

SW Engineering CSC648/848 Section 01 Fall 2016

Application: Gator Renter

Milestone 2

Oct. 25th, 2016

Rev 1.0

Team 08

Alex Aichinger (aichinger.alex24315@gmail.com)

Cheng Li

Darrel Daquigan

Jiri Loffelmann (CTO)

CONTENT

1	Use Cases	1
1.1	Case 1: Student Saves Apartment for Later Viewing.....	1
1.2	Case 2: Renter Rents out an Apartment	1
1.3	Case 3: Student Rents an Apartment.....	2
2	Data Definition	3
2.1	Apartment Model.....	3
2.2	User Model	4
3	List of Functional Specs	5
3.1	Priority 1	5
	Guests	5
	Registered Renters	5
	Registered Landlords	5
	Administrators	5
3.2	Priority 2	5
	Guests	5
	Registered Renters	5
	Registered Landlords	5
3.3	Priority 3	5
	Guests	5
	Registered Renters	6
	Registered Landlords	6
4	List of Non-Functional Specs	7
5	UI Mockups.....	8
6	High-Level System Architecture	10
6.1	Technologies Used	10
6.2	Architecture Diagram	10
7	High Level UML Diagrams	11
8	REST API Documentation.....	12
8.1	Apartment	12
8.2	User	16

9	Key Risks.....	19
9.1	Skill Risks:	19
9.2	Schedule Risks:	19
9.3	Technical Risks:.....	19
9.4	Teamwork Risks:	19
10	Team Members.....	20

1 Use Cases

1.1 Case 1: Student Saves Apartment for Later Viewing

Janice is a third year **undergraduate** at SFSU, majoring in Political Science. She lives in an apartment just off campus, but her lease is almost up with no option to renew. She **does most of her shopping online**, so she is very **competent navigating web pages**. She visits Gator Renter, and **on the first page, there are already apartments listed along with their addresses and prices**. However, the listed apartments are out of her price range. She sees **a list of filter categories**, and chooses to browse by apartments priced between \$1000 and \$1500. After that, the page refreshes and instead of showing the apartments that are too expensive for her, all the listed apartments are within the \$1000 - 1500 rent range. She sees one she likes, so she clicks on the listing.

This **brings up a page with more details about the apartment**. After looking at the pictures and the videos that were posted on the listing, she is interested in renting the place. However, it is almost time for class, and does not have the time to finish the process. **She signs up for an account and saves the listing**. She logs out and goes to class, and when she is done with class hours later, she logs back in. The apartment she saved hours ago is still listed in her account and she can now resume where she left off. Before applying for the apartment, she **finds the email of the renter and starts a line of communication**.

1.2 Case 2: Renter Rents out an Apartment

Shawn is a 70 year old retired cop from Chicago, and has a vacant room near the SFSU campus that he would like to rent out. He is **not the most tech savvy**, but he did hear that **our website was easy to use**, so he gives it a shot. He **signs up for an account and starts entering information about his apartment**. He completes the form easily. After submitting the form, he is prompted to post pictures and an optional video. After finding the files in his cluttered desktop, he easily uploads the files. He finishes the last form, and **waits for the website administrator to approve this listing**. Soon he **receive an email notifying the listing is approved** and his house is listed on Gator Renter.

Shawn **receives emails from students who want to check out the place**. He arranges for them to check the place out. In a few days, he sees a list of people applying for the apartment. Two of the applicants, Judy and Rick, love the place and want to rent it. On Shawn's account page, he can check out the profiles of Judy and Rick. These profiles help him decide to choose to rent his apartment to Judy. Later that day, his account balance now includes Judy's initial deposit, and he can transfer the money to his bank account.

