JS Applications Exam – GamesPlay

You are assigned to implement a **Web application** (**SPA**) using JavaScript. The application should dynamically display content, based on user interaction and support user profiles and CRUD operations, using a REST service.
The application **must start** from file **"index.html"** on port **3000** and **must** be in **.zip** format (see **Submitting Your Solution** at the end of this document.)

Students from previous seasons attending this exam **have the option to choose** whether they will use the given **REST service** or use a **cloud-based backend** of **their choice** (**Firebase, Backendless, etc**.). As well as the libraries that will be used for **rendering** views/forms/components (**Handlebars**) or **routing libraries** (**Sammy**) **depend on their preferences**.

1. Overview

Implement a front-end application (**SPA**) to see and manage **the games world**. The app allows visitors to **browse** different types of games, including the latest collections, with different levels of difficulty. Users can **register** with an **email** and **password**, which allows them to create their own games. Game authors can also **edit** or **delete** their own posts at any time.

2. Technical Details

You are provided with the following resources:

- **Project scaffold**: A **package.json** file, containing a list of common dependencies. You may change the included libraries to your preference. The sections **devDependencies** and **scripts** of the file are used by the automated testing suite, altering them may result in incorrect test operation.

  To **initialize** the project, execute the command **`npm install`** via the command-line terminal.

- **HTML and CSS files**: All views (pages) of the application, including **sample** user-generated **content**, are included in the file **index.html**, which links to CSS and other static files. **Each view is in a separate section** of the file, which can be identified by a **unique class name or id** attribute. Your application may use any preferred method (such as a **templating library** or manual visibility settings) to display only the selected view and to **navigate** between views upon user interaction.
- **Local REST service**: A special server, which contains **sample data** and supports **user registration** and **CRUD operations** via REST requests is included with the project. Each section of this document (where applicable) includes details about the necessary **REST endpoints**, to which **requests** must be sent, and the **shape** of the expected **request body**.

  For **more information** on how to use the included server, see **Appendix A: Using the Local REST Service** at the end of this document.

- **Automated tests**: A complete test suite is included, which can be used to test the correctness of your solution. **Your work will be assessed, based on these tests.**

  For **more information** on how to run the tests, see **Appendix B: Running the Test Suite** at the end of this document

  Do not use CDN for loading the dependencies because it can **affect the tests in a negative way**!

Note: When creating HTML Elements and displaying them on the page, **adhere as close as possible to the provided HTML** samples. Changing the structure of the document may **prevent the tests** from running correctly, which will

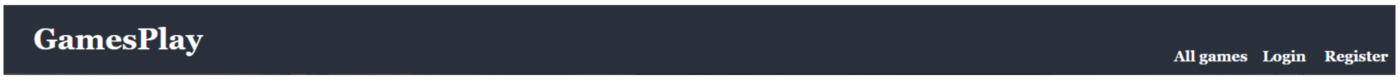
adversely affect your assessment grade. You may **add attributes** (such as **class** and **dataset**) to any HTML Element, as well as **change "href"** attributes on links and add/change the **method** and **action** attributes of HTML Forms, to facilitate the correct operation of a routing library or another method of abstraction. You may also add hidden elements to help you implement certain parts of the application requirements.

3. Application Requirements

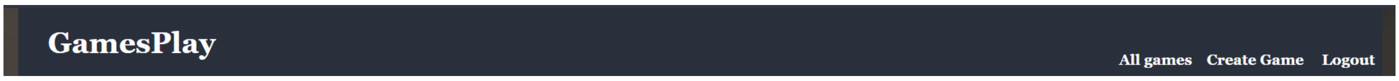
Navigation Bar (5 pts)

Navigation links should correctly change the current page (view). **GamesPlay** link should redirect to the **Home** page. **Guests** (un-authenticated visitors) can see the links to the **All Games (Catalogue)** page, as well as the links to the **Login** and **Register** pages. The logged-in user navbar should contain the links to **All Games (Catalogue)** page, the **Create** page and a link for e **Logout** action.

Guest navigation example:



User navigation example:

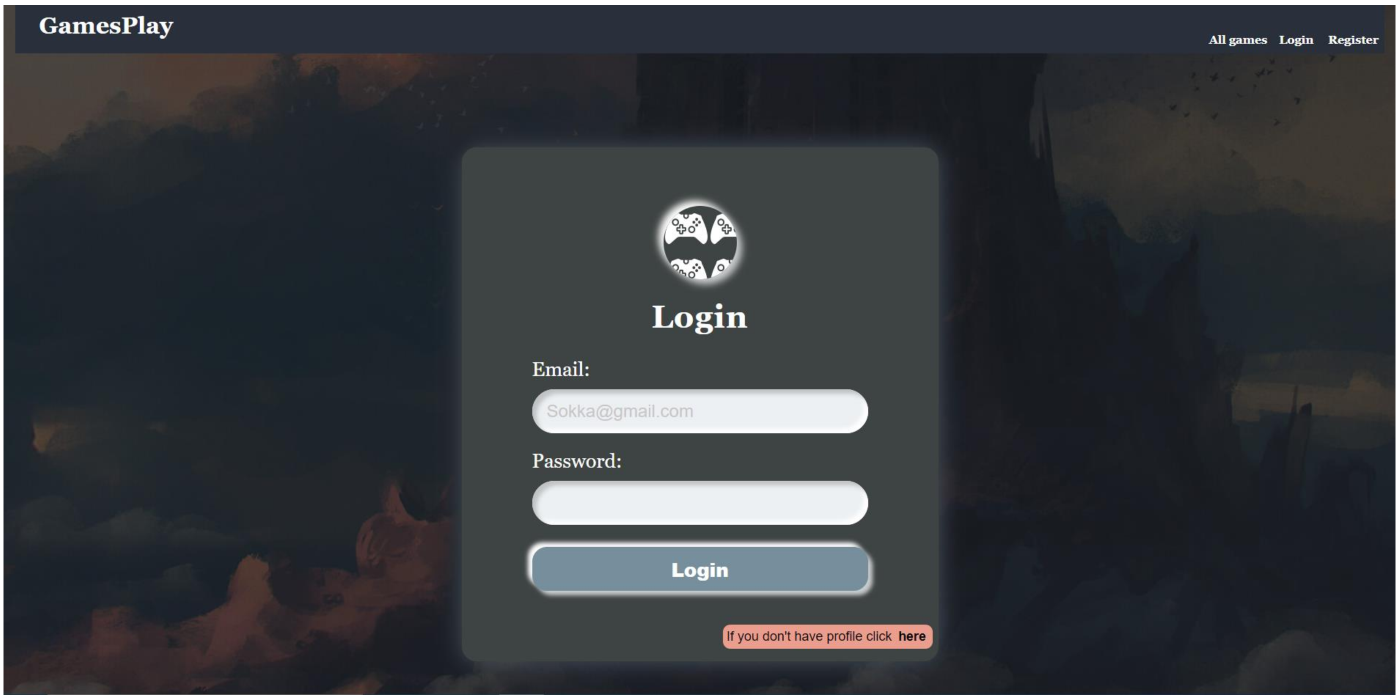


Login User (5 pts)

The **included REST service** comes with the following **premade** user accounts, which you may use for development:

```
{ "email": "peter@abv.bg", "password": "123456" }
{ "email": "john@abv.bg", "password": "123456" }
```

The **Login** page contains a form for existing user authentication. By providing an **email** and **password,** the app should login a user in the system if there are no empty fields.



Send the following **request** to perform login:

```
Method: POST
```

```
URL: /users/login
```

Required **headers** are described in the documentation. The service expects a body with the following shape:

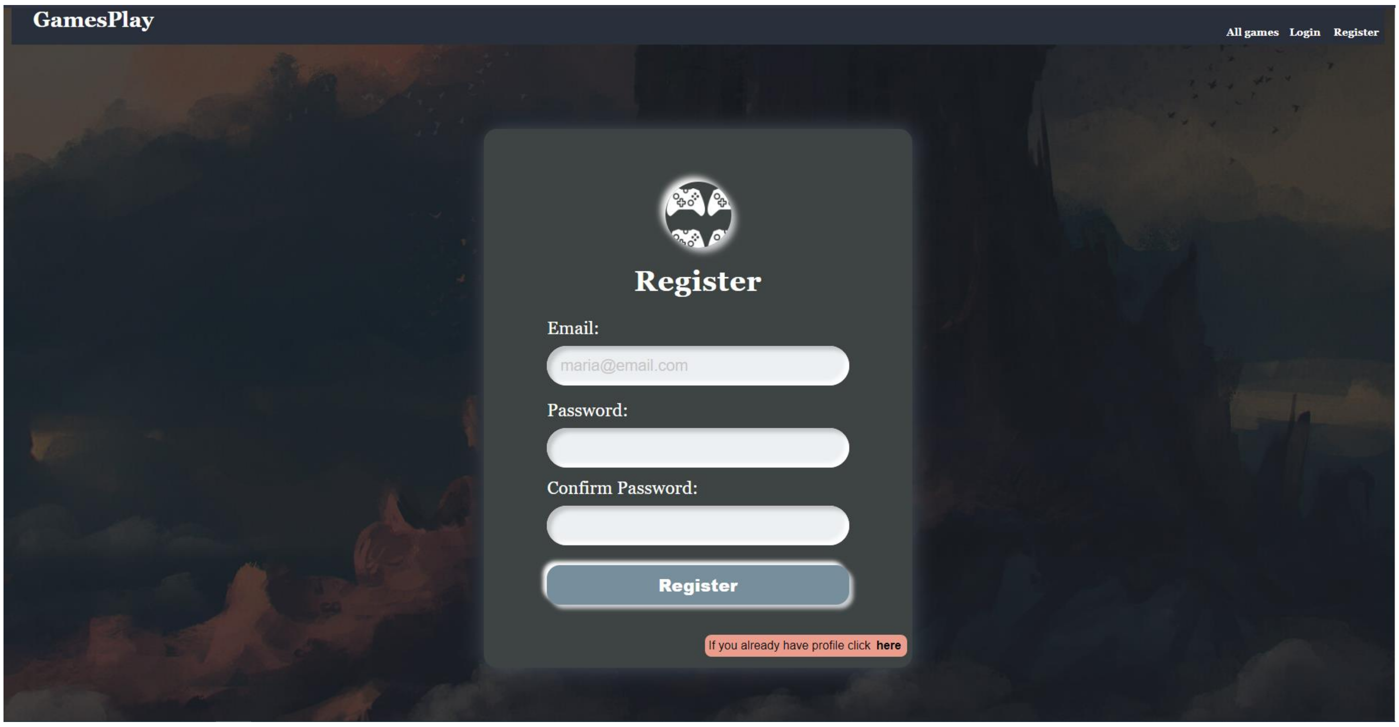
```
{
  email,
  password
}
```

Upon success, the **REST service** will return information about the existing user along with a property **accessToken**, which contains the **session token** for the user **–** you need to store this information using **sessionStorage** or **localStorage,** in order to be able to perform authenticated requests.

If the login was successful, **redirect** the user to the **Home** page. If there is an error, display an appropriate error message, using a system dialog (**`window.alert`**).

Register User (10 pts)

The **Register** page contains a form for new user registration. By providing an **email** and **password,** the app should register a new user in the system if there are no empty fields.



Send the following **request** to perform registration:

```
Method: POST
URL: /users/register
```

Required **headers** are described in the documentation. The service expects a body with the following shape:

```
{
  email,
  password
}
```

Upon success, the **REST service** will return the newly created object with an automatically generated **_id** and a property **accessToken**, which contains the **session token** for the user **–** you need to store this information using **sessionStorage** or **localStorage,** in order to be able to perform authenticated requests.

If the registration was successful, **redirect** the user to the **Home** page. If there is an error, or the **validations** don't pass, display an appropriate error message, using a system dialog (`window.alert`).

Logout (5 pts)

The logout action is available to logged-in users. Send the following **request** to perform logout:

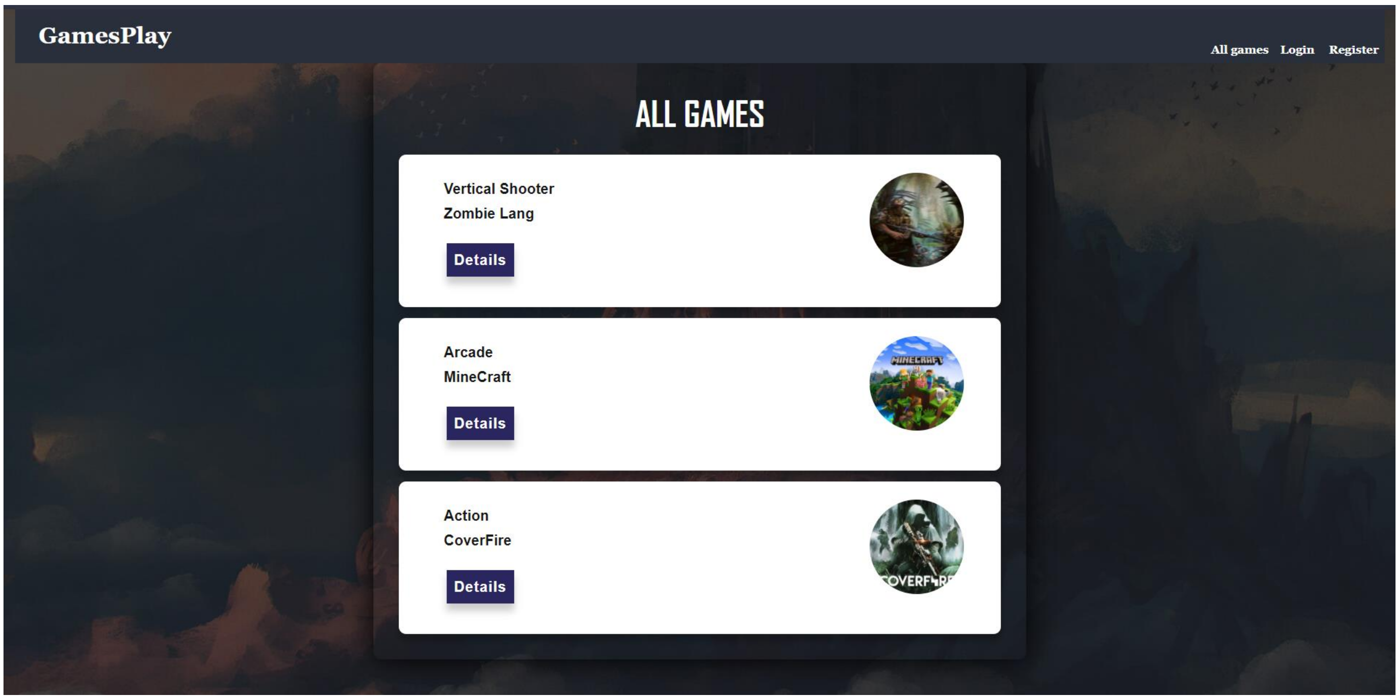
```
Method: GET
URL: /users/logout
```

Required **headers** are described in the documentation. Upon success, the **REST service** will return an **empty response**. Clear any session information you've stored in browser storage.

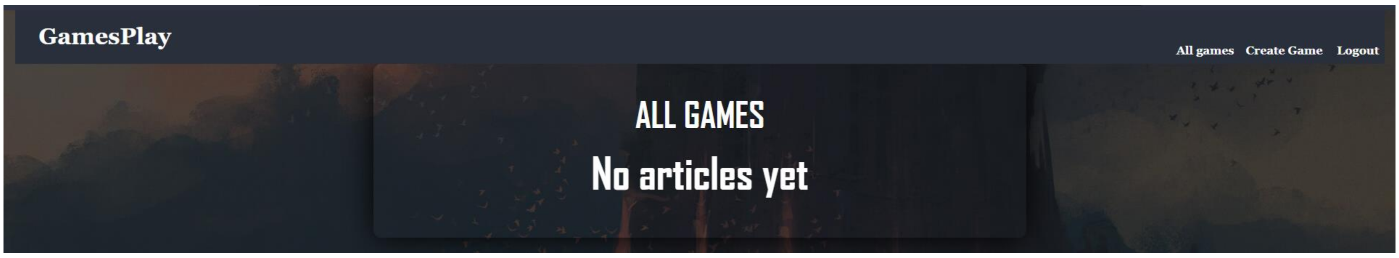
If the logout was successful, **redirect** the user to the **Home** page.

All Games Page (Catalogue) (10 pts)

This page displays a list of all games in the system, with their title and category. Clicking on any of the cards leads to the details page for the selected game.



If there are no games, the following view should be displayed:



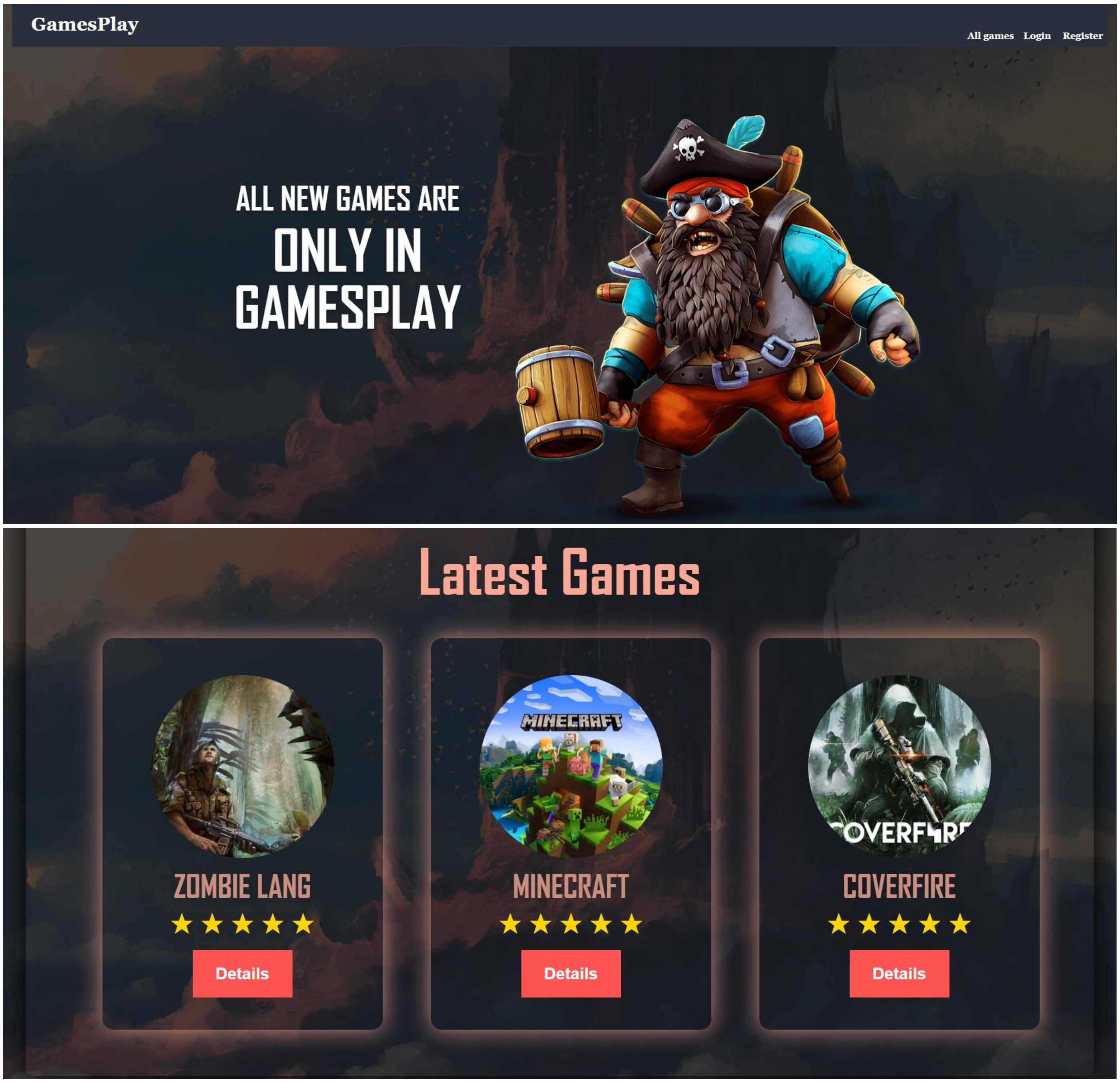
Send the following **request** to read the list of games:

```
Method: GET
URL: /data/games?sortBy=_createdOn%20desc
```

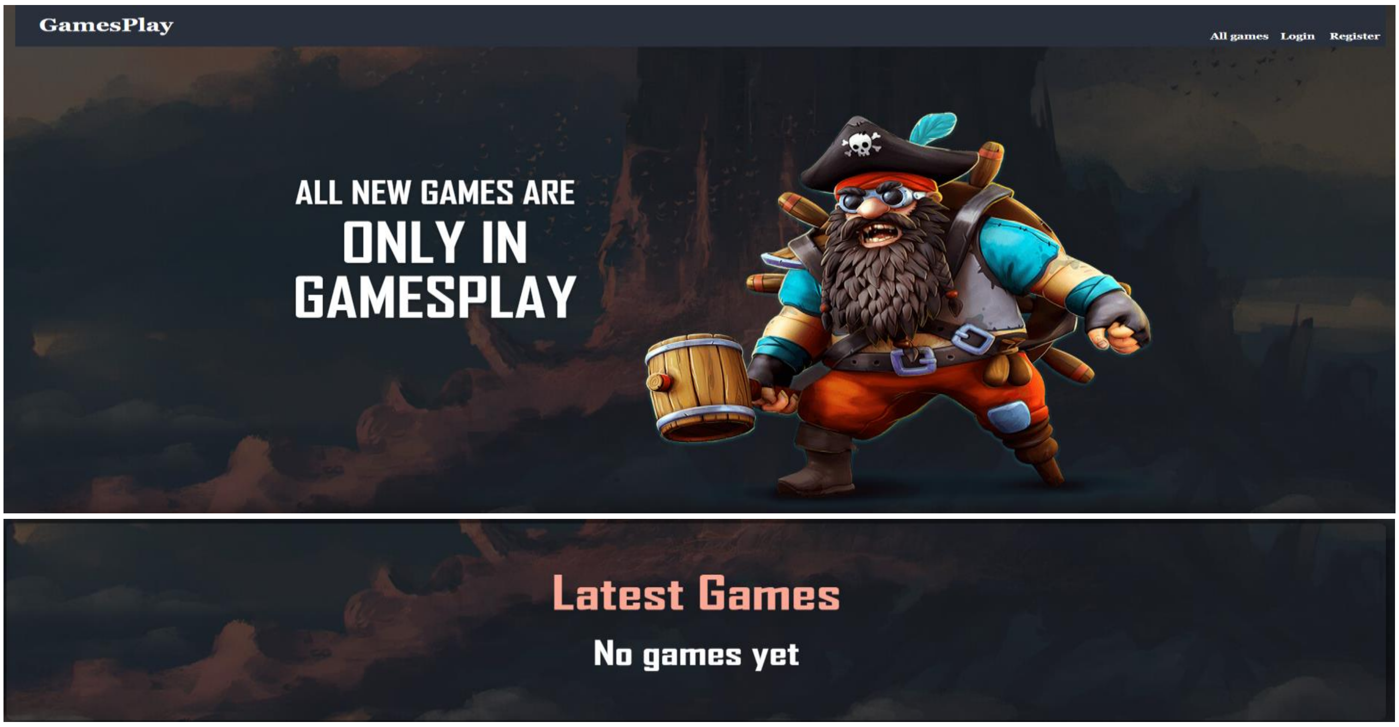
Required **headers** are described in the documentation. The service will return an array of games.

Home Page (Recent Games) (20 pts)

All users should be greeted from the homepage, where they should be able to see the three most recently added games. Clicking on the details links leads to the details page for the selected game.



If no games **have been added** yet, show the text **"No games yet"** instead.



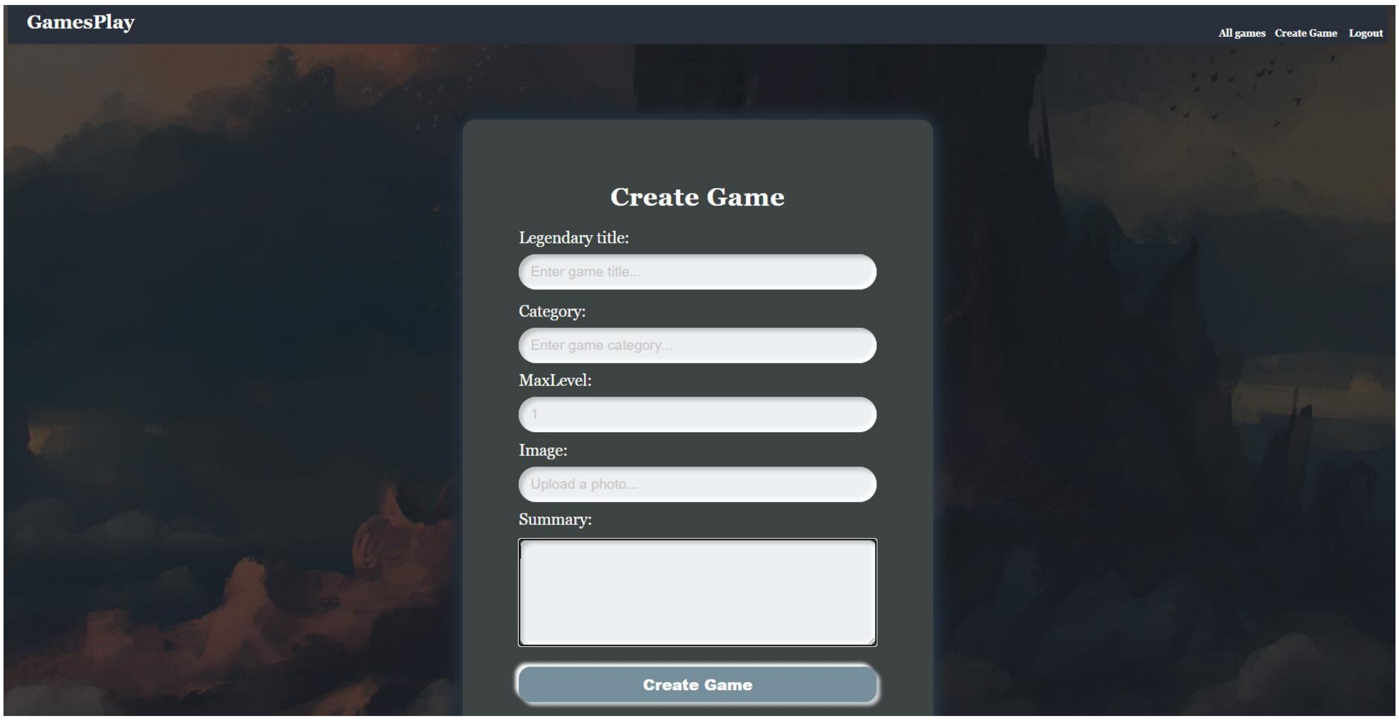
Send the following **request** to read the new games:

```
Method: GET
URL: /data/games?sortBy=_createdOn%20desc&distinct=category
```

Required **headers** are described in the documentation. The service will return an array of games.

Create Game (15 pts)

The Create page is available to **logged-in** users. It contains a form for creating new games. Check if all the fields are filled before you send the request.



To create a game, send the following **request**:

```
Method: POST
URL: /data/games
```

Required **headers** are described in the documentation. The service expects a body with the following shape:

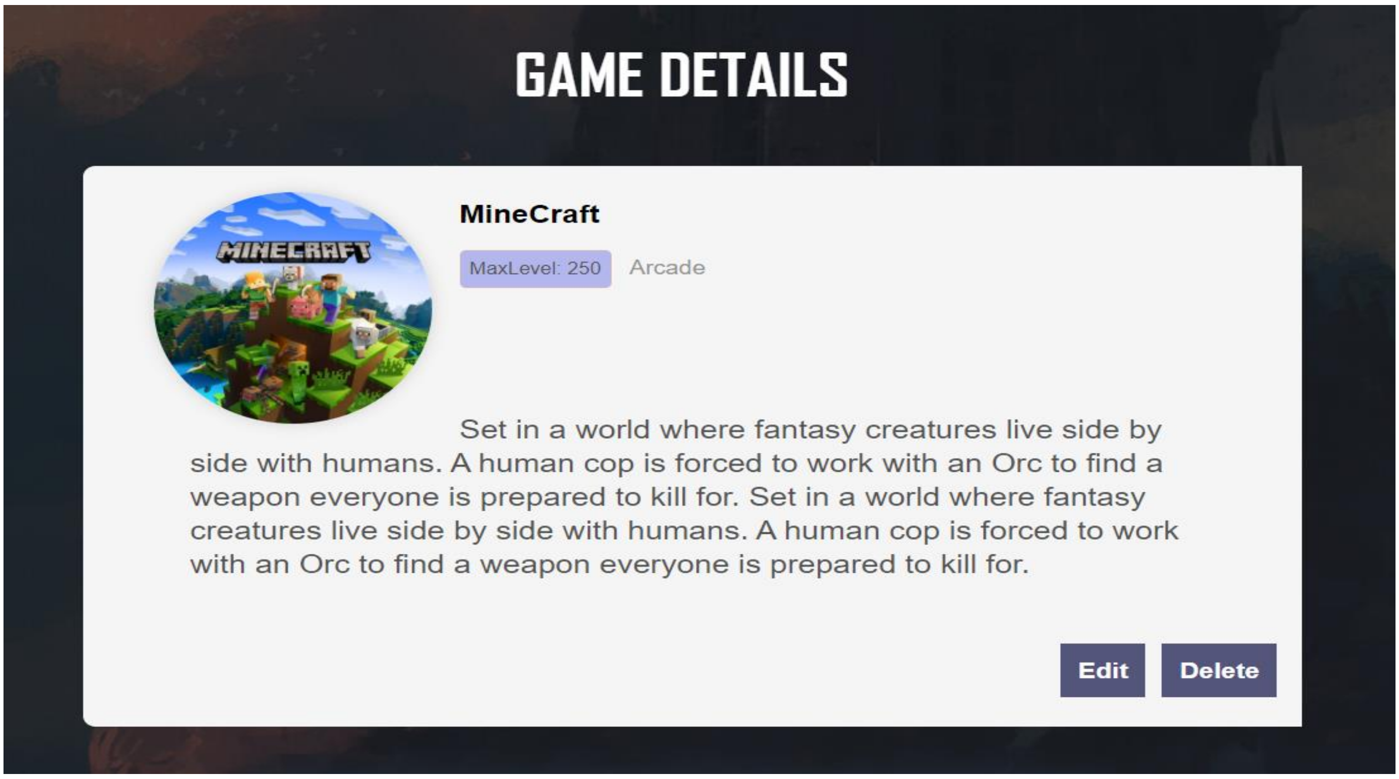
```
{
  title,
  category,
  maxLevel,
  imageUrl,
  summary
}
```

Required **headers** are described in the documentation. The service will return the newly created record. Upon success, **redirect** the user to the **Home** page.

Details (10 pts)

All users should be able to **view details** about games. Clicking the **Details** link in of a **game** should **display** the **Details** page:

- If the currently logged in user is the **creator** of the game, the **Edit** and **Delete** buttons should be displayed, otherwise they should not be available.
- The view for the **creators** should look like:

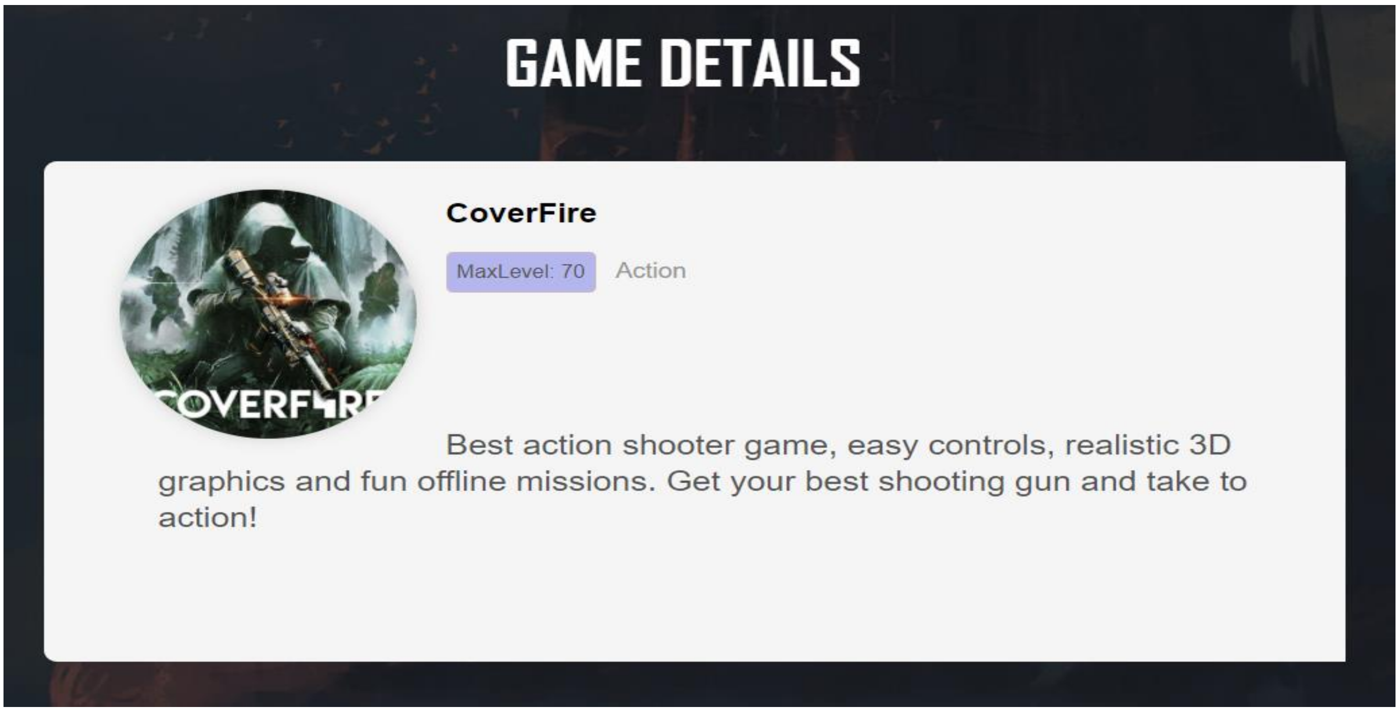

GAME DETAILS

MineCraft

MaxLevel: 250 Arcade

Set in a world where fantasy creatures live side by side with humans. A human cop is forced to work with an Orc to find a weapon everyone is prepared to kill for. Set in a world where fantasy creatures live side by side with humans. A human cop is forced to work with an Orc to find a weapon everyone is prepared to kill for.

The view for **guests** and **logged-in users** should look like:


GAME DETAILS

CoverFire

MaxLevel: 70 Action

Best action shooter game, easy controls, realistic 3D graphics and fun offline missions. Get your best shooting gun and take to action!

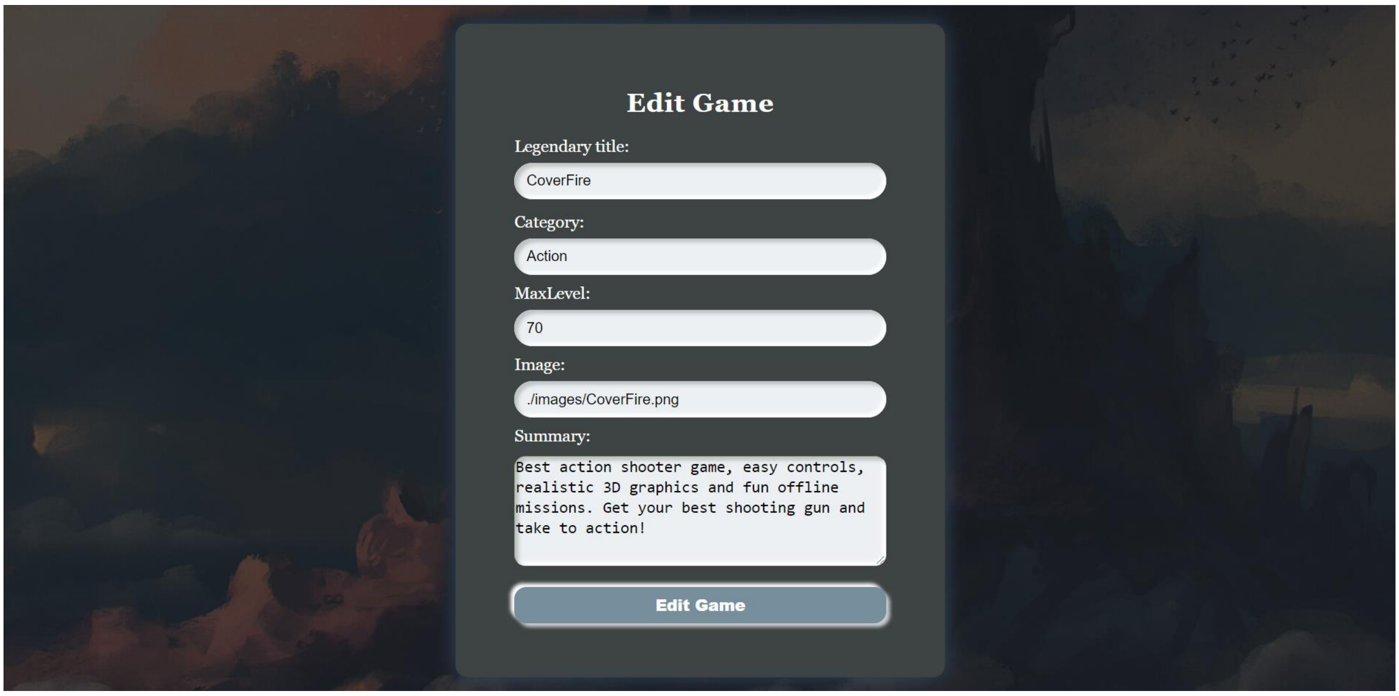
Send the following **request** to read a single game:

```
Method: GET
URL: /data/games/:id
```

Where ***:id*** is the **id** of the desired game. Required **headers** are described in the documentation. The service will return a single object.

Edit Game (15 pts)

The **Edit page** is accessible to logged-in users and allows the author to **edit** their own games. Clicking the **Edit** a specific game link on the **details page** should display the Edit page. It contains a form with input fields for all relevant properties. Make sure **all fields** are filled in before submitting the request. The fields **must be filled** in when the page is first loaded.


Edit Game

Legendary title: CoverFire

Category: Action

MaxLevel: 70

Image: ./images/CoverFire.png

Summary: Best action shooter game, easy controls, realistic 3D graphics and fun offline missions. Get your best shooting gun and take to action!

To edit a game, send the following **request**:

```
Method: PUT
URL: /data/games/:id
```

Where ***:id*** is the **id** of the desired game.

The service expects a body with the following shape:

```
{
  title,
  category,
  maxLevel,
  imageUrl,
  summary
}
```

Required **headers** are described in the documentation. The service will return the modified record. Note that **PUT** request **do not** merge properties and will instead **replace** the entire record. Upon success, **redirect** the user to the **Details** page for the current game.

Delete Game (10 pts)

The delete action is available to logged-in users, for game they have created. When the author clicks on the Delete action on any of their games, a confirmation dialog should be displayed, and upon confirming this dialog, the game should be deleted from the system.

To delete a game, send the following **request**:

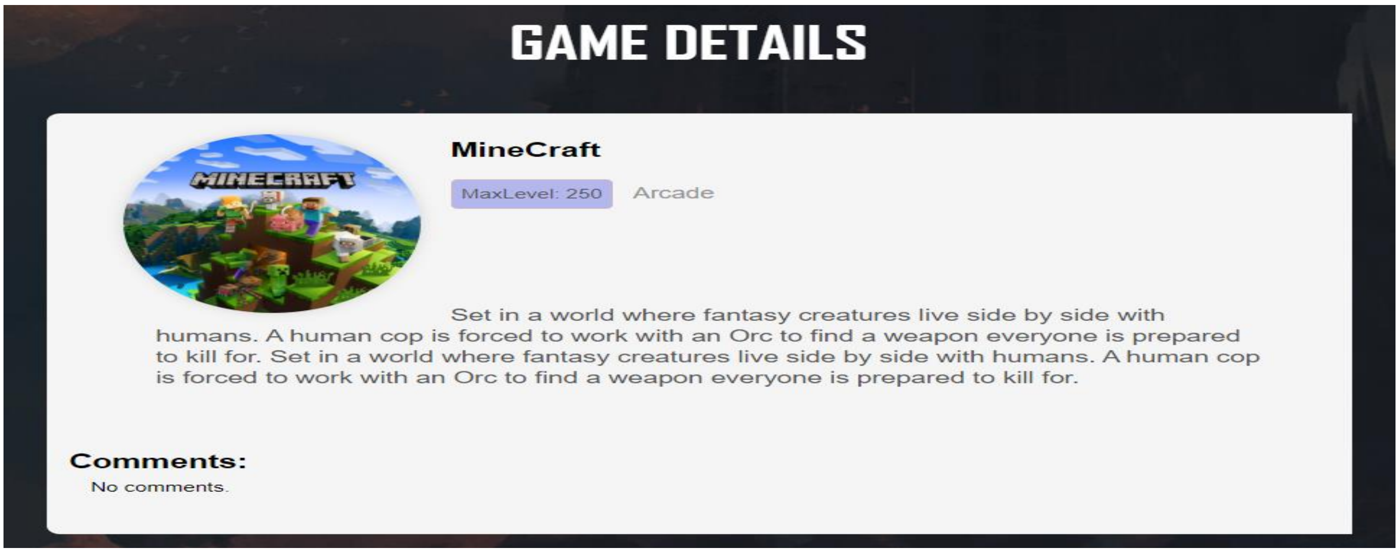
```
Method: DELETE
URL: /data/games/:id
```

Where ***:id*** is the **id** of the desired game. Required **headers** are described in the documentation. The service will return an object, containing the deletion time. Upon success, **redirect** the user to the **Home** page.

BONUS: Comments (10 pts)

Every logged-in user should be able to **comments other games**, but **not his own**.

Guest should not be able to see the section `Add new comment,` but should be able to see the section **Comments** The view for **guests** should look like:



GAME DETAILS

MineCraft

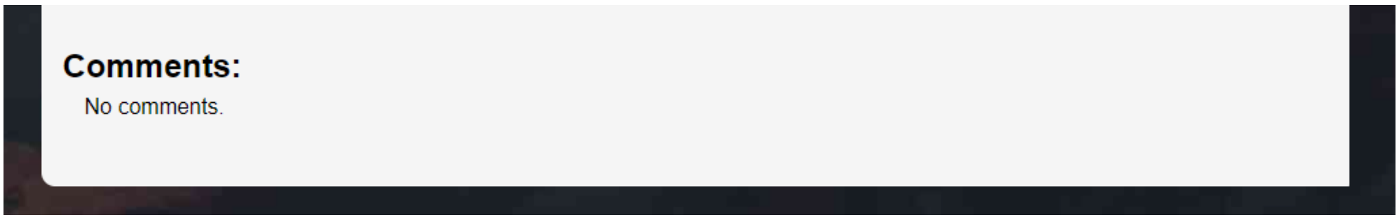
MaxLevel: 250 Arcade

Set in a world where fantasy creatures live side by side with humans. A human cop is forced to work with an Orc to find a weapon everyone is prepared to kill for. Set in a world where fantasy creatures live side by side with humans. A human cop is forced to work with an Orc to find a weapon everyone is prepared to kill for.

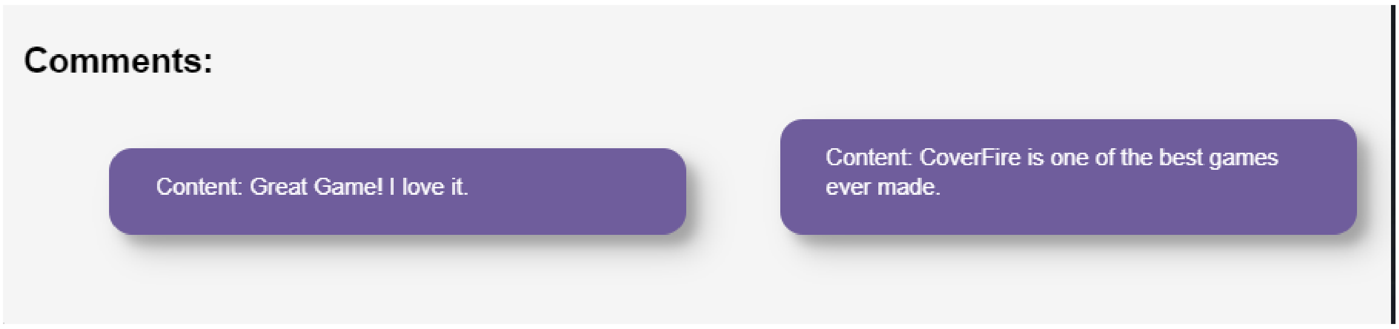
Comments:

No comments.

If there are **no comments all** users should see:

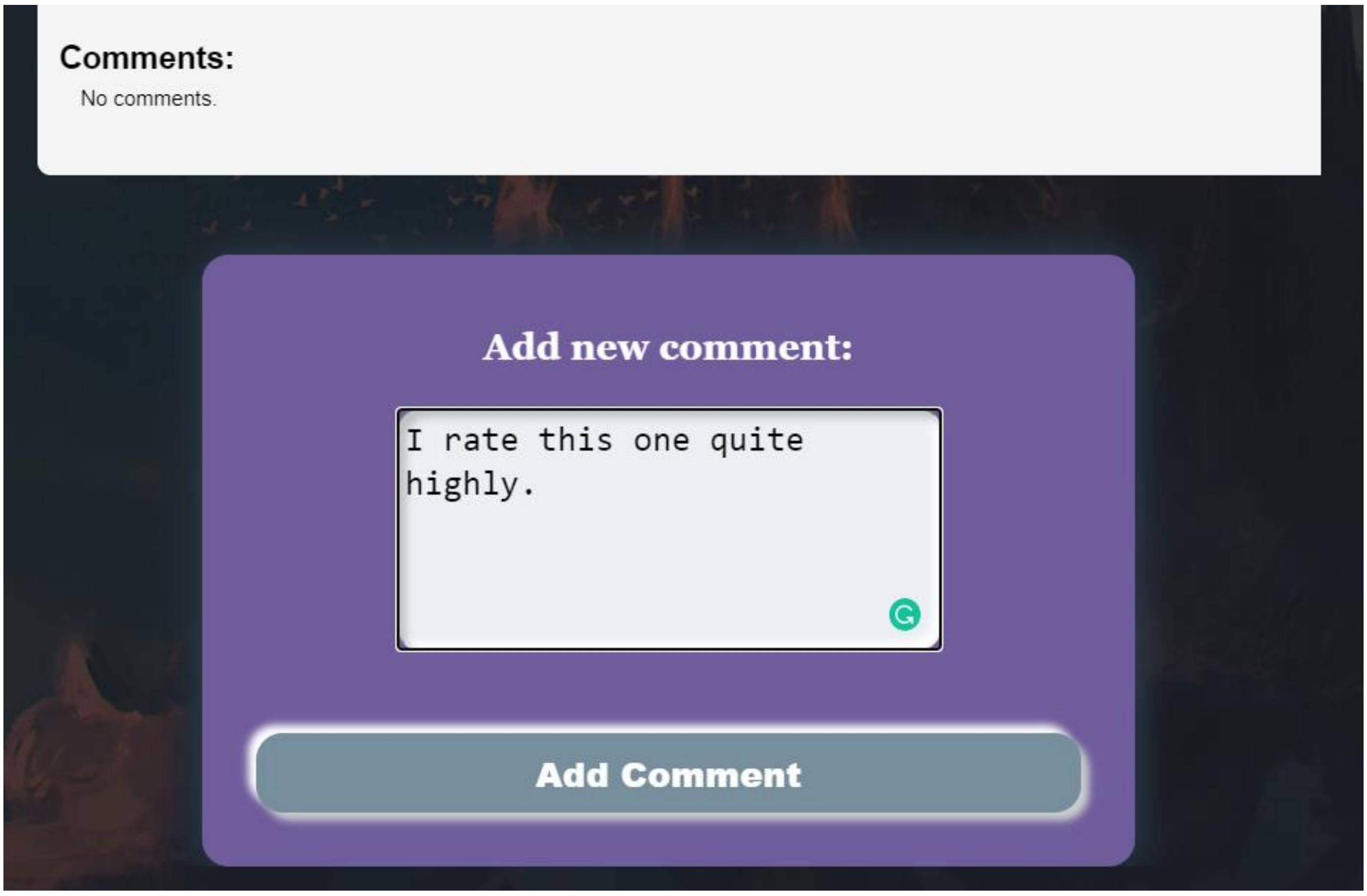

Comments:
No comments.

If there are **comments all** users should see:

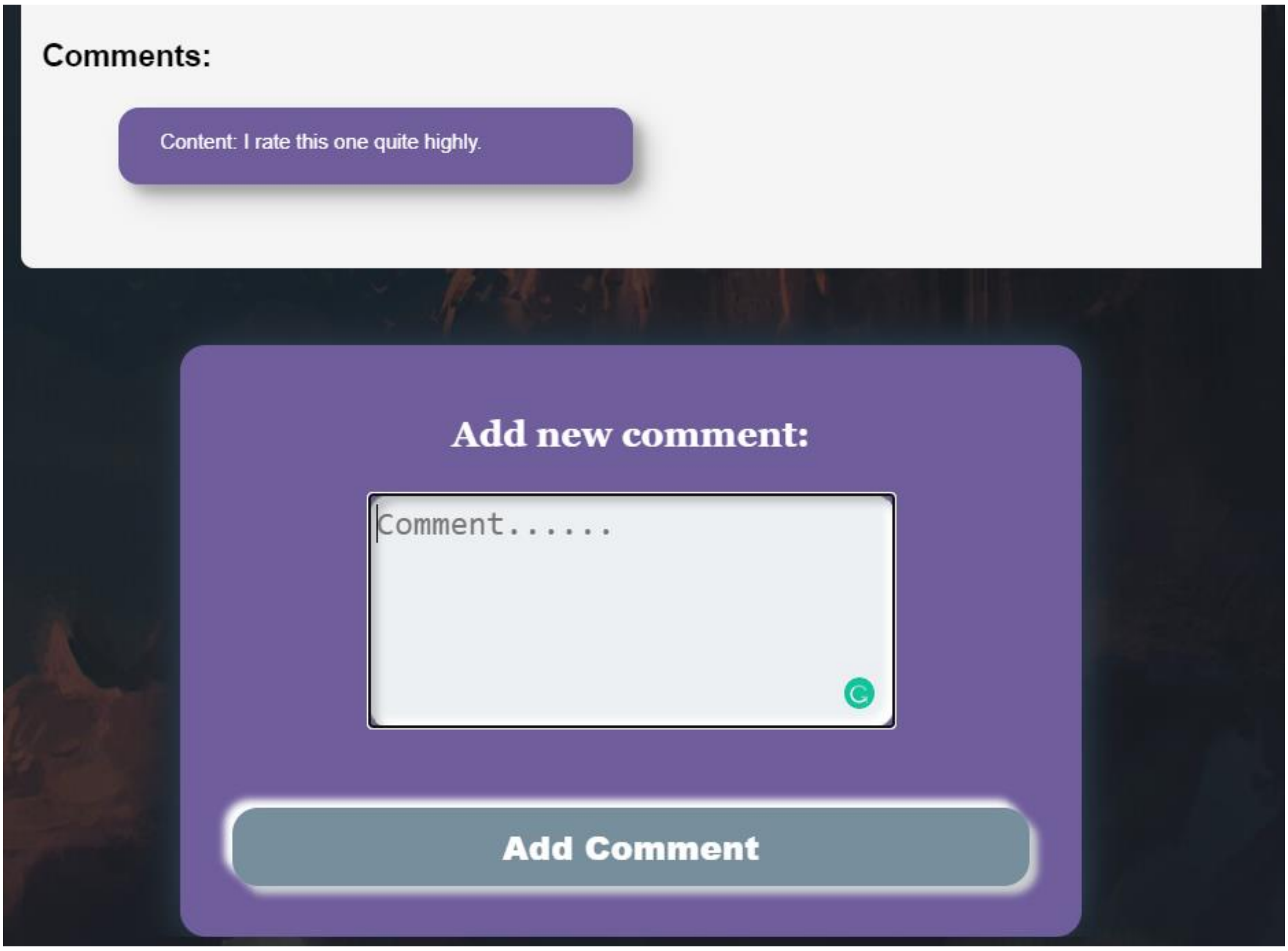

Comments:
Content: Great Game! I love it.
Content: CoverFire is one of the best games ever made.

Logged-in users see a **form** for adding a new comment. Every **registered** user can leave a **comment** under any games. **Authors** can't comment on their **own** games.

The view when there are no comments yet and the user did not press **`[Add Comment]`** button should look like:


Comments:
No comments.
Add new comment:
I rate this one quite highly.

The view when the logged-in user add **Comment** to the game should look like:



Upon success, **clear** the content from the textarea field.

To **load** all comments for game, send the following request:

```
Method: GET
URL: /data/comments?where=gameId%3D%22{gameId}%22
```

Where ***{gameId}*** is the **id** of the desired game

To **create** a new comment, send the following request:

```
Method: POST
URL: /data/comments
```

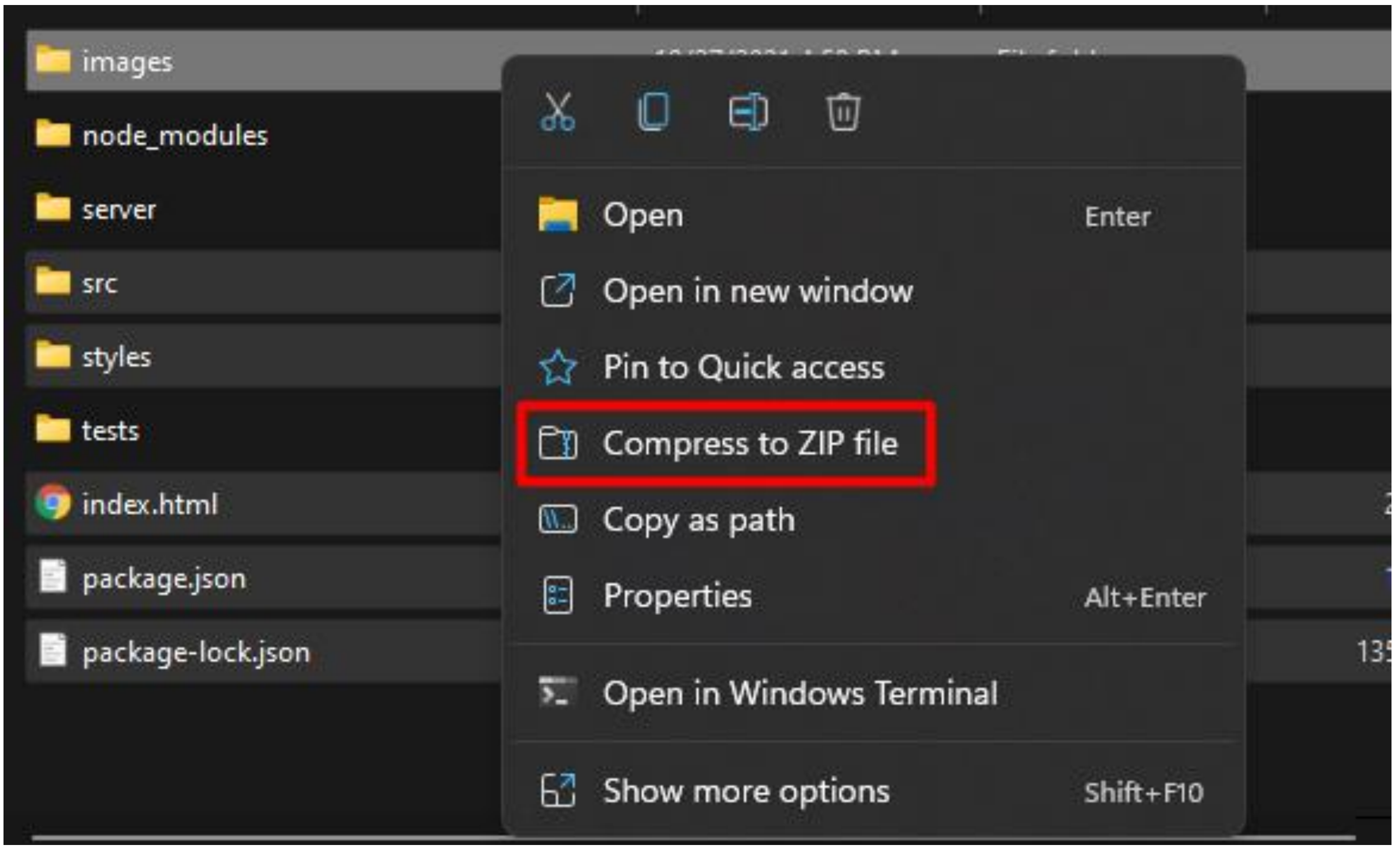
The **service** expects a body with the following shape:

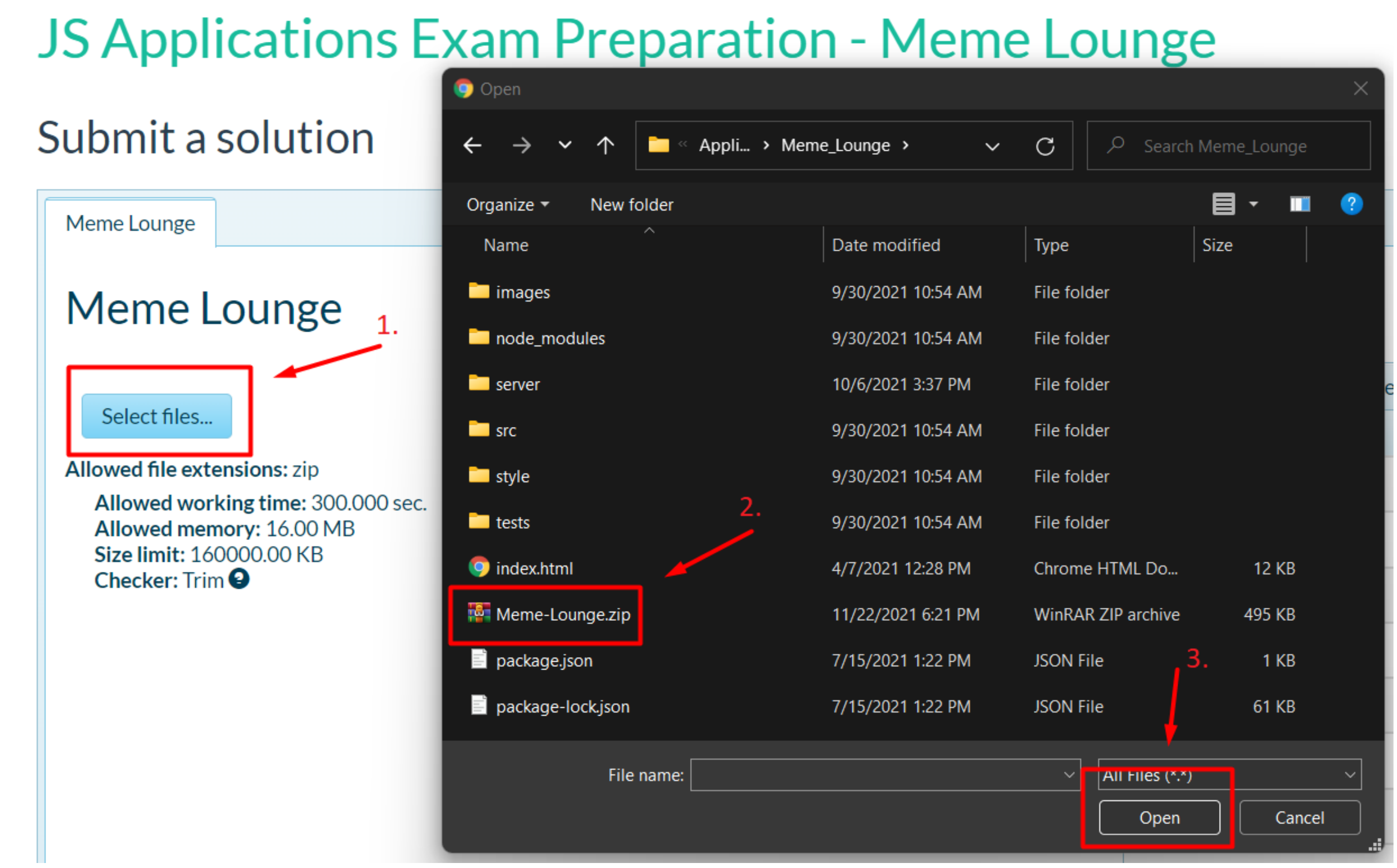
```
{
  gameId,
  comment
}
```

Where **`gameId`** is the **id** of the game, which the comment is associated with, and **`comment`** is the text content. Upon success, **redirect** the user to the **same** page.

4. Submitting Your Solution

Place in a **ZIP** file your project folder. Exclude the **`node_modules`**, **`server`** and **`tests`** folders. Upload the archive to Judge.




JS Applications Exam Preparation - Meme Lounge

Submit a solution

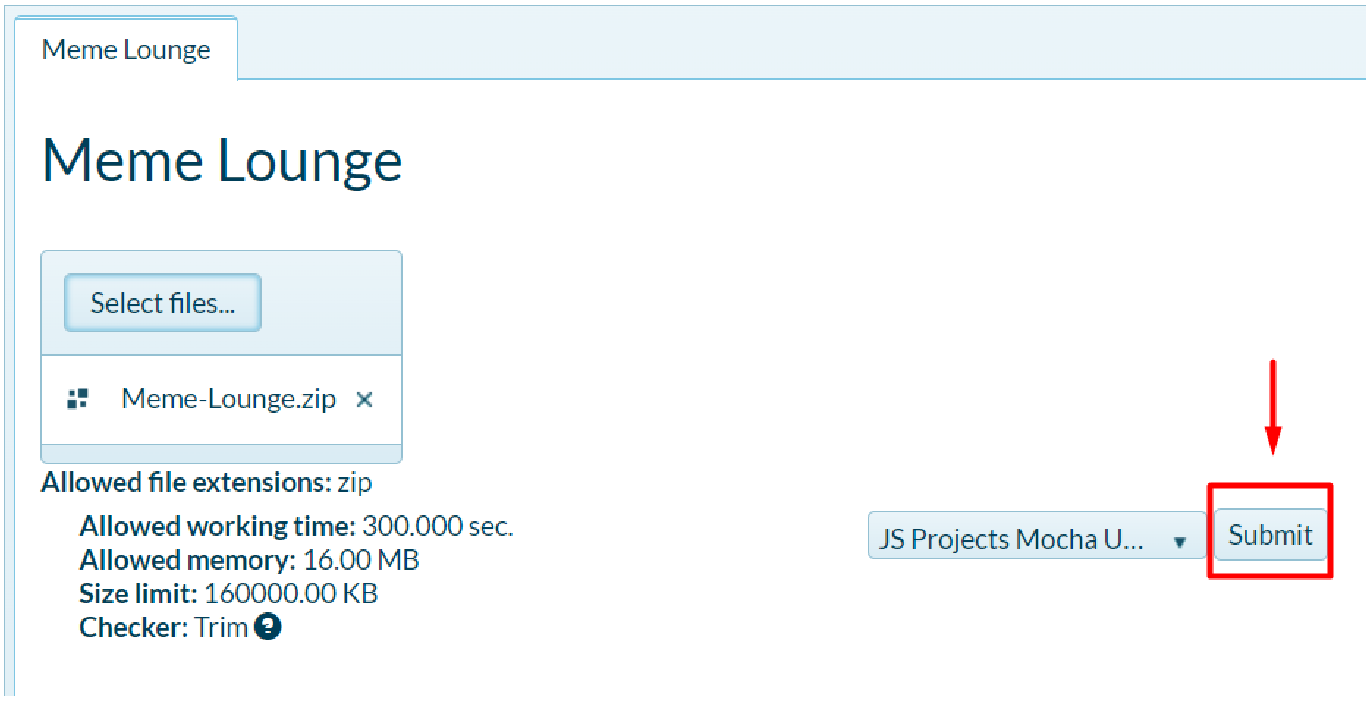
Meme Lounge

Meme Lounge

Select files...

Allowed file extensions: zip
Allowed working time: 300.000 sec.
Allowed memory: 16.00 MB
Size limit: 160000.00 KB
Checker: Trim

Name	Date modified	Type	Size
images	9/30/2021 10:54 AM	File folder	
node_modules	9/30/2021 10:54 AM	File folder	
server	10/6/2021 3:37 PM	File folder	
src	9/30/2021 10:54 AM	File folder	
style	9/30/2021 10:54 AM	File folder	
tests	9/30/2021 10:54 AM	File folder	
index.html	4/7/2021 12:28 PM	Chrome HTML Do...	12 KB
Meme-Lounge.zip	11/22/2021 6:21 PM	WinRAR ZIP archive	495 KB
package.json	7/15/2021 1:22 PM	JSON File	1 KB
package-lock.json	7/15/2021 1:22 PM	JSON File	61 KB


Meme Lounge

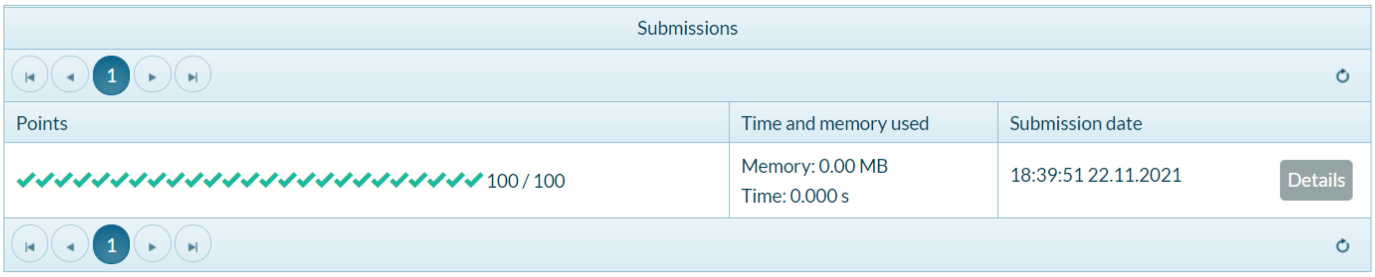
Meme Lounge

Meme-Lounge.zip

Allowed file extensions: zip

Allowed working time: 300.000 sec.
Allowed memory: 16.00 MB
Size limit: 160000.00 KB
Checker: Trim

It will take several minutes for Judge to process your solution!


Submissions

Points	Time and memory used	Submission date	
✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓ 100 / 100	Memory: 0.00 MB Time: 0.000 s	18:39:51 22.11.2021	Details

5. Appendix A: Using the Local REST Service

Starting the Service

The REST service will be in a folder named “server” inside the provided resources archive. It has no dependencies and can be started by opening a terminal in its directory and executing:

```
node server.js
```

If everything initialized correctly, you should see a message about the **host address and port** on which the service will respond to requests.

Sending Requests

To send a request, use the **hostname** and **port**, shown in the initialization log and **resource address** and **method** as described in the **application requirements**. If data needs to be included in the request, it must be **JSON-encoded**, and the appropriate **Content-Type header** must be added. Similarly, if the service is to return data, it will be JSON-encoded. Note that **some requests do not return a body** and attempting to parse them will throw an exception.

Read requests, as well as login and register requests do not require authentication. All other requests must be authenticated.

Required Headers

To send data to the server, include a **Content-Type** header and encode the body as a JSON-string:

```
Content-Type: application/json
{JSON-encoded request body as described in the application requirements}
```

To perform an authenticated request, include an **X-Authorization** header, set to the value of the **session token**, returned by an earlier login or register request:

```
X-Authorization: {session token}
```

Server Response

Data response:

```
HTTP/1.1 200 OK
Access-Contrl-Allow-Origin: *
Content-Type: application/json
{JSON-encoded response data}
```

Empty response:

```
HTTP/1.1 204 No Content
Access-Contrl-Allow-Origin: *
```

Error response:

```
HTTP/1.1 400 Request Error
Access-Contrl-Allow-Origin: *
Content-Type: application/json
{JSON-encoded error message}
```

More Information

You can find more details on the GitHub repository of the service.

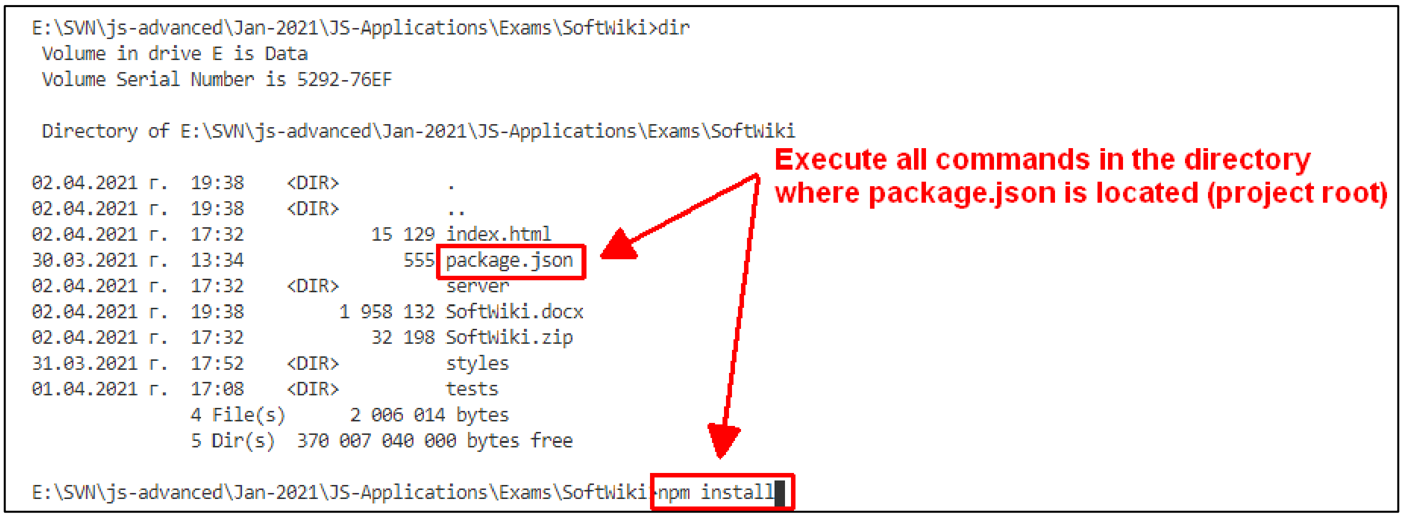
6. Appendix B: Running the Test Suite

Project Setup

The tests require a web server to deliver the content of the application. There is a development web server included in the project scaffold, but you may use whatever server you are familiar with. Note that specialized tools like **BrowserSync** may interfere with the tests. To initialize the project with its dependencies, open a terminal in the folder, containing the file **`package.json`** and execute the following:

```
npm install
```

