# Hodophilia

# Software Design

## CSCI-P465/565 (Software Engineering I)

## Project Team

### Akshay Murthy

### Griffin Wall

### Stephen Smith

### Venkata Indu Bhanu Pothineni

## 1. Introduction

### 1.1 System Description

Hodophilia is a web based touring management system which provides tour planning and managing solutions to users at the comfort of their homes. It seeks to provide users with all the tools they will need to select an ideal, personalized tour, book the tour, and follow the tour with minimal oversight during the trip.

### 1.2 Design Evolution

#### 1.2.1 Design Issues

Our team faces a few notable challenges: as compared to a professional team, we collectively have quite a bit less experience in web application development. Added to this is our quite narrow project completion window of time. This means completion time, and thus our ability to understand the technologies we're working with, is of the utmost importance.

Because our project requires a web application, we are required to have an appropriate system design - frontend with vanilla HTML, CSS, & JS (or an extending library/framework) and backend that can sufficiently serve web technologies through HTTP services. Following this, we also need a way of preserving user information on our own systems, thus requiring an appropriate database technology.

#### 1.2.2 Candidate Design Solutions

For our frontend, our most popular consideration was React. We also very briefly considered running a vanilla JS implementation, as well as a few other libraries such as Vue and Svelte. For file structure, we discussed the possibility of using a strict MVC design due to its predictable and strict guidelines. We also considered, for simplicity's sake and to align our structure more closely with a potential React project style, to simply form a distinction between individual components and screens.

For our backend, we most heavily considered Java Spring Boot and Python Flask, though Node.js + Express (using the MERN stack) was also a consideration. For our database, we liked the idea of a NoSQL DB - specifically MongoDB or Firebase - as we noted the object-based organization provided with JSON may be more closely aligned with the types of data we will be storing. Firebase was also alluring as it provided many abstractions for easy access, as well as a cloud storage system removing the hurdle of hosting a database server ourselves. PostgreSQL was also an appealing technology.

#### 1.2.3 Design Solution Rationale

We have decided on using React paired with Java Spring Boot as our frontend and backend. For React, this was due to its popularity, the availability of resources and assistance surrounding it, and Stephen's prior exposure to the library. We believe this option would give the best chance of providing a quality, functional product with all the required features in the shortest amount of time. For Java Spring Boot, it was due to popularity as a backend framework and that it is powered by Java (a language Indu, Griffin, and Akshay are familiar with). We went with PostgreSQL due to its easy integration with Java Spring Boot as well as due to the entire team's familiarity with SQL.

### 1.3 Design Approach

#### 1.3.1 Methods

Our frontend emphasized compartmentalization to divide code into easily reusable components.

Our backend has captured elements of the MVC architectural pattern in order to service the REST API. We use two model classes - Provider and User - to represent their respective systems (Provider: the service that provides the account login). We also use an authentication controller to accept input from POST calls and prepare a response.

#### 1.3.2 Standards

We have nested Screens inside the components folder to distinguish smaller components from an assembled page of components. We also take a heavy disposition towards a breadth-first file structure, particularly for the frontend, avoiding deeply nested directories.

We take every precaution to protect user passwords, providing encryption where possible.

Our frontend sticks to Javascript's standard for variable and function names; namely the fixed use of camel case. In addition to this, constant values are always written in uppercase.

