

Master project HS Fulda Fall2016

**Milestone 2**

**Group #1**

**17.11.16**

**Team:**

Michelle Rothenbücher

([michelle.rothenbuecher@informatik.hs-fulda.de](mailto:michelle.rothenbuecher@informatik.hs-fulda.de))

Jonas Kleinkauf

Patrick Hasenauer

Martin Herbener

Franz Weidmann

Steffen Scholl

Plisam Ekpai-Laodema

|  |  |
| --- | --- |
| Submission Version | Date |
| 1 | 17.11.2016 |

1. **Use Cases V2**
2. **Data Definitions V2**

This section does describe terms, which will be used in the further development process of project *fuldaflats.de.* These terms will be extended with more implementation details as the development goes on. Furthermore, the structure of the database with its tables is described in this section and a high-level UML diagram of the database is shown.

**Data Glossary**

Name: **Apartment**

Meaning: The apartment is the main object of fuldaflats.de.

Usage: An apartment is used as a trade object between users. It can be rent, bought or sold. Anyone can offer, rent or buy an apartment. In addition, a certain room can be rent or offered too.

Name: **User**

Meaning: A user is a human, which interacts with this application.

Usage: The user will be used with many other terms. Like the apartment, the user can be the new owner or the seller of it.

Name: **User Details**

Meaning: This term is needed to describe a user.

Usage: This term describes properties of a user. A property is e.g. the name or the E-Mail address of the user.

Name: **Attachments**

Meaning: A user should be able to make a better idea of the apartment.

Usage: An attachment is attached to the offer and can be an image of the apartment or a building plan.

Name: **Rating**

Meaning: To ease a user’s decision, there should be some kind of rating.

Usage: Every user can rate the owner of the apartment with a number of zero up to 5 stars. A

Name: **Comments**

Meaning: User should express their feelings regarding issues like the apartment.

Usage: user can write a small text beneath an apartment offer or users profile.

Name: **Offer**

Meaning: The main object of this application (the apartment), attachments, comments, rating and the owner should be grouped into one object.

Usage: An offer gets one page, which will be used to display the above-mentioned objects.

Name: **Owner**

Meaning: describes the user which owns an apartment and offers it

Usage: A trade will be made between an owner and a buyer

Name: **Buyer**

Meaning: describes the user who buys/rents an apartment

Usage: A trade will be made between an owner and a buyer

Name: **Trade**

Meaning: An offer can end in a trade, where the buyer gets the apartments to buy/rent.

Usage: When an owner accepts the buyer, a trade should be established and the offer ends.)

Name: **Message**

Meaning: Users should be able to communicate

Usage: A user can send an E-Mail to another user.

Name: **Account**

Meaning: The application needs an authorization process verify the users

Usage: A user creates an account with his E-Mail Address and a password. The user should verify himself with the password as he wants to login into his account.

Name: **Advertisement**

Meaning: source of income

Usage: Ads can be placed in various location on the webpage

**Table Definitions**

**Table 1: User**

|  |  |
| --- | --- |
| Column name | Description |
| User ID | Unique identifier of the user |
| Email-Address | User signs in with the E-Mail Address |
| Type | Whether the user is a normal or landlord user |
| Password | The hash of the password, which the user signs in with |
| First Name | First name of the user |
| Last Name | Last name of the user |
| Birthday | Date of the user’s birthday |
| Upgrade Date | Date of the upgrade to a landlord user |
| Creation Date | Date of user creation |
| Phone Number | A phone number which can be used to contact the user |
| ZIP Code | The ZIP code of the current user address |
| City | The city name of the current user address |
| Street | The street name of the current user address |
| House number | The house number of the current user address |
| Gender | The gender of the user |
| Offers | List of offer-IDs, which the user created |
| Profile Picture | MediaObject-ID of the user’s profile picture |
| Favorites | List of the offer-IDs, which the user favorites |
| Sessions | List of the session-IDs, which the user is associated with |

**Table 2: Offer**

|  |  |
| --- | --- |
| Column name | Description |
| Offer ID | Unique identifier of the offer |
| Title | Title of the offer |
| Offer type | Whether the offer is about an apartment, sublet, intermediate-rent, couch surfing or party-sleepover |
| Description | Contains a description of the offer |
| Rent | The price in euros to rent the offer |
| Rent type | Whether the rent includes warm water/heating |
| Side costs | Costs which have to be paid additional to the rent |
| Price type | Whether the payment is monthly, daily or per semester |
| Street | The street name of the apartments address |
| ZIP code | The ZIP code of the apartments address |
| House number | The house number of the apartments address |
| Floor | The floor number of the apartments address |
| Size | The size of the apartment in quadrat meters |
| Bathroom Number | Amount of bathrooms in the apartment |
| Bathroom Description | Description of the bathrooms in the apartment |
| Kitchen Description | Description of the kitchen in the apartment |
| Cellar | Whether the offer includes access to a cellar |
| Parking | Whether the offer includes access to one or more parking slots |
| Elevator | Whether the building of the apartment has an elevator |
| Accessibility | Whether the apartment is disability-friendly |
| WLAN | Whether the offer includes access to the internet via WLAN/Wi-Fi |
| LAN | Whether the offer includes access to the internet via a wire |
| Internet speed | The possible maximum internet bandwidth that the offer includes |
| Heating Description | Description of the heating system of the apartment |
| Television | Whether the offer includes access to cable TV |
| Dryer | Whether the offer includes one or more dryer |
| Washing machine | Whether the offer includes access to one or more washing machines |
| Telephone | Whether the offer includes access to a telephone line |
| Status | Whether the offer is hidden, public or closed |
| Creation Date | Date when the offer was created |
| Last Modified | Date when of the last time the offer was changed |
| Longitude | Longitude value to geolocate the apartment |
| Latitude | Latitude value to geolocate the apartment |
| Landlord | User-ID of the user, which created this offer |
| Reviews | List of reviews-IDs, which were created for this offer |
| Tags | List of tags, which are landlord set for this offer |
| Media | List of MediaObject-IDs, which the landlord attached to this offer |
| Statistics | List of Statistic-IDs, which are associated with this offer |

**Table 3: Review**

|  |  |
| --- | --- |
| Colum name | Description |
| Review ID | Unique identifier of the review |
| Rating | Value which represents the rating of an offer |
| Comment | Text which is a comment about an offer |
| Creator | Unique identifier of a user, which created this rating |
| Offer | Unique identifier of the offer, which this rating belongs to. |
| Media | List of MediaObject IDs, which are attached to this rating. |

**Table 4: Tag**

|  |  |
| --- | --- |
| Colum name | Description |
| Tag ID | Unique identifier of the tag |
| Title | Name of the tag |
| Category | Category to which the tag belongs |
| Offers | List of offer IDs, which are associated with this tag |

**Table 5: Favorite**

|  |  |
| --- | --- |
| Colum name | Description |
| Favorite\_ID | Unique identifier of this favorite |
| User | Unique identifier of a user, who favorited an offer |
| Offer | Unique identifier of an offer, which was favorited |

**Table 6: MediaObject**

|  |  |
| --- | --- |
| Column name | Description |
| MediaObject ID | Unique identifier of an media object |
| Type | Whether the media is an image or video |
| Creation Date | Date when the entry was created |
| Main URL | URL that locates the main media file |
| Thumbnail URL | URL that locates the smaller thumbnail media file |

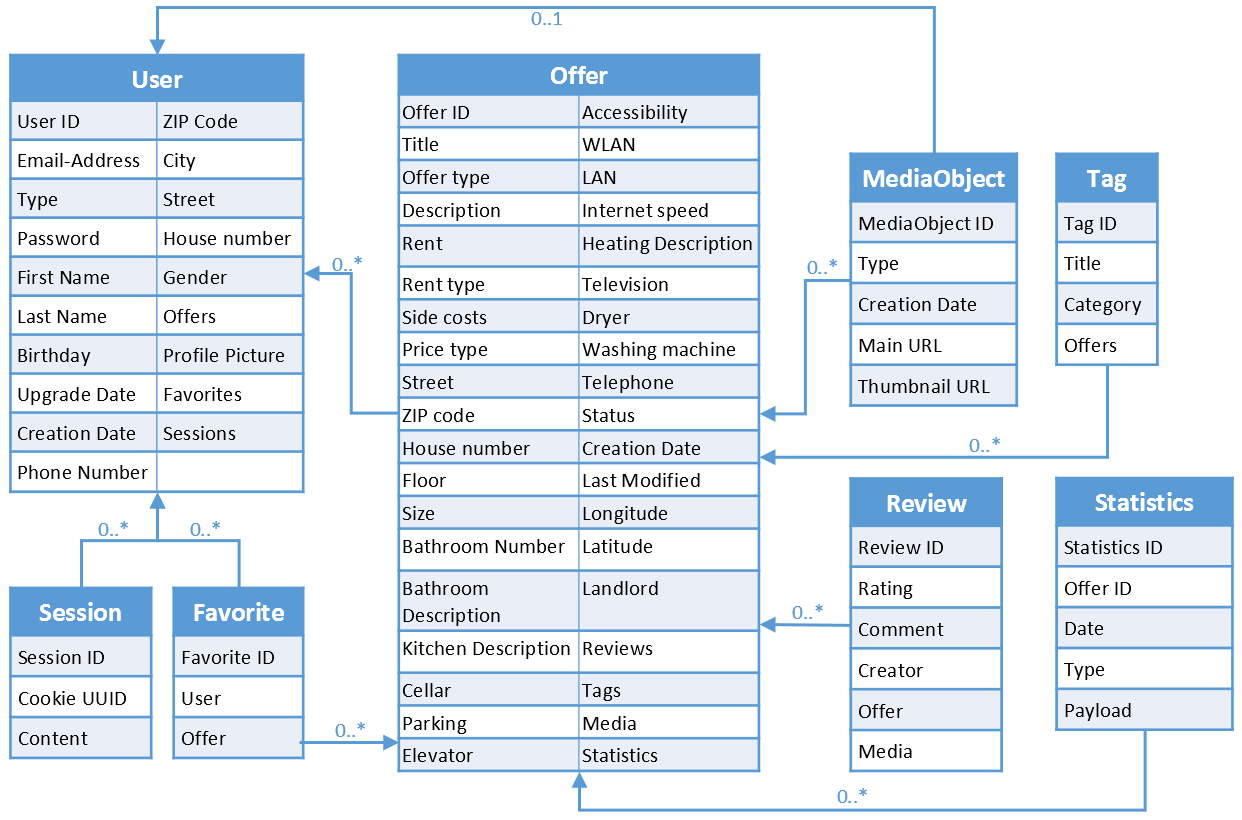
**Table 7: Session**

|  |  |
| --- | --- |
| Colum name | Description |
| Session ID | Unique identifier for the session |
| Cookie ID | ID of the cookie for this session |
| Content | Contains all session relevant data |

**Table 8: Statistics**

|  |  |
| --- | --- |
| Colum name | Description |
| Statistics ID | Unique identifier for the statistic |
| Offer ID | Unique identifier for the offer, which is associated with this statistic |
| Date | Date, when this statistic was created |
| Type | What the statistic is about |
| Payload | Payload of the http(s) packet |

**Database UML Diagram**



1. **Functional Requirements V2**

**1. Sign up (Priority 1)**A user should be able to sign up.

1. The sign-up process should collect some basic information about the user, like his first name, last name, gender, email address and birthday.
2. Additionally, should be set the user account password in the sign up process.
3. After a user signed up, he should be get an account on fuldaflats.de.
4. The “sign up” functionality should be available in the top bar on every page when the user is not signed in.

**2. Sign in (Priority 1)**A user should be able to sign in into his account.

1. The “sign in” functionality should be available in the top bar on every page when the user is not signed in.
2. After a user signed in, he should be get more permissions which are described in the non-functional specs under the permission concept.
3. The default sign in should be able with the account related email address and password.

**3. Create in offer (Priority 1)**Only a sign in user with a landlord account should be able to create an offer.

1. An offer creation should be done with a wizard, which show a small form on each step.
2. After an offer creation, the landlord get displayed the offer details page.
3. The “create in offer” functionality should be available when the user is sign in with a landlord account.
4. The “create in offer” functionality should be available on every page in the top bar and on the offer management page.
5. A special detail for an offer should be a tag able attribute. Available tags for this attribute should be provided from fuldaflats.de. The user should not be able to create his own tags.

**4. Landlord contact information (Priority 1)**A user should be able to get the contact information from a landlord.

1. The contact information from a landlord should be displayed on the offer details page.
2. A required contact information is an email address or a telephone number.
3. An additional contact information is an office address.

**5. Offer attachments (Priority 2)**A landlord should be able to add up to seven images and one video to an offer.

1. Added images and an added video should be displayed in a slider on the offer details page.
2. The first added image or the first frame from the added video should be displayed in the brief description from an offer.

**6. Offer review (Priority 1)**  
A user should be able to create a review to an offer.

1. A review consists a star rating, a comment and optional an image.
2. The “review creation” functionality should be available on the offer details page.
3. All review ratings which are related to a landlord over an offer, should be aggregated to a landlord rating. A landlord rating should be displayed on the landlord profile page, on the offer details page and on the brief description from an offer.

**7. Browse offers (Priority 1)**  
A user should be able to browse a filtered subset from offers.

1. This functionality should be available on the offer filter result page and on the start page, where default filtered offers are displayed.

**8. Offer details page (Priority 1)**A user should be able to see all details from an offer on one page.

1. This page should contain all attachments, all primary details and all other secondary details.
2. Additionally, should this page contain the landlord contact information and all reviews.
3. This page should be open when a user click on the title in the brief description from on offer.

**9. Manage offers (Priority 1)**A landlord should be able to manage his offers. This contains the ability to change offer details and the offer status.

1. This functionality should be available over the landlord account settings page.

**10. Sign out (Priority 1)**  
A sign in user should be able to sign out.

1. The “sign out” functionality should be available in the top bar on every page.
2. After a user used the sign out functionality, he should be redirected to the start page.

**11. Offers overview map (Priority 2)**A user should be able to see the location from an offer on a map.

1. One big map should be displayed on the offers filter result page.
2. Another map should be displayed in the offer details page.

**12. Filter offers (Priority 1)**A user should be able to filter all available offers.

1. A filter panel should be available on the start page, which contains the primary attributes from an offer as a filter.
2. Additionally, should the user be able to expand the default filter panel. The expanded filter panel should display all secondary attributes from an offer as a filter.
3. The filter functionality should be triggered though a button in the filter panel and should be forwarding the user to the offer filter result page.
4. The filter result page should contain a filter panel too.

**13. Manage user account and profile (Priority 2)**A user should be able to manage his account and his profile.

1. This contains the ability to update his profile details and to manage his favorite offers.
2. The user profile should be available over the top bar on every page.

**15. Add and manage favorite offers (Priority 2)**A user should be able to add an offer as a favorite offer and manage his favorite offers.

1. The “add an offer as favorite” functionality should be available everywhere an offer is displayed.
2. Favorite offers should be available over the user account settings page.

**16. Upgrade to a landlord account (Priority 1)**A user should be able to upgrade his account to a landlord account.

1. This functionality should be available in the bar on every page.
2. The upgrade process should collect more information about the user, like phone number, zip code, city, street and house number.

By using this functionality, the user should be get more permissions which are described in the non-functional specs under permission concept.

1. **Non-Functional Requirements V2**

## **General**

1. Application shall be served from the team's account.
2. Pay functionality (how to pay for goods and services) shall be simulated with proper UI, no backend.

## **User Interface**

1. Application shall be very easy to use and intuitive. No prior training shall be required to use the website.
2. Application shall be optimized for standard desktop/laptop browser, and shall render correctly on the two latest versions of all major browsers: Mozilla, Safari, Chrome. It shall degrade nicely for different sized windows using class approved programming technology and frameworks so it can be adequately rendered on mobile devices.
3. The language used shall be English.

## **Infrastructure**

1. Data shall be stored in the MySQL database on the class server in the team's account.
2. The web service application shall be run in a node.js web server like express.
3. Based on the service side application (RESTful service) shall it be possible to build other clients like an android app or a universal windows app.

## **Deployment and Upgrade**

1. An application deployment or an application upgrade shall not take longer than two hours.
2. An application deployment and an application upgrade shall be easy as possible for the administrator. For error-prone or time-consuming configurations shall be provided a configuration script and guidance.

## **Performance**

1. No more than 50 concurrent users shall be accessing the application at any time.

## **Development Tooling**

1. Application shall be developed using class provided LAM stack.
2. Application shall be developed using pre-approved set of SW development and collaborative tools provided in the class. Any other tools or frameworks shall be explicitly approved by Prof. Todtenhöfer on a case by case basis.
3. Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development, and only the tools and practices approved by instructors.

**File Upload**

1. To prevent the server from a user’s spam, an image size is limit of 5MB per file and a video size limit of 30MB.
2. A single offer is only allowed to contain up to 7 images und 1 videos.
3. A review for an offer shall contain only a single image.

