Capsitech
IT Solution for the Businesses

Web Development Training Curriculum

www.capsitech.com

Table of Contents:

Module 1: Static UI Development

1. HTML Basics

- Structure of an HTML document.
- Common elements: headings, paragraphs, links, images, lists.
- References:
  - W3Schools HTML Tutorial
  - freeCodeCamp: Learn HTML

2. CSS Basics

- Selectors, properties, and values.
- Styling text and elements: colors, fonts, spacing, and borders.
- Box model concepts.
- References:
  - W3Schools CSS Tutorial
  - MDN Web Docs: CSS Basics

3. Advanced HTML & CSS

- Semantic HTML: <header>, <footer>, <section>, <article>.
- Layout techniques: Flexbox and Grid.
- Positioning elements: static, relative, absolute, fixed, sticky.
- References:
  - MDN Web Docs: HTML Elements
  - CSS-Tricks: A Complete Guide to Flexbox
  - CSS-Tricks: A Complete Guide to Grid
  - MDN Web Docs: CSS Positioning

4. Responsive Design

- Principles of responsive design.
- Media queries and relative units (%, em, vh).
- References:
  - MDN Web Docs: Responsive Design
  - freeCodeCamp: Responsive Web Design Principles

5. CSS Frameworks: Tailwind CSS

- Introduction to utility-first CSS.
- Configuring Tailwind for a project.

- Common utility classes: typography, layout, spacing, and colors.
- Building components like buttons, cards, and navigation bars.
- References:
  - Tailwind CSS Documentation
  - freeCodeCamp: Tailwind CSS Crash Course

6. CSS Frameworks: Bootstrap
   - Overview of Bootstrap's grid system.
   - Using prebuilt components: modals, carousels, forms, and buttons.
   - Customizing themes using Bootstrap variables.
   - References:
     - Bootstrap Documentation
     - freeCodeCamp: Bootstrap 5 Tutorial
7. Animations and Interactivity
   - CSS transitions and animations.
   - Pseudo-classes (:hover, :focus) and pseudo-elements (::before, ::after).
   - Adding interactivity with Bootstrap and Tailwind utilities.
   - References:
     - MDN Web Docs: CSS Transitions
     - CSS-Tricks: A Complete Guide to Hover Effects
     - Tailwind CSS Animation Utilities
8. Figma for Developers
   - Understanding design files in Figma.
   - Extracting assets and inspecting design details.
   - Translating Figma designs into HTML/CSS code.
   - References:
     - Dev: Figma for Beginners

Task for Module 1

Build a Complete Responsive Website

Objective:

Create a responsive website that adheres to modern design and development standards using HTML, CSS, and a CSS framework of your choice (Tailwind CSS or Bootstrap).

Instructions:

- Design link will be provided by your mentor.
- Implement clean and well-structured code.
- Utilize the provided assets and design guidelines.
- Use Tailwind CSS or Bootstrap to style and layout the website.
- Ensure responsiveness across various screen sizes using appropriate grid and flex utilities from the chosen framework.
- Add interactivity and animations such as hover effects and transitions to enhance user experience.

Deliverables:

1. Comprehensive documentation detailing the following:

a. Design choices and considerations.

b. Frameworks and tools used.

c. Key features and implementation details.

Module 2: JavaScript

1. Introduction to JavaScript
    - What is JavaScript, and how to execute JavaScript code.
    - References:
        - MDN Web Docs: Introduction to JavaScript
2. Variables
    - let, const, var: Differences in behavior and scope.
    - References:
        - Var vs Const vs Let (ES6) - Beau teaches JavaScript
        - MDN Web Docs: var, let, and const
3. Comparisons
    - Different types of comparisons: ==, ===, Object.is.
    - References:
        - MDN Web Docs: Equality Comparisons
4. Data Types
    - Value types vs. reference types.
    - References:
        - JavaScript.info: Data Types
5. Functions
    - Writing and understanding functions in JavaScript.
    - References:
        - MDN Web Docs: Functions
6. Fetch API
    - Introduction to the fetch API for making network requests.
    - References:
        - MDN Web Docs: Fetch API
7. Local Storage
    - Storing and retrieving data using localStorage.
    - References:
        - MDN Web Docs: Window.localStorage
8. DOM Manipulation
    - Basic JavaScript DOM manipulation.

