

Contents

[Core API Endpoints 3](#_Toc178770850)

[1. Account Creation (Sign-Up) 3](#_Toc178770851)

[2. Login 4](#_Toc178770852)

[3. Security 5](#_Toc178770853)

[4. Session Management 5](#_Toc178770854)

[5. Educator Dashboard 5](#_Toc178770855)

[6. Educator Page 7](#_Toc178770856)

[7. File Upload 11](#_Toc178770857)

[8. Time Table View 14](#_Toc178770858)

[9. Add Class to Timetable 15](#_Toc178770859)

[10. Moderator Resource Management 17](#_Toc178770860)

[11. System Developer Module Management 20](#_Toc178770861)

[12. System Admin User Management 22](#_Toc178770862)

[13. Quizzes 25](#_Toc178770863)

[Conclusion 27](#_Toc178770864)

# Core API Endpoints

## 1. Account Creation (Sign-Up)

**Endpoint: POST /api/auth/register**

* **Description**: This endpoint allows users to create new accounts on the platform. It requires the user's personal information, such as name, surname, username, email, and password, and assigns a default role of "Student".
* **Parameters**:
  + Name: String (required) – The first name of the user.
  + Surname: String (required) – The last name of the user.
  + Username: String (required) – A unique identifier for the user.
  + Email: String (required) – The email address for user verification.
  + Password: String (required) – A secure password for the account.
  + Role: String (optional) – The role assigned to the user, defaults to "Student" if not specified.
* **Sample Request Body**:
  + "Name": "John",
  + "Surname": "Doe",
  + "Username": "johndoe",
  + "Email": "john@example.com",
  + "Password": "password123",
  + "Role ID": "7"
* **Response**:
  + **Success**:
    - "message": "User registered successfully",
    - "userId": 12345
  + **Error**:
    - **400 Bad Request**: Missing required fields.
    - **409 Conflict**: A user with the same username or email already exists.
* **Error Responses**:
  + 400 Bad Request: Occurs when any required fields are missing (Name, Surname, Username, Email, Password).
  + 409 Conflict: If a user with the same username or email already exists.
  + **Example**:
    - "error": "User already exists"

## 2. Login

**Endpoint: POST /api/auth/login**

* **Description**: This endpoint handles user authentication by verifying the username or email and password combination. Upon successful login, the user is granted a session or token for further requests.
* **Parameters**:
  + Username/Email: String (required) – The username or email address used during registration.
  + Password: String (required) – The password associated with the account.
* **Sample Request Body**:
  + "Username": "johndoe",
  + "Password": "password123"
* **Response**:
  + **Success**:
    - "userId": 12345,
    - "roleId": 2,
    - "message": "Login successful"
  + **Error**:
    - **401 Unauthorized**: Invalid username or password.
    - **400 Bad Request**: If the required fields are not provided.
* **Error Responses**:
  + 401 Unauthorized: If the provided username/email or password is incorrect.
    - **Example**:
      * "error": "Invalid username or password"
* **Error Handling**:
  + **500 Internal Server Error**: If there’s a problem with database connection or query execution.

## 3. Security

Both the account creation and login processes are secured using appropriate hashing and encryption for the password. The platform stores passwords securely, ensuring that user credentials are protected from unauthorized access.

## 4. Session Management

Upon successful login, the UserID and RoleID are stored in the session, which allows users to maintain their authenticated state across multiple pages during the session. This enables role-based access control for educators, moderators, and students.

## 5. Educator Dashboard

**Endpoint: GET /api/dashboard/educator**

* **Description**: This endpoint retrieves a list of available resources and modules for the authenticated educator, along with their personal dashboard information such as welcome message and assigned role.
* **Parameters**:
  + **UserID**: Integer (required) – The ID of the currently logged-in user, used to retrieve user-specific resources.
  + **RoleID**: Integer (required) – The role of the user (Educator, Moderator, etc.), which determines what modules and buttons are visible.
* **Workflow**:
  + The educator is authenticated through their session.
  + Based on the educator's role (RoleID = 6), they will see specific buttons and resources related to their modules.
  + Resources will be filtered to show only those with status\_id = 2 (approved) for educators, moderators, and students.
* **Buttons Available**:
  + **Profile**: Redirects the educator to their profile page.
  + **Logout**: Logs the user out and redirects to the landing page.
  + **Educator's Resources**: Directs the educator to their specific resource page.
* **Response**:
  + **Welcome Message**: A personalized message, e.g., "Welcome, John Doe."
  + **Role Description**: Indicates the user's role, e.g., "Signed in as: Educator."
  + **List of Available Resources**:
    - **ModuleName**: The title of the module associated with the resource.
    - **ResourceTitle**: The title of the educational resource.
    - **ResourceDescription**: A brief description of the resource.
    - **UploadDate**: The date the resource was uploaded.
    - **FileSize**: The size of the resource file.
    - **ResourceStatus**: The approval status of the resource.
    - **AverageRating**: The average rating of the resource based on student feedback (if applicable).
* **Query Logic**:
  + For educators (RoleID = 6), only resources with status\_id = 2 (approved) are displayed.
  + The query ensures that the educator sees only the resources assigned to their modules or approved resources.

