, posterID, receiverID }) => `
3
4     <div class="media d-block d-md-flex mt-4" id="${id}">
5       
6       <div class="media-body text-center text-md-left ml-md-3 ml-0">
7         <h5 class="font-weight-bold mt-0">
8           <p class="commentAuthor">
9             Author: ${author}
10           </p>
11           <p class="commentDate">
12             Date: ${date}
13           </p>
14           <p class="commentTime">
15             Time: ${time}
16           </p>
17         </h5>
18         <p class="commentText">
19           ${content}
20         </p>
21       </div>
22     `;
23


```

2 comment/s



**Author: Stefan Yovchev**  
Date: 15.06.2018  
Time: 18:43

Das Wetter ist schlecht.



**Author: Sebastian Führ**  
Date: 24.06.2018  
Time: 19:27

Hallo!

8.