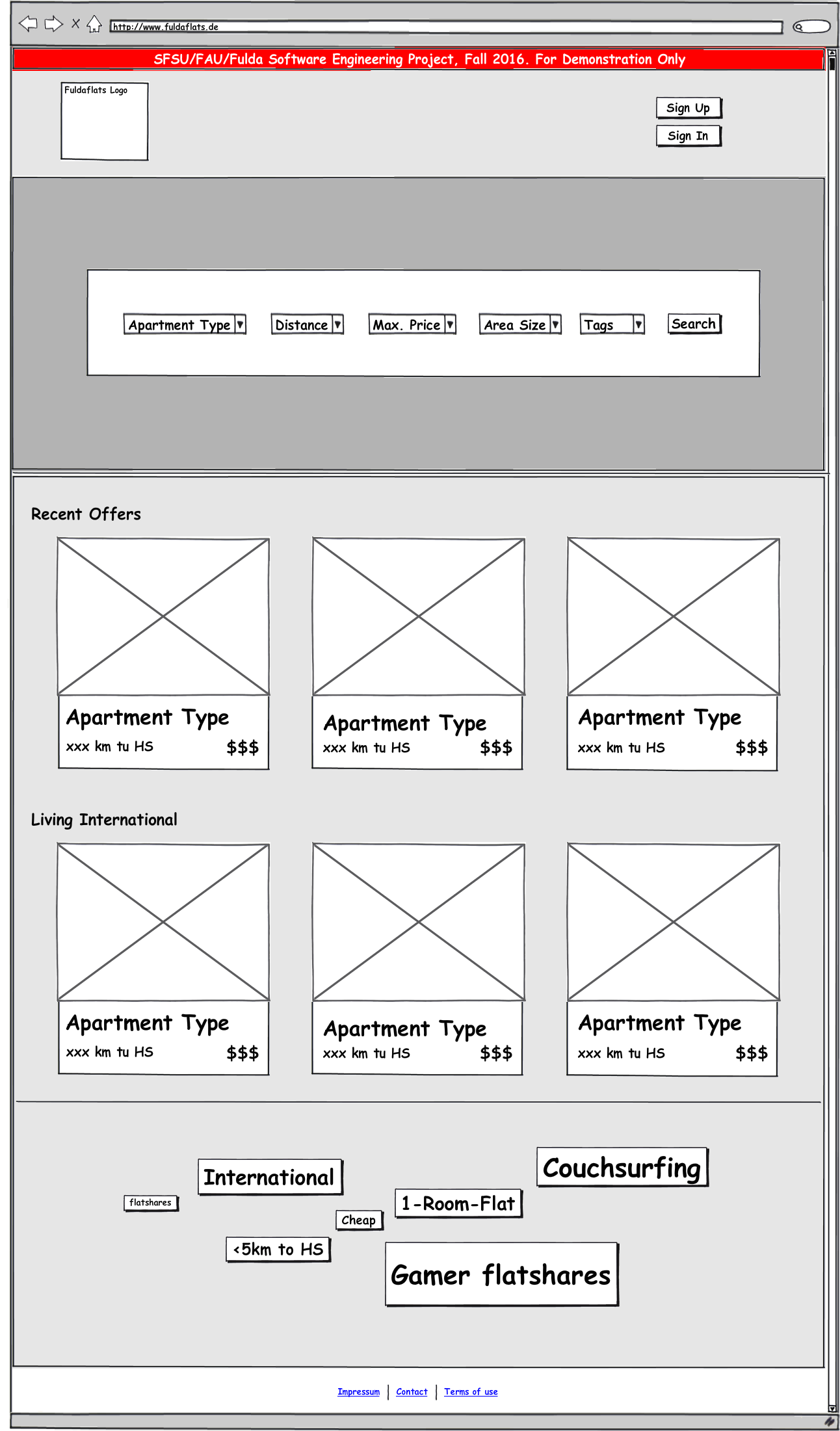
## **Security**

1. Privacy of users shall be protected and all privacy policies will be appropriately communicated to the users.
2. User account passwords shall be saved in the database as hash values.
3. Messaging between users shall be done only by class approved methods to avoid issues of security with e-mail services.
4. Site security: basic best practices shall be applied (as covered in the class).
5. The communication between the client application (browser) and the server application shall be run over HTTPS.
6. A user account shall be blocked after ten following failed sign in tries.

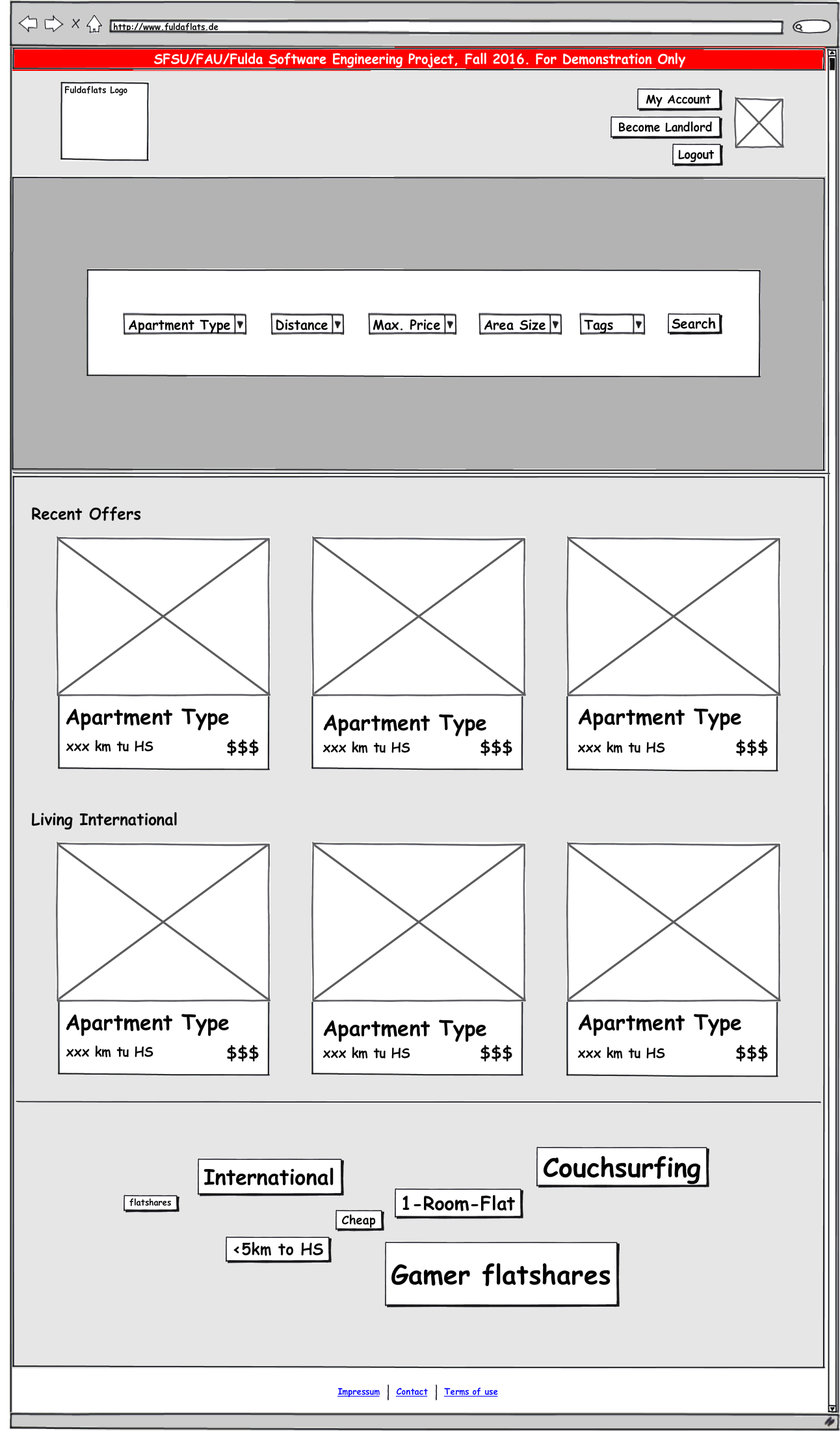
## **Permission Concept**

1. An unregistered user shall be having the permission to …
   1. sing up
   2. filter and browse for offers
   3. see primary details to an offer
2. A registered and singed in user has the same permission as an unregistered user. Additional to this permission he shall be having the permission to …
   1. sign in
   2. see the secondary details to an offer
   3. create a review to an offer
   4. retrieve the contact information from landlord to an offer
   5. create and manage his favorite offers
   6. manage his profile and account
   7. upgrade his account to a landlord account
3. A signed in landlord has the same permission as a registered and signed in user. Additional to this permission he shall be having the permission to …
   1. create an offer
   2. manage his offers
4. **UI Mockups and Storyboards (high level only)**

