# Kisi-kisi LKS
# Provinsi Jawa Tengah XXXIII
# Web Technologies
# 2025

# SERVER SIDE MODULE

## Contents

1. MODULE_SERVER_SIDE.docx
2. MODULE_SERVER_SIDE_MEDIA.zip

## Introduction

A new founded company is looking for full stack developers to create an online browser gaming platform. Game developers can upload their games to the platform and users can play them online in the browser.

There are three parts to the platform:

- Developer Portal: A web application for game developers to upload their games to the platform.
- Administrator Portal: A web application for administrators to manage the platform with its users and games.
- Gaming Portal: A web application for players to play games online in the browser.

The company wants to create a minimum viable product (MVP) for the platform. The MVP should already contain the aforementioned parts, but it is acceptable that the Game Developer Portal and the Administrator.

Portal are not fleshed out yet. The Gaming Portal should be fully functional, so that users can play games online in the browser.

## Description of Projects

The project is split into two phases:

- Phase one for building the API and static pages using a PHP framework and MySQL database.
- Phase two for building the frontend parts using HTML/CSS and a JavaScript framework, consuming the API developed in phase one.

You can find the provided media files to support your work:

- Provided frameworks (laravel, vuejs, reactjs)
- Postman collection and environment
- Template GUI (to build frontend UI)
- lks-server.sql (a database with structures and dummy data)

# Phase 1 : RESTful API

In this phase, you should build a RESTful API using the Laravel framework according to the documentation below.

Ensure users can login using credentials below:

**Administrator**

| Username | Password |
|---|---|
| admin1 | hellouniverse1! |
| admin2 | hellouniverse2! |

**Developer**

| Username | Password |
|---|---|
| dev1 | hellobyte1! |
| dev2 | hellobyte2! |

**Players**

| Username | Password |
|---|---|
| player1 | helloworld1! |
| player2 | helloworld2! |

## REST API Specification

**General information:**

- The response bodies contain some static example data. Dynamic data from the database should be used.
- Placeholder parameters in the URL are marked with a preceding colon (e.g. :slug or :id).
- The order of properties in objects does not matter, but the order in an arrays does.
- The Content-Type header of a request must always be application/json for POST, PUT, PATCH.
- The Content-Type header of a response is always application/json unless specified otherwise.
- Timestamps are formatted as ISO-8601 strings. E.g. 2032-01-31T21:59:35.000Z.
- The given URLs are relative to the base URL of the API. E.g. /api/v1/games is the URL to get all games.
- The API URLs must not end in .php or .html or any other file extension. The game files are an exception to this.

- The token for protected endpoints must be specified as a Bearer token in the `Authorization` header. I.e. `Authorization: Bearer <token>`

# 1. Authentication

You should create Login and Logout endpoints. The accessToken must be generated by sanctum and will be placed in the request headers Authorization Bearer.

## Sign Up

**POST [domain]/api/v1/auth/signup**

This endpoint creates a new user and returns a session token.

**Request Body**:

```
{
  "username": "testuser",
  "password": "asdf1234"
}
```

| PROPERTY | COMMENT |
|---|---|
| **username** | required, unique, min length 4, max length 60 |
| **password** | required, min length 5, max length 20 |

**Response:**

### Successful creation response:

Status Code: 201

Response Body:

```
{
 "status": "success",
 "token": "xxx"
}
```

## Sign In

**POST [domain]/api/v1/auth/signin**

This checks the username and password against all known users. If found, a session token is returned.

### Valid response:

Status Code: 200

Request Body**:**

```
{
 "username": "testuser",
 "password": "asdf1234"
}
```

| PROPERTY | DESCRIPTION |
|---|---|
| **username** | required, unique, min length 4, max length 60 |
| **password** | required, min length 5, max length 20 |

Response Body:

```
{
 "status": "success",
 "token": "xxx"
}
```

### Wrong username / password response:

Status Code: 401

Response Body:


```
{
 "status": "invalid",
 "message": "Wrong username or password"
}
```


### Sign Out

**POST [domain]/api/v1/auth/signout**
Deletes the current session token.

### Valid response:

Status Code: 200

Response Body:


```
{
 "status": "success"
}
```


## 2. Users

Get all admin data **GET [domain]/api/v1/admins** Returns an admin data.

**Response**:

Status Code: 200


```
{
  "totalElements": 2,

 "content": [
 {
     "username": "admin1",
     "last_login_at": "",
     "created_at": "2024-04-05 20:55:40",
     "updated_at": "2024-04-05 20:55:40",
 },
 {
     "username": "admin2",
     "last_login_at": "2024-04-05 20:55:40",
     "created_at": "2024-04-05 20:55:40",
     "updated_at": "2024-04-05 20:55:40",
```

```
    }
 ]
}
```


## User is not administrator response:

Status Code: 403

Response Body:


```
{
 "status": "forbidden",
 "message": "You are not the administrator"
}
```


**POST [domain]/api/v1/users** This endpoint can be used to create a user.

**Request Body**:


```
 {
    "username": "testuser",
    "password": "asdf1234"
}
```


| PROPERTY | COMMENT |
|---|---|
| **username** | required, unique, min length 4, max length 60 |
| **password** | required, min length 5, max length 20 |

**Response:**

## Successful creation response:

Status Code: 201

Response Body:


```
{
 "status": "success",
 "username": "testuser"
}
```


## Existing username:

If the username is not unique, the admin user cannot be created and instead the following response is returned.

**Response**:

Status Code: 400

{
"status": "invalid",
"message": "Username already exists"
}

## User is not administrator response:

Status Code: 403

Response Body:

{
"status": "forbidden",
"message": "You are not the administrator"
}

### GET [domain]/api/v1/users

Returns a user details.

**Response**:

Status Code: 200

{
"totalElements": 2,

"content": [
{
"username": "player1",
"last_login_at": "",
"created_at": "2024-04-05 20:55:40",
"updated_at": "2024-04-05 20:55:40",
},
{
"username": "player2",
"last_login_at": "2024-04-05 20:55:40",
"created_at": "2024-04-05 20:55:40",
"updated_at": "2024-04-05 20:55:40",
}

]
}

### User is not administrator response:

Status Code: 403

Response Body:

{
"status": "forbidden",
"message": "You are not the administrator"
}

**PUT [domain]/api/v1/users/:id**

This endpoint can be used to update a user.

**Request Body**:

{
"username": "testuser",
"password": "asdf1234"
}

| PROPERTY | COMMENT |
|---|---|
| **username** | required, unique, min length 4, max length 60 |
| **password** | required, min length 5, max length 20 |

**Response:**

### Successful creation response:

Status Code: 201

Response Body:

{
"status": "success",
"username": "testuser"
}

### Existing username:

If the username is not unique, the admin user cannot be created and instead the following response is returned.

**Response**:

Status Code: 400

{
"status": "invalid",
" message": "Username already exists"
}

### User is not administrator response:

Status Code: 403

Response Body:

{
"status": "forbidden",
"message": "You are not the administrator"
}

**DELETE [domain]/api/v1/users/:id**

This endpoint can be used to update a user.

### Successful deletion response:

Status Code: 204

This returns an empty body. The `Content-Type` header does not have to be `application/json`.

### User is not found response:

Status Code: 403

Response Body:

{
"status": "not-found",
"message": "User Not found"
}

### User is not administrator response:

Status Code: 403

Response Body:

{

"status": "forbidden",
"message": "You are not the administrator"
}

# 3. Games

### GET [domain]/api/v1/games

Returns a paginated list of games.

Query Parameters:

| PROPERTY | DESCRIPTION | DEFAULT |
|---|---|---|
| **page** | Page number. Starts at 0 | 0 |
| **size** | Page size. Must be greater or equal than 1 | 10 |
| **sortBy** | Field to sort by. Must be one of "title", "popular", "uploaddate" | title |
| **sortDir** | Describes sort direction. Must be one of "asc" or "desc" | asc |

The sort fields are explained here:

| FIELD | DESCRIPTION |
|---|---|
| **title** | Game title |
| **popular** | Counts the total number of scores per game and sorts by this count |
| **uploaddate** | Latest game version upload timestamp |

In `content`, the fields `thumbnail` and `uploadTimestamp` refer only to the latest version.

