Edu-Learn Platform - System Architecture

Table of Contents

High-Level Overview

Edu-Learn is a comprehensive online learning platform that connects students with courses and mentors. The platform supports course management, interactive quizzes, real-time chat, mentor sessions, and progress tracking.

Key Features

- **Multi-Role System**: Students, Mentors, and Admins
- **Course Management**: Create, edit, and browse courses with lessons and quizzes
- **Interactive Learning**: Quiz system with topic-based recommendations
- **Mentor System**: Book, confirm, and reschedule mentor sessions
- **Real-Time Chat**: WebSocket-based messaging between users
- **Admin Dashboard**: User management, course management, and analytics

---

System Architecture Diagram

```
graph TB
    subgraph "Client Layer"
        WEB[Web Browser]
        UI[React Frontend
Vite + TypeScript]
    end

    subgraph "Application Layer"
        API[Express.js API Server
Port 3001]
        WS[Socket.IO Server
WebSocket]
        AUTH[JWT Authentication]
```

```
    end

    subgraph "Business Logic Layer"
        ROUTES[API Routes]
        CONTROLLERS[Controllers]
        SERVICES[Services]
        MIDDLEWARE[Middleware]
    end

    subgraph "Data Layer"
        PG[(PostgreSQL Database)]
        CACHE[Session Cache]
    end

    WEB --> UI
    UI -->|HTTP/REST| API
    UI -->|WebSocket| WS
    API --> AUTH
    WS --> AUTH
    AUTH --> ROUTES
    ROUTES --> CONTROLLERS
    CONTROLLERS --> SERVICES
    SERVICES --> PG
    MIDDLEWARE --> AUTH
    API --> CACHE
```

Technology Stack

Frontend

Technology	Purpose	Version
React	UI Framework	18.x
TypeScript	Type Safety	5.x
Vite	Build Tool & Dev Server	Latest
TailwindCSS	Styling Framework	3.x
Axios	HTTP Client	Latest
Lucide React	Icon Library	Latest
Socket.IO Client	Real-time Communication	Latest

Backend

Technology	Purpose	Version
Node.js	Runtime Environment	18+
Express.js	Web Framework	4.x

Technology	Purpose	Version
TypeScript	Type Safety	5.x
PostgreSQL	Primary Database	13+
Socket.IO	WebSocket Server	Latest
JWT	Authentication	Latest
bcrypt	Password Hashing	Latest
pg	PostgreSQL Client	Latest

DevOps & Tools

- **ts-node-dev**: Development server with hot reload
- **dotenv**: Environment variable management
- **CORS**: Cross-origin resource sharing
- **ESLint**: Code linting

Frontend Architecture

Component Structure

```
graph TD
    APP[App.tsx]
    ROUTER[Router Component]
    LAYOUT[Layout Component]
    AUTH_CTX[AuthContext]

    APP --> ROUTER
    APP --> AUTH_CTX
    ROUTER --> LAYOUT

    subgraph "Pages"
        STUDENT[Student Pages]
        MENTOR[Mentor Pages]
        ADMIN[Admin Pages]
        COURSE[Course Pages]
        CHAT[Chat Pages]
    end

    ROUTER --> STUDENT
    ROUTER --> MENTOR
    ROUTER --> ADMIN
    ROUTER --> COURSE
    ROUTER --> CHAT

    subgraph "Student Pages"
        S_DASH[Dashboard]
        S_CATALOG[Course Catalog]
```

```
        S_DETAIL[Course Detail]
        S_LESSON[Lesson Viewer]
    end

    subgraph "Admin Pages"
        A_DASH[Admin Dashboard]
        A_COURSE[Course Management]
        A_CREATE[Course Creation]
        A_USERS[User Management]
    end

    subgraph "Mentor Pages"
        M_DIR[Mentor Directory]
        M_SETUP[Mentor Setup]
    end
```