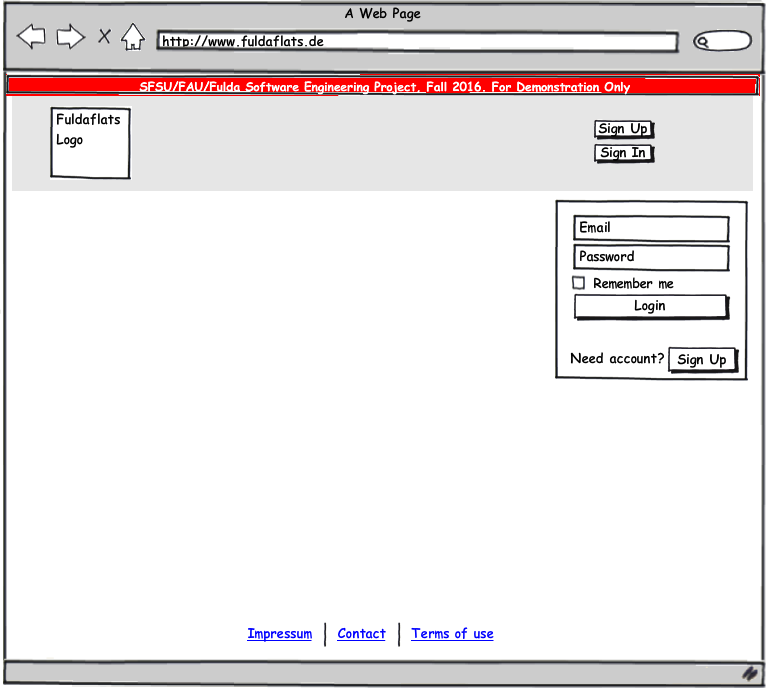
Startpage (User is logged out)



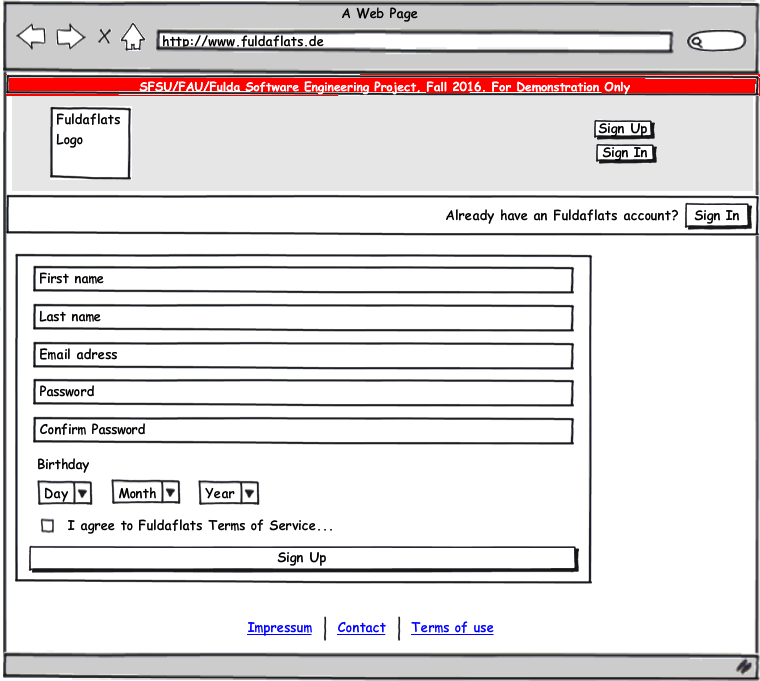
Startpage (User is logged in as standard user)



