

Cloud-Based Web Services Report 2 - WR

by William Rowlands

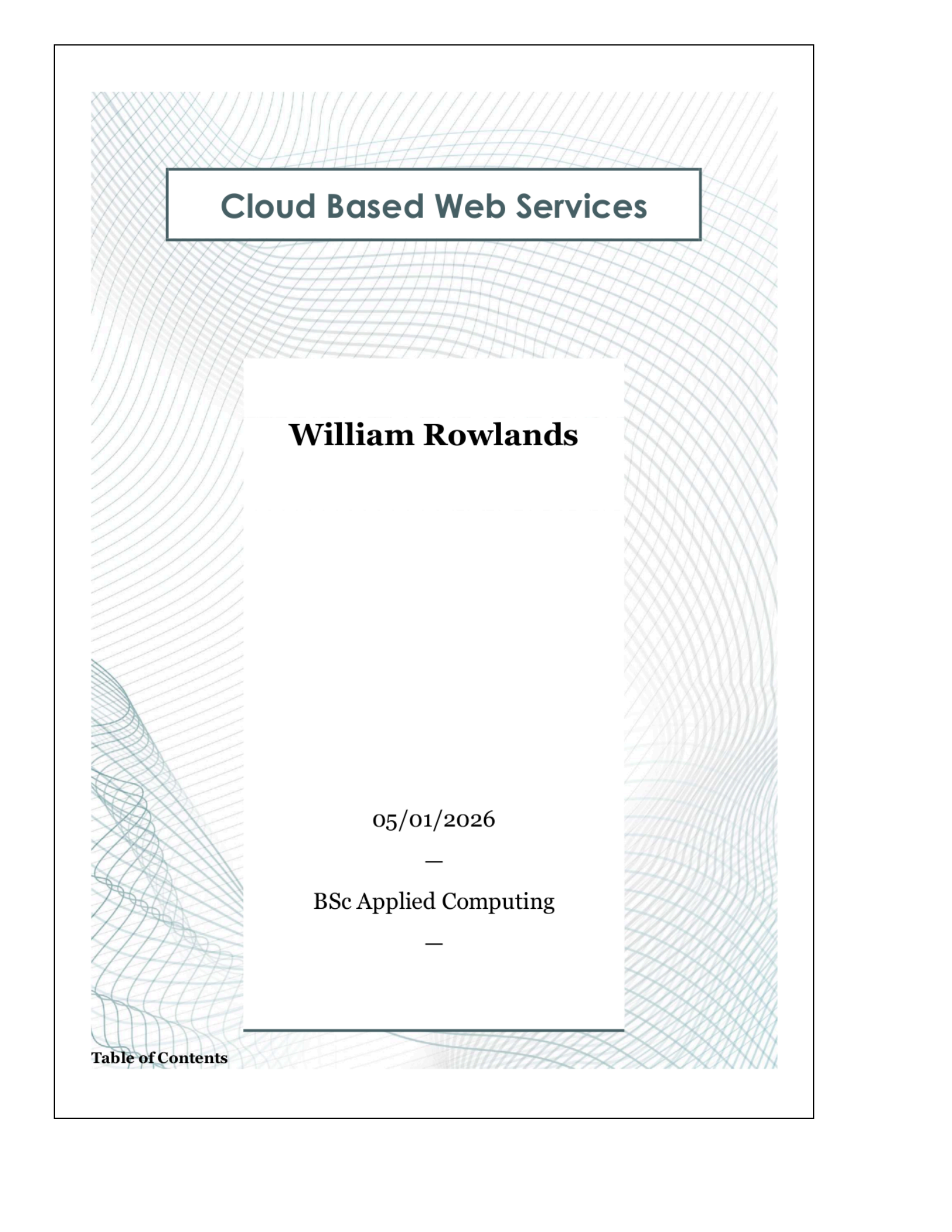
Submission date: 05-Jan-2026 11:50AM (UTC+0000)

Submission ID: 272110584

File name: 65957_William_Rowlands_Cloud-Based_Web_Services_Report_2_-_WR_708810_1022480047.docx
(3.38M)

Word count: 2624

Character count: 16645



Cloud Based Web Services

William Rowlands

05/01/2026

—

BSc Applied Computing

—

Table of Contents

Table of Contents

<u>1. Introduction</u>	<u>3</u>
<u>1.2 System Overview.....</u>	<u>3</u>
<u>1.3 User Interface Design</u>	<u>3</u>
<u>1.4 RESTful API Integration</u>	<u>4</u>
<u>1.5 Appointment Creation and Management</u>	<u>4</u>
<u>1.6 Searching and Filtering.....</u>	<u>5</u>
<u>1.7 Polling and Real-Time Updates</u>	<u>5</u>
<u>1.8 Docker Integration and Deployment.....</u>	<u>5</u>
<u>2. Coding Challenges and Error Resolution</u>	<u>6</u>
<u>3. System Evaluation and Reflection</u>	<u>6</u>
<u>4. Conclusion</u>	<u>6</u>

1. Introduction

The ability to integrate dynamic web applications with RESTful APIs has become crucial in modern web development. The creation of a barbershop appointment management system with an interactive and responsive user interface that enables users to make, view, edit, and remove appointments is the main goal of this project. The system shows how AJAX may be used practically for client-side filtering, searching, data polling, asynchronous communication, and RESTful API design. This project's main objective was to create a single-page application that effectively communicates with a backend API, mirroring the increasing trend in contemporary online applications where real-time data manipulation is essential.

The appointment system was selected as a typical illustration of a practical, small-scale application. Like many service-oriented businesses, barbershops need efficient scheduling solutions to handle customer reservations and maximise resources. Although there are a number of commercial options, creating a custom system enables a thorough investigation of fundamental web development concepts, such as user experience (UX) design, asynchronous data handling, dynamic DOM manipulation, and API interaction. This project focusses on expanding this functionality to a fully interactive front-end, building on the first assignment's implementation of the backend API using Node.js and Express.

The design decisions, implementation difficulties, and debugging procedures used during the system's development are all included in the study. It offers a thorough description of the code structure, important functions like search, filtering, and polling, as well as insights on errors made and fixes used. In order to promote reproducibility and ease of setup, it also investigates the deployment of the program using Docker. This story will serve as the foundation for the final evaluation, showcasing technical proficiency, problem-solving abilities, and a thoughtful comprehension of contemporary web development techniques.

1.2 System Overview

A front-end user interface and a RESTful backend API make up the single-page web application that is the barbershop appointment system. Booking appointments, accessing current appointments, marking appointments as finished, and removing appointments are just a few of the actions that users may accomplish with the application. While Node.js and Express are used to build the backend API, HTML, CSS, and JavaScript are used to develop the front-end. An in-memory data store, which can be expanded to a database for production systems, is used to handle data persistence in a lightweight manner.

To enable asynchronous communication between the front-end and backend, AJAX is widely used. This improves the user experience by enabling dynamic changes to the webpage without necessitating a complete reload. Appointments can be filtered by customer name, barber name, and status (e.g., booked or finished). JavaScript event listeners and array manipulation techniques are used to build the filtering and searching features, enabling real-time updates of the shown data.

The polling mechanism, which periodically checks the backend to guarantee that the shown appointments remain current, is a noteworthy aspect of the system. This is especially helpful in situations when several users may be dealing with the system at once, such when several barbers are handling conflicting schedules. Without the need for manual site refreshes, polling guarantees that viewers are shown the most recent information.

1.3 User Interface Design

Usability was given precedence over aesthetic sophistication in the design of the user interface, which was intended to be simple and practical. A form for making new appointments and a constantly updated section with the list of existing appointments are displayed on the home page. Customer

name, appointment date, appointment time, and barber option are all fields on the appointment creation form. Time slots, which correspond to regular work hours, are mechanically created in 15-minute intervals between 9:00 and 17:00.

A number of UI issues arose during development. At first, when an appointment was made, the time slot option did not update properly, resulting in duplicate or absent appointments. In order to ensure that users may only choose legitimate, available times, this was fixed by renewing the time slots upon form submission. In order to enable various filters (status, barber, and name) to function concurrently, the filtering logic also needed to be carefully considered. Separate functions were used for each criterion in early filtering attempts, which resulted in disputes and insufficient outcomes. The problem was fixed by combining the filtering into a single function that applies all active criteria one after the other. As a result, the UI now appropriately displays the combined filter selections. Displaying comprehensive appointment information presented another UX difficulty. Each appointment's HTML elements, including buttons to remove or mark appointments as finished, were dynamically generated as part of the implemented solution. By utilising DOM modification for dynamic updates and keeping an intuitive interface, our method made sure that users could directly interact with individual appointments.

1.4 RESTful API Integration

Endpoints for retrieving, creating, updating, and deleting appointments are provided by the RESTful API created in the initial assignment. The endpoints employ standard HTTP methods: DELETE is used to remove appointments, PATCH is used to update appointment status, POST is used to make new appointments, and GET is used to obtain appointment data. The Fetch API is used by the front-end to communicate with these endpoints, enabling asynchronous queries and answers.

Error handling had to be carefully considered when implementing the API integration. A number of problems were found during early testing, including efforts to alter or remove appointments that didn't exist, which led to HTTP failures. In order to fix this, the Fetch requests were enclosed in try-catch blocks, and in the event of a failure, the user was shown the relevant alert messages. In order to guarantee that only legitimate data changes appeared on the front end, answers were also examined for success status prior to processing.

Making appointments was one troubleshooting experience that was especially useful. At first, the appointments list did not show newly created appointments right away. After careful investigation, it was found that the problem stemmed from failing to refresh the list following the completion of the POST request. Upon successful creation, the `loadAppointments()` function was called, displaying the newly scheduled appointment instantly and giving the user rapid feedback.

1.5 Appointment Creation and Management

The creation of appointments is essential to the operation of the system. After users fill out the form, the data is sent to the backend using an AJAX POST request. The request is validated by the backend, which makes sure that the requested time slot is available and that all necessary fields are present.

The generated appointment object is then returned by the server and shown on the front end.

PATCH and DELETE requests are used to mark appointments as finished and remove them, respectively. The corresponding AJAX calls are triggered via buttons linked to each appointment. An error happened during implementation while marking appointments as finished; in certain cases, the backend accurately recorded the change, but the front-end status did not update. The DOM was not being updated following the PATCH request, according to the investigation. Reloading the appointment list fixed this, highlighting the significance of keeping the front-end display and backend state in sync.

Users can differentiate between scheduled and finished appointments thanks to the appointment list's dynamic status display. Careful consideration of string case was necessary when filtering by status; at first, filtering for "booked" failed since the backend returned lowercase values but the string

comparison was case-sensitive. The problem was fixed by normalising the strings' case before comparison, guaranteeing that status filtering functions consistently.

1.6 Searching and Filtering

Users can locate appointments by customer name using the search feature. An event listener is attached to a text input field in order to accomplish this, and each input event initiates a function. To make it easier to use, the function ignores case when filtering the appointments array for matches that contain the search string.

Select elements are used to implement filtering by barber and appointment status. At first, each filter was applied independently, which resulted in conflicts when several criteria were used at once. Accurate results are now shown in real-time by combining the filtering logic into a single function that iterates over all appointments and applies the active search and filter criteria successively. Maintaining consistency in filtering and searching once appointments were added, changed, or removed was a major difficulty. In order to guarantee that the displayed data consistently reflected the existing criteria, it was necessary to run the combined filter function following any changes to the appointment list. The outcome is a smooth user experience that allows for the simultaneous application of several filters and search queries, showcasing the usage of dynamic front-end logic to satisfy real-world user requirements.

1.7 Polling and Real-Time Updates

The system uses `setInterval` to build a polling method to keep the appointment display current. To retrieve the most recent appointments, the front-end submits a GET call to the API every 30 seconds. This period was selected to ensure that consumers receive timely updates without overtaxing the backend by striking a balance between data freshness and server load.

A problem occurred during implementation when polling resulted in duplicate elements showing up in the list of appointments. This resulted from rendering the modified data without first removing the existing DOM elements. The problem was fixed by resetting the `appointmentsDiv` content at the beginning of each polling cycle, resulting in a seamless and precise updating of the appointment list. In settings where there are several users at once, polling is very helpful since it guarantees that all users see consistent data. Polling was chosen due to its ease of use and adequacy for the anticipated scale of the program, even though WebSocket-based real-time communication could offer lower latency. This choice balances resource requirements and complexity, reflecting a realistic trade-off in system design.

1.8 Docker Integration and Deployment

The system is containerised using Docker for ease of deployment and reproducibility. The Dockerfile creates the working directory, installs dependencies, exposes the required port, and specifies the use of Node.js LTS. To define the service and map ports between the host and container, use Docker Compose.

Vulnerabilities in the base image were discovered during the first Docker build, underscoring the significance of utilising secure and maintained images. By updating the Node.js image to the most recent LTS version and checking npm dependencies for security advisories, the problem was lessened. Connecting to the API from the host computer presented another difficulty; at first, access was blocked due to improper port mapping in Docker Compose. The connectivity problem was fixed by changing the configuration to map port 3000 on the host to port 3000 in the container.

Docker integration removes problems associated with local development configurations and enables the program to execute consistently across various environments. This is especially helpful in collaborative or assessment environments, when it's crucial to make sure the program operates consistently across various machines.

2. Coding Challenges and Error Resolution

There were a number of mistakes and difficulties throughout development. For instance, inconsistent string cases caused appointment filtering to fail at first; this was fixed by normalising the text before comparison. Another problem was that the appointment list did not update right away after creation; this was resolved by invoking the `loadAppointments()` function following POST requests.

Furthermore, rigorous logic consolidation was needed for sophisticated filtering and search criteria. Multiple independent filters were utilised in early versions, which led to inaccurate or insufficient displays. The final implementation guarantees correct results by integrating all filters into a single function and applying them one after the other.

Another difficulty was creating time slots, especially when dealing with edge cases like the change of hours. Some positions were duplicated or skipped due to a computation error. All 15-minute intervals between 09:00 and 17:00 were produced appropriately by carefully evaluating the program and modifying the increment logic.

These encounters demonstrate the value of methodical testing, debugging, and attention to detail in web development. Every obstacle offered a chance to consider logic flow, code structure, and user experience, which ultimately led to a more dependable and resilient solution.

3. System Evaluation and Reflection

A number of fundamental concepts of contemporary web development, such as asynchronous data handling, API integration, dynamic DOM manipulation, and user interface design, are illustrated by the barbershop appointment system. Polling, filtering, and search features guarantee that the system is responsive and interactive, fulfilling the useful needs of appointment scheduling.

When one considers the development process, it is evident that debugging and iterative testing were essential to its success. Errors observed during the process gave important insights into the intricacies of dynamic web applications, especially when it came to managing multi-criteria filtering and real-time data updates. The system's dependability and repeatability were further improved by the incorporation of Docker, which demonstrated optimal deployment and environment management techniques.

Although the system works and exhibits all necessary capabilities, it might be improved by adding WebSocket-based real-time updates, integrating a persistent database like MongoDB or PostgreSQL, and improving the user interface with more sophisticated UX design. Scalability, maintainability, and general usefulness would all be enhanced by these extensions, laying the groundwork for additional advancements.

4. Conclusion

The creation of a barbershop appointment management system has been described in full in this study, together with important design choices, implementation techniques, and debugging procedures. In order to facilitate appointment creation, administration, search, filtering, and real-time updates via polling, the system combines a dynamic front-end with a RESTful API.

Filtering inconsistencies, DOM refresh problems, and Docker configuration failures were among the difficulties found during development that were methodically resolved, highlighting the significance of iterative testing and problem-solving in real-world web development.

This project offers a thorough illustration of contemporary web application development by fusing technical implementation with introspective assessment. In addition to delivering a useful, interactive, and maintainable appointment management solution, the system satisfies Assignment 2's criteria by shedding light on the difficulties and factors associated with creating dynamic web applications.

5. References

- [1] R. T. Fielding *et al.*, “Reflections on the REST architectural style and ‘principled design of the modern web architecture’ (impact paper award),” *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, Aug. 2017, doi: <https://doi.org/10.1145/3106237.3121282>.
- [2] S. Uttam Meshram, “View of Evolution of Modern Web Services – REST API with its Architecture and Design,” *Ijresm.com*, 2025. <https://journal.ijresm.com/index.php/ijresm/article/view/970/935> (accessed Jul. 03, 2025).
- [3] w3Schools, “JavaScript HTML DOM,” *W3schools.com*, 2019. https://www.w3schools.com/js/js_htmlDOM.asp
- [4] W3Schools, “JavaScript Events,” *W3schools.com*, 2019. https://www.w3schools.com/js/js_events.asp
- [5] W3Schools, “JavaScript Fetch API,” *www.w3schools.com*. https://www.w3schools.com/js/js_api_fetch.asp
- [6] GeeksforGeeks, “Ajax Introduction,” *GeeksforGeeks*, Jul. 09, 2018. <https://www.geeksforgeeks.org/javascript/ajax-introduction/>
- [7] GeeksforGeeks, “REST API Introduction,” *GeeksforGeeks*, Jun. 06, 2023. <https://www.geeksforgeeks.org/node-js/rest-api-introduction/>
- [8] GeeksforGeeks, “Express.js Tutorial,” *GeeksforGeeks*, Jan. 24, 2024. <https://www.geeksforgeeks.org/node-js/express-js/>
- [9] W3Schools, “Node.js Tutorial,” *W3schools.com*, 2019. <https://www.w3schools.com/nodejs/>
- [10] GeeksforGeeks, “JavaScript setInterval() Method,” *GeeksforGeeks*, Jan. 22, 2024. <https://www.geeksforgeeks.org/javascript/javascript-setinterval-method/>
- [11] GeeksforGeeks, “JSON Introduction,” *GeeksforGeeks*, Dec. 04, 2021. <https://www.geeksforgeeks.org/javascript/json-introduction/>
- [12] GeeksforGeeks, “What is Docker?,” *GeeksforGeeks*, Jul. 24, 2020. <https://www.geeksforgeeks.org/devops/introduction-to-docker/>
- [13] GeeksforGeeks, “Docker Compose,” *GeeksforGeeks*, Sep. 02, 2021. <https://www.geeksforgeeks.org/devops/docker-compose/>
- [14] L. Richardson and S. Ruby, “RESTful Web Services,” *Google Books*, 2026. https://books.google.co.uk/books?hl=en&lr=&id=XUaErakHsoAC&oi=fnd&pg=PR7&dq=RESTful+Web+Services+Richardson+Ruby&ots=5nhnFpsIsC&sig=Q3aPAJ-V5bXlS27ahnv-G6LlhyE&redir_esc=y#v=onepage&q=RESTful%20Web%20Services%20Richardson%20Ruby&f=false
- [15]

A. Mesbah and A. van Deursen, "A component- and push-based architectural style for ajax applications," *Journal of Systems and Software*, vol. 81, no. 12, pp. 2194–2209, Dec. 2008, doi: <https://doi.org/10.1016/j.jss.2008.04.005>.

[16]

M. Mikowski and J. Powell, "Single Page Web Applications," *Google Books*, 2026.
https://books.google.co.uk/books?hl=en&lr=&id=eDszEAAAQBAJ&oi=fnd&pg=PT16&dq=Single+Page+Web+Applications+Mikowski&ots=_FyIF2jlE-&sig=woghFVjniCdceifyRGxkCl4Des8&redir_esc=y#v=onepage&q=Single%20Page%20Web%20Applications%20Mikowski&f=false (accessed Jan. 05, 2026).

[17]

P. Th. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, Jun. 2003, doi: <https://doi.org/10.1145/857076.857078>.

Cloud-Based Web Services Report 2 - WR

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

56/100

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		2
Test documentation and testing		3
A reasonably well presented report and account of implementation.		

A detailed documentation and instructions would have been desirable. Reasonably good understanding demonstrated of the functionality of the SPA, middleware and API. Evidence of UI testing would have been desirable here.		
4. Practical Application Skills	Assignment 2: 50	
AJAX implementation REST API Nice organised clean code and files. A working UI but not connecting to UI. Attention to detail on ports and URIs would have been advantageous. Documentation with instructions would have been beneficial. A nice use of Docker for server side, should it have worked.		27
5. Transferable Skills for Life and Professional Practice	Assignment 2: 10	
Presentation		6

A well presented working UI. Refresh working would have been beneficial. A reasonable idea of the functionality of the code, better understanding of lines and functions would have been desirable.		
---	--	--

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8
