IT Software Solutions for Business
LKS Provinsi Banten 2025

Growth Seeker Academy

Contents

Application Overview

Growth Seeker Academy is on a mission to empower ambitious high-schoolers across Indonesia with world-class, expert-led online courses. Our client—an ed-tech startup founded by veteran educators—wants a rock-solid backend so that students can sign up, browse courses, apply coupons, and securely complete purchases.

In this competition scenario, you'll play the role of Growth Seeker's lead backend developer. Over the next three hours, you'll build a RESTful API (project name API_XX) on top of a provided SQL Server schema. Your endpoints will handle:

- User onboarding (registration, login with JSON Web Tokens, logout)
- Course catalog (listing, detail views)
- Purchases (with optional coupon discounts)
- Admin tools (managing coupons and courses)
- Transaction history (role-based visibility for students vs. admins)

Your work will directly enable Growth Seeker's back-end team to deliver a seamless, secure learning marketplace—and help high-school students everywhere seize new growth opportunities.

Instruction

- The client requests you as a developer to develop the APIs based on the specification below
- The table structure and data sample will be provided as SQL Server script and need to be imported and connected manually.
- The time limit will be 3 hours.
- Your Project should be named API_XX where XX is your PC number
- Code properly based on the description of the API and its specific requirement.
- Use **Base URL** format**:**
  `https://<localhost>/gsa-api/v1`

1. Authentication Endpoints

1.1. User Registration

Endpoint: `/users/register`
Method: POST
Description: Registers a new user. The default role value is `"student"` (admin role is set later in the database).

Requirement:

- Validate that the following fields are present:
  - username,
  - email,
  - full name,
  - password.
- Email must be in proper format and unique (cannot register when it already exists).
- Password must follow the required format (minimum 8 characters with an uppercase, a lowercase, a symbol, and a number)
- When successfully saved to DB, hash the password and set role attribute into "student".

Hash Method

```
using (SHA256 sha256 = SHA256.Create())
{
    byte[] inputBytes = Encoding.UTF8.GetBytes(password);
    byte[] hashBytes = sha256.ComputeHash(inputBytes);

    StringBuilder sb = new StringBuilder();
    foreach (byte b in hashBytes)
    {
        sb.Append(b.ToString("x2"));
    }

    return sb.ToString();
}
```

Request Body format:

```
{
  "username": "john_doe",
  "fullName": "John Doe",
  "email": "john@example.com",
  "password": "Pass123!",
}
```

Success Response (200):

```
{
  "message": "User registered successfully.",
}
```

Possible Error Responses:

- **Validation Error (422):**

  ```
  {
    "message": "Validation error: email is invalid."
  }
  ```

1.2. User Login

Endpoint: `/users/login`
Method: POST
Description: Authenticates a user (student or admin) using email and password. The JWT token returned embeds the user's role.

Requirement:

- Validate that the following fields are present:
  - email,
  - password.
- Email must be in proper format.
- Validate the credential on the DB, check the hashed password.
- When it authorizes, return the JWT Token that expires in 7 days and the appropriate role.
- The default password for existing users is `Pass123!` , sample user email for student could use `alice@example.com` and for admin could use `admin@growthseeker.com`

Request Body Example:

```
{
  "email": "bob@example.com",
  "password": "Pass123!"
}
```

Success Response (200):

```
{
  "message": "Login successful.",
  "data": {
    "userId": 1,
    "username": "john_doe",
    "role": "student",
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}
```

Possible Error Responses:

- **Validation Error (422):**

```
{
  "message": "Validation error: password must be at least 8
characters."
}
```

- **Unauthorized (401):**

```
{
  "message": "Invalid email or password."
}
```

1.3. Logout

Endpoint: `/users/logout`
Method: POST
Description: Invalidates the JWT token to log out the user.

Headers:
`Authorization: Bearer <token>`

Success Response Example:

```
{
  "message": "Logout successful."
}
```

Possible Error Responses:

- **Unauthorized (401):**

```
{
  "message": "Authorization token missing or invalid."
}
```

---

2. Course Management Endpoints

2.1. Get List of Courses

Endpoint: `/courses`
Method: GET
Description: Retrieves a list of available courses based on database with its search filter and pagination.

Query Parameters (Optional):

- `title`: Filter courses by course title (partial match).
- `sort`: sort data based on course creation (default will be descending).
- `page`: Page number for pagination (default value is 1).
- `size`: Number of courses per page (default value is 10).

