MPOVR Training Management System Documentation

Contents

MPOVR Training Management System Documentation 1

1. Introduction 4

2. System Overview 4

2.1 Technology Stack 4

2.2 System Architecture 4

2.3 Key Features 4

3. Project Structure 5

3.1 Backend Structure 5

3.2 Frontend Structure 5

4. Database Schema 6

4.1 Core Entities 6

4.2 Training Content Entities 6

4.3 Communication Entities 7

4.4 Session and Progress Entities 8

4.5 Enrollment and Payment Entities 8

5. API Endpoints 9

5.1 Authentication Endpoints 9

5.2 User Management Endpoints 9

5.3 Program Management Endpoints 10

5.4 Content Management Endpoints 10

5.5 Assignment Management Endpoints 11

5.6 Quiz Management Endpoints 12

5.7 Discussion Management Endpoints 13

5.8 Virtual Session Management Endpoints 15
5.9 Messaging Endpoints 16
5.10 WebSocket Endpoint 17
6. Authentication and Authorization 17
6.1 Authentication Flow 17
6.2 Token Generation and Validation 17
6.3 Password Hashing 17
6.4 Role-Based Access Control 17
7. User Roles and Permissions 18
7.1 Learner Role 18
7.2 Trainer Role 18
7.3 Admin Role 18
7.4 Role Admin 18
8. Core Features 19
8.1 User Management 19
8.2 Program Management 19
8.3 Content Management 19
8.4 Assignment Management 19
8.5 Quiz Management 20
8.6 Virtual Session Management 20
8.7 Discussion Forums 20
8.8 Messaging System 20
8.9 Progress Tracking 20
9. Frontend Components 21
9.1 Trainer Portal Components 21
9.2 Website Components 22
10. Deployment and Configuration 23

10.1 Environment Variables 23
10.2 Database Configuration 23
10.3 Server Configuration 23
10.4 Deployment Process 23
11. Development Guidelines 24
11.1 Code Style 24
11.2 Git Workflow 24
11.3 Testing 24
11.4 Documentation 24
12. Troubleshooting 24
12.1 Common Issues 24
12.2 Logging 25
12.3 Monitoring 25

1. Introduction

The MPOVR Training Management System is a comprehensive web-based platform designed to facilitate online IT training for individuals seeking to enter high-growth global IT careers. The system caters to users with at least 2+ years of work experience, strong academic records, and good communication skills.

The platform consists of three main portals: - **Trainer Portal**: For instructors to manage training content, sessions, and student progress - **Learner Portal**: For students to access training materials, participate in sessions, and complete assignments - **Admin Portal**: For administrators to oversee the entire system, manage users, and configure workflows

This documentation provides a comprehensive overview of the system architecture, functionality, and implementation details to facilitate seamless knowledge transfer to new developers joining the project.

2. System Overview

2.1 Technology Stack

The MPOVR Training Management System is built using the following technologies:

- **Backend**:
  - FastAPI (Python web framework)
  - PostgreSQL (Relational database)
  - SQLAlchemy (ORM for database interactions)
  - Alembic (Database migration tool)
  - WebSockets (Real-time communication)
- **Frontend**:
  - React.js (JavaScript library for building user interfaces)
  - Tailwind CSS (Utility-first CSS framework)

2.2 System Architecture

The system follows a client-server architecture:

1. **Client-side**: React.js application that provides the user interface for all three portals
2. **Server-side**: FastAPI application that handles API requests, database operations, and business logic
3. **Database**: PostgreSQL database that stores all application data
4. **WebSockets**: For real-time communication features like chat and notifications

2.3 Key Features

- User authentication and role-based access control
- Program enrollment and management

- Content delivery (videos, documents, links)
- Assignment submission and grading
- Quiz creation and assessment
- Virtual session scheduling and attendance tracking
- Discussion forums
- Messaging system with file attachments
- Progress tracking
- Profile management

3. Project Structure

The project is organized into the following main directories:

3.1 Backend Structure

```
mpovr_backend/
├── .env                    # Environment variables
├── alembic.ini             # Alembic configuration
├── config.py               # Application configuration
├── requirements.txt        # Python dependencies
├── run.py                  # Application entry point
└── app/
    ├── __init__.py         # Application initialization
    ├── auth.py             # Authentication logic
    ├── crud.py             # Database CRUD operations
    ├── database.py         # Database connection setup
    ├── models.py           # SQLAlchemy models
    ├── routes.py           # API endpoints
    ├── schemas.py          # Pydantic schemas for request/response
validation
    ├── websocket_handler.py # WebSocket implementation
    ├── config/             # Configuration files
    └── uploads/            # Uploaded files storage
```

