Конкурсное задание

Республиканского конкурса профессионального мастерства WorldSkills Kazakhstan 2022

по компетенции Веб технологии

Module A: Back-end

Адаптировали:

Главный эксперт Попов Денис

Заместитель главного эксперта Афанасьева Дина

Contents

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:

- Game 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.

The project is split into two phases:

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

When phase two starts, a backend implementation will be provided so focus can be put on the frontend.

DESCRIPTION OF PROJECT AND TASKS

This phase one requires you to implement the backend of the platform. The backend should be implemented using a PHP framework and a MySQL database.

The backend provides:

- Administrator Portal (using server-side rendering, no JavaScript except games filtering)
- REST API for the Developer and Gaming portals

DOMAIN MODEL

The platform has the following core domain model:

- Admin User: Can log into the Administrator Portal and use all of its functionality.
  - username
  - password
  - registered timestamp
  - last login timestamp
- Platform User: Can sign up and log into the Gaming and Developer Portal and use all of its functionality.
  - username
  - password
  - registered timestamp
  - last login timestamp
  - game scores (relates to a set of game scores)
  - uploaded games (relates to a set of games)
- Game Score: Each time a user plays a game, a game score is created.
  - user (relates to a platform user)
  - game version (relates to a game version)
  - timestamp
  - score (number of points, signed, [-2147483648, 2147483647])
- Game: A game that can be played on the platform.
  - title
  - description
  - optional thumbnail
  - slug (a unique and human-readable name that can be used as part of an URL to identify the game, based on the game title)
  - author (relates to a platform user)
- Game Version: A version of a game that can be played on the platform. Developers can upload new versions but not delete old versions.
  - game (relates to a game)
  - version timestamp
  - path to game files

ADMINISTRATOR PORTAL

The Administrator Portal is a web application for administrators to manage the platform with its users and games.

The login to the portal must be reachable via the path /XX-module-a/admin.

The portal can only be accessed by logging in using a username and password. The username and password are stored in the database. The admin users are not tied to the users of the Developer or Gaming Portal. This means the admin users can have the same username as a user of the Developer or Gaming Portal. There is no registration or sign up for administrators, they are instead added via direct database access (during the MVP).

Functionalities of the Administrator Portal:

- List admin users:
  - The admin can see all admin users in the database with username, created timestamp and last login timestamp.
- Manage platform users:
  - The admin can see all platform users with username, registration timestamp, last login timestamp and click a link to their profile page.
    - Link format: XX-module-a/user/:username, where :username is replaced with the user's username.
  - The admin can block and unblock users given a choice of reasons.
    - The blocked users can no longer log in and instead are shown the block reason when they try to log in.
    - The blocked user's scores no longer show up in any of the highscores.
    - The blocked user's profile is no longer reachable and instead treated as it would not exist (HTTP Status 404 NOT FOUND).
    - If the user was already logged in, they will be logged out and shown the reason the next time they interact with the backend. I.e. any request to the backend that requires authentication will instead return HTTP Status 401 UNAUTHORIZED with a JSON body that includes the reason.
    - The reasons can be one of: "You have been blocked by an administrator", "You have been blocked for spamming", "You have been blocked for cheating".

- Manage games:
    - The admin can see all games with title, description, optional thumbnail, author, all version timestamps and click a link to the game page.
        - Link format: XX-module-a/game/:slug, where :slug is replaced with the game's slug.
        - The admin can filter games in-page by searching for game title. Note: JavaScript allowed here.
    - The admin can mark a game as deleted in the database. Platform users (players or developers) will not see the game anymore. The admin still sees it with status "deleted".
    - The admin can see all scores of all players per game and per version.
    - The admin can reset the highscores for a game. This means all the scores related to any of the versions of a game are removed from the database and the players will not see them anymore.
    - The admin can delete a single player score from a game.
    - The admin can delete all scores of a single player for a given game (for all the game versions).

The Administrator Portal should be implemented using server side rendering. This means the backend should render the HTML pages and each interaction with the backend should result in a full page reload. No JavaScript is allowed, except for games filtering.

The pages do not have to adhere to good design practices, but should be functionally usable.

REST API

The API is consumed by the frontend using AJAX requests. The API is stateless and uses JSON for data exchange.

The API provides:

- Authentication
    - User Sign Up
    - User Sign In
    - User Sign Out
- Developer scope:

- CRUD (create, read, update, delete) operations for authored games

- Player scope:
  - Finding available games
  - Loading game assets and play the game
  - Saving game score
  - Seeing a user's information
    - username, registered timestamp
    - scores per game
    - uploaded games
  - Seeing each player's highest score of a game

Game Files

Developers can upload new versions of games by providing a ZIP file. The ZIP file must at least contain an index.html file.