#### 1.3.3 Tools

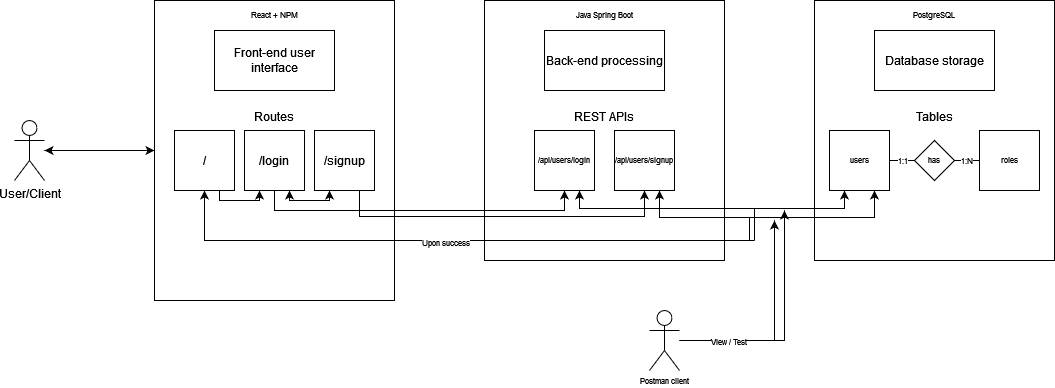
Our primary IDE is Visual Studio Code. We will use this for multiple reasons. First, its excellent git management controls ensures that we will have multiple options for quick, easy version control. Related to this, it provides easy access to multiple types of terminals. This will allow us to quickly view error logs and frontend/backend server status - and give commands to each - all in one place. We will use it alongside recommended Java extensions (notably "Extension Pack for Java") provided via Visual Studio Code's extensions marketplace. This will give us all the support we will need for running, autocompleting, and debugging our Java Spring Boot server. The primary use of Visual Studio Code will, without a doubt, be in managing the front-end due to its fantastic support for web languages. However, we will also be using it to develop our Java Spring Boot application.

We will also be using Node.js and the Node Package Manager to provide and update node modules critical for operating the front-end, such as React.

## 2. System Architecture

### 2.1 System Design

We have chosen to use React in the frontend to communicate with our Java Spring Boot server on the backend with a REST API. This API will retrieve and submit data to/from our database storage with PostgreSQL. We use an encryption algorithm to encrypt/decrypt these passwords before storing in and after retrieving from the database. We then generate a JWT and send it back to the user/client through our REST link between frontend and backend. To test these calls before assembling front-end fetch requests, we will use Postman.



### 2.2 External Interfaces

Login + Signup routes: Provides a web interface to submit a form, where inputs are converted to a JSON object and submitted via POST to two separate REST APIs. Once data has been submitted, the input will be validated. If validation fails, the user will be prompted to try again, notifying them what errors occurred. The errors will be sent back, alongside an error code, in a JSON response object.

React <Login /> route: This form has a field for **email**, **username**, and **password**. Uses the api/users/login endpoint.

React <Signup /> route: This form has a field for **first name**, **last name**, **email**, **username**, and **password**. Uses the api/users/signup endpoint.

## 3. Component Design

**Component Name**

User Management: Login/Sign-up component

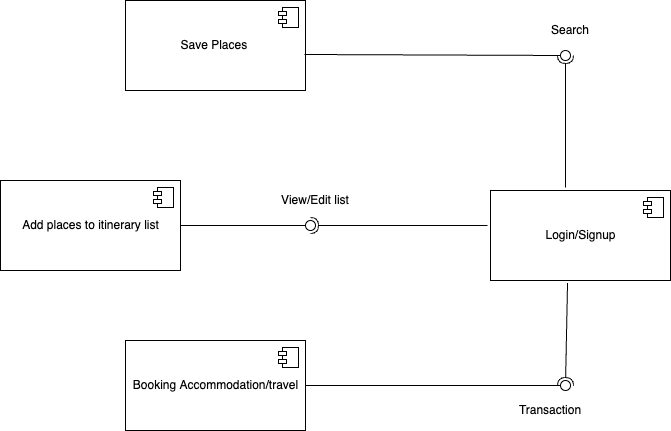
* **Component Description**

This component allows users to login to their account when the user tries to save locations to visit, make a booking or add a place to their itinerary. It provides multiple ways for the user to login to the system through their existing Google/Facebook (using OAuth 2.0) or email accounts and also provides high password encrypted techniques to allow the user to securely login to Hodophilia.

* **Responsible Development Team Member**

Akshay Murthy takes the responsibility in both design and implementation of this component.

