Assignment for automation tester 2

Timeline: 5 days

General Requirements:

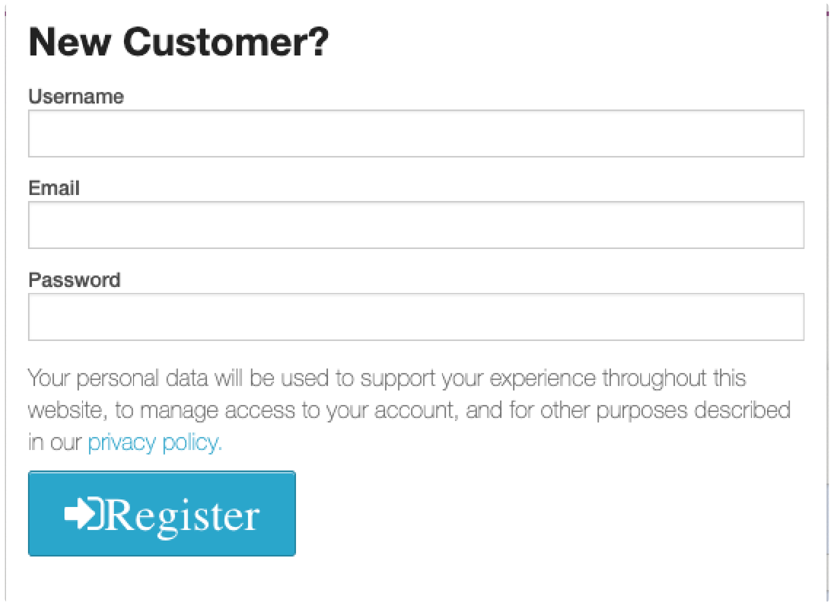
- The Readme document should provide guidelines on how to set up and execute tests.
- Your working solution must be pushed to the git repository (GitHub: Let's build from here).

PART 1 - Test Design

Feature - Register a customer

As a guest buyer, I would like to create an account so that I can place an order and write a review

Mockup



New Customer?

Username

Email

Password

Your personal data will be used to support your experience throughout this website, to manage access to your account, and for other purposes described in our privacy policy.

Requirements:

Design and write test cases using a template that provides enough information for test execution and report.

What we expected:

- The candidate should be able to apply testing techniques in designing test cases.
- Clearly categorize different testing types for each designed test case.
- Both functional and non-functional should be covered.
- Test cases should be ready for automation when needed.

PART 2 - UI Automation Test

Context

We would like to get a sense of your abilities to create UI tests. For that, we will use a project called the-internet, which is a simple web-based mock application that provides common web page scenarios. For your convenience, this is the GIT repo for the project. GitHub - s aucelabs/the-internet: An example application that captures prominent and ugly functionality found on the web. Perfect for writing automated acceptance tests against.

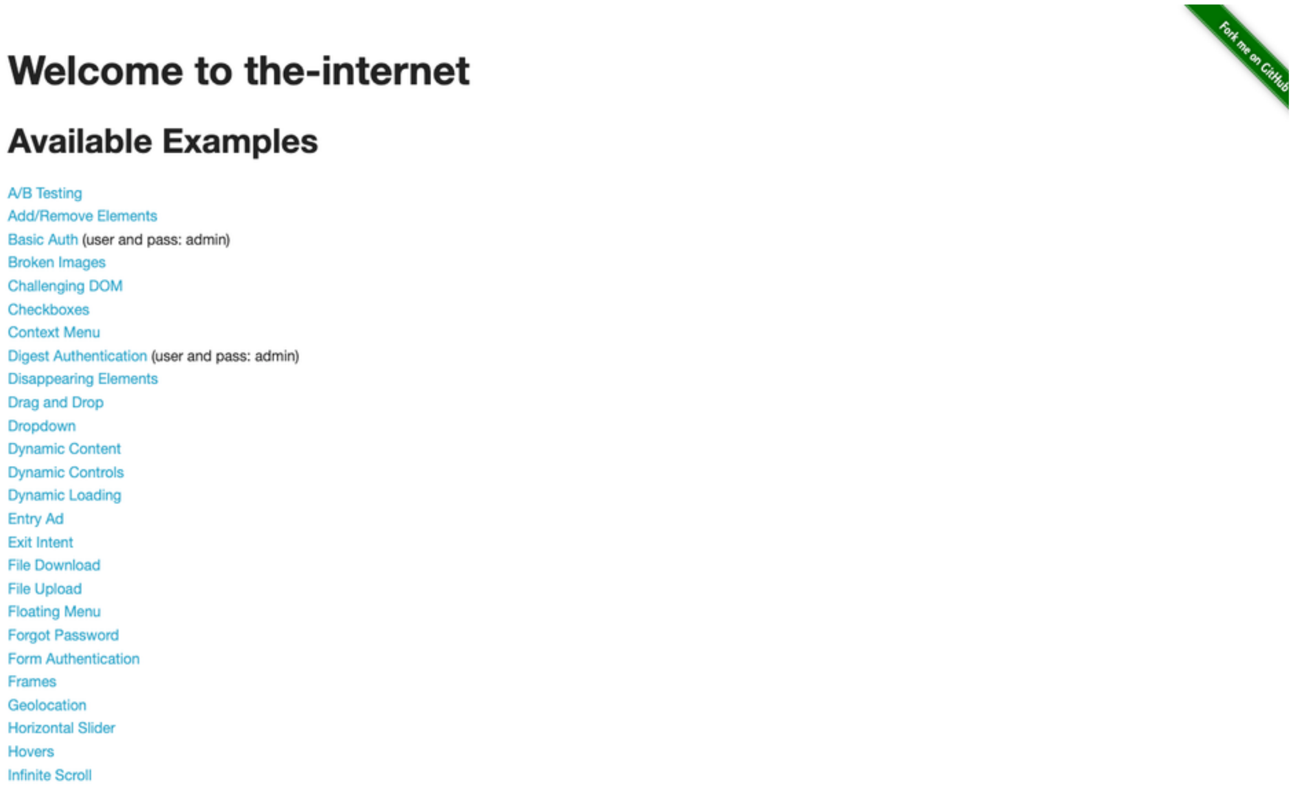
Setup Guideline

- Run the website locally on port 8080 by using Docker

```
docker run --rm -d -p 8080:5000 gprestes/the-internet
```

⚠ If you use an ARM computer add --platform linux/amd64 to the docker run parameters

- Navigate to http://localhost:8080, you should see the following page:



Welcome to the-internet

Available Examples

A/B Testing
Add/Remove Elements
Basic Auth (user and pass: admin)
Broken Images
Challenging DOM
Checkboxes
Context Menu
Digest Authentication (user and pass: admin)
Disappearing Elements
Drag and Drop
Dropdown
Dynamic Content
Dynamic Controls
Dynamic Loading
Entry Ad
Exit Intent
File Download
File Upload
Floating Menu
Forgot Password
Form Authentication
Frames
Geolocation
Horizontal Slider
Hovers
Infinite Scroll

Requirements:

Design and write automation scripts, using any automation frameworks/tools that you are currently familiar with.

1. **Form Authentication (Login Page):**
   - Verify that users can successfully log in with valid credentials.
   - Verify that an error message appears for invalid credentials.
2. **Checkboxes -** Verify that checkboxes can be checked and unchecked.
3. **Dropdown -** Verify that each option in a dropdown can be selected.
4. **File uploads -** Verify that a file can be successfully uploaded.

What we expected:

- Show valid verification and assertion points.
- Well-structured and designed
- Follow coding standards.

PART 3 - API Automation Test

Context

We want to see how you approach testing automation. For that, we chose to use json-server which provides a simple REST API. The input to json-server is a JSON file named db.json, the structure of the json file creates a REST API that allows you to use CRUD operations using Postman or even code.

For your general knowledge, this is the repository for it: GitHub - typicode/json-server: Get a full fake REST API with zero coding in less than 30 seconds (seriously). We will again ask you to use Docker to have a local running version of the code.
Since our company deals with e-commerce, the `db.json` file is a mock of a simple e-commerce store.
The store will run on port 3000 on your local computer.

Setup Guideline

- Create a `db.json` file with data as the following

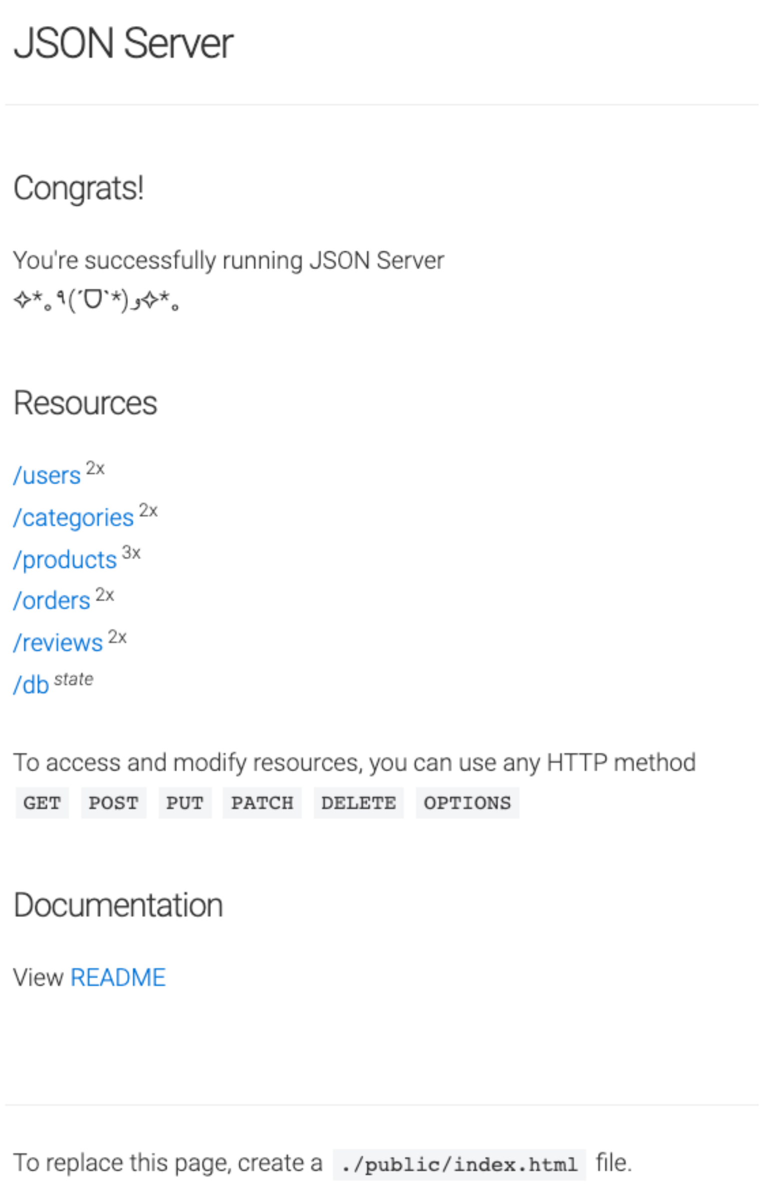
```
{
  "users": [
    {
      "id": 1,
      "username": "john_doe",
      "email": "john@example.com",
      "password": "hashed_password_here"
    },
    {
      "id": 2,
      "username": "jane_doe",
      "email": "jane@example.com",
      "password": "hashed_password_here"
    }
  ],
  "categories": [
    {
      "id": 1,
      "name": "Electronics"
    },
    {
      "id": 2,
      "name": "Clothing"
    }
  ],
  "products": [
    {
      "id": 1,
      "name": "iPhone 13",
      "description": "Latest Apple iPhone",
      "price": 999,
      "stock": 20,
      "categoryId": 1
    },
    {
      "id": 2,
      "name": "Samsung Galaxy S21",
      "description": "Latest Samsung Phone",
      "price": 899,
      "stock": 15,
      "categoryId": 1
    },
    {
      "id": 3,
      "name": "T-shirt",
      "description": "Cotton T-shirt",
      "price": 20,
      "stock": 100,
      "categoryId": 2
    }
  ],
  "orders": [
    {
      "id": 1,
      "userId": 1,
      "status": "shipped",
      "items": [
        {
          "productId": 1,
          "quantity": 1
        }
      ]
    },
    {
      "id": 2,
      "userId": 2,
      "status": "processing",
      "items": [
        {
          "productId": 3,
          "quantity": 2
        },
        {
          "productId": 1,
          "quantity": 1
        }
      ]
    }
  ],
  "reviews": [
    {
      "id": 1,
      "userId": 1,
      "productId": 1,
      "rating": 5,
      "comment": "Great product!"
    },
    {
      "id": 2,
      "userId": 2,
      "productId": 2,
      "rating": 4,
      "comment": "Very good, but could be better."
    }
  ]
}
```

- Run the following command to start the docker container:

```
docker run -p 3000:80 -v ./db.json:/data/db.json clue/json-server
```

⚠ If you use an ARM computer add --platform linux/amd64 to the docker run parameters

- Open your browser with `http://localhost:3000`
- You should see the following page:



JSON Server

Congrats!

You're successfully running JSON Server
✧*｡٩(ˊᗜˋ*)و✧*｡

Resources

/users 2x
/categories 2x
/products 3x
/orders 2x
/reviews 2x
/db state

To access and modify resources, you can use any HTTP method
GET POST PUT PATCH DELETE OPTIONS

Documentation

View README

To replace this page, create a `./public/index.html` file.

Requirements:

Design and write automation scripts, using any tool that you feel comfortable working with (Postman, Jmeter, LocustIO, ...).

1. Perform functionality test for creating a user
   Endpoint: `/users`
   Method: `POST`
2. Write a flow:
   - Create a user successfully
   - Search products with a category named: “Electronics“ successfully

  - Place an order with a product of the previous step successfully
  - Write a review for the purchased product successfully

What we expected:

- Tests should cover all aspects of the API, including input validation, outputs, and error handling.
- Be able to leverage automation capability for automation tests.
  - Data input and output follow some specific templates or models so that you can create test scripts only once.
  - Automation with a data-driven approach.