CS301 G2T3

Term 1 - AY 2023/2024

Project Code: project-2023-24t1-g2-t3B

Name	Matriculation Number
Declan Joshua Tan	01398714
Goh Wei Han	01431924
Kim Han Nah	01407978
Loh Jin Han	01392051
Sean Tan Ming Hoe	01377385
Shen Qin	01386267
Wong Ginn Munn	01447943

Repository Links
Frontend UI Panel: https://github.com/cs301-itsa/project-2023-24t1-g2-t3b-frontend
Backend Services: https://github.com/cs301-itsa/project-2023-24t1-g2-t3b-service
Documentation: https://documenter.getpostman.com/view/23771384/2s9YXmYgCa

Table Of Contents

Stakeholders

In this table, we categorise the principal Stakeholders engaging with the application, detailing their specific permissions across various services. The acronym CRUD represents the capacities to Create, Read, Update, and Delete within each service.

Stakeholder	Stakeholder Description	Permissions			
		User Storage	Point Ledger	Logs	Role
Owner	Ascenda engineers, managers.	CRUD	RU	R	CRUD
Manager	Ascenda managers	CRU	RU	R	CRUD
Engineer	Ascenda engineers	R	R	R	R
Product Manager	Ascenda PMs	R (only see non-admins)	R	N/A	R
Customer Support and Service Teams	Assist customers, resolve issues, and answer inquiries related to loyalty programs	R	R	R	R
IT Security Team	Vetting the logs for Customers	R	N/A	R	R
IT Infrastructure Team	Managing all the relevant permissions in managing Ascenda's IT infrastructure	CRUD	CRUD	R	CRUD
Default	For users that were not assigned any roles when signed up	R	N/A	N/A	N/A

Table 1: Stakeholders and Permissions

Key Use Cases

Use Case 1 - Admin wants to create a role	
Description	Admin should be able to create new roles and allow them to change permission dynamically. It needs to be scalable Roles and permission should be manageable on the fly without redeploying the services
Actors	Owner, Managers, IT Infrastructure team
Main Flow of Events	1. The admin accesses the ‘Role” page and clicks on “Create role” for the Admin Proxy UI 2. The system presents a registration form to the admin, requesting the following information with its respective CRUD permissions: a. Role Name - Unique Identifier b. User Storage - C/R/U/D c. Point Ledger - C/R/U/D d. Logs - R e. Role - C/R/U/D 3. The Admin will enter the required information into the registration form. 4. The Admin submits the form 5. At the Admin UI page, validation checks take place to ensure correct input formatting for the different fields. 6. Requests will be sent to the proxy app. In the meantime, a log for the requested action will be written to the logs database. 7. A newly created Role will be inserted into the Role DB, and a response will be sent back to the admin proxy app. 8. On receiving the response, the proxy app will write a log for the successful action to the logs database, as well as send back a successful response to the Admin UI page
Alternative Flow of Events	● Error while creating Role ○ Role name already taken ○ Error while adding to the database as the database is down.
Pre-conditions	Assuming the admin already successfully logged in and has the relevant permissions to add a role.
Post-conditions	● Should be able to assign roles to users with the respective CRUD permission to each system

Use Case 2 - Admin wants to enrol new user	
Description	This use case describes the process of an admin enrolling a new user. It ensures that the admin is authorised to create a new user. It outlines the steps involved and the interactions between the admin and the system.
Actors	Owner, Manager, IT Infrastructure

<table>
<tr><td>Main Flow of Events</td><td>1. The admin accesses the “Users” page which retrieves users and points data via GET requests to their respective apps. This information is displayed to the user on the “Users” page.
2. User clicks on “Enrol User”, and enters the “Create New User”.
3. The system presents a registration form to the admin, requesting the following information:<table><tr><td>Name</td><td>Email Address</td><td>Role (Optional)</td></tr></table>4. The user enters the required information into the registration form.
5. Admin submits form
6. At the Admin UI page, validation checks take place to ensure correct input formatting for the different fields.
7. The POST requests will be sent to the “Admin Proxy App”, after which the proxy app forwards the request to the “User” service. In the meantime, a log for the requested action will be written to the “Logs” database.
8. At the “User” service, validation checks take place to ensure correct input formatting for the different fields, as well as a check to ensure that the requested User object does not already exist in the database.
9. A newly created User object will be inserted into the User database, and a response will be sent all the way back to the “Admin Proxy App”.
10. On receiving the response, the “Admin Proxy App” will write a log to the “Logs database” for the successful action, as well as send back a successful response to the client.</td></tr>
<tr><td>Alternative Flow of Events</td><td>● Error while creating user
○ Invalid email/naming convention.
○ Error while adding to the database as the database is down.
○ The user is already registered.
■ Continuing from point 8 in the main flow of events, a custom error message will be returned to the “Admin Proxy App” where it is returned to the user. A log will be written to the “Logs database” for the unsuccessful action.</td></tr>
<tr><td>Pre-conditions</td><td>● Admin must have permission to create new users.
● The user is not already registered with the system.</td></tr>
<tr><td>Post-conditions</td><td>● A new user is created.</td></tr>
</table>

Use Case 3 - Owner wants to update the points of a user	
Description	This use case describes the process when an owner needs to adjust the loyalty points for a specific user. The owner submits a points adjustment request through Form 3, which interacts with the Points Ledger.
Actors	Owner

Main Flow of Events	1. The owner accesses the "Users” Page, which queries data from User app and the Points Ledger app, displaying user details, including points balance. 2. The owner selects the specific user for whom points need adjustment from the “Users” Page. 3. The owner then enters the specific page for the user with the relevant ID. 4. Form 3 is presented below the details of the user, allowing the owner to specify the points adjustment, which includes: ● Adding or deducting a specific number of points. 5. The owner submits the points adjustment request via Form 3. 6. Form 3 communicates with the Proxy app which will then communicate with Points Ledger app, sending the adjustment request for processing 7. During this process, a log for the requested action will be written to the logs database 8. The Points Ledger validates the request and updates the user's points balance accordingly. 9. The Points Ledger app sends a response confirming the successful points adjustment back to Form 3 via the Proxy app. 10. A log entry for the points adjustment is created by the Proxy app to record the action. 11. Form 3 displays a success message to the owner.
Alternative Flow of Events	● If there are technical issues preventing the points adjustment: ○ Form 3 displays an error message and advises the owner to try again later. ○ The use case terminates.
Pre-conditions	● The owner must be logged into the Ascenda admin system. ● To edit the points, one must have the appropriate permissions to do so. ● The user for whom points are being adjusted must exist in the system.
Post-conditions	● The user's points balance is updated as per the admin's request. ● A log entry is generated in the Logs DB to record the points adjustment action. ● The owner receives a successful response indicating the completion of the points adjustment process. ● The owner, after receiving the successful response, can verify the updated points balance on the Users Listing Page. ● The owner can view a log of the points adjustment on the "User” Logs Page.

Proposed Budgets

Development Budget

Activity Name	Description	Cost
Project Management	Planning, organising, and overseeing a project	1*40 hours/person=40hours
Frontend Development	Develop the UI/UX of the webpages	2*15 hours/person=30hours
Backend Development	Develop endpoints of webpages with backend logic	5*15 hours/person=75hours
Databases	Develop the multiple databases	3*5 hours/person=15hours
Bug fixes and refactoring	Fixing and refactoring code	7*10 hours/person=70hours
Testing	Unit tests, Integration Testing, Component Testing, User Acceptance Testing, Security Testing	7*10 hours/person=70hours
Documentation	Documenting and improving the readability of code	7*3 hours/person=21hours
Continuous Integration/ Continuous Development	Setting up and deploying the automation pipelines (Integration, Tests, Delivery) to streamline development workflows	7*10hours/person=70 hours
		Total = 411hours

Production Budget[1]

AWS Services	Description and General Assumptions	Monthly Costs
Route 53	Domain Name System (DNS) web service Assume 120 million monthly requests	98.90 USD
Simple Storage Service (S3)	Front-end Admin Panel UI will be hosted on S3 Assume the size of the static pages is 2MB in total and 1TB of data transfer	170.88 USD
	Storage of logs - S3(Infrequent Access) Assume	131.88 USD
API Gateway	Internal facing API gateway. 120 million Rest API calls	110.00 USD
ECS	Admin Proxy, User and Point applications Assume minimum of 6 c6g.medium	416.98 USD
Lambda	Lambda functions for Maker Checker, Logs and Roles	39.80 USD
Simple Email Service (SES)	SES for Maker-Checker to approve or deny the request	120.00 USD
Elastic Load Balancer	2 Internal Facing Network Load Balancers 1 Internet Facing Network Load Balancers 1 Internet Facing Application Load Balancers	188.00 USD
DynamoDB (On Demand)	One DynamoDB table for each service [Approval Table, Role Table, User Table, Points Table]	382.40 USD
CloudWatch	View logs from ecs, lambda, Sumo Logic	100.00 USD
Secrets Manager	Storing of secrets	7 USD
Simple Queue Service	Message queuing system to deliver logs messages to Logs service on AWS lambda (150 million queue requests per month)	59.60 USD
AWS WAF	Web Application Firewall that fronts and protects our Application against common web exploits (One web access control, 5 rules, 200 million web requests per month)	160.00USD
AWS Certificate Manager	Provisioning of SSL/TLS Certificates	Free
		Total 1,985.44 USD/Month

[1] Refer to Appendix F: Budget Assumptions

Key Architectural Decisions

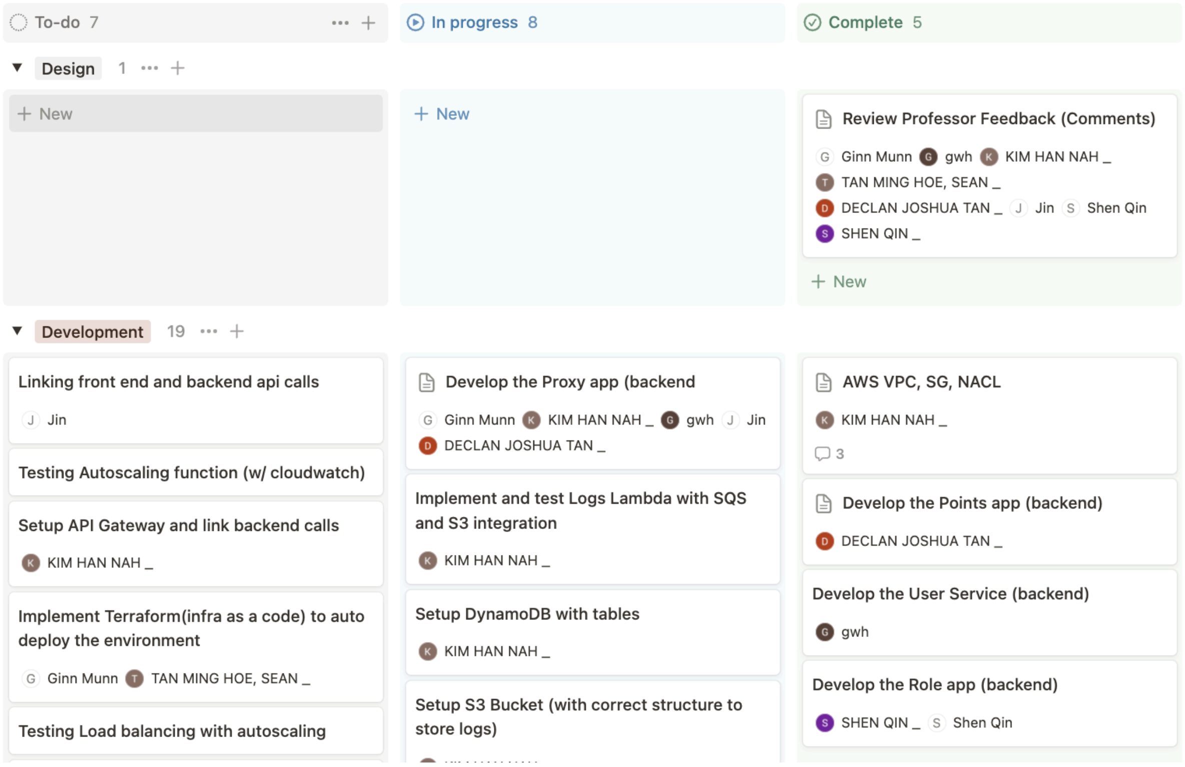
The following table outlines the critical architectural choices we made during the implementation of our solution.