**Errors**

* **Session Expired**: If the session is no longer valid (e.g., UserID is null), the system will return an error indicating that the session has expired and prompt the user to log in again.
  + **Error**: "Session has expired. Please log in again."
* **User Not Found**: If the user’s information cannot be retrieved from the database, the system will return an error.
  + **Error**: "User not found."
* **No Resources Found**: If no resources in progress or approved resources are found for the educator, an error message will be displayed.
  + **Error**: "No resources in progress found."
* **Database Connection Error**: If an error occurs while connecting to the database or executing a query, the system will handle the error and display an appropriate message.
  + **Error**: "An error occurred while retrieving resources: [Error Message]"

## 6. Educator Page

**Overview**

The frmEducator page allows educators and higher-level users to manage students by assigning them to modules, viewing their information, and removing them from specific modules. The page is designed to function based on user role and dynamically loads student data for educators or admins. Additionally, it offers functionalities for uploading resources, managing announcements, and FAQs.

**Endpoints & Functionalities**

**1. GET Educator Page (Page\_Load)**

**Endpoint:** /frmEducator.aspx

* **Description:** Loads the Educator Page for authenticated users with appropriate roles. If the user is not logged in, or their role is insufficient, they are redirected to other pages.
* **Roles Allowed:** Educator (RoleID: 6), Admin (RoleID: other higher roles)
* **Responses:**
  + 200 OK: Educator page loaded successfully.
  + 302 Redirect: Redirects to:
    - frmDashboard.aspx (if UserID < 7).
    - frmLandingPage.aspx (if the user is not logged in).

**2. GET Module List for Educator**

**Endpoint:** /frmEducator.aspx?action=populateModules

* **Description:** Retrieves a list of modules either specific to the educator or accessible by higher-level users (e.g., admins). This populates the module filter in the form of a RadioButtonList.
* **Roles Allowed:** Educators, Admins.
* **Database Query:**
  + For Educators: Retrieves only the modules they are assigned to.
  + For Admins: Retrieves all modules.
* **Responses:**
  + 200 OK: Module list loaded successfully and rendered in rblStudentFilter.
  + 404 Not Found: No modules found for the educator.
  + 500 Server Error: Error while retrieving modules.

**3. GET Students for Module**

**Endpoint:** /frmEducator.aspx?action=loadStudents&moduleTitle={moduleTitle}

* **Description:** Fetches the list of students assigned to the selected module. Optionally filters the student list by a search term.
* **Parameters:**
  + moduleTitle: The title of the module selected by the educator.
  + filter: (optional) Search term for filtering the student list by name or surname.
* **Roles Allowed:** Educators, Admins.
* **Database Query:**
  + Retrieves student data, including full name, username, email, and associated module.
* **Responses:**
  + 200 OK: Students for the selected module loaded successfully and displayed in dgvStudents.
  + 404 Not Found: No students found for the module.
  + 500 Server Error: Error while retrieving students for the module.

**4. GET Unassigned Students**

**Endpoint:** /frmEducator.aspx?action=loadUnassignedStudents

* **Description:** Retrieves students who are not assigned to any module.
* **Roles Allowed:** Educators, Admins.
* **Database Query:**
  + Retrieves unassigned students based on the role\_id = 7 (students).
* **Responses:**
  + 200 OK: Unassigned students loaded successfully and displayed in dgvStudents.
  + 404 Not Found: No unassigned students found.
  + 500 Server Error: Error while retrieving unassigned students.

**5. POST Remove Student from Module**

**Endpoint:** /frmEducator.aspx?action=removeStudent&studentId={studentId}&moduleId={moduleId}

* **Description:** Removes a student from a specific module.
* **Parameters:**
  + studentId: The ID of the student to remove.
  + moduleId: The ID of the module from which the student is to be removed.
* **Roles Allowed:** Educators, Admins.
* **Database Query:**
  + Deletes the student-module relationship in the Stu\_To\_Module table.
* **Responses:**
  + 200 OK: Student successfully removed from the module.
  + 404 Not Found: Student or module not found.
  + 500 Server Error: Error while removing student from the module.

