Emergency Social Network Application Team: RW2 CodeX

A system that provides civilians with the social network they can use in case of emergency. It supports the small communities of civilians that are seriously affected by natural disasters such as earthquakes, erosions, tornadoes, tsunami, etc…
...

Technical Constraints

- The system will be deployed on heroku
- The application will be run via the browser, and require internet connection to be used
- Clients may connect to the application using mobile devices and/or personal computers.
- Back-end tier will be made of Node.js and express.js
- The system will have the RESTful API working without user interface,
- The system will support the real time dynamic updates.

High-Level Functional Requirements

The application will contain the following features:

- Join the community
- Chat publicly
- Share Status
- Chat Privately
- Post announcement
- Search information
- Read-and-support
- Emergency panic button
- External health service companies
- Administer user profile
- Refactor/ update post announcement

Non functional requirements

Usability, extensibility and testability.

Design patterns

1. **Singleton**: we will use this pattern to avoid creating many instances of the same object many times. This pattern is used in controllers where we export the class that plays as the API controller.
2. **Filter:**we have used this pattern to filter the search features to search content based on the user's criteria.

Architectural Decisions with Rationale

- Client-Server is the main architectural style
- Server-side JS (node.js) for small footprint and performance
- Lightweight MVC on the server side using **express** framework
- RESTful API provides core functionality and reduces coupling between UI and back-end
- Web-sockets allow event-based fast dynamic updates.
- Well structured code, and camelCase variables is our style.
- MongoDB (NoSQL) is our utilized database.
- We will use a kind of delegation to let each part independently and do a specific task.
- Using modular programming.

Design Decisions with Rationale

- Encapsulate data and behavior in models for easy testing and better modularization.
- Group the same type of code files together for better structure and better modularization.
- Differentiate tasks with different files and folders, allow each to perform a particular task known to it. **Eg:.Models are created in model package**
- The database will be fully normalized to keep retrieval simple and accurate.

Responsibilities of Main Components

- **models:** encapsulate models and the schema of all database collections.
- **controllers**: encapsulate all CRUD operations to/from the database.
- **front_end**: encapsulates the user interface designing code and activities.
- **Middlewares**: encapsulate all middlewares operations needed and used in the system.
- **Utils:** this component encapsulates all helper classes, files, and functions

UM deployment Diagram

Find it here (1st diagram)

UML package diagram

Find it here (2nd diagram)