```
const express = require('express');
const router = express.Router();
const bcrypt = require('bcrypt');
const { Game, Review, User } = require('../models');

// Express route to handle review submission
router.post('/:gameId/reviews', async (req, res) => {
    const GameId = req.params.gameId;
    const { username, password, reviewText, rating } = req.body;

    try {
        const user = await User.findOne({ where: { username } });
        if (!user) {
            return res.json({ error: 'User does not exist' });
        }

        // Compare the provided password with the hashed password stored in the
database
        const passwordMatch = await bcrypt.compare(password, user.password);
        if (!passwordMatch) {
            return res.json({ error: 'Invalid username or password combination'
});
        }

        // If the username and password are valid, proceed to create the review
        await Review.create({ GameId, UserId: user.id, reviewText, rating });

        res.json({ message: 'Review submitted successfully.' });
    } catch (error) {
        res.json({ error: 'Failed to write review' });
    }
});



router.get('/search', async (req, res) => {
    const { query, criteria } = req.query;
    try {
        const games = await Game.findAll({
            attributes: [
                'id',
                'title',
                'genre',
                'company',
```

```
                'system',
                'thumbnail',
                'release_date',
                'description'
            ]
        })

        const filteredGames = games.filter(game =>
game[criteria].toLowerCase().includes(query.toLowerCase()));

        res.json(filteredGames);
    } catch(error) {
        return res.json({ error: 'Failed to get Games', message: '' })
    }

})

router.get('/:id', async (req, res) => {
    const { id } = req.params
    try {
        const games = await Game.findOne({
            attributes: [
                'id',
                'title',
                'genre',
                'company',
                'system',
                'thumbnail',
                'release_date',
                'description'
            ],
            where: { id }
        })

        res.json(games);
    } catch {
        return res.json({ error: "Failed to get Games" })
    }
})

router.get('/', async (req, res) => {
    try {
        const games = await Game.findAll({
            attributes: [
```

```
                'id',
                'title',
                'genre',
                'company',
                'system',
                'thumbnail',
                'release_date',
                'description'
            ]
        })

        res.json(games);
    } catch {
        return res.json({ error: "Failed to get Games" })
    }
})

router.post('/populate', async (req, res) => {
    const genres = await
fetch('https://api.rawg.io/api/genres?key=e5697c7dfc4641458bbb23fb7a9f6748', {
        'Content-Type': 'application/json'
    });
    let data = await genres.json();
    const games = [];

    data.results.forEach(genreElement => {
        const genre = genreElement.name;
        genreElement.games.forEach(game => {
            games.push({
                title: game.name,
                genre: genre,
                company: null,
                system: null,
                thumbnail: null,
                release_date: null,
                description: null,
                RAWGid: game.id
            })
        })
    });

    const companies = await
fetch('https://api.rawg.io/api/publishers?key=e5697c7dfc4641458bbb23fb7a9f6748&pag
e_size=72717', {
```

```
        'Content-Type': 'application/json'
    });
    data = await companies.json();

    data.results.forEach(pub => {
        const pubName = pub.name;
        pub.games.forEach(pubGame => {
            const id = pubGame.id;
            games.forEach(game => {
                if (game.RAWGid === id) {
                    game.company = pubName;
                }
            })
        })
    })

    const fetchGameInfo = games.map(async game => {
        const gameInfo = await
fetch(`https://api.rawg.io/api/games/${game.RAWGid}?key=e5697c7dfc4641458bbb23fb7a
9f6748`, {
            'Content-Type': 'application/json'
        });
        data = await gameInfo.json();
        game.description = data.description;
        game.thumbnail = data.background_image;
        game.release_date = data.released;
        game.system = data.platforms[0].platform.name;
        delete game.RAWGid;
    })

    await Promise.all(fetchGameInfo);

    try {
        games.map(async game => {
            await Game.create(game);
        })
    } catch {
        return res.json({ error: "failed to populate table" })
    }

    return await res.json(games);
})

router.post('/', async (req, res) => {
```

```
    const post = req.body;
    await Game.create(post);
    res.json(post);
})

module.exports = router;
```

- Express Router: The code initializes an Express router instance to define routes for handling various HTTP requests.

