Syntax School - Technical Design Document

Table of Contents

Introduction

1.1 Project Name

Syntax School

1.2 Description

Syntax School is a modern e-learning platform designed to facilitate computer science education. It enables instructors to create and manage courses with YouTube-hosted video lessons, quizzes, and supplementary text-based resources. Students can enroll in courses, access resources, and assess their progress through interactive quizzes, all within an intuitive and responsive interface.

1.3 Objectives

- Provide an engaging platform for learning computer science concepts.
- Enable instructors to seamlessly upload and manage educational content.
- Offer interactive features like quizzes to reinforce learning.
- Ensure a scalable and cost-effective system by leveraging free resources (e.g., YouTube for video hosting).

System Overview

2.1 Overview

Syntax School is an e-learning platform tailored to computer science education, leveraging the MERN stack (MongoDB, Express.js, React.js, Node.js). It incorporates modern web development practices to deliver a scalable, maintainable, and cost-effective solution for instructors and students.

2.2 Key Features

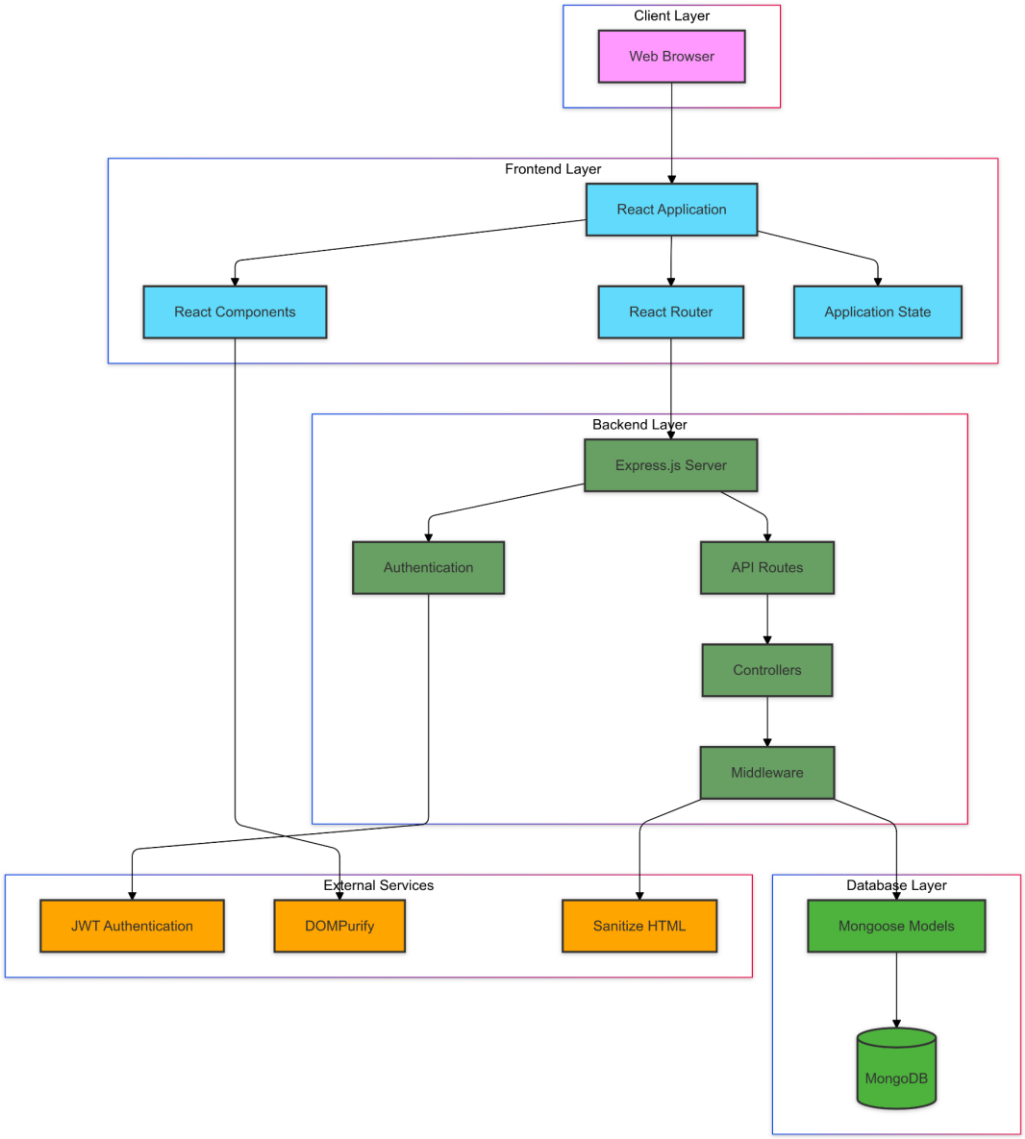
- **Role-Based System:**
  - **Instructors:** Can create, update, and manage courses with embedded YouTube videos, text resources, and quizzes.
  - **Students:** Can enroll in courses, access resources, and take quizzes to assess their learning.
- **Core Functionalities:**
  - **Authentication:** Secure login and registration with JWT.
  - **Course Management:** Support for video lessons, text resources, and quizzes.
  - **Progress Tracking:** Students can track completed lessons and quizzes.
- **Free Resource Usage:**
  - Videos are hosted on YouTube to eliminate hosting costs.
  - MongoDB Atlas for free-tier database hosting.

