



SMU

SINGAPORE MANAGEMENT
UNIVERSITY

CS301 IT Solution Architecture

AY 2022/2023 Semester 1

Final Report

Team: G2T8

Github Team Name: project-2022-23t1-g2-t8

Instructors:

Dr. Ouh Eng Lieh

Team 8	Student ID
Seah Pei Ming	01405085
Darien Tan Shi Feng	01369598
Lee Shuoan	01391584
Ng Jun Hng Aloysius	01412905
Royston Lek Chun Keat	01415111
Myo Min Tun	01374353

Stakeholders

Stakeholder	Stakeholder Description	Permissions
Customers	Customers are the bank's end users using the rewards platform. Customers will have to be enrolled and have their accounts verified via 2FA. To access the platform, they will then need to be authenticated	<ul style="list-style-type: none"> View/Update user profile and details
Administrator	Administrators have visibility over the list of users. Admins with write and delete permissions are able to manage user roles and permissions to prevent any unauthorised access to PII data	Read-only roles <ul style="list-style-type: none"> View user account information Write/Delete roles: <ul style="list-style-type: none"> View/Update/Delete user roles and permissions
Banks	Banks manage customers' accounts and their access to our services. The bank's backend services will need to have access to our Authorization server's JWKS endpoint to retrieve the public key	<ul style="list-style-type: none"> GET public key from authorisation server

Key Use Cases

Use Case Title - Enrollment	
Use case ID	1
Description	Customers are required to register an account in Ascenda and the user must have an existing account with Ascenda's partner to access the rewards platform. In addition, 2FA will be used to verify account ownership
Actors	Customers, Banks
Main Flow of events	<ol style="list-style-type: none"> Users register an account in Ascenda's website by keying in their particulars: First Name, Last Name, Email, Birthdate, and Password Ascenda's website will validate if the user has an account with the partner in the database by cross checking the database with user's input Ascenda will verify ownership of user's account through 2FA
Alternative Flow of events	<ol style="list-style-type: none"> If enrolment fails due to failed validation, the client will throw an error If 2FA verification fails, the client will throw an error
Pre-conditions	User accounts from Ascenda's partners from the daily sync which will

	be seeded into AWS Relational Database (RDS)
Post-conditions	Customers will be able to log into their accounts subsequently and will not be required to enrol again

Use Case Title - User Authentication	
---	--

Use case ID	2
--------------------	---

Description	Customers have a choice to either log in to Ascenda's platform through hosted login or Bank SSO. It is important that they are authenticated before they can access any of Ascenda's services
--------------------	---

Actors	Customers, Banks
---------------	------------------

Main Flow of events	<p>User logs in via hosted login</p> <ol style="list-style-type: none"> 1. User clicks on login 2. Client send authorization code request 3. Authorization server redirects user to login prompt 4. User authenticate and consent 5. Authorization server gives authorization code 6. Client sends authorization code along with client secret and id to authorization server 7. Authorization server returns access token which is a signed and encrypted JWT and a refresh token <p>User logs in via Bank SSO</p> <ol style="list-style-type: none"> 1. User clicks on bank SSO to login 2. Redirect user to bank log in 3. Enter user credentials and Bank SSO validates 4. Authenticate and consent 5. Bank SSO grants access 6. Authorization app makes POST request to Bank SSO for access token 7. Bank SSO creates access token 8. Authorization app checks if access token is valid and has not expired 9. GET request to Bank SSO to retrieve user info 10. Bank SSO returns user info 11. User views user profile
----------------------------	--

Alternative Flow of events	<ol style="list-style-type: none"> 1. If log in details are invalid (Invalid email or wrong password), an error will be shown to the user 2. If user fails to consent, frontend client will redirect back to login
-----------------------------------	--

Pre-conditions	User needs to be enrolled
-----------------------	---------------------------

Post-conditions	For hosted login, if refresh token does not expire, user will be automatically logged in to their user profile page as refresh token will
------------------------	---

	help issue a new access token when the current access token expires
--	---

Use Case Title - View Access Control Management Page	
Use case ID	3
Description	Due to the differences in access rights, it is important that the platform will have different views according to the permission rights of the admin logging in. Upon logging in, admins with read-only roles will not be able to perform write and delete operations while admins with these privileges will be allowed to
Actors	Administrators
Main Flow of events	<u>Read-only role</u> <ol style="list-style-type: none"> 1. User logs in via hosted login or Bank SSO 2. Authorization server authenticates user login and returns client with an access token 3. Client checks access token role and if role is an admin, client sends GET request for users from bank's backend service 4. Bank's backend service requests public key from the authorization server's JWKS endpoint 5. Access token is verified with the public key 6. Users are returned to the client and admin control panel is display <u>Write / Delete role</u> <ol style="list-style-type: none"> 7. Admin edits user and submits 8. Client sends GET request to the bank's backend service to update user 9. Bank's backend service requests public key from the authorization server's JWKS endpoint 10. Access token is verified with the public key 11. Bank's backend service updates the user 12. Update success is returned to the client and success message is then displayed to the user
Alternative Flow of events	<ol style="list-style-type: none"> 1. When user or administrator keys in wrong user credentials, frontend will throw an error
Pre-conditions	User needs to be enrolled and successfully authenticated
Post-conditions	View of the control panel is dependent on the privileges of the user

Proposed Budgets

Production Budget

Hardware /	Description	Cost (monthly)
------------	-------------	----------------

