

JS Applications Exam Preparation – Meme Lounge

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.

1. Overview

Implement a front-end app (SPA) for viewing and managing **memes**. The application allows visitors to browse through the memes catalog. Users may **register** with a **username**, **email**, **password** and **gender**, which allows them to **create** their own posts. Post authors can also **edit** or **delete** their own publications 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.

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)

Implement a **NavBar** for the app: navigation links should correctly change the current screen (view).

Navigation links should correctly change the current page (view). **Guests** (un-authenticated visitors) can see the links to the **Home Page**, **All Memes** page, as well as the links to the **Login** and **Register** pages. The logged-in user navbar should contain the links to **All Memes** page, the **Create Meme** page, **Welcome, { user's email address }**, **My Profile** page and a link for the **Logout** action.

User navigation example:



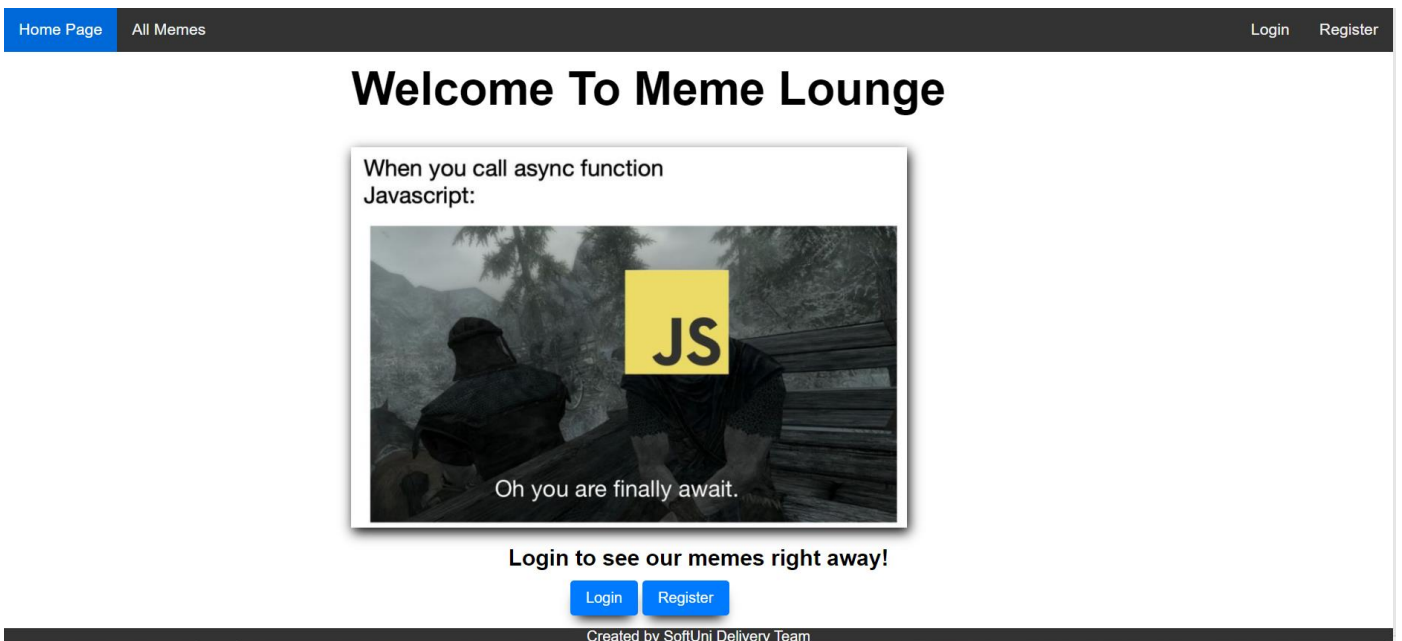
Guest navigation example:



Home Screen (5 pts)

The initial screen should display the navigation, register, login and the initial image + footer.

Note: This page should be only visible for guest users. Logged-in users should be redirected to the **All Memes** page.



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": "mary@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.

Login

Email

Password

Login

Dont have an account? [Sign up.](#)

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

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

Register User (10 pts)

By given **username, email, password and gender (Male or Female)** the app should register a new user in the system. All fields are required – if any of them is empty, display an error.

Register

Username

Email

Password

Repeat Password

☐ Female ☒ Male

Register

Already have an account? [Sign in.](#)

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:

```
{
  username,
  email,
  password,
  gender
}
```

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 **All Memes** 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.

Create Meme Screen (15 pts)

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

Create Meme

Title

Enter Title

Description

Enter Description

Meme Image

Enter meme ImageUrl

Create Meme

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

Method: POST
URL: /data/memes

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

```
{  
  title,  
  description,  
  imageUrl  
}
```

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

All Memes (10 pts)

This page displays a list of all memes in the system. Clicking on the details button in the cards leads to the details page for the selected meme. This page should be visible to guests and logged-in users.



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

No memes in database.

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

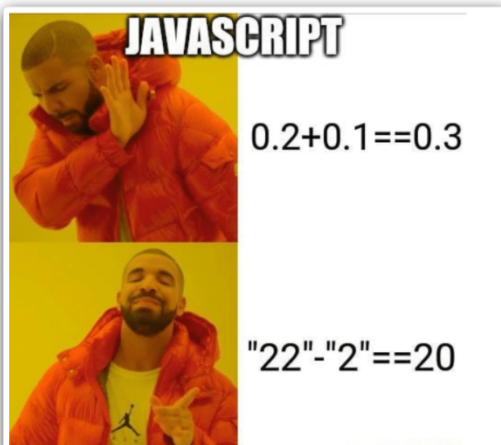
Method: GET
URL: /data/memes?sortBy=_createdOn%20desc

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

Meme Details (10 pts)

All users should be able to **view details** about memes. Clicking the **Details** link in of a **meme card** should **display** the **Details** page. If the currently **logged-in user is the creator** of the meme, the **Edit** and **Delete** buttons should be displayed, otherwise they should not be available.

Meme Title: Java Script



Meme Description

Programming is often touted as a smart and lucrative career path. It's a job that (sometimes) offers flexibility and great benefits. But it's far from sunshine and Nyan Cat rainbows. The hours are long. The mistakes are frustrating. And your eyesight is almost guaranteed to suffer. These memes cover most of the frustration (and funny moments) of programming. At least we can laugh through the pain.

[Edit](#)[Delete](#)

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

Method: GET

URL: /data/memes/:*id*

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

Edit Meme Screen (15 pts)

The Edit page is available to logged-in users and it allows authors to **edit** their **own** memes. Clicking the **Edit** link of a particular meme on the **Details** page should display the **Edit** page, with all fields filled with the data for the meme. It contains a form with input fields for all relevant properties. Check if all the fields are filled before you send the request.

Edit Meme

Title

Java Script

Description

Programming is often touted as a smart and lucrative career path. It's a job that (sometimes) offers flexibility and great benefits. But it's far from sunshine and Nyan Cat rainbows. The hours are long. The mistakes are frustrating. And your eyesight is almost

Image Url

/images/4.png

Edit Meme

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

Method: PUT
URL: /data/memes/:*id*

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

The service expects a body with the following shape:

```
{
  title,
  description,
  imageUrl
}
```

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

Delete Meme (10 pts)

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


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

Method: DELETE
URL: /data/memes/:*id*

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

User Profile (10 pts)


Each **logged-in user** should be able to view his own profile by clicking [**My Profile**]. Username, Email and My memes count should be filled with the data for the current user. Note that the Gender of the user determines which picture is displayed as their avatar.



Username: Peter
Email: peter@abv.bg
My memes count: 3

User Memes

Debugging



Details

Java Script



Details

Yes, arrays are ob...

Details

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



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

Method: GET

URL: /data/memes?where=_ownerId%3D%22{userId}%22&sortBy=_createdOn%20desc

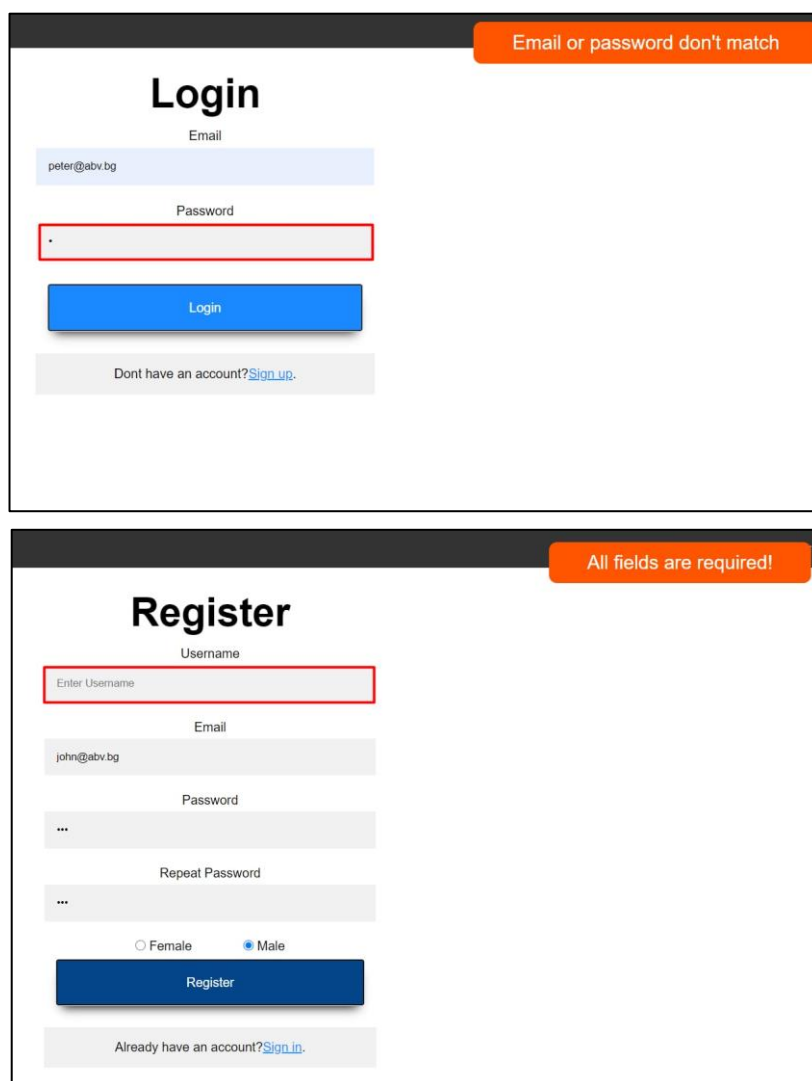
Where **{userId}** is the id of the currently logged-in user.

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

BONUS: Notifications (5 pts)

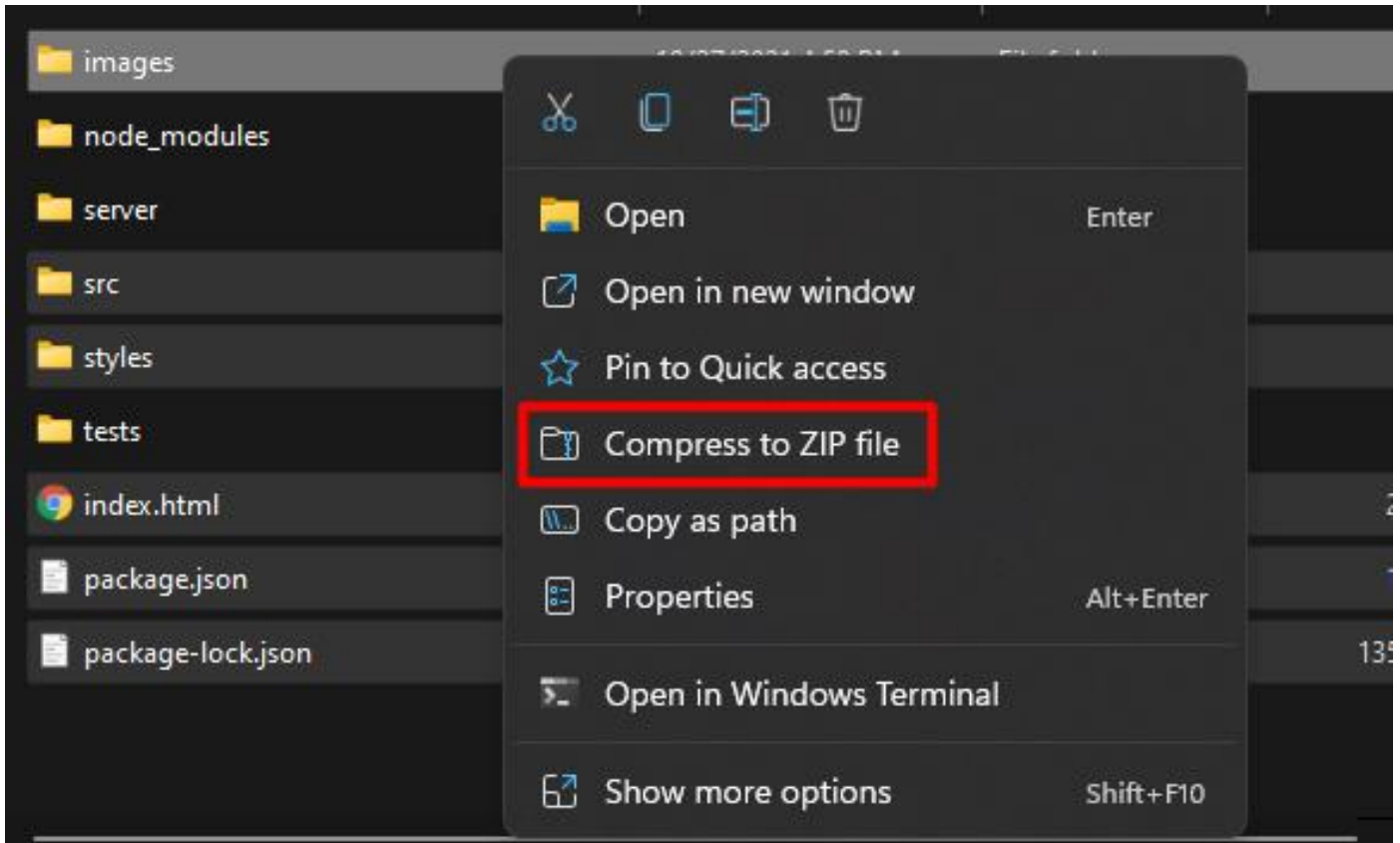
In case of **error caused by the user's actions**, the application should display an **error notification message**, which disappears after 3 seconds. There is a **styled section** with id "notifications" in the provided HTML file.

Errors may include **validation errors** or error **messages returned** by the REST service, such as incorrect user credentials, on the following pages: **Register**, **Login**, **Create** and **Edit**.

Two screenshots of web application forms. The top screenshot is the "Login" form, which has a title "Login" and a subtitle "Email". It contains an email input field with the value "peter@abv.bg", a password input field with a red border and a red error message "Email or password don't match", a blue "Login" button, and a link "Don't have an account? Sign up.". The bottom screenshot is the "Register" form, which has a title "Register" and a subtitle "Username". It contains a username input field with a red border and a red error message "All fields are required!", an email input field with the value "john@abv.bg", a password input field with a red border and a red error message "All fields are required!", a repeat password input field with a red border and a red error message "All fields are required!", radio buttons for "Female" and "Male" (with "Male" selected), a blue "Register" button, and a link "Already have an account? Sign in.".

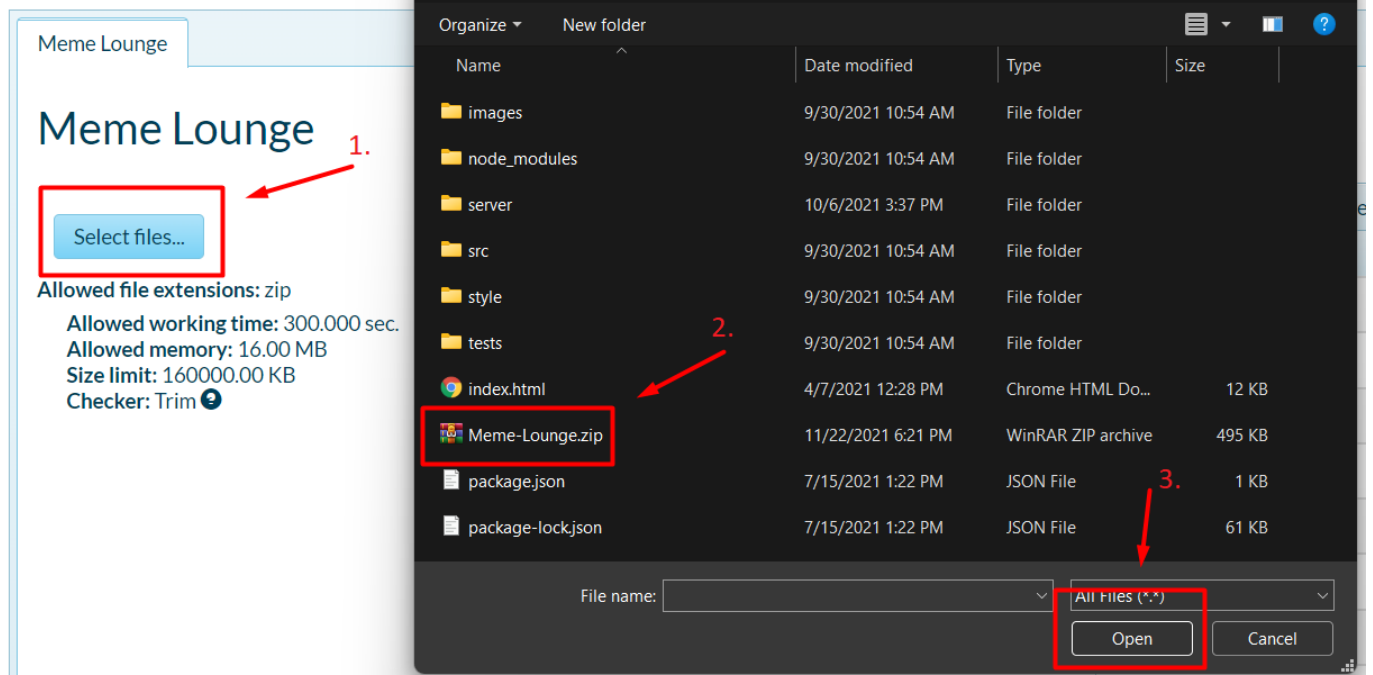
Submitting Your Solution

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



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-Control-Allow-Origin: `*`

Content-Type: `application/json`

{JSON-encoded response data}

Empty response:

HTTP/1.1 204 No Content

Access-Control-Allow-Origin: `*`

Error response:

HTTP/1.1 400 Request Error

Access-Control-Allow-Origin: `*`

Content-Type: `application/json`

{JSON-encoded error message}

More Information

You can find more details on the [GitHub repository of the service](#).

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  r.   19:38    <DIR>        .
02.04.2021  r.   19:38    <DIR>        ..
02.04.2021  r.   17:32         15 129 index.html
30.03.2021  r.   13:34        555 package.json
02.04.2021  r.   17:32    <DIR>        server
02.04.2021  r.   19:38       1 958 132 SoftWiki.docx
02.04.2021  r.   17:32       32 198 SoftWiki.zip
31.03.2021  r.   17:52    <DIR>        styles
01.04.2021  r.   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

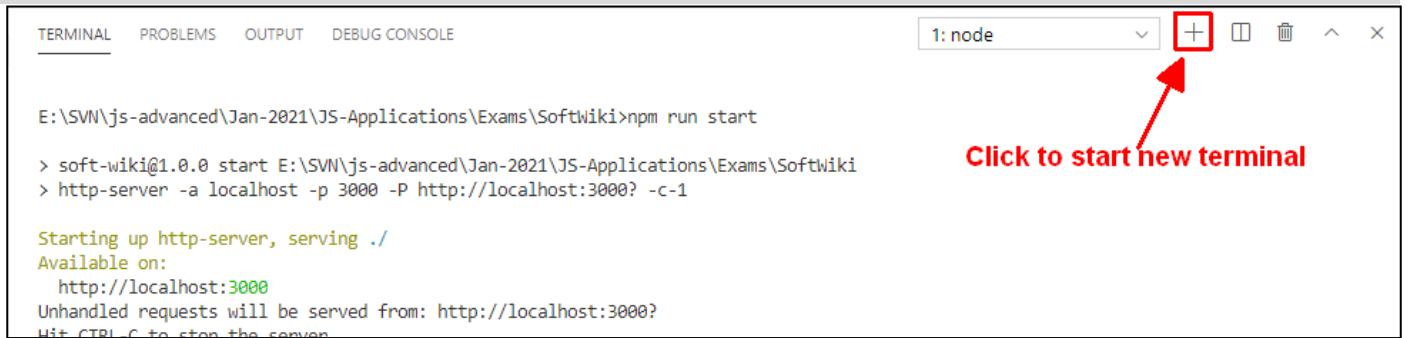
Before running the test suite, make sure a web server is operational, and the application can be found at the root of its network address. To start the included dev-server, open a terminal in the folder containing **package.json** and execute:

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



Click to start new terminal

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:

```

5  const host = 'http://localhost:3000'; // Application host (NOT service host - that can be anything)
6  const interval = 300;
7  const timeout = 6000;
8  const DEBUG = false;
9  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:

```

62  it.only('register makes correct API call [ 5 Points ]', async () => {
63      const data = mockData.users[0];
64      const { post } = await createHandler(endpoints.register, { post: data });
65

```

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**:

```

5  const host = 'http://localhost:3000'; // Application host (NOT service host - that can be anything)
6  const interval = 300;
7  const timeout = 6000;
8  const DEBUG = false;
9  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):

```

5  const host = 'http://localhost:3000'; // Application host (NOT service host - that can be anything)
6  const interval = 300;
7  const timeout = 6000;
8  const DEBUG = true;
9  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)

```

Test failed at row 229