Laboratory work Nr.1

Application Suitability The application has multiple components, like fitness tracking, university management, and personal mentoring. These components can potentially be developed and maintained independently, making it suitable for a microservices architecture. Each module can have its own team responsible for development and updates.

Scalability Even though we aren’t expecting a lot of users, since we are pitching this as an application for UTM students only, as of now, in theory, this application has the capability to scale to more universities in even more countries, which means a large user base, microservices can help with scalability, making it easy to scale individual services based on demand. For instance, the fitness and wellness module might experience different usage patterns compared to the university assistant module.

Resilience Since our application is modular, if one service fails, it won't necessarily bring down the entire application. Users can still access other functionalities.

Third-party Integrations Our application needs to integrate with various third-party services and APIs (e.g., Mi Fitness, calendar services, speech-to text and text to speech services,chat gpt,etc). Microservices can make it easier to manage these integrations separately.

Real Life examples Our applications is pretty similar to Coursera, LinkedIn, Duolingo, MiFitness, which are similar in such matters as notifications, scheduling, progress tracking, and personalized learning. They serve as valuable examples of how technology can enhance the educational and wellness aspects of students' lives. We take inspiration from these platforms while adding a unique combination of features.

Set Up Deployment and Scaling Azure pipelines, and Docker. Azure Pipelines enables you to set up a CI/CD pipeline for Dockerized application. This means that whenever changes are pushed to code repository, Azure Pipelines can automatically build new Docker images, run tests, and deploy the updated containers to target environment, ensuring a rapid and reliable release process.

Docker simplifies the deployment process. You can create a Docker image once and deploy it to multiple environments without worrying about differences in the underlying infrastructure. Azure Pipelines automates the deployment process.

Technology Stack and Communication Patterns .net MAUI, python, C# , Ruby -languages REST-APIs, RabbitMQ - notifications

Design Data Management

User authentication:

- JSON Request to Register a New User:

```
{
"username": "john_doe",
"email": "john.doe@example.com",
"password": "secure_password123"
}
```

- JSON Response for User Registration:

```
{
"status": "success",
"message": "User registered successfully"
}
```

- JSON Request to Authenticate a User (Login):

```
{
"username": "john_doe",
"password": "secure_password123"
}
```

- JSON Response for User Authentication (Login):

```
{
"status": "success",
"message": "User authenticated successfully",
"user_id": "12345",
"access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Fitness Tracking Module:

- JSON Request to Log a Workout:

```
{
"user_id": "12345",
"activity": "Running",
"distance_km": 5.0,
"duration_minutes": 30,
"date": "2023-09-27"
}
```

- JSON Response for Workout Logging:

```
{
"status": "success",
"message": "Workout logged successfully",
"workout_id": "67890"
}
```

University Management Module:

- JSON Request to Enroll in a Course:

```
{
"user_id": "12345",
"course_id": "CS101",
"semester": "Fall 2023"
}
```

- JSON Response for Course Enrollment:

```
{
"status": "success",
"message": "Enrolled in CS101 for Fall 2023"
}
```

Personal Mentoring Module:

- JSON Request to Schedule a Mentorship Session:

```
{
"user_id": "12345",
"mentor_id": "67890",
"session_datetime": "2023-10-05T15:00:00",
"topic": "Career Guidance"
}
```

- JSON Response for Mentorship Session Scheduling:

```
{
"status": "success",
"message": "Mentorship session scheduled for 2023-10-05 at 3:00 PM"
}
```

Notifications Module (using RabbitMQ):

- JSON Request to Send a Notification:

```
{
  "user_id": "12345",
  "message": "Don't forget your upcoming workout session tomorrow at 8:00 AM."
}
```

- JSON Response for Notification Request:

```
{
"status": "success",
"message": "Notification sent successfully"
}
```

```
syntax = "proto3";

message FitnessActivityNotification {
  string user_id = 1;
  string activity_id = 2;
  string message = 3;
  repeated string channels = 4;
}

message NotificationResponse {
  bool success = 1;
```

```
  string message = 2;
}

service NotificationService {
  rpc SendFitnessActivityNotification(FitnessActivityNotification) returns
(NotificationResponse);
}
```