3.2 Frontend Structure

```
src/
├── App.css                 # Global CSS
├── App.js                  # Main application component
├── index.css               # Entry CSS file
├── index.js                # Application entry point
├── components/             # Reusable UI components
│   ├── login.tsx           # Login component
│   ├── theme.js            # Theme configuration
│   ├── trainerMainPage.tsx # Main page for trainers
│   ├── trainer_portal/     # Trainer portal components
│   └── website/            # Public website components
├── fonts/                  # Custom fonts
└── images/                 # Static images
```

4. Database Schema

The database schema is defined in `models.py` and consists of the following main entities:

4.1 Core Entities

4.1.1 User

The `User` table stores information about all system users, including learners, trainers, and administrators.

Key fields: - `user_id`: Primary key - `unique_id`: Unique identifier for the user - `username`: Username for login - `email`: User's email address - `password_hash`: Hashed password - `role`: User role (learner, trainer, admin, role_admin) - `program_id`: Foreign key to the associated program

Relationships: - One-to-one relationship with `Profile` - One-to-many relationships with various entities like `Application`, `Payment`, `Enrollment`, etc.

4.1.2 Profile

The `Profile` table stores detailed information about users.

Key fields: - `profile_id`: Primary key - `user_id`: Foreign key to the User table - `full_name`: User's full name - `date_of_birth`: User's date of birth - `phone_number`: User's phone number - `address`: User's address - `education_history`: JSON field for education history - `work_experience`: JSON field for work experience

4.1.3 Program

The `Program` table stores information about training programs offered by MPOVR.

Key fields: - `program_id`: Primary key - `name`: Program name - `description`: Program description - `duration`: Program duration in weeks/months - `fee`: Program fee - `start_date`: Program start date

Relationships: - One-to-many relationships with `Assignment`, `Application`, `Payment`, `Enrollment`, etc.

4.2 Training Content Entities

4.2.1 Module

The `Module` table represents sections or units within a program.

Key fields: - `module_id`: Primary key - `program_id`: Foreign key to the Program table - `name`: Module name - `description`: Module description - `order_number`: Order of the module within the program

4.2.2 Content

The `Content` table stores various types of learning materials.

Key fields: - `content_id`: Primary key - `title`: Content title - `content_type`: Type of content (video, document, link, etc.) - `description`: Content description - `file_path`: Path to the content file - `url`: URL for external content - `program_id`: Foreign key to the Program table - `user_id`: Foreign key to the User table (creator) - `week`: Week number in the program

4.2.3 Assignment

The `Assignment` table stores assignments for learners.

Key fields: - `assignment_id`: Primary key - `program_id`: Foreign key to the Program table - `description`: Assignment description - `due_date`: Assignment due date - `uploaded_content`: Path to the assignment file - `user_id`: Foreign key to the User table (creator) - `week`: Week number in the program

4.2.4 Quiz

The `Quiz` table stores quizzes for learners.

Key fields: - `quiz_id`: Primary key - `title`: Quiz title - `description`: Quiz description - `creator_id`: Foreign key to the User table (creator) - `program_id`: Foreign key to the Program table - `start_date`: Quiz start date - `end_date`: Quiz end date - `week`: Week number in the program

Related tables: - `QuizQuestion`: Stores quiz questions - `QuizOption`: Stores options for quiz questions - `QuizAttempt`: Stores learner attempts at quizzes - `QuizResponse`: Stores learner responses to quiz questions

4.3 Communication Entities

4.3.1 Message

The `Message` table stores messages between users.

Key fields: - `message_id`: Primary key - `sender_id`: Foreign key to the User table (sender) - `program_id`: Foreign key to the Program table - `content`: Message content - `status`: Message status (pending, approved, rejected) - `parent_id`: Foreign key to the Message table (for replies) - `attachments`: Array of attachment file paths - `attachments_size`: Array of attachment file sizes

4.3.2 Discussion

The `Discussion` table stores discussion topics.