1.3 Case 3: Student Rents an Apartment

Mark is looking for an apartment with his friend Dominic. Both are first year graduate students at SFSU. To save money, Mark always shops online for older editions of his textbooks, **so he is comfortable using web applications**. Mark already has an account with us. He **used the filters to find and apply to a few apartments with two bedrooms**. Mark was planning to take his friends to one of the apartments he was interested in renting, but he **gets an email notifying him that the apartment has been rented out and is no longer vacant**. So, instead of wasting their time looking at a place they could not rent, they spent their afternoon at the beach. After the beach, Mark **emails the tenants on his list of apartments inquiring about his application**. Within the next few days, Mark gets **email notifications about apartments he did not get into, but he gets one email telling** him that his application was accepted by one of his top choice apartments. Mark account now shows that he has been accepted to rent

The apartment at 123 College St. is **one of Mark's top choice apartments**, and he immediately wants to rent it. He commits to the rental, and is given a move-in date. He now **has account balance of \$2000 which is the cost of the initial deposit**. After pooling money with Dominic, Mark uses Gator Renter to pay his balance. Now, Mark and Dominic can rest easy knowing that their living situation has been resolved.

2 Data Definition

The application will consist of Apartment and User models. The tables below specify fields in each model, not a database tables. Design of the database is irrelevant in this high level description and will be taken care of and specified later. The purpose of this section is to define model fields and establish consistency in naming.

2.1 Apartment Model

Defines data fields for apartment postings.

Machine Name	Data Type	Note
id	int	
active	boolean	
created_at	date	<i>Automatic, when was the model created in db</i>
updated_at	date	<i>Automatic, when was the model last updated</i>
address_line_1	string	
address_line_2	string	
street	string	
city_id	int	
state_id	int	
country_id	int	
zip	string	<i>Must be a string, some zips start with '0'</i>
description	text	
sq_feet	double	
nr_bedrooms	int	
nr_bathrooms	int	
nr_roommates	int	
floor	int	
private_bath	boolean	

Machine Name	Data Type	Note
kitchen_in_apartment	boolean	
monthly_rent	double	
security_deposit	double	
pictures	array of strings	
move_in_date	date	
lease_end_date	date	
flagged	array of dates	

2.2 User Model

Defines data fields for users of the application.

Machine Name	Data Type	Note
id	int	
active	boolean	
created_at	date	<i>Automatic, when was the model created in db</i>
updated_at	date	<i>Automatic, when was the model last updated</i>
password	string	<i>Hashed string</i>
first_name	string	
last_name	string	
email	string	
role_id	int	<i>Admin / Regular</i>
school_id	int	
major_id	int	
expected_grad_date	date	

3 List of Functional Specs

3.1 Priority 1

Guests

- A non-technical guest shall be able to browse, filter or search the listings easily

Registered Renters

- Registered renters (including those non-technical renters) shall be able to browse, search, and filter the listings.
- Registered renters shall be able to store their information for a quick application process to landlords

Registered Landlords

- Registered Landlords shall be able to check the prospective renter interest with minimal effort.
- Registered landlords (including those cannot use computer and web fluently) shall be able to post their houses within minimal steps to ensure potential landlords are not discouraged from using the website
- Landlords shall be able to upload their choice of pictures or videos to show potential renters, in a simple and minimal step process.

Administrators

- Administrators shall be able to thwart spammers by putting in a system that discourages them from using the website

3.2 Priority 2

Guests

- Guests shall be able to look at listings with a map view or list view

Registered Renters

- Registered renters shall be able to communicate with the registered landlords
- Registered renters shall be able to save the rental properties that they would like to bookmark quickly and easily

Registered Landlords

- Registered landlords shall be able to communicate with the registered renters

3.3 Priority 3

Guests

- Guests shall be able to filter the listing houses based on some specified criteria (some SFSU specific features....)

Registered Renters

- Registered Renters shall easily see how what their current balance is for rent
- Registered Renters shall be able to see a schedule of payments they owe
- Registered renters shall be able to make a payment easily and quickly

Registered Landlords


- Registered Landlords shall be able to see a schedule of payments they receive
- Registered landlords shall be able to withdraw the money into their bank account

4 List of Non-Functional Specs

- Application shall be developed using class provided LAMP stack
- 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 Marc Sosnick on a case by case basis.
- Application shall be hosted and deployed on Amazon Web Services as specified in the class
- Application shall be optimized for standard desktop/laptop browsers, and shall render correctly on the two latest versions of all major browsers: Mozilla, Safari, and 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
- Data shall be stored in the MySQL database on the class server in the team's account
- Application shall be served from the team's account
- No more than 50 concurrent users shall be accessing the application at any time
- Privacy of users shall be protected and all privacy policies will be appropriately communicated to the users.
- The language used shall be English.
- Application shall be very easy to use and intuitive. No prior training shall be required to use the website.
- Google analytics shall be added for major site functions.
- Messaging between users shall be done only by class approved methods to avoid issues of security with e-mail services.
- Pay functionality (how to pay for goods and services) shall be simulated with proper UI, no backend.
- Site security: basic best practices shall be applied (as covered in the class)
- 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
- The website shall prominently display the following text on all pages "SFSU/FAU/Fulda Software Engineering Project, Fall 2016. For Demonstration Only". (Important so as to not confuse this with a real application)

