COMPX527 Cloud Computing Group Project

1st Dean Mason
School of Computing & Mathematical Sciences
The University of Waikato
Hamilton, New Zealand
dm214@students.waikato.ac.nz

2nd Oscar Ashburn
School of Computing & Mathematical Sciences
The University of Waikato
Hamilton, New Zealand
oa57@students.waikato.ac.nz

3rd Logan Cook
School of Computing & Mathematical Sciences
The University of Waikato
Hamilton, New Zealand
lc240@students.waikato.ac.nz

4th Emily McCullagh
School of Computing & Mathematical Sciences
The University of Waikato
Hamilton, New Zealand
em177@students.waikato.ac.nz

5th Rachel Martin
School of Computing & Mathematical Sciences
The University of Waikato
Hamilton, New Zealand
om41@students.waikato.ac.nz

***Abstract*—The goal was to create a cloud-based application that harnesses user data and open-source datasets to provide significant business value or societal benefits. Our solution, Blockbuster Battle, is an engaging and interactive movie rating guessing game. In this game, users are presented with two movie posters and must determine whether the second movie has a higher or lower rating compared to the first. This fun and educational game leverages movie ratings data to challenge and entertain users while providing insights into movie popularity and trends. This report will contain a detailed project proposal and frontend and backend architecture of the cloud-based application. It will cover the AWS services utilized, security considerations, and expenditure management. Additionally, the report will outline the workload breakdown and contributions of each team member.**

I. Introduction

In an era where digital entertainment and data-driven insights are increasingly intertwined, creating applications that offer both engagement and valuable information is crucial. This project addresses the intersection of these domains by developing a cloud-based application designed to harness user data and open-source datasets. The focus of our initiative is Blockbuster Battle, an innovative movie rating guessing game that challenges users to compare the ratings of movies based on their posters.

Blockbuster Battle not only provides a fun and interactive experience but also serves as an educational tool, offering insights into movie popularity and trends. The game is designed to engage users while leveraging rich datasets to enhance their understanding of film ratings.

The aim of this report is to present a detailed proposal for Blockbuster Battle, including the architecture of both the frontend and backend of the cloud-based application. It will cover the AWS services utilized, outline security considerations, and address management of expenditures. Additionally, the report will provide a comprehensive breakdown of the project workload and the contributions of each team member, demonstrating the collaborative effort behind the development of this innovative application.

II. Proposal

Our proposed AWS cloud project is a movie rating guessing game where each round the user is given two movie posters, and they must guess if the second movie has a higher or lower rating than that of the first movie being displayed on the left side. Regardless of whether the user is correct, it is revealed to the user the rating of the movie on the right side. If the user guesses correctly, they earn a point and move on to the next round, whereas if the user incorrectly guesses, their streak ends, and they are prompted to play again. On a correct guess, the movie on the right side continues into the next round. This movie is then placed up against a new random movie. The user again guesses if the new film has a higher or lower rating than the movie from the previous round. For bonus points the user can guess the exact value of the unknown rating. We aim to incorporate user accounts into this application. Users will create an account, storing basic user details, however, will harbor a means of tracking statistics like top user score. By storing users top scores, we envision enabling our users to participate in a global leaderboard system where they will be able to play against others to take the top score.

There are many potential extensions of the system if time allows. These include adding different tiers of difficulty for the time the user has during a round, toggling between critic or user rating (the dataset provides both these ratings) and providing a profile page for users who opt to create a profile with statistics about their game play (e.g. average correct high/low guess rate and average game score).

III. Frontend Architecture

The front-end architecture is built on the React + Vite framework, utilizing TypeScript as the primary programming language. This choice allows for a highly performant, modern web application, where React provides an efficient way to manage dynamic components and state, while Vite's fast development environment enhances the developer experience. The use of TypeScript strengthens the architecture by enforcing type safety, reducing runtime errors, and improving code maintainability. We chose this stack as it provides the flexibility needed for complex component structures, API integrations, and a scalable application architecture that will grow as project requirements evolve.

The front-end employs the Axios library to handle API requests, ensuring efficient communication with the back-end server. Axios simplifies the process of sending asynchronous HTTP requests, handling responses, and managing errors. Each API call has been implemented with error handling to provide clear, user-friendly prompts when issues arise, such as connection failures or invalid data responses. These notifications ensure users are well-informed about any problems without overwhelming the users with technical details. Axios' promise-based API is conducive to the asynchronous nature of modern web applications, enhancing both performance and user experience.

The front-end build files are deployed to an AWS S3 Bucket, which acts as a scalable and secure storage solution for hosting static assets. Once uploaded, these files are distributed via AWS CloudFront, a global content delivery network (CDN) that ensures low-latency access and high availability for users around the world. CloudFront accelerates content delivery by caching assets in edge locations close to the user's geographical region, thus enhancing page load speeds and overall performance. This deployment strategy provides a cost-effective, resilient, and scalable solution, ensuring that the website is fast, reliable, and secure across various traffic conditions.

IV. Backend Architecture

The back-end architecture of the application is built on Node.js, utilizing Express.js to manage the RESTful API that facilitates communication with the database. Node.js, a JavaScript runtime, ensures efficient handling of server-side operations, while Express.js provides a robust framework for building the API endpoints.

Authentication and route protection are managed using JavaScript Web Tokens (JWT), which secure the application by ensuring that only authenticated users can access certain routes and resources. This token-based authentication system helps in maintaining secure and controlled access to the application.

The RESTful API interacts with Amazon RDS (Relational Database Service), which serves as the database for storing and managing user information and movie data. Through this API, the application can perform various CRUD (Create, Read, Update, Delete) operations, allowing for the retrieval, posting, updating, and deletion of data. This setup ensures a scalable and efficient back-end infrastructure capable of handling user requests and managing data seamlessly.

V. AWS Services

A. *AWS Operations, Administration and Management Services*

1) CloudWatch: CloudWatch monitors AWS resources in real time, collecting metrics and provides the enablement of alarms to measure infrastructure health and performance.

Within the Blockbuster Battle AWS web application, we create several alarms to monitor CPU utilization, available storage, system status errors, storage capacity and percentage of bad HTTPS requests (4xx, 5xx).

2) Secrets Manager: Secrets manager securely encrypts, stores, and retrieves credentials for databases and services, allowing you to make calls to secrets manager for credentials instead of hard-coding them in your applications.

Within the Blockbuster Battle AWS web application, we utilised secrets manager to store JSON web token encryption keys and database credentials.

3) Identity and Access Management (IAM): Identity and Access Management (IAM) serves as a central location for managing identities of cloud users and administrators and their access to AWS services/resources.

Within the Blockbuster Battle AWS web application, we implemented IAM by creating custom roles, attaching AWS and customer-managed policies, granting access only where necessary.

Below are the IAM roles created and assigned to the respective team members based on their roles in the project:

Role name: BudgetManagerRole
Permissions: AWSBudgetsActionsWithAWSResourceControlAccess (Managed policy)
Users: Rachel

Role name: FrontEndDeveloperRole
Permissions: AmazonEC2FullAccess (Managed policy)

Users: Oscar, Rachel, Dean

Role name: BackEndDeveloperRole
Permissions: AmazonEC2FullAccess (Managed policy), AmazonRDSFullAccess (Managed policy), SecretsManagerReadWrite (Managed policy)
Users: Logan, Emily

Role name: RDSInstanceStateControlRole
Permissions: AmazonRDSInstanceStateControl (Customer-managed policy)
Users: Logan, Emily, Dean, Rachel, Oscar

Role name: IAMManagerRole
Permissions: IAMFullAccess (Managed policy)
Users: Dean, Rachel

Role name: S3ManagerRole
Permissions: AmazonS3FullAccess (Managed policy)
Users: Oscar

Role name: CloudFrontManagerRole
Permissions: CloudFrontFullAccess (Managed policy)
Users: Oscar

Role name: CloudWatchManagerRole
Permissions: CloudWatchFullAccess (Managed policy)
Users: Dean

Role name: ELBManagerRole
Permissions: ElasticLoadBalancingFullAccess (Managed policy)
Users: Oscar

We recognize that more restrictive permissions could have been applied, rather than granting full access to various resources, and that implementing resource-based policies rather than identity-based policies would have enhanced security. However, this approach would have hindered the team's ability to collaborate effectively, especially given that our project was a learning opportunity for everyone, involving the use of a wide range of services and resources. In the real world, principles like least privilege and other security best practices should be applied, but they must be balanced with the need for flexibility to maintain team efficiency.

B. AWS Network Services

1) Amazon Elastic Load Balancing (ELB): Amazon elastic load balancing (ELB) automatically routes incoming application traffic across multiple targets, such as EC2 instances to ensure high availability and fault tolerance.

Within the Blockbuster Battle AWS web application, we used elastic load balancing to enforce HTTPS and route the incoming traffic to the EC2 instances.

C. AWS Overlay Services

1) CloudFront: CloudFront accelerates the delivery of static and dynamic web content by routing requests through its global network of edge locations for low-latency performance. If content isn't cached at the closest edge, CloudFront retrieves it from your specified origin, like an S3 bucket or web server.

Within the Blockbuster Battle AWS web application, we used CloudFront to deliver the appropriate content to the user securely through a domain with a signed SSL/TLS certificate through AWS certificate manager.

D. AWS Compute Services

1) Amazon Elastic Compute Cloud (EC2): Amazon elastic compute cloud allows you to run virtual servers, known as instances, on-demand. It provides scalable computing capacity in the cloud, which means you can quickly scale your computing resources up or down based on your needs.

Within the Blockbuster Battle AWS web application, we used elastic compute cloud to run the web application backend and its related services.

E. AWS Storage Services

1) Amazon Simple Storage Service (S3): Amazon simple storage service provides distributed object storage. Data can be stored and retrieved anywhere, anytime. It offers scalability, data availability, security and performance.

Within the Blockbuster Battle AWS web application, we used S3 buckets to store the static build files for the web application front end.

F. AWS Database Services

1) Amazon Relational Database Service (RDS): Amazon RDS (Relational Database Service) is a managed database service from AWS that simplifies the setup, operation, and scaling of relational databases in the cloud. It supports multiple database engines, such as MySQL, PostgreSQL, Oracle, and SQL Server, and offers features like automatic backups, high availability, and scalability, allowing us to focus on application development rather than database maintenance.

Within the Blockbuster Battle AWS web application, we used RDS to store user information, including email addresses, password hashes and top scores. We also stored movie data including the title, rating, producers and year of the movie.

The specific flavour of database chosen was MySQL due to its wide use (and therefore, it wide base of support and documentation online), and its price (free to use with AWS RDS). We chose to run RDS on a T2.Micro EC2 instance as this was one of the cheapest instances available, and we don't expect the number of calls to our database to exceed what the T2.Micro can provide.

VI. Security Considerations

A. Back-end Security Considerations

The back-end application is hosted on an AWS EC2 instance and is accessible via a load balancer. To ensure secure communication, the load balancer only permits access through the HTTPS protocol.

Passwords are hashed on the back-end to enhance security. Although passwords are transmitted in plain text over HTTPS, hashing is performed server-side to prevent exposure of the hashing algorithm and protect sensitive data.

SQL injection is mitigated through the MySQL Node.js library we use, which sanitizes all user input before it is used in SQL queries. This ensures that any data received from users is properly cleaned to prevent injection attacks.

The JWT encryption key is securely stored in the Secrets Manager, eliminating the need to hard-code it in the application. This approach enhances security by keeping sensitive information out of the source code.

B. Front-end Security Considerations

All requests to the back-end are transmitted over HTTPS, ensuring secure and encrypted communication.

Passwords are transmitted in plain text over HTTPS to protect the hashing algorithm from being exposed if an attacker gains access to the source code.

Deprecated libraries were avoided to mitigate potential security vulnerabilities.

Password creation adheres to NIST guidelines, which include constraints on minimum and maximum length, as well as restrictions on sequential and repeating characters fig.12.

C. AWS Security Considerations

All cloud resources were allocated to a specific Virtual Private Cloud (VPC), ensuring that only the resources relevant to this project are contained within it.

Requests are routed internally through the load balancers when API requests are being made as the load balancer enforces HTTPS and routes the request to an EC2 based on load fig.11.

Only EC2 instances running the backend API are included in the load balancer's target group.

The AWS Secrets Manager was user to store the database credentials.

To enable HTTPS, the domain required a signed SSL/TLS certificate, which was obtained using AWS Certificate Manager.

IAM (Identity and Access Management) was used to control and manage user access and permissions across AWS resources, ensuring that only authorized users and applications could interact with specific services and data according to defined security policies and roles.

VII. AWS Expenditure

A. Overall Expenditure

TABLE I
AWS Costs Breakdown as at 15/09/2024 15:30

Description	**Amount (USD)**
Amazon Elastic Load Balancing	2.903
Amazon Elastic Compute Cloud - Compute	1.866
Amazon Virtual Private Cloud	1.761
Tax	1.12
EC2 - Other	0.84
Others	0.044
TOTAL:	**8.534**

The cost breakdown highlights that the majority of the expenditure is driven by core services like Amazon Elastic Load Balancing (ELB) and Elastic Compute Cloud (EC2), which together account for a significant portion of the total $8.534 USD. Notably, smaller costs such as taxes and miscellaneous "Others" contribute minimally, indicating that compute and networking services dominate the overall AWS usage.