Key Frontend Components

Pages Directory Structure

```
src/pages/
├── admin/
│   ├── AdminDashboard.tsx       # Admin overview & stats
│   ├── AdminCourseManagement.tsx # Course CRUD operations
│   ├── CourseCreation.tsx       # Create/Edit courses & lessons
│   └── UserManagement.tsx       # User role management
├── student/
│   └── StudentDashboard.tsx     # Student progress & enrolled courses
├── courses/
│   ├── CourseCatalog.tsx        # Browse all courses
│   └── CourseDetail.tsx         # Course overview & enrollment
├── lessons/
│   └── LessonViewer.tsx         # View lessons & take quizzes
├── mentors/
│   └── MentorDirectory.tsx      # Browse & book mentors
├── mentor/
│   └── MentorSetup.tsx          # Mentor profile setup
├── chat/
│   └── ChatPage.tsx             # Real-time messaging
├── auth/
│   ├── LoginPage.tsx
│   └── RegisterPage.tsx
└── profile/
    └── ProfilePage.tsx
```

State Management

- **AuthContext**: Global authentication state (user, role, token)
- **Local State**: Component-level state with React hooks

- **API State**: Managed via axios with cache-busting

Backend Architecture

Layered Architecture

```
graph TB
    subgraph "Presentation Layer"
        ROUTES[Routes]
    end

    subgraph "Business Logic Layer"
        CONTROLLERS[Controllers]
        SERVICES[Services]
        MIDDLEWARE[Middleware]
    end

    subgraph "Data Access Layer"
        DB_POOL[Database Pool]
        QUERIES[SQL Queries]
    end

    subgraph "External Services"
        SOCKET[Socket.IO]
        JWT_SVC[JWT Service]
    end

    ROUTES --> MIDDLEWARE
    MIDDLEWARE --> CONTROLLERS
    CONTROLLERS --> SERVICES
    SERVICES --> DB_POOL
    DB_POOL --> QUERIES
    CONTROLLERS --> SOCKET
    MIDDLEWARE --> JWT_SVC
```

API Routes Structure

```
backend/src/routes/
├── auth.routes.ts           # /api/auth/*
│   ├── POST /register
│   └── POST /login
├── admin.routes.ts          # /api/admin/*
│   ├── GET /stats
│   ├── GET /courses
│   ├── POST /courses
│   ├── PUT /courses/:id
│   ├── PUT /courses/:id/lessons
│   ├── POST /courses/:id/seed-lessons
│   ├── GET /lessons/all
```

```
│   ├── GET /users
│   ├── PATCH /users/:id/role
│   └── DELETE /users/:id
├── courses.routes.ts        # /api/courses/*
│   ├── GET /catalog
│   ├── GET /:id
│   ├── POST /:id/enroll
│   └── GET /lessons
├── quiz.routes.ts           # /api/quizzes/*
│   ├── GET /:id
│   ├── POST /:id/attempt
│   └── GET /attempts/:attemptId
├── mentor.routes.ts         # /api/mentors/*
│   ├── GET /
│   └── GET /:id
├── mentorSession.routes.ts  # /api/mentor-sessions/*
│   ├── POST /book
│   ├── PATCH /:id/confirm
│   └── PATCH /:id/reschedule
├── chat.routes.ts           # /api/chat/*
│   ├── GET /conversations
│   ├── GET /conversations/:id/messages
│   └── POST /conversations/:id/messages
├── student.routes.ts        # /api/student/*
│   └── GET /dashboard
└── profile.routes.ts        # /api/profile/*
    ├── GET /
    └── PUT /
```