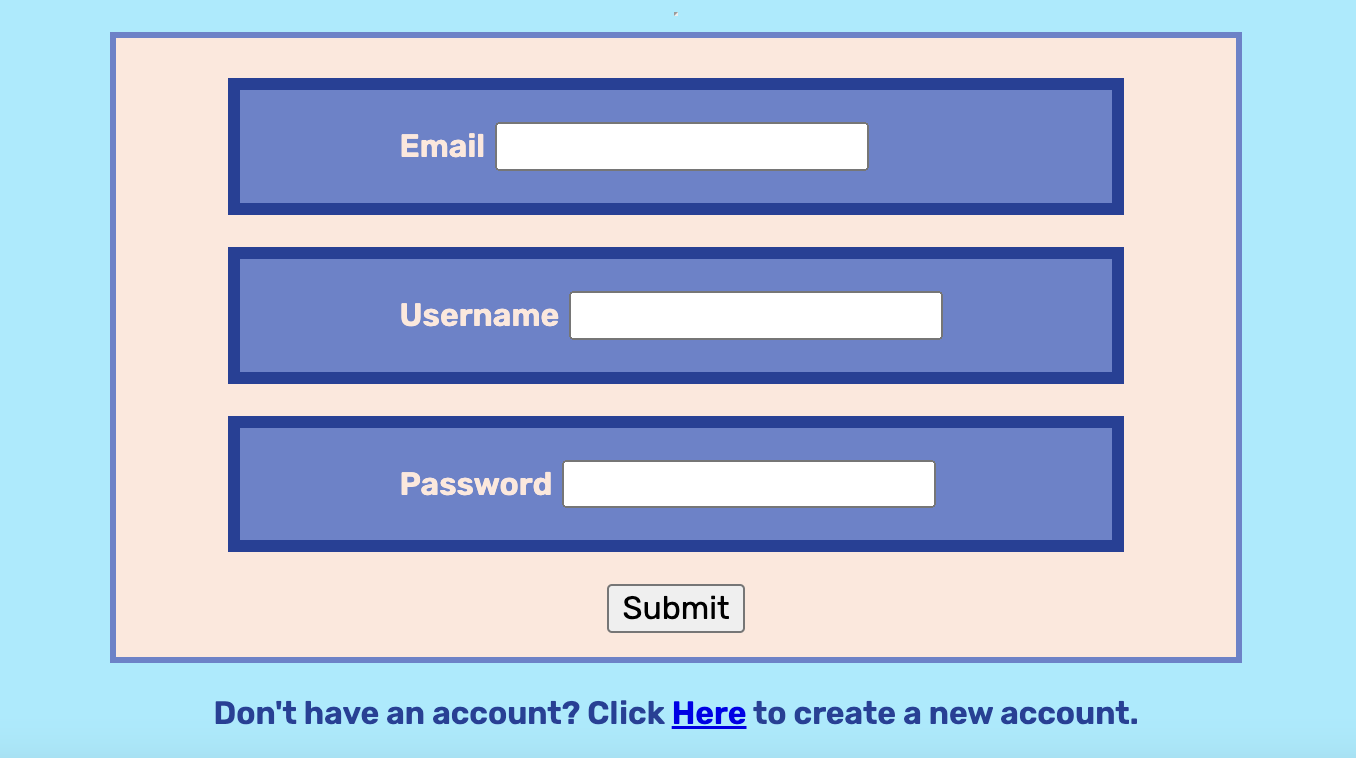
* **Component Diagram**



* **Component User Interface**

The UI of this component includes the login and sign-up pages where the login page allows users to provide Email/Username along with their login password which is further encrypted using Bcrypt hashing method for a safe access and the sign-up page takes the basic inputs from the user and registers an account while storing the user details in the PostgreSQL database.