Architectural Decision - Usage of DynamoDB	
ID	**1**
Issue	Budgeting concerns with data access and storage
Architectural Decision	We opted to use DynamoDB as the database for all our microservices as it provides sufficient flexibility and features to fulfil the requirements of our data access and storage.
Assumptions	Items have size <4kB; No future requirements for SQL-like joins or complex transactions; Acceptable latency from using strongly consistent reads/ writes
Alternatives	AWS RDS
Justification	AWS RDS allows for more features, like complex transactions, joins and strong consistency by default. However these features are not required within the scope of our key use cases, and the fair markup on using AWS RDS makes it an inferior solution for data access and storage for our project.
Architectural Decision - Usage of AWS ECS	
ID	**2**
Issue	Deployment, management and scaling of applications
Architectural Decision	We chose to use AWS ECS as it provided a high level of abstraction for the deployment of containerised applications with provisioning, patching, and scaling fully managed for us.
Assumptions	No future requirement for fine-grained management of underlying infrastructure
Alternatives	AWS EC2
Justification	ECS itself manages clusters of EC2 instances for us. Even though EC2 allows for full control over the operating system, installed software, and configurations in a virtual server, this level of detail is not required for our use cases, and we believe that developer resources are better utilised on application development over infrastructure configuration.
Architectural Decision - Use AWS Lambda for Maker-Checker and Role	
ID	**3**
Issue	Applications with unpredictable or low traffic
Architectural Decision	AWS Lambda is used as its pay-per-use nature works well with such applications.
Assumptions	Role and maker-checker applications will have low and sporadic usage in practice, respectively.
Alternatives	ECS
Justification	Using a constantly running ECS task is not appropriate for applications with unpredictable or low traffic as it will incur unnecessary costs.

Architectural Decision - Use AWS Network Load Balancer (NLB) for Frontend, User and Points services	
ID	**4**
Issue	Minimising latency with load balancer
Architectural Decision	We decided to use AWS NLB due to its low-latency, high-throughput, and TCP/UDP traffic handling.
Assumptions	No future requirement for advanced routing features for content-based and path-based routing.
Alternatives	AWS ALB
Justification	AWS ALB provides support for advanced routing features at the cost of latency. Since these routing features are not required as per our use cases, we decided that NLB was the better option.
Architectural Decision - Use AWS Application Load Balancer (ALB) for Admin proxy	
ID	**5**
Issue	Provides advance routing feature
Architectural Decision	We decided to use AWS ALB due to requirements on the Admin Proxy to be secured.
Assumptions	No requirement for low latency networking.
Alternatives	AWS NLB
Justification	AWS NLB provides low latency networking; however, we require advanced routing features and additional security features such as implementing a Web Application Firewall in front of the ALB
Architectural Decision - Amazon S3 for Logs	
ID	**6**
Issue	Cost effectiveness with consistent log storage
Architectural Decision	Logs from all services are stored together, which requires a high volume of writes to the database. We decided to use Amazon S3 to store our logs as it offers seamless scalability without the need for a complex setup at a low cost.
Assumptions	No future requirement to consistently read logs data.
Alternatives	DynamoDB
Justification	DynamoDB allows for both write and read-intensive workloads, as compared to S3 which can only work well with write-intensive workloads. However, in accordance with our use cases – the logs database will largely only be written to, and only read from sporadically.
Architectural Decision - Use of Spring Security	
ID	**7**
Issue	Authentication and Authorization
Architectural Decision	We chose to use Spring Security for handling User Authentication and Authorization to various protected routes in our application
Assumptions	We are using the Spring framework to develop the application.
Alternatives	Other third-party Authentication services such as Okta, AWS Cognito, or Google auth.
Justification	Since we are using the Spring framework to develop our admin proxy app, which deals with the application's authentication and authorization, we chose

Development View

Scrumban



To-do 7

In progress 8

Complete 5

Design 1

+ New

+ New

Review Professor Feedback (Comments)
Ginn Munn · gwh · KIM HAN NAH _ · TAN MING HOE, SEAN _ · DECLAN JOSHUA TAN _ · Jin · Shen Qin · SHEN QIN _

+ New

Development 19

Linking front end and backend api calls
Jin

Testing Autoscaling function (w/ cloudwatch)

Setup API Gateway and link backend calls
KIM HAN NAH _

Implement Terraform(infra as a code) to auto deploy the environment
Ginn Munn · TAN MING HOE, SEAN _

Testing Load balancing with autoscaling

Develop the Proxy app (backend
Ginn Munn · KIM HAN NAH _ · gwh · Jin · DECLAN JOSHUA TAN _

Implement and test Logs Lambda with SQS and S3 integration
KIM HAN NAH _

Setup DynamoDB with tables
KIM HAN NAH _

Setup S3 Bucket (with correct structure to store logs)

AWS VPC, SG, NACL
KIM HAN NAH _
3

Develop the Points app (backend)
DECLAN JOSHUA TAN _

Develop the User Service (backend)
gwh

Develop the Role app (backend)
SHEN QIN _ · Shen Qin

Figure 1: Notion Scrumban Board

We opted to follow the more fluid and flexible Scrumban approach, over Scrum which was used for previous projects. We found the increased flexibility and fluidity of the Scrumban approach, where we split general tasks in a large semester-long Scrumban bucket greatly improved our ability to independently and quickly implement changes to our various tasks. Due to the nature of the project, there were periods with low workloads, where we could spend more time working on the project, as well as crunch periods where our focus lay elsewhere. This made a pure Scrum approach with strict sprint deadlines less effective in completing this project and showed a situation where the Scrumban approach could really shine.

Testing

We developed unit test cases for our fundamental services (User and Points service) to verify that CRUD operations on our DynamoDB remain functional even after introducing new features to our application.

Continuous Integration / Continuous Development (CI/CD)

Our team has implemented a streamlined CI/CD pipeline, which is automatically triggered by any push to our main branch. This pipeline efficiently handles the deployment of all our microservices. It's designed to either build a Dockerized image of the application for deployment to our ECS cluster or package the Python files for deployment to Lambda Functions. We utilised GitHub Actions, outlining the workflow processes for each microservice through \`push-ecr.yml\` and \`deploy.yml\` files. The CI/CD process unfolds as follows:

ECS Deployment (push-ecr.yml)	**Lambda Functions Deployment** (deploy.yml)
1. Checks out the source code. 2. Configures AWS credentials for accessing AWS services. 3. Set up JDK 17 and build the JAR file for the Java project. 4. Builds, tags, and pushes the Docker image to AWS ECR. 5. Updates the ECS task definition with the latest Docker image ID. 6. Deploys the updated task definition to the specified ECS service and cluster. 7. Waits for the ECS service to reach stability.	1. Development occurs on the dev branch, where linting is performed using Flake8 and Pylint with GitHub Actions. 2. When pushed to the main branch, GitHub Actions initiates \`deploy.yml\`. 3. Check out code from the repository. 4. Packages the Python file in the src directory for each Lambda function. 5. Deploys the zipped file to the respective Lambda function on AWS via AWS CLI commands, using environment variables set from GitHub Secrets for AWS credentials and region.

This automated pipeline has significantly streamlined our deployment process, enabling rapid updates to our production environment with just a simple push. It encourages us to make small, incremental changes rather than waiting for the completion of large services before production deployment, thus minimising potential disruptions to our production environment.

Solution View (Maintainability)

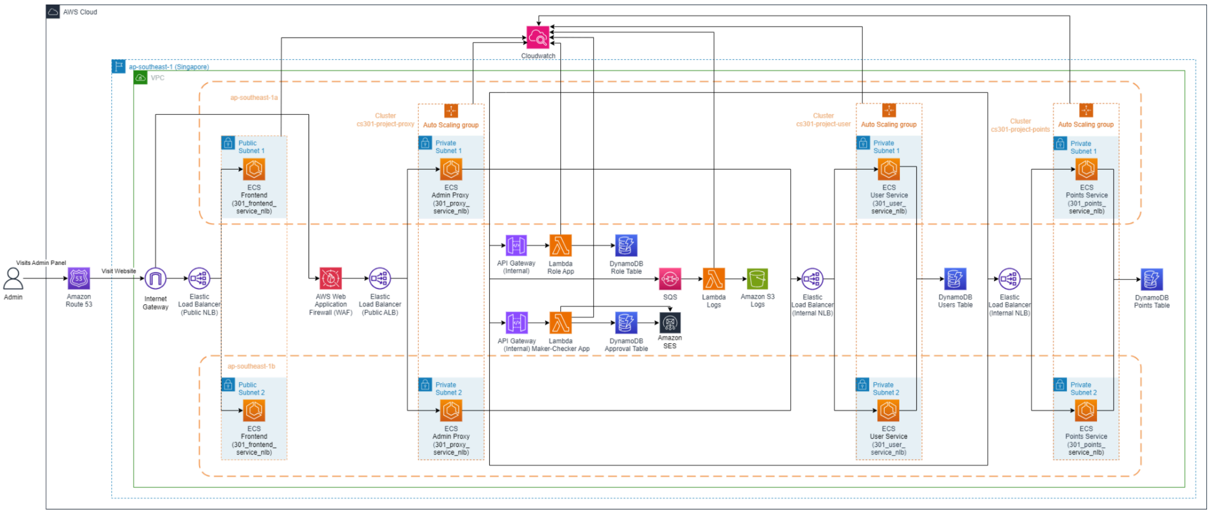


Figure 2: AWS Architecture Diagram

Link to the Diagram

Design Patterns

In terms of design patterns, all of our microservices make use of a clear MVC (Model View Controller) design that makes it more extensible for future use. We observed the Single Responsibility principle by making sure each class has a well-defined function and purpose and extracting any repeated code into other classes. We also followed the dependency inversion principle by doing a constructor injection for the creation of all our classes. We used Lombok Builder annotation to implement a builder pattern for our User class.

Multi-layered Architecture

In the Points microservice, a multi-layered architecture design is adopted, with API, Business and Data layers that use separate models for a conceptual Points object. In the API layer, we have Points, while in the Business layer we use PointsBiz and in the Data layer, PointsDoc is used. An object mapper is used to map fields from one object to another, allowing us to transform the object as it travels up or down the layers. While the objects have the same fields for now, this architectural style leads to loose coupling between the layers that handle API calls, run business logic and access the underlying database. This allows us to easily make changes to the business logic, for example, without needing to change the other layers.

Microservice Architecture

We adopted a microservice architecture in our design by separating in accordance with data access needs. For example, we know that the application must be able to access both a Users and Points database, hence, we separate the 2 into microservices. Repeating the process, we ended up with our Admin Proxy (composite) microservice, which handles calls to the Points, User, Role and Maker-Checker atomic microservices. This allows us to easily scale up and down different microservices as required and leads to loose coupling between separate services within our application.

AWS Lambda

AWS Lambda allowed us to separate small focused functions like Logs, Roles and Maker-Checker, which further promotes the benefits mentioned in Microservice Architecture for these small and sporadic-use functions.

Integration Endpoints

Source System	Destination System	Protocol	Format	Communication Mode
User	DNS Server	UDP	*DNS Message format*	Synchronous
Admin Proxy Service (AWS ECS)	Role Service (AWS Lambda) *via private API gateway*	HTTP	JSON	Synchronous
Admin Proxy Service (AWS ECS)	Maker-Checker Service (AWS Lambda) *via private API gateway*	HTTP	JSON	Synchronous
Admin Proxy Service (AWS ECS)	AWS SQS	HTTP	JSON	Synchronous
AWS SQS	Logs Service (AWS Lambda)	HTTP	JSON	Asynchronous
Logs Service (AWS Lambda)	Logs Storage (Amazon S3)	HTTP	JSON	Synchronous
Admin Proxy Service (AWS ECS)	User Service (AWS ECS) *via ELB*	HTTP	JSON	Synchronous
Admin Proxy Service (AWS ECS)	Points Service (AWS ECS) *via ELB*	HTTP	JSON	Synchronous

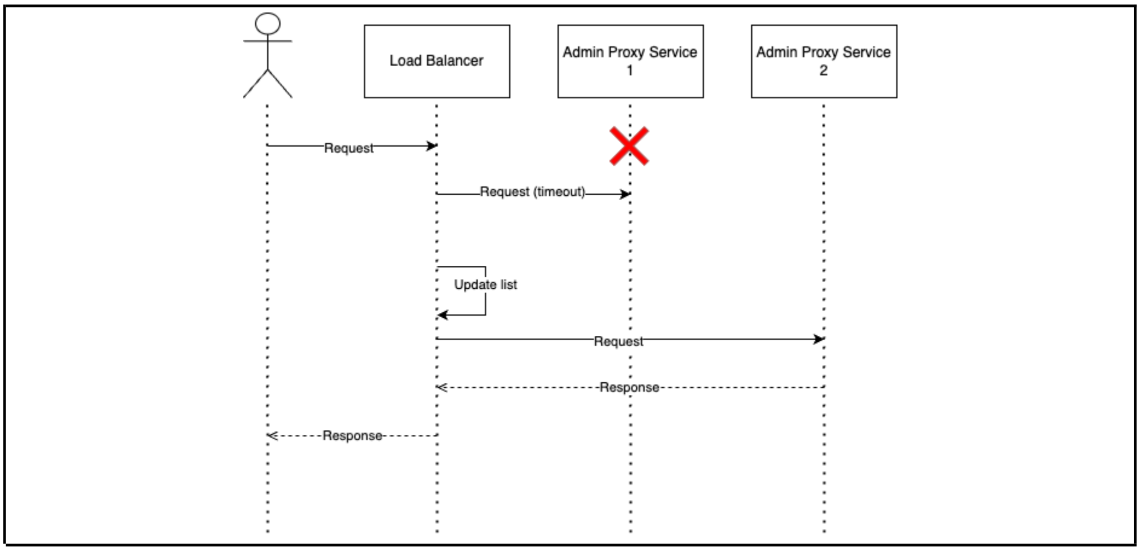
Availability View

Node	**Redundancy**	**Clustering**			**Replication (if applicable)**			
		Node Config	**Failure Detection**	**Failover**	**Repl. Type**	**Session State Storage**	**DB Repl. Config.**	**Repl. Mode**
Frontend UI Service	Horizontal	Active-Active	Health Check	Load balancer	NA			
Admin Proxy Service	Horizontal	Active-Active	Health Check	Load balancer	NA			
User Service	Horizontal	Active-Active	Health Check	Load balancer	NA			

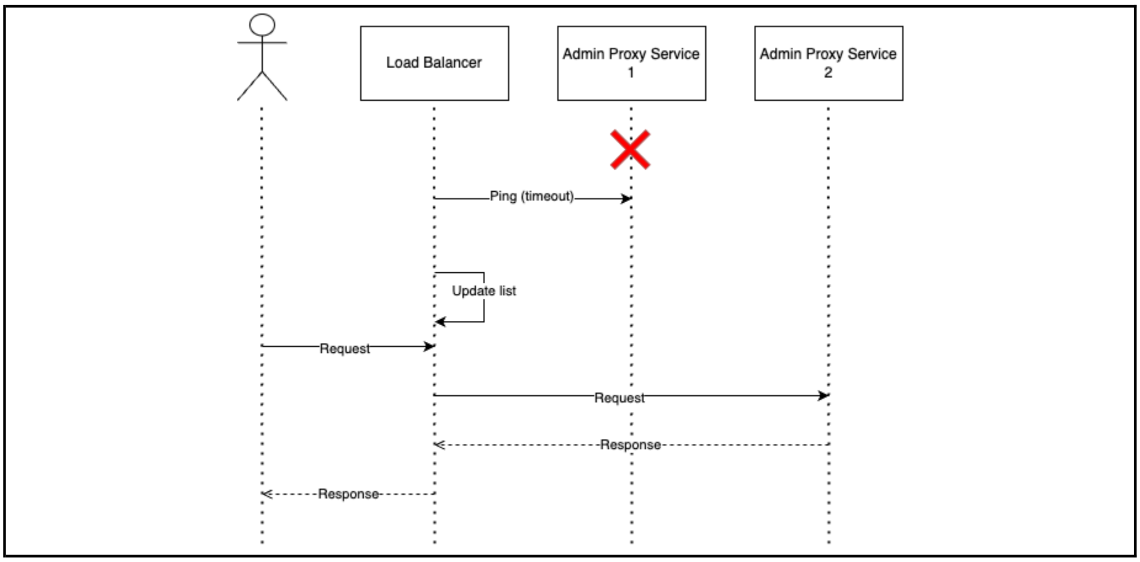
Points Service	Horizontal	Active-Active	Health Check	Load balancer	NA			
DynamoDB Tables	Horizontal	Active-Active	Heartbeat	Managed by AWS	Cross-region	Database	Master Master	Asynchronous

Availability View

Admin Proxy 1 fails (Scenario 1)



Admin Proxy 1 fails (Scenario 2)



Other load balancers are similar

Security View

No	Asset/Asset	Potential Threat/Vulnerability pair	Possible Mitigation Controls
1	Domain	DNS cache poisoning	We used AWS Route 53 to handle this threat. Route 53 only operates authoritative DNS servers, and hence it is not vulnerable to a cache poisoning attack.
2	Exposed endpoints	Packet Sniffing	We will issue SSL certificates to both the exposed endpoints for the frontend and backend. This will ensure that sensitive data sent over the network is properly encrypted.
3	Microservices communication	Man-in-the-middle attacks, Eavesdropping, Tampering	As both the frontend and the load balancer for admin proxy are public facing, hence we have implemented HTTPS communication as it uses data encryption to protect against unauthorised access to the transferred information. The importance of HTTPS implementation pushed us to use Application Load Balancer instead. In the future, full HTTPS communication would be implemented between backend microservices in order to authenticate both sender and receiver, and to ensure data integrity in our communications. Additionally, we have implemented a private network with a private subnet to reduce the overall attack surface of our network.
4	Client-facing services	Denial of Service Attacks (DoS)	We have used an Application Load Balancer alongside AWS WAF to implement rate limiting, especially, at the Admin Panel UI and Admin Proxy service.
5	Hashing of Passwords	Misuse of credentials by malicious attackers	In the database, we are salting and hashing the password in the database which makes it harder for malicious attackers to retrieve the password. This makes it computationally infeasible for malicious attackers to use rainbow tables or hash collisions to reverse-engineer the original passwords.
6	Email (and other sensitive data) masking	Sensitive data exposure	We mask all sensitive data when publishing logs. In this project, this only applies to the email, where we mask it such that the full email is not exposed on our terminal logs nor the logs that we publish to S3.
7	ECS-hosted microservices	Container vulnerabilities	We will regularly patch and update container images.
8	Load Balancers	Improper configuration	Improper configuration of load balancers may lead to unintended exposure of microservices. We ensure that a second person checks the configuration whenever a change is made to the load balancer configurations.

For our appendix, we have included images of our AWS settings and configurations for security.

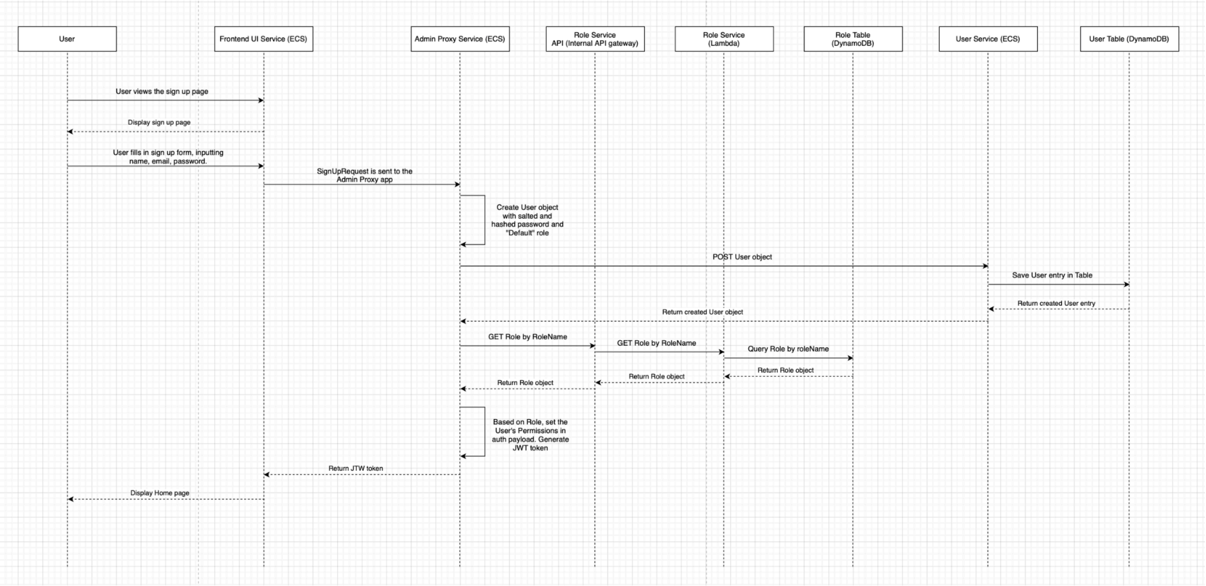
1. AWS Security group settings
2. AWS Web Application Firewall rule settings
3. AWS Certificate Manager
4. AWS Secrets Manager

Performance View

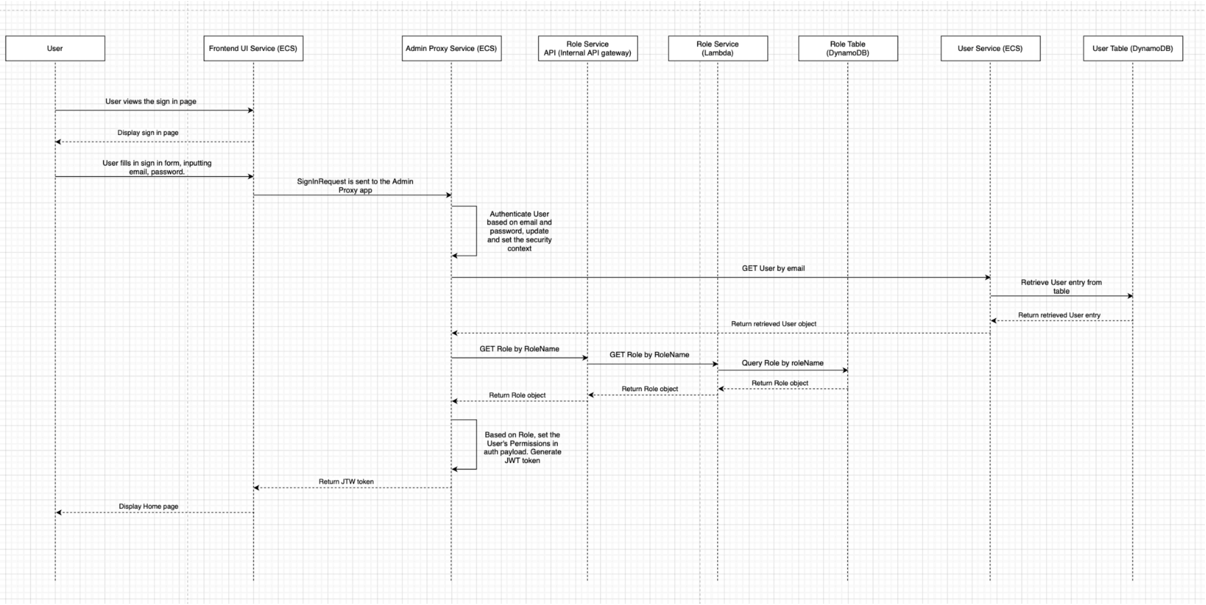
No	Description of the Strategy	Justification
1	Use of NLB over ALB where possible	When the additional features that ALB provides are not needed, we opt to use NLB instead, which decreases the latency in the inter-microservice communication and hence increases the throughput and ability to handle a larger number of concurrent users.
2	Separation of peripheral services	Minor services like logging, as well as other sporadic-use services are separated from our microservices, to allow them to run concurrently. This also isolates the different services which means that if a particular service is running a resource heavy task, the performance of other services will not be impacted.
3	Autoscaling policy: For each Service in the ECS Cluster, when CPUUtilization exceeds 70% for 3 data points within 3 minutes, an alarm is triggered to scale out another instance. We use a minimum number of 1 instance and a maximum of 4.	Scaling out and in accordance to fluctuating demand allows our application to be performant even under sudden high load periods. More testing will be needed to determine if the minimum and maximum number of instances are appropriate.
4	Logging system where logs are being sent to Amazon SQS before being processed in lambda, as compared to sending the logs write requests directly to our logs lambda function.	By using SQS as an intermediate queue, the process of writing logs becomes asynchronous. The Admin Proxy app can push logs to SQS without needing to wait for a response. This will result in a reduced waiting time for our Admin Proxy processes, as they are not blocked by the time consuming process of writing the logs to S3. This is especially important since the writing of logs is done for every single user action, and making it asynchronous will reduce the chance that the Admin Proxy application becomes a performance bottleneck.

Appendix

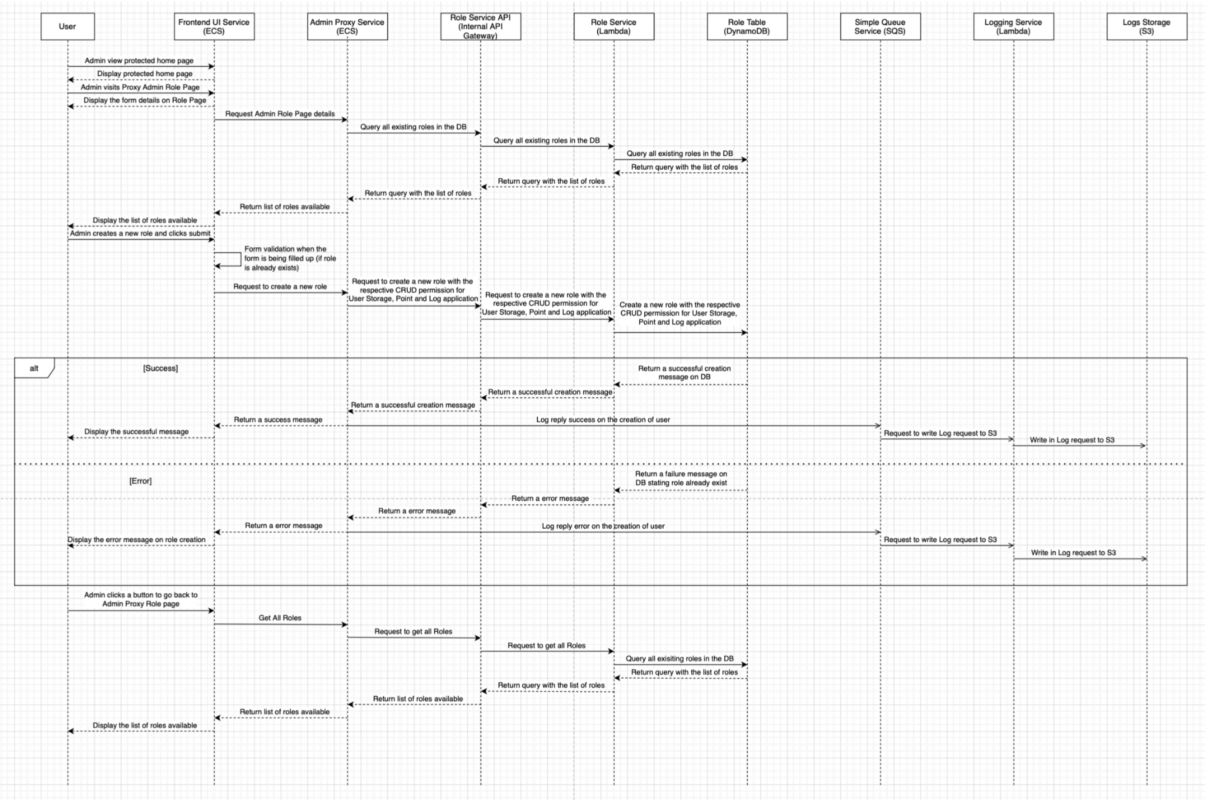
Appendix A : Use Cases


User
Frontend UI Service (ECS)
Admin Proxy Service (ECS)
Role Service API (Internal API gateway)
Role Service (Lambda)
Role Table (DynamoDB)
User Service (ECS)
User Table (DynamoDB)
User views the sign up page
Display sign up page
User fills in sign up form, inputting name, email, password.
SignUpRequest is sent to the Admin Proxy app
Create User object with salted and hashed password and "Default" role
POST User object
Save User entry in Table
Return created User entry
Return created User object
GET Role by RoleName
GET Role by RoleName
Query Role by roleName
Return Role object
Return Role object
Return Role object
Based on Role, set the User's Permissions in auth payload. Generate JWT token
Return JTW token
Display Home page

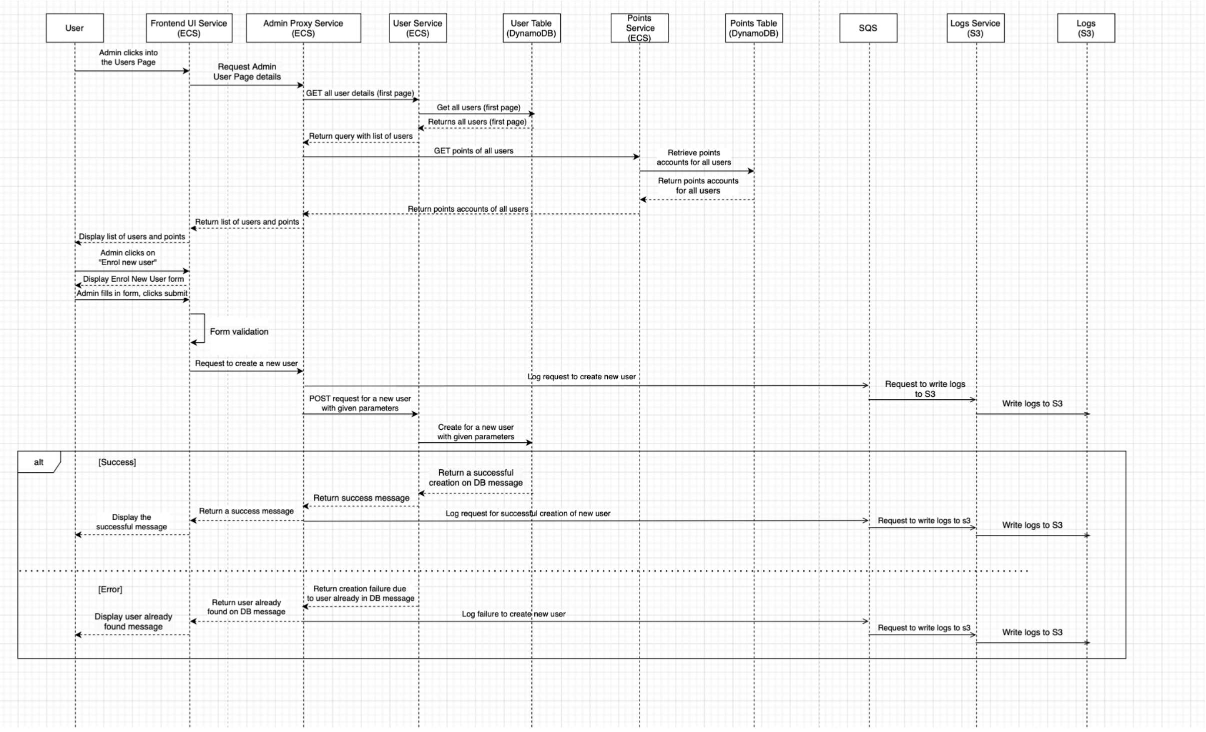
Flow Diagram 1: Authentication sign up


User
Frontend UI Service (ECS)
Admin Proxy Service (ECS)
Role Service API (Internal API gateway)
Role Service (Lambda)
Role Table (DynamoDB)
User Service (ECS)
User Table (DynamoDB)
User views the sign in page
Display sign in page
User fills in sign in form, inputting email, password.
SignInRequest is sent to the Admin Proxy app
Authenticate User based on email and password, update and set the security context
GET User by email
Retrieve User entry from table
Return retrieved User entry
Return retrieved User object
GET Role by RoleName
GET Role by RoleName
Query Role by roleName
Return Role object
Return Role object
Return Role object
Based on Role, set the User's Permissions in auth payload. Generate JWT token
Return JTW token
Display Home page