- References:
  - MDN Web Docs: DOM Manipulation

9. Form Handling
   - Form submission and error handling with JavaScript.
   - References:
     - JavaScript.info: Forms
10. Task: Contact Form
    - Create a contact form with HTML, CSS, and JavaScript.
    - Apply proper validation for each field using JavaScript.
11. Arrays
    - JavaScript arrays and array helper methods like map, forEach, filter.
    - References:
      - MDN Web Docs: Array Methods
12. Objects
    - JavaScript objects and helper methods like Object.keys, Object.values, and Object.entries.
    - References:
      - MDN Web Docs: Working with Objects
13. Spread and Rest Operators
    - Using spread operators to create shallow copies of objects and arrays.
    - Understanding rest operators.
    - References:
      - JavaScript.info: Rest and Spread
14. Destructuring
    - Array and object destructuring.
    - References:
      - Destructuring in ES6 - Beau teaches JavaScript
      - MDN Web Docs: Destructuring Assignment
15. Functions as Variables
    - Treating functions like normal variables and using callbacks.
    - References:
      - JavaScript.info: Functions

16. Timers

- Using setTimeout and setInterval.
- References:
  - MDN Web Docs: Timers

17. Promises and Async/Await

- Understanding promises, then, catch.
- Using async/await and handling errors with try-catch blocks.
- References:
  - MDN Web Docs: Promises
  - JavaScript.info: Async/Await

Task for Module 2

Build a Complete Contact Form

Objective: Create a fully functional contact form that adheres to modern JavaScript standards, including validation, interactivity, and proper error handling.

Instructions:

1. Use HTML, CSS, and JavaScript to create a contact form.
2. Include validation for fields such as name, email, and message.
3. Leverage JavaScript DOM manipulation to dynamically show error messages.
4. Implement a feature to simulate form submission and success feedback using promises.
5. Style the form using modern CSS techniques or frameworks like Tailwind CSS or Bootstrap.
6. Ensure responsiveness for various screen sizes.

Deliverables:

- Comprehensive documentation including:
  - Validation and interactivity features.
  - Frameworks and tools used.
  - Code structure and comments for better readability.

Module 3: React

1. React Introduction

- Overview of React and its features.
- Differences between direct JS DOM manipulation and React's virtual DOM.
  - Imperative vs. Declarative Programming: Understand the conceptual differences.
- References:
  - React Official Documentation
  - MDN Web Docs: Declarative vs Imperative Programming

2. Components

- Functional components and their structure.
- Props: Passing data to components and understanding component composition.

3. State in React

- Managing state with the useState hook.
- Understanding state updates and their impact on components.
- When and why a component re-renders.
- References:

State in React - Beau teaches JavaScript

4. Event Handlers

- Adding interactivity with event handlers in React.
- Passing data and handling events in child components.
- References:

freeCodeCamp: React Event Handlers

5. The useEffect Hook

- Lifecycle management with useEffect.
- Side effects like fetching data and managing subscriptions.
- Integrating a third-party API using useEffect and Axios.
- References:
  - React: Using the Effect Hook
  - Axios GitHub Documentation

6. Form Handling

- Managing state and submitting data from forms in React.
- Handling complex forms using Formik.

- References:
  - React: Controlled Components
  - Formik Documentation

Reference:

Youtube reference crashcourse

Tasks for Module 3

Task 1: Implement a Student Registration Form

Objective: Build a student registration form using React and Formik.

Instructions:

1. Set up a React project with Formik for form management.
2. Create input fields for:

a. Name

b. Age

c. Email

d. Course selection

3. Add proper validation using Formik.
4. Submit the form data and log it to the console.

Deliverables:

- A functional student registration form with validation.

