Document: Initial Planning and Ideas
Date: 12 August 2024
Contents: Application brainstorming, system architecture style and next steps

Brainstorming for the application

- Reporting system: Quick easy-to-access button to make a report. Should allow user to easily make a report of a particular type
- Reporting types:
  - Medical emergency
  - Weather issue
  - Fire incident
  - Natural disaster
  - Security threat
  - [?] Road report

- Teams to link up with:

Campus Transportation	[!Required] Have their system listen to our api to help ensure safe transport – main concerns on weather issues, natural disasters, fire incidents and security threats. Also must allow Campus Protection services to respond to incidents rapidly: use their GPS system for quick navigation, etc.
Events & Activities	Have their system listen to our api for any issues in the Wits area – allow them to handle their actions in the event of an emergency.
Infrastructure Management	Have their system listen to our api to help ensure availability of classrooms – important cancellation of classrooms in the event of fires or natural disaster. [?] Can backward integrate using their reporting system as a report event for us to display. [Issues] Does not promote much integration and could complicate the system too much. This feature is likely not needed since the Infrastructure Management team will have their own reporting feature
Dining Services	Have their system listen to our api – in the event of a fire or natural disaster, help with evacuation notice. [Issues] No way of backward integration: we will not need any of their features in our app.
Campus tutoring	[??] Not many options with this group

System Architecture Style

- Combined Micro-service and Multi-tier
- Micro-service architecture:
  - Reason for choice:
    - Inspired by the system's similarity to Uber: Having multiple services available that need to be integrated
    - Variety of "micro-systems" means we need to ensure SOLID principles in the future
    - Easiest way to do so is to split systems is to separate into smaller services that can be integrated >> helps set up our system to make sure it ensures (for example) the Single Responsibility Principle
  - API topology
    - Small services with few modules
    - Lightweight so API is best
    - Want to ensure services are decoupled and do not grow to unsustainable size
  - Services:
    - Emergency Alert service
    - Incident Report service
    - Safety Resource service
    - Push Notification service
    - Location service
- Multi-tier architecture
  - Reason for choice:
    - Allows us to separate concerns of parts of the application
    - Means we can keep our system of microservices isolated from other points of contact
    - Can handle traffic as needed to ensure reliability of system (no shortcutting to database from frontend)
    - Ensure scalability
    - Allow easy testing
    - Only 3 layers so complexity is not too great
  - Tiers:
    - Presentation: Frontend
    - Business (Service): Backend services (see above services)
    - Persistence: Handle data access
    - Database: Actual AZURE database

Next steps

- Create a backlog of items
- Decide which items are relevant for the first sprint
- Develop user stories and their UATs
- Frontend team:
  - UI/UX scope development
  - Research of application layout and structure
  - Wireframe to Mockup development
- Backend team:
  - Full integration with database
  - Planning in accordance with chosen system architecture style
  - Secure API setup (keys / tokens , etc.)
  - Look at OpenAPI and start setting up the doc