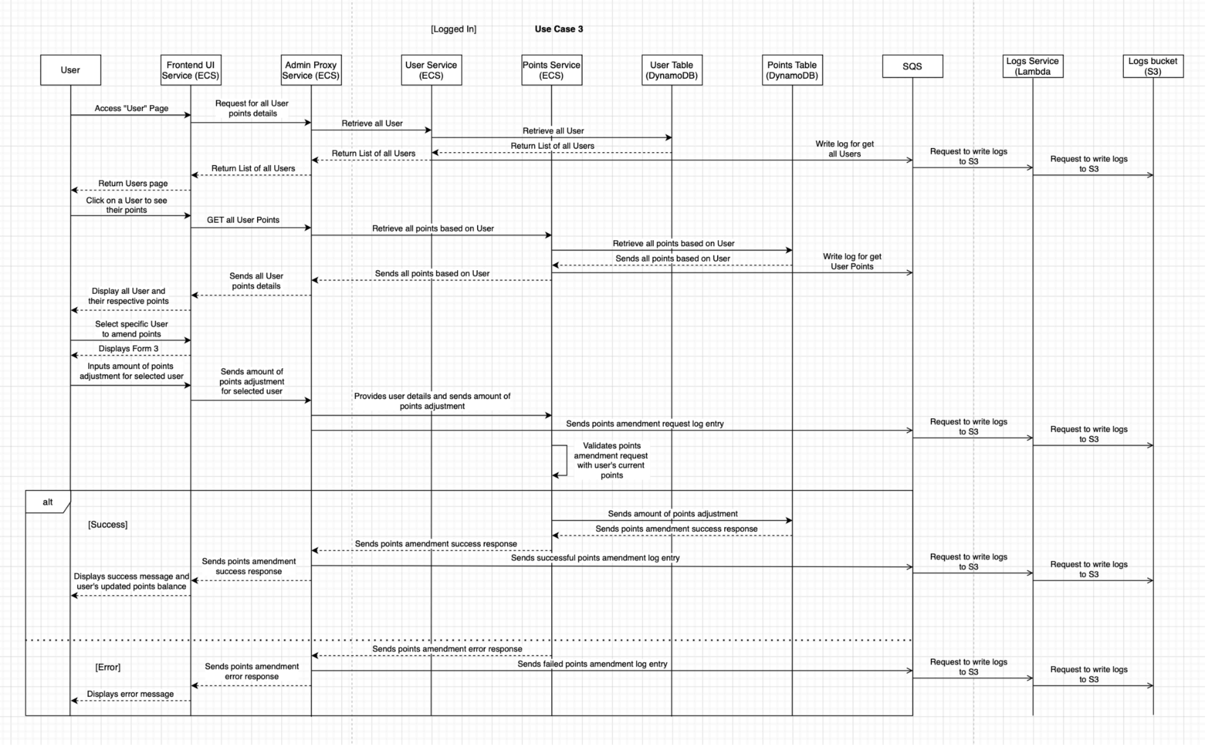
Flow Diagram 2: Authentication sign in


User
Frontend UI Service (ECS)
Admin Proxy Service (ECS)
Role Service API (Internal API Gateway)
Role Service (Lambda)
Role Table (DynamoDB)
Simple Queue Service (SQS)
Logging Service (Lambda)
Logs Storage (S3)
Admin view protected home page
Display protected home page
Admin visits Proxy Admin Role Page
Display the form details on Role Page
Request Admin Role Page details
Query all existing roles in the DB
Query all existing roles in the DB
Query all existing roles in the DB
Return query with the list of roles
Return query with the list of roles
Return query with the list of roles
Return list of roles available
Display the list of roles available
Admin creates a new role and clicks submit
Form validation when the form is being filled up (if role is already exists)
Request to create a new role
Request to create a new role with the respective CRUD permission for User Storage, Point and Log application
Request to create a new role with the respective CRUD permission for User Storage, Point and Log application
Create a new role with the respective CRUD permission for User Storage, Point and Log application
alt
[Success]
Return a successful creation message on DB
Return a successful creation message
Return a successful creation message
Return a success message
Log reply success on the creation of user
Display the successful message
Request to write Log request to S3
Write in Log request to S3
[Error]
Return a failure message on DB stating role already exist
Return a error message
Return a error message
Return a error message
Log reply error on the creation of user
Display the error message on role creation
Request to write Log request to S3
Write in Log request to S3
Admin clicks a button to go back to Admin Proxy Role page
Get All Roles
Request to get all Roles
Request to get all Roles
Query all exisiting roles in the DB
Return query with the list of roles
Return query with the list of roles
Return list of roles available
Return list of roles available
Display the list of roles available

Flow Diagram 3: (Use Case 1) Admin create new role

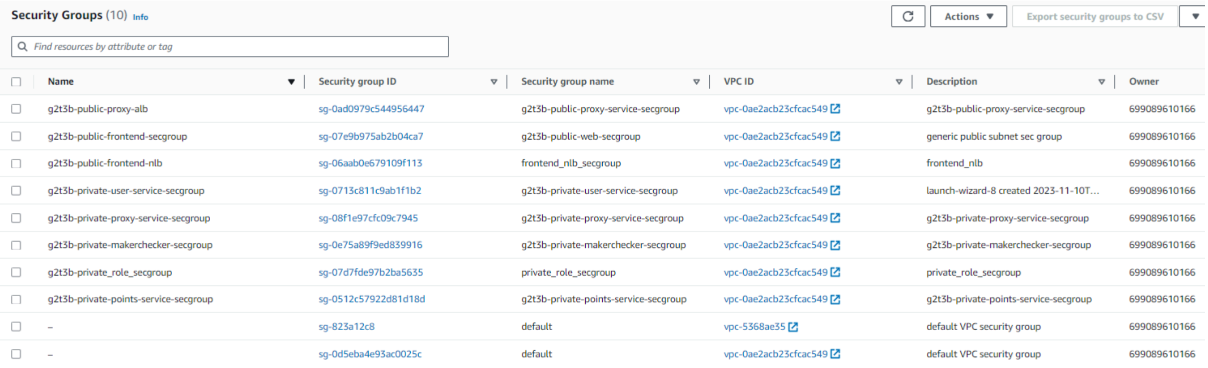

User
Frontend UI Service (ECS)
Admin Proxy Service (ECS)
User Service (ECS)
User Table (DynamoDB)
Points Service (ECS)
Points Table (DynamoDB)
SQS
Logs Service (S3)
Logs (S3)
Admin clicks into the Users Page
Request Admin User Page details
GET all user details (first page)
Get all users (first page)
Returns all users (first page)
Return query with list of users
GET points of all users
Retrieve points accounts for all users
Return points accounts for all users
Return points accounts of all users
Return list of users and points
Display list of users and points
Admin clicks on "Enrol new user"
Display Enrol New User form
Admin fills in form, clicks submit
Form validation
Request to create a new user
Log request to create new user
Request to write logs to S3
Write logs to S3
POST request for a new user with given parameters
Create for a new user with given parameters
alt
[Success]
Return a successful creation on DB message
Return success message
Return a success message
Log request for successful creation of new user
Display the successful message
Request to write logs to s3
Write logs to S3
[Error]
Return creation failure due to user already in DB message
Return user already found on DB message
Log failure to create new user
Display user already found message
Request to write logs to s3
Write logs to S3

Flow Diagram 4: (Use Case 2) Admin enrol new user


[Logged In]
Use Case 3
User
Frontend UI Service (ECS)
Admin Proxy Service (ECS)
User Service (ECS)
Points Service (ECS)
User Table (DynamoDB)
Points Table (DynamoDB)
SQS
Logs Service (Lambda
Logs bucket (S3)
Access "User" Page
Request for all User points details
Retrieve all User
Retrieve all User
Return List of all Users
Return List of all Users
Write log for get all Users
Request to write logs to S3
Request to write logs to S3
Return List of all Users
Return Users page
Click on a User to see their points
GET all User Points
Retrieve all points based on User
Retrieve all points based on User
Sends all points based on User
Write log for get User Points
Sends all points based on User
Sends all User points details
Display all User and their respective points
Select specific User to amend points
Displays Form 3
Inputs amount of points adjustment for selected user
Sends amount of points adjustment for selected user
Provides user details and sends amount of points adjustment
Sends points amendment request log entry
Request to write logs to S3
Request to write logs to S3
Validates points amendment request with user's current points
alt
[Success]
Sends amount of points adjustment
Sends points amendment success response
Sends points amendment success response
Sends successful points amendment log entry
Request to write logs to S3
Request to write logs to S3
Sends points amendment success response
Displays success message and user's updated points balance
Sends points amendment error response
[Error]
Sends failed points amendment log entry
Request to write logs to S3
Request to write logs to S3
Sends points amendment error response
Displays error message

Flow Diagram 5: (Use Case 3) Owner wants to update the points of a user

Appendix B : Security Group settings

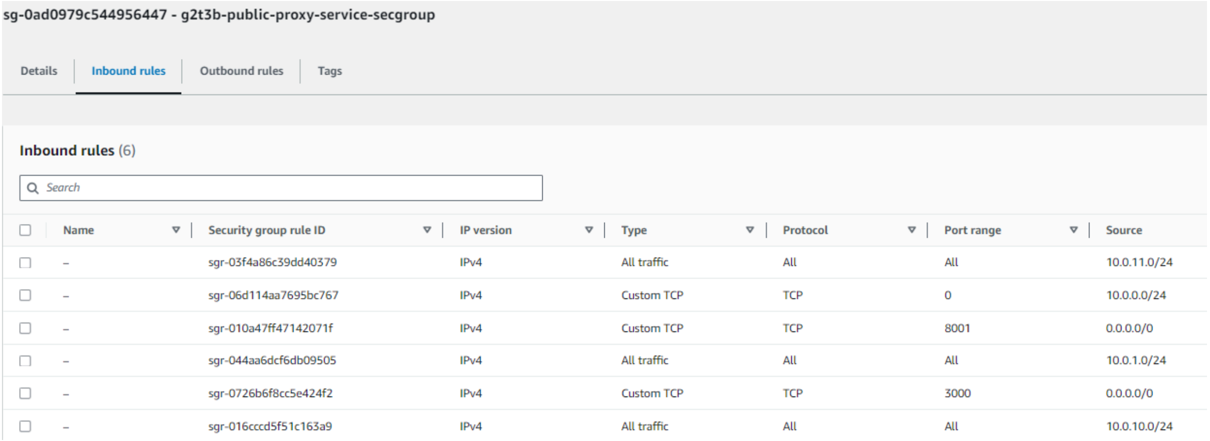


Security Groups (10) Info

Find resources by attribute or tag
Actions
Export security groups to CSV

Name	Security group ID	Security group name	VPC ID	Description	Owner
g2t3b-public-proxy-alb	sg-0ad0979c544956447	g2t3b-public-proxy-service-secgroup	vpc-0ae2acb23cfcac549	g2t3b-public-proxy-service-secgroup	699089610166
g2t3b-public-frontend-secgroup	sg-07e9b975ab2b04ca7	g2t3b-public-web-secgroup	vpc-0ae2acb23cfcac549	generic public subnet sec group	699089610166
g2t3b-public-frontend-nlb	sg-06aab0e679109f113	frontend_nlb_secgroup	vpc-0ae2acb23cfcac549	frontend_nlb	699089610166
g2t3b-private-user-service-secgroup	sg-0713c811c9ab1f1b2	g2t3b-private-user-service-secgroup	vpc-0ae2acb23cfcac549	launch-wizard-8 created 2023-11-10T…	699089610166
g2t3b-private-proxy-service-secgroup	sg-08f1e97cfc09c7945	g2t3b-private-proxy-service-secgroup	vpc-0ae2acb23cfcac549	g2t3b-private-proxy-service-secgroup	699089610166
g2t3b-private-makerchecker-secgroup	sg-0e75a89f9ed839916	g2t3b-private-makerchecker-secgroup	vpc-0ae2acb23cfcac549	g2t3b-private-makerchecker-secgroup	699089610166
g2t3b-private_role_secgroup	sg-07d7fde97b2ba5635	private_role_secgroup	vpc-0ae2acb23cfcac549	private_role_secgroup	699089610166
g2t3b-private-points-service-secgroup	sg-0512c57922d81d18d	g2t3b-private-points-service-secgroup	vpc-0ae2acb23cfcac549	g2t3b-private-points-service-secgroup	699089610166
–	sg-823a12c8	default	vpc-5368ae35	default VPC security group	699089610166
–	sg-0d5eba4e93ac0025c	default	vpc-0ae2acb23cfcac549	default VPC security group	699089610166

Figure : All Security Group settings

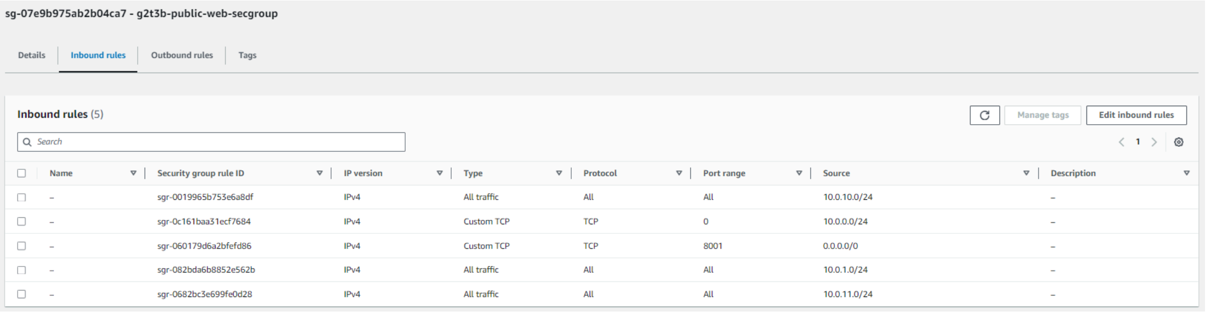


sg-0ad0979c544956447 - g2t3b-public-proxy-service-secgroup

Inbound rules (6)

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source
–	sgr-03f4a86c39dd40379	IPv4	All traffic	All	All	10.0.11.0/24
–	sgr-06d114aa7695bc767	IPv4	Custom TCP	TCP	0	10.0.0.0/24
–	sgr-010a47ff47142071f	IPv4	Custom TCP	TCP	8001	0.0.0.0/0
–	sgr-044aa6dcf6db09505	IPv4	All traffic	All	All	10.0.1.0/24
–	sgr-0726b6f8cc5e424f2	IPv4	Custom TCP	TCP	3000	0.0.0.0/0
–	sgr-016cccd5f51c163a9	IPv4	All traffic	All	All	10.0.10.0/24