References:

- Formik Tutorial

Task 2: Create a Weather App

Objective: Build a weather application using React and a third-party API.

Instructions:

1. Set up a React project and install Axios.
2. Use the OpenWeatherMap API (or any weather API) to fetch weather data.
3. Create a search bar to input a city name.
4. Display weather details like temperature, humidity, and conditions.
5. Use the useEffect hook to handle API calls.

Deliverables:

- A fully functional weather app with responsive design.

References:

- OpenWeatherMap API Documentation
- React Weather App Tutorial

Module 4: Backend Basics

1. Introduction to Backend Development

- What is Backend Development?
  - Overview of backend vs frontend development.
  - The role of backend in applications: managing databases, server logic, and communication with external services.
  - Examples of backend programming languages (Node.js, Python, Ruby, PHP, Java, etc.).
- Why is Backend Development Important?
  - Server-side operations (business logic, user authentication, data processing).
  - Interaction with databases and APIs.
  - Security considerations (data encryption, authorization).
- YouTube Resource:
  - Backend web development - a complete overview
- References:
  - freeCodeCamp: Backend Development

2. REST APIs (Representational State Transfer)

- What are REST APIs?
  - Understanding the concept of RESTful services and their role in backend development.
  - REST principles: statelessness, client-server architecture, uniform interface, cacheability.
- Anatomy of a REST API Request & Response
  - Request: HTTP methods (GET, POST, PUT, DELETE, PATCH), request headers, query parameters, body content.
  - Response: Status codes (200, 400, 404, 500), response headers, body content (JSON, XML, etc.).
- HTTP Methods
  - Overview and practical use cases for GET, POST, PUT, DELETE, PATCH methods.
- YouTube Resource:
  - Understanding RESTful APIs in 8 Minutes
- References:
  - freeCodeCamp: REST APIs
  - MDN Web Docs: HTTP Methods

3. Request Methods, Query Parameters, Headers, and Body

- Request Methods
  - Detailed understanding of GET, POST, PUT, DELETE, PATCH.
  - How to choose the correct method for different operations.
- Query Parameters
  - Understanding URL query parameters and their usage in GET requests (e.g., ?search=abc).
- Headers
  - Common HTTP headers: Content-Type, Accept, Authorization, etc.
  - Custom headers for security and validation.
- Request Body Types
  - Form Data: Sending form data (application/x-www-form-urlencoded).
  - JSON: Sending JSON data (application/json).
  - XML: Sending XML data (application/xml).
- YouTube Resource:
  - Understanding HTTP Request Methods - Explained in Simple Terms
- References:
  - MDN Web Docs: HTTP Headers
  - freeCodeCamp: Understanding REST APIs

4. Authentication and Authorization

- What is Authentication?
  - Understanding the process of verifying a user's identity (e.g., login with username and password).
  - Methods: Session-based authentication, token-based authentication (JWT).
- What is Authorization?
  - Understanding the process of granting access based on user roles (e.g., admin, user).
  - Role-based access control (RBAC).
- Authentication Mechanisms
  - Basic Authentication: Sending username and password in request headers.
  - JWT (JSON Web Tokens): Secure authentication using tokens.
  - OAuth2: Delegated authorization for third-party access.
- Securing APIs

- Introduction to OAuth2, JWT, and API keys for securing endpoints.
- YouTube Resource:
- JWT Authentication Tutorial
- References:
- freeCodeCamp: Introduction to JWT
- MDN Web Docs: HTTP Authentication

5. Communicating with Third-Party APIs using Postman

- Introduction to Postman
- Overview of Postman as a tool for testing and interacting with APIs.
- Setting up Postman for API requests and exploring response data.
- Sending Requests via Postman
- Sending GET, POST, PUT, DELETE requests.
- Using headers, query parameters, and request body data.
- Testing Authentication
- Sending authentication tokens (e.g., JWT, API keys) in headers.
- Handling API Responses
- Understanding and analyzing status codes, response body, and headers.
- YouTube Resource:
- Postman Tutorial for Beginners
- References:
- Postman Documentation
- freeCodeCamp: Postman Tutorial

6. Databases and Types of Databases

- What are Databases?
- Understanding the purpose of a database: storing, retrieving, and managing data.
- SQL Databases (Relational Databases)
- Key concepts: tables, rows, columns, primary keys, foreign keys.
- Introduction to SQL (Structured Query Language).
- Examples: MySQL, PostgreSQL, SQLite.

- NoSQL Databases (Non-relational Databases)
  - Key concepts: documents, collections, key-value pairs, wide-column stores.
  - When to use NoSQL databases.
  - Examples: MongoDB, Redis, CouchDB, Cassandra.
- SQL vs NoSQL
  - Differences between relational and non-relational databases.
  - When to use SQL or NoSQL based on application needs.
- YouTube Resource:
  - Learn SQL for Beginners
  - MongoDB Crash Course | MongoDB Tutorial For Beginners
- References:
  - freeCodeCamp: Introduction to SQL
  - MongoDB Documentation

7. Basics of MongoDB and CRUD Operations

- Introduction to MongoDB
  - Overview of MongoDB: document-oriented database.
  - Structure: documents (JSON-like) and collections.
- CRUD Operations in MongoDB
  - Create: Inserting documents.
  - Read: Querying documents.
  - Update: Modifying documents.
  - Delete: Removing documents.
- Using MongoDB Compass
  - Connecting to MongoDB via MongoDB Compass.
  - Performing CRUD operations using the Compass UI.
- Basic Queries in MongoDB
  - Querying documents with filters.
  - Using operators: $gt, $lt, $in, $or, etc.
- YouTube Resource:
  - MongoDB Crash Course
- References:

- MongoDB Documentation
- freeCodeCamp: MongoDB Crash Course

8. MongoDB Aggregation Framework

- What is MongoDB Aggregation?
  - Introduction to MongoDB's aggregation framework.
  - Importance of aggregation for complex queries and data transformations.
- Aggregation Stages
  - $match: Filtering documents based on conditions.
  - $project: Reshaping documents by including or excluding fields.
  - $lookup: Joining collections.
  - $unwind: Deconstructing arrays into individual documents.
  - $group: Grouping documents based on fields.
- Complex Aggregations
  - Combining multiple stages for more advanced queries.
  - Using $sort, $limit, and $skip in the aggregation pipeline.
- YouTube Resource:
  - MongoDB Aggregation Tutorial
- References:
  - MongoDB Aggregation Docs
  - freeCodeCamp: MongoDB Aggregation Pipeline

9. MVC Architecture

- What is MVC?
  - Overview of the Model-View-Controller (MVC) design pattern.
  - Why MVC is used in backend development.
- MVC Breakdown
  - Model: Data representation and database interaction.
  - View: User interface (handled by frontend, but understand the interaction).
  - Controller: Business logic and interaction between Model and View.
- Implementing MVC in Backend Development

- Structuring backend applications with MVC.
- Routing and handling HTTP requests in controllers.
- Frameworks Supporting MVC
- Examples: Express.js (Node.js), Django (Python), Ruby on Rails (Ruby).
- References:
- Express.js MVC Guide

Task for Backend Basics

Build a Simple API

Objective:

Create a simple backend application using REST API principles, MongoDB for data storage, and implement basic authentication.

Instructions:

- Implement a basic API with endpoints (e.g., GET, POST, PUT, DELETE).
- Use MongoDB to store and manage data (perform CRUD operations).
- Implement JWT-based authentication for secure access to the API.
- Document your API endpoints with details (request body, parameters, response).

Deliverables:

- API documentation detailing endpoints, request/response format, authentication, and security measures.

