avenuDub Design Document

1. Overview

1.1 Project Summary

avenuDub is an app to familiarize UW students with the Ave and businesses surrounding campus. It includes a catalog of local businesses, including student discounts available nearby. It also includes safety information important for students to be aware of while on the Ave.

1.2 Problem Statement

- **Problem Definition**: Students new to UW are often unaware of local businesses and discounts available to them there. They may also be unaware of safety considerations to keep in mind while navigating campus.
- **Root Cause**: There are no resources available to familiarize students with the areas around campus.
- **Solution Approach**: Create a comprehensive guide of nearby businesses, discounts, and crime to enable students to discover more spots around campus with safety.
- **Target Users**: Students new to UW.

2. Team Structure

2.1 Leadership

- **Project Lead**: Samantha Autrey
- **Club Lead Guide**: Joey Kang

2.2 Team Composition

Role	**Name**	**Responsibilities**
Team Manager	Ananya Tripathi	-Onboarding new team members -Keeping track of/updating the task board in GitHub
Lead	Samantha Autrey	-Keeping track of holistic project progress -Leading/organizing meetings

		-Maintaining firm deadlines/expectations of progress
Frontend	Megan Yam Ananya Tripathi Samantha Autrey	-Managing the user interface; first point of contact if something looks incorrect -Reporting on UI progress/development setbacks -Communicating with the backend team to ensure smooth connection
Backend	Joonho Choi Nimai Belur	-Managing the back-end of the application, including databases, API's, and web scraping technology -Reporting on development/data setbacks -Communicating with the front end team to ensure smooth connection

3. Product Requirements

3.1 MVP Features (P0)

Feature	**Description**	**Success Criteria**	**Testing Approach**
Student Discounts	Upon clicking a business on the map, users will be able to see if they offer discounts for students and what they have to do in order to verify student status. Data will be gathered from the UW provided list of student discounts, cultural RSOs, etc.	- Users can click on a business and view discount information - Verification method is clearly displayed for each business participating in student discount -	Make sure that a place with known discounts displays said discounts on our map or catalog. Examine each number provided for accuracy - check for inadvertent truncation/incorrect information

Feature	Description	Success Criteria	Testing Approach
Location of Businesses	Pinpoints of locations of businesses and a brief description of what kind of food/products it primarily offers.	- Each business location is marked accurately on the map - Description correctly matches the type of food/products primarily offered	-Make sure all edge cases are caught: ex, business name or attribute is null or empty -Cross reference with already functioning models (an existing navigation platform)
Incident Reporting	Pinpoints on map of where incidents occur within a month. Data will be gathered from UW 60 day crime log, UW/Seattle police, user reports, etc.	- Locations on map matches locations on UW crime log - Old data is erased from map after a month	-Cross reference with actual crime-log to check for location accuracy, check for SQL injection vulnerability -Make sure all edge cases are caught: ex, a field is null or empty -In conjunction with web-scraper testing: ensure that any faulty data is filtered out
Areas of low-visibility/high-risk	For high-risk areas, crime data will be analyzed to calculate what areas crime are more likely to occur in. Areas of high-risk will be highlighted in red. Areas of low-visibility include areas that are not well-lit, or have broken street lights. Depending on whether it's a broken streetlight or an area of low visibility, it will be displayed on the	- High-risk area is appropriately calculated - Data gathered from sources match - Outages that are resolved are removed from map shortly after resolution	-Cross-reference with user submission testing: ensure that user-input is not vulnerable to injection/filters bad data -Ensure that user-input is filtered for database-breaking keywords (null) -Cross-reference generated map points with city data/energy logs

Feature	Description	Success Criteria	Testing Approach
	map as a pinpoint or highlighted in yellow. Data will be gathered from the Seattle City Lights/Puget Sound Energy's outage map and from user reports		

3.2 Post-MVP Features (P1/P2)

Priority	Feature	Description
Low	Menu	When clicking a restaurant on the app, the user will see what they have to offer and their prices. If implemented, data will be gathered from a range of sources, such as Yelp, business' website, etc.
Very Low	Events	Locations of local events happening on the Ave. This could be anything ranging from concerts, food festivals, farmers markets, etc. If implemented, this feature may rely solely on user input.

4. User Experience Design

4.1 Wireframes

- **Design Links**: https://www.figma.com/team_invite/redeem/ZMC4xu8M1VgGMQzpuTp7eF
- **Design System**: Material Design

4.2 User Journeys

Journey 1: Students Discovering Businesses and Safety Tips for the Ave.

1. Initial State: Splash Screen -> Map with information (similar to Google Maps)
2. User Actions:

- Tap on a point on map and see information for that location
- Click on search bar to find a business
- Toggle safety information and business information on/off using buttons under search bar
- Double click/hold either toggle to get a text list version of either the safety or business information
- Click user account button on bottom left to access settings page.
- Click report button on bottom right to report some sort of safety hazard.

3. End State:
   - Search Bar: User sees dropdown with desired results, as well as history of past 5-6 searches, and the option to filter results
   - Detailed Catalog of businesses/crime information, with favorited locations further up
   - User account: Includes name, account information, favorites
   - Reporting page: can request user location (or user can manually input them), and they can input a description of the issue + a photo

Journey 2: Businesses Adding/Modifying their Information

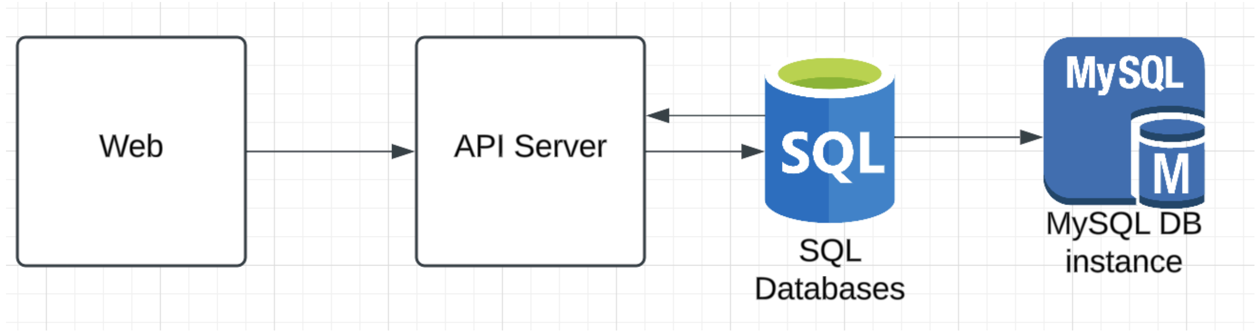
1. [Same as Journey 1, with the following additions]
2. End State:
   a. User Account also includes business information, and allows modification of their information (and does not include favorites)

5. Technical Architecture

5.1 System Overview

[Insert high-level architecture diagram]
Check out https://app.diagrams.net/

https://lucid.app/lucidchart/112c3255-cc64-48bd-9e00-24a8625617e1/edit?viewport_loc=-11%2C-11%2C1579%2C671%2C0_0&invitationId=inv_a0dd726a-b151-4247-a27d-720c1b32d0c2



5.2 Frontend Specification

- **Frameworks**: *React Native*
- **Key Libraries**: *Mapbox, Beautiful Soup*

Core Components

Component	Purpose	Props/State
BusinessMarker	Used to store information about a specific business on the ave. When clicked on, the user can see relevant information like its name, address, and student discounts if applicable.	**Props** *name*: string, *address*: string, *discountInfo?*: { { *discountDesc*: string, *verifyMethod*: string} }, *mainProduct*: string, *coords*: {{*xCoord*: bigint, *yCoord*: bigint}} **State** *descVisible*: boolean
IncidentMarker	Marks map on location where crime happened. When clicked on, users can see any additional information if applicable.	**Props** *crimeType*: string, *address*: string, *coords*: {{*xCoord*: bigint, *yCoord*: bigint}}, *date*: {{month: number, day: number, year: number, hour: number, minute: number, second: number}}, *addtInfo?*: string **State** *descVisible*: boolean
HighRiskLowVis	Highlights areas where crime tends to happen, or areas that have low visibility at night (not including street lights). When clicked on, the user can see relevant details, such as list of crimes that happened in the area	**Props** *highRisk:* boolean, *location:* List<Path>, *addtInfo?:* string **States** *descVisible*: boolean
BrokenLight	Pinpoints location of where there's a broken street light. When clicked on, the user can see additional details,	**Props** *coords:* {{xCoord: bigint, yCoord: bigint}}, *address?*: string, *startDate:* {{month:

Component	Purpose	Props/State
	such as photo proof and start date.	number, day: number, year: number, hour: number, minute: number, second: number}} **State** *descVisible*: boolean

5.3 Backend Specification

API Endpoints

Endpoint: [PATH]

Method: [GET/POST]

Description: API is called on when the user wants to fetch data about a location or upload information about crimes/businesses in their area. Different use cases will dictate what the API request and response will look like.