Key fields: - `discussion_id`: Primary key - `title`: Discussion title - `description`: Discussion description - `file_path`: Path to an attached file - `program_id`: Foreign key to

the Program table - `user_id`: Foreign key to the User table (creator) - `week`: Week number in the program

Related tables: - `DiscussionReply`: Stores replies to discussions

4.4 Session and Progress Entities

4.4.1 VirtualSession

The `VirtualSession` table stores information about virtual training sessions.

Key fields: - `session_id`: Primary key - `program_id`: Foreign key to the Program table - `user_id`: Foreign key to the User table (creator) - `title`: Session title - `description`: Session description - `scheduled_datetime`: Session date and time - `duration_minutes`: Session duration - `platform`: Platform for the session (e.g., Zoom) - `meeting_link`: Link to join the session - `access_code`: Access code for the session - `week`: Week number in the program - `attended_count`: Number of attendees

Related tables: - `VirtualSessionAttendance`: Tracks learner attendance in virtual sessions

4.4.2 ProgressTracking

The `ProgressTracking` table tracks learner progress through content.

Key fields: - `progress_id`: Primary key - `user_id`: Foreign key to the User table - `content_id`: Foreign key to the Content table - `completed_at`: Completion date and time

4.4.3 ProgramProgress

The `ProgramProgress` table tracks the current week of a program.

Key fields: - `progress_id`: Primary key - `program_id`: Foreign key to the Program table - `current_week`: Current week number - `updated_at`: Last update date and time

4.5 Enrollment and Payment Entities

4.5.1 Application

The `Application` table stores learner applications to programs.

Key fields: - `application_id`: Primary key - `user_id`: Foreign key to the User table - `program_id`: Foreign key to the Program table - `status`: Application status (pending, approved, rejected) - `interview_slot`: Interview date and time

4.5.2 Payment

The `Payment` table stores payment information.

Key fields: - `payment_id`: Primary key - `user_id`: Foreign key to the User table - `program_id`: Foreign key to the Program table - `amount`: Payment amount - `status`: Payment status (pending, completed, refunded) - `stripe_payment_id`: Stripe payment ID

4.5.3 Enrollment

The `Enrollment` table stores learner enrollments in programs.

Key fields: - `enrollment_id`: Primary key - `user_id`: Foreign key to the User table - `program_id`: Foreign key to the Program table - `start_date`: Enrollment start date - `end_date`: Enrollment end date - `status`: Enrollment status (active, completed, discontinued)

4.5.4 Agreement

The `Agreement` table stores signed agreements.

Key fields: - `agreement_id`: Primary key - `user_id`: Foreign key to the User table - `program_id`: Foreign key to the Program table - `agreement_text`: Agreement text - `signed_at`: Signing date and time - `docusign_envelope_id`: DocuSign envelope ID

5. API Endpoints

The API endpoints are defined in `routes.py` and are organized by functionality.

5.1 Authentication Endpoints

5.1.1 Login

```
POST /token
```

Description: Authenticates a user and returns an access token.

Request body: - `username`: User's username - `password`: User's password

Response: - `access_token`: JWT access token - `token_type`: Token type (bearer)

5.2 User Management Endpoints

5.2.1 Create User

```
POST /users/
```

Description: Creates a new user.

Request body: - User information (email, password, role, etc.)

Response: - Created user information

5.2.2 Get Current User

```
GET /users/me/
```

Description: Returns information about the currently authenticated user.

Response: - Current user information

5.2.3 Get Trainer Profile

```
GET /profile
```

Description: Returns the profile information of the currently authenticated trainer.

Response: - Trainer profile information

5.2.4 Update Trainer Profile

`PUT /profile`

Description: Updates the profile information of the currently authenticated trainer.

Request body: - Profile information to update

Response: - Updated profile information

5.3 Program Management Endpoints

5.3.1 Get Program Details

`GET /program_details`

Description: Returns details about the program associated with the currently authenticated trainer.

Response: - Program details

5.3.2 Get Program Content

`GET /program_content`

Description: Returns content for the program associated with the currently authenticated user.

Query parameters: - `skip`: Number of items to skip - `limit`: Maximum number of items to return

Response: - List of program content items

5.3.3 Get Upcoming Events

`GET /upcoming_events`

Description: Returns upcoming events for the program associated with the currently authenticated user.

Query parameters: - `limit`: Maximum number of events to return

Response: - List of upcoming events

5.4 Content Management Endpoints

5.4.1 Create Content

`POST /content/`

Description: Creates new content.