It can optionally also contain a thumbnail.png file that is displayed to admins and users.

Each version of a game receives a unique ID and is served under a special path.

Database

The database has to be set up by you. There is no schema provided. You are required to use the database to store the data referenced in the domain model. You can decide on types, add tables and columns as you see fit. Your schema is expected to be normalized (all the attributes (database columns) are functionally dependent on solely the primary key; third normal form or higher is acceptable).

You are also required to fill the database with the following example data for evaluation purposes. So, please ensure you check the spelling.

You can make reasonable assumptions for timestamp values that are not specified here.

At the end, besides a set-up database, you must provide the ER diagram and an SQL dump that creates the schema and inserts the data below into the database. Store it as database-dump.sql in your work folder.

Admins

username	password
admin1	hellouniverse1!
admin2	hellouniverse2!

Users

username	password
player1	helloworld1!
player2	helloworld2!
dev1	hellobyte1!
dev2	hellobyte2!

Games

title	slug	description	author
Demo Game 1	demo-game-1	This is demo game 1	dev1
Demo Game 2	demo-game-2	This is demo game 2	dev2

Game Versions

game	files
Demo Game 1	upload files provided to you under media files named demo-game-1-v1/ (this will be the first version, not shown to anyone)

game	files
Demo Game 1	upload files provided to you under media files named demo-game-1-v2/ (this should be the latest version)
Demo Game 2	upload files provided to you under media files named demo-game-2-v1/

Scores

user	game version	score
player1	first version of "Demo Game 1"	10.0
player1	first version of "Demo Game 1"	15.0
player1	latest version of "Demo Game 1"	12.0
player2	latest version of "Demo Game 1"	20.0
player2	latest version of "Demo Game 2"	30.0
dev1	latest version of "Demo Game 1"	1000.0
dev1	latest version of "Demo Game 1"	-300.0
dev2	latest version of "Demo Game 1"	5.0
dev2	latest version of "Demo Game 2"	200.0

Authentication

The authentication of admin users and platform users is done using a username and password, however the mechanism is different.

Administrator Authentication

The administrators log in and receive a session cookie. The session cookie is stored on the server file system (stateful) and expires automatically after the configured session duration (ensure this is ≥ 1hr). The session cookie is sent with every request to the backend.

REST API Authentication

When a platform user logs in, the API returns a session token. The client is then responsible for storing the token and sending it with each request to the backend as a HTTP request header. The backend stores the token in the database with an expiration timestamp. The token is stateless and expires automatically after one hour.

If a user logs out, the session token is removed from the database.

Note: A user can log in with several devices / browsers at the same time and should not be logged out when they log in with a different device / browser (or use Incognito mode).

Administrators can revoke a user's session tokens to forcefully log a user out when they want to block a user.

You can use any form of token (e.g. UUID or anything that cannot be guessed by a machine within reasonable time).

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 :username).
- 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>`

POST /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 8, max length 2^16

Response:

Successful creation response:

Status Code: 201

Response Body:

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

POST /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	Comment
username	required, min length 4, max length 60
password	required, min length 8, max length 2^16

Response Body:

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

Wrong username / password response:

Status Code: 401

Response Body:

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

POST /api/v1/auth/signout

Deletes the current session token.

Valid response:

Status Code: 200

Response Body:

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

GET /api/v1/games

Returns a paginated list of games.

Query Parameters:

Parameter	Description	Default
page	Page number. Starts at 0.	0
size	Page size. Must be greater or equal than 1.	10

Parameter	Description	Default
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 /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	Comment
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 /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 /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 uploaded file is a ZIP file that is extracted and later provided under a public path.
- The path has to be stored in the game record, so that players can find the game files.

- If the ZIP file contained a thumbnail.png, its path is stored as thumbnail path in the game record.

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
- ZIP file extraction fails
- File size too big
- Unspecified IO error

Serve game files GET /games/:slug/:version/

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

The idea being, that the frontend can create an iframe pointing to that path. This loads the index.html which in term can load Javascript and CSS files from a relative path.

PUT /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 /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 /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 /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 /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 /api/v1/users/**`

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"
}
```

INSTRUCTIONS TO THE COMPETITOR

- In case of not finishing every task, it is competitor's choice and responsibility to allow assessment team to access completed tasks.
- You should consider the quality of code.
- All assessment is done on server. No assessment process in workstation.
- Please use proper version controlling (branches and commits) during development.

MARKING SCHEME

		Work Organization	Communication and Presentation	Layout	Front-end	Back-end	Total
A1	DB & Work Organization	1.25	1		-	2.75	5
A2	API	1				6	7
A3	Admin		0.5	0.25	0.5	7.75	9
	Total						21