Module 5: Express JS

1. Introduction to Express.js

- Objective: Gain a basic understanding of Express and how to set up a simple Express server.
- Topics Covered:
  - What is Express.js? Why is it used for backend API development?
  - Setting up an Express project with Node.js.
  - Basic Express server setup and handling requests.
- Key Concepts:
  - Installing Express via npm.
  - Creating a simple Express application.
  - Handling basic HTTP requests using GET and POST.
- YouTube Resource:
  - Express.js Crash Course
- References:
  - Express.js Documentation

2. Routing in Express.js

- Objective: Learn how to define and manage routes for handling different types of requests in Express.
- Topics Covered:
  - What is routing in Express?
  - Defining routes for HTTP methods (GET, POST, PUT, DELETE).
  - Route parameters and handling dynamic data in routes.
  - Creating basic routes for an API.
- Key Concepts:
  - Routing methods (app.get(), app.post(), etc.).
  - Query parameters using req.query.
  - Path parameters using req.params.

- YouTube Resource:
  - Express Routing Explained

3. Parsing Data from Request Query Params, Body, Headers, and URL Path Params

- Objective: Understand how to parse data from the various parts of the request.
- Topics Covered:
  - Query Parameters: Using req.query to get query parameters from the URL.
  - Body Parameters: Using express.json() middleware to parse JSON data in the body of POST requests.
  - Headers: Accessing request headers using req.headers.
  - URL Path Parameters: Accessing dynamic route parameters with req.params.
- Key Concepts:
  - Middleware for parsing request bodies (express.json(), express.urlencoded()).
  - Accessing headers and parameters.
- YouTube Resource:
  - Express.js: Accessing Data from Request
- References:
  - Express.js Documentation on Request Object

4. Sending JSON Response from an API

- Objective: Learn how to send JSON responses from an Express API.
- Topics Covered:
  - Using res.json() to send JSON data in the response.
  - Setting HTTP status codes with res.status().
  - Responding with different HTTP methods.
- Key Concepts:
  - Sending JSON responses.
  - Proper HTTP status codes for success and error responses.
- YouTube Resource:
  - How to Send JSON Response in Express

5. Creating a Basic CRUD API for Todos

- Objective: Implement a simple CRUD (Create, Read, Update, Delete) API for managing "todos".
- Topics Covered:
  - Create: POST request to add a new todo.
  - Read: GET request to retrieve todos.
  - Update: PUT/PATCH request to update an existing todo.
  - Delete: DELETE request to remove a todo.
  - Storing todos in an in-memory array (for simplicity) or using a database like MongoDB.
- Key Concepts:
  - Defining routes for CRUD operations.
  - Storing data in memory or using a database.
  - Handling form data or JSON payloads.
- YouTube Resource:
  - Building a Simple CRUD API with Express
- References:
  - Express.js CRUD API Example

6. Middleware in Express.js

- Objective: Learn how middleware works in Express and how to use it for additional functionality.
- Topics Covered:
  - What are middlewares? How do they fit into the request-response cycle?
  - Built-in middleware (e.g., express.json(), express.static()).
  - Writing custom middleware functions for logging, authentication, etc.
- Key Concepts:
  - How middleware functions process requests.
  - Order of middleware execution.
  - Writing reusable middleware functions.

- YouTube Resource:
  - Express.js Middleware Tutorial
- References:
  - Express.js Middleware Documentation

7. Creating a Validation Middleware

- Objective: Learn how to create and use a middleware for validating incoming data before it is processed.
- Topics Covered:
  - Zod: A TypeScript-first schema declaration and validation library.
  - node-input-validator: A data validation library for Node.js applications.
  - Using these libraries to validate request body data (e.g., validating a todo before creating it).
- Key Concepts:
  - Defining validation schemas.
  - Integrating validation middleware into routes.
- YouTube Resource:
  - Zod Validation Tutorial
  - Node Input Validator Tutorial
- References:
  - Zod Documentation
  - node-input-validator Documentation

Task for Backend API with Express.js

Objective:

Build a complete backend API for managing a "To-Do" list application with Express, which should support CRUD operations. Implement validation middleware using either Zod or node-input-validator to ensure correct data structure for each request.