5 UI Mockups

Homepage

GatorRenter.com

POST AN APARTMENT

Search

FILTERS

☒ Private Room

☐ Private Bathroom

☐ Kitchen in Apartment

☐ No Security Deposit

☐ No Credit Score Check

NEIGHBORHOOD

Marina

NUMBER OF ROOMMATES

Less than 2 roommates

PRICE

MIN


MAX

45 RESULTS

LIST VIEW

MAP VIEW

SORT BY PRICE




1 day ago

\$1,200

Ashton San Francisco Apartments

1 roommate, private room, private bath

APPLY NOW



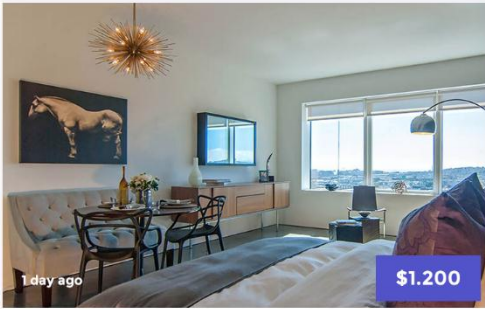
2 days ago

\$950

Ashton San Francisco Apartments

1 roommate, private room, private bath

APPLY NOW




1 day ago

\$1,200

Ashton San Francisco Apartments

1 roommate, private room, private bath