2.3 System Goals

- **Scalability:** Designed to support a growing number of users and courses.
- **Cost Efficiency:** Uses free services for video hosting and database management.
- **Responsiveness:** Optimized for various devices, ensuring accessibility for all users.

Architecture

High-Level Architecture



Syntax School follows a three-tier architecture to ensure scalability, maintainability, and separation of concerns:

1. **Frontend**:
    - Built with **React.js** to create a responsive and user-friendly interface.
    - Handles routing, state management, and interaction with the backend via APIs.
2. **Backend API**:
    - Developed with **Node.js** and **Express.js**.
    - Handles authentication, API endpoints, and business logic.
3. **Database**:
    - Utilizes **MongoDB** for storing user data, courses, quizzes, and progress tracking.

Component Interactions

- **Frontend → Backend API:**
The frontend communicates with the backend via RESTful APIs to retrieve and update data (e.g., courses, quizzes).

- **Backend → Database:**
The backend interacts with MongoDB for data storage and retrieval using Mongoose.

- **YouTube Integration:**
Embedded YouTube videos are directly streamed to the frontend via provided URLs.

Backend Design

4.1 Technologies Used

- Node.js: JavaScript runtime environment for server-side programming.
- Express.js: Framework for building RESTful APIs.
- MongoDB: NoSQL database for managing structured data.
- Mongoose: ODM (Object Data Modeling) library for MongoDB.
- JWT (JSON Web Tokens): For secure user authentication.
- bcrypt: Library for hashing user passwords.

Project Structure

```
backend/
├── index.js          # Entry point of the application
├── routes/           # API route handlers
│   ├── auth.js       # Authentication routes
│   ├── courses.js    # Course management routes
│   └── quizzes.js    # Quiz management routes
├── models/           # MongoDB schemas
│   ├── User.js       # User model
│   ├── Course.js     # Course model
│   └── Quiz.js       # Quiz model
├── middleware/       # Middleware for validation and authentication
│   └── auth.js       # JWT authentication middleware
└── config/           # Configuration files
    └── db.js         # MongoDB connection setup
```

4.3 API Design

Authentication Routes (/api/auth)

- **POST /register**: Register a new user.
- **POST /login**: Log in an existing user and return a JWT.

Course Routes (/api/courses)

- **GET /**: Retrieve all courses.
- **POST /**: Create a new course (Instructor only).

Quiz Routes (/api/quizzes)

- **GET /:courseId**: Retrieve quizzes for a course.
- **POST /**: Create a quiz for a course (Instructor only).

4.4 Database Schema

User Model (User.js)

```
{
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  passwordHash: { type: String, required: true },
  role: { type: String, enum: ['student', 'instructor'], required: true }
}
```

Course Model (Course.js)

```
{
  title: { type: String, required: true },
  description: { type: String, required: true },
  videoUrl: { type: String, required: true },
  resources: { type: [String], default: [] },
  createdBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true }
}
```

Quiz Model (Quiz.js)

```
{
  courseId: { type: mongoose.Schema.Types.ObjectId, ref: 'Course', required: true },
  questions: [
    {
      question: { type: String, required: true },
      options: { type: [String], required: true },
      answer: { type: String, required: true }
    }
  ]
}
```

4.5 Middleware

Authentication Middleware (auth.js):

- Validates JWT tokens in the Authorization header.
- Protects routes that require user authentication.
- Attaches user details to the request object if the token is valid.