B. Monthly Expenditure

TABLE II
AWS Costs for August 2024 as at 15/09/2024 15:30

Description	**Amount (USD)**
Amazon Elastic Load Balancing	0
Amazon Elastic Compute Cloud - Compute	0.078
Amazon Virtual Private Cloud	0.034
Tax	0.04
EC2 - Other	0.12
Others	0.001
TOTAL:	**0.273**

TABLE III
AWS Costs for September 2024 as at 15/09/2024 15:30

Description	**Amount (USD)**
Amazon Elastic Load Balancing	2.903
Amazon Elastic Compute Cloud - Compute	1.788
Amazon Virtual Private Cloud	1.727
Tax	1.08
EC2 - Other	0.72
Others	0.043
TOTAL:	**8.261**

VIII. Workload breakdown

A. *Oscar Ashburn*

- Front-end Development
- AWS Load Balancer
- Cloud Front setup
- Presentation
- Report

B. *Dean Mason*

- Front-end Development
- Identity and Access Management (IAM) Configuration
- Cloud Watch Configuration
- Presentation
- Report

C. *Logan Cook*

- Back-end Development
- RDS
- DNS redirect via S3

D. *Rachel Martin*

- Front-End Development
- Billing and Cost-Management
- Ec2 Fronted Deployment Investigation
- Presentation
- Identity and Access Management (IAM) Configuration

E. *Emily McCullagh*

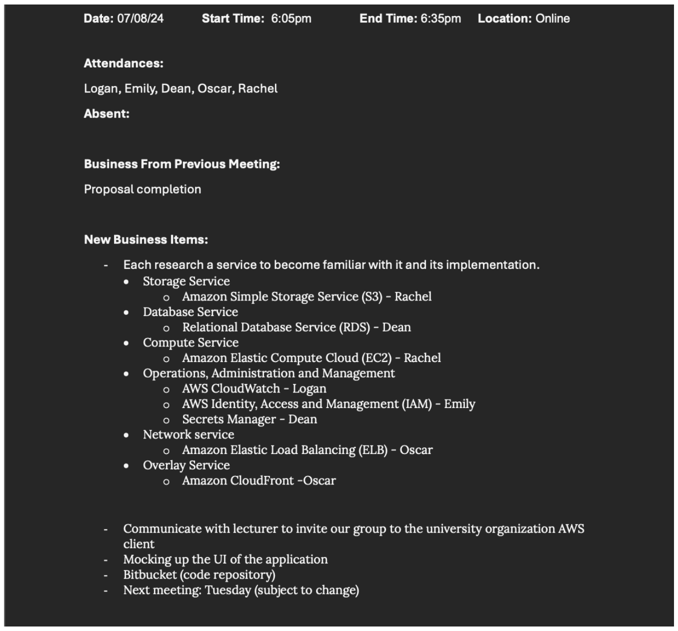
- Back-end Development Deployment
- RDS
- Secrets Manager
- Presentation

We believe that every team member made an equal contribution to the project, leveraging their unique skills to achieve a balanced and successful outcome. Dean, Oscar, and Rachel focused primarily on front-end development, while Emily and Logan concentrated on back-end development. Additionally, all team members contributed to configuring AWS services, providing the ability for all members to interact and experiment with AWS services. All members were present during both online and in person meetings and made valuable contributions to the project as a whole.

IX. Team Communications

Weekly meetings were held online through the Discord app, and in person in the R block labs. Minutes were recorded for these meetings, which are available to be viewed in the appendix.

Appendix A
Meeting Minutes Session 1



Date: 07/08/24 **Start Time:** 6:05pm **End Time:** 6:35pm **Location:** Online

Attendances:

Logan, Emily, Dean, Oscar, Rachel

Absent:

Business From Previous Meeting:

Proposal completion

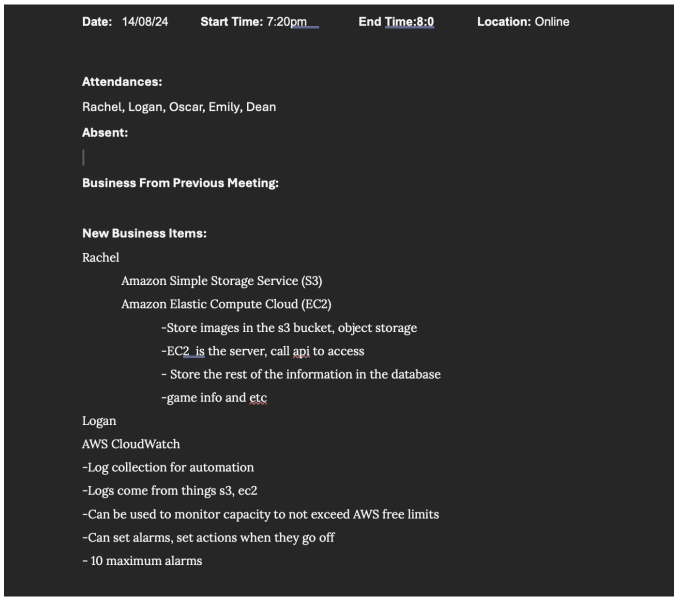
New Business Items:

- Each research a service to become familiar with it and its implementation.
  - Storage Service
    - Amazon Simple Storage Service (S3) - Rachel
  - Database Service
    - Relational Database Service (RDS) - Dean
  - Compute Service
    - Amazon Elastic Compute Cloud (EC2) - Rachel
  - Operations, Administration and Management
    - AWS CloudWatch - Logan
    - AWS Identity, Access and Management (IAM) - Emily
    - Secrets Manager - Dean
  - Network service
    - Amazon Elastic Load Balancing (ELB) - Oscar
  - Overlay Service
    - Amazon CloudFront -Oscar

- Communicate with lecturer to invite our group to the university organization AWS client
- Mocking up the UI of the application
- Bitbucket (code repository)
- Next meeting: Tuesday (subject to change)

Fig. 1. Meeting Minutes 07/08/24

Appendix B
Meeting Minutes Session 2



Date: 14/08/24 **Start Time:** 7:20pm **End Time:**8:0 **Location:** Online

Attendances:

Rachel, Logan, Oscar, Emily, Dean

Absent:

Business From Previous Meeting:

New Business Items:

Rachel

Amazon Simple Storage Service (S3)

Amazon Elastic Compute Cloud (EC2)

-Store images in the s3 bucket, object storage

-EC2 is the server, call api to access

- Store the rest of the information in the database

-game info and etc

Logan

AWS CloudWatch

-Log collection for automation

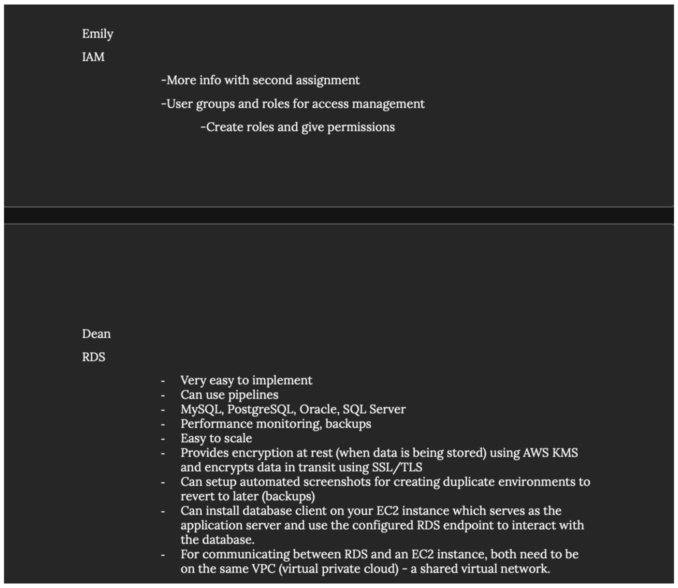
-Logs come from things s3, ec2

-Can be used to monitor capacity to not exceed AWS free limits

-Can set alarms, set actions when they go off

- 10 maximum alarms

Fig. 2. Meeting Minutes 14/08/24 A


Emily

IAM

-More info with second assignment

-User groups and roles for access management

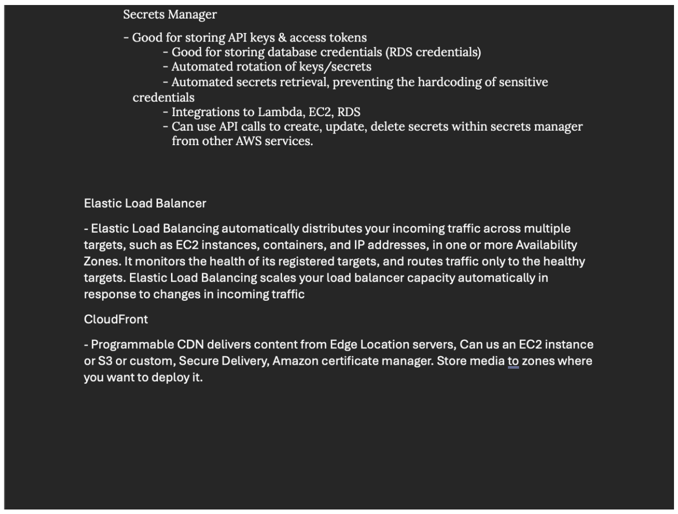
-Create roles and give permissions

Dean

RDS

- Very easy to implement
- Can use pipelines
- MySQL, PostgreSQL, Oracle, SQL Server
- Performance monitoring, backups
- Easy to scale
- Provides encryption at rest (when data is being stored) using AWS KMS and encrypts data in transit using SSL/TLS
- Can setup automated screenshots for creating duplicate environments to revert to later (backups)
- Can install database client on your EC2 instance which serves as the application server and use the configured RDS endpoint to interact with the database.
- For communicating between RDS and an EC2 instance, both need to be on the same VPC (virtual private cloud) - a shared virtual network.

Fig. 3. Meeting Minutes 14/08/24 B


Secrets Manager

- Good for storing API keys & access tokens
  - Good for storing database credentials (RDS credentials)
  - Automated rotation of keys/secrets
  - Automated secrets retrieval, preventing the hardcoding of sensitive credentials
  - Integrations to Lambda, EC2, RDS
  - Can use API calls to create, update, delete secrets within secrets manager from other AWS services.

Elastic Load Balancer

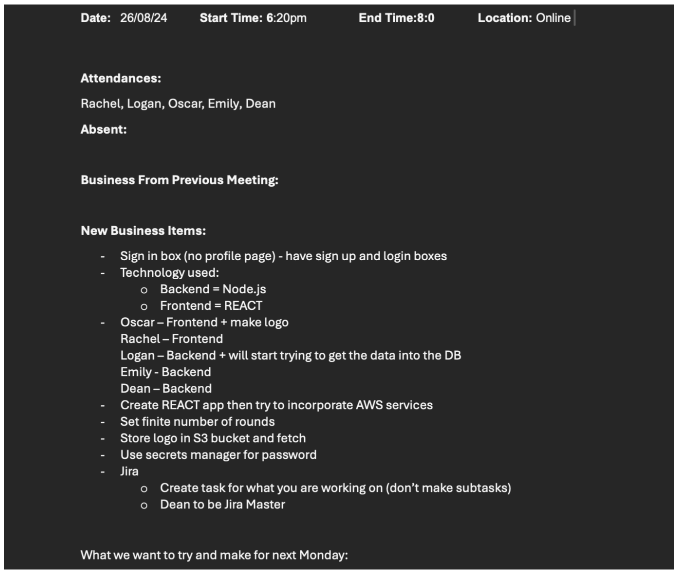
- Elastic Load Balancing automatically distributes your incoming traffic across multiple targets, such as EC2 instances, containers, and IP addresses, in one or more Availability Zones. It monitors the health of its registered targets, and routes traffic only to the healthy targets. Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic

CloudFront

- Programmable CDN delivers content from Edge Location servers, Can us an EC2 instance or S3 or custom, Secure Delivery, Amazon certificate manager. Store media to zones where you want to deploy it.

Fig. 4. Meeting Minutes 14/08/24 C

A. *Meeting Minutes Session 3*


Date: 26/08/24 **Start Time: 6**:20pm **End Time:8:0** **Location:** Online

Attendances:

Rachel, Logan, Oscar, Emily, Dean

Absent:

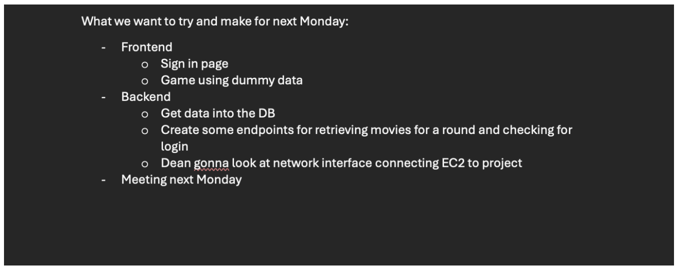
Business From Previous Meeting:

New Business Items:

- Sign in box (no profile page) - have sign up and login boxes
- Technology used:
  - Backend = Node.js
  - Frontend = REACT
- Oscar – Frontend + make logo
  Rachel – Frontend
  Logan – Backend + will start trying to get the data into the DB
  Emily - Backend
  Dean – Backend
- Create REACT app then try to incorporate AWS services
- Set finite number of rounds
- Store logo in S3 bucket and fetch
- Use secrets manager for password
- Jira
  - Create task for what you are working on (don't make subtasks)
  - Dean to be Jira Master

What we want to try and make for next Monday:

