Host Your House

* TOC needs to be finished, sections need to be added, or are present under different headings, tables need numbering, contributions need to be added

# Table Of Contents:

# **Overview**

# **Project Plan**

# **Requirements Specification**

# **System Specification**

# **User's Guide**

# **Test Plan and Results**

# **Design and Alternate designs**

# **Development History**

# **Conclusions**

### 

# 

# **Overview**

* Consumers are frequently plagued by hotel prices that are exorbitant and frequently have fees that are designed to keep the customer from canceling or changing the time and date of the rental. On top of that, hotels are not always located in rural areas that don’t get a lot of traffic or that aren’t in major cities. To alleviate that, Airbnb was created to allow users to rent out their houses if they met certain criteria. This would afford frequent travelers the ability to stay in places that felt like home and weren’t directly in the city where most hotels were located. The problem is that there aren’t many Airbnb competitors, which means stagnation in the industry and a monopoly on the housing market. To combat that, we developed Host a House, a web-app that would allow users to host their own house to rent, giving another option in the house rental space.
* Host Your House is a Full stack Web application that uses React for the frontend, Node.js for the backend, and MongoDB for the database. frameworks/technologies like Express, Mongoose, Bootstrap, to facilitate development and design.

## 1.1 Individual Contributions Summary

* 1. Khalil Savory:
  2. Caleb Elkins:
  3. Geraldo Moura:
  4. Troy Raines:
  5. Thomas Lindley:
  6. Paul Ambrester:

# **Project Plan**

## 2.1 Project Overview

### 2.1.1 Purpose, Objectives, and Scope

#### 2.1.1.1 Project Purpose & Objectives

* Consumers are frequently plagued by hotel prices that are exorbitant and frequently have fees that are designed to keep the customer from canceling or changing the time and date of the rental. On top of that, hotels are not always located in rural areas that don’t get a lot of traffic or that aren’t in major cities. To alleviate that, Airbnb was created to allow users to rent out their houses if they met certain criteria. This would afford frequent travelers the ability to stay in places that felt like home and weren’t directly in the city where most hotels were located. The problem is that there aren’t many Airbnb competitors, which means stagnation in the industry and a monopoly on the housing market. To combat that, we plan to develop *Host a House,* a web-app that would allow users to host their own house to rent, giving another option in the house rental space.

#### 2.1.1.2 Project Scope

* This effort is determined to develop a high-quality web-app with a modern and responsive front-end and a robust back-end API that supports filtering and querying of properties and users. The application should allow users to create accounts and rent a property for a specified time frame. The application will also allow users to cancel or change the date of the request and will have an account page in which current reservations can be seen.

#### 2.1.1.3 Out of Scope

* Out of scope for this effort is the ability to send confirmation notices via email or SMS messages and support for a fully functioning payment system is also out of scope. The ability to send the owner of the property a message within the app was also determined to be out of scope at this time but can be added in if time permits. Finally, we have determined that due to time constraints and technological limitations that the ability to post a property for renting will also only be able to be completed if time allows.

### 2.1.2 Assumptions and Constraints

#### 2.1.2.1 Assumptions

* Currently, there are no assumptions as there are no resources outside of personnel being provided to the team in the form of money, hardware, or software. The team at this moment in time also does not have any assumptions due to a lack of prior knowledge and frameworks.

#### 2.1.2.2 Constraints

* Current constraints on the project are a shortened time frame of only a month and a half and a limitation on programmatic tools to open-source software and free programmatic tools.

### 2.1.3 Project Deliverables

* The project deliverables for this project have been determined to be a fully functional web-app. Project files will be hosted in an accessible Github repository for easy downloading, and the MongoDB database will be based on their ATLAS platform which can be referenced until one week after the end of the course.

### 2.1.4 Schedule Summary

#### 2.1.4.1 Schedule Summary

* As of the time this report was written, there are 6 weeks to develop the application and polish and deploy deliverables for the project. From March 30 to April 5, the development team will focus on user creation and the login and authorization module. This will include back-end work such as creating user models, setting up the database, and defining user retrieval methods. It will also include front-end work in the form of developing a log-in screen and confirmation of successful login. From April 6 to April 19, the team will be focused on developing the rental listing module. This module will allow users to search for, filter, and look at rental properties. Most of the work for this module will be in developing the front-end pages for displaying the list of properties and showing more detailed information but work must also be done in the back end to retrieve property details and images. From April 20 to April 26, we will be developing the ability to create and review reservations and save potential properties which will require equal amounts of front-end and back-end work. From April 27 to May 3, we will be working on the rating system and coming up with a weekly featured page that shows the hottest properties of the week. This will also be the time when we accomplish any stretch goals for the project. Finally, from May 4 to May 10, we will work on polishing the application and quality assurance testing in order to hand over deliverables at the end of the timeline.

## 2.2 Project Organization

### 2.2.1 Roles and Responsibilities

* The team will consist of 3 different roles: Project Manager, Project Owner, and Software developers. The project manager role has been filled by Troy Raines and is responsible for write-ups of documentation and ensuring the team is adhering to the timeline that has been presented in earlier sections. The product owner is Khalil Savoy and has the responsibilities of managing the task list and assisting in any task that has fallen behind or where assistance is required. Caleb Elkins, Geraldo Moura De Oliveira, Thomas Lindley, Paul Armbrester makes up our list of software developers and their responsibilities lie solely in developing the application.

## 2.3 Managerial Process Plans

### 2.3.1 Start-up Plan

#### 2.3.1.1 Estimation Plan

* As only open-source software and resources are utilized the cost will be nil, the projected start date is March 16th, 2022, with a target end date of May 10, 2022.

#### 2.3.1.2 Staffing Plan

* This project shall have six members that contribute to it. The current roles assigned, and the number of people assigned to each role are as follows: One project manager, One Project owner, and 4 software developers with all members of the team doing testing and quality assurance.
* Each Member of the group is also expected to review documents when available to ensure the documents have correct grammar, spelling, and the documents design and flow is acceptable.

### 2.3.2 Work Plan

#### 2.3.2.1 Work Activities

* Work activities will consist of daily software development in either the back end or the front-end of the application. Features of the project have been broken down into smaller actionable items that can be worked on in parallel by each member of the team. Tasks relating to infrastructure or system architecture will be relegated to the product owner and if needed, a software developer.

#### 2.3.2.2 Schedule Allocation

* There are currently no constraints when it comes to scheduling but it can be assumed that each member of the team is able to contribute 1-2 hours of development time a day on average. Tasks will be broken into week-long sprints, twice a week, once on Wednesday and then on Friday the team can discuss what went wrong, what went well, and how we can improve in order to deliver a high-quality product.