Database Schema

Entity Relationship Diagram

```
erDiagram
    AUTH_USERS ||--|| PROFILES : "has"
    PROFILES ||--o{ ENROLLMENTS : "enrolls"
    PROFILES ||--o{ QUIZ_ATTEMPTS : "attempts"
    PROFILES ||--o{ MENTOR_SESSIONS : "books"
    PROFILES ||--o{ MENTOR_SESSIONS : "mentors"
    PROFILES ||--o{ CHAT_MESSAGES : "sends"
    PROFILES ||--o{ CONVERSATION_PARTICIPANTS : "participates"

    COURSES ||--o{ LESSONS : "contains"
    COURSES ||--o{ ENROLLMENTS : "enrolled_in"

    LESSONS ||--o{ QUIZ_QUESTIONS : "has"
    LESSONS ||--o{ QUIZ_ATTEMPTS : "attempted"

    QUIZ_ATTEMPTS ||--o{ QUIZ_ANSWERS : "contains"
    QUIZ_QUESTIONS ||--o{ QUIZ_ANSWERS : "answered"
```

```
    CHAT_CONVERSATIONS ||--o{ CHAT_MESSAGES : "contains"
    CHAT_CONVERSATIONS ||--o{ CONVERSATION_PARTICIPANTS : "has"

    AUTH_USERS {
        uuid id PK
        string email UK
        string encrypted_password
        timestamp created_at
    }

    PROFILES {
        uuid id PK,FK
        string full_name
        enum role
        timestamp created_at
    }

    COURSES {
        uuid id PK
        string title
        text description
        string category
        enum difficulty_level
        int estimated_duration_hours
        timestamp created_at
    }

    LESSONS {
        uuid id PK
        uuid course_id FK
        string title
        enum content_type
        text content
        int order_index
    }

    QUIZ_QUESTIONS {
        uuid id PK
        uuid lesson_id FK
        text question_text
        jsonb options
        int correct_option_index
        string topic
    }

    QUIZ_ATTEMPTS {
        uuid id PK
        uuid user_id FK
        uuid lesson_id FK
        int score
        timestamp completed_at
    }
```

```
    QUIZ_ANSWERS {
        uuid id PK
        uuid attempt_id FK
        uuid question_id FK
        int selected_option
        boolean is_correct
    }

    ENROLLMENTS {
        uuid id PK
        uuid user_id FK
        uuid course_id FK
        timestamp enrolled_at
    }

    MENTOR_SESSIONS {
        uuid id PK
        uuid student_id FK
        uuid mentor_id FK
        timestamp scheduled_at
        timestamp rescheduled_at
        int duration_minutes
        enum session_type
        enum status
        text session_notes
    }

    CHAT_CONVERSATIONS {
        uuid id PK
        enum type
        timestamp created_at
    }

    CHAT_MESSAGES {
        uuid id PK
        uuid conversation_id FK
        uuid sender_id FK
        text message_text
        enum message_type
        jsonb metadata
        timestamp sent_at
    }

    CONVERSATION_PARTICIPANTS {
        uuid id PK
        uuid conversation_id FK
        uuid user_id FK
        timestamp joined_at
    }
```

Key Database Tables

Authentication Schema (`auth` namespace)

- **users**: Email, encrypted password, user ID

Public Schema

- **profiles**: User profiles with role (student, mentor, admin)
- **courses**: Course metadata and details
- **lessons**: Course content (text, video, quiz)
- **quiz_questions**: Questions with options and correct answers
- **quiz_attempts**: User quiz submissions and scores
- **quiz_answers**: Individual question answers
- **enrollments**: Student-course relationships
- **mentor_sessions**: Scheduled sessions (status: scheduled, completed, cancelled)
- **chat_conversations**: Direct and group conversations
- **chat_messages**: Messages with type (text, session_booking)
- **conversation_participants**: User-conversation relationships

---

API Structure

RESTful API Endpoints

Authentication

```
POST /api/auth/register
POST /api/auth/login
```

Courses

```
GET  /api/courses/catalog          # Browse all courses
GET  /api/courses/:id              # Get course details
POST /api/courses/:id/enroll       # Enroll in course
GET  /api/courses/lessons?topic=X  # Get lessons by topic
```

Quizzes

```
GET  /api/quizzes/:id              # Get quiz questions
POST /api/quizzes/:id/attempt      # Submit quiz attempt
GET  /api/quizzes/attempts/:id     # Get attempt results
```