POST Request: Adds crime/business information to the database

```
{
  "type": "object",
  "properties": {
    "reportType": {
      "type": "string",
      "enum": ["crime", "business"],
      "description": "Type of report being submitted"
    },
    "details": {
      "type": "string",
      "description": "Details about the crime or business information"
    },
    "location": {
      "type": "object",
      "properties": {
        "latitude": { "type": "number", "description": "Latitude of the location" },
        "longitude": { "type": "number", "description": "Longitude of the location" }
      },
      "required": ["latitude", "longitude"]
    },
    "timestamp": {
      "type": "string",
      "format": "date-time",
      "description": "Time the incident or business info was reported"
    },
  },
  "required": ["reportType", "details", "location", "timestamp"]
}
```

EXAMPLE:

```
{
  "reportType": "crime",
  "details": "Suspicious activity observed",
  "location": {
    "latitude": 47.6062,
    "longitude": -122.3321
  },
  "timestamp": "2024-11-19T10:00:00Z"
}
```

Response:

```
{
  "type": "object",
  "properties": {
    "status": {
      "type": "string",
      "description": "Status of the report submission (e.g., 'success' or 'failure')"
    },
    "reportId": {
      "type": "string",
      "description": "Unique identifier for the submitted report"
    },
    "message": {
      "type": "string",
      "description": "Additional information about the status"
    }
  }
}
```

EXAMPLE:

```
{
        “status”: “Success”,
        “reportId”: “abcdef”,
        “Message”: “Submitted successfully”
}
```

GET Request: Gets crime/business information from database

```
{
  "type": "object",
  "properties": {
    "type": {
      "type": "string",
      "enum": ["crime", "business"],
      "description": "Type of information to retrieve"
    },
    "location": {
      "type": "object",
      "properties": {
        "latitude": { "type": "number", "description": "Latitude of the location" },
        "longitude": { "type": "number", "description": "Longitude of the location" },
        "radius": { "type": "number", "description": "Radius (in kilometers) to search" }
      },
      "required": ["latitude", "longitude", "radius"]
    }
  },
  "required": ["type", "location"]
}
```

EXAMPLE RESPONSE:

```
{
  "status": "success",
  "data": [
    {
      "id": "crime123",
      "type": "crime",
      "details": "Robbery at 5th Avenue.",
      "location": { "latitude": 47.6097, "longitude": -122.3331 },
      "timestamp": "2024-11-18T14:30:00Z"
    },
    {
      "id": "crime124",
      "type": "crime",
      "details": "Car theft reported.",
      "location": { "latitude": 47.6100, "longitude": -122.3340 },
      "timestamp": "2024-11-19T08:45:00Z"
    }
  ],
  "message": "2 records found."
}
```

Status Codes:

- 200: Successful request
- 400: Bad request (invalid parameters)
- 500: Server error

Core Libraries

- **Web Framework (if applicable)**: Node.js
- **Database Driver (if applicable)**: MySQL Connector
- **Other Dependencies**: bcrypt

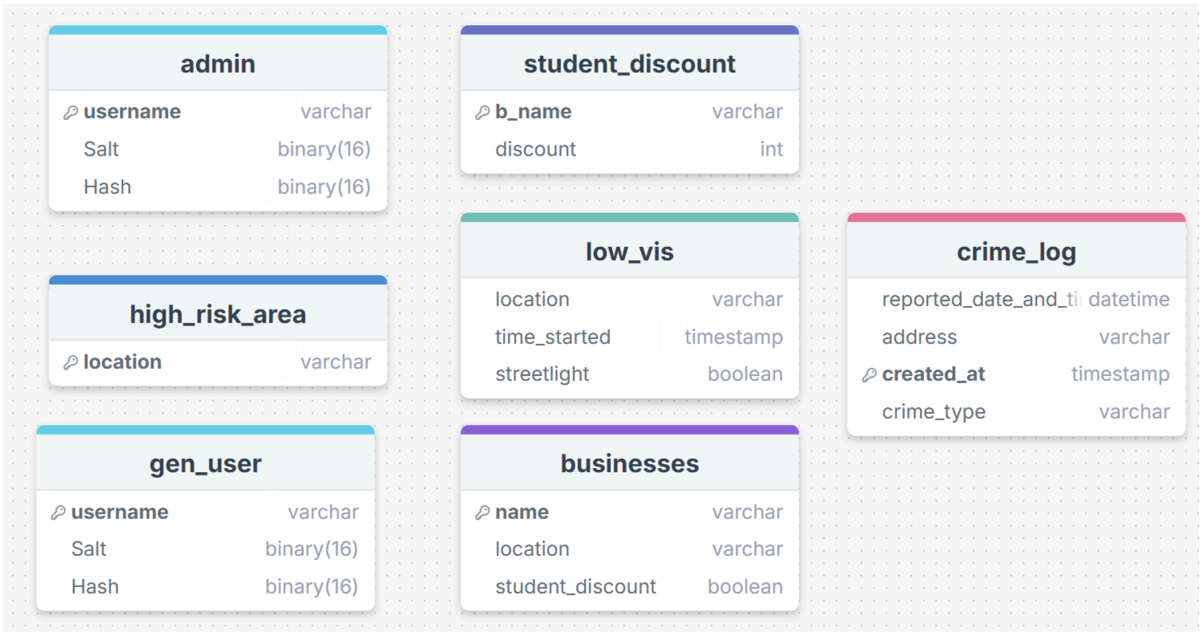
5.4 Data Model

Database Selection

- **Type**: SQL
- **Technology**: MySQL
- **Justification**: Having a table to represent our data is the most appropriate choice for our project, as the data we're storing is simple, making any dynamic storing of data unnecessary and complicated.

Schema

https://drawsql.app/teams/na-849/diagrams/avenudub



5.5 External Services

Service	Purpose	Integration Method
FreeDB	Cloud server that stores SQL tables	MySQL Connector

6. Testing Strategy

6.1 Testing Layers

Layer	Framework/Method
Unit	Serenity/JS, Jest Request (for REST API testing) Pytest
Integration	Serenity/JS - Mapbox -> REST API -> React Native - BeautifulSoup -> Database - Database -> REST API -> React Native
E2E	Appium + Serenity/JS

Layer	Framework/Method
	- Accessing, updating & displaying data from database Android SDK - Ensure UI is navigable (i.e. interactive map displays accurate information legibly) via Android Emulator

6.2 Quality Assurance

- **Code Review Process**:
    - Peer reviewing/testing code for each MVP
    - Using version control (GitHub) to review code before fulfilling pull requests
        - Using branches to prevent version conflict
- **CI/CD Integration**:
    - CI
        - Version Control: GitHub
        - Automated Testing: (See 6.1)
        - Automated Builds: Jenkins + Gradle
    - CD
        - Infrastructure as Code (IaC): Define infrastructure using code (e.g., Terraform or AWS CloudFormation) to ensure consistent environments.
        - Artifact Creation: Docker images
        - Deployment Scripting: Ansible, Kubernetes, or Terraform
            - Pipeline Orchestration: Tekton Pipelines
        - Blue-Green Deployments: Implement blue-green deployments to minimize downtime during releases.

7. Non-Functional Requirements

7.1 Security

- How do we prevent our database from being tampered with
    - Use prepared statements to avoid SQL injection attacks
    - CAPTCHA & filtering to avoid spam attacks (i.e. mass submission of inappropriate content through the user submission field)
    - Establish contingencies for when database fails / is compromised

7.2 Privacy

- Login info: how do we encrypt a user's login information to protect their data?
    - Bcrypt
- Location: how do we ensure that a user's location cannot be accessed?
    - Usage of a secure library for location-based interactive map

7.3 Accessibility

- Accessibility mode
    - Larger fonts
    - Screen reader
    - Vibration alerts for safety
    - Voice-to-text
    - Details on navigational accessibility (i.e. stairs, steep inclines, etc)
    - Colorblindness-friendly UI

8. Project Timeline

8.1 Development Phases

Phase	**Deliverables**	**Timeline**
Learning/Setup	-A working React Native framework on which to start development -A working database back-end that meets our specifications -Working knowledge of python/react in order to meet Front/Back end needs	Fall Quarter
MVP Development	-Working business catalog including student discounts as well as location -Working crime database/corresponding accurate markers on map	Winter Quarter/Early Spring Quarter
Testing	-Ensure UI functionality, including dropdowns, gesture responses (even unintended ones), and buttons/paths	Mid/Late Spring Quarter

Phase	Deliverables	Timeline
	-Ensure database functionality, including proper retrieval/storage of entries	
Post-MVP	-Implement menus for restaurants -Have a separate list for events on the Ave -Catalog locations of utilities/services	Summer Quarter and Beyond

8.2 Milestones

1. Club-wide spec presentation
2. Design review
3. MVP Completion
4. All Tests Pass
5. Showcase!