Note that if you changed the section **`devDependencies`** of the project, the tests may not initialize properly.



```
E:\SVN\js-advanced\Jan-2021\JS-Applications\Exams\SoftWiki>dir
 Volume in drive E is Data
 Volume Serial Number is 5292-76EF

 Directory of E:\SVN\js-advanced\Jan-2021\JS-Applications\Exams\SoftWiki

02.04.2021 г.  19:38    <DIR>          .
02.04.2021 г.  19:38    <DIR>          ..
02.04.2021 г.  17:32            15 129 index.html
30.03.2021 г.  13:34               555 package.json
02.04.2021 г.  17:32    <DIR>          server
02.04.2021 г.  19:38         1 958 132 SoftWiki.docx
02.04.2021 г.  17:32            32 198 SoftWiki.zip
31.03.2021 г.  17:52    <DIR>          styles
01.04.2021 г.  17:08    <DIR>          tests
               4 File(s)      2 006 014 bytes
               5 Dir(s)  370 007 040 000 bytes free

E:\SVN\js-advanced\Jan-2021\JS-Applications\Exams\SoftWiki>npm install
```

Execute all commands in the directory
where package.json is located (project root)

Executing the Tests

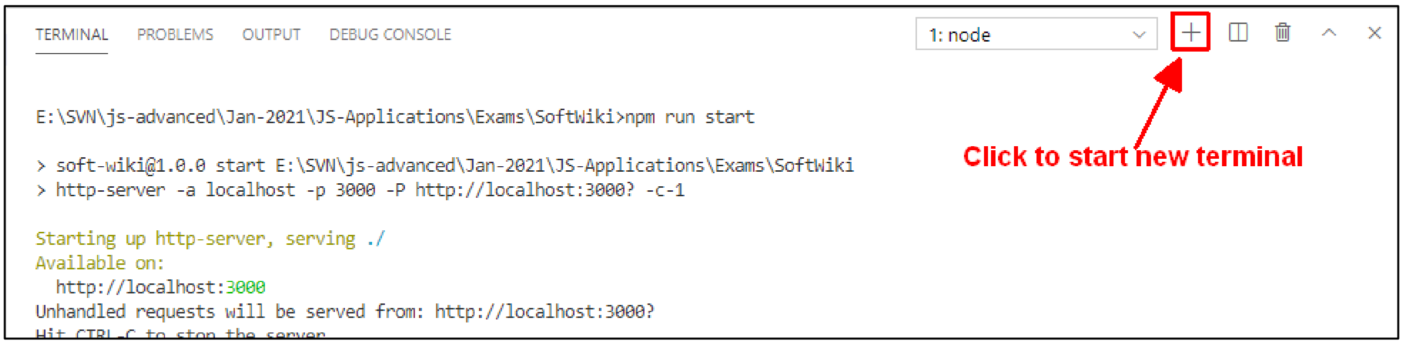
Before running the test suite, make sure a web server is operational, and the application can be found at the root of its network address. To start the included dev-server, open a terminal in the folder containing **`package.json`** and execute:

```
npm run start
```

This is a one-time operation unless you terminate the server at any point. It can be restarted with the same command as above.

To execute the tests, open a new terminal (do not close the terminal, running the web server instance) in the folder containing **`package.json`** and execute:

```
npm run test
```



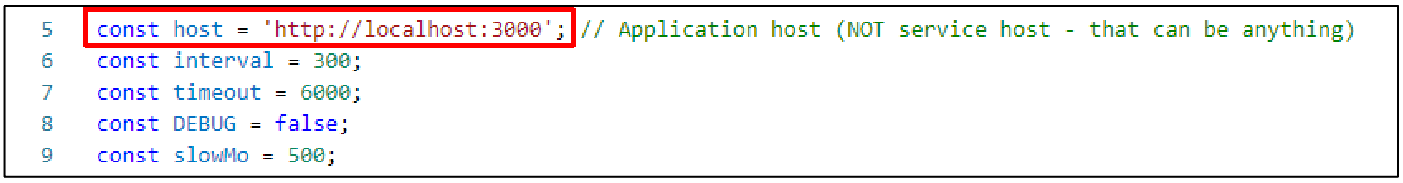
```
E:\SVN\js-advanced\Jan-2021\JS-Applications\Exams\SoftWiki>npm run start

> soft-wiki@1.0.0 start E:\SVN\js-advanced\Jan-2021\JS-Applications\Exams\SoftWiki
> http-server -a localhost -p 3000 -P http://localhost:3000? -c-1

Starting up http-server, serving ./
Available on:
  http://localhost:3000
Unhandled requests will be served from: http://localhost:3000?
Hit CTRL-C to stop the server
```

Test results will be displayed in the terminal, along with detailed information about encountered problems. You can perform this operation as many times as it is necessary by re-running the above command.

Debugging Your Solution

If a test fails, you can view detailed information about the requirements that were not met by your application. Open the file **e2e.test.js** in the folder **tests** and navigate to the desired section as described below.

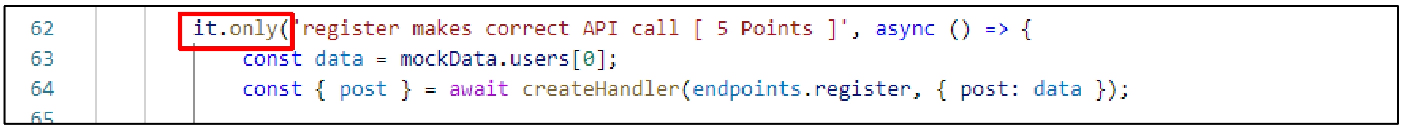
This first step will not be necessary if you are using the included web server. Make sure the application host is set correctly:



```
const host = 'http://localhost:3000'; // Application host (NOT service host - that can be anything)
const interval = 300;
const timeout = 6000;
const DEBUG = false;
const slowMo = 500;
```

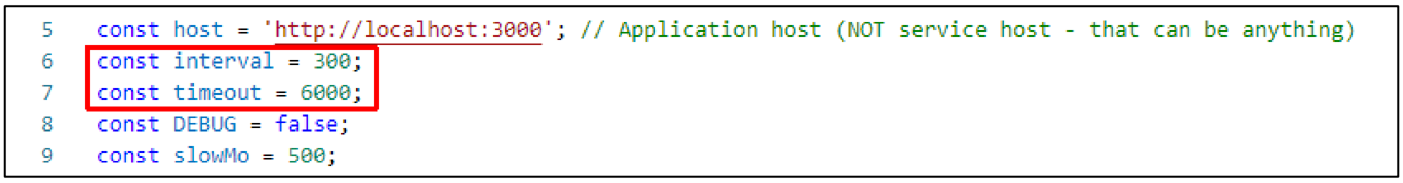
The value for **host** must be the address where your application is being served. Make sure that entering this address in a regular internet browser shows your application.

To make just a single test run, instead of the full suite (useful when debugging a single failing test), find the test and append **.only** after the **it** reference:



```
    it.only('register makes correct API call [ 5 Points ]', async () => {
        const data = mockData.users[0];
        const { post } = await createHandler(endpoints.register, { post: data });
```

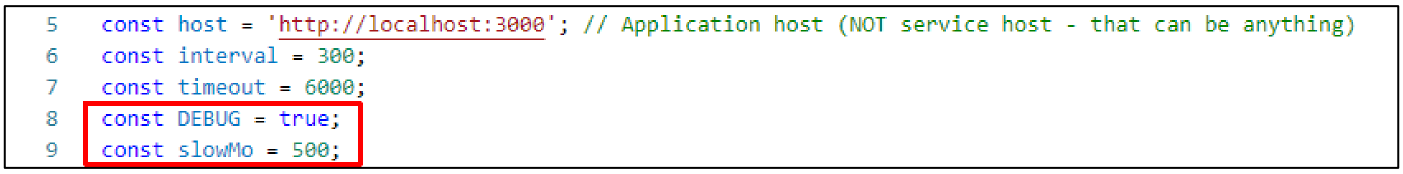
On slower machines, some of the tests may require more time to complete. You can instruct the tests to run more slowly by slightly increasing the values for **interval** and **timeout**:



```
const host = 'http://localhost:3000'; // Application host (NOT service host - that can be anything)
const interval = 300;
const timeout = 6000;
const DEBUG = false;
const slowMo = 500;
```

Note that **interval** values greater than 500 and **timeout** values greater than 10000 are not recommended.

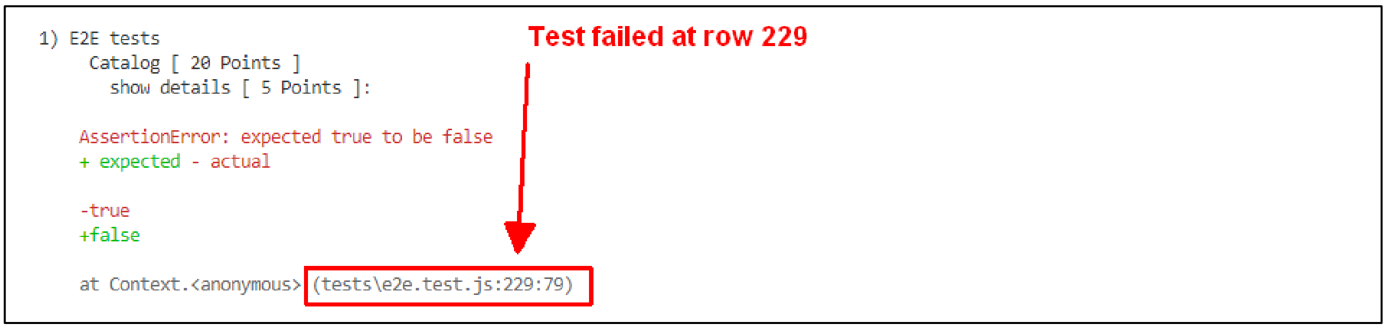
If this doesn't make the test pass, set the value of **DEBUG** to **true** and run the tests again – this will launch a browser instance and allow you to see what is being tested, what the test sees and where it fails (or hangs):



```
const host = 'http://localhost:3000'; // Application host (NOT service host - that can be anything)
const interval = 300;
const timeout = 6000;
const DEBUG = true;
const slowMo = 500;
```

If the actions are happening too fast, you can increase the value of **slowMo**. If the browser hangs, you can just close it and abort any remaining tests by focusing the terminal window and pressing **[Ctrl+C]** followed by the letter "y" and **[Enter]**.

The final thing to look for is the exact row where the test fails:



```
1) E2E tests
     Catalog [ 20 Points ]
       show details [ 5 Points ]:

    AssertionError: expected true to be false
    + expected - actual

    -true
    +false

    at Context.<anonymous> (tests\e2e.test.js:229:79)
```