- Review Submission Route: There's a POST route defined for submitting reviews for a specific game (/:gameId/reviews). This route expects parameters such as the game ID, username, password, review text, and rating in the request body.

- User Authentication: The code checks the validity of the provided username and password combination by querying the database for the user's existence and comparing the hashed password.

- Review Creation: If the provided credentials are valid, the code proceeds to create a new review for the specified game, associating it with the authenticated user.

- Error Handling: The code includes error handling logic to catch and handle any exceptions that may occur during the review submission process.

- Search Route: There's a GET route defined for searching games based on query parameters (/search). It retrieves games from the database and filters them based on the specified criteria.

- Game Details Route: Another GET route (/:id) retrieves details of a specific game by its ID.

- All Games Route: A GET route (/) retrieves a list of all games from the database.

- Data Population Route: A POST route (/populate) fetches data from external APIs (such as genres, publishers, and game details) and populates the database with the retrieved information.

- Game Creation Route: Lastly, there's a POST route (/) for creating new game entries in the database. It expects game details in the request body and inserts them into the database.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dashboard</title>
    <link rel="stylesheet" href="../css/index.css">
</head>
<body>
    <nav>
        <ul>
            <li><a href="index.html">Dashboard</a></li>
            <li><a href="search.html">Search</a></li>
            <li><a href="profile.html">Profile</a></li>
            <li><a href="login.html">Login</a></li>
            <li><a href="register.html">Register</a></li>
        </ul>
    </nav>
    <div class="container">
        <h1>Game Dashboard</h1>
        <!-- Scroll of pictures representing games -->
        <div id="gameScroll"></div>
    </div>
    <script src="../js/index.js"></script>
</body>
<footer>
    <h3>Made By: Matteo Saputo</h3>
</footer>
</html>
```

```
// Function to get games from node backend
async function getGames() {
    try {
        const response = await fetch('http://localhost:3001/game', {
            method: 'GET'
        });
        // Check if response is successful
```

```
        if (!response.ok) {
            throw new Error('Failed to fetch games');
        }
        const data = await response.json(); // Extract JSON data from the response
        return data; // Return the extracted data
    } catch (error) {
        console.error('Error fetching games:', error.message);
        return []; // Return an empty array in case of an error
    }
}

// Function to render games as a scroll of pictures
async function renderGameScroll() {
    const games = await getGames();
    const gameScroll = document.getElementById('gameScroll');
    gameScroll.innerHTML = '';

    games.forEach(game => {
        // Create a container div for the clickable area
        const gameContainer = document.createElement('div');
        gameContainer.classList.add('game-container');

        // Create an anchor tag for the image
        const gameLink = document.createElement('a');
        gameLink.href = `game.html?id=${game.id}`; // Link to game page
        gameLink.classList.add('game-link');

        // Create a paragraph tag for the game title
        const gameTitle = document.createElement('p');
        gameTitle.textContent = game.title; // Set the text content to the game
title

        // Create an image tag for the game thumbnail
        const gameImg = document.createElement('img');
        gameImg.src = game.thumbnail;

        // Append the image to the anchor tag
        gameLink.appendChild(gameImg);

        // Append the title and the anchor tag to the container div
        gameContainer.appendChild(gameTitle);
        gameContainer.appendChild(gameLink);

        // Append the container div to the game scroll container
```

```
        gameScroll.appendChild(gameContainer);
    });

    // Use setTimeout to repeat the scroll after a certain interval
    setTimeout(renderGameScroll, 5000); // Repeat every 5 seconds (adjust as
needed)
}



// Call the renderGameScroll function when the page loads
window.onload = renderGameScroll;
```