Success Response (200):

```
{
  "data": [
    {
      "id": 1,
      "title": "Introduction to C#",
      "description": "Learn the basics of C# programming.",
      "price": 99.99
    },
    {
      "id": 2,
      "title": "Advanced Web Development",
      "description": "Deep dive into modern web technologies.",
      "price": 149.99
    }
  ],
  "pagination": {
    "page": 1,
    "size": 10,
    "totalPages": 5
  }
}
```

Possible Error Response (422):

- **Validation Error (422):**

  ```
  {
    "message": "Validation error: 'page' must be a positive integer."
  }
  ```

2.2. Get Course Details

Endpoint: `/courses/{courseId}`
Method: GET
Description: Retrieves detailed information from database about a specific course based on the `courseId` parameter

Path Parameter:

- `courseId` (numeric)

Success Response (200):

```
{
  "data": {
    "id": 1,
    "title": "Introduction to C#",
    "description": "Learn the basics of C# programming including syntax and
OOP concepts.",
    "price": 99.99,
    "duration": "300 minutes",
    "modules": [
      "Getting Started",
      "Data Types",
      "Control Structures"
    ]
  }
}
```

Possible Error Responses:

- **Not Found (404):**

  ```
  {
    "message": "Course not found."
  }
  ```

- **Validation Error (422):**

  ```
  {
    "message": "Validation error: courseId must be numeric."
  }
  ```

2.3. Purchase a Course with Coupon

Endpoint: `/courses/{courseId}/purchase`
Method: POST
Description: Processes the purchase of a course by a user that logs in. An optional coupon code may be applied.

Requirement:

- Check the purchase data properly, userId (retrieved from active login user) and payment method should be mandatory
- Payment Method that accepted only debit Card, Credit Card, or Paypal
- When Coupon is applied, make sure the coupon available when check out based on its expiry date and quota
- Upon successful checkout, display the final price, computed by deducting the discount value from the original course price.

Headers: `Authorization: Bearer <token>`

Path Parameter:

- `courseId` (numeric)

Request Body Example:

```
{
  "paymentMethod": "credit_card",
  "couponCode": "SAVE20"
}
```

Success Response (200):

```
{
  "message": "Course purchased successfully.",
  "data": {
    "purchaseId": "123456",
    "courseId": 1,
    "userId": 1,
    "purchaseDate": "2025-03-25T12:34:56Z",
    "paymentMethod": "credit_card",
    "originalPrice": 99.99,
    "discountApplied": 20.0,
    "paidAmount": 79.99
  }
}
```

Possible Error Responses:

- **Forbidden (403):**

```
{
  "message": "Please login/register to purchase the course"
}
```

- **Validation Error (422):**

```
{
  "message": "Validation error: coupon code has expired or quota
exceeded."
}
```

---

3. Coupon & Admin Course Management Endpoints

Note: Admin operations use the same endpoints as public ones. Operations that modify coupons or courses require the user's role (from the JWT) to be `"admin"`. Non-admin attempts result in a 403 Forbidden error.

3.1. Coupon Management

3.1.1 List of Coupons

Endpoint: `/coupons`
Method: GET
Description: Retrieves a list of available coupons based on database sort by expiry date descending.

Success Response Example:

```
{
  "data": [
    {
        "couponId": 1,
        "couponCode": "SPRINGSALE",
        "discountValue": 15.0,
        "expiryDate": "2025-04-30T23:59:59Z",
        "quota": 100
    },
    {
        "couponId": 2,
        "couponCode": "SAVE20",
        "discountValue": 20.0,
        "expiryDate": "2025-04-30T23:59:59Z",
        "quota": 100
    }
  ]
}
```

Possible Error Responses:

- **Validation Error (422):**

```
{
  "message": "Validation error: 'page' must be a positive integer."
}
```

3.1.2 Create a Coupon

Endpoint: `/coupons`
Method: POST
Description: Allows an admin to add a new coupon.

Requirement:

- Only admin roles that authorize access to this endpoint.
- The required attribute based on request body example and in the proper data type
- Coupon Code must be unique.
- Quota and discount value must be higher than 0.
- The Expiration date must be greater than today's date.