Admin

```
GET    /api/admin/stats            # Platform statistics
GET    /api/admin/courses          # All courses
POST   /api/admin/courses          # Create course
PUT    /api/admin/courses/:id      # Update course
PUT    /api/admin/courses/:id/lessons  # Update lessons
GET    /api/admin/lessons/all      # All lessons
GET    /api/admin/users            # All users
PATCH  /api/admin/users/:id/role   # Update user role
DELETE /api/admin/users/:id        # Delete user
```

Mentors

```
GET   /api/mentors                 # Browse mentors
GET   /api/mentors/:id             # Mentor details
POST  /api/mentor-sessions/book    # Book session
PATCH /api/mentor-sessions/:id/confirm    # Confirm session
PATCH /api/mentor-sessions/:id/reschedule # Reschedule session
```

Chat

```
GET  /api/chat/conversations
GET  /api/chat/conversations/:id/messages
POST /api/chat/conversations/:id/messages
```

Real-Time Communication

WebSocket Architecture

```
sequenceDiagram
    participant Client
    participant SocketIO
    participant Server
    participant Database

    Client->>SocketIO: Connect with JWT
    SocketIO->>Server: Authenticate
    Server->>SocketIO: Connection Established

    Client->>SocketIO: Join Conversation Room
    SocketIO->>Server: Add to Room

    Client->>SocketIO: Send Message
    SocketIO->>Server: Process Message
    Server->>Database: Store Message
```

```
    Database->>Server: Confirm
    Server->>SocketIO: Broadcast to Room
    SocketIO->>Client: Deliver Message
```

Socket.IO Events

- **Connection**: User authentication and room joining
- **new_message**: Real-time message delivery
- **message_updated**: Update existing messages (e.g., session confirmations)
- **typing**: Typing indicators
- **disconnect**: Clean up user sessions

Authentication & Authorization

JWT-Based Authentication Flow

```
sequenceDiagram
    participant User
    participant Frontend
    participant API
    participant Database

    User->>Frontend: Enter Credentials
    Frontend->>API: POST /api/auth/login
    API->>Database: Verify Credentials
    Database->>API: User Data
    API->>API: Generate JWT Token
    API->>Frontend: Return Token + User
    Frontend->>Frontend: Store Token

    Frontend->>API: API Request + Bearer Token
    API->>API: Verify JWT
    API->>Database: Fetch Data
    Database->>API: Return Data
    API->>Frontend: Response
```

Role-Based Access Control

- **Student**: Access courses, take quizzes, book mentors, chat
- **Mentor**: Manage sessions, chat with students, view profile
- **Admin**: Full access to user management, course management, analytics

Middleware Stack

1. **CORS**: Cross-origin request handling
2. **JSON Parser**: Request body parsing
3. **requireAuth**: JWT token validation

4. **Role Check**: Route-specific role validation

Data Flow Diagrams

Course Enrollment Flow

```
flowchart TD
    A[Student Browses Catalog] --> B{Course Selected}
    B --> C[View Course Details]
    C --> D{Already Enrolled?}
    D -->|Yes| E[Continue to Course]
    D -->|No| F[Click Enroll]
    F --> G[POST /api/courses/:id/enroll]
    G --> H[Create Enrollment Record]
    H --> I[Update UI State]
    I --> E
    E --> J[View Lessons]
    J --> K{Lesson Type?}
    K -->|Text/Video| L[Display Content]
    K -->|Quiz| M[Load Quiz Questions]
    M --> N[Student Answers]
    N --> O[Submit Quiz Attempt]
    O --> P[Calculate Score]
    P --> Q{Failed Topics?}
    Q -->|Yes| R[Show Recommendations]
    Q -->|No| S[Show Success]
```

Mentor Session Booking Flow

```
flowchart TD
    A[Student Browses Mentors] --> B[Select Mentor]
    B --> C[Choose Date/Time]
    C --> D[POST /api/mentor-sessions/book]
    D --> E[Create Session Record]
    E --> F[Find/Create Conversation]
    F --> G[Send Booking Message]
    G --> H[Emit Socket Event]
    H --> I[Mentor Receives Notification]
    I --> J{Mentor Action}
    J -->|Confirm| K[PATCH /:id/confirm]
    J -->|Reschedule| L[PATCH /:id/reschedule]
    K --> M[Update Session]
    L --> M
    M --> N[Update Chat Message]
    N --> O[Notify Student via Socket]
```