```
/* Body styles */
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
}

/* Navigation styles */
nav {
    background-color: #333;
    color: #fff;
    padding: 10px;
}

nav ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
}

nav ul li {
    display: inline;
    margin-right: 20px;
}

nav ul li a {
    color: #fff;
    text-decoration: none;
    font-weight: bold;
}
```

```css
/* Container styles */
.container {
    max-width: 1200px;
    margin: 20px auto;
    padding: 0 20px;
}

/* Header styles */
h1 {
    margin-top: 0;
    font-size: 32px;
}

/* Game scroll container styles */
#gameScroll {
    display: flex;
    overflow-x: auto;
    gap: 20px;
}

/* Game container styles */
.game-container {
    text-align: center;
}

/* Game link styles */
.game-link {
    display: block;
    text-decoration: none;
    color: inherit;
}

/* Game title styles */
.game-container p {
    margin-bottom: 10px;
}

/* Game image styles */
.game-link img {
    max-width: 200px;
    border-radius: 5px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    transition: transform 0.3s ease-in-out;
```

```css
}

/* Game image hover styles */
.game-link img:hover {
    transform: scale(1.1);
}


/* Game details container styles */
#gameDetails {
    border: 1px solid #ccc;
    border-radius: 5px;
    padding: 20px;
    background-color: #f9f9f9;
}

/* Game image styles */
#gameDetails img {
    display: block;
    max-width: 100%;
    border-radius: 5px;
    margin-bottom: 20px;
}

/* Game details paragraph styles */
#gameDetails p {
    margin-bottom: 10px;
    line-height: 1.5;
}


/* Style for individual search result */
.search-result {
    border: 1px solid #ccc;
    border-radius: 5px;
    padding: 5px;
    margin-bottom: 10px;
}

/* Style for search result title */
.search-result h3 {
    font-size: 16px;
    margin-bottom: 5px;
}
```

```css
/* Style for search result thumbnail */
.search-result img {
    max-width: 80px; /* Adjust the maximum width of the thumbnail */
    height: auto;
    margin-bottom: 5px;
}

/* Style for search result description */
.search-result p {
    font-size: 12px; /* Adjust the font size of the description */
    color: #666;
}

/* Style for no results message */
#searchResults p {
    font-size: 14px; /* Adjust the font size of the message */
    color: #999;
    text-align: center;
}
```

- HTML Structure: The HTML code defines a basic structure for the dashboard page, including a navigation bar (nav) and a container (div.container) for displaying game information.
- Navigation Bar: The navigation bar (nav) contains links (a tags) to different pages of the website, such as the dashboard, search, profile, login, and register pages.
- Script Import: The HTML file imports an external JavaScript file (index.js) using the <script> tag. This JavaScript file contains logic to fetch games from the backend and render them on the dashboard.
- Rendering Games: The renderGameScroll() function is responsible for fetching games from the backend using the fetch API and rendering them as a scroll of pictures on the dashboard page.
- Dynamic Game Rendering: Games are dynamically rendered on the dashboard using the fetched data. Each game is displayed as a clickable image (<img>) wrapped in an anchor tag (<a>) that links to the game details page.
- CSS Styling: The CSS code defines styles for various elements of the dashboard page, including navigation bar (nav), container (div.container), game scroll container (#gameScroll), game containers (div.game-container), game links (a.game-link), etc.

- Responsive Design: The CSS styles ensure that the dashboard page is responsive and adapts well to different screen sizes. The max-width and margin properties are used to control the layout and spacing of elements.
- Hover Effects: Hover effects are applied to game images (<img>) using the :hover pseudo-class in CSS. When users hover over a game image, it scales up (transform: scale(1.1)) to provide visual feedback.
- Footer: The HTML code includes a footer (<footer>) with a simple "Made By" message, indicating the creator of the website.
- Error Handling: Error handling is implemented in the JavaScript code to catch and log any errors that occur during the fetching of game data from the backend. If an error occurs, a message is logged to the console, and an empty array is returned as a fallback.