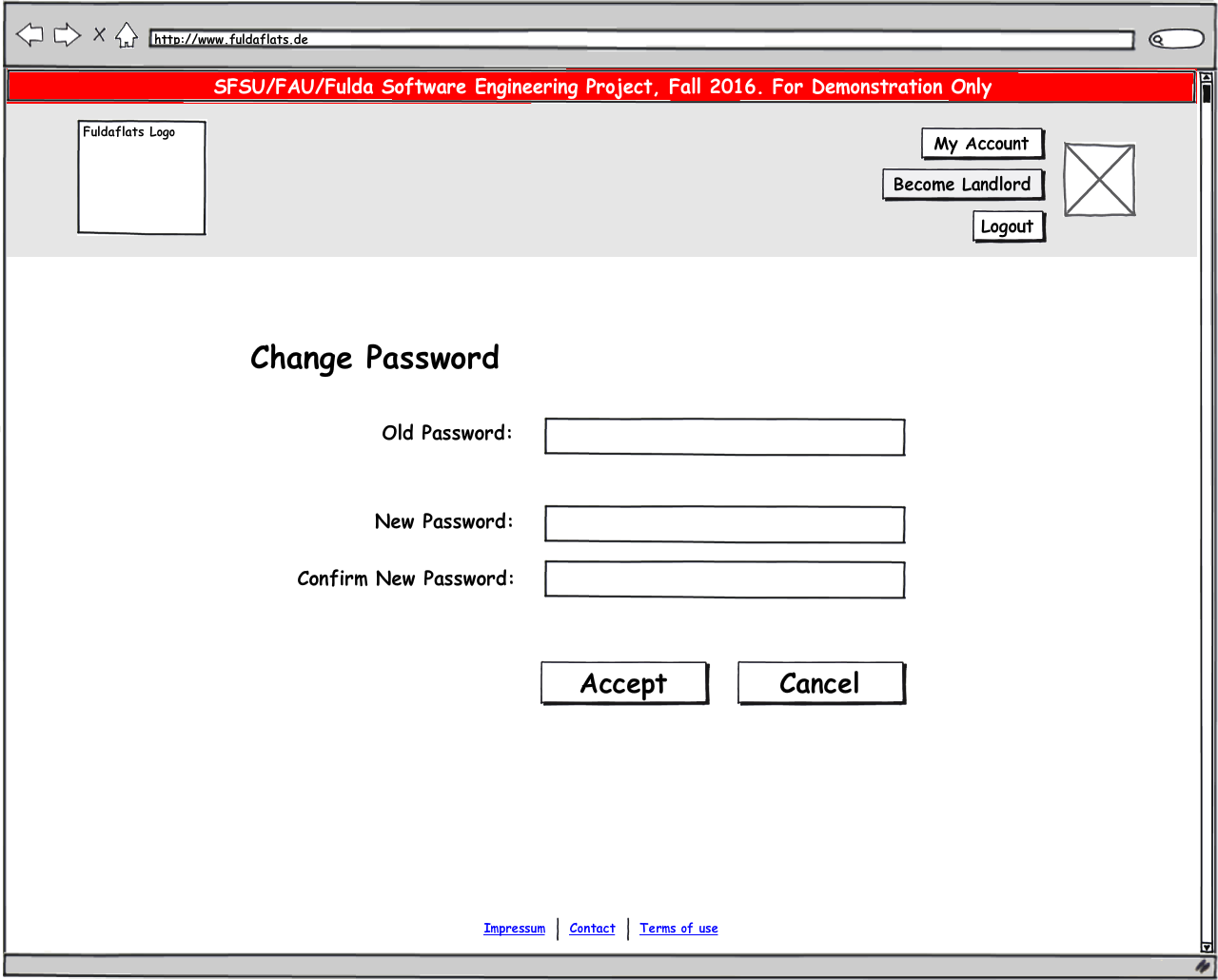
Sign-In page



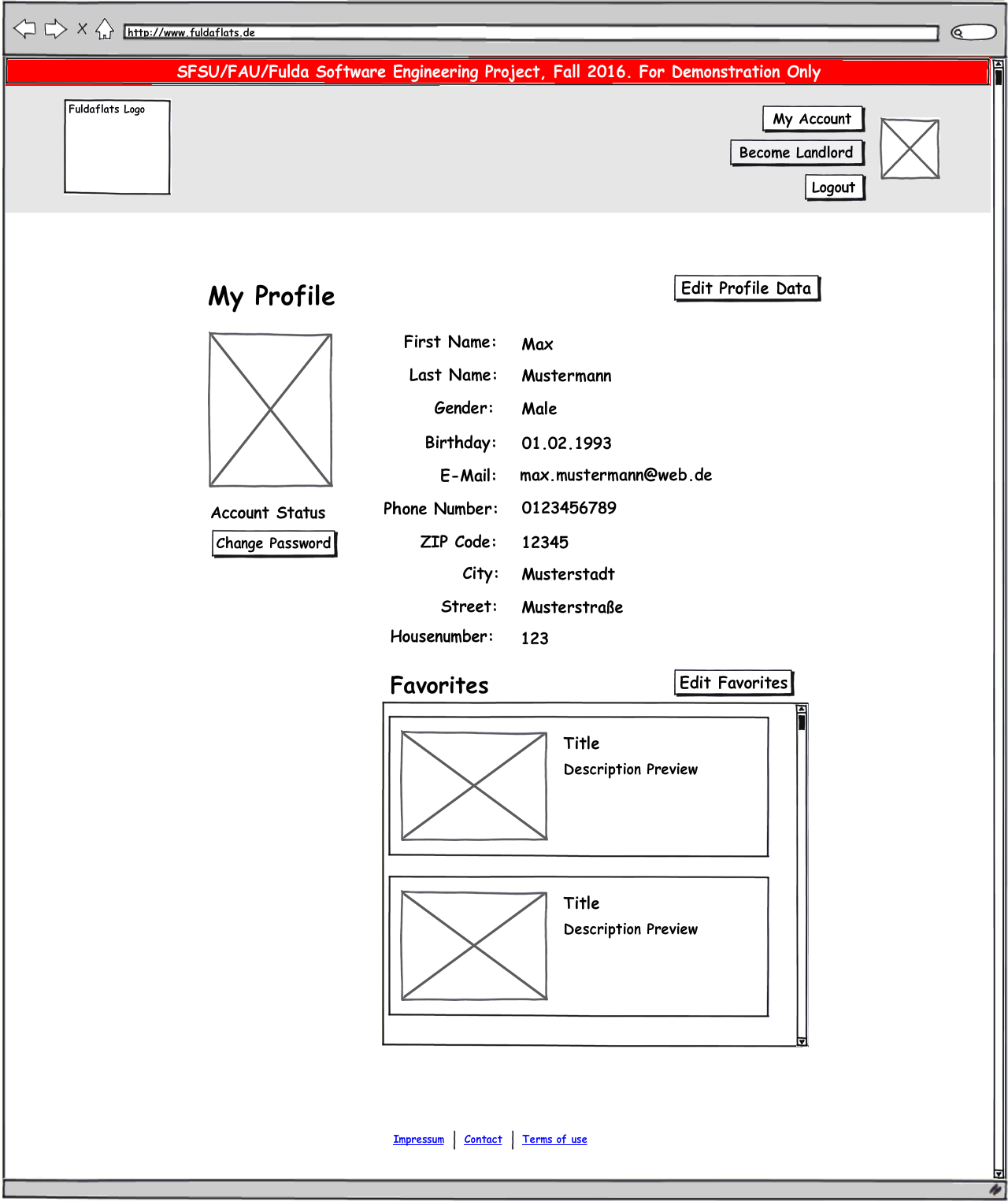
Registration Page



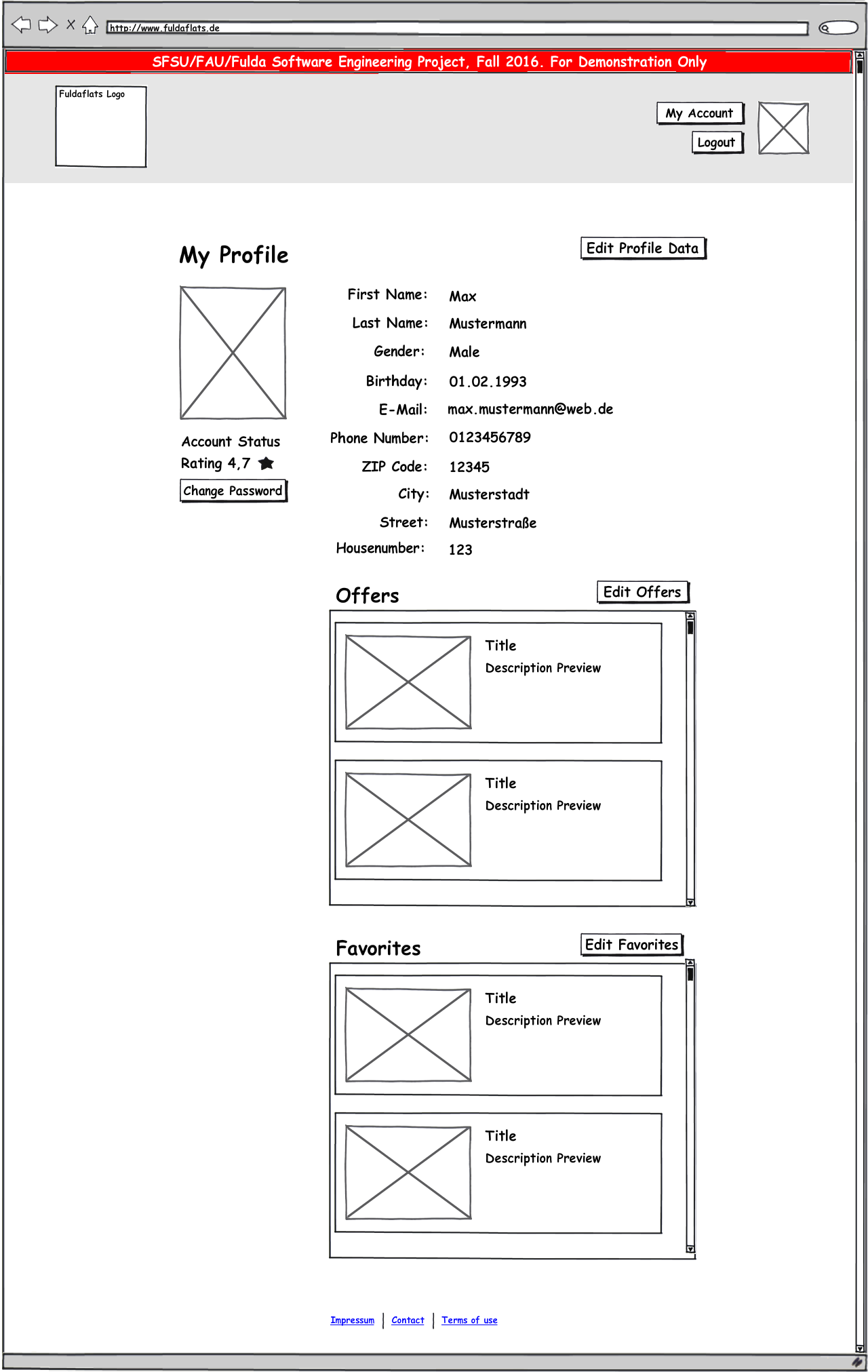
Change Password



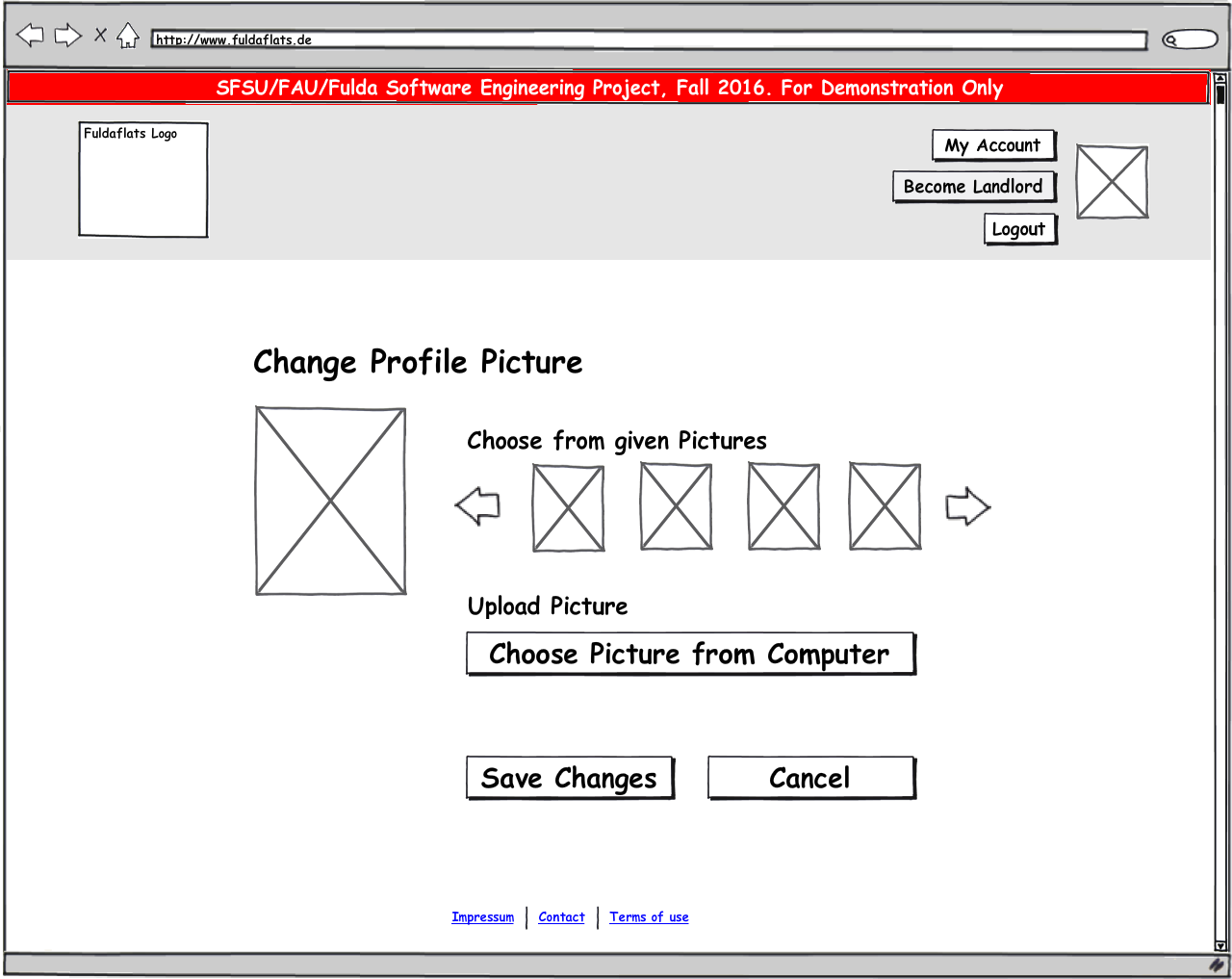
Profile standard user



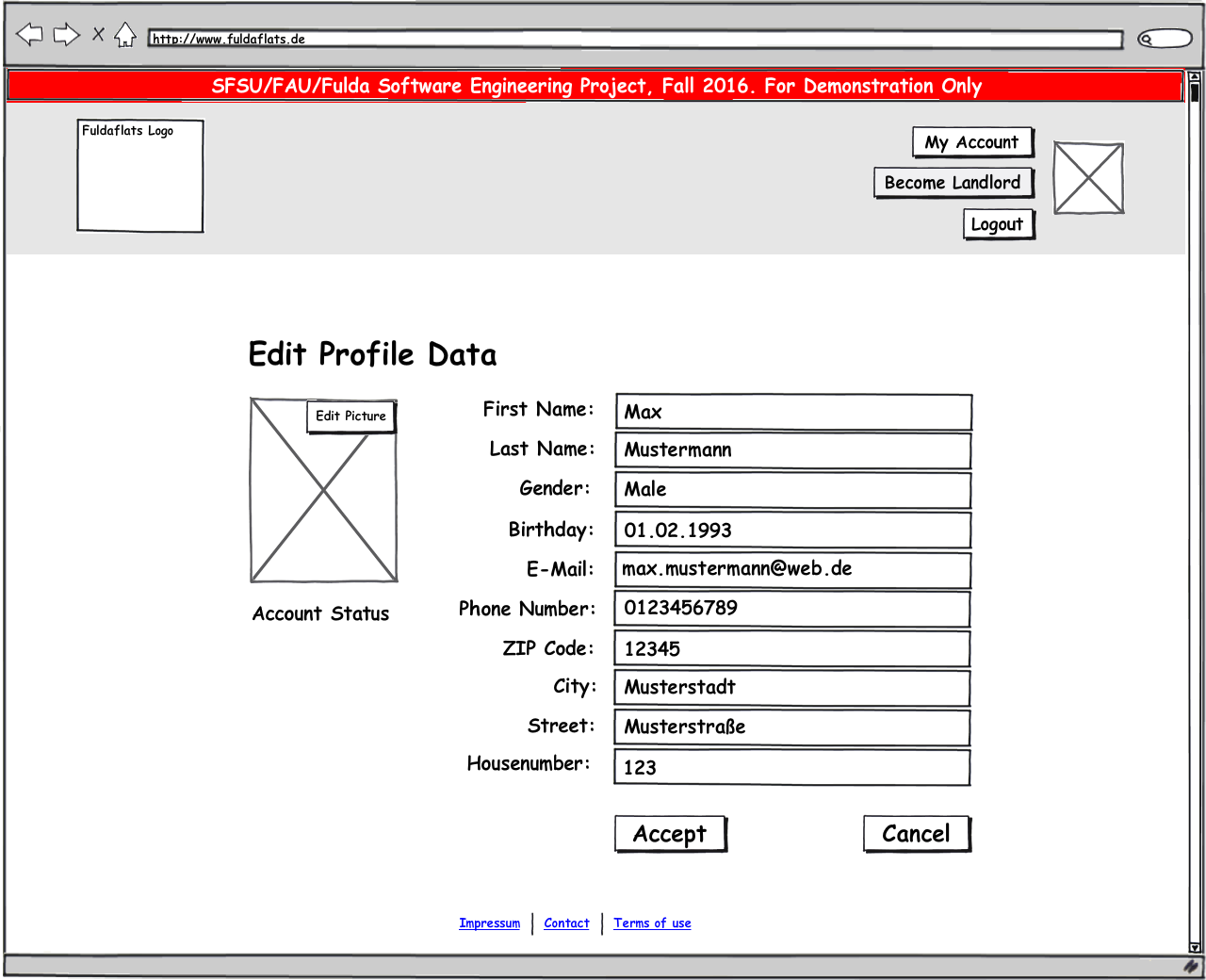
Profile Landlord



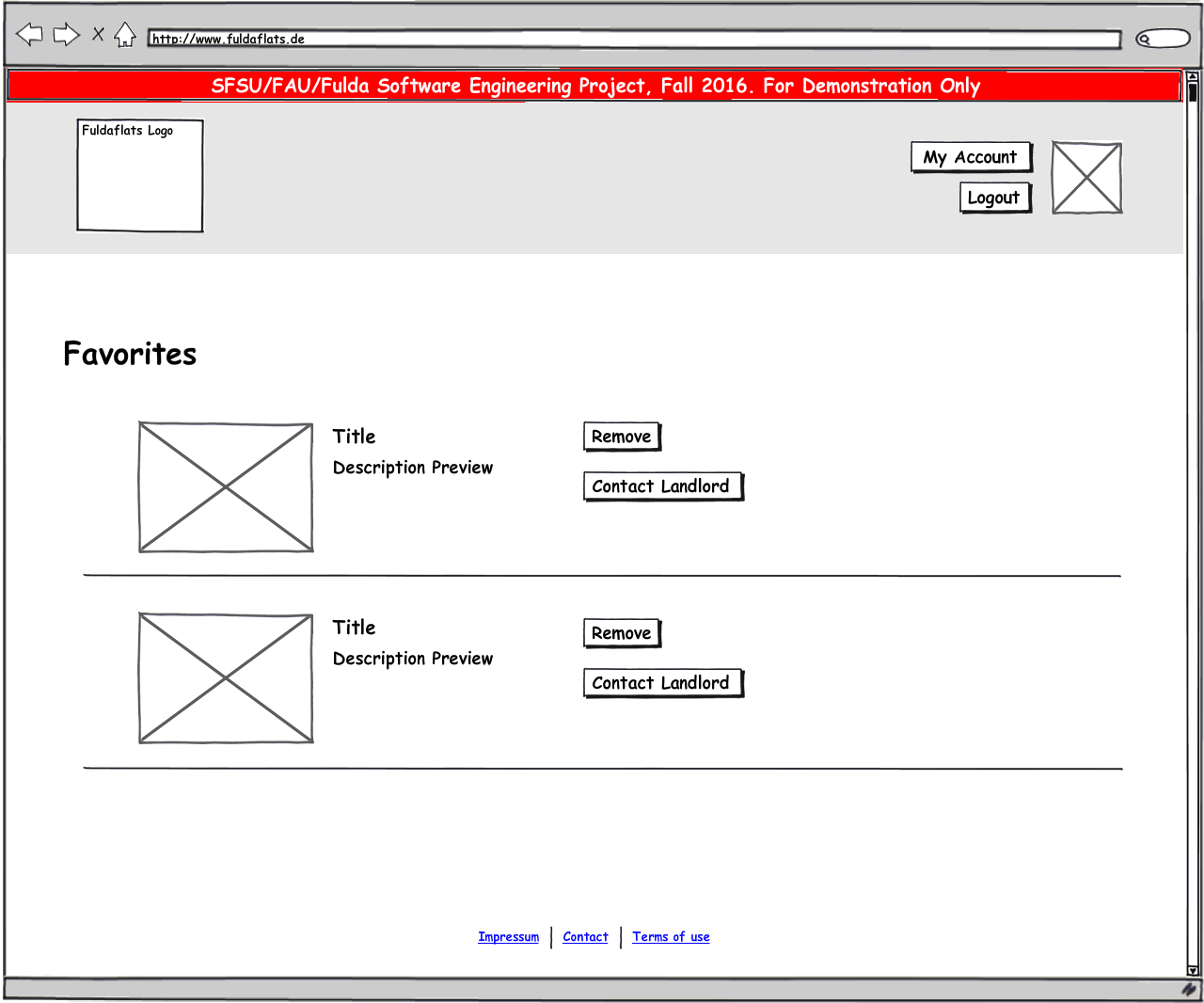
Change Profile Picture



Edit Profile Data



Favorites



1. **High Level Architecture, Database Organization**

This section describes the system architecture and technology stack of fuldaflats.de. It contains the most important software components, frameworks, libraries and development tools that are used in the development process.

**General Architecture Purpose**

The general purpose for our system architecture is to create a loose coupling between the web client in a user’s browser and our server sided program code. We aim to achieve this by using a RESTful web service architectural style. Our server sided code focuses purely on business logic, authentication, authorization and database connectivity. It does not process any kind of HTML templates. Our whole HTML, CSS and JavaScript code will be delivered as static files to the client. Interactivity is created by using client sided JavaScript code.

Client sided JavaScript is used to manipulate the user interface, react to user input and to load data from the server asynchronously using AJAX and JSON technology. Thus, we aim to create a 3-Tier architecture with a database, a thin server layer and a more powerful client.

