

ACCA6024 Assignment 2 JE

by Jacob Evans

Submission date: 04-Jan-2026 10:24PM (UTC+0000)

Submission ID: 272040623

File name: 50439_Jacob_Evans_ACCA6024_Assignment_2_JE_708810_1064077129.docx (584.93K)

Word count: 1909

Character count: 9658



ACCA6024 - CLOUD BASED WEB SERVICES

Assignment 2

Design, develop, test and deploy a web application in a
cloud environment with use of AJAX methods.

Jacob Evans
2311160@student.uwtsd.ac.uk

Contents

Introduction	2
Webpage Changes.....	3
AJAX Methods	6
Issues and Solutions	8
ReadMe file created	9
Updated Tests	10
Conclusion	12
References	13

Introduction

This is a continuation of the first assignment, by designing, developing, testing and deploying a web application in a cloud environment.

The API has been updated for a more dynamic webpage, which allows the users to search and filter through the songs created and added to the webpage. A dedicated section of the webpage has been created which allows users to view the created songs and are able to click on them to view more details on that specific song.

This has been done by creating a backend RESTful API on the Node.js server, which includes the HTTP methods “GET”, “POST”, “PUT”, “DELETE” and “PATCH”, seen in Assignment 1 [1]. The user is also alerted if something they have done via these HTTP methods have worked or not worked. The dynamic webpage allows the user to know if their request has worked by looking at the frontend webpage.

The frontend is the dynamic webpage application (index.html), which can be used independently of the backend API. This dynamic webpage can be opened via the file found in the “music.api” file, or with Visual Studio Code’s “Live Server” extension. This is all done through AJAX FETCH requests. AJAX requests allow the webpage to be updated without having to reload the whole page [2], which makes the webpage more dynamic. A more dynamic webpage is more scalable, more user-friendly, and greater interactivity [3].

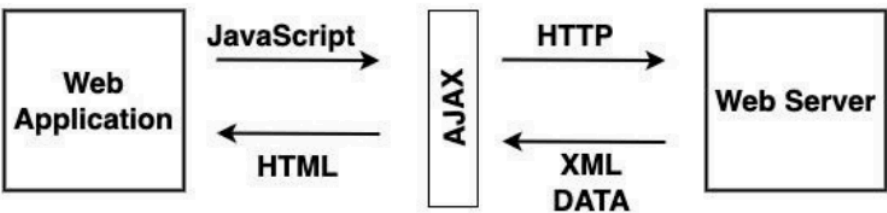


FIGURE 1 - FRONTEND TO BACKEND AJAX [4]

From Figure 1, we can see that an AJAX engine has been created between the web server and the web application.

AJAX methods aren’t the most reliable methods, however.

TABLE 1. SUMMARY OF RESULTS

Strategy	Dataset size	Avg. payload size	Time to interactive (TTI)	Recommended use case
1. Simple AJAX	100 records	15 KB	<100 ms	Small, simple datasets; static lists
1. Simple AJAX	10,000 records	1.5 MB	>5000 ms	Not recommended
2. Hybrid Datatable	Any (per page)	5-10 KB (per page)	200-300 ms	Large, paginated datasets; requires sorting/filtering

FIGURE 2 -AJAX TESTING [5]

From figure 2, we can see that AJAX is highly dependent on the dataset size and average payload of the data. From this figure we can conclude that a simple AJAX methods (like the ones used in this project) would actually cause lag and issues for the user.

Webpage Changes

Organisation: Jake's Music Library that stores songs (title, artist, rating, status).

Create / Edit Songs

Song ID (auto when you click a row) Title

Artist Rating (1-5)

[GET all songs](#) [GET by ID](#) [POST create](#) [PUT update](#) [PATCH rating](#) [Publish \(workflow\)](#) [DELETE](#) [Clear form](#)

Tip: Click a row in the table to view item details and auto-fill the form.

Song List

ID	Title	Artist	Rating	Status
1	POKERFACE	LADY GAGA	5	published
2	Pretend	Bad Omens	5	draft

Showing 2 song(s).

Item Details

ID: 1
Title: POKERFACE
Artist: LADY GAGA
Rating: 5
Status: published

Clicking a row shows item details without refreshing the page.

