Requirements Report

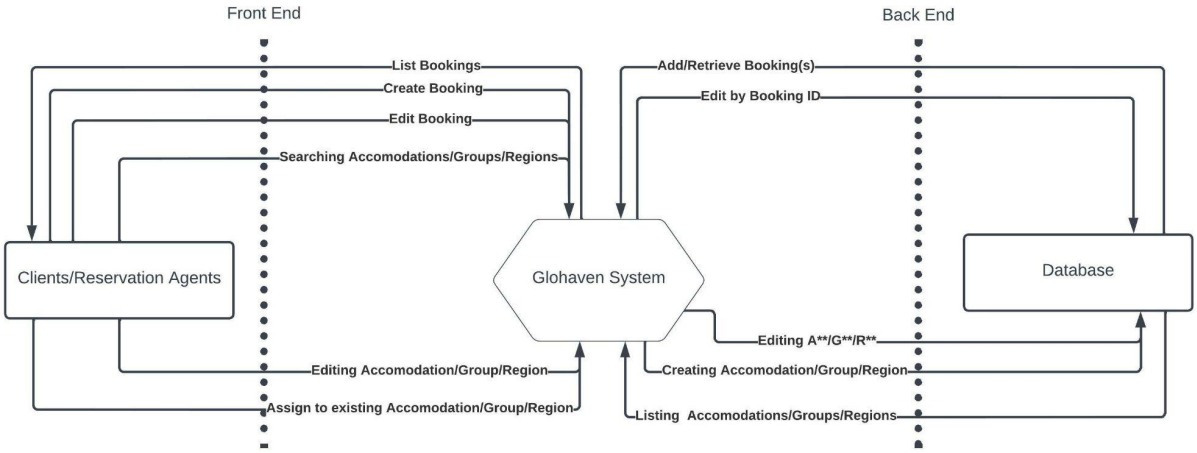
High-level description of software and target user groups: Mashad

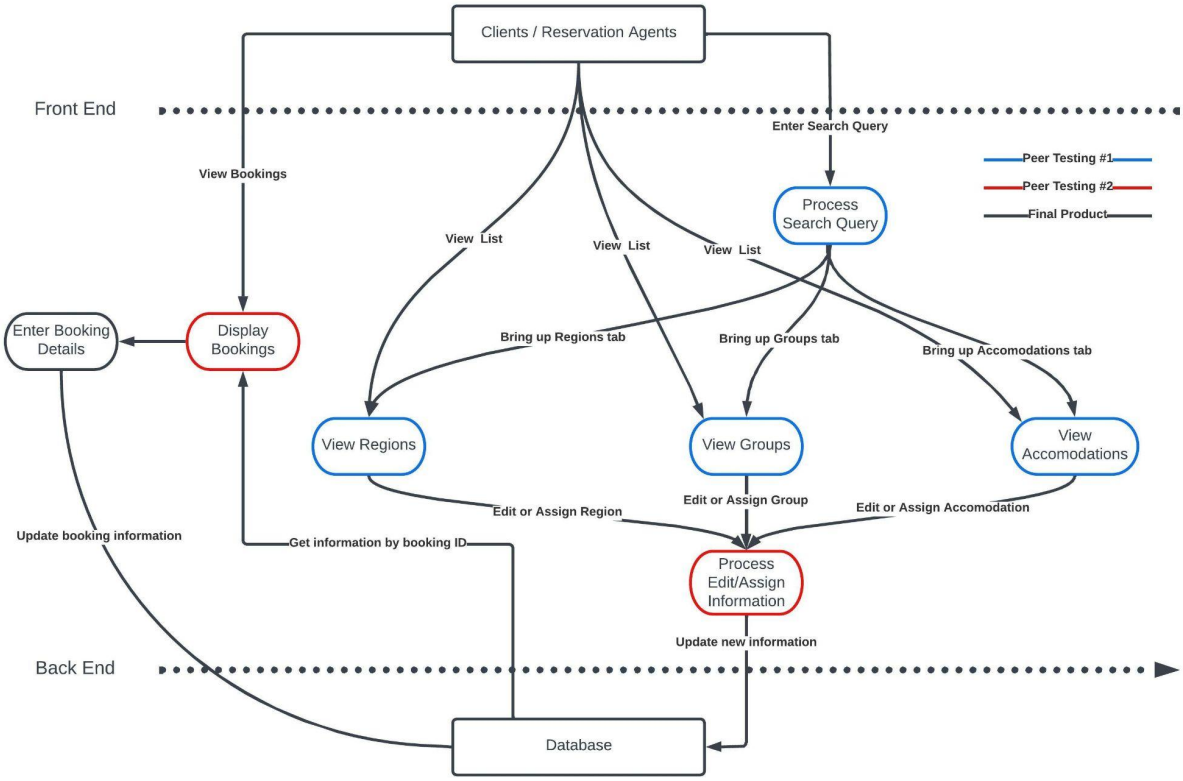
In the event of an emergency, the last thing disaster response personnel should go through is a convoluted software to manage information and resources. Therefore, our software is an Emergency Management Solution that will allow the user to seamlessly browse through a list of all the possible accommodation services available during an emergency. Be it wildfires, pandemics, or floods, a simple yet useful software is key to resolving emergency situations.