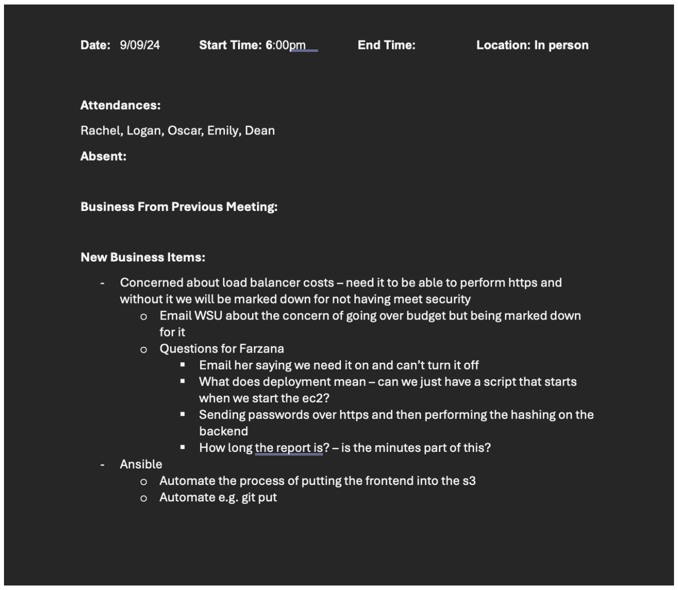
Fig. 5. Meeting Minutes 26/08/24 A


What we want to try and make for next Monday:

- Frontend
  - Sign in page
  - Game using dummy data
- Backend
  - Get data into the DB
  - Create some endpoints for retrieving movies for a round and checking for login
  - Dean gonna look at network interface connecting EC2 to project
- Meeting next Monday

Fig. 6. Meeting Minutes 26/08/24 B

Appendix C
Meeting Minutes Session 4



Date: 9/09/24 **Start Time:** 6:00pm **End Time:** **Location:** In person

Attendances:

Rachel, Logan, Oscar, Emily, Dean

Absent:

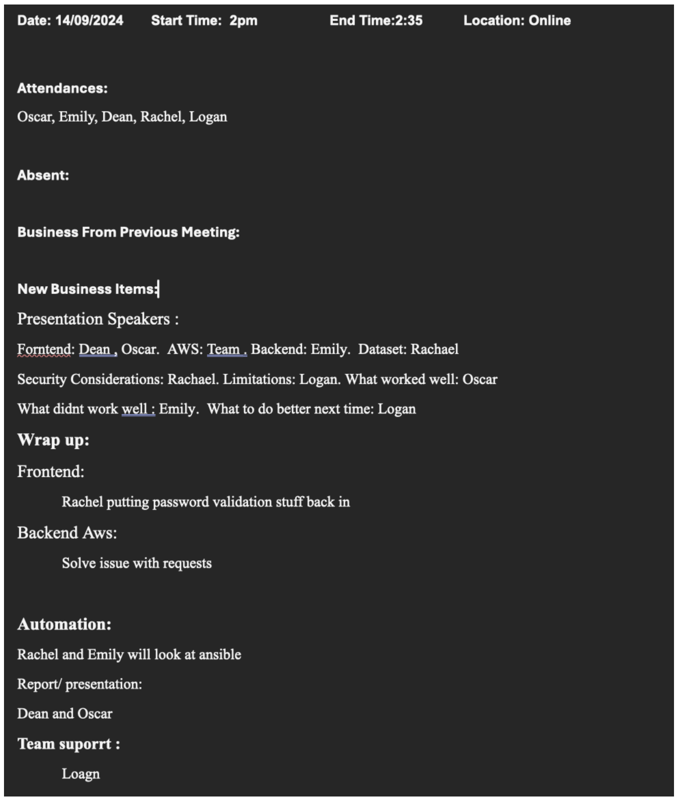
Business From Previous Meeting:

New Business Items:

- Concerned about load balancer costs – need it to be able to perform https and without it we will be marked down for not having meet security
  - Email WSU about the concern of going over budget but being marked down for it
  - Questions for Farzana
    - Email her saying we need it on and can't turn it off
    - What does deployment mean – can we just have a script that starts when we start the ec2?
    - Sending passwords over https and then performing the hashing on the backend
    - How long the report is? – is the minutes part of this?
- Ansible
  - Automate the process of putting the frontend into the s3
  - Automate e.g. git put

Fig. 7. Meeting Minutes 09/09/24

Appendix D
Meeting Minutes Session 5



Date: 14/09/2024 **Start Time:** 2pm **End Time:**2:35 **Location:** Online

Attendances:

Oscar, Emily, Dean, Rachel, Logan

Absent:

Business From Previous Meeting:

New Business Items:

Presentation Speakers :

Forntend: Dean , Oscar. AWS: Team . Backend: Emily. Dataset: Rachael

Security Considerations: Rachael. Limitations: Logan. What worked well: Oscar

What didnt work well : Emily. What to do better next time: Logan

Wrap up:

Frontend:

Rachel putting password validation stuff back in

Backend Aws:

Solve issue with requests

Automation:

Rachel and Emily will look at ansible

Report/ presentation:

