

2011907 Cloud Based Web Services

by Josh Archer

Submission date: 16-Jan-2024 10:36PM (UTC+0000)

Submission ID: 222292287

File name: 24846_Josh_Archer_2011907_Cloud-Based_Web_Services_536492_1495811470.docx (633.29K)

Word count: 2050

Character count: 11828



CLOUD BASED WEB SERVICES

ACCA6024

Josh Archer
2011907

Josh Archer
2011907@student.uwtsd.ac.uk
Page 0 of 17

Contents

1.	Report Overview	2
1.1	Introduction	2
2.	Main Body	3
2.1	Design Approaches	3
2.1.1	Single Page Application (SPA)	3
2.1.2	Progressive Enhancement	3
2.1.3	Server-Side Rendering (SSR)	3
2.1.4	Client-Side Rendering (CSR)	3
2.2	Purpose and Functionality	5
2.3	Set up and Running of Application	6
3.	Assignment 2 Overview	9
3.1	Introduction	9
4.	Assignment 2 Main Body	10
4.1	Evaluation of Documented Information on Software Testing	10
4.2	Test Summary Report Example	12
4.3	Testing of Applications Code	13

1. Report Overview

1.1 Introduction

The aim of this project is to create a dynamic web application that utilizes AJAX and a RESTful web API to exchange data with one or more web services. The application will cater to the needs of an organization, and for the purpose of this demonstration, let's consider it as a seed selling store.

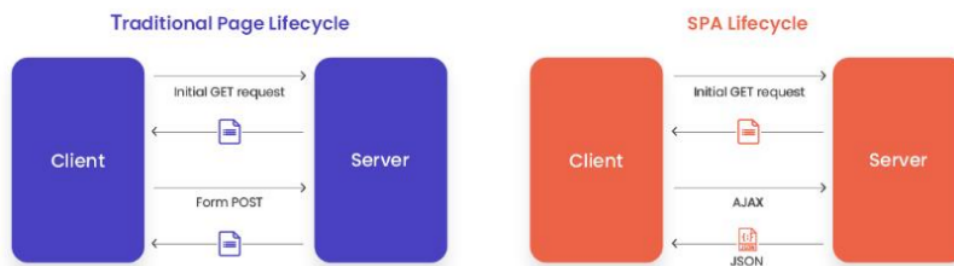
The project will follow creative solutions in both design and implementation to ensure an engaging and efficient user experience. Emphasis will be on meeting user requirements while providing flexibility for future enhancements. The application's success will be measured not only by functionality but also by the clarity and thoroughness of its documentation.

2. Main Body

2.1 Design Approaches

2.1.1 Single Page Application (SPA)

SPAs initiate loading from a solitary HTML page and dynamically modify content as users engage with the application. The Benefit of this approach is it delivers an uninterrupted user experience, accelerates navigation, and lessens server load by retrieving only essential data. However, Initial page loading may experience delays, and there could be complications related to Search Engine Optimization (SEO) [1] .



[2]

2.1.2 Progressive Enhancement

Progressive Enhancement Commence with a fundamental static webpage and gradually introduce improvements using technologies such as JavaScript, CSS, and AJAX. The advantages of this are it Guarantees a fundamental functionality level for all users, irrespective of their device or browser capabilities. The drawbacks however are Development time may extend, and user experience might differ across diverse browsers [3].

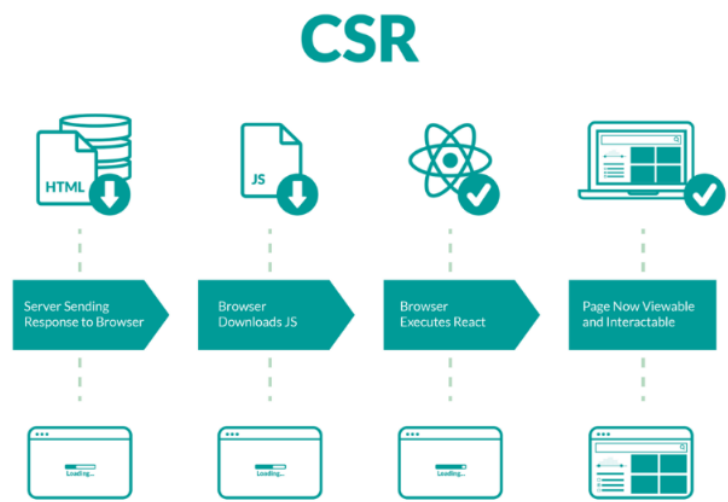
2.1.3 Server-Side Rendering (SSR)

When deploying an SSR, the server crafts HTML for each request, completing a fully formed page before dispatching it to the client. The main benefit of this model is Enhances SEO as content is visible in the initial HTML, resulting in improved performance for users with limited internet connections. However, with this comes the Interactions may decelerate since the entire page must reload for each request [4].

2.1.4 Client-Side Rendering (CSR)

This model ensures that the client, or browser, generates the webpage by interpreting JavaScript that retrieves and manipulates data. This Supplies a prompter user interface, ideal for applications with

frequent data updates. However, Initial loading times may be prolonged, and SEO challenges may arise as search engines might not index dynamically loaded content [5].



2.2 Purpose and Functionality

The purpose of the application is to provide a web application for a seed company. Here's an overview of the purpose and functionality of the application:

Seed catalogue:

- The application displays a catalogue of seeds with sample data like sunflower, rose, and tulip.
- Seeds are represented with properties such as name, quantity, description, and price.

Order Placement:

- Users can place orders for seeds through a form.
- When a user places an order, the order details are added to a list, and the corresponding seed's quantity is updated.

Shopping Cart:

- Users can add seeds to a shopping cart.
- The shopping cart keeps track of the selected seeds and their quantities.

Checkout:

- Users can proceed to checkout, which involves sending the selected seeds to the server (in this case, the /cart endpoint).
- The checkout process triggers further actions, such as displaying a confirmation modal or redirecting to a checkout page.

Seed Details:

- The application allows users to view details about a specific seed.
- Seed details are displayed in a modal when users click on a seed.

Seed Addition:

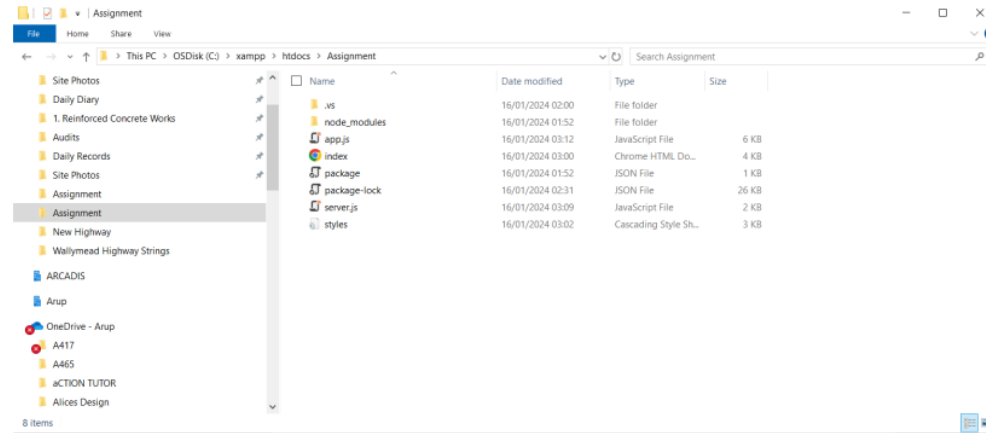
- Users can add new seeds to the catalogue through a form.
- Newly added seeds are sent to the server for processing and are displayed in the catalogue.

Server Interaction:

- The application communicates with a server (express server in your case) to perform operations such as retrieving seed data, adding new seeds, updating seed quantities, and handling the shopping cart.

2.3 Set up and Running of Application

First, a directory must be created holding the server side code (server.js) and client side (app.js, index.html & styles.css).

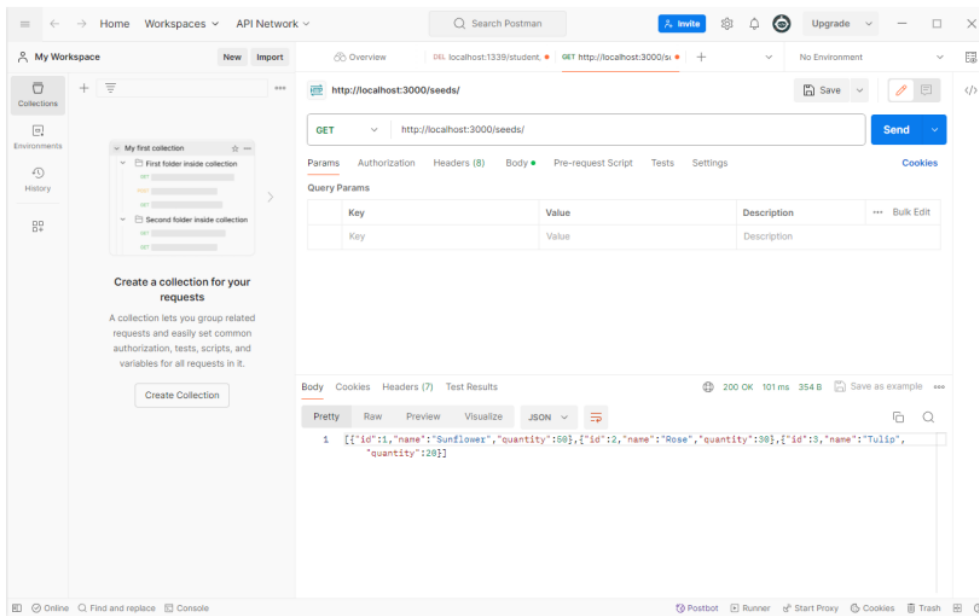


With the creation of the server.js file you can then run the server by accessing the command panel and typing node server.js.

```
Microsoft Windows [Version 10.0.19045.3803]
(c) Microsoft Corporation. All rights reserved.

C:\xampp\htdocs\Assignment>node server.js
```

You will then be given a local code of **http://localhost:3000**. You can now use applications such as postman to test the code against the GET, POST, DELETE & PATCH functions.



Within the directory bring up the command prompt and type:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3803]
(c) Microsoft Corporation. All rights reserved.

C:\xampp\htdocs\Assignment>node server.js
```

This will begin to run the server side of the code (server.js file). Next open another CMD prompt in the same directory and type:

```
http-server
Microsoft Windows [Version 10.0.19045.3803]
(c) Microsoft Corporation. All rights reserved.

C:\xampp\htdocs\Assignment>http-server
```

This will then start up a simple local HTTP location. Within the command prompt you will then receive the following messages:

```
C:\xampp\htdocs\Assignment>http-server
Starting up http-server, serving ./
http-server version: 14.1.1

http-server settings:
CORS: disabled
Cache: 3600 seconds
Connection Timeout: 120 seconds
Directory listings: visible
Autolinks: visible
Serve GZIP files: false
Serve Brotli files: false
Default File Extension: none

Available on:
http://172.27.17.121:8080
http://192.168.56.1:8080
http://192.168.0.33:8080
http://172.0.0.1:8080
```

You can choose from the Available ports such as <http://172.27.17.121:8080>. This will then take you to your webpage and will start running the application through this command window.

```
2024-01-16T02:46:13.001Z "GET /" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36"
(node:113088) [DEP0066] DeprecationWarning: OutgoingMessage.prototype._headers is deprecated
(Use node --trace-deprecation ... to show where the warning was created)
2024-01-16T02:46:39.072Z "GET /seedType-Sunflower&quantity=1" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36"
2024-01-16T02:47:26.295Z "GET /seedType-Rose&quantity=1" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36"
2024-01-16T02:47:42.279Z "GET /seedType-Tulip&quantity=3" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36"
2024-01-16T03:00:17.708Z "GET /seedType-Tulip&quantity=3" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36"
2024-01-16T03:02:41.853Z "GET /seedType-Tulip&quantity=3" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36"
2024-01-16T03:02:43.415Z "GET /seedType-Tulip&quantity=3" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36"
2024-01-16T03:04:17.893Z "GET /seedType-Sunflower&quantity=1" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36"
2024-01-16T03:12:48.144Z "GET /seedType-Sunflower&quantity=1" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36"
2024-01-16T03:12:48.158Z "GET /styles.css" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36"
2024-01-16T03:12:48.168Z "GET /app.js" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36"
```

3. Assignment 2 Overview

3.1 Introduction

In this part of the assignment a web API is to be created. This API should demonstrate certain REST methods including GET, POST and DELETE. A PATCH method has also been added. Within the API a staged checkout should be designed. This will allow the user to go through a check out experience such as, adding items to a cart then using ecommerce to purchase said items.

Information regarding what is needed when documenting software testing is to be investigated and evaluated. These will then be selected by which would most appropriately test the given web application. Finally, a series of tests will then be designed to test certain features within the application.

4. Assignment 2 Main Body

4.1 Evaluation of Documented Information on Software Testing

When conducting software tests, it's essential to systematically document the testing process, findings, and other relevant information. Clear and detailed documentation is crucial for effective communication between testing teams and other stakeholders. It provides a comprehensive overview of the testing process and facilitates better understanding and decision-making. Below is the information that should be recorded:

Test Case Details:

- Test case identifier and name.
- Description outlining the purpose of the test case.
- Step-by-step instructions for executing the test.
- Expected outcomes for the test case.
- Actual results observed during testing.

Test Environment:

- Specifications of the testing environment, encompassing hardware, software, browsers, and devices in use.
- Version information for the application being tested.
- Specific configurations or settings applied during the testing phase.

Test Data:

- Data employed for the test, including valid and invalid values.
- Particulars of data setup or configuration prerequisites.

Test Execution Date and Time:

- Record of the date and time when the test was carried out.

Test Execution Status:

- The overall status of each test case, denoted as pass, fail, or incomplete.
- Supplementary details elucidating the reasons behind failure or incompleteness.

Defects and Issues:

- Documentation of any identified defects or issues.
- Comprehensive descriptions of each problem, along with steps to replicate the issue.

Screenshots and Logs:

- Captured images depicting critical steps or encountered issues.
- Logs and error messages deemed relevant to the understanding of test outcomes.

Testers' Comments:

- Additional remarks or observations noted by the testing team.
- Feedback concerning the user interface, functionality, or overall user experience.

Browser/Device-Specific Information:

- Notation of any distinctions or problems specific to different browsers or devices.
- Inclusion of particulars regarding browser versions and device specifications.

Performance Metrics:

- Recording of performance-related measurements, such as response times, resource utilization, and scalability.

Security Testing Results:

- Documentation detailing any identified security vulnerabilities.
- Information regarding the severity assigned to each identified vulnerability.

Regression Testing:

- Results obtained from regression testing endeavours.
- Documentation indicating whether recent changes impacted existing functionalities.

User Feedback:

- Compilation of feedback received from end-users or stakeholders.
- Inclusion of comments addressing usability, functionality, or any expressed concerns.

Conclusion:

- A concise summary encapsulating the entirety of the testing initiative.
- An evaluation of the application's readiness for release.
- Outlining of recommended subsequent actions, such as retesting, additional testing, or resolution of identified defects [6].

4.2 Test Summary Report Example

Attached is a Test Summary Report containing the information needed when carrying out tests on the given application.

[illegible]

4.3 Testing of Applications Code

Test Title:	<i>Add Item to Shopping Cart</i>
Test Author:	<i>Josh Archer</i>
Test Description: Testing addToCart and updateCartUI	
<i>Tests the addToCart and updateCartUI functions to ensure they add seeds to the cart and update the UI accordingly. First Create a file client-tests/app.test.js for testing frontend logic. Within the File add // client-tests/app.test.js</i> <pre><i>import { render, fireEvent } from '@testing-library/dom';</i> <i>import '@testing-library/jest-dom';</i> <i>import { addToCart, updateCartUI } from '../app'; // Assuming your client-side logic is in app.js</i></pre> <i>describe('Client App Tests', () => {</i> <i> it('should add seeds to cart', () => {</i> <i> });</i> <i>You can then Mock any necessary elements to test the add to cart function. For example, Render a button and simulate a click.</i> <i>Finally, use Jest to execute client side test. To do so access command prompt and enter, npm run test-server.</i>	
Test Title:	<i>/seeds endpoint</i>
Test Author:	<i>Josh Archer</i>
Test Description: Tests the /seeds endpoint	
<i>Tests the /seeds endpoint to check if it returns a status of 200 and an array of seeds. This will verify individual functions or modules in the server logic and Ensure that functions behave as expected and handle edge cases properly. Firstly Create a file server-tests/server.test.js for testing server logic. Add code</i> <pre><i>const chai = require('chai');</i> <i>const supertest = require('supertest');</i> <i>const app = require('../server'); // Assuming your server file is named server.js</i></pre> <i>const { expect } = chai;</i> <i>const request = supertest(app);</i>	

```
describe('Server API Tests', () => {
  it('should retrieve seed data', async () => {
    const response = await request.get('/seeds');
    expect(response.status).toEqual(200);
    expect(response.body).toBe.an('array');
  });
});
```

Finally in Command Prompt enter `npm run test-server`.

Test Title:	End-to-End Tests for User Interactions
--------------------	--

Test Author:	Josh Archer
---------------------	-------------

Test Description: Simulates placing an order, updating the UI, and checks if the UI reflects the changes correctly.
--

The Purpose of this test is to Simulate user interactions and test the entire flow of actions on the client side. This will Ensure that frontend functions behave correctly and modify the UI as expected. Firstly Create a file `client-tests/e2e.test.js` for testing user interactions. Add the following code to said file: `import { render, fireEvent } from '@testing-library/dom';`

`import '@testing-library/jest-dom';`

`import '../app'; // Assuming your client-side logic is in app.js`

```
describe('Client App E2E Tests', () => {
  it('should place an order and update UI', () => {
  });
});
```

You can mock necessary elements to simulate user interactions and assert changes in the UI or server communication.

Finally Open command prompt and run: `npm run test-client`. This will execute the test code.

Test Title:	Integration Tests for API Endpoints
--------------------	-------------------------------------

Test Author:	Josh Archer
---------------------	-------------

Test Description: Tests the /seeds POST endpoint to verify that a new seed can be added, and the response matches the expected data.

The purpose of this test is to test the interaction of multiple components or modules, especially focusing on API endpoints. It will also Check if the API endpoints handle requests and responses correctly. Firstly create a file `server-tests/api.test.js` for testing API endpoints. Add the following code into the file:

```
const chai = require('chai');
```

```
const supertest = require('supertest');

const app = require('../server'); // Assuming your server file is named
server.js

const { expect } = chai;
const request = supertest(app);

describe('API Endpoint Tests', () => {
  it('should add a new seed', async () => {
    const newSeed = { name: 'NewSeed', quantity: 10 };
    const response = await request.post('/seeds').send(newSeed);
    expect(response.status).to.equal(200);
    expect(response.body).to.deep.include(newSeed);
  });
});
```

Finally Run in the command prompt : `npm run test-server`

4.4 Youtube Link

<https://www.youtube.com/watch?v=rpVq8aAkNvQ>

5. References

- [1] A. KAUR, "What is Single Page Application (SPA)? Pros and Cons with Examples," Net Solutions, 2021.
- [2] P. Sagar, "What Is A Single Page Application? Meaning, Pitfalls & Benefits," Excellent WebWorld, 2024.
- [3] R. Michaels, "The Content First Approach to Responsive Design," Stauffer, 2018.
- [4] S. Mhatre, "Advantages and disadvantages of server side rendering," DEV, 2023.
- [5] D. Taylor, "Client-Side Vs. Server-Side Rendering," Search Engine Journal, 2023.
- [6] Browser Stack, "How to write a good Test Summary Report?," Browser Stack, 15 06 2023. [Online]. Available: <https://www.browserstack.com/guide/how-to-write-test-summary-report>. [Accessed 04 01 2023].

2011907 Cloud Based Web Services

GRADEMARK REPORT

FINAL GRADE

50/100

GENERAL COMMENTS

Assignment 1:

=====

Rest & Ajax (45): 20

Requirements (15): 12

Design & Docs (30): 16

Demo (10): 6

Overall: **54%**

Assignment 2:

=====

Rest API (40): 18

Patch (10): 3

Test doc (30): 17

Testing (10): 3

Demo (10): 5

Overall: **46%**

Feedback:

=====

I was hoping for a little more requirements analysis and more thought to the design of the application.

There is AJAX provided it mainly focuses on POST operation.

The REST API is very brief and you used a list instead of a database.

Also I could not find the test code that was presented in the report, I guess it was intended to be implemented.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18