Our dashboard will include a search bar to search up Accomodations, Groups or Regions, as well as a summary of all accommodations. Then in our accommodations page, a snippet for every accommodation will be listed, along with information such as Name, Address and Daily Room Rate and more.

Our target user group/s include the government, the Disaster Response Management (EMBC), and indigenous communities. These clients will then use our software to supply useful information to other target groups like evacuees, firefighters and the military.

DFD at levels 0 and 1: Jon





List of functional requirements for each milestone: Mohammed

- User sign in
- Allows the option to search
- Search includes accommodations, regions, or groups
- The system allows user to check for passed booking
- The system allows user to make new bookings
- Regions, accommodations, and group lists are available
- Make sure the user has input in all required fields
- The system prompts user to save after inputting new information
- User input can be checked for validity (eg. email)
- The system recommends based on user input (eg. pet friendly)
- Give out confirmation # of booking
- The system notifies the user if there are any booking problems.

List of non-functional requirements and environmental constraints: Mohammed

- The reliability will be determined reliable if it passes tests with accuracy above 75%
- The software will be easy and simple to use by making the interface more vivid and information really easy to access (Usability Inspection)
- By using MariaDb our software is more adaptable as it is open source and has compatibility with most industry-standard technology
- The software will be secure as we are allowed to develop it locally. Don't allow weak passwords for account creation, the same email can't be used twice.
- The user won't be adding much data to affect scalability, and specific people user platform removing any chance of overload
- In case the application process cannot connect to the database, the process shouldn't crash. It should terminate and prompt an error message

Tech stack (report what client wants): Brendan

The tech stack for the project will be mariaDB and redis/ Laravel/ Breeze / Vue & Inertia. Everything but the backend database connection is required by the client, with MariaDB and Redis being suggested by the client as they work well with the rest of the stack. (sentence about why we chose maria or redis). Laravel is the backend PHP framework,this is used along with Breeze which provides authentication features to the web app. The frontend for our web app will be handled by Vue. Vue is a javascript framework with some interesting features such as being able to handle HTML and CSS in one file. Inertia is used to bridge the gap between the backend and frontend. It allows for the server side rendering of views to instead be Javascript components.

Testing and integration methods: Brendan

The tech we will be using for testing is Laravel, Vitest and Github Actions for continuous integration. Laravel is built with testing in mind, so nothing extra is needed to create and run tests for the backend of our project. Vitest is the recommended testing framework for Vue. Github actions will be set up for both of these tests to automate some of the testing before a merge to the main branch. Github actions could also be set up to track execution time between versions and give us helpful information to improve our code. Running all previous tests again will be done when merging to the main branch to ensure that everything works as intended. In the future if running all tests becomes too costly we can implement Test Case Prioritization to only retest features that have the highest likelihood of being affected. Unit Tests for individual functions will be done alongside the creation of the function. Unit Tests along with Feature tests will be done manually before while working on a feature and added to the automated tests.