Headers:

```
Authorization: Bearer <token>
```

Request Body Example:

```
{
  "couponCode": "SPRINGSALE",
  "discountValue": 15.0,
  "expiryDate": "2025-04-30T23:59:59Z",
  "quota": 100
}
```

Success Response Example:

```
{
  "message": "Coupon created successfully.",
  "data": {
    "couponId": 1,
    "couponCode": "SPRINGSALE",
    "discountValue": 15.0,
    "expiryDate": "2025-04-30T23:59:59Z",
    "quota": 100
  }
}
```

Possible Error Responses:

- **Unauthorized (401):**

```
{
  "message": "Authorization token missing or invalid."
}
```

- **Forbidden (403):**

```
{
  "message": "Access denied. Admin role required."
}
```

- **Validation Error (422):**

```
{
  "message": "Validation error: quota must be a positive integer."
}
```

3.1.3 Update a Coupon

Endpoint: `/coupons/{couponId}`
Method: PUT
Description: Allows an admin to update coupon details (expiry date, quota, activation status).

Requirement:

- Only admin roles that authorize access to this endpoint.
- Only update the attributes that changed
- Coupon Code must be unique.
- Quota and discount value must higher than 0 and numeric
- Expiry date must be greater than today's date

Headers:
`Authorization: Bearer <token>`

Path Parameter:

- `couponId` (numeric)

Request Body Example:

```
{
  "couponCode": "SPRINGSALE",
  "discountValue": 15.0,
  "expiryDate": "2025-05-15T23:59:59Z",
  "quota": 150
}
```

Success Response (200):

```
{
  "message": "Coupon updated successfully.",
  "data": {
    "couponId": 1,
    "couponCode": "SPRINGSALE",
    "discountValue": 15.0,
```

```
    "expiryDate": "2025-05-15T23:59:59Z",
    "quota": 150
  }
}
```

Possible Error Responses:
(Same as Create a Coupon)

3.2. Admin Course Management

3.2.1 Add New Course

Endpoint: `/courses`
Method: POST
Description: Allows an admin to create a new course and modules.

Requirement:

- Only admin roles that authorize access to this endpoint.
- The required attribute based on request body example and in the proper data type
- Modules at least 3 data
- price and duration must be integer and more than 0. (duration value will be in minutes)

Headers:
`Authorization: Bearer <token>`

Request Body Example:

```
{
  "title": "Mastering JavaScript",
  "description": "A comprehensive course on JavaScript.",
  "price": 129.99,
  "duration": 300,
  "modules": [
    "Introduction",
    "Advanced Topics",
    "Project Work"
  ]
}
```

Success Response (200):

```
{
  "message": "Course created successfully.",
  "data": {
    "courseId": 3,
    "title": "Mastering JavaScript",
    "description": "A comprehensive course on JavaScript.",
```

```
    "price": 129.99,
    "duration": "300 minutes",
    "modules": [
      "Introduction",
      "Advanced Topics",
      "Project Work"
    ]
  }
}
```

Possible Error Responses:

- **Unauthorized (401):**

  ```
  {
    "message": "Access denied. login required."
  }
  ```

- **Forbidden (403):**

  ```
  {
    "message": "Access denied. Admin role required."
  }
  ```

- **Validation Error (422):**

  ```
  {
    "message": "Validation error: price must be numeric."
  }
  ```

3.2.2 Update Course

Endpoint: `/courses/{courseId}`
Method: PUT
Description: Allows an admin to update an existing course and the related modules

Requirement:

- Only admin roles that authorize access to this endpoint.
- Only update the attributes that changed
- Modules at least 3 data
- price and duration must be integer and more than 0. (duration value will be in minutes)

Headers:
`Authorization: Bearer <token>`

Path Parameter:

- `courseId` (numeric)

Request Body Example:

```
{
  "title": "Mastering JavaScript - Updated",
  "description": "Updated description with new content.",
  "price": 119.99,
  "instructor": "Alice Smith",
  "duration": "160",
  "modules": [
    "Introduction",
    "Advanced Topics",
    "New Module"
  ],
  "active": true
}
```

Success Response (200):

```
{
  "message": "Course updated successfully.",
  "data": {
    "courseId": 3,
    "title": "Mastering JavaScript - Updated",
    "description": "Updated description with new content.",
    "price": 119.99,
    "duration": "160",
    "modules": [
      "Introduction",
      "Advanced Topics",
      "New Module"
    ]
  }
}
```

Possible Error Responses:
(Same as for Add New Course)

---

4. Transaction History Endpoints

These endpoints allow students and admins to view purchase transaction history.
student users see only their own transactions.
Admin users can view all transactions (and filter by user email).