Request body (form data): - `title`: Content title - `description`: Content description - `content_type`: Type of content - `week`: Week number - `uploaded_content`: Content file (optional)

Response: - Created content information

5.4.2 Get Contents

`GET /contents`

Description: Returns all content for the program associated with the currently authenticated user.

Response: - List of content items

5.4.3 Get Content Details

`GET /content/{content_id}`

Description: Returns details about a specific content item.

Path parameters: - `content_id`: ID of the content item

Response: - Content details

5.4.4 Update Content

`PUT /content/{content_id}`

Description: Updates a specific content item.

Path parameters: - `content_id`: ID of the content item

Request body (form data): - `title`: Content title - `description`: Content description - `content_type`: Type of content - `url`: Content URL (optional) - `file`: Content file (optional)

Response: - Updated content information

5.4.5 Record Content View

`POST /content/{content_id}/views`

Description: Records that the currently authenticated user has viewed a specific content item.

Path parameters: - `content_id`: ID of the content item

Response: - Success message

5.5 Assignment Management Endpoints

5.5.1 Create Assignment

`POST /assignments/`

Description: Creates a new assignment.

Request body (form data): - `title`: Assignment title - `description`: Assignment description - `due_date`: Assignment due date - `week`: Week number - `uploaded_content`: Assignment file (optional)

Response: - Created assignment information

5.5.2 Get Assignments

`GET /assignments/`

Description: Returns all assignments for the program associated with the currently authenticated user.

Response: - List of assignments

5.5.3 Get Assignment Submissions

`GET /assignment/{assignment_id}/submissions`

Description: Returns all submissions for a specific assignment.

Path parameters: - `assignment_id`: ID of the assignment

Response: - List of assignment submissions

5.5.4 Update Assignment

`PUT /assignment/{assignment_id}`

Description: Updates a specific assignment.

Path parameters: - `assignment_id`: ID of the assignment

Request body: - Assignment information to update

Response: - Updated assignment information

5.5.5 Grade Assignment

`POST /assignment/{assignment_id}/grade`

Description: Grades a submission for a specific assignment.

Path parameters: - `assignment_id`: ID of the assignment

Request body: - `unique_id`: Unique ID of the learner - `grade`: Grade to assign

Response: - Updated submission information

5.6 Quiz Management Endpoints

5.6.1 Create Quiz

`POST /quizzes/`

Description: Creates a new quiz.

Request body: - Quiz information (title, description, questions, options, etc.)

Response: - Created quiz information

5.6.2 Get Quizzes

`GET /quizzes`

Description: Returns all quizzes for the program associated with the currently authenticated user.

Response: - List of quizzes

5.6.3 Get Quiz Details

`GET /quiz/{quiz_id}`

Description: Returns details about a specific quiz.

Path parameters: - `quiz_id`: ID of the quiz

Response: - Quiz details

5.6.4 Get Quiz Attempts

`GET /quiz/{quiz_id}/attempts`

Description: Returns all attempts for a specific quiz.

Path parameters: - `quiz_id`: ID of the quiz

Response: - List of quiz attempts

5.6.5 Update Quiz

`PUT /quiz/{quiz_id}`

Description: Updates a specific quiz.

Path parameters: - `quiz_id`: ID of the quiz

Request body: - Quiz information to update

Response: - Updated quiz information

5.7 Discussion Management Endpoints

5.7.1 Create Discussion

`POST /discussions/`

Description: Creates a new discussion.

Request body (form data): - `title`: Discussion title - `description`: Discussion description - `week`: Week number - `uploaded_file`: Discussion file (optional)

Response: - Created discussion information

5.7.2 Get Discussions

`GET /discussions/`

Description: Returns all discussions for the program associated with the currently authenticated user.

Response: - List of discussions

5.7.3 Get Discussion

`GET /discussions/{discussion_id}`

Description: Returns details about a specific discussion.

Path parameters: - `discussion_id`: ID of the discussion

Response: - Discussion details

5.7.4 Update Discussion

`PUT /discussions/{discussion_id}`

Description: Updates a specific discussion.

Path parameters: - `discussion_id`: ID of the discussion

Request body: - Discussion information to update

Response: - Updated discussion information

5.7.5 Create Discussion Reply

`POST /discussions/{discussion_id}/replies`

Description: Creates a reply to a specific discussion.