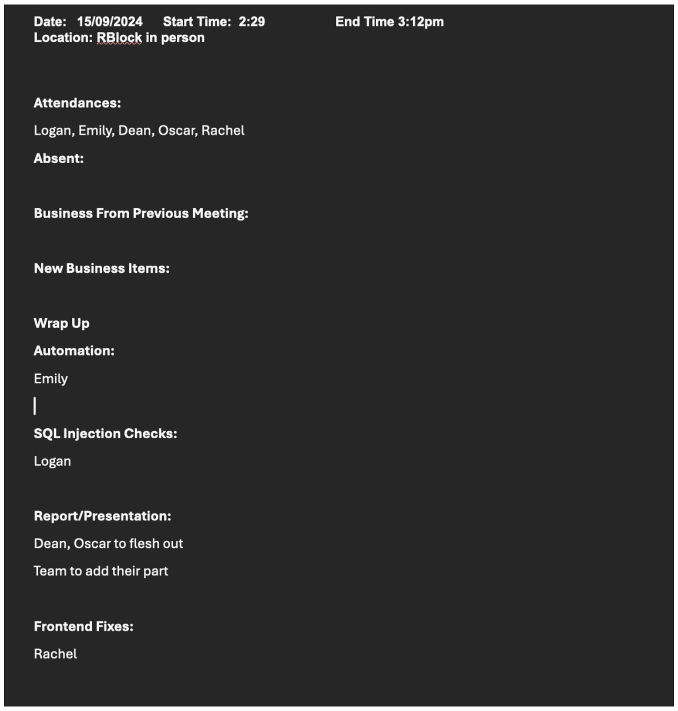
Dean and Oscar

Team suporrt :

Loagn

Fig. 8. Meeting Minutes 14/09/24

Appendix E
Meeting Minutes Session 5


Date: 15/09/2024 Start Time: 2:29 End Time 3:12pm
Location: RBlock in person

Attendances:

Logan, Emily, Dean, Oscar, Rachel

Absent:

Business From Previous Meeting:

New Business Items:

Wrap Up

Automation:

Emily

SQL Injection Checks:

Logan

Report/Presentation:

Dean, Oscar to flesh out

Team to add their part

Frontend Fixes:

Rachel

Fig. 9. Meeting Minutes 14/09/24

Appendix F
Load Balancer and CDN

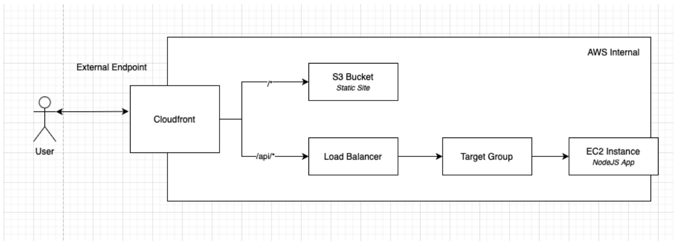


Fig. 10. Content Delivery Network

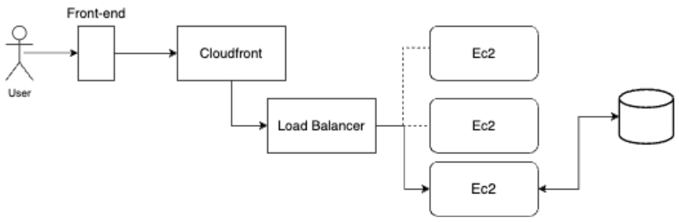


Fig. 11. User Request Flow

Appendix G
Password Validation

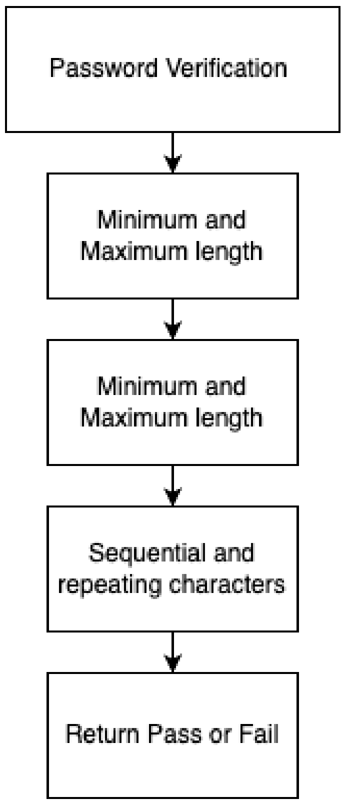

Password Verification
Minimum and Maximum length
Minimum and Maximum length
Sequential and repeating characters
Return Pass or Fail

Fig. 12. Password Validation