### 2.3.3 Control Plan

#### 2.3.3.1 Requirements Control Plan

* Product requirements will be tracked in Trello for visibility across the entire team and in order to avoid overlap in tasks assigned. We have also created a discord server in order to communicate changes in deadlines, expected delivery dates, and roadblocks that may appear.

#### 2.3.3.2 Schedule Control Plan

* Utilization of the Trello board will be key in keeping members of the team focused and accountable for missed deadlines and milestones. The Trello board provides statistics for the current amount of development time spent on a ticket and can be configured to warn a user if the ticket has been sitting for too long. The team has also constructed and committed to a plan that involves 2 weekly check-ins in order to discuss progress, roadblocks, and potentially missed deadlines. In the case that a task must be dropped, or a developer needs assistance, it is the responsibility of the product owner to step in where applicable.

#### 2.3.3.3 Quality Control Plan

* In order to assure quality deliverables, the team will be utilizing GitHub for branch management and testing. Users must have their branch reviewed, tested, and approved by another member of the team before the code can be added to the master branch. This ensures that no breaking changes are added to the application which could delay timelines.

## 2.4 Risk Management Plan

* Risks such as member outages, missed deadlines, and roadblocks happen but our goal is to mitigate them as much as possible. Our method of implementation to mitigate these risks is by investing in great communication. Having two weekly check-ins enables the team to communicate when things are not going as planned, work is falling behind, or when any number of risk factors pop up.

## 2.5 Technical Process Plans

### 2.5.1 Process Model

* Our process model is one based on the agile framework that has been popularized in recent times. Each ticket contributes to a larger feature that makes up one part of the application but is essentially independent of each other in that they can be worked on by different developers without confrontation happening. Information regarding the project will be communicated in the discord channel that has been set up previously and timelines and major milestones have been determined and communicated prior to this document being written with our Trello ticketing system also relaying that information to developers.

### 2.5.2 Methods, Tools and Techniques

* Project files will be hosted in an accessible Github repository for easy downloading, and the MongoDB database will be based on their ATLAS platform which can be referenced until one week after the end of the course. This project will use the MERN stack, which is MongoDB for database, Express JS as framework for backend server, React JS as the frontend framework, Node JS backend server.
* Hardware required for development is a computer with at least 8GB of ram, at least 4 cores CPU, and internet access. The machine will need to have NodeJS version 16 or later, NPM version 6, and an IDE. Recommended IDE is Visual Studio Code with the following extensions: ESLint, GitHub Pull Request and Issues, Path Intellisense, Prettier - Code formatter.

### 2.5.3 Infrastructure Plan

* Hardware required for development is a computer with at least 8GB of ram, at least 4 cores CPU, and internet access. The machine will need to have NodeJS version 16 or later, NPM version 6, and an IDE. Recommended IDE is Visual Studio Code with the following extensions: ESLint, GitHub Pull Request and Issues, Path Intellisense, Prettier - Code formatter.

## 2.6 Supporting Process Plans

### 2.6.1 Documentation Plan

* Deliverables will be compiled previously and will be delivered in the form of a compressed file containing all project source files and a README which will instruct the user on how to utilize the system. An alternative server that is already up and running will also be delivered so that the user does not have to set up the project themselves. Finally, documentation on how to use the system will be included along with the source files and on a separate web page linked to our running server.

### 2.6.2 Quality Assurance Plan

* There are two ways that the team will utilize in order to assure quality control and assurance. The first part of that plan is code reviews before a ticket or feature is added to the official project. This ensures that no breaking changes are ever introduced into the code and will also inspire communication between team members. The second method is our two-weekly check-ins. This allows each team member to relay their status and blockers before it becomes too late to fix them. These two preventative measures will ensure that problems and bugs are handled swiftly and don’t interrupt our current timeline.

### 2.6.3 Problem Resolution Plan

* In order to reduce and track problems that arise we will be utilizing the review and comment features that GitHub has. By noting problems in the code on Merge requests, we are able to keep track of problems that arise, and we can refer back to them if need be. We will also be using the comment feature on the Trello tickets in order to track changes to criteria or feature development for any reason. This also helps us resolve and refer to issues that crop up in the future.

# **Requirements Specification**

# **System Specification**

# **User's Guide**

## 5.1 Configuring the Application - For Local Users

### 5.1.2 System Requirements

* Node version 16.14.2
* NPM version 8.6.0
* MongoDB atlas cloud database deployed on an AWS M0 Cluster
* React version 18.0.0

### 5.1.3 Running the Application

1. Open a terminal
2. Clone the repo from the following URL: [Host A House Repo](https://github.com/CMSC495HostYourHouse/host-your-house-app.git)
3. In the backend directory of the project, add the desired port and the connection URL of the mongoDB database to the config.env file
4. In the root directory of the project run npm run dev in order to start both the backend server and the frontend client

## 5.2 Landing Page

* The Host A House landing page is what all users will be redirected to if they are not signed in or do not put in a specific path in the URL bar. When no information is entered, the landing page displays the featured list of houses which is a list of the top-rated properties available for booking at that current moment. However, users are also able to sort the list by date, location, rating, & price. If a timeframe has been entered by a user, the application will not show properties that have already been booked for the timeframe entered. When users click on any of the properties, more information on the property is shown, including but not limited to more pictures of the property. In order to reserve a property users must sign in using the sign-in button found on the right-hand side of the navigation bar.

## 5.3 Login and Registering

* If a user is not signed in, hitting the sign-in button to the right of the navigation bar will take them to the login page. The login page will then ask for the user's email and password but will also have a button that will take users to the Sign-Up page if they have not created an account.
* In the Sign-Up page the user will be able to create an account by inputting a valid email, password, and confirming the previously entered password. By clicking the register button, a new user will be created if the information is verified to be correct, and the email is not a duplicate. If the user wants to cancel signing up or wants to return to the login page, buttons for those actions will also be available.

## 5.4 Reserving Properties

* Properties can be reserved for a specified timeframe by logging into the application and clicking on a property on the landing page. Once more information about that property has been shown, a reserve button can be clicked for the user to reserve the house. If a user has not entered a timeframe on the landing page, the reserve button will prompt a user for the dates they want to reserve, at which point the system will let them know whether the property is available for booking or not. If the user has entered a timeframe on the landing page, the timeframe will be applied to the booking automatically and the property will be booked.

## 5.5 User Management

* In the top right corner of our navigation bar, you will find an account button that allows users to go to the user management page or sign out. If a user is not signed into their account, it will take them to the signup page where the user can apply for an account. If the user management button is clicked and a user is already signed in, users are taken to the user management page where they can change the details of their account and reference reservations. Account details that can be changed include their address, their name, and their password. In the future we also have plans to add a payment section where users can store payment information for quick checkout.