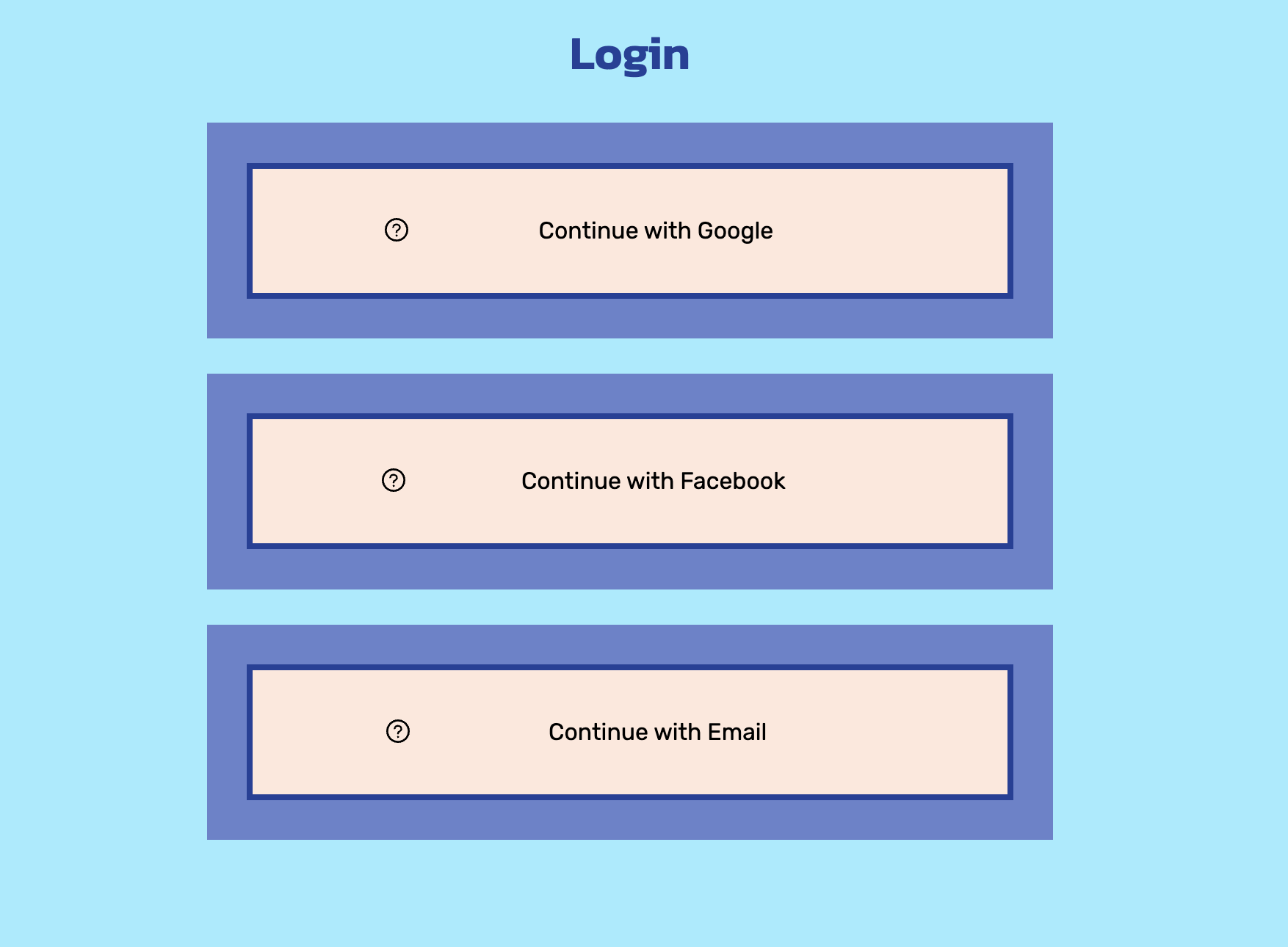
LOGIN PAGE:



SIGN-UP PAGE:



LOGIN WITH GOOGLE/FACEBOOK:



* **Component Objects**

o The signup page allows the signup API to validate the user information and stores the information under ‘users’ table in PostgreSQL database.

o Login API allows user authentication where the login page provides details required to authenticate the user to the login API and then it validates the user details from database and provides an access token which informs the API that the user is authorized to use the service.

o Both login and signup APIs take user details in json format and throws a status code and response message depending upon the test case.

o Google/Facebook's external API services are used to provide more flexibility to the user to access these services by simply logging in through Google/Facebook if the user already has an existing account in it.

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

Internal interface includes developing Rest APIs for login/signup page and validating user details in backend together with the input form validation of the user to send a valid request. Also, JSON format is used to exchange information in the web application using Rest APIs.

Refer to Section 2.2 for external interfaces.

* **Component Error Handling**

**Error Case 1:** Input Validity Check:

For input validity, we are using Spring boot’s built-in annotations-based methods to check for input errors. We are using @NotBlank annotation, @Size to check for blank user inputs and large size input check. We are using JpaRepository’s existsByUsername to check if the user already exists. JSON message is being sent back with appropriate error status code with a message.

**Error Case 2:** User Authentication Check: User can login through third party providers with the help of OAuth. Spring boot Authentication manager is used to authenticate with username and password, which returns a access token to be stored for a particular session started by the user.

## Revision History

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **Change Description** |
| V1 | Oct 4, 2022 | Post Sprint 1 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |