## **Date: 30/10/2024**

**Problem Statement 10 :**

**Scenario:** In today’s urban environments, personal safety and well-being are increasingly becoming major concerns, particularly for those who live or work in high-risk areas. Individuals may feel vulnerable to threats like crime, harassment, or accidents, and often lack the tools needed to effectively protect themselves or respond to emergencies. Moreover, the stresses of daily life, coupled with concerns for personal safety, can take a toll on mental and emotional well-being. While there are various apps available for tracking safety or providing mental health support, few offer a comprehensive solution that integrates both aspects in a user-friendly manner. The need for a reliable, all-in-one solution that addresses both safety and well-being has never been greater, especially in a world where individuals are more mobile and exposed to various risks.

**Team Details :**

|  |  |  |  |
| --- | --- | --- | --- |
| **Group** | **Learner 1** | **Learner 2** | **Capstone Project Number** |
| 10 | V M V C Jhasank Bharadwaj T | Aman Mohammed P | 10 |

## **Date:01/11/2024**

1. Understanding basic requirement of problem statement.

2. Analysing the different ideas.

3. Generating abstract.

4. Designing basic flow of project.

**Abstract for a .NET Microservice Architecture for CareConnect.**

**Objective:**

To design a .NET-based microservice architecture for CareConnect that provides robust, scalable, and secure services for personal safety and well-being, utilizing external Open APIs for added functionality (e.g., location data, mental health resources). The architecture will include discrete microservices for different functionalities—Safety, Mental Health, User Management, Notifications, and Analytics—each performing specific tasks and communicating through secure APIs.

**Overview of Microservice Components**

**User Management Service:** Manages user profiles, legitimate email verification.= trusted contacts, and location, - permissions.

**Safety Service:** Monitors location data, manages geofencing, triggers alerts, and connects with emergency contacts.

**Mental Health Service:** Provides access to well-being resources like guided meditations, mood tracking, and connects with mental health professionals via Open APIs.

**Notification Service:** Sends real-time notifications (e.g., alerts for high-risk areas) and communications to user and trusted contacts.

**Analytics Service:** Collects and analyzes user data (e.g., location, usage patterns) to provide insights, such as the SafeWell Score.

**API Gateway:** Manages requests between the client applications and the backend services, handles authentication, and routes requests to appropriate services.

**Phase 1: Initial Setup and Design**

Setup Development Environment:

Install .NET Core SDK, Visual Studio or VS Code, and Docker.

Set up Git for version control.

Define Requirements for Each Service:

List the main tasks and functionalities each microservice will handle.

Identify Open APIs (e.g., Map API for geofencing, mental health API for support services) that can be integrated.

Create API Contracts:

Use OpenAPI/Swagger to define endpoints for each microservice.

Example: User Management Service may have endpoints like /users/create, /contacts/add, /profile/update.

**Phase 2: Implement Microservices**

Develop User Management Service:

Technologies: ASP.NET Core Web API, Entity Framework Core for data handling.

Tasks: CRUD operations for user profiles, authentication and authorization with JWT tokens, and trusted contact management.

Database: Configure a SQL database to store user data.

Develop Safety Service:

Technologies: ASP.NET Core, third-party map API (e.g., Google Maps) for location data and geofencing.

Tasks: Track user location, configure geofencing, send alerts when users enter high-risk zones.

Open API Integration: Use map/location Open API to fetch real-time data for alerts.

Develop Mental Health Service:

Technologies: ASP.NET Core Web API.

Tasks: Serve mental health resources, such as guided meditations and connect with external mental health APIs.

Open API Integration: Integrate with mental health resources API to provide data on demand.

Develop Notification Service:

Technologies: ASP.NET Core Web API, Firebase for push notifications.

Tasks: Handle real-time notifications and alert trusted contacts in emergencies.

Develop Analytics Service:

Technologies: ASP.NET Core, a NoSQL database (e.g., MongoDB) for data storage.

Tasks: Generate insights based on usage patterns, track well-being, and calculate SafeWell Scores.

**Phase 3: API Gateway Setup**

Configure API Gateway:

Technologies: Ocelot (a .NET API Gateway).

Tasks: Route incoming client requests to respective microservices, handle authentication and authorization checks, and ensure security.

OpenAPI Documentation: Integrate Swagger for endpoint documentation.

Centralized Authentication and Authorization:

Use Identity Server for token-based authentication across microservices.

Implement OAuth2 and JWT tokens to ensure secure access for each request.

**Phase 4: Deploy and Orchestrate with Docker and Kubernetes**

Containerize Each Microservice:

Write Dockerfiles for each microservice, build, and create images.

Use Docker Compose for initial testing of containers locally.

Kubernetes for Deployment and Orchestration:

Set up Kubernetes clusters to manage deployment, scaling, and service orchestration.

Configure Kubernetes Services for external access and Ingress for routing.

**Phase 5: Testing, Monitoring, and Documentation**

Testing:

Perform Unit Testing on each microservice.

Set up Integration Tests to ensure each service works well with others.

Run End-to-End Testing to simulate user scenarios across services.

Monitoring and Logging:

Set up Azure Application Insights or Elastic Stack (ELK) for monitoring, logging, and alerting.

Track performance metrics for each service to identify bottlenecks.

Documentation and User Guide:

Use Swagger to generate documentation for each service’s API.

Write a user-friendly onboarding document explaining how to deploy, scale, and interact with each microservice.

**Phase 6: Launch and Maintenance**

Launch and Initial Monitoring:

Deploy to production, set up monitoring and alerts.

Conduct a final round of testing and gather feedback from real users.

Continuous Improvement:

Regularly check for updates to Open APIs and security patches.

Use analytics data to improve user experience and introduce new features over time.

## **Date : 04-11-2024**

Agenda

1. Block Diagram / Architecture diagram designing

2. Api architecture.

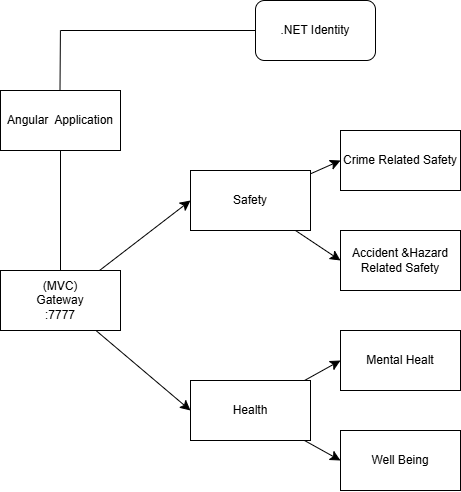
## **Date : 05-11-2024**

1. making changes on Architecture

2. understanding the requirements and designing class structure

3. starting to design class diagram

Fig 1 : Updated Architecture



## **Date : 06-11-2024**

1. Finalising the Class Diagram

2. Finalising the Database Design

3. Analysing Sequence / Activity Diagram

Fig 1 : Class Diagram

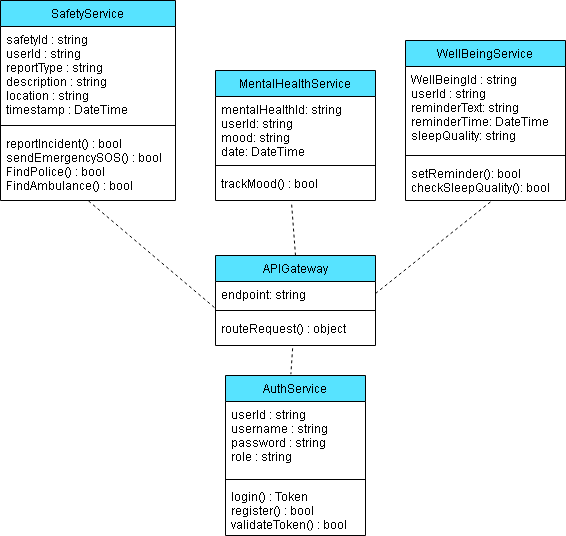


Fig 2 : Database Design :

**AuthService Database :**

Identity Table add ons

ID :

Name:

Username :

Password :

UserDataDto: AutoMapper connection .

|  |  |
| --- | --- |
| Id | Name |
|  |  |

**SafetyService Database :**

UserIncidentReport Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Name of User | TypeOfIncident | Location | Time | Short Description | Submission DateTime |
|  |  |  |  |  |  |  |

ID : Userid from Auth Database => Identity Table (FK)

Name of User : Username from Auth Database => Identity Table (FK)

TypeOfIncident : Dropdown with types. Options (Accident, Crime, Hazard)

Location : Get the location from Google API

Time : Incident Time

Short Description : This will be a string entered by the

Submission DateTime : update db time.

\*\* postReport(), GetReport(), putReport(Id),

EmergencyService Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SerialNo | Name of Service Provider | Phone Number | Email | Address |
|  |  |  |  |  |

SerialNo : Used for just indexing

Name of the service provider : We have 4 Options. Dropdown (Police Station, Hospital, FireForce)

Phone Number : number of provider

Email : email id of provider

Address : Location details of the Service Provider

\*\*getByService(option), getByLocation(), getBySOS() [pass message to nearest service providers]

**MentalHealthService**

Mood Tracker Table :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Datetime | Name | History(bool) | Diagnosis | CurrentMood | Score |
|  |  |  |  |  |  |

Dto-

UserTrackDto:

|  |  |  |
| --- | --- | --- |
| DateTime | Name | Score |
|  |  |  |

\*\*getScore(), (Dto)getUser(Id)

**WellBeingService**

Reminder Schedule Table

|  |  |  |  |
| --- | --- | --- | --- |
| DateTime | userId | reminderText | reminderTime |
|  |  |  |  |

DateTime : Store the timestamp of the submition

userID: id of the user (FK)

reminderText : this stores a string what to show when showing reminder

reminderTime : this stores the time on which it should be displayed

Sleep Quality Table

|  |  |  |  |
| --- | --- | --- | --- |
| DateTime | userId | sleepQuality | Remarks |
|  |  |  |  |

DateTime : Stores the timestamp of submitting the data

userId : UserId of the user (FK)

sleepQuality : Rating bar

Remarks : if anything to store as a string

\*\* postReminder(DateTime,ReminderText), postSleep(Quality, remarks)

## **Date : 07-11-2024**

1. Finalising Activity Diagram
2. Framing UI Concept using Figma
3. Testing Google Maps API

**Activity Diagram :**

Specifications for individual components are provided. Each component is designed with clear interfaces and responsibilities to support modular development and easy maintenance.