Admin User Management Flow

```
flowchart TD
    A[Admin Navigates to /admin/users] --> B[GET /api/admin/users]
    B --> C[JOIN auth.users + profiles]
    C --> D[Display User Table]
    D --> E{Admin Action}
    E -->|Change Role| F[Select New Role]
    E -->|Delete User| G[Click Delete]
    E -->|Search| H[Enter Search Term]

    F --> I[PATCH /api/admin/users/:id/role]
    I --> J[UPDATE profiles SET role]
    J --> K[Return Updated User]
    K --> L[Update UI]

    G --> M[Show Confirmation Modal]
    M --> N{Confirm?}
    N -->|Yes| O[DELETE /api/admin/users/:id]
    N -->|No| D
    O --> P[DELETE FROM profiles]
    P --> Q[Remove from UI]

    H --> R[GET /api/admin/users?search=X]
    R --> S[Filter Results]
    S --> D
```

Deployment Architecture

Development Environment

```
Frontend: http://localhost:5173 (Vite Dev Server)
Backend:  http://localhost:3001 (Express + ts-node-dev)
Database: PostgreSQL (Local or Remote)
```

Production Considerations

1. **Frontend**: Build with `npm run build`, serve static files
2. **Backend**: Compile TypeScript, run with Node.js
3. **Database**: PostgreSQL with connection pooling
4. **Environment Variables**: Secure storage of JWT_SECRET, DB credentials
5. **CORS**: Configure allowed origins
6. **WebSocket**: Ensure WebSocket support in hosting environment

---

Security Measures

Implemented Security Features

1. **Password Hashing**: bcrypt with salt rounds
2. **JWT Tokens**: 7-day expiration
3. **SQL Injection Prevention**: Parameterized queries
4. **CORS**: Configured cross-origin policies
5. **Role-Based Access**: Middleware-enforced permissions
6. **Input Validation**: Server-side validation for all inputs

Recommended Enhancements

- Rate limiting on API endpoints
- HTTPS enforcement
- Refresh token rotation
- Session management
- Input sanitization
- CSRF protection
- Content Security Policy headers

Performance Optimizations

Current Optimizations

1. **Database Connection Pooling**: Reuse connections
2. **Cache-Busting**: Timestamp-based fresh data loading
3. **Lazy Loading**: Component-level code splitting
4. **WebSocket**: Efficient real-time updates
5. **Indexed Queries**: Database indexes on foreign keys

Future Optimizations

- Redis caching layer
- CDN for static assets
- Database query optimization
- Image optimization
- Pagination for large datasets
- Server-side rendering (SSR)

Monitoring & Logging

Current Logging

- Console logging for errors and key events
- Database query logging (development)
- Socket.IO connection logs

Recommended Additions

- Structured logging (Winston, Pino)
- Error tracking (Sentry)

- Performance monitoring (New Relic, DataDog)
- Database query analytics
- User activity tracking

Conclusion

The Edu-Learn platform is built with a modern, scalable architecture that separates concerns between frontend, backend, and database layers. The use of TypeScript throughout ensures type safety, while the modular structure allows for easy maintenance and feature additions.

Key Architectural Strengths

- **Separation of Concerns**: Clear boundaries between layers
- **Type Safety**: TypeScript on both frontend and backend
- **Real-Time Capabilities**: WebSocket integration for live updates
- **Scalable Database Design**: Normalized schema with proper relationships
- **RESTful API**: Standard HTTP methods and status codes
- **Role-Based Access**: Secure, permission-based routing

Future Architecture Considerations

- Microservices migration for high-traffic modules
- GraphQL API for flexible data fetching
- Caching layer for frequently accessed data
- Message queue for asynchronous processing
- Containerization with Docker
- Kubernetes orchestration for scaling