Figure: g2t3b-public-proxy-alb Security Group - Inbound

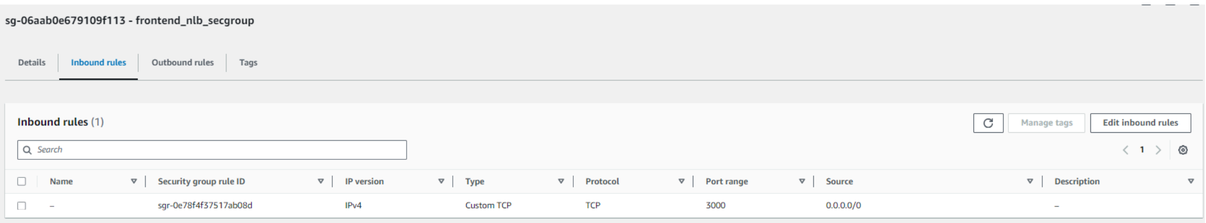


sg-07e9b975ab2b04ca7 - g2t3b-public-web-secgroup

Inbound rules (5)

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
–	sgr-0019965b753e6a8df	IPv4	All traffic	All	All	10.0.10.0/24	–
–	sgr-0c161baa31ecf7684	IPv4	Custom TCP	TCP	0	10.0.0.0/24	–
–	sgr-060179d6a2bfefd86	IPv4	Custom TCP	TCP	8001	0.0.0.0/0	–
–	sgr-082bda6b8852e562b	IPv4	All traffic	All	All	10.0.1.0/24	–
–	sgr-0682bc3e699fe0d28	IPv4	All traffic	All	All	10.0.11.0/24	–

Figure: g2t3b-public-frontend-nlb Security Group - Inbound

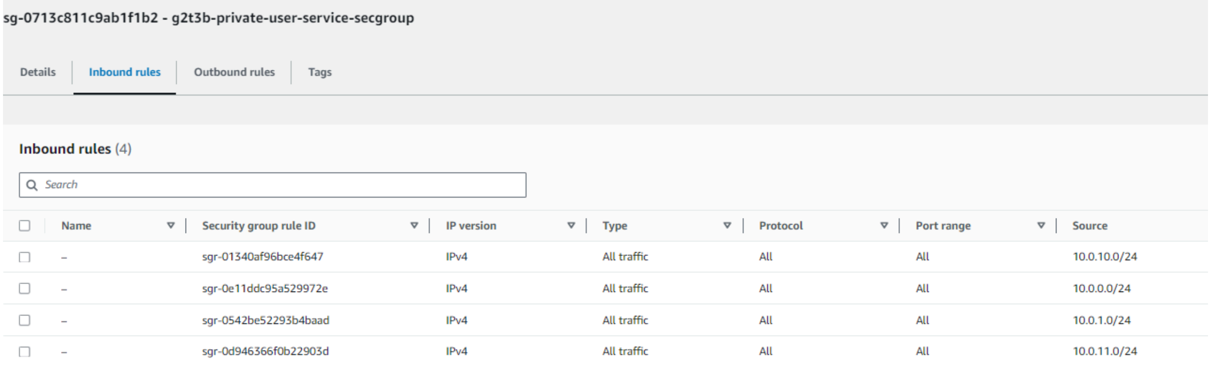


sg-06aab0e679109f113 - frontend_nlb_secgroup

Inbound rules (1)

□	Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
□	–	sgr-0e78f4f37517ab08d	IPv4	Custom TCP	TCP	3000	0.0.0.0/0	–

Figure: g2t3b-private-user-service-secgroup Security Group - Inbound

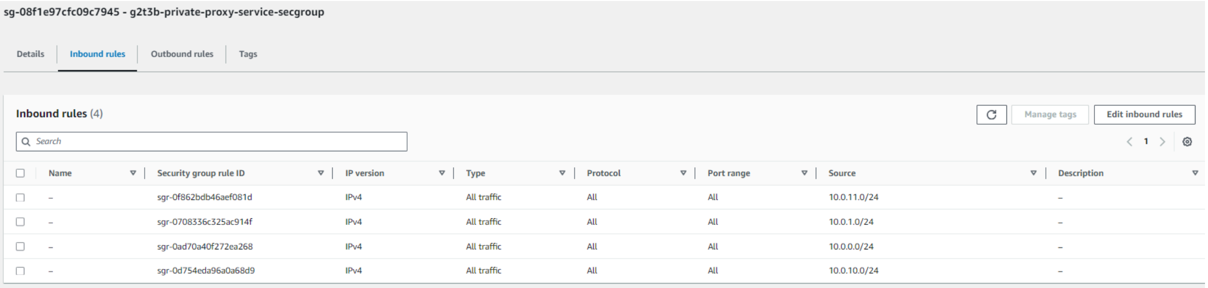


sg-0713c811c9ab1f1b2 - g2t3b-private-user-service-secgroup

Inbound rules (4)

□	Name	Security group rule ID	IP version	Type	Protocol	Port range	Source
□	–	sgr-01340af96bce4f647	IPv4	All traffic	All	All	10.0.10.0/24
□	–	sgr-0e11ddc95a529972e	IPv4	All traffic	All	All	10.0.0.0/24
□	–	sgr-0542be52293b4baad	IPv4	All traffic	All	All	10.0.1.0/24
□	–	sgr-0d946366f0b22903d	IPv4	All traffic	All	All	10.0.11.0/24

Figure:g2t3b-private-proxy-service-secgroup - Inbound

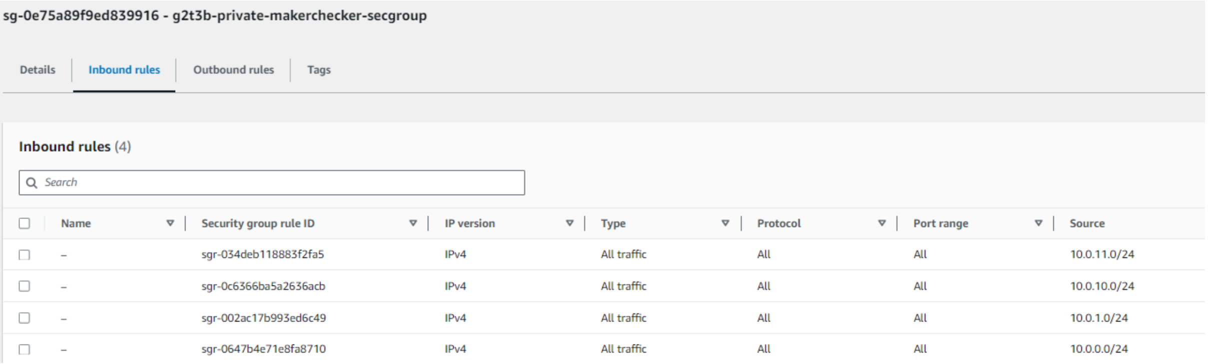


sg-08f1e97cfc09c7945 - g2t3b-private-proxy-service-secgroup

Inbound rules (4)

□	Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
□	–	sgr-0f862bdb46aef081d	IPv4	All traffic	All	All	10.0.11.0/24	–
□	–	sgr-0708336c325ac914f	IPv4	All traffic	All	All	10.0.1.0/24	–
□	–	sgr-0ad70a40f272ea268	IPv4	All traffic	All	All	10.0.0.0/24	–
□	–	sgr-0d754eda96a0a68d9	IPv4	All traffic	All	All	10.0.10.0/24	–

Figure: g2t3b-private-makerchecker-secgroup - Inbound

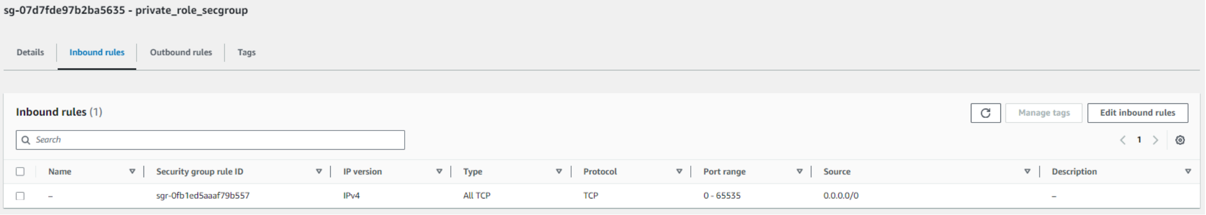


sg-0e75a89f9ed839916 - g2t3b-private-makerchecker-secgroup

Inbound rules (4)

□	Name	Security group rule ID	IP version	Type	Protocol	Port range	Source
□	–	sgr-034deb118883f2fa5	IPv4	All traffic	All	All	10.0.11.0/24
□	–	sgr-0c6366ba5a2636acb	IPv4	All traffic	All	All	10.0.10.0/24
□	–	sgr-002ac17b993ed6c49	IPv4	All traffic	All	All	10.0.1.0/24
□	–	sgr-0647b4e71e8fa8710	IPv4	All traffic	All	All	10.0.0.0/24

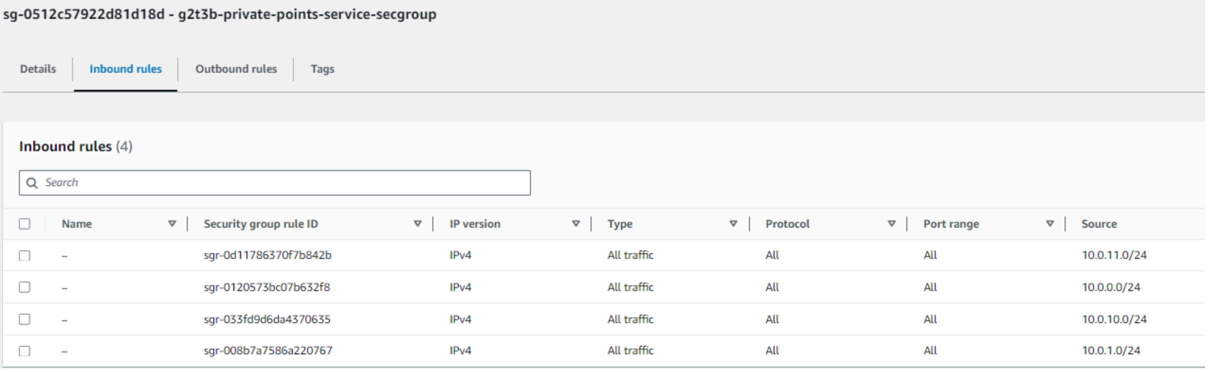
Figure: g2t3b-private_role_secgroup - Inbound


sg-07d7fde97b2ba5635 - private_role_secgroup

Inbound rules (1)

	Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
□	–	sgr-0fb1ed5aaaf79b557	IPv4	All TCP	TCP	0 - 65535	0.0.0.0/0	–

Figure: g2t3b-private_role_secgroup - Inbound

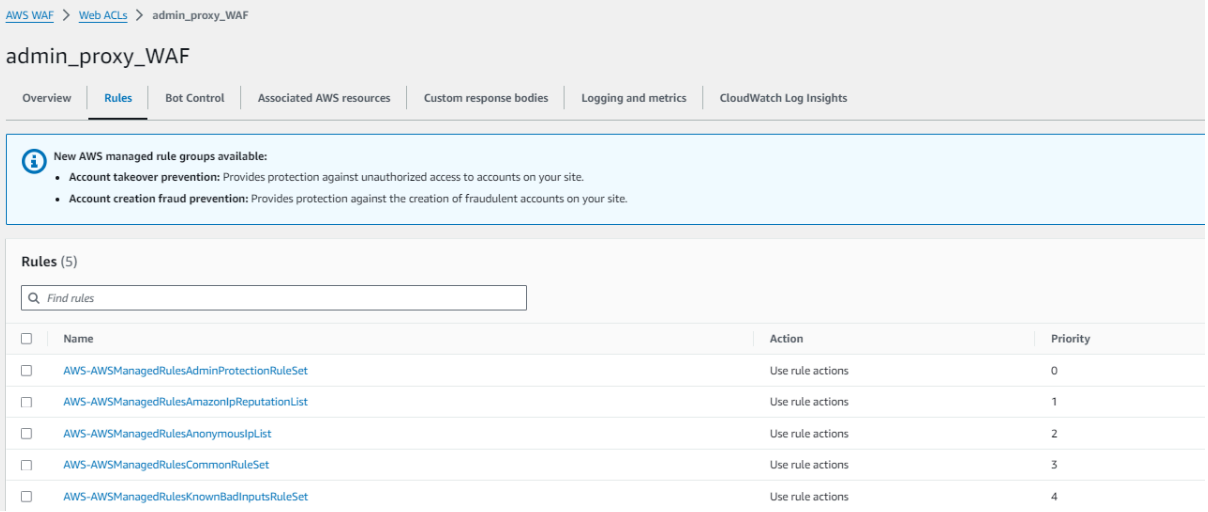

sg-0512c57922d81d18d - g2t3b-private-points-service-secgroup

Inbound rules (4)