The field `scoreCount` is the sum of scores over all versions.

### Response:

Status Code: 200

Response Body:

{
"page": 0,

```
 "size": 10,
  "totalElements": 15,

 "content": [
 {
     "slug": "demo-game-1",
     "title": "Demo Game 1",
     "description": "This is demo game 1",
     "thumbnail": "/games/:slug/:version/thumbnail.png",
     "uploadTimestamp": "2032-01-31T21:59:35.000Z",
     "author": "dev1",
     "scoreCount": 5
 }
 ]
}
```


Response page fields explained:

| FIELD | DESCRIPTION |
|---|---|
| **page** | The requested page number. Starts at 0 |
| **size** | The actual page size. Must be less or equal than the requested page size |
| **totalElements** | The total number of elements regardless of page |
| **content** | An array of the games in the page |

It can be computed how many pages there are:

pageCount = ceil(totalElements / requestedSize)

It can also be computed if the returned page is the last page by multiplying the (page+1) by requested page size and checking if the result is less than or equal to the total elements.

isLastPage = (page + 1) * requestedSize >= totalElements

Note 1: If there is a game that has no game version yet, it is not included in the response nor the total count.
Note 2: If there is no thumbnail, the thumbnail field is null.

### POST [domain]/api/v1/games

This endpoint can be used to create a game. However, the game version needs to be uploaded in a separate step. If a game does not have a version yet, it is not returned in this endpoint.

**Request Body:**

{
"title": "Demo Game 3",
"description": "This is demo game 3"
}

| PROPERTY | DESCRIPTION |
|---|---|
| **title** | required, min length 3, max length 60 |
| **description** | required, min length 0, max length 200 |

**Response:**

### Successful creation response:

Status Code: 201

Response Body:

{
"status": "success",
"slug": "generated-game-slug"
}

### Existing slug:

If the generated slug is not unique, the game cannot be created and instead the following response is returned.

**Response**:

Status Code: 400

{
"status": "invalid",
"slug": "Game title already exists"
}

**GET [domain]/api/v1/games/:slug**
Returns a games details.

**Response**:

Status Code: 200

{
"slug": "demo-game-1",
"title": "Demo Game 1",
"description": "This is demo game 1",
"thumbnail": "/games/:slug/:version/thumbnail.png",
"uploadTimestamp": "2032-01-31T21:59:35.000Z",
"author": "dev1",
"scoreCount": 5,
"gamePath": "/games/demo-game-1/1/"
}

If there is no thumbnail, the thumbnail field is null.

The `gamePath` field points to a URL path that browsers can use to render the game. This means this is a reachable asset path.

## Game file upload

### POST [domain]/api/v1/games/:slug/upload

The user can upload a new version of a game if they are the author of that game.

- This is not a REST endpoint and rather it accepts a file upload. The parameter name is `zipfile`.
- The version of the game is an integer and incrementing. The first version is `1`. The user cannot control the version.
- The session token needs to be provided as form parameter `token`.
- The path has to be stored in the game record, so that players can find the game files.

If the upload fails because of one of these possible reasons, the response must be a plain text explanation of the error.

- User is not author of the game

## Serve game files

### GET /games/:slug/:version/

The game files that were uploaded are served under that path which is public.

### PUT [domain]/api/v1/games/:slug

This endpoint allows the author of the game to update the game title and description.

**Request Body**:

{
"title": "Demo Game 1 (updated)",
"description": "Updated description"
}

**Note: This does not update the game's slug.**

**Response**:

### Successful update response:

Status Code: 200

Response Body:

```
{
 "status": "success"
}
```

### User is not game author response:

Status Code: 403

Response Body:

```
{
 "status": "forbidden",
 "message": "You are not the game author"
}
```

**DELETE [domain]/api/v1/games/:slug**

The author can delete their game. This deletes the game, all versions and all scores.