**6. POST Upload Resources**

**Endpoint:** /frmUploadFiles.aspx

* **Description:** Redirects to the file upload page where educators can upload resources for students.
* **Roles Allowed:** Educators, Admins.
* **Responses:**
  + 302 Redirect: Redirect to the file upload page.

**7. POST Add Announcement**

**Endpoint:** /frmAddAnnouncement.aspx

* **Description:** Redirects to the announcement creation page where educators or admins can add announcements.
* **Roles Allowed:** Educators, Admins.
* **Responses:**
  + 302 Redirect: Redirect to the add announcement page.

**8. POST Add FAQ**

**Endpoint:** /frmAddFAQ.aspx

* **Description:** Redirects to the FAQ creation page where educators or admins can add new FAQs.
* **Roles Allowed:** Educators, Admins.
* **Responses:**
  + 302 Redirect: Redirect to the add FAQ page.

**9. POST Search Students (txtSearchStudents\_TextChanged)**

**Endpoint:** /frmEducator.aspx?action=searchStudents&moduleTitle={moduleTitle}&searchTerm={searchTerm}

* **Description:** Searches for students based on a search term within the selected module. The result is displayed in the dgvStudents.
* **Parameters:**
  + moduleTitle: The title of the selected module.
  + searchTerm: The term to search for (can match name or surname).
* **Roles Allowed:** Educators, Admins.
* **Database Query:**
  + Filters students based on the search term and selected module.
* **Responses:**
  + 200 OK: Filtered student list displayed in dgvStudents.
  + 404 Not Found: No students found matching the search term.
  + 500 Server Error: Error while filtering students.

**Error Handling**

All queries and actions return appropriate messages on encountering an error.

* SqlException: Displays a message indicating a database error.
* Exception: Displays a message indicating a general application error.

## 7. File Upload

**Overview**

The frmUploadFiles.aspx page allows educators to upload files related to specific modules. Uploaded files can include images or other resource types. For image files under 1MB, a watermark is automatically applied using an external watermarking API. The files are stored in the database, along with metadata (file path, type, and size) and are associated with the selected module.

**Endpoints & Functionalities**

1. **GET File Upload Page**
   * **Endpoint**: GET /frmUploadFiles.aspx
   * **Description**: Loads the file upload page for authenticated users who have sufficient roles. Users who are not logged in or do not have the required role are redirected to the dashboard.
   * **Roles Allowed**: Educator (RoleID 4 or higher)
   * **Responses**:
     + 200 OK: File upload page loaded successfully.
     + 302 Redirect: Redirects to frmDashboard.aspx if the user is not authorized or not logged in.
2. **GET Module List for Educator**
   * **Endpoint**: GET /frmUploadFiles.aspx?action=loadModules
   * **Description**: Retrieves the list of modules for which the educator can upload resources.
   * **Roles Allowed**: Educators (RoleID 4 or higher)
   * **Database Query**:
     + If RoleID < 4 (Admins): Retrieves all modules.
     + If RoleID >= 4 (Educators): Retrieves only the modules assigned to the educator.
   * **Responses**:
     + 200 OK: Module list loaded successfully and displayed in the ModuleDropDown.
     + 404 Not Found: No modules found for the educator.
     + 500 Server Error: Error while retrieving modules.
3. **POST Upload File**
   * **Endpoint**: POST /frmUploadFiles.aspx?action=uploadFile
   * **Description**: Uploads a file to the server and associates it with the chosen module. If the file is an image and smaller than 1MB, it is watermarked using the external watermarking API.
   * **Parameters**:
     + FileUploadControl: The file to upload.
     + ModuleDropDown: The selected module.
     + txtTitle: The title of the resource.
     + txtDesc: The description of the resource.
   * **Roles Allowed**: Educators
   * **Steps**:
     + **File Validation**:
       - Ensure a file is selected.
       - Verify the selected module ID from ModuleDropDown.
     + **File Type Check**:
       - Image files (.jpg, .jpeg, .png, .gif) are checked for size (must be under 1MB).
       - Other file types are uploaded without additional processing.
     + **Watermarking**:
       - If the file is an image under 1MB, the external watermark API is called to apply a watermark.
       - The original image is overwritten with the watermarked version.
     + **Database Insertion**:
       - Insert resource details (user\_id, module\_id, title, description) into the Resources table.
       - Insert file details (resource\_id, file\_path, file\_type, file\_size) into the Files table.
   * **Responses**:
     + 200 OK: File uploaded successfully, and watermarked if applicable.
     + 400 Bad Request: Invalid module selected or file not uploaded.
     + 413 Payload Too Large: The uploaded file exceeds the size limit.
     + 500 Server Error: Error during file upload or database insertion.
4. **GET Check Image Accessibility**
   * **Endpoint**: GET /frmUploadFiles.aspx?action=checkImage&url={imageUrl}
   * **Description**: Validates whether the image at the provided URL is accessible.
   * **Parameters**:
     + imageUrl: The URL of the image to check.
   * **Roles Allowed**: Educators
   * **Responses**:
     + 200 OK: Image is accessible.
     + 404 Not Found: Image is not accessible.