Through this we get the advantage that we can easily split our team in frontend and backend developers that interfere less with each other’s work and connect to each other via a predefined, standardized HTTP interface. This also makes it very easy to provide test data in JSON format and makes the application more stable and user friendly against server errors (HTTP 500), since this makes it more unlikely that a user sees them.

**Server Foundation Software**

The overall architecture goal for fuldaflats.de is to create a multi-tier web application that uses a REST-Webservice and AJAX technology as main data exchange interfaces. We use a **Linux/Debian** virtual machine that is hosted on the **Microsoft Azure Cloud** as a server for our project.

The data-tier of the application is handled by a **MySQL Database** which can be administrated by the phpMyAdmin user interface.

**Server Sided Programming**

The application / logic tier uses **Node.js 7.0[[1]](#footnote-1)** as technology platform to enable **server-sided JavaScript** coding. Node.js itself is quite a bare software platform, but it can be extended through the package-manager **npm[[2]](#footnote-2)**. Node.js modules that are mandatory for the fuldaflats.de project are **CaminteJS[[3]](#footnote-3)** as an ORM library for database access and **Express[[4]](#footnote-4)** to create RESTful web service endpoints and start a web server within node.js (Those two are comparable to JPA and JAX-RS in Java EE).

**File Upload**

Uploaded image files will be stored in the .png image format directly on the server’s file system (contrary to storing them as BLOBS in the database), because that makes it much easier to backup text-based data in the database and to provide demo data. When providing demo images, they do not need to be stored in the database first to be used. To store the images on the file system, the Express module **multer** will be used, which is capable of handling *multipart/form-data* HTTP requests that contain uploaded images. To prevent the server from a user’s spam, an image size limit of 5MB per file will be programmed. Furthermore, a single flat offer is only allowed to contain up to 7 images.

There are also plans to allow users to upload one video file per flat offer which interested users can then stream from the server. However, we classified this feature as prio2 and it would eventually be developed as a separate web service function.

**Search Algorithm**

The search functionality is one of the key features of fuldaflats.de. It is very prominent on the home page of the application and is refined in the result overview. Since our application only focusses on the area around Fulda, a full-text search of different search criteria is not that necessary (for example to enter a city to search in). Instead, we have a wide variety of predefined search criteria for the user to choose. This includes the apartment type, the distance from the Hochschule Fulda, the price, the size of the apartment and many more (see story boards for more details). While the search function on the home page only lists the most important criteria, a detailed search mask is also available on the results page.

On the backend, this search uses query functions that are included in CaminteJS. The algorithm will first search for the criteria that are chosen by the user and will put the results into a list. This list is then ordered by the creation date of the offer, but the user can also choose another sort criteria in the detailed search mask.

**Client Sided Programming**

For the web client, we use **jQuery[[5]](#footnote-5)** as a standard library to enhance browser APIs, as well as some small JavaScript libraries for handling user input and server connectivity (for example **KnockoutJS[[6]](#footnote-6)** for lightweight DOM-Databinding). We also use **Twitter Bootstrap[[7]](#footnote-7)** as a presentation framework (mainly for its CSS) for responsive web design.

**Map Data**

For showing Map data to the users, we want to use the library **Leaflet.js** which enables interactive maps from different map providers in the browser. We will use **Open Street Map (OSM)** as a free, open source map provider. Geolocation data is stored as float-valued latitude and longitude coordinates. To get the coordinates for a given street name and house number, we want to use the **Open Street Map** lookup service **Nominatim**. We will query Nominatim server sided upon the creation of a flat offer.

**Development Tools**

The fuldaflats.de project uses **git[[8]](#footnote-8)** as a source code management system, the code is hosted as a **private repository on** **GitHub[[9]](#footnote-9)**. GitHub issues and milestones are also used for project management and team communication. We use **Visual Studio Code[[10]](#footnote-10)** as development environment for web applications and node.js (which is not Visual Studio 2015, but rather just a free, enhanced text editor tool from Microsoft).

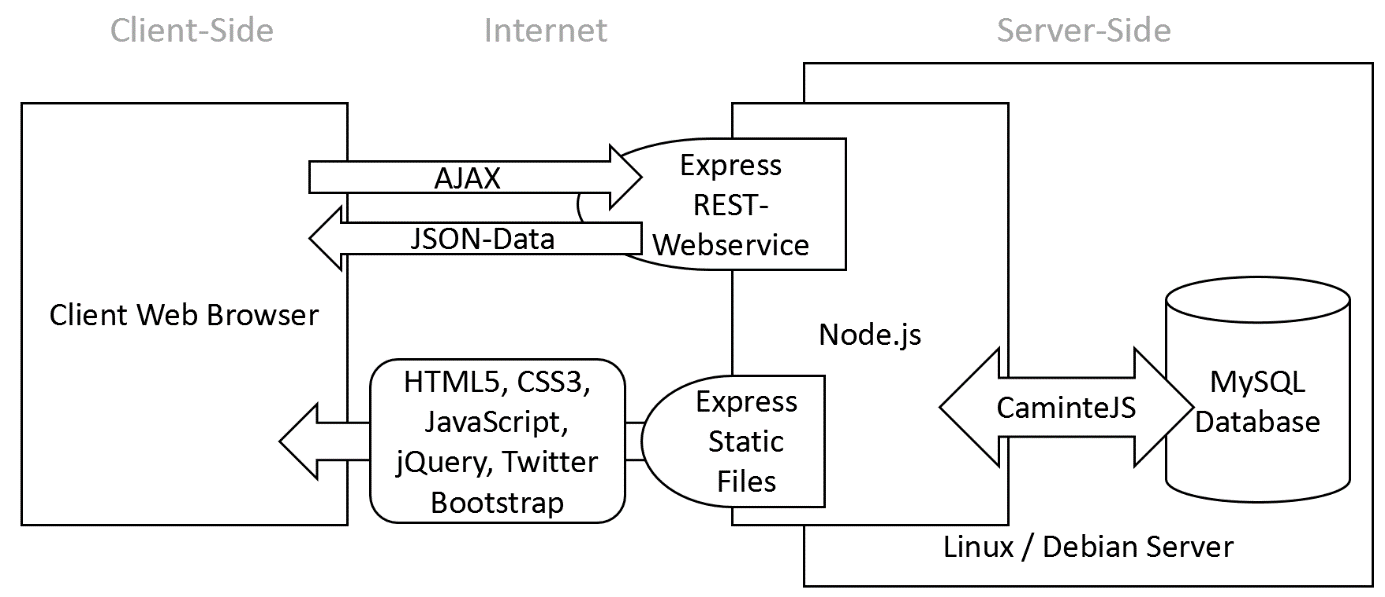
**Compatibility**

The final product will support and be tested on the following Browsers:

* **Google Chrome (Version 54.0)**
* **Mozilla Firefox (Version 42.0)**
* **Apple Safari (Version 10.12)**

