COSC 499
GEMS Group A

REQUIREMENTS REPORT

YOUR CONNECTION TO LOCAL™

Glohaven™
Community Hub

Michael Vallido
Tanner Wright
Gabe McLachlan
Andrew Dunn

Description of the GEMS Software

The Glohaven Emergency Management Solutions (GEMS) will be a web-based application, made to help manage victims, accommodations, and entire regions when emergencies occur. This will be a web app with a home page, login, and dashboard where most of the functionality of the app will reside

The dashboard will contain a variety of features.

There will be a place to input the information of accommodations like their location, how many people they can hold, the amenities nearby, whether they're pet/child friendly, and many other possibilities. It will be like a place to register hotel rooms, but more general.

Similarly, there will be a place to input group information, like whether they're victims of the disaster, or emergency services personnel, their contact information, how many people there will be in the group, and their individual needs. It's important to note that though this information is about the group, it is an agent that registers this information, so there will be a logical order, or follow through

The biggest and most important feature of the app is the functionality to put these accommodations and groups together based on compatibility. It will be important that this process is streamlined, so it will probably be handled via a search that will list all compatible or near compatible accommodations.

The other features and services will serve to assist one of these three main features, like a search bar that will search for accommodations, or a list that lists currently occupied accommodations or other features of this sort. All of this to help make the entire registration and management process less stressful for our target user groups.

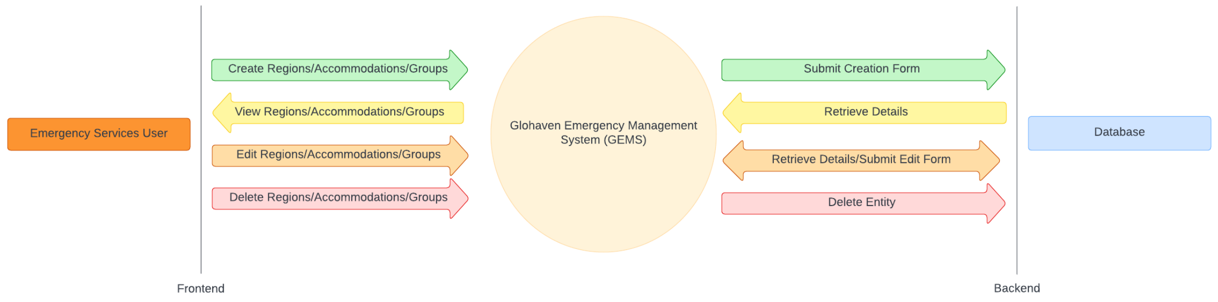
Target User Groups

The target user group is solely registration agents of municipal/provincial emergency services: those who have to manage the safety and security of people when disaster strikes. They will be the ones to handle where these groups come from and connect them and set them up with the accommodations for their relocation.

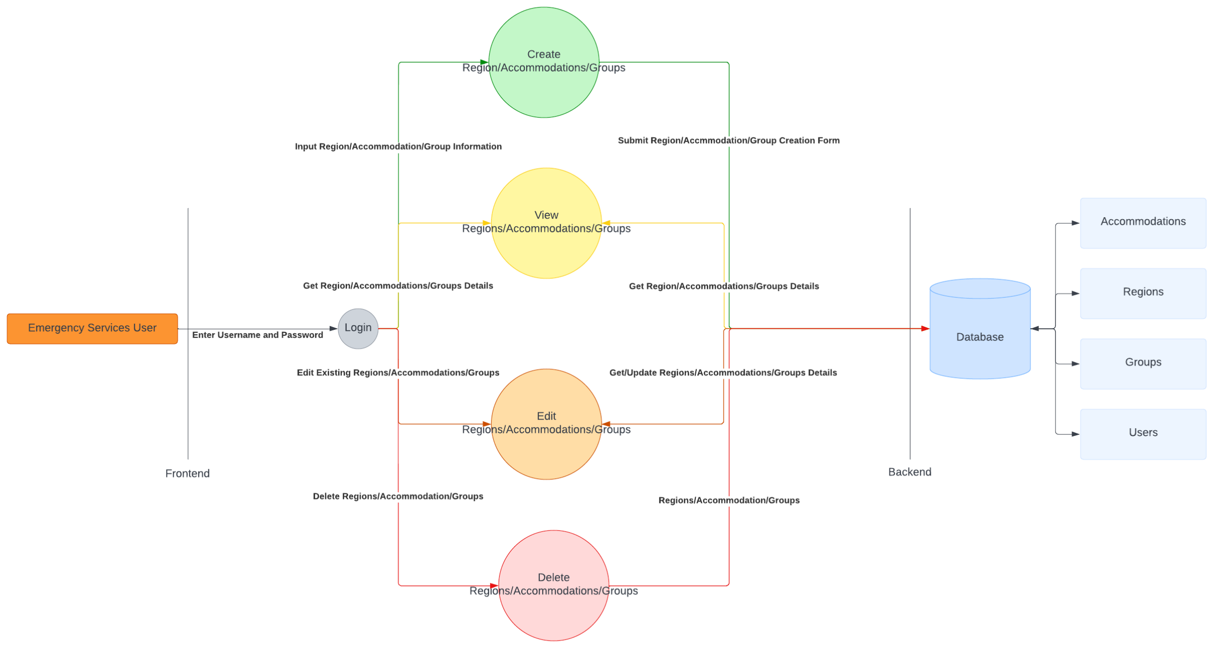
As of October, there are no additional user groups being considered. This being said, depending on the progression of the project, there is the possibility of adding functionality to allow the groups of people to register their own groups to streamline the accommodation process. These users would need to register their cohort, give the information regarding physical/accommodation needs, and be able to access this registration when dealing with our primary users, the reservation agents. As stated before, though, this isn't being considered at this moment.

