Test Strategy for Timetable Generator Project

1. Purpose

This document outlines a basic test approach for the Timetable Generator Blazor app. The goal is to ensure all features work as expected, the user experience is smooth, and the application is secure and reliable.

2. Testing Objectives

- Ensure that users can successfully create courses, tasks, timetables, etc.
- Verify admin tools (e.g., user management, term management) work as intended.
- Validate key user interactions, such as drag-and-drop for timetables.
- Confirm the app works on multiple browsers and devices.
- Ensure external authentication (Google login) is functioning properly.

3. Types of Testing

3.1 Functional Testing

Check that all features of the application work as expected.

- Test course, task, task-type creation.
- Verify timetable generation and drag-and-drop functionality.
- Ensure form validation (e.g., required fields, dropdown selections).

3.2 Unit Testing

Write tests to check the core logic of smaller pieces of code (e.g., service classes).

- Use **xUnit** for testing backend logic.
- Focus on CRUD operations for MongoDB.

3.3 Integration Testing

Ensure different components of the system work together.

- Test database operations with MongoDB.

- Verify Google login integration.

3.4 UI Testing

Ensure the app works from the user's point of view.

- Test navigation, form submissions, and modal functionality.
- Validate the drag-and-drop feature for tasks in timetables.

3.5 Cross-Browser and Device Testing

Ensure the app looks and functions well across different browsers (Chrome, Firefox, Edge) and on both desktop and mobile devices.

3.6 Security Testing

Check that users can't access admin tools without permission and that data is protected.

4. Testing Tools

- **Unit Testing**: xUnit
- **UI Testing**: Manual checks.
- **Cross-Browser Testing**: Manual testing on major browsers and devices.

5. Test Cases

Example Test Case: Course Creation

- **Objective**: Ensure users can create a course.
- **Steps**:
    1. Open the course creation page.
    2. Fill in the form fields.
    3. Click "Save."
    4. Verify the course is saved and the preview modal appears.
- **Expected Result**: The course is successfully saved and displayed in the preview.

Example Test Case: MongoDB Course Insertion

- **Objective**: Ensure that the MongoDB CreateCourseAsync function correctly inserts a course into the database.
- **Steps**:
    1. Set up a test instance of MongoDB (or use an in-memory database).
    2. Initialize the MongoCourseData service class.

3. Create a sample Course object with valid data (e.g., course name, term, etc.).
4. Call the CreateCourseAsync method with the sample course.
5. Query the database to verify the course was inserted correctly.
6. Optionally, clean up the test database by removing the inserted course.

- **Expected Result**: The course is successfully inserted into the MongoDB collection, and the retrieved course matches the inserted data.

6. Responsibilities

- **Developers**: Write unit tests and ensure all features are implemented correctly.
- **QA/Testers AKA Chris and Joah**: Perform manual and UI tests to check functionality, look for bugs, and verify everything works across different browsers.