Path parameters: - `discussion_id`: ID of the discussion

Request body: - Reply information

Response: - Created reply information

5.7.6 Get Discussion Replies

`GET /discussions/{discussion_id}/replies`

Description: Returns all replies to a specific discussion.

Path parameters: - `discussion_id`: ID of the discussion

Response: - List of discussion replies

5.7.7 Update Discussion Reply

`PUT /discussions/replies/{reply_id}`

Description: Updates a specific discussion reply.

Path parameters: - `reply_id`: ID of the reply

Request body: - Reply information to update

Response: - Updated reply information

5.8 Virtual Session Management Endpoints

5.8.1 Create Virtual Session

`POST /virtual_sessions/`

Description: Creates a new virtual session.

Request body: - Virtual session information (title, description, scheduled date/time, etc.)

Response: - Created virtual session information

5.8.2 Get Virtual Sessions

`GET /virtual_sessions/`

Description: Returns all virtual sessions for the program associated with the currently authenticated user.

Response: - List of virtual sessions

5.8.3 Get Virtual Session

`GET /virtual_sessions/{session_id}`

Description: Returns details about a specific virtual session.

Path parameters: - `session_id`: ID of the virtual session

Response: - Virtual session details

5.8.4 Update Virtual Session

`PUT /virtual_sessions/{session_id}`

Description: Updates a specific virtual session.

Path parameters: - `session_id`: ID of the virtual session

Request body (form data): - Virtual session information to update - `recording_file`: Recording file (optional)

Response: - Updated virtual session information

5.8.5 Get Trainee Attendance

`GET /virtual_session/{session_id}/attendance`

Description: Returns attendance information for a specific virtual session.

Path parameters: - `session_id`: ID of the virtual session

Response: - List of trainee attendance records

5.8.6 Join Virtual Session

`POST /virtual_sessions/{session_id}/join`

Description: Records that the currently authenticated user has joined a specific virtual session.

Path parameters: - `session_id`: ID of the virtual session

Response: - Success message

5.8.7 Leave Virtual Session

`POST /virtual_sessions/{session_id}/leave`

Description: Records that the currently authenticated user has left a specific virtual session.

Path parameters: - `session_id`: ID of the virtual session

Response: - Success message

5.9 Messaging Endpoints

5.9.1 Create Message

`POST /messages/`

Description: Creates a new message.

Request body: - Message information

Response: - Created message information

5.9.2 Get Messages

`GET /messages/list`

Description: Returns all messages for the program associated with the currently authenticated user.

Query parameters: - `skip`: Number of items to skip - `limit`: Maximum number of items to return

Response: - List of messages

5.9.3 Send Message

`POST /send_messages`

Description: Sends a new message.

Request body (form data): - `content`: Message content - `attachments`: Message attachments (optional)

Response: - Sent message information

5.9.4 Reply to Message

`POST /reply_message`

Description: Replies to a message.

Request body: - Reply information

Response: - Created reply information

5.10 WebSocket Endpoint

`WebSocket: /ws/chat`

Description: WebSocket endpoint for real-time chat functionality.

Query parameters: - `token`: Authentication token

6. Authentication and Authorization

6.1 Authentication Flow

The system uses JWT (JSON Web Tokens) for authentication. The authentication flow is as follows:

1. User submits their username and password to the `/token` endpoint.
2. The server verifies the credentials against the database.
3. If the credentials are valid, the server generates a JWT access token and returns it to the client.
4. The client includes this token in the `Authorization` header of subsequent requests.
5. The server validates the token and identifies the user for each request.

6.2 Token Generation and Validation

Token generation and validation are implemented in `auth.py`:

- `create_access_token()`: Generates a JWT access token with a specified expiration time.
- `get_current_user()`: Validates the token and returns the corresponding user.
- `get_current_active_user()`: Ensures that the user is active (not disabled).

6.3 Password Hashing

User passwords are hashed using bcrypt before being stored in the database. The hashing is implemented in `auth.py`:

- `get_password_hash()`: Hashes a password.
- `verify_password()`: Verifies a password against a hash.

6.4 Role-Based Access Control

The system implements role-based access control through the `UserRole` enum in `models.py`:

- `learner`: Regular learners who access training content.
- `trainer`: Trainers who create and manage training content.
- `admin`: Administrators who manage the system.

- `role_admin`: Super administrators with additional privileges.

API endpoints check the user's role to determine if they have permission to access the requested resource.