4.1. Get Transaction History

Endpoint: `/transactions`
Method: GET
Description: Retrieves transaction history.

- For **student users**: Returns only the transactions for the authenticated user (filter by `courseName`, sort by purchase date).
- For **admin users**: Returns all transactions. Admins can filter by `userEmail` and `courseName`.

Headers:
`Authorization: Bearer <token>`

Query Parameters (Optional):

- `courseName`: Filter transactions by course name.
- `sortBy`: (For students) Sort by purchase date (`asc` or `desc`) (default value is `asc`).
- `userEmail`: (For admins) Filter transactions by user email.
- `page`: current active page (default value is 1)
- `size`: maximum size per page (default value is 10)

Success Response (200 - Student):

```
{
  "data": [
    {
      "transactionId": 1,
      "courseTitle": "Introduction to C#",
      "purchaseDate": "2025-03-25T12:34:56Z",
      "amount": 99.99,
      "couponCode": "SAVE20"
      "paidAmount": 79.99
    },
    {
      "transactionId": 2,
      "courseId": 202,
      "courseTitle": "Advanced Web Development",
      "purchaseDate": "2025-04-01T09:12:34Z",
      "amount": 149.99,
      "couponCode": ""
      "paidAmount": 149.99
    }
  ],
```

```
  "pagination": {
    "page": 1,
    "size": 10,
    "totalPages": 5
  }

}
```

Success Response (200 - Admin):

```
{
  "data": [
    {
      "transactionId": 1,
      "userEmail": "john@example.com",
      "courseTitle": "Introduction to C#",
      "purchaseDate": "2025-03-25T12:34:56Z",
      "amount": 99.99,
      "couponCode": "SAVE20"
      "paidAmount": 79.99
    },
    {
      "transactionId": 3,
      "userEmail": "jane@example.com",
      "courseId": 203,
      "courseTitle": "Mastering JavaScript",
      "purchaseDate": "2025-04-02T15:20:10Z",
      "amount": 129.99,
      "couponCode": "SPRINGSALE"
      "paidAmount": 110.49
    }
  ],
  "pagination": {
    "page": 1,
    "size": 10,
    "totalPages": 5
  }
}
```

Possible Error Responses:

- **Unauthorized (401):**

  ```
  {
    "message": "Authorization token missing or invalid."
  }
  ```

- **Validation Error (422):**

```
{
  "message": "Validation error: sortBy must be 'asc' or 'desc'."
}
```