## 5.6 Posting Properties - Coming Soon

* There is currently no ability to post properties in the application but if time permits, we will be adding this capability and adding a new section here.

# **Test Plan and Results**

## 6.1 Test Plan Identifier

* Host A House Initial Test Plan
* App version: 1.00
  + Version Date: 02APR2022
* Test Plan Version: 1.04
  + Version Date: 04APR2022

## 6.2 Revision History

* v1.00 - (01APR2022) initial document
* v1.01 - (02APR2022) added test cases for featured rentals, rentals list, and overall program test
* v1.02 - (03APR2022)
  + added more test cases to featured rentals and rental list
  + cleaned up duplicates for easier use, will need to test some test items more than once to ensure proper functionality (like logged in user save more than one property, different properties, etc.) and can be added with a indented letter on table to show it is a sub test
  + added tests for all main features
  + filled out expect outputs for all tests
  + filled out approach, deleted test items, not sure if needed
  + unsure of other sections if needed or not
* v1.03 - (04APR2022) added more to introduction, filled out deliverables, and deleted sections not used
* v1.04 - (05APR2022) Added user guide section from discussion post, Paul corrected spelling issues, Geraldo wrote user guide.

## 6.3 Introduction

* **Purpose:** Create an application users can use to rent houses in various locations.
* **Description:** Application uses MERN stack to create a fully functioning website.
* **Goals:** The goal of this project is to create an online, easy to use, and modern looking rental application that uses quality software and quality UI design to deliver an enjoyable experience to customers

## 6.4 Features to Be Tested

* Rental properties
  + list of rental properties
  + search rentals
    - by location
    - by date
    - by price
  + sort rentals
    - by location
    - by date
    - by price
  + featured rentals
* Saved/Reservation management
  + logged in user make reservation
  + logged in user save properties
  + logged in user view save/reserved properties
* User management
  + create user
  + log in/logout

## 6.5 Main features test

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Overall Test Case | Requirement | Expected Output | Actual Output | Pass/Fail |
|  | Rental Properties (1) | Shown on the main page, featured at top, all at bottom. |  |  |
|  | Featured rentals (1a) | Shows featured properties at top of page |  |  |
|  | list of rental properties (1b) | List of all properties below featured, with search and sort |  |  |
|  | Search rentals (1b) | Allow searching with price, dates and location |  |  |
|  | Sort rentals (1b) | Allow sorting by date, location, and price |  |  |
|  | Saved Property and Reservation Management (2) | Page for logged in users, shows saved properties and reservations |  |  |
|  | Make reservation (2a) | Clicking reserve on a rental property takes user to page with more details and dates |  |  |
|  | Save property (2b) | Clicking save on a rental property adds property to saved properties |  |  |
|  | User Management (3) | Top right of page, allows user account management |  |  |
|  | Login (3a) | Top right of page, allow users to login |  |  |
|  | Logout (3b) | Top right of page, allow users to logout |  |  |
|  | Create user (3c) | Create user, require certain things for username and password |  |  |
|  | Change Password (3d) | Allow users to change password |  |  |
|  | Save and Reserve Properties (3e) | Allow users to save and reserve properties. After doing so, show in users Saved/Reserved Page |  |  |

## 6.6 Approach

* Testing frontend features for correct operations will bring to light bugs in the front end and the backend. The above main features to be tested are an overview of where the project is at and should not be marked pass until detailed testing below is done for the feature

## 6.7 Item Pass/Fail Criteria

### 6.7.1 Rental properties

#### 6.7.1.1 Featured Rentals

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case 1a. | Input | Expected Output | Actual Output | Pass/Fail |
|  | Loading web page, featured properties at top | Web page loaded with featured properties at top |  |  |
|  | Featured properties at top scroll to different ones after time | After set amount of time, featured properties changes to show more properties |  |  |
|  | User clicks button to scroll left on featured properties to see more | Clicking button to move left on featured properties shows more properties that are featured |  |  |
|  | User clicks button to scroll right on featured properties to see more | Clicking button to move right on featured properties shows more properties that are featured |  |  |
|  | Scrolling right to end of featured loops back around to beginning of featured | Featured properties loops when reaching end |  |  |
|  | scrolling left to end of featured loops back around to beginning of featured | Featured properties loops when reaching end |  |  |
|  | User logged in, clicking property takes user to page with property details | Logged in user clicking a property takes them to page with more details on property |  |  |
|  | User not logged in, clicking property takes user to page with property details | Not logged in user clicking a property takes them to page with more details on property |  |  |
|  | Clicking reserve here takes user to page with more details, not logged in | Not logged in user clicking reserve takes them to details page |  |  |
|  | Clicking reserve here takes user to page with more details, logged in | Logged in user clicking reserve takes them to details page |  |  |
|  | User logged in, save property in featured | Logged in user clicking save property shows confirmation property was added to users saved properties |  |  |
|  | User not logged in, save property in featured | not logged in user clicking save receives prompt to log in in order to save properties |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