7. User Roles and Permissions

7.1 Learner Role

Learners have the following permissions:

- View and access assigned training content
- Submit assignments
- Take quizzes
- Participate in discussions
- Join virtual sessions
- Send messages (subject to approval)
- View their own profile and progress

7.2 Trainer Role

Trainers have the following permissions:

- Create and manage training content
- Create and grade assignments
- Create and evaluate quizzes
- Create and manage discussions
- Schedule and conduct virtual sessions
- Send messages
- View learner profiles and progress
- Access analytics and reports

7.3 Admin Role

Administrators have the following permissions:

- All trainer permissions
- Manage users (create, update, disable)
- Approve or reject messages
- Configure system settings
- Access all analytics and reports

7.4 Role Admin

Role administrators have the following permissions:

- All admin permissions
- Manage roles and permissions

- Access system logs and audit trails
- Configure advanced system settings

8. Core Features

8.1 User Management

The system provides comprehensive user management functionality:

- User registration and profile creation
- Role assignment
- Password management
- Two-factor authentication
- User activity tracking

8.2 Program Management

Programs are the core organizational unit in the system:

- Program creation and configuration
- Module organization
- Week-by-week content scheduling
- Progress tracking
- Enrollment management

8.3 Content Management

The system supports various types of training content:

- Videos
- Documents
- Links
- Assignments
- Quizzes
- Virtual sessions
- Discussions

Content is organized by program, week, and type, making it easy for learners to navigate.

8.4 Assignment Management

Assignments allow trainers to assess learner progress:

- Assignment creation with file attachments
- Due date setting
- Submission collection
- Grading and feedback

8.5 Quiz Management

Quizzes provide interactive assessment:

- Quiz creation with multiple-choice questions
- Time limits and scheduling
- Automatic grading
- Result analysis

8.6 Virtual Session Management

Virtual sessions enable real-time interaction:

- Session scheduling
- Platform integration (Zoom, etc.)
- Attendance tracking
- Recording management

8.7 Discussion Forums

Discussions facilitate collaborative learning:

- Topic creation
- Threaded replies
- File attachments
- Moderation

8.8 Messaging System

The messaging system enables communication between users:

- Direct messaging
- File attachments
- Message approval workflow
- Threaded conversations

8.9 Progress Tracking

The system tracks learner progress through various metrics:

- Content views
- Assignment completion
- Quiz scores
- Virtual session attendance
- Overall program progress

9. Frontend Components

9.1 Trainer Portal Components

The trainer portal consists of the following main components:

9.1.1 Trainer Main Page (`trainerMainPage.tsx`)

This is the main dashboard for trainers, providing an overview of their programs, upcoming events, and quick access to key features.

9.1.2 Assignments Page (`assignmentsPage.tsx`)

This component allows trainers to create, view, and grade assignments.

9.1.3 Content Page (`contentPage.tsx`)

This component allows trainers to create, view, and manage various types of content.

9.1.4 Content Modal (`contentModal.tsx`)

This modal component is used for creating and editing content.

9.1.5 Discussion Modal (`discussionModal.tsx`)

This modal component is used for creating and editing discussions.

9.1.6 Edit Profile (`editProfile.tsx`)

This component allows trainers to edit their profile information.

9.1.7 Message Component (`messageComponent.tsx`)

This component displays messages and allows trainers to send new messages.

9.1.8 Message Input (`MessageInput.tsx`)

This component provides an input field for composing messages.

9.1.9 Navbar (`navbar.tsx`)

This component provides navigation for the trainer portal.

9.1.10 Quick Access (`quickAccess.tsx`)

This component provides quick access to frequently used features.

9.1.11 Quiz Creation Page (`quizCreationPage.tsx`)

This component allows trainers to create and edit quizzes.

9.1.12 Quizzes Page (quizzesPage.tsx)

This component displays all quizzes and allows trainers to manage them.

9.1.13 View Profile (viewProfile.tsx)

This component displays the trainer's profile information.

9.1.14 Virtual Session Page (virtualSessionPage.tsx)

This component allows trainers to schedule and manage virtual sessions.

9.2 Website Components

The public website consists of the following main components:

9.2.1 Landing Page (LandingPage.jsx)

This is the main entry point for the website, providing an overview of MPOVR and its programs.

9.2.2 Basic Profile Form (basicProfileForm.tsx)

This component allows users to create a basic profile during the application process.