**Error Handling**

* **SqlException**: Displays a message indicating a database error occurred.
* **HttpRequestException**: Displays a message indicating an issue with the HTTP request to the watermark API.
* **General Exception**: Displays any general errors encountered during file upload or module retrieval.

## 8. Time Table View

**Endpoint**: GET /api/timetable/view

* **Description**: This endpoint retrieves the user's timetable, listing all scheduled classes based on the user ID.
* **Parameters**:
  + **UserId**: Integer (required) – The ID of the user whose timetable is being requested.
* **Response**:
  + **Success**:
    - **200 OK**: Returns a list of timetable entries for the user.
      * **Fields**:
        + timeTableId: Integer – The unique ID of the timetable entry.
        + userId: Integer – The ID of the user.
        + moduleId: Integer – The ID of the module.
        + dayId: Integer – The ID representing the day of the week.
        + classStartTime: String – The class start time (HH

format).

* + - * + classEndTime: String – The class end time (HH

format).

* + **Error**:
    - **400 Bad Request**: Missing or invalid parameters.
      * **Description**: Occurs if the required fields, such as UserId, are not provided or are invalid.
    - **500 Internal Server Error**: Occurs if there's an issue with the database connection or query execution.

## 9. Add Class to Timetable

**Endpoint**: POST /api/timetable/add

* **Description**: This endpoint allows students to add a new class to their time table by selecting a module, day, and time range.
* **Parameters**:
  + **UserId**: Integer (required) – The ID of the student adding the class.
  + **ModuleId**: Integer (required) – The ID of the module to which the class belongs.
  + **DayId**: Integer (required) – The ID representing the day of the week.
  + **ClassStartTime**: String (required) – The start time of the class (HH

format).

* + **ClassEndTime**: String (required) – The end time of the class (HH

format).

* **Validations**:
  + Start time and end time must be in HH:mm format.
  + End time must be after the start time.
* **Response**:
  + **Success**:
    - **201 Created**: Class added successfully.
      * **Fields**:
        + timeTableId: Integer – The unique ID of the newly created time table entry.
  + **Error**:
    - **400 Bad Request**: Missing or invalid parameters.
      * **Description**: Occurs if required fields are not provided or the time format is invalid.
    - **500 Internal Server Error**: Occurs if there's an issue with the database connection or query execution.

**Error Handling:**

* **500 Internal Server Error**: If there’s a problem with the database connection or query execution, an internal server error is returned.
  + **Resolution**: Ensure the database is running correctly and connections are properly handled.

## 10. Moderator Resource Management

**Get In-Progress Resources**

**Endpoint:** GET /api/moderator/resources/in-progress

**Description:** Retrieves a list of resources with a status of "In Progress" (status\_id = 3).

**Parameters:**

* **filter** (optional): String - Filter resources by module title or resource title.

**Sample Request Query:** ?filter=example

**Response:**

* **Success:**
  + **resources**: List of resource objects
    - **resource\_id**: Integer
    - **moduleTitle**: String
    - **resourceTitle**: String
    - **description**: String
    - **uploadDate**: DateTime
    - **fileType**: String
    - **fileSize**: Integer
* **Error:**
  + **404 Not Found**: No resources found.

**Error Responses:**

* **400 Bad Request**: Invalid request.
* **500 Internal Server Error**: Server error.

**Update Resource Status**

**Endpoint:** PATCH /api/moderator/resources/{resourceId}/status

**Description:** Updates the status of a resource.

**Parameters:**

* **resourceId** (required): Integer - The ID of the resource.
* **statusId** (required): Integer - The new status ID (1 = Rejected, 2 = Approved).
* **feedback** (required): String - Moderator feedback.