#### 6.7.1.2 All Rentals, Search, And Sort

* list of rental properties tests(subject to change depending on design, I am just imagining that at the top of the page are the featured locations, and then below that the list of all properties. maybe the search will be here too.)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case 1b. | Input | Expected Output | Actual Output | Pass/Fail |
|  | Scrolling down to list of all properties | When the user scrolls down to where the list of all properties is shown, the list is populated and there are buttons to sort and search the list. |  |  |
|  | Clicking sort by date ascending | Clicking sort by date ascending button shows properties with earliest availability dates first |  |  |
|  | Clicking sort by date descending | Clicking sort by date descending button shows properties with latest availability dates first |  |  |
|  | Clicking sort by location ascending | Clicking sort by location ascending button shows properties locations in ascending order based on location city |  |  |
|  | Clicking sort by location descending | Clicking sort by location descending button shows properties locations in descending order based on location city |  |  |
|  | Clicking sort by price ascending | Clicking sort by price ascending button shows cheapest by nightly rate first |  |  |
|  | Clicking sort by price descending | Clicking sort by price descending button shows most expensive by nightly rate first |  |  |
|  | Search for existing property by location | Searching for an existing property based on location shows properties in that location, i.e. Las Vegas |  |  |
|  | Search for existing property by date | Searching for an existing property based on date shows properties that have availability for that day |  |  |
|  | Search for existing property by price | Searching for an existing property based on price shows properties that are the price or cheaper. |  |  |
|  | Search for non-existing property by location | Searching for non-existing property prompts user there are no properties that match their search |  |  |
|  | Search for non-existing property by date | Searching for non-existing property prompts user there are no properties that match their search |  |  |
|  | Search for non-existing property by price | Searching for non-existing property prompts user there are no properties that match their search |  |  |
|  | User logged in, save property in list that initially shows when loading page | Saving property prompts user property was saved and is viewable on saved/reserved page |  |  |
|  | User logged in, save property in list after list is sorted by date | Saving property prompts user property was saved and is viewable on saved/reserved page |  |  |
|  | User logged in, save property in list after list is sorted by price | Saving property prompts user property was saved and is viewable on saved/reserved page |  |  |
|  | User logged in, save property in list after list is sorted by location | Saving property prompts user property was saved and is viewable on saved/reserved page |  |  |
|  | User logged in, save property after searching for it by location | Saving property prompts user property was saved and is viewable on saved/reserved page |  |  |
|  | User logged in, save property after searching for it by price | Saving property prompts user property was saved and is viewable on saved/reserved page |  |  |
|  | User logged in, save property after searching for it by date | Saving property prompts user property was saved and is viewable on saved/reserved page |  |  |
|  | User not logged in, save property in list that initially shows when loading page | Trying to save a property when not logged in prompts user to log in |  |  |
|  | User not logged in, save property in list after list is sorted by price | Trying to save a property when not logged in prompts user to log in |  |  |
|  | User not logged in, save property in list after list is sorted by date | Trying to save a property when not logged in prompts user to log in |  |  |
|  | User not logged in, save property in list after list is sorted by location | Trying to save a property when not logged in prompts user to log in |  |  |
|  | User not logged in, save property after searching for it by price | Trying to save a property when not logged in prompts user to log in |  |  |
|  | User not logged in, save property after searching for it by date | Trying to save a property when not logged in prompts user to log in |  |  |
|  | User not logged in, save property after searching for it by location | Trying to save a property when not logged in prompts user to log in |  |  |
|  | User logged in, clicking reserve on initial list, takes user to page with more details | Clicking reserve takes user to page with more details on the property like availability dates |  |  |
|  | User logged in, clicking reserve on list sorted by price, takes user to page with more details | Clicking reserve takes user to page with more details on the property like availability dates |  |  |
|  | User logged in, clicking reserve on list sorted by date, takes user to page with more details | Clicking reserve takes user to page with more details on the property like availability dates |  |  |
|  | User logged in, clicking reserve on list sorted by location, takes user to page with more details | Clicking reserve takes user to page with more details on the property like availability dates |  |  |
|  | User not logged in, clicking reserve on initial list, takes user to page with more details | Clicking reserve takes user to page with more details on the property like availability dates |  |  |
|  | User not logged in, clicking reserve on list sorted by price, takes user to page with more details | Clicking reserve takes user to page with more details on the property like availability dates |  |  |
|  | User not logged in, clicking reserve on list sorted by date, takes user to page with more details | Clicking reserve takes user to page with more details on the property like availability dates |  |  |
|  | User not logged in, clicking reserve on list sorted by location, takes user to page with more details | Clicking reserve takes user to page with more details on the property like availability dates |  |  |

### 6.7.2 Save and Reservation

* make reservation test (I am imagining that users can click reserve on the main page from featured/list rental properties, and that takes users to a page that shows more details about the rental property and dates available, that has another reservation button that allows the user to select dates and complete the reservation)

#### 6.7.2.1 Reservation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case 2a. | Input | Expected Output | Actual Output | Pass/Fail |
|  | User not logged in tries to make reservation on page with more details for property | User prompted to login or create account |  |  |
|  | User logged in tries to make reservation | Form should open allowing user to enter dates for reservation |  |  |
|  | User logged in tries to make reservation with dates outside properties availability dates | User prompted entered reservation dates are not available for the property |  |  |
|  | User logged in tries to make reservation with dates inside properties availability dates | Confirmation of successful reservation, reservation added to users saved/reserved page |  |  |
|  | User logged in tries to make reservation with invalid dates (like the 35th of April) | User prompted that dates entered are invalid |  |  |
|  | User logged in tries to make reservation with dates inside properties availability dates | Properties availability dates are updated for other users |  |  |
|  | User logged in tries to make reservation with dates that conflict with other user’s reservation dates | User is prompted dates are not available for the property as they have already been reserved |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

#### 6.7.2.2 Save

* logged in user save properties (already tested if user can save properties, just not on the details page for a property

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case 2b. | Input | Expected Output | Actual Output | Pass/Fail |
|  | logged in user saves property from detailed page of property | User prompted save was successful |  |  |
|  | not logged in user saves property from detailed page of property | User prompted to log in |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

### 6.7.3 User Management

#### 6.7.3.1 Login

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case 3a. | Input | Expected Output | Actual Output | Pass/Fail |
|  | User logs in with valid username and password | User logged in, shown at top of page |  |  |
|  | User logs in with valid username and password that has capital letters | User logged in, shown at top of page |  |  |
|  | User logs in with valid username and password uses capital letters but user does not enter properly (FreeBird12 vs freebird12) | User not logged in, prompted password incorrect |  |  |
|  | User logs in with invalid password | User not logged in, prompted password incorrect |  |  |
|  | User tries to log in without an account | User prompted incorrect account info |  |  |
|  | User leaves username and password field blank and tries to log in | User prompted to enter username and password |  |  |
|  | User tries to log in by using username in URL without entering password | User not logged in |  |  |
|  | Username field left blank, try to log in with a valid password to an account | User prompted to enter username, not logged in |  |  |
|  | Password field left blank, try to log in without password | User prompted to enter password, not logged in |  |  |
|  | User tries to go to member only page while not logged in | User prompted to log in to view saved/reserved properties |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

#### 6.7.3.2 Logout

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case 3b. | Input | Expected Output | Actual Output | Pass/Fail |
|  | User logs out after logging in | User is logged out and shown not logged in at top of page, cannot save/reserve properties, and cannot view save/reserved properties |  |  |
|  | User logs out and tries to go to member only page | User prompted to log in to view page |  |  |
|  | User logs out and refreshes page | User remains logged out, cannot save/reserve properties, and cannot view save/reserved properties |  |  |
|  | User logs out and tries to save a property | User prompted to log in |  |  |
|  | User logs out and tries to reserve a property | User prompted to log in |  |  |
|  | User logs out and then logs back in with valid account info | User can save/reserve properties and view them |  |  |
|  | User logs out and then logs back in with invalid account info | User prompted account info incorrect |  |  |
|  | User logs out then immediately logs back in and saved a property | Property is saved and viewable |  |  |
|  | User logs out then immediately logs back in and reserves a property | Property is reserved and viewable |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