Instructions:

1. Set up an Express server.
2. Implement routes for creating, reading, updating, and deleting todos.
3. Add validation middleware for creating and updating todos.
4. Test the API using Postman or any API client.

Deliverables:

1. The complete Express application with the required CRUD functionality.
2. Documentation for the API routes and their expected request/response formats.
3. A short explanation of the validation middleware used and why it's important.

Module 6: C# Basics and Core Concepts

1. Introduction to C# and OOP Concepts

Objective: Understand the fundamentals of C# and Object-Oriented Programming (OOP).

Topics:

- C# syntax and structure:
  - Reference: C# Basics - Microsoft Docs
- Core OOP principles: Encapsulation, inheritance, polymorphism, abstraction.
- Classes, objects, interfaces, and abstract classes:
  - Reference: Classes and Objects in C# - Microsoft Docs

2. Data Types and Exception Handling

Objective: Learn about data types and handling runtime errors in C#.

Topics:

- Value types, reference types, nullable types:
  - Reference: Data Types in C# - Microsoft Docs
- Exception handling with try-catch-finally:
  - Reference: Exception Handling in C# - Microsoft Docs
- Custom exception creation:
  - Reference: How to Create Custom Exceptions - Microsoft Docs

3. Advanced Features in C#

Objective: Explore modern C# features.

Topics:

- Events, delegates, and lambda expressions:
  - Reference: Events in C# - Microsoft Docs
  - Reference: Lambda Expressions - Microsoft Docs
- Collections: Generic (List<T>, Dictionary<K,V>), LINQ:
  - Reference: Collections in C# - Microsoft Docs
  - Reference: LINQ Basics - Microsoft Docs
- Optional parameters, enums, indexers:
  - Reference: Optional Parameters in C# - Microsoft Docs
  - Reference: Enums in C# - Microsoft Docs

- Var vs dynamic:
  - Reference: Var and Dynamic in C# - GeeksforGeeks

4. MultiThreading and Async Programming

Objective: Gain foundational knowledge of concurrency.

Topics:

- Threads and thread safety:
  - Reference: Threading in C# - Microsoft Docs
- Task-based asynchronous programming with async/await:
  - Reference: Asynchronous Programming in C# - Microsoft Docs

Module 7: .NET Core for Web Development

1. Getting Started with .NET Core Web API

Objective: Learn the structure and setup of a .NET Core Web API project.

Topics:

- Creating a new project in Visual Studio:
  - Reference: Getting Started with ASP.NET Core Web API - Microsoft Docs
- Understanding project structure: Controllers, Models, Services, Startup.cs.
  - Reference: ASP.NET Core Project Structure - Microsoft Docs

Activity: Create a basic “Hello World” API.

2. Routing and Request Handling

Objective: Understand routing and data handling in requests.

Topics:

- Attribute and conventional routing:
  - Reference: Routing in ASP.NET Core - Microsoft Docs
- Parsing query parameters, request body, headers, and path parameters:
  - Reference: Model Binding in ASP.NET Core - Microsoft Docs

Activity: Build an API to handle different types of request data.

3. MongoDB Integration

Objective: Connect to MongoDB and perform CRUD operations.

Topics:

- Setting up MongoDB driver for .NET Core:
  - Reference: MongoDB Driver Documentation for .NET
- Performing CRUD operations with a MongoDB collection:
  - Reference: MongoDB CRUD Operations - MongoDB Docs

Activity: Create a database-backed CRUD API for task management.

4. Advanced API Features

Objective: Enhance API capabilities.

Topics:

- Dependency injection:
  - Reference: Dependency Injection in ASP.NET Core - Microsoft Docs
- Middleware (logging, error handling):
  - Reference: Middleware in ASP.NET Core - Microsoft Docs
- Filtering, sorting, and pagination:
  - Reference: Sorting and Filtering in ASP.NET Core Web API - TutorialsTeacher
- Request validation with libraries like FluentValidation:
  - Reference: FluentValidation Documentation

Task for Backend API with .NET Core

Objective:

Build a complete backend API for managing a "To-Do" list application with .NetCore, which should support CRUD operations.