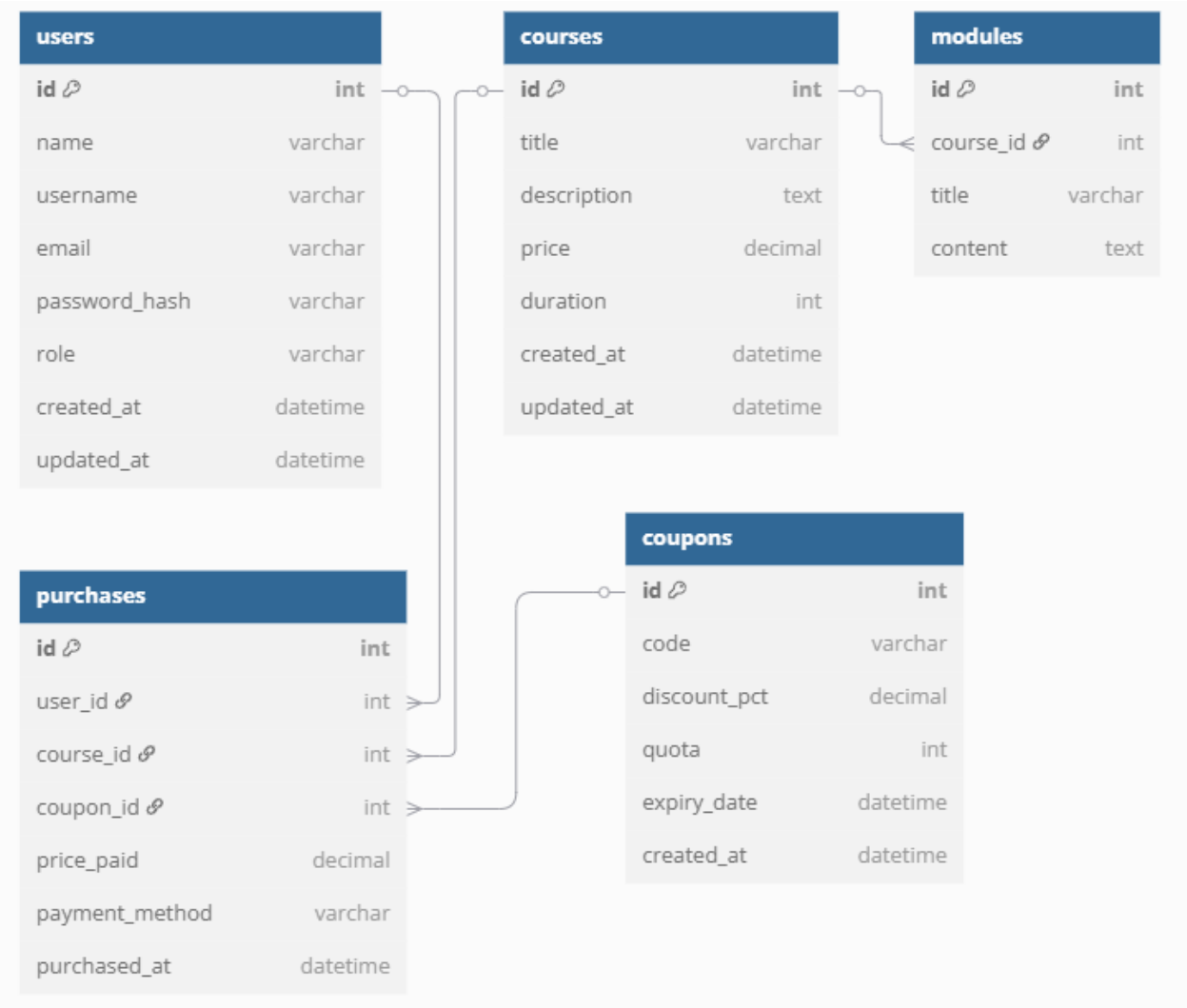
Error Code Summary

- **401 Unauthorized:** Returned when the authorization token is missing or invalid.
- **403 Forbidden:** Returned when a non-admin user attempts an admin-only operation.
- **422 Validation Error:** Returned when required fields are missing, improperly formatted, or do not meet business rules.
- **404 Not Found:** Returned when a requested resource (e.g., course) does not exist.

Table-Glossary & ERD

Below are the database table definitions used in this competition.

ERD



Users

Column Name	Data Type	Description
`id`	INT	Primary key
`name`	VARCHAR	Full name of the user
`username`	VARCHAR	username
`email`	VARCHAR	Unique email
`password_hash`	VARCHAR	Hashed password
`role`	ENUM	Role: `'admin'` or `'student'`

Column Name	Data Type	Description
created_at	TIMESTAMP	Account creation time
updated_at	TIMESTAMP	Last update time

Courses

Column Name	Data Type	Description
id	INT	Primary key
title	VARCHAR	Title of the course
description	TEXT	Detailed description of the course
price	DECIMAL	Price of the course
duration	INT	Duration in minutes
created_at	TIMESTAMP	Course creation time
updated_at	TIMESTAMP	Last update time

Modules

Column Name	Data Type	Description
id	INT	Primary key
course_id	INT	FK to Courses
title	VARCHAR	Title of the module
content	TEXT	Module content (can be video/text)

Coupons

Column Name	Data Type	Description
id	INT	Primary key
code	VARCHAR	Unique coupon code
discount_pct	DECIMAL	Discount percentage (e.g., 15 for 15%)
quota	INT	Maximum number of uses
expiry_date	TIMESTAMP	Expiration date
created_at	TIMESTAMP	Coupon creation time

Purchases

Column Name	Data Type	Description
id	INT	Primary key
user_id	INT	FK to Users
course_id	INT	FK to Courses
coupon_id	INT	FK to Coupons (nullable)
price_paid	DECIMAL	Final price after discount
payment_method	VARCHAR	E.g., `'credit_card'`, `'paypal'`, etc.
purchased_at	TIMESTAMP	Purchase timestamp