#### 6.7.3.3 Register

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case 3c. | Input | Expected Output | Actual Output | Pass/Fail |
|  | Username too short | Prompt username too short |  |  |
|  | Username too long | Prompt username too long |  |  |
|  | Username contains invalid characters | Prompt username contains invalid characters and list valid characters |  |  |
|  | Username contains invisible characters | Prompt username contains invalid characters and list valid characters |  |  |
|  | Password too short | Prompt password too short |  |  |
|  | Password too long | prompt password too long |  |  |
|  | Password does not contain lowercase letters | Prompt password missing requirement |  |  |
|  | Password does not contain uppercase letters | Prompt password missing requirement |  |  |
|  | Password does not contain numbers | Prompt password missing requirement |  |  |
|  | Password contains invalid characters | Prompt password contains invalid characters |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

#### 6.7.3.4 Change Password

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case 3d. | Input | Expected Output | Actual Output | Pass/Fail |
|  | user not logged in tries to reset password | Prompt user must be logged in to reset a password |  |  |
|  | logged in user resets password with a valid password, then logs out and back in with new password | New password allows user to log in |  |  |
|  | logged in user resets password with a valid password, then logs out and back in with old password | Old password does not allow user to log in |  |  |
|  | Logged in user tries to reset password with an invalid password, then logs out and back in | User prompted new password is invalid |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

#### 6.7.3.5 Logged In user Save/Reserve Properties

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case 3e. | Input | Expected Output | Actual Output | Pass/Fail |
|  | Logged in user saved property from featured, views saved properties | Property is listed in saved/reserved page |  |  |
|  | Logged in user saved property from initial list of all properties, views saved properties | Property is listed in saved/reserved page |  |  |
|  | Logged in user saved property from list of all properties sorted by date, views saved properties | Property is listed in saved/reserved page |  |  |
|  | Logged in user saved property from list of all properties sorted by price, views saved properties | Property is listed in saved/reserved page |  |  |
|  | Logged in user saved property from list of all properties sorted by location, views saved properties | Property is listed in saved/reserved page |  |  |
|  | Logged in user made valid reservation, views reservations | Property is listed in saved/reserved page |  |  |
|  | Logged in user made invalid reservation, views reservations | Property is listed in saved/reserved page |  |  |
|  | Not logged in user tries to save property and then view it in saved properties | User prompted to log in to save, and does not have access to saved/reserved page |  |  |
|  | Not logged in user tries to reserve property and then view it in reserved properties | User prompted to log in to reserve, and does not have access to saved/reserved page |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## 6.8 Test Deliverables

* Test deliverables should include this document with actual outputs and pass/fail information. Screenshots of tests passing should be provided in folders named after features tested, and files should follow test case naming scheme i.e. for changing password test directly above 10.A 10.B etc.
* Deliverables for failed tests should be inserted below the above test tables. These should state which test it is referring to, include an image of failure, if possible, detailed description of why it failed and possible solutions.

# **Design and Alternate designs**

## 7.1 Introduction

* Title: Host A House

### 7.1.2 Purpose

* This document’s intended purpose is to give more context surrounding the design and implementation of the “Host-A-House” project. The intended audience for this document is our software developers and the stakeholders of the product, which in this case would be the entire team and our professor respectively.

### 7.1.3 Scope

* The scope for the Host-A-House platform includes several core user functionalities regarding property management. Those core functionalities are property searching, property filtering, property reservation, and user management and each core functionality can be broken down into smaller, more manageable tasks. Each core functionality will have a frontend component and a backend component to support visualizations and personalized user feedback. At the end of the development cycle, we are hoping to present a fully fleshed-out property management system that allows users to search and filter existing properties against a wide array of criteria and then reserve these properties for vacations or other ventures.

### 7.1.4 Overview

* This document is meant to provide technical information on the project and is broken down into several sections to communicate this information effectively. The System Overview section discusses the project at a high level and communicates everything in the following sections succinctly and without context. The System Architecture section follows that and explains how we’ve structured the project, whether that be file structure, or the architecture patterns used. After, we have the Data Design section discussing how we plan to store and represent the information we need and the Human Interaction Design section in which we discuss the planned user experience and user interface in the form of visual mockups. Finally, we end with a Requirements Matrix and an appendix for any links or citations that we might have.

### 7.1.5 Reference Material

* <https://www.mongodb.com/languages/mern-stack-tutorial>
* <https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013>

## 7.2 Design Considerations

## 7.3 Assumptions and Dependencies

## 7.4 General Constraints

## 7.5 Goals and Guidelines

## 7.6 Development Methods

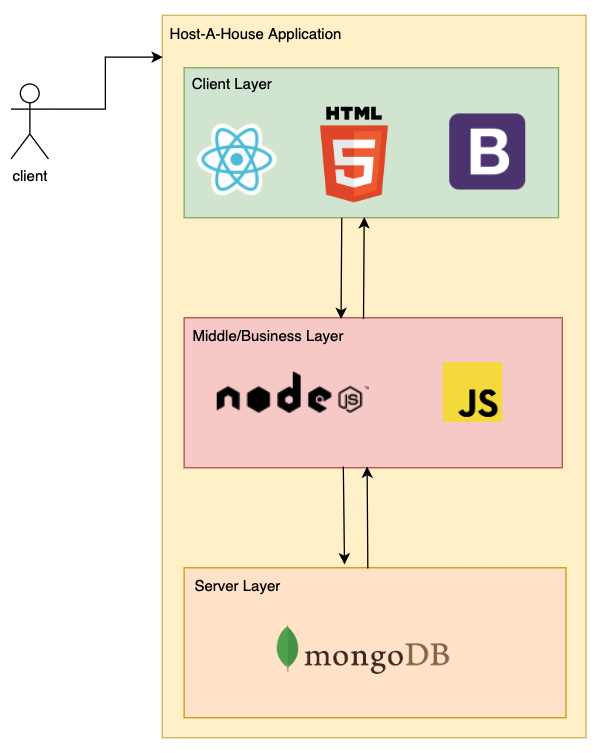
## 7.7 System Overview

* The Host-A-House application uses a layered architectural pattern in order to decompose the application into smaller parts that can be worked on in parallel. The first layer is the UI layer which we built in React and allows us to serve the users dynamic information based on information that they input into the system. We have several views that can appear based on core functionality and features that were presented in prior requirements documentation. Most of the business logic is also baked into the UI layer, such is the nature of the React framework. How users interact and use the system is truly all dictated by the visual components that are rendered on screen. Our second layer is there to provide middle ware for connecting the backend and the front-end. This layer contains logic for how the client part of the application is structured, the different web pages that exist, user authentication, and the endpoints responsible for retrieving certain information. By having an additional layer, we can change certain logic such as the way data is manipulated or global variables with minimal changes to both the UI layer and the data persistence layer. For our middle layer, we are using a combination of express and pure Javascript to manage both our UI and our data persistence layer. Finally, our data persistence layer is responsible for managing the database and making sure we can connect to it. We have config files that are used in order to properly communicate to the database and a cloud MongoDB server for data to be persisted across the application.

