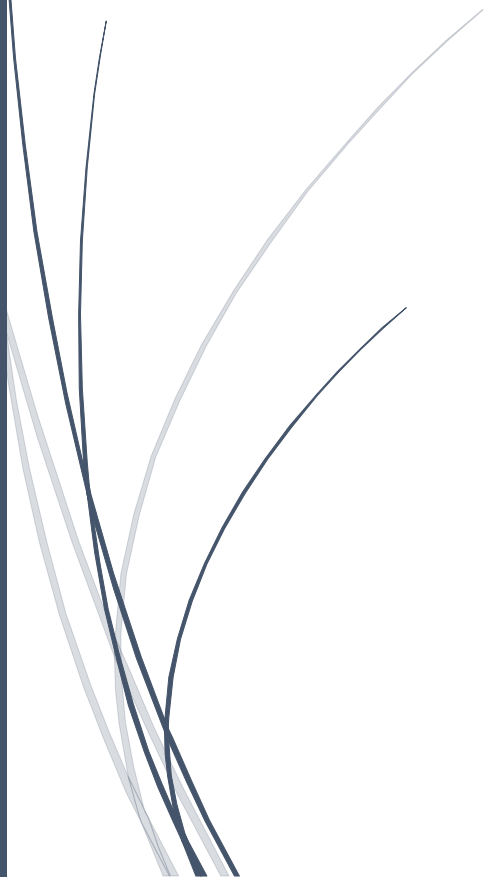
2025

# Project Rubrics

## Web Development Course



# Table of Contents

# Documentation

Documentation is crucial component to ensure clear communication, facilitate development, support maintenance, and guide users. Here are some essential documentations required:

### *Project Overview Document*

- Introduction to the project, its objectives, and scope.
- Entities involved and their roles.

### *Functional Requirements Document (FRD)*

- Detailed description of the system's functionalities from a user perspective. (Two functions)

### *Technical Requirements Document (TRD)*

- Detailed technical specifications covering architecture, components, and integrations.
- Infrastructure requirements, including server configurations, databases, and third-party services.
- Development tools, frameworks, and libraries used.
- For designing architecture:
  - Mermaid for creating diagrams and visualizations using text and code
  - Draw.io diagramming tool

### *Database Schema Documentation*

- Description of the database schema, including tables, columns, relationships, and constraints.

### *API Documentation*

- Documentation for integration with external systems about internal APIs used within the application, including endpoints, request/response formats, and authentication mechanisms.
- Sample requests and responses for common API operations.
- Usage guidelines and best practices for developers consuming the APIs.

API Documentation Template Example:

# API Template

## Description

This is an api to fetch books

## Base URL

The base URL for all API requests is:

`https://example-library-api.com`

## Endpoints

`GET /books`

Returns a list of all books in the library.

### Parameters

- `limit` (optional): The maximum number of books to return. Default is 10.
- `offset` (optional): The number of books to skip before starting to return results. Default is 0.

### Response

Returns a JSON object with the following properties:

- `count`: The total number of books in the library.
- `results`: An array of book objects, each with the following properties:
  - `id`: The unique identifier of the book.
  - `title`: The title of the book.
  - `author`: The author of the book.
  - `description`: A brief description of the book.
  - `publication_date`: The publication date of the book.

### Example

Request:

```
GET /books?limit=5&offset=10
```

Response:

JSON Copy

```
{
    "count": 50,
    "results": [
        {
            "id": 11,
            "title": "The Great Gatsby",
            "author": "F. Scott Fitzgerald",
            "description": "A novel about the decadent excesses of the Jazz Age.",
            "publication_date": "1925-04-10"
        },
        {
            "id": 12,
            "title": "To Kill a Mockingbird",
            "author": "Harper Lee",
            "description": "A novel about racial injustice in the American South.",
            "publication_date": "1960-07-11"
        },
        ...
    ]
}
```

## Errors

This API uses the following error codes:

- `400 Bad Request`: The request was malformed or missing required parameters.
- `401 Unauthorized`: The API key provided was invalid or missing.
- `404 Not Found`: The requested resource was not found.
- `500 Internal Server Error`: An unexpected error occurred on the server.