	Name	Security group rule ID	IP version	Type	Protocol	Port range	Source
□	–	sgr-0d11786370f7b842b	IPv4	All traffic	All	All	10.0.11.0/24
□	–	sgr-0120573bc07b632f8	IPv4	All traffic	All	All	10.0.0.0/24
□	–	sgr-033fd9d6da4370635	IPv4	All traffic	All	All	10.0.10.0/24
□	–	sgr-008b7a7586a220767	IPv4	All traffic	All	All	10.0.1.0/24

Figure: g2t3b-private-points-service-secgroup - Inbound

Appendix C : Web Application Firewall rules settings on admin proxy



AWS WAF > Web ACLs > admin_proxy_WAF

admin_proxy_WAF

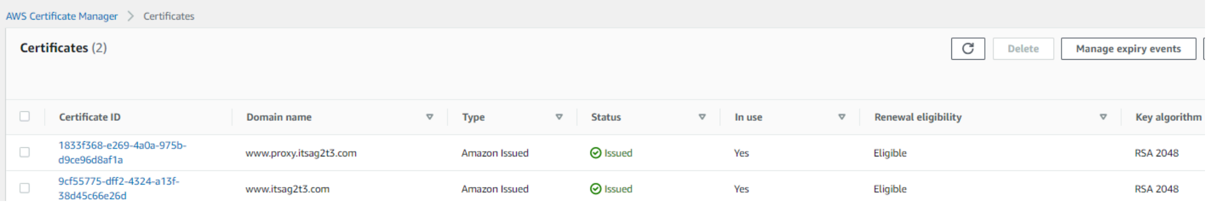
New AWS managed rule groups available:

- **Account takeover prevention:** Provides protection against unauthorized access to accounts on your site.
- **Account creation fraud prevention:** Provides protection against the creation of fraudulent accounts on your site.

Rules (5)

□	Name	Action	Priority
□	AWS-AWSManagedRulesAdminProtectionRuleSet	Use rule actions	0
□	AWS-AWSManagedRulesAmazonIpReputationList	Use rule actions	1
□	AWS-AWSManagedRulesAnonymousIpList	Use rule actions	2
□	AWS-AWSManagedRulesCommonRuleSet	Use rule actions	3
□	AWS-AWSManagedRulesKnownBadInputsRuleSet	Use rule actions	4

Appendix D : AWS Certificate settings for Frontend (itsag2t3.com) and Admin Proxy (www.proxy.itsag2t3.com)

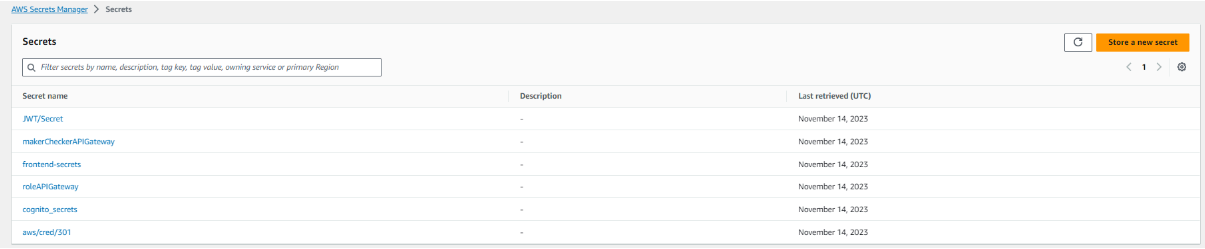


AWS Certificate Manager > Certificates

Certificates (2)

□	Certificate ID	Domain name	Type	Status	In use	Renewal eligibility	Key algorithm
□	1833f368-e269-4a0a-975b-d9ce96d8af1a	www.proxy.itsag2t3.com	Amazon Issued	✓ Issued	Yes	Eligible	RSA 2048
□	9cf55775-dff2-4324-a13f-38d45c66e26d	www.itsag2t3.com	Amazon Issued	✓ Issued	Yes	Eligible	RSA 2048

Appendix E : AWS Secrets Manager



AWS Secrets Manager > Secrets

Secrets

Store a new secret
Filter secrets by name, description, tag key, tag value, owning service or primary Region
1

Secret name	Description	Last retrieved (UTC)
JWT/Secret	-	November 14, 2023
makerCheckerAPIGateway	-	November 14, 2023
frontend-secrets	-	November 14, 2023
roleAPIGateway	-	November 14, 2023
cognito_secrets	-	November 14, 2023
aws/cred/301	-	November 14, 2023

Appendix F: Budget Assumptions [AWS Calculator]

Items	Assumptions
General No. Authenticated API calls	**Peak Traffic:** 100 requests per second throughout office hours (08:00 - 18:00) 100 * 60^2 seconds/hour = 360,000 requests/hour **Off-Peak Traffic:** 5 requests per second throughout non-office hours (18:00 - 08:00) 5 * 60^2 seconds/hour = 18,000 requests/hour

	For simplicity, if we consider the above office hours, it would be 3.6 million requests during the peak hours and 252,000 requests during the off-peak hours. **Total Requests = 3,852,000 ≈ 4 million authenticated requests/day** **120 million requests/month**
Cognito	5,000 Monthly Active Users
Frontend Web Page	Size: 2MB Outbound Data Transfer: 1TB (Retrieval of Web Page)
ECS	Max 18 instances
Lambda	200,000,000 Requests/month 15ms for each request
Elastic Load Balancer	Average 45 new TCP connections per second

Appendix G: Admin UI Views

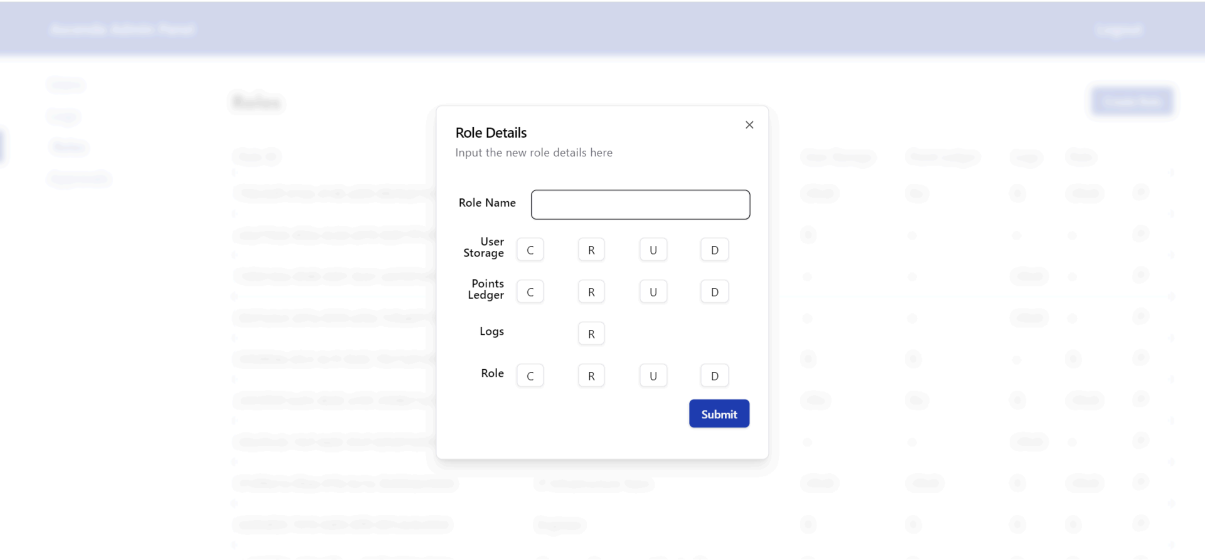
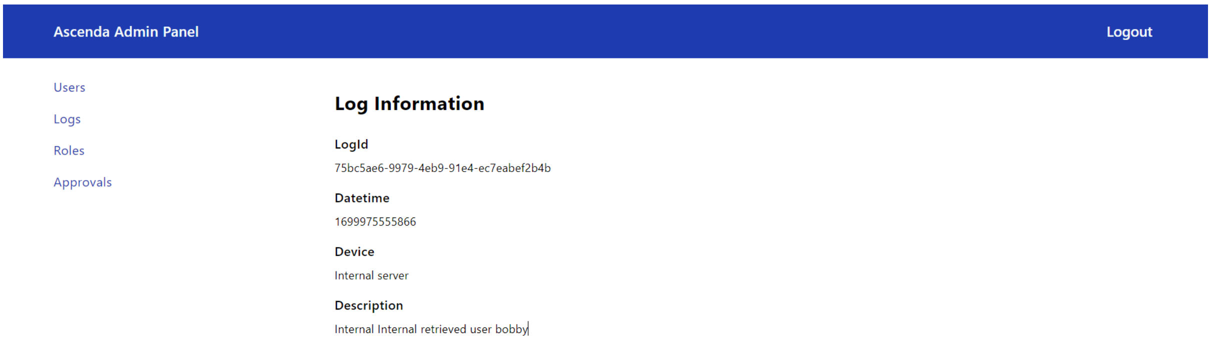