## 7.8 SYSTEM ARCHITECTURE

### 7.8.1 Architectural Design

* Our program structure uses a multi-layered architecture in order to make developing the system more efficient and to also abstract parts of the application away from each other in such a way that we can work on multiple parts of the application at the same time. The client layer consists of the react layer, pure HTML5 for templating, and bootstrap to keep a consistent design across the application. Our middle or business logic layer consists of the node and express frameworks. Express allows us to set up a server that holds our connections to our backend layer and node allows us to dynamically build and load in libraries needed to build the application. Our last layer is our backend layer which consists of a cloud version of MongoDB for persistence across the properties and our users and some pure javascript for manipulating any data before it is loaded into the application.



### 7.8.2 Design Rationale

* We went with a layered architectural system since this would allow us to adhere to agile methodologies much easier than if we had used an architecture that incorporated more waterfall techniques. By using a different system architecture and tightly grouping functionality together, our developers would be more reliant on each other's work before they could move on to the next task. This creates unnecessary roadblocks and means that we wouldn’t be able to work as efficiently as we would have desired otherwise. By using a layered architecture system, we can have one developer working to build the backend and the models that we’ll need in order to support our visualizations, another developer working on the User Interface that will display this data in the frontend, and a final developer working on the endpoint that would connect these two systems if one was needed. This architecture is what we have found to work for us the best after doing research and proper requirements development.

## 7.9 DATA DESIGN

### 7.9.1 Data Description

* The Host a House application uses MongoDB to store all data required for the application users and all the properties data. Mongo DB, unlike SQL databases, uses JSON documents instead of tables to store the data. The data is designed with normalized data models. The normalized data model uses references, user\_id as an example, to create relationships between documents. The application will contain two documents, a user's document, and a properties document. These documents will have the models our data will use for the database. The user's model will contain a name with a type of string, email with a type of string, password with a type of string, and isAdmin with a type of boolean. The properties model will contain the user (house owner) with a type of string, a name, an image (which is the path of the image in the website assets), a description, a state, and a city with a type of string. Also, a zipCode, rating, number of reviews, price with a type of number, and reservations will be an array of strings. There is also a review schema that will hold the information users will provide as reviews. The review contains a name, and comment with a string type, and a rating with a number type.

### 7.9.2 Data Dictionary

The properties model document will have three schemas.

* **Review Schema**
  + name: { type: String, required: true }
  + rating: { type: Number, required: true }
* comment: { type: String, required: true }
* **House Schema**
  + user: { required: true, ref: ‘User’ }
  + name: { type: String, required: true }
  + image { type: String, required: true }
  + description: { type: String, required: true }
  + state: { type: String, required: true }
  + city: { type: String, required: true, }
  + zipCode: { type: Number, required: true, }
  + reviews: [reviewSchema],
  + rating: { type: Number, required: true, default: 0 }
  + numReviews: { type: Number, required: true, default: 0}
  + price: { type: Number, required: true, default: 0 },
  + reservations: [{ type: String, required: true, default: 0}]
* **User Schema**
  + name: { type: String, required: true }
  + email: { type: String, required: true, unique: true }
  + password: { type: String, required: true }
  + isAdmin: { type: Boolean, required: true, default: false }