Frontend Design

5.1 Technologies Used

- React.js: For building a responsive and interactive user interface.
- React Router: For managing client-side routing.
- Tailwind CSS: A utility-first CSS framework for fast and customizable styling.
- Vite: Build tool for faster development and optimized production builds.

5.2 Project Structure

```
frontend/
├── public/              # Static assets like favicon.ico
├── src/
│   ├── components/      # Reusable UI components (e.g., Navbar, Footer)
│   ├── pages/           # Page components corresponding to routes (e.g., Home, Courses)
│   ├── App.jsx          # Root component managing routes
│   ├── main.jsx         # Entry point of the application
│   ├── App.css          # Global styles
│   └── index.css        # Tailwind CSS directives
└── vite.config.js       # Vite configuration
```

5.3 Routing

Implemented using **React Router**:

- /: Home page displaying an overview of Syntax School.
- /courses: Displays a list of available courses.
- /about: Information about Syntax School.
- /login: User login page.
- /register: User registration page.
- /course/:id: Displays course details, including video and resources.

5.4 State Management

Local State: Managed using React's useState and useEffect hooks.
Authentication State:

- JWT tokens are stored in localStorage.
- Custom hooks manage user authentication and authorization state.

Data Fetching:

- RESTful API calls using the Fetch API or Axios.
- Loading and error states are handled locally.

5.5 Key Components

Navbar Component

- Displays navigation links dynamically based on user authentication state.
- Provides quick access to key sections (e.g., Home, Courses, Login).

CourseCard Component

- Displays course details (title, description, instructor name).
- Includes a “Learn More” button linking to the course details page.

QuizComponent

- Renders quizzes for students to complete.
- Displays immediate feedback for multiple-choice questions.

ProgressTracker Component

- Visual representation of lesson and quiz completion within a course.

Security Considerations

6.1 Authentication

JWT Tokens:

- Securely generated on user login and signed with a secret key.
- Stored in localStorage for persistent user sessions.
- Tokens are validated for all protected routes on the backend.

Password Security:

- User passwords are hashed using bcrypt before being stored in the database.
- Plaintext passwords are never stored or logged.

6.2 Authorization

Protected Routes:

- Backend routes validate JWT tokens to ensure only authenticated users have access.
- Role-based access control (RBAC) restricts actions based on user roles (e.g., instructor vs. student).

Frontend Restrictions:

- Role-based navigation dynamically displays options based on user permissions.

6.3 Input Validation

Backend Validation:

- All incoming requests are validated to prevent injection attacks (e.g., SQL/NoSQL injection).
- Validation libraries (e.g., express-validator) ensure data consistency.

Frontend Validation:

- Input forms sanitize and validate data before sending requests to the backend.

6.4 CORS Configuration

Backend CORS policy is configured to allow requests only from trusted origins (e.g., frontend URL).

Proper headers are set, such as:

- Access-Control-Allow-Origin: Specifies allowed domains.
- Access-Control-Allow-Methods: Limits HTTP methods (e.g., GET, POST).

6.5 Error Handling

Detailed error messages are logged on the backend but never exposed to the frontend to avoid information leakage.

Frontend displays user-friendly error messages for failed operations.

6.6 Session Security

JWT expiration times are set to limit the lifespan of authentication tokens.

Refresh tokens or re-login prompts are used to re-authenticate users.