Figure: Creation of New Role



Log Information

LogId

75bc5ae6-9979-4eb9-91e4-ec7eabef2b4b

Datetime

1699975555866

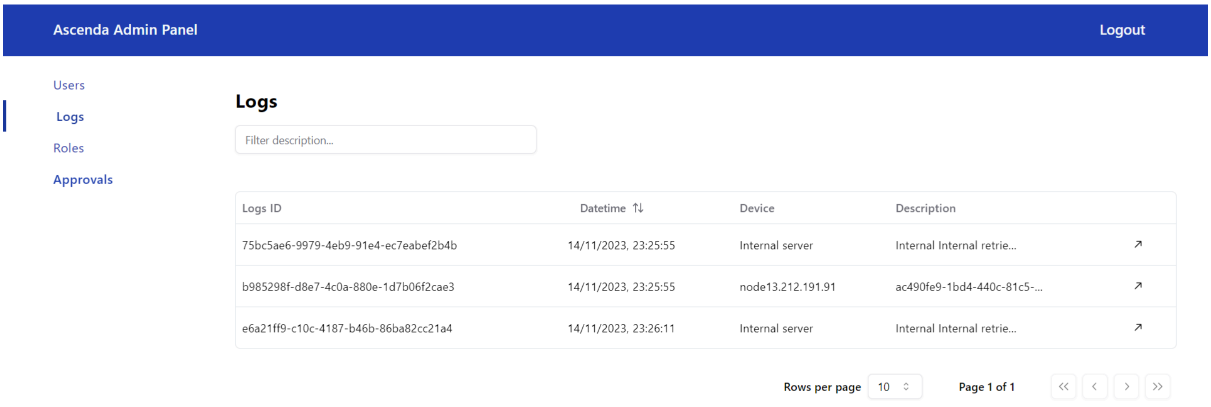
Device

Internal server

Description

Internal Internal retrieved user bobby

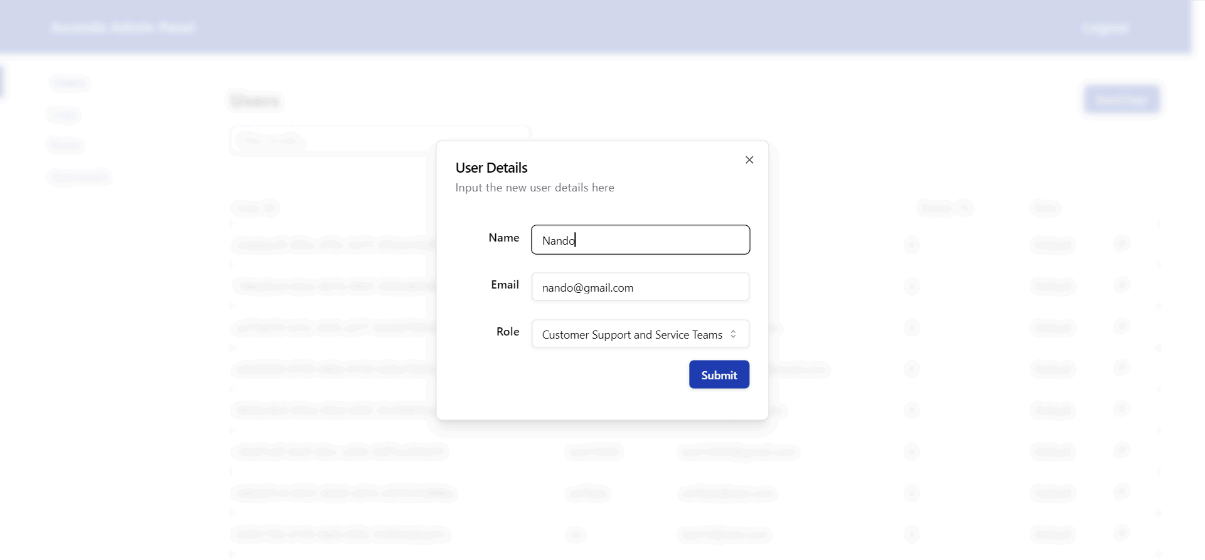
Figure: Logs Information



Logs

Logs ID	Datetime ⇅	Device	Description	
75bc5ae6-9979-4eb9-91e4-ec7eabef2b4b	14/11/2023, 23:25:55	Internal server	Internal Internal retrie...	↗
b985298f-d8e7-4c0a-880e-1d7b06f2cae3	14/11/2023, 23:25:55	node13.212.191.91	ac490fe9-1bd4-440c-81c5-...	↗
e6a21ff9-c10c-4187-b46b-86ba82cc21a4	14/11/2023, 23:26:11	Internal server	Internal Internal retrie...	↗

Figure: Logs Overview



User Details

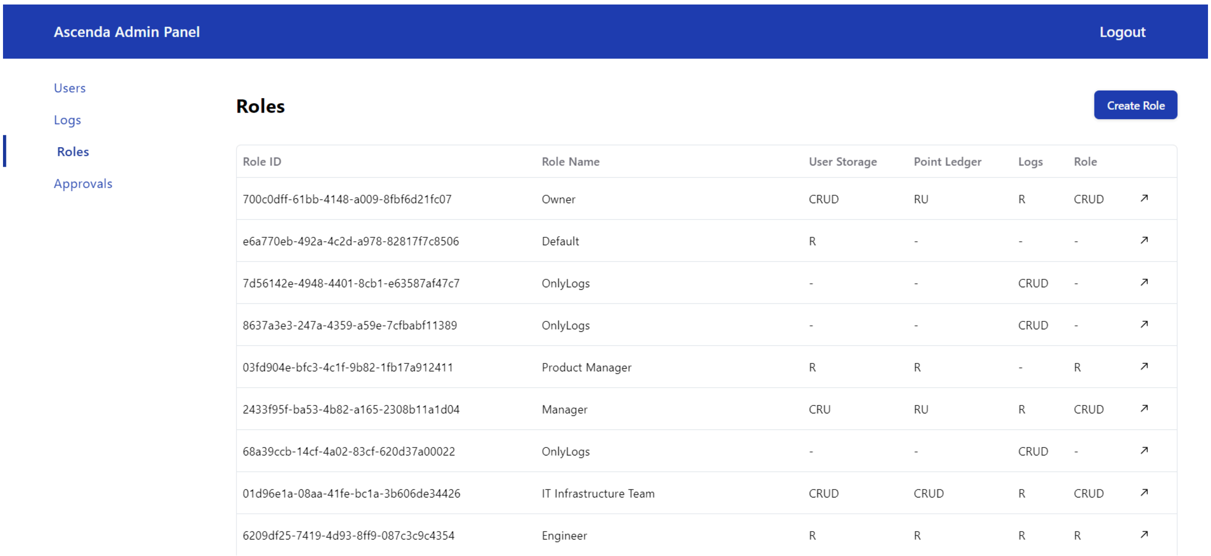
Input the new user details here

Name: Nando

Email: nando@gmail.com

Role: Customer Support and Service Teams

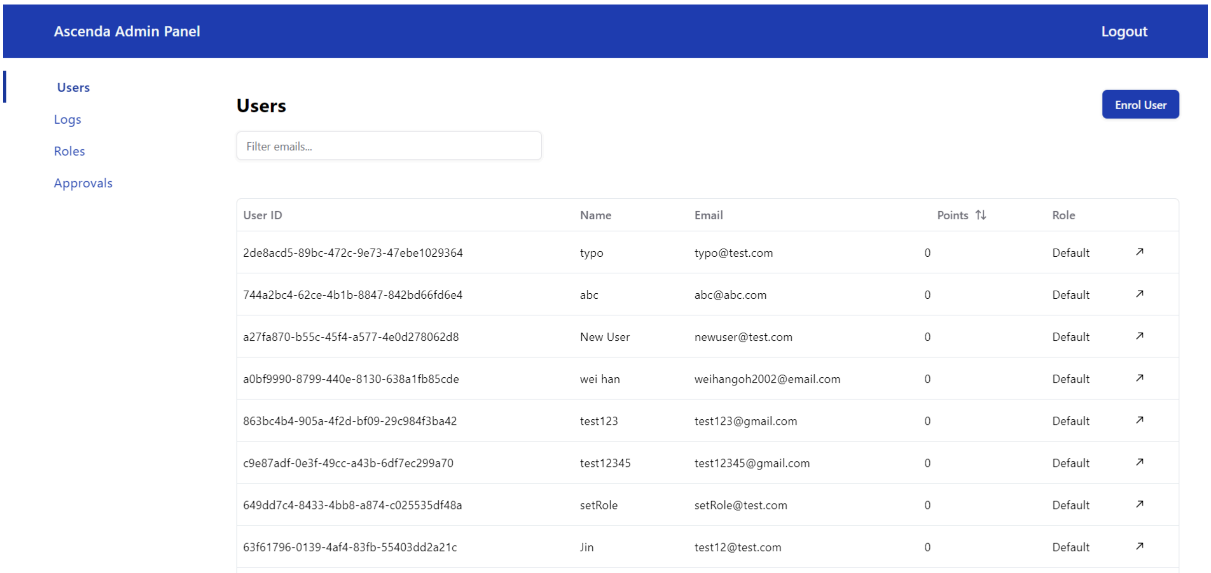
Figure: Creation of New User



Roles

Role ID	Role Name	User Storage	Point Ledger	Logs	Role	
700c0dff-61bb-4148-a009-8fbf6d21fc07	Owner	CRUD	RU	R	CRUD	↗
e6a770eb-492a-4c2d-a978-82817f7c8506	Default	R	-	-	-	↗
7d56142e-4948-4401-8cb1-e63587af47c7	OnlyLogs	-	-	CRUD	-	↗
8637a3e3-247a-4359-a59e-7cfbabf11389	OnlyLogs	-	-	CRUD	-	↗
03fd904e-bfc3-4c1f-9b82-1fb17a912411	Product Manager	R	R	-	R	↗
2433f95f-ba53-4b82-a165-2308b11a1d04	Manager	CRU	RU	R	CRUD	↗
68a39ccb-14cf-4a02-83cf-620d37a00022	OnlyLogs	-	-	CRUD	-	↗
01d96e1a-08aa-41fe-bc1a-3b606de34426	IT Infrastructure Team	CRUD	CRUD	R	CRUD	↗
6209df25-7419-4d93-8ff9-087c3c9c4354	Engineer	R	R	R	R	↗

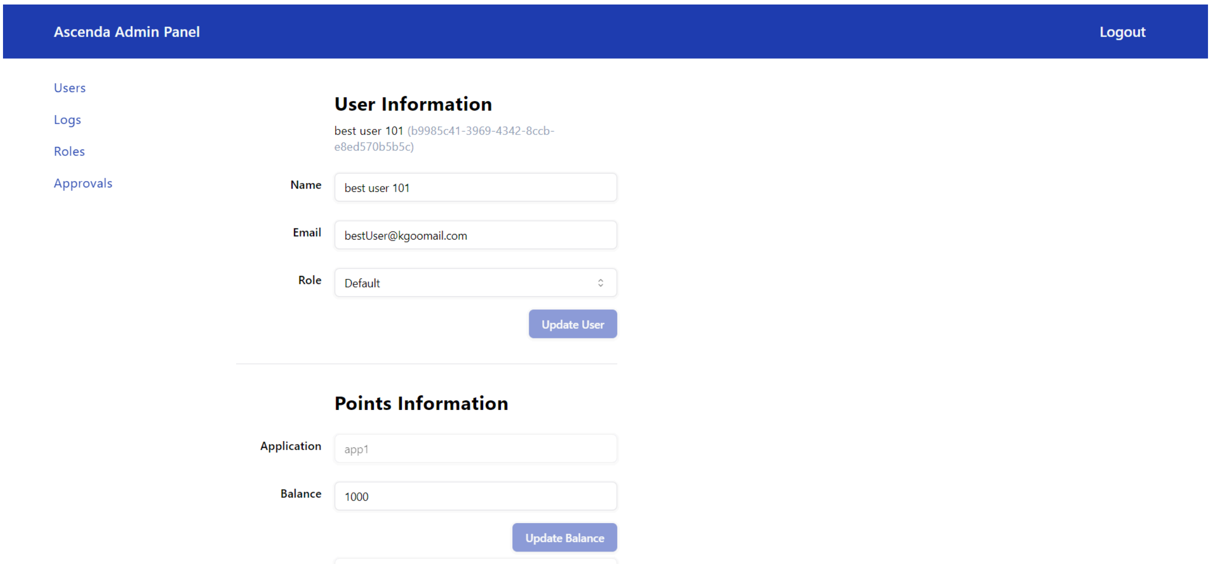
Figure: Roles Overview



Users

User ID	Name	Email	Points ⇅	Role	
2de8acd5-89bc-472c-9e73-47ebe1029364	typo	typo@test.com	0	Default	↗
744a2bc4-62ce-4b1b-8847-842bd66fd6e4	abc	abc@abc.com	0	Default	↗
a27fa870-b55c-45f4-a577-4e0d278062d8	New User	newuser@test.com	0	Default	↗
a0bf9990-8799-440e-8130-638a1fb85cde	wei han	weihangoh2002@email.com	0	Default	↗
863bc4b4-905a-4f2d-bf09-29c984f3ba42	test123	test123@gmail.com	0	Default	↗
c9e87adf-0e3f-49cc-a43b-6df7ec299a70	test12345	test12345@gmail.com	0	Default	↗
649dd7c4-8433-4bb8-a874-c025535df48a	setRole	setRole@test.com	0	Default	↗
63f61796-0139-4af4-83fb-55403dd2a21c	Jin	test12@test.com	0	Default	↗

Figure: Users Overview



User Information

best user 101 (b9985c41-3969-4342-8ccb-e8ed570b5b5c)

Name: best user 101

Email: bestUser@kgoomail.com

Role: Default

Points Information

Application: app1

Balance: 1000

Figure: Update User details and Points details