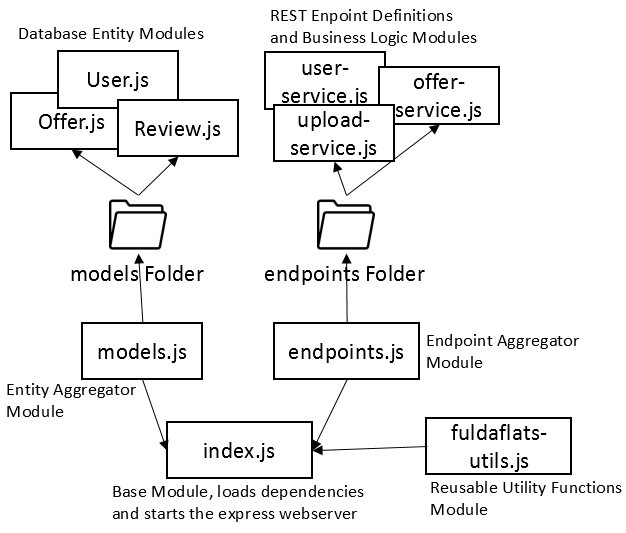
1. **High Level UML Diagrams**

**System Overview Graphic**

****

**Node.js Module Overview**

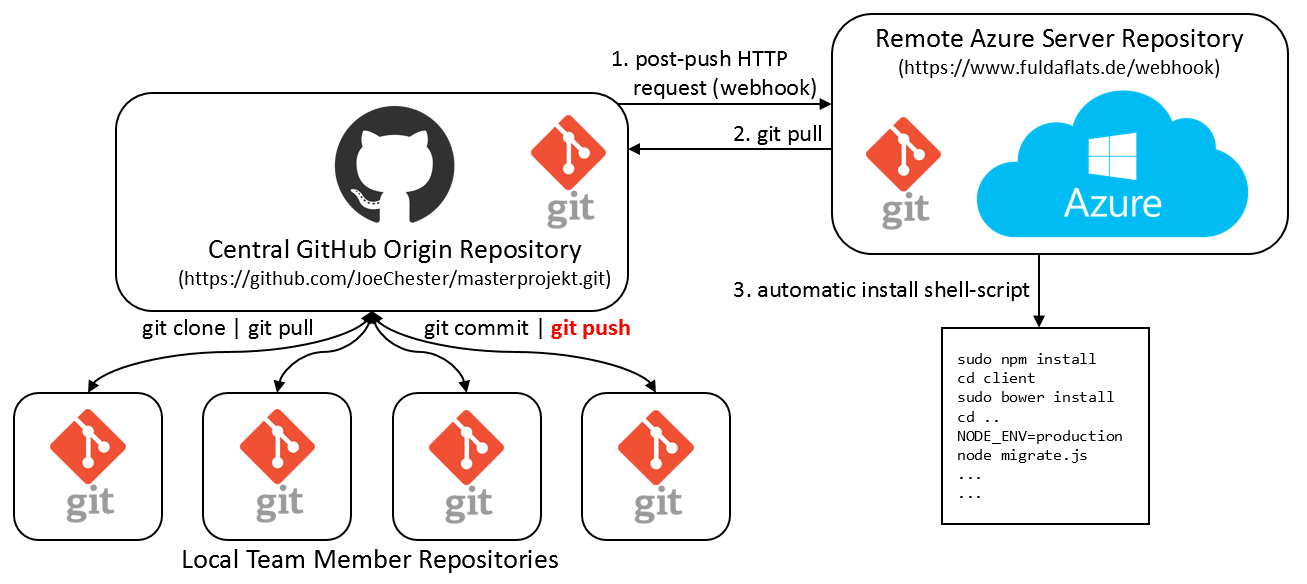
Instead of classes in conventional object oriented programming, Node.js mainly uses modules to divide program code into self-contained parts. The following diagram gives an overview of how the codebase of our server side will be organized.



**Deployment**

The deployment process to get our application running on the Azure cloud server is fully automated by incorporating GitHub’s webhook feature. We configured our GitHub repository in that way that every time a developer pushes something into the GitHub origin repository, GitHub automatically sends a specific POST http request to our server that runs on Microsoft Azure. This POST request is handled by a little node.js script that checks the content of the request for a secret passphrase that we configured on GitHub and runs a shell installation script afterwards if the passphrase is correct.

This shell script pulls the latest commits from the GitHub, installs all dependencies that are configured in npm, migrates the database, inserts demo data into the database and restarts the server application.

****

1. **High Level APIs**

This subsection describes the various REST API Endpoints of the Fuldaflats.de server code. The client accesses these endpoints by sending asynchronous HTTP Requests (AJAX) and retrieves JSON formatted data back. Colons are used to mark path parameters like **:id**.

API Methods that are marked blue in these tables require a logged in user to work, otherwise they will just return a **403 Forbidden** status code.

**User related endpoints**

|  |  |  |
| --- | --- | --- |
| **Method** | **Path** | **Description** |
| **POST** | /api/users/auth | Authenticate current Session with login data |
| **POST** | /api/users | Create a new User (Registration functionality) with given JSON data in the body |
| **GET** | /api/users/me | Retrieve own user data (when logged in) |
| **PUT** | /api/users/me | Manipulate own user data (when logged in) with given JSON data in the body |
| **GET** | /api/users/:id | Get user data of user with :id (PRIO 2, for future use) |
| **DELETE** | /api/users/auth | Logout (delete authentication in current session) |

**Offer related endpoints**

|  |  |  |
| --- | --- | --- |
| **Method** | **Path** | **Description** |
| **POST** | /api/offers/search | Most important API function of the application. Search for a list of offers based on search criteria that is sent in the body in JSON format. |
| **GET** | /api/offers/recent | Get list of recent offers |
| **GET** | /api/offers/:id | Get detailed information on the offer with given :id. (More information when session is authenticated) |
| **POST** | /api/offers | Create a new offer with given JSON data in the body. Requires the account to be upgraded to landlord. |
| **PUT** | /api/offers/:id | Manipulate the data of the offer with given :id with given JSON data in the body under the prerequisite that the offer is owned by the currently logged in user |
| **DELETE** | /api/offers/:id | Delete the offer with given :id und the prerequisite that the offer is owned by the currently logged in user |
| **POST** | /api/offers/:id/review | Post a review for the offer with the given :id when a user is logged in |
| **DELETE** | /api/offers/:id/review | Delete a posted review for offer with given :id with the prerequisite that the review was created by the currently logged in user |
| **PUT** | /api/offers/:id/favorite | Mark the offer with given :id as favorite for the currently logged in user |
| **DELETE** | /api/offers/:id/favorite | Unmark the offer with given :id as favorite for the currently logged in user |
| **GET** | /api/tags | Get a list of all available tags for filtering |

**File upload related endpoints**

|  |  |  |
| --- | --- | --- |
| **Method** | **Path** | **Description** |
| **POST** | /api/files | Upload a file as multipart/form-data onto the file server system. Retrieve the MediaObject link when the upload is completed |
| **GET** | /uploads/:filename | Get the uploaded file with given :filename (this is the link that is stated in the MediaObject responses) |
| **DELETE** | /uploads/:filename | Delete file resource with given :filename from the server, given it was uploaded by the currently logged in user |

1. **Key Risks**

## **Skills risks**

Our group consists of two people with significant experience in web development, three people with network/server management experience, one in project management, one in testing and we are all strongly motivated to learn web technologies. Each one of us has the basis to appropriate the new web technologies we need for our project.

However, the project team needs time to acquire new skills for the project. There is a risk that productivity will be low. The quality training for certain skills can be difficult to secure because of the short period available to learn them.

## **Schedule risks**

## Working in a group of 8 makes the scheduling very complex and difficult. We use GitHub for tasks and project management. It offers all of the distributed version control and source code management (SCM) functionality of GIT as well as adding its own features. So far it work well, thanks to our expert in project management.

Some risks like prioritized non-essential changes, continually schedule extending, dependencies impact from each project, fair tasks sharing must be well managed.

## **Technical risks**

## The tools which we are using are well documented and tested. We have them already installed and tested on each team member‘s personal computer. The technical leader ensures that all details are well documented.

Technical problems with the project management tools, development tools compatibility, platforms tools or technology components maintaining difficulties, integration with legacy components that are no longer in support, components that are difficult to extend with new capabilities, may complicate the realization of the project.

## **Teamwork risks**

## Our team is doing a very good job of management, installation, testing and communications. Everyone has accomplished his/her tasks within the time deadlines. The use of instant messaging has enabled us to create an interesting sharing dynamics.

It should however be noted that misinterpretation by the project team, low team motivation, unfavorable communication may cause the project to fail. Team leaders must be patient because, they may need to communicate the same idea many times in different ways before people remember it.

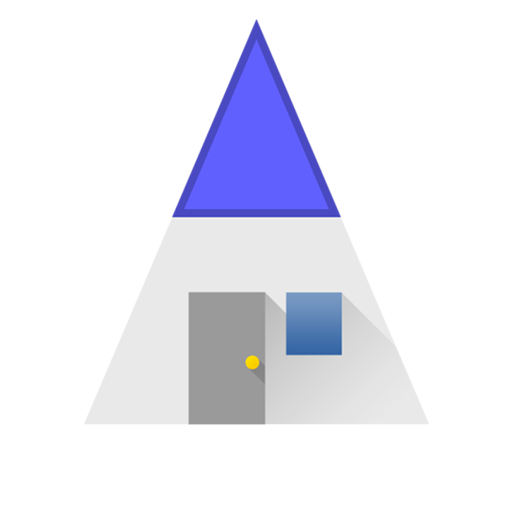
## **Legal and content risks**

## The decision, to focus on open source technologies, tools and free contents, distant all legal and copyright risks in connection with the completion of this project.

Media / Videos related to this project are exclusively free license and special material produced are only for the project.

**10. Team Organization**

|  |  |
| --- | --- |
| Name | Initial Role |
| Michelle Rothenbücher | Team Lead / Frontend |
| Jonas Kleinkauf | Technical Lead / Backend Lead |
| Patrick Hasenauer | Frontend Lead |
| Martin Herbener | Server Admin / Frontend |
| Franz Weidmann | Backend / Frontend |
| Steffen Scholl | Backend / Frontend |
| Plisam Ekpai-Laodema | Backend / Frontend |



1. <https://nodejs.org> [↑](#footnote-ref-1)
2. <https://www.npmjs.com> [↑](#footnote-ref-2)
3. <http://www.camintejs.com> [↑](#footnote-ref-3)
4. <http://expressjs.com> [↑](#footnote-ref-4)
5. <https://jquery.com> [↑](#footnote-ref-5)
6. [http://knockoutjs.com](http://knockoutjs.com/) [↑](#footnote-ref-6)
7. <http://getbootstrap.com> [↑](#footnote-ref-7)
8. <https://git-scm.com> [↑](#footnote-ref-8)
9. [https://github.com](https://github.com/) [↑](#footnote-ref-9)
10. <https://code.visualstudio.com> [↑](#footnote-ref-10)