Diagrams of the System Architecture

DFD Level 0


Emergency Services User
Create Regions/Accommodations/Groups
View Regions/Accommodations/Groups
Edit Regions/Accommodations/Groups
Delete Regions/Accommodations/Groups
Glohaven Emergency Management System (GEMS)
Submit Creation Form
Retrieve Details
Retrieve Details/Submit Edit Form
Delete Entity
Database
Frontend
Backend

DFD Level 1


Create Region/Accommodations/Groups
Input Region/Accommodation/Group Information
Submit Region/Accmmodation/Group Creation Form
View Regions/Accommodations/Groups
Get Region/Accommodations/Groups Details
Get Region/Accommodations/Groups Details
Emergency Services User
Enter Username and Password
Login
Edit Existing Regions/Accommodations/Groups
Get/Update Regions/Accommodations/Groups Details
Edit Regions/Accommodations/Groups
Delete Regions/Accommodation/Groups
Regions/Accommodation/Groups
Delete Regions/Accommodations/Groups
Frontend
Backend
Database
Accommodations
Regions
Groups
Users

DFD Description

Throughout the course of the first two milestones, we will be aiming to complete the four main functions of the application. Peer testing #1 will include the implementation of the main entities creation, edit, and viewing. Peer testing #2 will consist of improvements taken from peer testing #1 in addition to deleting and filtered searching for entities. In between peer testing #2 and the final product, we would like to have completed the main functionality of the app and include a public facing aspect of the application, activity heatmap, and API routes if time allows.

List of Requirements

Milestone 2

Functional

- A secure user login page. Needs to be able to take in many different types of data and keep it secure.
- Needs multiple pages such as accommodations, groups, regions, bookings.
- Data saved in one page needs to be linked to other pages, i.e. an accommodation is offered by a group, which is located in a region.

Non-Functional

- Ease of use, the ease and speed with which an emergency services worker can enter data is very important.

Milestone 3

Functional

- An ability to search the database for accommodation with specific features (wheelchair accessible, pets allowed, etc) in (x) radius kilometers from (y)
- Change values of existing values in the database such as rooms available based on people that have booked.
- Display data entries as well as create new data entries for all of regions/groups/accommodations

Non-Functional

- The database must be able to handle as many different entries as possible, heavily scalable.

Milestone 4

Functional

- Public facing page that does not require login
- Ability for the public to request bookings using the public facing page.
- Algorithm to place multiple groups of people that want to book in 'optimal' accommodations for each?
- API Routes
- Heatmap of activity in a given region.

Non-Functional

- The filtering/searching algorithm must be as fast as possible to accommodate placing people with accommodation quickly during an emergency.

The Tech Stack

The Tech Stack for this project is relatively flexible. The only hard requirement was that the stack had to include the MVC framework for PHP, Laravel, and they highly recommended using Vue alongside it with Inertia.js.

Backend

The only hard requirement of the project is that we must use Laravel, an MVC framework for the PHP scripting language.

For consistency with the rest of the section, here are some pros and cons to this backend.