**Sample Request Parameters:**

* resourceId=123
* statusId=2
* feedback=Example+feedback

**Response:**

* **Success:**
  + **message**: "Resource status updated successfully"
* **Error:**
  + **400 Bad Request**: Invalid request.
  + **404 Not Found**: Resource not found.
  + **500 Internal Server Error**: Server error.

**Search In-Progress Resources**

**Endpoint:** GET /api/moderator/resources/in-progress/search

**Description:** Searches for in-progress resources by module title or resource title.

**Parameters:**

* **searchTerm** (required): String - Search term.

**Sample Request Query:** ?searchTerm=example

**Response:**

* **Success:**
  + **resources**: List of resource objects (same as Get In-Progress Resources)
* **Error:**
  + **404 Not Found**: No resources found.

**Select Resource**

**Endpoint:** POST /api/moderator/resources/select

**Description:** Selects a resource for moderation.

**Parameters:**

* **resourceId** (required): Integer - The ID of the resource.

**Sample Request Parameters:**

* resourceId=123

**Response:**

* **Success:**
  + **message**: "Resource selected successfully"
* **Error:**
  + **400 Bad Request**: Invalid request.
  + **404 Not Found**: Resource not found.

**Approve or Reject Resource**

**Endpoint:** PATCH /api/moderator/resources/{resourceId}/approve-reject

**Description:** Approves or rejects a selected resource.

**Parameters:**

* **resourceId** (required): Integer - The ID of the resource.
* **approveReject** (required): Integer - 1 = Reject, 2 = Approve.

**Sample Request Parameters:**

* resourceId=123
* approveReject=2

**Response:**

* **Success:**
  + **message**: "Resource approved/rejected successfully"
* **Error:**
  + **400 Bad Request**: Invalid request.
  + **404 Not Found**: Resource not found.
  + **500 Internal Server Error**: Server error.

## 11. System Developer Module Management

**1. Add Module**

* **Endpoint:** /api/modules
* **Method:** POST
* **Description:** Creates a new module.
* **Request Body:**
  + title (string): Module title.
  + description (string): Module description.
  + educatorId (integer): Educator ID.
* **Response:**
  + **201 Created:** Module added successfully.
  + **400 Bad Request:** Invalid request.
  + **500 Internal Server Error:** Server error.

**2. Get Modules**

* **Endpoint:** /api/modules
* **Method:** GET
* **Description:** Retrieves a list of modules.
* **Response:**
  + **200 OK:** List of module objects.
  + **404 Not Found:** No modules found.

**3. Get Module by ID**

* **Endpoint:** /api/modules/{moduleId}
* **Method:** GET
* **Description:** Retrieves a module by ID.
* **Path Parameters:**
  + moduleId (integer): Module ID.
* **Response:**
  + **200 OK:** Module object.
  + **404 Not Found:** Module not found.

**4. Update Module**

* **Endpoint:** /api/modules/{moduleId}
* **Method:** PATCH
* **Description:** Updates a module.
* **Path Parameters:**
  + moduleId (integer): Module ID.
* **Request Body:**
  + title (string): Module title.
  + description (string): Module description.
  + educatorId (integer): Educator ID.
* **Response:**
  + **200 OK:** Module updated successfully.
  + **400 Bad Request:** Invalid request.
  + **404 Not Found:** Module not found.

**5. Delete Module**

* **Endpoint:** /api/modules/{moduleId}
* **Method:** DELETE
* **Description:** Deletes a module and related records.
* **Path Parameters:**
  + moduleId (integer): Module ID.
* **Response:**
  + **200 OK:** Module deleted successfully.
  + **404 Not Found:** Module not found.

**Educator Management Endpoints**

**1. Get Educators**

* **Endpoint:** /api/educators
* **Method:** GET
* **Description:** Retrieves a list of educators.
* **Response:**
  + **200 OK:** List of educator objects.
  + **404 Not Found:** No educators found.

**2. Get Educator by ID**

* **Endpoint:** /api/educators/{educatorId}
* **Method:** GET
* **Description:** Retrieves an educator by ID.
* **Path Parameters:**
  + educatorId (integer): Educator ID.
* **Response:**
  + **200 OK:** Educator object.
  + **404 Not Found:** Educator not found.

**Error Responses**

* **400 Bad Request:** Invalid request.
* **404 Not Found:** Resource not found.
* **500 Internal Server Error:** Server error.