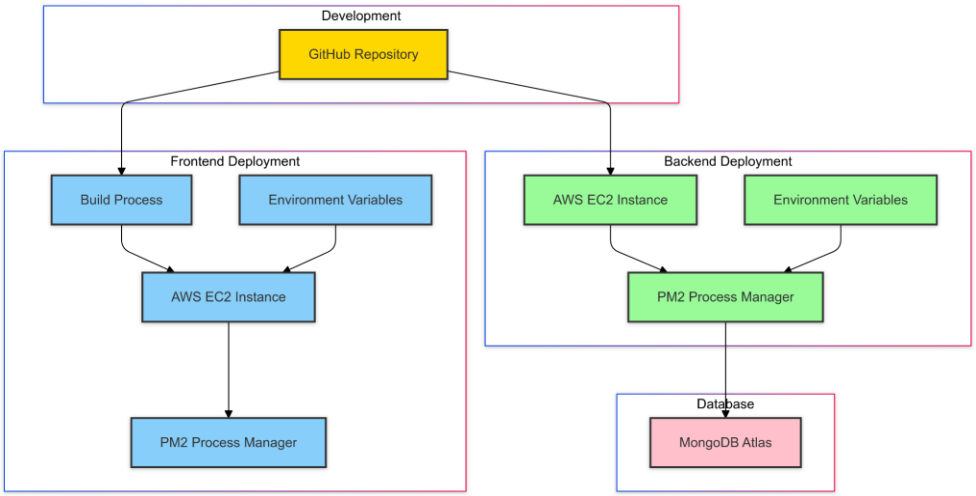
Deployment Plans

Environment Setup

- **Backend Environment Variables**:
  - PORT: Port number for the server.
  - DB_CONNECTION_STRING: MongoDB connection URI.
  - JWT_SECRET: Secret key for signing JWTs.
- **Frontend Environment Variables**:
  - VITE_API_URL: Base URL for the backend API.

Deployment Steps

1. **Backend Deployment**:
   - Host on platforms like Heroku, AWS EC2, or DigitalOcean.
   - Ensure environment variables are securely set.
   - Use process managers like PM2 for process management.
2. **Frontend Deployment**:
   - Build the React application using npm run build.
   - Host static files on services like Vercel or AWS.
3. **Domain and SSL**:
   - Configure a custom domain.
   - Set up SSL certificates for secure HTTPS communication.
4. **Database Hosting**:
   - Use managed MongoDB services like MongoDB Atlas.
   - Configure IP whitelisting and security measures.
5. **Continuous Deployment**:
   - Set up CD pipelines to automatically deploy on code changes.
   - Use GitHub Actions or other CI/CD tools.



Future Plans

8.1 Technical Improvements

1. **Implement TypeScript**:
    - Introduce TypeScript for type safety and improved maintainability of both frontend and backend code.
2. **State Management Library**:
    - Migrate to **Redux** or **React Context API** for managing complex application state.
3. **Real-Time Features**:
    - Use **WebSockets** or **Socket.IO** to enable real-time notifications, collaborative learning tools, and live quiz competitions.

8.2 Feature Enhancements

1. **Discussion Forums**:
    - Add forums for students and instructors to engage in topic-specific discussions.
2. **Gamification**:
    - Introduce leaderboards, badges, and rewards to enhance engagement.
3. **Course Recommendations**:
    - Implement a recommendation engine to suggest courses based on user interests and activity.
4. **Student Progress Reports**:
    - Provide detailed analytics on course completion, quiz performance, and learning milestones.
5. **Mobile Application**:
    - Build a dedicated mobile app for iOS and Android using **React Native** to improve accessibility.

8.3 Advanced Learning Features

1. **Live Classes**:
    - Integrate live video classes using tools like **Zoom API** or **WebRTC**.
2. **Code Editor**:
    - Embed an interactive code editor (e.g., **Monaco Editor**) for coding assignments directly on the platform.
3. **Certificates**:
    - Issue certificates of completion for students finishing courses.
4. **Multilingual Support**:
    - Localize the platform to support multiple languages for broader accessibility

Conclusion

Syntax School is a modern, scalable, and accessible platform designed to revolutionize the way computer science is taught and learned. By leveraging the MERN stack, the platform delivers an engaging and interactive learning experience through features like video lessons, quizzes, and resource sharing, all while maintaining cost-efficiency through free hosting solutions like YouTube and MongoDB Atlas.

The architecture ensures flexibility and future scalability, making it possible to enhance the platform with advanced features such as live classes, gamification, and personalized recommendations. Security measures like JWT authentication and input validation guarantee a safe environment for both students and instructors.

This project reflects a strong commitment to providing accessible education and sets the foundation for a highly impactful e-learning system. Syntax School is not just a platform—it's a step toward empowering learners and educators in the digital age.