Language: PHP

Framework: Laravel

Pros:

- Very old, established, and standardized language
- Well documented both on Stack Overflow and at https://www.php.net

- Laravel itself has exceptional documentation
- Tanner and Gabriel have worked with Laravel during their Co-ops
- Laravel provides many packages for things like
    - linting
    - authentication
    - testing
    - much more
- Laravel handles many of the nuances with regards to databases and security middleware

Cons:

- Odd syntax
    - $ before variables
    - C++ like without the versatility
    - use of arrows instead of dots
- Inconsistent type hinting for functions and arguments
- Not as cutting edge as some other backends
- Andrew and Michael haven't ever even touched PHP
- Initial setup is challenging if we're not creating a strictly PHP app (which we're not)

Though there was no choice in this respect, we're still alright with using PHP and Laravel. The versatility that this backend offers is appreciated, and the fact that Gabriel and Tanner are familiar with the framework is an asset.

Frontend

The frontend is where the team had a large degree of freedom. After looking into Laravel we found that there are a few different frontend frameworks that integrate seamlessly with our required backend.

After looking into Inertia.js (the JavaScript package the client recommended using), we found that there were a few different ways we could go. Vue was the recommended frontend, but we also could possibly use React and Svelte. So, we looked into each and here are some of the pros and cons we found with each:

Vue

Pros:

- Good for modern single page apps
- Multiple APIs allow for different programming paradigms
- Documentation is clean, and separates its different versions and APIs
- Recommended by the Glohaven development team
- Gabriel and Tanner worked with this in their Co-op Cons:
- Two versions actively maintained
    - Newer version's documentation is still a work in progress, but older version can be considered bad practice
- Multiple APIs mean that inconsistencies can cause friction

React

Pros:

- Oldest, best established of the three
- Numerous guides and Stack Overflow questions
- Documentation is exceptional
- Michael has used it during his Co-op and work experience

Cons:

- Documentation doesn't cover the framework's entire functionality

Svelte

Pros:

- Cool name
- Newest of the three frameworks, and has lots of buzz about it

Cons:

- Nobody has used before
- Not as configurable as the other frameworks
- Younger life means that it's not as well documented
- Does things a bit differently from the standard the other two set

Initially we narrowed our options down to React and Vue, since those are both frameworks that at least one person has worked with. It was a close match, but we decided on using Vue, primarily because it's what the Glohaven team advocates for, plus multiple people in the group have used it in the past.

This being said, it's all relatively arbitrary, as all of them are easy to pick up if you've used HTML and JavaScript before, and all of them do the same job in a slightly different way.

Auxiliary

There are many choices for things like containerization and databases that it's all relatively arbitrary, as Laravel handles a majority of the difficult parts.

Therefore, for brevity, we're only going to go through the choices that our group decided upon and leave it at that.

Containerization: Docker

Pros:

- Laravel provides out of the box Docker setup and integration via Laravel Sail
- Lightweight: each container only has the files and dependencies for the single app it contains
- Helpful GUI via Docker Desktop, available on all systems
- Whole team has used it before
- Standard in the web applications domain

Cons:

- Post initial setup configuration can be more trouble than it's often worth
- Modulation can be non-trivial to those familiar with Virtualization

Database: MySQL

Pros:

- Whole team has used extensively before
- Integrates seamlessly with Laravel

    - In fact, is the default database for Laravel
- Open-source implementation via MariaDB
- Easy to set up in Docker

Cons:

- Whole team is kind of tired of using MySQL for the *n*th time

IDE: Visual Studio Code

Pros:

- Open source and highly extensible
- Available on all systems
- Heavily customizable

Cons:

- Personalization could lead to inconsistencies, will likely need QA to set guidelines/recommendations for extensions

Other Tech

Just a short list of some other technology that we'll have in the tech stack

- Inertia.js for frontend-backend communication; recommended by Glohaven team
- Windows Subsystem for Linux for those not running on a UNIX system.
- GitHub the prime choice for a remote repository
- Tailwind makes CSS a breeze
- To be expanded as time goes on

Test Explanation

When new code is being added, users must create a PEST test for the code being developed.

PEST testing is commonly used with the Laravel framework since both use PHP. Creating a PEST test will require 3 steps to be completed for a feature to fully be tested.

Prepare your test first (creating models from factories, other setup) Act – do the test, execute the code that is being tested Assert that the 'Act' step has had the intended consequences

By fully implementing PEST testing users will be able to test their features to ensure no uncaught errors make it to the production stage.

GitHub also offers the ability to create workflows through the Github actions. Creating a workflow to ensure that both the frontend and backend are correctly tested by checking error messages and html errors and error responses. This can be easily implemented with the use of existing automations with Docker and Github.

Implementing these will help limit the number of errors that are not picked up from PEST testing as well as Manual testing.

Manual testing will also be the final step in guaranteeing no broken code ends up in production.

By making sure that the features you implemented work correctly by manually trying to see if they work in many different ways, as well as any potential areas you may have changed are correctly functional.

Using these three methods of testing will help limit errors and unwanted bugs in our project.