Place for Space

Software Design

**1. Introduction**

**1.1 System Description**

Provide customers with a platform to list and rent homes, apartments, and condos. This could be just for quick vacation or for your forever home. The process should be easy and convenient with effective search tools to find the right place for you.

**1.2 Design Evolution**

**1.2.1 Design Issues**

Before we started, we knew we wanted to work with Mongodb for our database, and we wanted users to have the option to sign up through Google.

**1.2.2 Candidate Design Solutions**

When we went into this project, the only design philosophy we had in mind was making sure things were organized. We knew we wanted to have the front and back ends relatively close together, which would be made easy with the tools we chose (like Node and Handlebars). We also wanted to prevent things from being to unruly, so we made sure to split the code up into different files where it was appropriate.

**1.2.3 Design Solution Rationale**

Because our design philosophy was so simple, it was not hard to handle. We simply had to make note of when files were getting out of out, as well as make sure things were placed appropriately within the file structure.

**1.3 Design Approach**

**1.3.1 Methods**

The frontend started with rough prototypes. They then added on to them, making them more fleshed out as the sprint progressed. Furthermore, rather than having the code for the header in every handlebars file, we simply use a header tag to import it into every file. This cut down on repasting of code.

For the backend, things were simpler. There wasn’t really much that need to be repeated, so we could just place the code in an appropriate file.

**1.3.2 Standards**

File structure was the most important part of our design. Essentially just keeping everything in an appropriate folder. For example, all of our layouts stay in views/layouts. We also have folders dedicated to routing, models, etc.

In JavaScript, semicolons are optional. We decided to avoid them when possible simply to keep the code looking clean. We also went with camelCase for everything except the names of the views. Because they will be displayed in the URL, we decided to separate the words with hyphens. Ex: about-selling instead of aboutSelling

**1.3.3 Tools**

We have a long list of dependencies. These include the following:

* Express is our node framework
* Mongoose helps us interact with our database
* Dotenv lets us manage our config files
* Morgan handles logging our HTTP requests
* Handlebars helps with the frontend
* Passport and Google Passport are used for authentication
* Flash is for handling our popup messages
* MongoStore helps us save our data to the database

## 2. System Architecture

### 2.1 System Design

Diagram provided at the end of the document

### 2.2 External Interfaces

Google OAuth – handles login/authentication. Data (name, icon, ID number, etc.) on the user will be gathered when they create their account. This data will be stored by us in our database using models. There will be a user model, as well as possibly others. We may have a model for a listing, for example.

Mongodb – our database solution. It will hold info including account information, listings, ratings, comments, chat logs, etc.

## 3. Component Design

The Component Design section details the proposed design of each system component.  A system conponent is a functional partition of the system.  Components may be organized as you see fit - a component may be a collection of objects, or a single object.  However, a system must be composed of multiple components (that is, a system cannot be one component).  The layout of this section is at your discretion, but please include the following (at a minimum) information for each component:

* **Component Name**  
  Layouts
* **Component Description**  
  Contain the links to the CSS as well as the JavaScript needed to make the given page work properly. Also contains sections that import HTML from the views (which will be described next), as well as smaller pieces that are reused, such as the header. Layouts require other files to work properly, and are only executed when that page is accessed.
* **Responsible Development Team Member**  
  Everyone on our team contributed to the layouts.
* **Component Diagram**  
  Relationships between all components are in package diagram at the end of this document.
* **Component User Interface**

The layouts combined with the views are the UI itself. The current layouts include a main page, login screen, registration screen, and a user dashboard. All will contain links to other pages, and some (login and registration) contains forms to be submitted. These will then be sent to the backend for processing.

* **Component Objects**

As mentioned previously, each layout contains references to the CSS, the views containing the HTML, as well as JavaScript.

The login and registration pages will send data to the backend, all of which is text. In addition, the layouts themselves are called by the backend with render(), which loads the page.

* **Component Interfaces (internal and external)**  
  As mentioned previously, the login and registration pages will be sending data to the backend via the corresponding view. This includes information including first name, last name, email, password, and password again (for verification).
* **Component Error Handling**

Given this component is all just UI, there won’t really be any error handling necessary.

 **Component Name**  
Views

 **Component Description**

This is the HTML (in the form of a handlebars file) that gets inserted into the body of the corresponding layout. Whenever a page is loaded, this is what renders on screen using the CSS linked to in the layout. Like the layouts, this only gets executed when the corresponding page is accessed.

 **Responsible Development Team Member**  
Nona and Tanner wrote the HTML. Nona, Adam, and Tanner integrated it into the backend.

 **Component Diagram**  
Relationships between all components are in package diagram at the end of this document.

 **Component User Interface**

As mentioned in the layouts, this will have objects like buttons, form, etc. that can be interacted with. Each view represents a different screen that can be rendered. When entering information into the registration form, there will be notifications if the passwords do not match, the password isn’t long enough, or if any of the fields are left blank. The login page will have something similar for if the password does not match the given email.

 **Component Objects**

They are comprised of different HTML elements, like buttons, forms, dropdown menus, and an image. Each view is put into the body of the appropriate layout. As mentioned previously, text entered into the registration/login pages will be sent to the backend.

 **Component Interfaces (internal and external)**  
All interactions with other components have been listed above.

 **Component Error Handling**  
When registering, the page will ensure a few things: the passwords match, the password is at least 8 characters long, and all fields are entered.

 **Component Name**  
Backend

 **Component Description**  
For this sprint, the backend will do a few things. First is routing. This tells the site what redirects to what, as well as which layout to load for a given page. Next is handling authorization/login. This comes in 2 flavors, either through Google or just our site. Whenever you create an account, the data gets sent to Mongodb to be stored. Just before, the password gets encrypted by passport so that no one has access to it. Lastly, the backend is what gets the site running. For example, it runs on port 3000, sets up all the dependencies, etc.

 **Responsible Development Team Member**  
Connor and Adam handled the backend

 **Component Diagram**  
Relationships between all components are in package diagram at the end of this document.

 **Component User Interface**  
The backend does not have any UI. Everything forward facing is either a view or layout.

 **Component Objects**

app.js: sets up dependencies and makes the calls necessary for the app to run, including calls to passport, flash, express, and each file in routes.

users.js: tells the app which page to render for a given URL. Also handles error checking for login/registration. For example, if someone tries to register with an email that already has an account, they will receive a popup and the account will not be created again.

index.js: handles rendering the dashboard specifically.

auth.js: This is the file that communicates with Google to handle OAuth. It has functions that login the user, handles login failures, and one for logging out.

 **Component Interfaces (internal and external)**

The backend will receive data from the front end about user login/registration. This includes name, email, and password. External interfaces were mentioned in 2.2

 **Component Error Handling**

All functions that may have an error are encapsulated in a try/catch statement, with the catch printing out the error received.

Diagram

Description automatically generatedDiagram

Description automatically generatedDIAGRAMS: