

Cloud Based Web Service - Mathew Jones

by Mathew Jones

Submission date: 05-Jan-2026 06:36AM (UTC+0000)

Submission ID: 272074867

File name: 50104_Mathew_Jones_Cloud_Based_Web_Service_-_Mathew_Jones_708810_1747478968.docx
(178K)

Word count: 1952

Character count: 11204



Assignment 2: Cloud Based Web Service

Cloud Based Web Services

Mathew Jones (2317425)

COMPUTING (SOFTWARE ENGINEERING)



Contents

1. Introduction.....	2
2. Application overview	2
2.1 Organisation context	2
2.2 Core features	2
3. System Architecture	3
3.1 Client-Server Architecture	3
3.2 Frontend Architecture	3
3.3 Backend Architecture	3
4. REST API Design and Cognitive Understanding	3
4.1 REST Principles	3
4.2 REST Resources	4
4.3 HTTP Methods.....	4
4.4 Stateless Communication and JSON	4
5. Middleware and Server-Side Processing	4
6. AJAX and Asynchronous Communication.....	5
6.1 Use of AJAX.....	5
6.2 Polling implementation.....	5
7. Application Setup and Usage Instructions.....	5
7.1 Prerequisites	5
7.2 Running the Backend.....	6
7.3 Running the Frontend	6
7.4 Usage of the Application	7
8. Evaluation and Design Decisions	7
9. Software Testing.....	8
10. Conclusion	9
11. References.....	10

1. Introduction

This assignment involves the design and implementation of a single-page web application for an organisation called Penlan Game Store. The purpose of the application is to manage and display video games that are available within the business using RESTful web API and a client-server where the two interact with each other in real time.

The application demonstrates the use of REST principles, middleware, and AJAX techniques to create a responsive and interactive user experience without requiring full page reloads. Individuals using the site are able to view, search, filter, create, edit and delete game records, with updates reflected instantly in the interface.

A Node.js and Express-based backend provides a REST API that exposes game data as resources, while the frontend uses JavaScript's Fetch API to interact with the server. Middleware is used to handle JSON parsing and cross-origin requests, enabling separation between the client and server.

Additionally, the application implements polling which will periodically check for data updates, ensuring information remains current without having to manual refresh. This report documents the design decisions, REST API structure, middleware usage, and overall functionality of the application, alongside instructions for setup and execution.

2. Application overview

2.1 Organisation context

Penlan Game Store is a fictional retail organisation that specialises in the sale of video games across multiple platforms such as PC, PlayStation, Xbox and Nintendo Switch. The web application is designed to function as an inventory management system that allows staff to manage product listing efficiently. It provides a clean interface for browsing available games and performing tasks such as adding new games, editing existing records, and removing games that are no longer needed.

2.2 Core features

The main features of the application include:

- Displaying a list of games retrieved from a REST API
- Ability to view detailed information for a selected game
- Creating new game records
- Editing existing game records
- Deleting game records
- Searching games by title or description
- Filtering games by platform, genre, and stock availability
- Sorting games by title or price

- Automatically updating displayed data using polling

These features demonstrate the use of RESTful communication, middleware processing, and AJAX-based interaction.

3. System Architecture

3.1 Client-Server Architecture

The application follows a client-server architecture. The frontend and backend are separate to each other and communicate through HTTP requests. The frontend is responsible for rendering the user interface and handling any of the users' interactions, while the backend is responsible for data storage, processing, and validation. This separation allows the backend API to be reused or extended independently of the frontend. This allows for future scalability and easier maintainability.

3.2 Frontend Architecture

The frontend is implemented as a single-page application. All interactions occur within a single HTML page, and JavaScript automatically updates the content based on server responses. Bootstrap is used to provide a standard UI for components such as tables, buttons and modals.

User actions such as searching, filtering, and submitting forms trigger JavaScript event handlers that send AJAX requests to the REST API.

3.3 Backend Architecture

The backend is implemented using Node.js and it allows clients to interact with inputted data. Data is stored in a JSON file, which is read and written by the server when required.

4. REST API Design and Cognitive Understanding

4.1 REST Principles

REST or Representational State Transfer is an architecture style which is used for designing networked applications. They allow applications to access resources (Data) inside of another application, service or database. Resources can be identified by URLs and are manipulated using HTTP methods. These HTTP methods correspond with CRUD (Create, Read, Update, Delete) which is critical for REST. REST APIs are highly compatible with other applications, flexible and have improved security due to usage of tokens for authentication but you are required to be connected to the web to execute operations, and performance can be negatively affected by poor internet speeds.

REST APIs also promote a clear separation between the client and server by enforcing a uniform interface. This allows frontend and backend systems to evolve independently, as long as the API contract remains consistent. In this application, the frontend does not need to know how data is stored or processed internally by the server; it only interacts with exposed endpoints using standard HTTP requests. This separation improves maintainability and supports future enhancements without requiring major architectural changes. In this application, each game is treated as a resource, and the API provides a consistent interface for interacting with these resources.

4.2 REST Resources

The primary REST resource in the application is the game.

- `/api/games/` – represents the collection of all game resources
- `/api/games/{id}` – represents an individual game resource

4.3 HTTP Methods

There are different HTTP methods used such as [1]:

- GET - To retrieve all games or a specific game
- POST – To create a new game
- PUT – To update an existing game
- DELETE – To delete a new game

4.4 Stateless Communication and JSON

Communication between the client and the server is done using JSON as the data format. The reason for this is JSON is lightweight, easy to parse in JavaScript, and is widely supported. Each request is handled independently, and the server does not store session data between requests ensuring statelessness

5. Middleware and Server-Side Processing

Middleware functions are pieces of code that run before a request reaches the main route handler on the server. They perform common tasks that otherwise would need to be repeated in every route [2].

In this application, two main middleware functions were used. The `express.json()` middleware automatically reads and converts incoming JSON data into a JavaScript object, this makes it easier for the server to access data sent from the frontend. The `cors()` middleware allows the frontend application to communicate with the backend API, without this they would not be able to communicate as they use different ports. The frontend runs on port 8080 and the API on port 3000, so without CORS the browser would block the request.

The usage of middleware reduces code duplications and keeps route handlers simple and focused on their main purpose. Frontend and backends are able to work independently thus improving the overall structure and maintainability of the application.

6. AJAX and Asynchronous Communication

6.1 Use of AJAX

AJAX (Asynchronous JavaScript and XML) enables the application to communicate with the server without the requirement of referencing the webpage. In this project, AJAX was implemented using the Fetch API [3].

Examples of AJAX usage include:

- Loading the initial list of games
- Loading detailed game data in the modal
- Submitting create, update, and delete requests

6.2 Polling implementation

The application implements polling to periodically check for updates on the server. Every few seconds, the frontend sends a GET request to `/api/meta`. If the returned timestamp differs from the last known update, the application reloads the game list automatically.

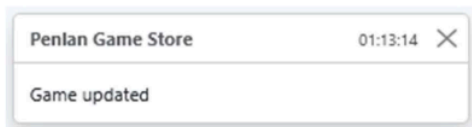


Fig. 1. Update automatically working.

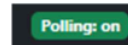


Fig. 2. Image showing Polling is working.

7. Application Setup and Usage Instructions

7.1 Prerequisites

- Visual Studio Code was used to type up code
- Node.js (LTS version) was required to be downloaded [4]
- Web browser to load the webpage
- PowerShell

7.2 Running the Backend

1. Navigate to the server directory:

```
PS C:\WINDOWS\system32> cd c:\Users\mathe\game-store-app\server
```

Fig. 3. Changing server directory.

2. Install dependencies:

```
PS C:\Users\mathe\game-store-app\server> npm install
removed 1 package, and audited 71 packages in 1s
15 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
```

Fig. 4. Installing dependencies.

3. Start Server:

```
PS C:\Users\mathe\game-store-app\server> npm start
> start
> node server.js
API running on http://localhost:3000
```

Fig. 5. Starting server.

7.3 Running the Frontend

1. Navigate to the server directory:

```
PS C:\Users\mathe\game-store-app\server> cd c:\Users\mathe\game-store-app\client
```

Fig. 6. Changing client directory.

2. Start a local web server:

```
PS C:\Users\mathe\game-store-app\client> python -m http.server 8080
Serving HTTP on :: port 8080 (http://[::]:8080/) ...
::ffff:127.0.0.1 - - [04/Jan/2026 17:42:33] "GET / HTTP/1.1" 200 -
::ffff:127.0.0.1 - - [04/Jan/2026 17:42:33] "GET /styles.css HTTP/1.1" 200 -
::ffff:127.0.0.1 - - [04/Jan/2026 17:42:33] "GET /app.js HTTP/1.1" 200 -
::ffff:127.0.0.1 - - [04/Jan/2026 17:42:33] code 404, message File not found
::ffff:127.0.0.1 - - [04/Jan/2026 17:42:33] "GET /favicon.ico HTTP/1.1" 404 -
::ffff:127.0.0.1 - - [04/Jan/2026 17:55:43] "GET / HTTP/1.1" 200 -
::ffff:127.0.0.1 - - [04/Jan/2026 17:55:43] "GET /app.js HTTP/1.1" 304 -
::ffff:127.0.0.1 - - [04/Jan/2026 17:55:43] "GET /styles.css HTTP/1.1" 200 -
::ffff:127.0.0.1 - - [04/Jan/2026 17:57:28] "GET / HTTP/1.1" 304 -
::ffff:127.0.0.1 - - [04/Jan/2026 18:02:21] "GET / HTTP/1.1" 200 -
```

Fig. 7. Starting local web server.

3. Open a browser and navigate to <http://localhost:8080>

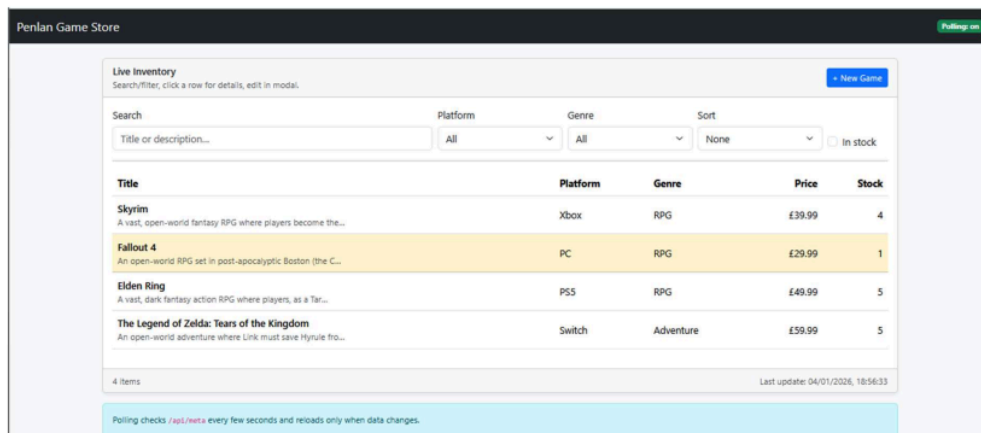


Fig. 8. Penlan Game Store.

7.4 Usage of the Application

- Use the search bar and filters to locate games
- Click a table row to view and edit details
- Use the "New Game" button to add entries
- Delete entries using the delete button

8. Evaluation and Design Decisions

Several design decisions were made to balance simplicity and functionality. Polling was chosen over web Sockets to reduce complexity while still demonstrating continuous server communication [5]. Bootstrap was used to ensure consistent layout, and usability with minimal custom CSS.

9. Software Testing

Test Title:	Load inventory list (GET)
Test Author:	Mathew Jones
Test Description:	Open http://localhost:8080 while the API is running. Observe that the table loads automatically without refreshing the page.
Expected Outcome:	The game table populates with items from the API. No page refresh is required.

Test Title:	View item details (GET)
Test Author:	Mathew Jones
Test Description:	Click on a game row in the table to open the details modal/panel.
Expected Outcome:	Details for the selected game load and display correctly (title, platform, genre, price, stock, description).

Test Title:	Create new game (POST)
Test Author:	Mathew Jones
Test Description:	Click "New Game", enter valid details, and save.
Expected Outcome:	A new game is created, a success message shows, the modal closes, and the table updates with the new item.

Test Title:	Update existing game (PUT)
Test Author:	Mathew Jones
Test Description:	Open an existing game, change price/stock/description, and save.
Expected Outcome:	The game updates successfully and the table reflects changes without page reload.

Test Title:	Delete a game (DELETE)
Test Author:	Mathew Jones
Test Description:	Open a game and click Delete, confirm deletion.
Expected Outcome:	The game is removed from the inventory and no longer appears in the table.

Test Title:	Search by title/description (GET)
Test Author:	Mathew Jones
Test Description:	Type a keyword into the Search input.

Expected Outcome:
Should show the game you inputted.

Test Title:	Filter by platform/genre (GET)
Test Author:	Mathew Jones
Test Description:	Select a platform/genre from the dropdown menu.
Expected Outcome:	Only games with the selected platform/genre should appear.

Test Title:	Sorting by title (GET)
Test Author:	Mathew Jones
Test Description:	Choose "Title A→Z" from the Sort dropdown.
Expected Outcome:	Games display in alphabetical order by title.

Test Title:	Polling updates inventory automatically (/api/meta polling)
Test Author:	Mathew Jones
Test Description:	Open the app in two windows. In Window A, create/edit/delete a game. Do nothing in Window B.
Expected Outcome:	Within the polling interval (~6 seconds), Window B updates automatically and shows "Live update received".

Test Title:	Data persists after restart (data.json storage)
Test Author:	Mathew Jones
Test Description:	Create a new game, stop the server, restart it, reload the app.
Expected Outcome:	The created game still exists and appears in the table (persisted in data.json).

10. Conclusion

This project successfully demonstrates the development of a RESTful web application using AJAX and middleware. The Penlan Game Store application implements a clean client-server architecture, follows REST principles, and uses middleware to streamline server-side processing. Through the use of asynchronous communication and polling, the application provides a dynamic and responsive user experience.

11. References

- [1] W3Schools Team. "HTTP Methods GET vs POST." W3Schools. Accessed: Jan 3 2026. [Online]. Available: https://www.w3schools.com/tags/ref_httpmethods.asp
- [2] AWS Team. "What is Middleware?" AWS Amazon. Accessed: 03 Jan 2026. [Online]. Available: <https://aws.amazon.com/what-is/middleware/>
- [3] W3Schools Team. "AJAX Introduction." W3Schools. Accessed: Jan 3 2026. [Online]. Available: https://www.w3schools.com/xml/ajax_intro.asp
- [4] Nodejs Team. "Node.js — Download Node.js®." Accessed: Jan 4 2026. [Online]. Available: <https://nodejs.org/en/download>
- [5] T. Schmidt. "Polling or WebSockets: Choosing with Amazon API Gateway." AWS Fundamentals. Accessed: Jan 5 2026. [Online]. Available: <https://awsfundamentals.com/blog/polling-vs-websockets-with-amazon-api-gateway>

Cloud Based Web Service - Mathew Jones

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

35/100

Generic Assessment Criteria	Marks available	Marks awarded
1. Engagement with Literature Skills		
2. Knowledge and Understanding Skills		
3. Cognitive and Intellectual Skills	Assignment 2: 40	
Design and documentation		35
Test documentation and testing		
Well presented report of implementation.		

Comprehensive documentation and instructions given. Reasonable understanding demonstrated of the functionality of the SPA, middleware and API. UI testing with evidence is good, maybe adding a PASS or FAIL box would have been beneficial. Some reasonable use of screenshots and code snippets.		
4. Practical Application Skills	Assignment 2: 50	
AJAX implementation REST API As in assignment 1, a word .docx file was provided, not the code requested by the brief.		0
5. Transferable Skills for Life and Professional Practice	Assignment 2: 10	
Presentation No demonstration given.		0

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11