# Business Requirements

- User Registration and Access Control: Provide users with the ability to create accounts, authenticate securely, and manage access permissions within the system.
- Data Management: Ability to create, update, retrieve, and delete records or data entities with associated attributes.
- Collection and Workflow Assembly: Support user-driven aggregation of records, tasks, assets, or services into actionable collections or workflows, enabling process orchestration.
- Process Execution: Facilitate a smooth and secure execution of core processes, supporting multiple operational or transactional options as appropriate.
- Request and Workflow Management: Provide capabilities to monitor, update, and manage request lifecycles, status changes, notifications, and exception handling.
- Asset and Resource Optimization: Monitor, manage, and optimize availability and utilization of assets, inventory, human resources, or service capacities, with proactive threshold alerts and predictive analytics.
- Logistics and Calculation Services: Accurately compute operational costs, logistics, taxes, or service charges based on system rules and user-specific contexts.
- Responsive and Adaptive Interface: Design the user interface to function effectively across multiple device types and screen sizes.
- Search, Query, and Filtering Tools: Provide mechanisms for users to perform searches, apply dynamic filters, and retrieve relevant data efficiently.
- User Feedback and Evaluation: Allow users to submit feedback, ratings, or evaluations of system entities to inform and guide other users.

# Technical Requirements

## Common

### *Setup and Architecture*

- Set up a project structure that promotes scalability; to move to an enterprise-level solution in the future.
    - All tests should be contained in their own folder.
    - Separate modules are created for any processing.
- Set up a npm project.
    - package.json should contain both devDependencies, and dependencies.
    - Scripts should be created for testing, linting/prettier and starting the server.
    - Build script should run without error.
- Version Control and Collaboration:
    - Manage project using a version control system like Git.
    - Organize commits frequent, descriptive, and properly.
    - Share project with teammates and manage contributions using branching and pull requests.

### *Functionality*

- Set up JWT tokens in your API using modern authentication methods.
    - JWTs should be:
        - Part of the HTTP response.
        - Validated on requests requiring JWT (user secure routes).

        - Generated for each user.

### *README & Requirements Documentation*

- Create a README.md file covering installation instructions, usage guidelines, and any other relevant information for developers and users.
- README file Helper Links:
    - Syntax: https://www.mygreatlearning.com/blog/readme-file/
    - Editor: https://stackedit.io/app#

## Backend

### *Functionality*

- Add and use Express to a node.js project.
    - Start script should run without error.
    - Provided endpoint should open in the browser with status 200.
- Configure middlewares.
- Database
    - Create a database and connect to it.
    - Secure important information by adding salt to user passwords.
        - Encrypt the password field on the user table using the **bcrypt** library.
    - Create CRUD endpoints for models in the application.
        - Split the routes into grouped handler files for better code organization.

## Front-end

### *Functionality*

- Dashboard:
    - Create a dashboard page where authenticated users can view basic information or perform actions related to the app's purpose.
- Profile Management:
    - Allow users to view and edit their profile information.
    - Include basic fields like name, email, and profile picture.
- Component Architecture:
    - Project must follow a component-based architecture.
    - Components are appropriately modularized and reusable.
    - Components must be structured logically, with clear separation of concerns.
- Responsive Design:
    - Ensure the app layout is responsive and works well on various screen sizes, including mobile devices.
    - Use CSS frameworks like Material-UI or Bootstrap to facilitate responsiveness.
- Routing
    - Implement client-side routing by building a SPA with React Router.
    - Routes are defined logically, with appropriate handling of nested routes and redirects.
- Forms Handling
    - Work with Forms and user inputs with **Formik**.
    - Validate form’s inputs properly, with error messages displayed as needed.

- State Management:
  - Utilize React's built-in state management for managing local component state.
- Error Handling:
  - Implement basic error handling to display informative error messages to users when something goes wrong.
  - Handle common error scenarios like network failures, invalid inputs, or server errors gracefully.

# [Bonuses]

## Documentation

### *API Documentation*

- Create an API documentation using **swagger**.
- Create a collection for the backend API on **Postman**.

## Backend

### *Functionality*

- Database
  - Secure database access info with environment variables
    - You can use **dotenv** to create environment variables.

### *Code Quality*

- Write relevant unit tests to improve code quality and refactoring.
  - Test script runs and all tests created pass.
  - There is at least 1 test per endpoint and at least one test for image processing.