Instructions:

1. Implement routes for creating, reading, updating, and deleting todos.
2. Test the API using Swagger or any API client.

Deliverables:

1. The complete .NET Core application with the required CRUD functionality.
2. Documentation for the API routes and their expected request/response formats.
3. A short explanation of the validation middleware used and why it's important.

Project: Task Management Application (Full Stack)

Objective:

Build a full-stack task management application using React for the frontend, Express.js/.NET Core for the backend API, and MongoDB for the database. The application will allow users to manage teams, projects, and tasks with various filters and pagination functionality.

Tasks

1. Team Management Module

- Task 1.1: Create an Express route for managing team members.
  - Route: /api/teams
  - Methods: GET, POST, PUT, DELETE
  - Data: Name, email, designation (validate these fields)
- Task 1.2: Set up the team model in MongoDB using Mongoose.
  - Fields: Name, email, designation
- Task 1.3: Create a React component for adding/editing team members.
  - Create a form with inputs for name, email, and designation.
- Task 1.4: Display the list of team members.
  - Show basic information (name, email, designation).
  - Add pagination for the list.

2. Project Management Module

- Task 2.1: Create an Express route for managing projects.
  - Route: /api/projects
  - Methods: GET, POST, PUT, DELETE
  - Data: Name, description, team members (validate these fields)
- Task 2.2: Set up the project model in MongoDB using Mongoose.
  - Fields: Name, description, team members (array of team member IDs)
- Task 2.3: Create a React component for adding/editing projects.
  - Create a form with fields for name, description, and team members (dropdown or multi-select).
- Task 2.4: Display a list of projects with their basic details.
  - Show name, description, and associated team members.
  - Add pagination for the project list.

3. Task Management Module

- Task 3.1: Create an Express route for managing tasks.
  - Route: /api/tasks
  - Methods: GET, POST, PUT, DELETE
  - Data: Title, description, deadline, project, assigned members, status (validate these fields)
- Task 3.2: Set up the task model in MongoDB using Mongoose.
  - Fields: Title, description, deadline, project (ID), assigned members (array of member IDs), status (to-do, in-progress, done, cancelled)
- Task 3.3: Create a React component for adding/editing tasks.
  - Create a form with fields for title, description, deadline, project, and assigned members.
- Task 3.4: Display the list of tasks with filters:
  - Filters: By project, member, status, title/description, and date range.
  - Add pagination for the task list.

4. Filtering and Pagination

- Task 4.1: Implement server-side filtering and pagination for tasks in the backend.
  - Filters: Use query parameters to filter tasks by project, member, status, search term, and date range.
  - Pagination: Use query parameters for pagination (page, limit).
- Task 4.2: Implement filtering and pagination in the frontend.
  - Add filter components (dropdowns, search bars, date pickers).
  - Add pagination controls to navigate between pages.

5. Validation and Data Integrity

- Task 5.1: Implement validation for all routes using libraries like Joi or Zod.
  - Validate the required fields (name, email, description, etc.) for teams, projects, and tasks.
- Task 5.2: Add error handling to the API to return appropriate error messages when validation fails.

6. Typesafe Code

- Task 6.1: Set up TypeScript for both the frontend and backend.
  - Configure TypeScript for React, Express, and Mongoose.
- Task 6.2: Write type-safe code in both frontend and backend.
  - Ensure correct type definitions for API responses, requests, and form inputs.

7. GitHub Repository

- Task 7.1: Set up a GitHub repository to manage the project code.
  - Use Git for version control and commit changes regularly.
- Task 7.2: Push the project to GitHub and document your repository.
  - Write a README.md file to explain the project setup and usage.

Deliverables

- **Backend**: Fully functional API with proper validation, filtering, pagination, and CRUD operations for teams, projects, and tasks.
- **Frontend**: A responsive React application with a UI for managing teams, projects, and tasks. It should include filters, pagination, and forms for CRUD operations.