## 12. System Admin User Management

**1. Get Users**

* **Endpoint:** /api/users
* **Method:** GET
* **Description:** Retrieves a list of users.
* **Response:**
  + **200 OK:** List of user objects.
  + **404 Not Found:** No users found.

**2. Get User by ID**

* **Endpoint:** /api/users/{userId}
* **Method:** GET
* **Description:** Retrieves a user by ID.
* **Path Parameters:**
  + userId (integer): User ID.
* **Response:**
  + **200 OK:** User object.
  + **404 Not Found:** User not found.

**3. Delete User**

* **Endpoint:** /api/users/{userId}
* **Method:** DELETE
* **Description:** Deletes a user.
* **Path Parameters:**
  + userId (integer): User ID.
* **Response:**
  + **200 OK:** User deleted successfully.
  + **404 Not Found:** User not found.

**4. Update User Role**

* **Endpoint:** /api/users/{userId}/role
* **Method:** PATCH
* **Description:** Updates a user's role.
* **Path Parameters:**
  + userId (integer): User ID.
* **Request Body:**
  + roleId (integer): Role ID.
* **Response:**
  + **200 OK:** User role updated successfully.
  + **400 Bad Request:** Invalid request.
  + **404 Not Found:** User not found.

**Role Management Endpoints**

**1. Get Roles**

* **Endpoint:** /api/roles
* **Method:** GET
* **Description:** Retrieves a list of roles.
* **Response:**
  + **200 OK:** List of role objects.
  + **404 Not Found:** No roles found.

**Authentication Endpoints**

**1. Authenticate**

* **Endpoint:** /api/authenticate
* **Method:** POST
* **Description:** Authenticates a user.
* **Request Body:**
  + username (string): Username.
  + password (string): Password.
* **Response:**
  + **200 OK:** Authentication successful.
  + **401 Unauthorized:** Invalid credentials.

**Error Responses**

* **400 Bad Request:** Invalid request.
* **401 Unauthorized:** Unauthorized access.
* **404 Not Found:** Resource not found.
* **500 Internal Server Error:** Server error.

## 13. Quizzes

**1. Get Quiz by ID**

* **Endpoint:** /api/quizzes/{quizId}
* **Method:** GET
* **Description:** Retrieves a quiz by ID.
* **Path Parameters:**
  + quizId (integer): Quiz ID.
* **Response:**
  + **200 OK:** Quiz object.
  + **404 Not Found:** Quiz not found.

**2. Get Quiz Questions**

* **Endpoint:** /api/quizzes/{quizId}/questions
* **Method:** GET
* **Description:** Retrieves quiz questions.
* **Path Parameters:**
  + quizId (integer): Quiz ID.
* **Response:**
  + **200 OK:** List of question objects.
  + **404 Not Found:** Quiz or questions not found.

**3. Submit Quiz Responses**

* **Endpoint:** /api/quizzes/{quizId}/responses
* **Method:** POST
* **Description:** Submits quiz responses.
* **Path Parameters:**
  + quizId (integer): Quiz ID.
* **Request Body:**
  + responses (array): List of response objects.
    - questionId (integer): Question ID.
    - answer (string): User's answer.
* **Response:**
  + **201 Created:** Quiz responses submitted successfully.
  + **400 Bad Request:** Invalid request.

**Question Management Endpoints**

**1. Get Question by ID**

* **Endpoint:** /api/questions/{questionId}
* **Method:** GET
* **Description:** Retrieves a question by ID.
* **Path Parameters:**
  + questionId (integer): Question ID.
* **Response:**
  + **200 OK:** Question object.
  + **404 Not Found:** Question not found.

**2. Get Question Type**

* **Endpoint:** /api/questions/{questionId}/type
* **Method:** GET
* **Description:** Retrieves question type.
* **Path Parameters:**
  + questionId (integer): Question ID.
* **Response:**
  + **200 OK:** Question type (MCQ, FillBlank, LongForm).
  + **404 Not Found:** Question not found.

**Error Responses**

* **400 Bad Request:** Invalid request.
* **404 Not Found:** Resource not found.
* **500 Internal Server Error:** Server error.

**Authentication**

* All endpoints require authentication.
* User ID is stored in Session["UserID"].

**Authorization**

* Only authorized users can access quiz and question endpoints.
* Quiz submission requires valid user ID and quiz ID.

# Conclusion

The API design in Siyafunda ensures secure, scalable, and modular functionality across various use cases from user authentication to resource management. Forming the backbone of the entire system.