1. Overview

The goal of this project is to develop a system that retrieves data from GitHub repositories and presents it to the user through a web interface. The system has three main components:

1. **Data Collection Module**: Fetches repository data from GitHub using the GitHub API and saves it to the database.
2. **Microservice**: Provides APIs to interact with the repository data.
3. **Web Application**: Displays the repository data to the user with browse and search functionalities.

2. Data Design

GitHub Repository Data

The following fields will be retrieved from GitHub:

- **ID**: Unique identifier for the repository.
- **Name**: Name of the repository
- **Stars**: Number of stars the repository has.
- **Description**:description of the repository.
- **Forks**: Number of forks the repository has.
- **Languages**: Programming languages used in the repository.
- **Number of Forks**: Same as Forks.
- **Topics**: Topics associated with the repository.

Database Schema

The data will be stored in a MongoDB database. The schema for the repository documents will look like this:

```
{
  "repoId": Number,// Unique
  "name": String,
  "stargazers_count": Number,
  "description": String,
  "forks": Number,
  "languages": [String],
  "forks_count": Number,
  "topics": [String],
}
```

3. Process Design

a. Data Collection Module

- **Framework**: Node.js with Express.

- **Purpose**: Retrieve the data of a specific repository using the GitHub API and return it with an API.

- **Implementation**: The module will make an API request to the GitHub REST API search endpoint with the repository name provided. Using the repository name and owner it will make another API request to the repository languages endpoint to fetch all the necessary data .

- **Data Transformation**: After fetching both the repository and the repository languages, Data is mapped to an object to match the appropriate schema.

- **APIs**:
GET /api/collect?name=:name: Starts data collection by repository name, and returns a specific repository.

- **Error Handling**:
- If trying to create a repository that already exists (has the same repoId) it will return the repository and not create it in the database.

- If a repository from the GitHub API is not found it will return a 404 with "Repository not found" message

```
{
    "message": "Repository not found"
}
```

• If there is an internal error it will return a 500 with the appropriate error message

```
{
    "message": "example"
}
```

b. Microservice

• **Framework**: Node.js with Express.

• **Purpose**: Retrieve the data of repositories or a specific repository from the database, or retrieve the data from the Data Collection layer .

• **APIs**:
GET /api/repositories?page=:page&perPage=:perPage: Returns all repositories from the database with limited fields (repold, name, description). Pagination is optional; if page and perPage are not passed, all repositories will be returned.

GET /api/repository?id=:id | /api/repository?name=:name: Searches for a repository either by ID or name. If the repository by name is not found in the database, the Data Collection layer is triggered. Searching by ID takes priority over name if both are searched.

• **Error Handling**:
• If a repository by name is not found, fetch it using the Data Collection Module.

• If either endpoint doesn't fetch a repository it will return a 404 with a "Repository not found" message

```
{
    "message": "Repository not found"
}
```

- If there is an internal error it will return a 500 with the appropriate error message

```
{
    "message": "example"
}
```

Web Application

- **Framework**: Angular 18.
- **CSS Framework**: Tailwind.
- **Component Library**: DaisyUI.
- **Features**:
    1. **Browse**: Front page will list all repositories with limited details. When a repository is clicked, a modal will open with the full repository information shown.
    2. **Find**: A search functionality in the header where users can enter a repository name. The repository will be displayed in the front page if found.
- **Data Handling**: All data will be retrieved from the Microservice APIs.

4. Functional Design

User Workflow

1. The user opens the web application and can browse all repositories or search for a specific repository by name.

2. When a repository is selected, a modal will open with the full repository data displayed.

3. The user can search for a specific repository, If the repository is not in the database, the system will fetch the data from GitHub and display it to the user.

Error Scenarios

• **Repositories Not Found** | **Repository Not Found (ID)**: A message will be displayed to the user indicating that the repositories or repository was not found.

• **Repository Not Found (Name)**: If the repository is not in the database, the system will fetch it from GitHub, store it, and then display it.

5. Tools & Technologies

- **Backend**: Node.js with Express, mongoose and MongoDB Client, GitHub REST API.
- **Frontend**: Angular 18, Tailwind, daisyUI.
- **Database**: MongoDB.
- **Other Tools**: Git for version control.