APPLY NOW



2 days ago

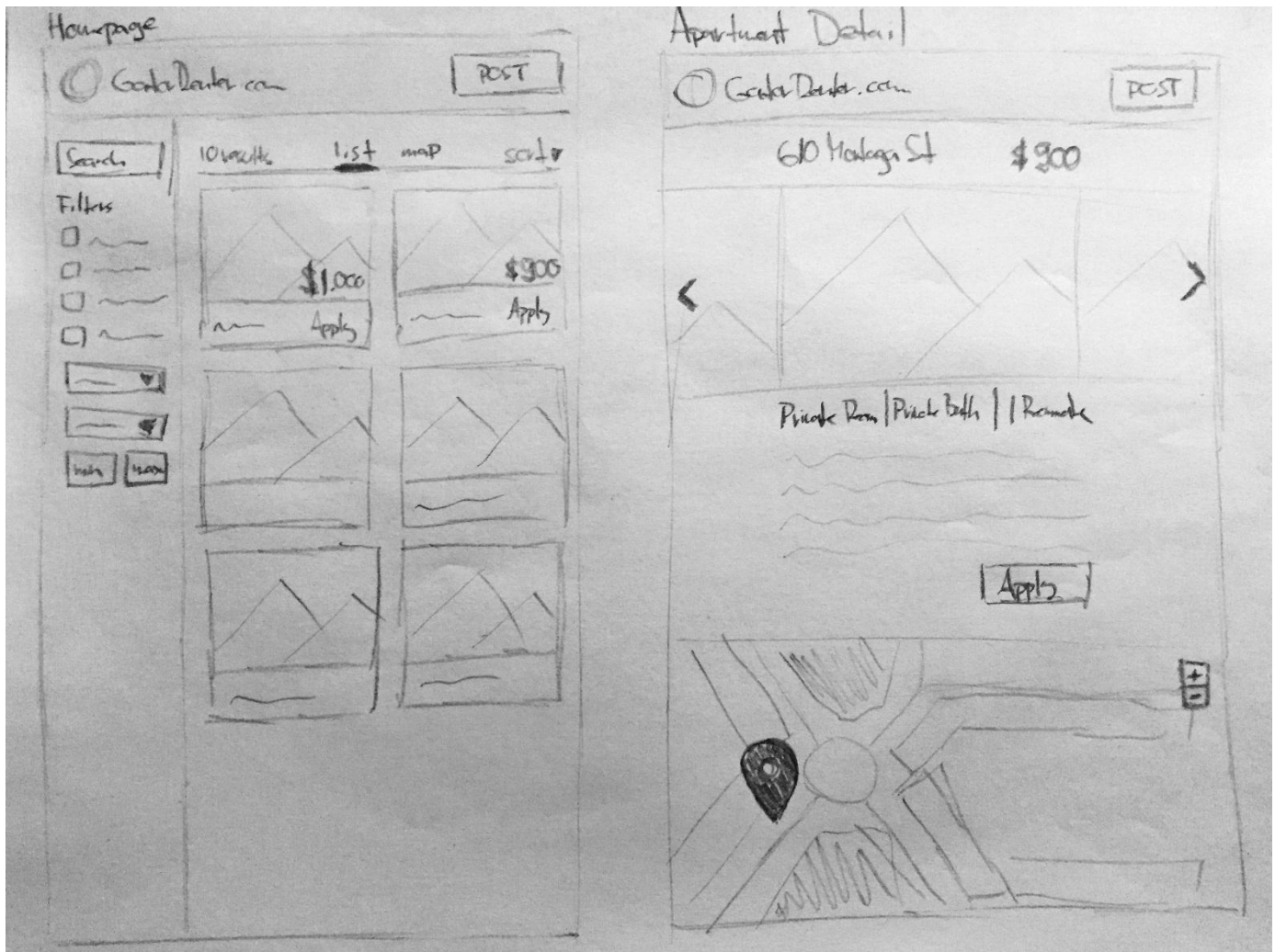
\$950

Ashton San Francisco Apartments

1 roommate, private room, private bath

APPLY NOW

Homepage & Apartment Detail



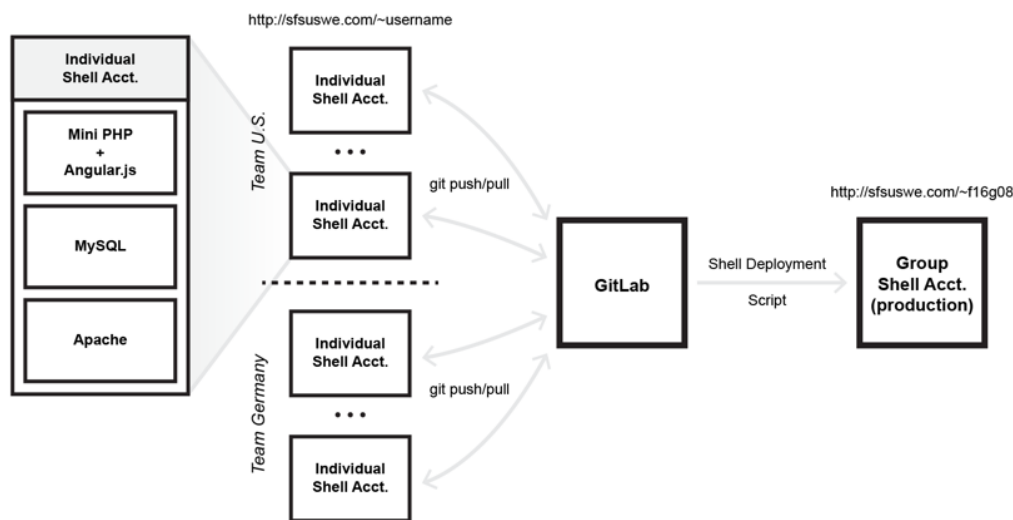
6 High-Level System Architecture

The application will be built on a standard LAMP stack, with 'P' representing the Mini PHP framework. Every team member has his own LAMP instance on the sfsuswe server, connects to it from local machine using SSH, develops and pushes updates to shared GitLab repository. Team Lead or team CTO deploy from GitLab to shared group shell account using a deployment script.