**Response**:

### Successful deletion response:

Status Code: 204

This returns an empty body. The `Content-Type` header does not have to be `application/json`.

### User is not game author response:

Status Code: 403

Response Body:

```
{
 "status": "forbidden",
 "message": "You are not the game author"
```

```
}
```

### GET [domain]/api/v1/users/:username

**Returns the user details. Response**:

Status Code: 200

Response Body:

```
{
 "username": "dev1",
 "registeredTimestamp": "2032-01-31T21:59:35.000Z",
 "authoredGames": [
      {
         "slug": "demo-game-1",
         "title": "Demo Game 1",
         "description": "This is demo game 1"
      }
 ],
 "highscores": [
      {
         "game": {
            "slug": "demo-game-1",
            "title": "Demo Game 1",
            "description": "This is demo game 1"
         },
         "score": 15,
         "timestamp": "2032-01-31T21:59:35.000Z"
      }
 ]
}
```

The authoredGames is an array that returns all games with at least one version where this user is the author. If the user requesting the user details is the user itself, this returns also games that have no version yet.

The highscores is an array of highest scores per game played.

### GET [domain]/api/v1/games/:slug/scores

Returns the highest scores of each player that played any version of the game, sorted by score (descending).

**Response**:
Status Code: 200

Response Body:


```
{
 "scores": [
      {
         "username": "player2",
         "score": 20,
         "timestamp": "2032-01-31T21:59:35.000Z"
      },
      {
         "username": "player1",
         "score": 15,
         "timestamp": "2032-01-31T21:59:35.000Z"
      }
   ]

}
```


**POST [domain]/api/v1/games/:slug/scores**
When a user ends a game run, the score can be posted to this endpoint.

**Request Body**:


```
{
 "score": 100
}
```


The game version associated to the score is the latest one available.

**Response**:

## Successful creation response:

Status Code: 201

Response Body:


```
{
 "status": "success"
}
```

## Invalid request body

If the POST or PUT request had invalid fields, they are validated and a response is returned that lists the violations.

Status Code: 400

Response Body:

```
{
 "status": "invalid",
 "message": "Request body is not valid.",
 "violations": {
        "field_name": {
            "message": "required"
        },
        "field_name": {
            "message": "must be at least 4 characters long"
        },
        "field_name": {
            "message": "must be at most 60 characters long"
        }
 }
}
```

In the above example, all possible violations are shown. The actual returned violations should only be the fields which were actually invalid. At most one validation per field is shown. Validations are executed in order of appearance in the example above. field_name must be replaced with the actual field name.

**The messages for length must include the actual length requirement value.**

## Missing or invalid auth header

If the consumer makes a call to a path that requires the auth header to be present, this must be the response:

Status Code: 401

Response Body:

```
 {
    "status": "unauthenticated",
    "message": "Missing token"
```

```
}
```

If the consumer makes a call to a path that requires the auth header to be present, but provides an invalid or not existing token, this must be the response:

Status Code: 401

Response Body:

```
{
   "status": "unauthenticated",
   "message": "Invalid token"
}
```

If the consumer makes a call to a path that requires the auth header to be present, but the user is blocked, this must be the response:

Status Code: 403

Response Body:

```
{
   "status": "blocked",
   "message": "User blocked",
   "reason": "You have been blocked by an administrator"
}
```

Note: The reason is a dynamic value, chosen by the admin that blocks the user.

The following method and path patterns require a valid session header to be present:

- POST /api/v1/auth/signout
- POST, PUT, DELETE /api/v1/games/**
- GET, POST, PUT, DELETE /api/v1/users/**
- GET /api/v1/admins/**

## Non-existing API path

If the consumer makes a call to a non-existing path, or a resource that does not exist, this must be the response:

Status Code: 404

Response Body:

```
{
   "status": "not-found",

   "message": "Not found"
}
```

## ERD