9.2.3 Contact Page (contactPage.tsx)

This component provides contact information and a contact form.

9.2.4 Enrollment Process (enrollmentProcess.tsx)

This component explains the enrollment process to prospective learners.

9.2.5 FAQ Page (faqPage.tsx)

This component displays frequently asked questions and answers.

9.2.6 Individual Program (indivisualProgram.tsx)

This component displays detailed information about a specific program.

9.2.7 Program Schedule (programSchedule.tsx)

This component displays the schedule for a specific program.

9.2.8 Programs Page (programsPage.tsx)

This component displays all available programs.

9.2.9 Landing Page Components

The landing page is composed of several sub-components:

- `contactSection.tsx`: Contact information section

- `courses.tsx`: Featured courses section
- `faqSection.tsx`: FAQ section
- `featureSection.tsx`: Features section
- `footer.tsx`: Page footer
- `header.tsx`: Page header
- `navbar.tsx`: Navigation bar
- `processSection.tsx`: Process explanation section
- `programTermsSection.tsx`: Program terms section
- `siteMap.tsx`: Site map section
- `statsSection.tsx`: Statistics section

10. Deployment and Configuration

10.1 Environment Variables

The system uses environment variables for configuration, which are stored in the `.env` file:

- `DATABASE_URL`: PostgreSQL connection string
- `SECRET_KEY`: Secret key for JWT token generation
- `ALGORITHM`: Algorithm used for JWT token generation (e.g., HS256)
- `ACCESS_TOKEN_EXPIRE_MINUTES`: JWT token expiration time in minutes
- `UPLOAD_DIRECTORY`: Directory for storing uploaded files
- `STRIPE_API_KEY`: Stripe API key for payment processing
- `DOCUSIGN_API_KEY`: DocuSign API key for digital signatures

10.2 Database Configuration

Database configuration is managed through `alembic.ini` and `database.py`:

- `alembic.ini`: Contains database connection information for migrations
- `database.py`: Sets up the database connection and session management

10.3 Server Configuration

Server configuration is managed through `config.py`:

- API settings
- CORS settings
- Security settings
- File upload settings

10.4 Deployment Process

The recommended deployment process is as follows:

1. Set up a PostgreSQL database
2. Configure environment variables

3. Install dependencies from `requirements.txt`
4. Run database migrations using Alembic
5. Start the FastAPI application using a production ASGI server like Uvicorn or Gunicorn
6. Deploy the React frontend to a static file server or CDN

11. Development Guidelines

11.1 Code Style

The project follows these code style guidelines:

- **Python**: PEP 8 style guide
- **JavaScript/TypeScript**: ESLint with Airbnb configuration
- **SQL**: Uppercase keywords, lowercase identifiers

11.2 Git Workflow

The recommended Git workflow is:

1. Create a feature branch from `main`
2. Develop and test the feature
3. Submit a pull request
4. Review and merge the pull request

11.3 Testing

The project uses the following testing approaches:

- **Backend**: pytest for unit and integration tests
- **Frontend**: Jest and React Testing Library for component tests

11.4 Documentation

All code should be documented according to these guidelines:

- **Python**: Docstrings following the Google style
- **JavaScript/TypeScript**: JSDoc comments
- **API**: OpenAPI/Swagger documentation

12. Troubleshooting

12.1 Common Issues

12.1.1 Database Connection Issues

If the application cannot connect to the database:

1. Verify that the PostgreSQL server is running

2. Check the `DATABASE_URL` environment variable
3. Ensure that the database user has the necessary permissions
4. Check for network connectivity issues

12.1.2 Authentication Issues

If users cannot authenticate:

1. Verify that the `SECRET_KEY` environment variable is set correctly
2. Check that the user's password hash in the database is correct
3. Ensure that the user is not disabled
4. Check for token expiration issues

12.1.3 File Upload Issues

If file uploads are failing:

1. Verify that the `UPLOAD_DIRECTORY` exists and is writable
2. Check file size limits in the FastAPI configuration
3. Ensure that the client is sending the correct content type
4. Check for disk space issues

12.2 Logging

The application uses Python's built-in logging module to log events and errors:

- Error logs are written to `error.log`
- Access logs are written to `access.log`
- Debug logs are written to `debug.log` in development mode

12.3 Monitoring

The application can be monitored using:

- Prometheus for metrics collection
- Grafana for visualization
- Sentry for error tracking