FIGURE 3 - UPDATED WEBPAGE

Figure 3 is the new updated Webpage, similar to the webpage in assignment 1. We can see that an "Item Details" has been added. When a song is clicked in the table "Song List", the details of that song are highlighted in the designated area of the webpage. The webpage does not need to refresh completely, as the user is able to click between the songs with little-to-no delay. This makes the webpage more dynamic.

```

function renderSongs(songs) {
  tbody.innerHTML = "";
  songs.forEach(song => {
    const tr = document.createElement("tr");
    tr.innerHTML = `
      <td>${song.id}</td>
      <td>${escapeHtml(song.title)}</td>
      <td>${escapeHtml(song.artist)}</td>
      <td>${song.rating}</td>
      <td>${song.status}</td>
    `;

    // Row shows details
    tr.addEventListener("click", () => showDetails(song));

    tbody.appendChild(tr);
  });

  countline.textContent = `Showing ${songs.length} song(s).`;
}

```

FIGURE 4 - TABLE CREATION

Figure 4 is the JavaScript used to create this table dynamically.

Music Library
Organisation: Jake's Music Library that stores songs (title, artist, rating, status).

Song not found

Create / Edit Songs
Song ID (auto when you click a row): Title:
Artist: Rating (1-5):
 GET all songs GET by ID POST create PUT update PATCH rating Publish (workflow) DELETE Clear form
 Tip: Click a row in the table to view item details and auto-fill the form.

Song List

ID	Title	Artist	Rating	Status
1	POKERFACE	LADY GAGA	5	published
2	Pretend	Bad Omens	5	draft

Showing 2 song(s).

Item Details
Select a song from the table to see detailed information here.

FIGURE 5 - SEARCHING A DELETED SONG

Figure 5 shows when the user attempts to search a song, an error message in red shows up on the top of the form. This notifies the user that there is an error, and that the song is missing/deleted.

The webpage feels more dynamic and fluid, as the HTTP methods are fluid and instantaneous; the user does not have to reload or wait for the page to be updated.

Organisation: Jake's Music Library that stores songs (title, artist, rating, status).

Invalid workflow transition: published → published

Create / Edit Songs

Song ID (auto when you click a row) Title

Artist Rating (1-5)

Tip: Click a row in the table to view item details and auto-fill the form.

Song List

ID	Title	Artist	Rating	Status
1	Lady GAGA	Bad Omens	5	published

Showing 1 song(s).

Item Details

ID: 1
Title: Lady GAGA
Artist: Bad Omens
Rating: 5

FIGURE 6 - WORKFLOW CREATION

We can see in Figure 6 that the “Publish” button has been implemented to change the status of a song from “draft” to “publish”. This is implemented so that there is control over the songs created. This API could be used to target only published songs. The “published” songs would already be approved, and the draft songs “awaiting approval”, which would be useful for a music library.

We can also see the error message for “invalid workflow transition”. This is shown when the user attempts to publish an already published song. This make sure that invalid actions are prevented.



FIGURE 7 - WORKFLOW CONCEPT

Figure 7 shows this project’s simple Workflow Concept, showing only two states (Draft, Published), and the transition between the two (Publish).

AJAX Methods

The webpage feeling more dynamic and fluid is due to the AJAX method used in the “script.js” file. As mentioned in the introduction that AJAX allows the frontend and backend to communicate asynchronously. Declaring the function as “async” allows for it to be asynchronous as seen below in figure 8.

```
// Load songs when page loads
async function getAllSongs() {
  clearMessage();
  const res = await fetch(`${API_BASE}/songs`);
  const data = await res.json();
  renderSongs(data);
}
```

FIGURE 8 - AJAX FETCH/AWAIT

In figure 8, we can see the use of an AJAX method used to “getAllSongs”. This AJAX method allows for the webpage to update from the “FETCH” information from the backend API. This means the page does reload, but the data is “exchanged”.

The “await” is used so that AJAX waits for a response, which stops the JavaScript from continuing until there is an API response.

```
// Load songs when page loads
setInterval(() => {getAllSongs();
}, 5000);
```

FIGURE 9 - SET INTERVALS

In figure 9, the “setInterval” AJAX method was used. This refreshes the “getAllSongs” AJAX request every 5 seconds, which allows the user to update the webpage by using the backend API rather than the webpage itself. This is a dynamic change that updates the webpage. The interval decided allows for the webpage to be updated frequently enough not to be noticeable yet does not cause a server load issue. As there is only one user, the server load could be set even lower, as the number of requests will be increased. If this API was publicly available, then we would have to take consideration for the number of requests per hour multiplied by the number of users, which can cause issues due to increased traffic.

A test was conducted using the test.http file to “POST” a song via the backend API. Without the interval, the user would have to reload the page themselves as the webpage has no way of knowing that they are expecting more information. The interval refresh allows the webpage to be updated without the user having to manual reload the page. This is known as POLLING [5].

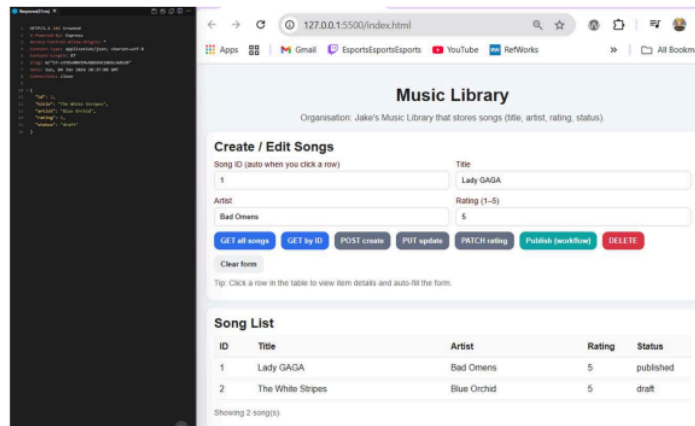


FIGURE 10 - POST THROUGH TEST.HTTP

Figure 10 attempts to showcase this refresh, as we can see that a new song has been POSTed through the test.http, and has been updated automatically on the webpage.

Issues and Solutions

The first “issue” encountered was when running the UI and API, localhost:3000 states the figure below.

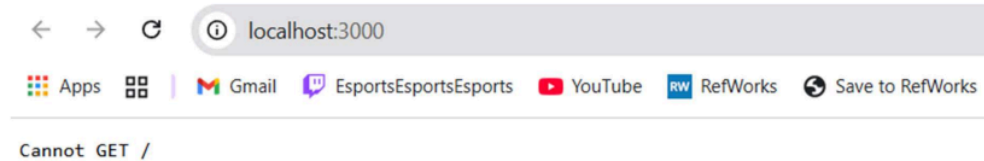


FIGURE 11 - CANNOT GET /

It took a while to realise that the information on this page was the information provided by the backend. This was opened initially, before any songs were created, therefore, obviously, there was no information for the API to GET. Once a song was added, we can see the information was added in this format seen below in figure 12. This demonstrates the separation between the API and UI.

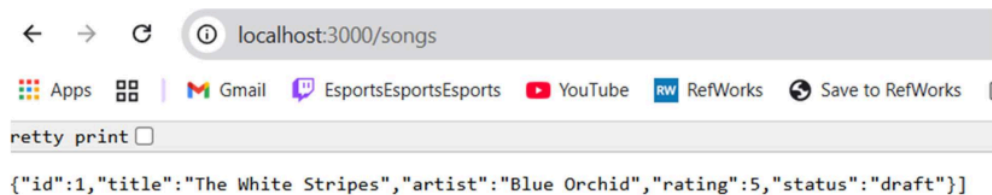


FIGURE 12 - UPDATED API

Another issue found was the AJAX Polling. The page now updates automatically every 5 seconds, which may not be enough time for the user to notice that their updated song has an error. While “Publishing” the songs, the error does come up, but due to the webpage updating every 5 seconds, future projects may need to be conscious of this and apply a more suited interval.

When searching/filtering, having the ID be the search method seems terrible, as a user would have to remember an ID to search for a specific song. If the project were to be done again, other search methods would be implemented, via searching for a specific artist, or which songs have/haven't been published. This would allow the user to filter through artists that they like and also see which songs have been “approved” via the workflow process.

ReadMe file created

```
1  README.md > ## # AUTHOR
2  # RESTful and AJAX Music Library API
3  ## API Endpoints
4  |METHOD| ENDPOINT | DESTINATION |
5  |-----|-----|-----|
6  |GET| /songs | Retrieve All Songs |
7  |GET| /songs/:id | Retrieve specific song from ID |
8  |POST| /songs | Create's a song |
9  |PUT| /songs/:id | Fully Update specific song |
10 |PATCH| /songs/:id | Partially update song |
11 |DELETE| /songs/:id | Delete specific song |
12 |POST| /songs/:id/publish | Publish a song (workflow) |
13
14 ## Workflow Process
15 Each song has only two states;
16 -draft - new song, no processing
17 -published - song has been processed/approved
18
19 ## Running Project
20
21 ### 1. Install Dependencies
22 Open terminal in project folder and run:
23 bash npm install
24
25 ### 2. Start API Server
26 bash npm start
27
28 The terminal should state "Music API running on http://localhost:3000"
29
30 ### 3. Opening the Webpage
31
32 #### Option 1. Live Server
33 If this is opened in Visual Studio Code, the Live Server extension must be installed. Finding the extensions tab located
34 on the left, we can search "Live Server", which can be installed.
35
36 Once installed, index.html can be right-clicked and opened with Live Server.
37
38 #### Option 2. Music API file
39 Locate the downloaded Music API file. One file should have the name "index.html". Open this with your browser of choice
40 (tests were used through Google Chrome)
41
42 ## Testing
43
44 Testing can be carried out either via the webpage itself, or via the test.http file. Users with Vissual Studio Code REST
45 Client Extension can press the CRUD operations to test through the backend API.
46
47 Test cases include:
48 -CRUD Operations (GET, POST, PUT, DELETE)
49 -Workflow Validation (Publish)
50 -AJAX error handling
51 -UI updates without manual page reload (setIntervals)
52
53 # Notes
54
55 -If user opens UI via http://localhost:3000, "Cannot GET /" will be shown, indicating no information has been created.
56 -UI and API run on seperate ports, indicating client-server separation
57
58 # AUTHOR
59 Jacob Evans
60 UMTSD Swansea
61 Applied Computing Student
62 2311160
```

FIGURE 13 - README FILE

 README	04/01/2026 21:20	Markdown Source ...	3 KB
--	------------------	---------------------	------

FIGURE 14 - README FILE 2

As seen in figure 13 and 14, a ReadMe file has been added for documentation and allows other users to use this API.

Updated Tests

Test Title:	<i>Launching API</i>
Test Author:	<i>J. Evans</i>
Test Description:	<i>API is launched, and a https://localhost is created for the music API.</i>
Expected Outcome:	<i>A string of information added through the webpage when localhost:3000 is opened. If no information has been added, Cannot GET /. This is the backend API.</i>
	<i>PASS – Figure 10 and Figure 11</i>
	<i>Outcome – is correct, initially confused</i>
Test Title:	<i>Launching Webpage</i>
Test Author:	<i>J. Evans</i>
Test Description:	<i>Webpage is launched using Live Server extension or opening "index.html" in the music.api file.</i>
Expected Outcome:	<i>Webpage should be displayed neatly, with all available HTTP methods displayed.</i>
	<i>PASS – Figure 2</i>
	<i>Outcome – as intended</i>

Test Title:	<i>New song created</i>
Test Author:	<i>J. Evans</i>
Test Description:	<i>When new song is created via the webpage without reloading page.</i>
Expected Outcome:	<i>When a song is POST through the webpage, song is added to designated area below song creation, with no error message and no page reload.</i>
	<i>PASS – Figure 9</i>
	<i>Outcome – as intended</i>

Test Title:	<i>GET All Songs</i>
Test Author:	<i>J. Evans</i>
Test Description:	<i>All songs created are noted down in table at designated area of webpage.</i>
Expected Outcome:	<i>All songs created should display in a table in ID order.</i>
	<i>PASS – Figure 2</i>
	<i>Outcome – as intended</i>

Test Title:	<i>Get Song from Specific ID or Clicking on Song</i>
Test Author:	<i>J. Evans</i>
Test Description:	<i>Ensures that the specific song ID is retrieved from search or clicking the song in the table.</i>
Expected Outcome:	<i>Table created via creation of songs can be searched/filtered via the song ID specifically (GET by ID button on webpage), or user can click on the song in the table, which gives the user more detailed view of the song selected from the table.</i> <i>PASS – Figure 5</i> <i>Outcome – as intended, although different search/filter method would be more desirable (search via Artist name or published song)</i>
Test Title:	<i>AJAX Set Interval</i>
Test Author:	<i>J. Evans</i>
Test Description:	<i>Application of Webpage reload through AJAX Interval</i>
Expected Outcome:	<i>Webpage should reload every 5 seconds. This allows for users to POST created songs through the backend API, and the webpage to update without the user having to reload manually.</i> <i>PASS – Figure 9</i> <i>Outcome – as intended</i>
Test Title:	<i>Invalid Re-Publish - Alerts</i>
Test Author:	<i>J. Evans</i>
Test Description:	<i>Attempt to publish already published song</i>
Expected Outcome:	<i>Error message should appear alerting user that this is not possible.</i> <i>PASS – Figure 5</i> <i>Outcome – as intended</i>
Test Title:	<i>Delete Song</i>
Test Author:	<i>J. Evans</i>
Test Description:	<i>Song is deleted through specific ID or through Table Selected song</i>
Expected Outcome:	<i>Song removed on webpage table. Song cannot be found through ID search button. Error message displayed “Song Not Found”.</i> <i>PASS – Figure 4</i> <i>Outcome – as intended</i>

Conclusion

Both assignments showcase how important and useful CRUD operations and AJAX methods are when used in cloud-based Services such as websites/webpages. This project showcases the Node.js Express backend well, as it supports the CRUD operations. The backend API communicates well with the UI, created via HTML, CSS and JavaScript, and even allows for AJAX methods to be used to make the page more dynamic, which makes the webpage far more appealing to use and more user-friendly to use. Basic testing was used to make sure the webpage worked as intended, with no real issues. However, there are few issues mentioned where if this type of project were to be done again, that would need to be addressed to allow for the webpage to work better than previously.

Overall, module has showcased the importance of cloud-based service methods.

References

- [1] "HTTP request methods", Mozilla Developer, <https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods>
- [2] "Advantages and Disadvantages of Dynamic Websites", Verbat Blog, <https://www.verbat.com/blog/advantages-and-disadvantages-of-dynamic-websites/>
- [3] "What is AJAX?", tutorialspoint, https://www.tutorialspoint.com/ajax/what_is_ajax.htm
- [4] "View of Multi-strategy AJAX and event-driven state management for responsive web applications," Accessed: Jan 4, 2026. Available at: <https://itvisnyk.kpi.ua/article/view/341787/335511>.
- [5] "What is Polling in AJAX", GeeksforGeeks.org, <https://www.geeksforgeeks.org/javascript/what-is-polling-in-ajax/>

ACCA6024 Assignment 2 JE

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

86/100

	Marks available	Marks awarded
Generic Assessment Criteria		
1. Engagement with Literature Skills		
2. Knowledge and Understanding Skills		
3. Cognitive and Intellectual Skills	Assignment 2: 40	
Design and documentation		3
Test documentation and testing		6
Well presented report of implementation.		

Comprehensive documentation and instructions given through a readme file. Reasonably good understanding demonstrated of the functionality of the SPA, middleware and API. UI testing with evidence is good. Some good use of screenshots and code snippets. A good honest account of issues shows development through the project and understanding.		
4. Practical Application Skills	Assignment 2: 50	
AJAX implementation REST API Nice organised clean code. Really good comprehensive readme.md file. Fully working UI.		4 5
5. Transferable Skills for Life and Professional Practice	Assignment 2: 10	
Presentation Good demonstration of UI and knowledge of workings.		5

Better knowledge of code, especially what the functions do would have been desirable.		
--	--	--

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