6.1 Technologies Used

- LAMP Stack
- Mini PHP Framework
- Angular.js (communication / data handling with RESTful API)
- GitLab

6.2 Architecture Diagram



7 High Level UML Diagrams

Apartment

- + id : int
- + active : boolean
- + created_at : date
- + updated_at : date
- + address_line_1 : string
- + address_line_2 : string
- + street : string
- + city_id : int
- + state_id : int
- + country_id : int
- + zip : string
- + description : text
- + sq_feet : double
- + nr_bedrooms : int
- + nr_bathrooms : int
- + nr_roommates : int
- + floor : int
- + private_bath : boolean
- + kitchen_in_apartement : boolean
- + monthly_rent : double
- + security_deposit : double
- + pictures : string[]
- + move_in_date : date
- + lease_end_date : date
- + flagged : date[]

User

- + id : int
- + active : boolean
- + created_at : date
- + updated_at : date
- + password : string
- + first_name : string
- + last_name : string
- + email : string
- + role_id : int
- + school_id : int
- + major_id : int
- + expected_grad_date : date

8 REST API Documentation

Each API route represents a model in the system. We have two models in the system - apartment and user.

8.1 Apartment

Route: /api/apartments/:id

Fields

Name	Data type	Example
id	integer	1
active	boolean	true
created_at	datetime	2016-10-26T19:00:00.511Z
updated_at	datetime	2016-10-26T19:00:00.511Z
address_line_1	string	820 Serano Dr.
address_line_2	string	Apt 7A
city	string	San Francisco
state	string	CA
country	string	USA
zip	string	94123
title	string	Spacious room for rent
description	string	Welcome to your new home! This updated apartment is located in the Inner Richmond...
sq_feet	float	450.5
nr_bedrooms	integer	1
nr_bathrooms	integer	1
nr_roommates	integer	2
floor	integer	7
private_room	boolean	true

Name	Data type	Example
private_bath	boolean	true
kitchen_in_apartment	boolean	true
has_security_deposit	boolean	true
credit_score_check	boolean	false
monthly_rent	float	1400
security_deposit	float	1400
pictures	array of strings	["http://crossfiremedia.realtor.com/ashtonsanfranciscogs/photos/ph1139_h800.jpg"]
available_since	datetime	2016-10-30T00:00:00.511Z
lease_end_date	datetime	2017-09-29T00:00:00.511Z
flagged	boolean	false

HTTP GET /api/apartments

Returns a JSON collection of all apartments.

Example:

```
[
  {
    "id": 1,
    "active": true,
    "created_at": "2016-10-26T19:00:00.511Z",
    ...
  }, {
    "id": 2,
    "active": true,
    "created_at": "2016-10-26T20:00:00.511Z",
    ...
  },
  ...
]
```

Response code when resource is found: **200 OK**

Response code when resource couldn't be found: **404 Not Found**

Response code in case of server error: **500 Internal Server Error**

HTTP GET /api/apartments/:id

Returns a specific apartment based on provided id.

Example:

```
{
  "id": 1,
  "active": true,
  "created_at": "2016-10-26T19:00:00.511Z",
  ...
}
```

Response code when resource is found: **200 OK**

Response code when resource couldn't be found: **404 Not Found**

Response code in case of server error: **500 Internal Server Error**

HTTP POST /api/apartments

Creates an apartment in the database, returns created resource in JSON format.

Example payload:

```
{
  "active": true,
  "created_at": "2016-10-26T19:00:00.511Z",
  ...
}
```

Notice that provided payload doesn't contain the id - the id is unknown before creation.

Example response:

```
{
  "id": 5,
  "active": true,
  "created_at": "2016-10-26T19:00:00.511Z",
  ...
}
```

Notice that response contains the id.

Response code when successfully created: **201 Created**

Response code in case of malformed request (the fields didn't validate or are missing): **400 Bad request**

Response code in case of error: **500 Internal Server Error**

HTTP PUT /api/apartments/:id

Updates an apartment in the database, returns created resource in JSON format.

Example payload:

```
{
  "id": 5,
  "active": false,
  "created_at": "2016-10-26T19:00:00.511Z",
  ...
}
```

Notice that provided payload contains the id - we are updating known object.

Example response:

```
{
  "id": 5,
  "active": true,
  "created_at": "2016-10-26T19:00:00.511Z",
  ...
}
```

Response code when successfully updated: **200 OK**

Response code in case of malformed request (the fields didn't validate): **400 Bad request**

Response code in case of error: **500 Internal Server Error**

HTTP DELETE /api/apartments/:id

Deletes a resource with a provided id, returns deleted resource in JSON format.

Example response:

```
{
  "id": 5,
  "active": true,
  "created_at": "2016-10-26T19:00:00.511Z",
  ...
}
```

Response code when successfully deleted: **200 OK**

Response code when resource couldn't be found: **404 Not Found**

Response code in case of server error: **500 Internal Server Error**

8.2 User

Route: /api/users/:id

Fields

Name	Data type	Example
id	integer	1
active	boolean	true
created_at	datetime	2016-10-26T19:00:00.511Z
updated_at	datetime	2016-10-26T19:00:00.511Z
first_name	string	Donathan
last_name	string	Drumpf
email	string	donathan.drumpf@gmail.com
role	string	regular
program	string	undergraduate
major	string	csc
expected_graduation	datetime	2017-09-29T00:00:00.511Z

HTTP GET /api/users

Returns a JSON collection of all users.

Example:

```
[
  {
    "id": 1,
    "active": true,
    "created_at": "2016-10-26T19:00:00.511Z",
    ...
  }, {
    "id": 2,
    "active": true,
    "created_at": "2016-10-26T20:00:00.511Z",
    ...
  },
  ...
]
```

]

Response code when resource is found: **200 OK**

Response code when resource couldn't be found: **404 Not Found**

Response code in case of server error: **500 Internal Server Error**

HTTP GET /api/users/:id

Returns a specific user based on provided id.

Example:

```
{
  "id": 1,
  "active": true,
  "created_at": "2016-10-26T19:00:00.511Z",
  ...
}
```

Response code when resource is found: **200 OK**

Response code when resource couldn't be found: **404 Not Found**

Response code in case of server error: **500 Internal Server Error**

HTTP POST /api/users

Creates a user in the database, returns created resource in JSON format.

Example payload:

```
{
  "active": true,
  "created_at": "2016-10-26T19:00:00.511Z",
  ...
}
```

Notice that provided payload doesn't contain the id - the id is unknown before creation.

Example response:

```
{
  "id": 5,
  "active": true,
  "created_at": "2016-10-26T19:00:00.511Z",
  ...
}
```

Notice that response contains the id.

Response code when successfully created: **201 Created**

Response code in case of malformed request (the fields didn't validate or are missing): **400 Bad request**

Response code in case of error: **500 Internal Server Error**

HTTP PUT /api/users/:id

Updates a user in the database, returns created resource in JSON format.

Example payload:

```
{
  "id": 5,
  "active": false,
  "created_at": "2016-10-26T19:00:00.511Z",
  ...
}
```

Notice that provided payload contains the id - we are updating known object.

Example response:

```
{
  "id": 5,
  "active": true,
  "created_at": "2016-10-26T19:00:00.511Z",
  ...
}
```

Response code when successfully updated: **200 OK**

Response code in case of malformed request (the fields didn't validate): **400 Bad request**

Response code in case of error: **500 Internal Server Error**

HTTP DELETE /api/users/:id

Deletes a resource with a provided id, returns deleted resource in JSON format.

Example response:

```
{
  "id": 5,
  "active": true,
  "created_at": "2016-10-26T19:00:00.511Z",
  ...
}
```

Response code when successfully deleted: **200 OK**

Response code when resource couldn't be found: **404 Not Found**

Response code in case of server error: **500 Internal Server Error**

9 Key Risks

9.1 Skill Risks:

- Global team, might have issue with communication between the local and the German team.
- Document writing. Might need a better document writing skill to make concise and clear documents

9.2 Schedule Risks:

- Due to the team dealing with unfamiliar languages/frameworks predicting how long it will take to implement different features may be challenging.

9.3 Technical Risks:

- AngularJS is a new language to the majority of the team.
- Google Maps API has not been used to the extent needed by any team member.
- The team has not done much backend programming, which may also prove to slow the project down.

9.4 Teamwork Risks:

- Working with an international team adds risk because of geographic distance, cultural differences, and other unforeseen challenges.

9.5 Legal/Content Risks:

- No foreseeable legal or content risks.

10 Team Members

Alex Aichinger	Team Lead/Front end /Back end
Cheng Li	Front end/Back end
Darrel Daquigan	Front end/Back end
Jiri Loffelmann	Chief Technical Officer/Front end /Back end