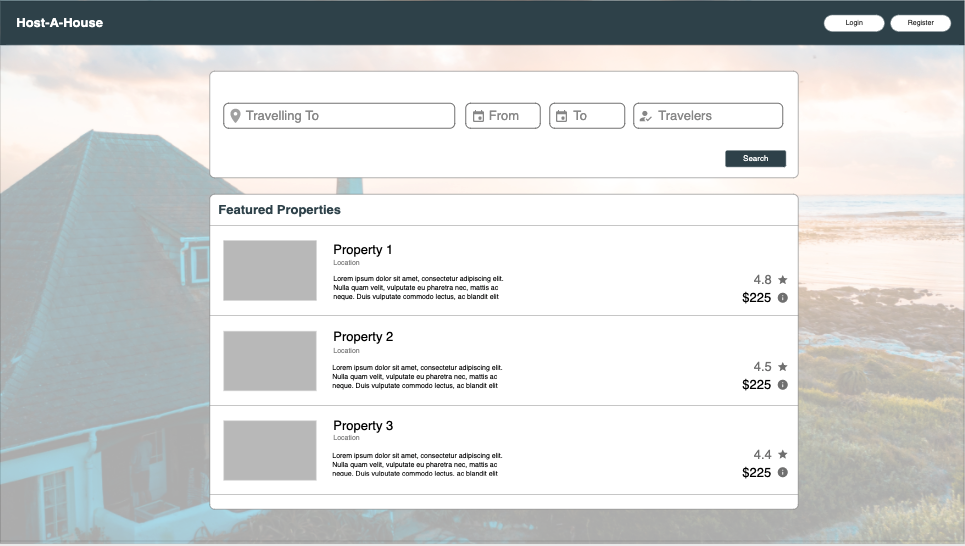
## 7.10 Human Interface Design

### 7.10.1 Overview of User Interface

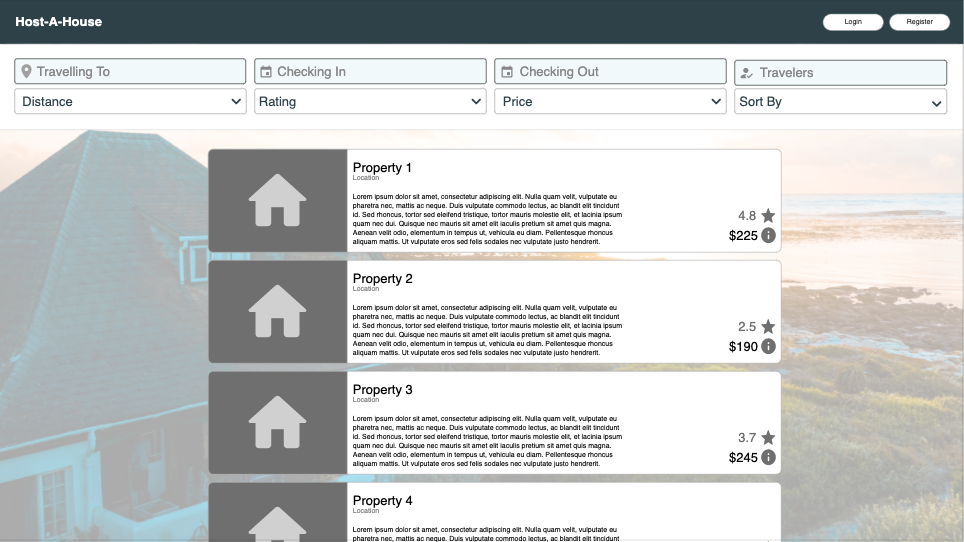
* From a user’s perspective, when they initially hit our application’s webpage they are greeted by our main page. The page is populated with a search section, a featured list, and a navigation bar at the top to allow users to sign in and then subsequently access user management features. By clicking the logo in the top bar on any screen you are brought back to this page. The search bar allows users to input a destination along with check-in and out dates, and how many travelers there will be. if any of this information is invalid, the field is highlighted with a tooltip displaying the error. The featured section simply lists our top 5 properties and when clicked will take you to the details page where users can get more information on the property and reserve the property.
* Logging in and registering are done from the account buttons found on the right side of the navbar. When either of the buttons is clicked, a modal will pop up for the appropriate action. If any of the information is invalid the field will be highlighted red and a tooltip will show giving the user error more information. If there are any errors for any reason, a window alert will populate the screen displaying the error message.
* The listings page is what users are brought to when they complete a search successfully. The listings page keeps the navbar and the search bar at the top of the page but adds more options in order to filter the results that are found underneath the search bar. If no results can be found, the listings found underneath the search bar will be substituted with an error message that says no results found. Each result, if there are any, has its own card that consists of a cover image, the title and location of the property, A brief description of the image, the rating of the property, and the price per night. By clicking on any of the results, the user is brought to a details page that gives more information.
* The details page is what users are brought to when they click on any property on any screen. It consists of a section for images that the user can browse through, the information that is found on the listings page, and more information on whatever amenities the property offers. In some cases, additional information is given for attractions that are close to the property.
* The reservations page is structured similarly to the listings page but has a singular card with two buckets. One bucket contains all the saved properties that a user has selected and presents all of the information that is present on the listing itself. The reserved bucket presents the same information but also has a small section on each property that lists when the reservation is set to take place. If there are no listings in either bucket, the cards will give an error message stating that there are either no reservations or no saved properties.
* Finally, we have the user management page. This is where users can fill out additional information in order to make reservations quicker. If a user has not filled out this information, then they must fill it out whenever they try to reserve a property, so it behooves them to fill it out here. The user management has fields for entering a new password and email and has an address section. There is currently no verification that the address is in fact a valid address, so it is crucial that users enter in the email and check it over.

### 7.10.2 Screen Images

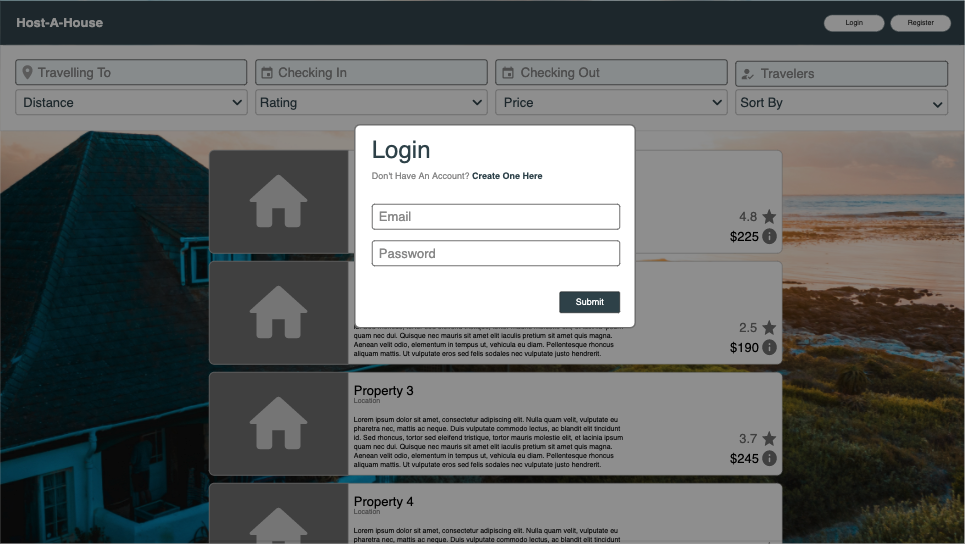
* **Main Page**



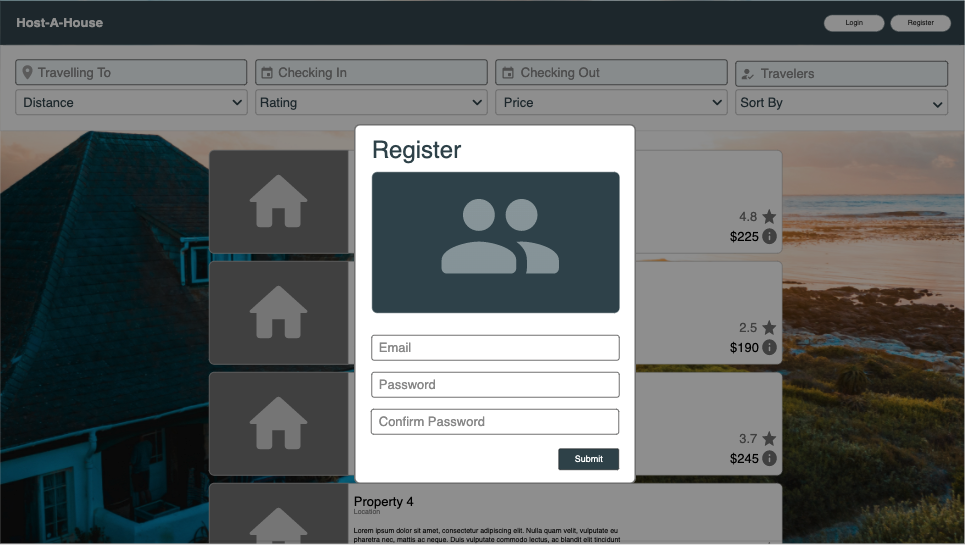
* **Listings Page**



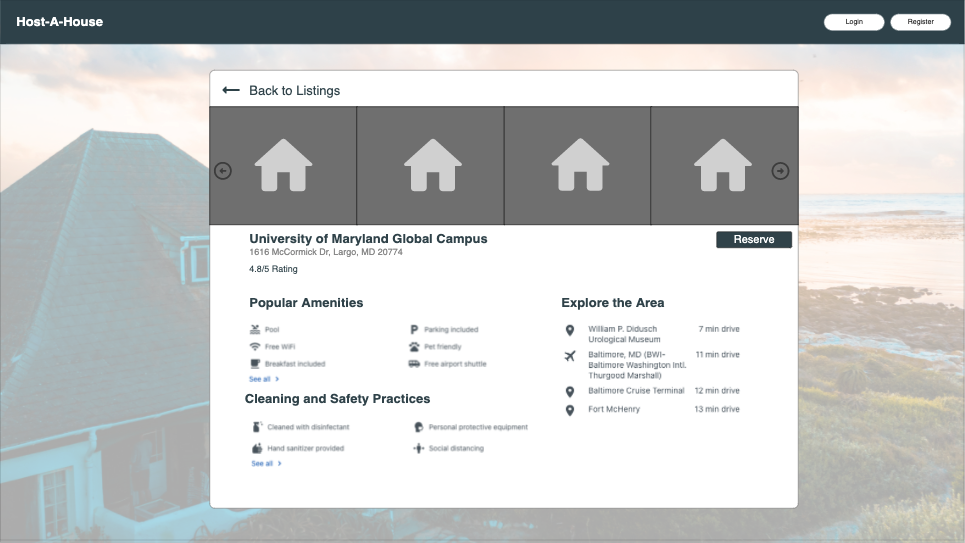
* **Login Page**



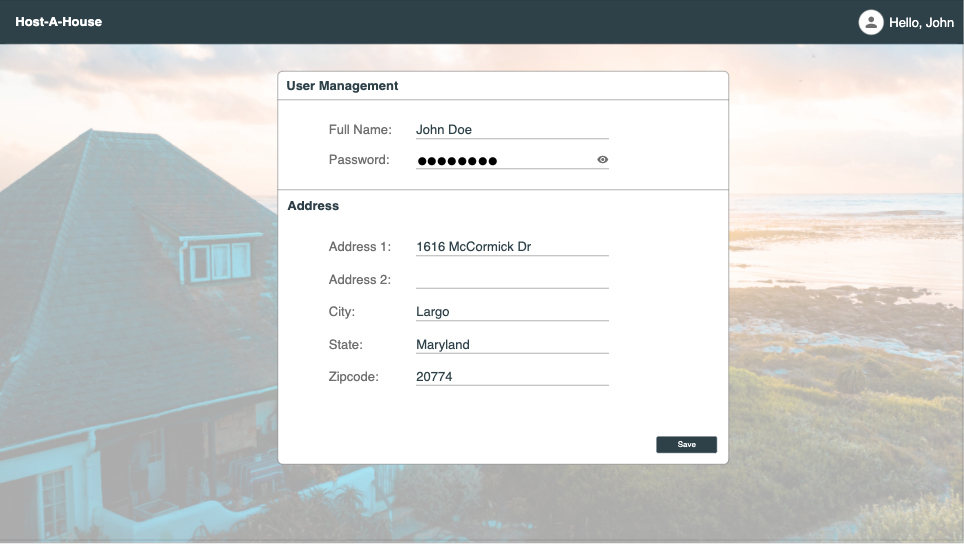
* **Register Page**



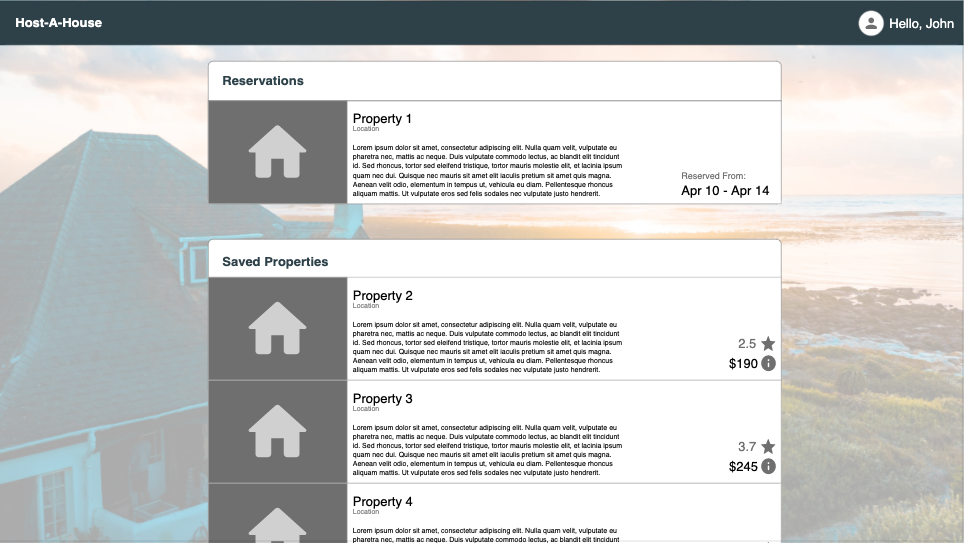
* **More Details**

****

* **User Management**



* **Reservations Page**

****

## 7.11 REQUIREMENTS MATRIX

* Searching rentals by location, date available, and price would be completed via a search bar for input which would tie into the backend to calculate and return results
* Sort rentals by location, date, price, and featured rentals will be accomplished by a drop-down menu.
* Saved/Reservation management will be done by a logged-in user to make reservations, to save properties, and to save/reserve properties this will be accomplished by logging in with a username that would compose the user’s email.
* User management such as creating users and logging in/out will be accomplished by a registration form, the login and logout will be accomplished via a login, and entering a username and password and logout will be accomplished by a logout button

\*See above images for visualizations

|  |  |
| --- | --- |
| **Requirement** | **Expected Output** |
| Rental Properties (1) | Listings Page, Main Page |
| list of rental properties (1a) | Listings Page |
| Search rentals (1b) | Main Page, Listings Page, Search Bar |
| Sort rentals (1c) | Listings Page, Search Bar |
| Featured rentals (1d) | Main Page |
| User Management (2) | Account -> Account Management |
| Create user (2a) | Account -> Register |
| Login/Logout (2b) | Account -> Login |
| Saved Property and Reservation Management (3) | Account -> Saved/Reservation |
| Make reservation (3a) | Main Page, Listing Page -> reserve |
| Save property (3b) | Main Page, Listing Page -> save |

# **Development History**

# **Conclusions including lessons learned, design strengths, limitations and suggestions for future improvement**