Software / Service		
AWS S3	We will be using AWS S3 to host the static frontend web page, allowing low cost, high availability and easily to be managed.	First 50TB/Month - \$0.023 per GB Next 450 TB / Month - \$0.022 per GB Over 500 TB / Month - \$0.021 per GB Assume usage of typical static website hosted in S3 in general by AWS users to be \$3 (Maximum) Total cost = \$3
Compute - AWS EC2 Instances for web & authorization server with Auto Scaling	We will be using 2 EC2 instances as authorization servers per AZ. These instances will be under an Auto Scaling group with a minimum capacity of 2 as there are 2 availability zones. Since customers/banks may be using the services at any time of the services at any time of the day, it's crucial to have a server up all the time	2 x t4g.small, 2 Vepu, 2GiB, NURI Total cost = \$26.76 (AWS pricing calculator)
Application Load Balancer	To ensure high availability, an application load balancer is used to distribute traffic across multiple EC2 instances, in multiple AZs	Assuming 720 Application Load Balancer-hours Total cost = \$52.79
Database - RDS Cluster	RDS is used to store our user data for hosted login as the attributes will not change and hence vertical scaling is preferred as to accommodate store more users using in the future	Assuming that the average amount of data in one month is GB AZ, 3 RDS t4g.small instance 10GB General Purpose SSD storage 33.25 per GB-month Total cost = \$332.50
AWS Secrets Manager and AWS Key Management Service (KMS)	AWS Secrets Manager stores credentials and has integration with AWS KMS to store encryption keys. AWS Secrets manager stores secrets, and transparently decrypts and returns them to you in plaintext. It's designed specially to store application secrets, such as login credentials, that change periodically. It	Secrets manager: \$0.05 per 10,000 API calls: \$0.40 per secret KMS: \$0.03 per 10,000 requests Considering 10 million

	encrypts the value with a unique data key which is protected by an AWS KMS key.	requests and API calls + 10 secrets: \$84
API Gateway	API Gateway will act as a proxy and be responsible for routing HTTPS requests to the backend server	Assuming 3 million requests of 12KB with REST API Total cost = \$18.30
AWS Private Link	AWS PrivateLink provides private connectivity between VPCs, AWS services, and on-premises networks, without exposing the traffic to the public internet	Total Cost = \$0.013 per VPC endpoint per AZ (\$/hour) * 4 * 720 = \$37.44
AWS WAF	AWS WAF helps you protect against common web exploits and bots that can affect availability, compromise security, or consume excessive resources.	1 Web ACLs per month = \$7.02 (WAF Web ACLs cost) 1 Rules added per Web ACL + 4 Managed Rule Groups = 5.00 Total billable Rules per Web ACL 1 Web ACLs per month x 5.00 Billable Rules per web ACL per month x \$1.40 = \$7.02 (WAF Rules cost) 1 requests per month x 1000000 multiplier for million x 0.0000006 USD = \$0.84 (WAF Requests cost) \$7.02 + \$7.02 + \$0.84 = \$14.88
Maintenance	Code revision and updates	10 Man hours per month
Other Services	Amazon Cloudfront, Amazon Cloudwatch, AWS WAF, AWS ACM, AWS IAM, Amazon Route 53, AWS SNS (Assuming at most 1 million requests per month)	Free
Total cost	Overall costs	569.67 + 10 Man Hours

Key Architectural Decisions

Architectural Decision - Front End running on S3	
ID	1
Issue	There are multiple ways to deploy the frontend client for our application
Architectural Decision	We chose to deploy our website using S3 hosted static pages instead of using EC2 web application instances
Assumptions	-
Alternatives	AWS EC2 (Together with/Separate from backend)
Justification	<p>Having a frontend together with the backend is not selected due to the tight coupling. Hosting our frontend on another EC2 is not chosen as it is more pricey due to a wastage of resources such as compute power.</p> <p>S3 being a PaaS is more easily maintainable than EC2 as it is an IaaS. There are lesser things that we have to worry about when hosting our frontend on S3 as AWS will manage more things than if we were to host our frontend on EC2.</p>

Architectural Decision - Database	
ID	2
Issue	There are different methods to store our data
Architectural Decision	Using a relational database (RDS, MySQL) over a non-relational database (DynamoDB)
Assumptions	Standard bank user information data formats for each customer of a bank
Alternatives	Non relational database (DynamoDB) and Database in another ec2
Justification	<p>As highly accurate and durable data is required for bank customers, it is crucial that our database complies with ACID and hence, we selected RDS over DynamoDB</p> <p>As our data is not complex, hosting the database in another EC2 instance is not selected due to the high maintenance costs.</p>

Architectural Decision - Monolithic Architecture	
ID	3
Issue	There are several methods to develop our application
Architectural	Creating our application using a monolithic architecture over

Decision	microservices
Assumptions	-
Alternatives	Microservices architecture
Justification	As the authorisation server is inherently stable and scalable sub functions, a monolithic architecture is chosen as it reduces the complexity of development, testing, maintenance and deployment

Architectural Decision - Authorization server running on EC2 instances	
ID	4
Issue	There are several methods to deploy our authorization server
Architectural Decision	Deploying our authorization server on EC2 over AWS Lambda and ECS
Assumptions	-
Alternatives	AWS Lambda, ECS
Justification	<p>EC2 allows flexibility of options, allowing us to configure features to our needs and demands when necessary. This provides long-running tasks since instances are available for different types of requirements with different configurations. In addition, implementation of EC2 Auto Scaling groups can provide an error free process for scalability.</p> <p>Cold startup and be an issue for AWS Lambda, causing latency, hence using EC2 can help to provide better availability.</p> <p>EC2 can function without the need of containerization, offering great flexibility with wide-ranging OS support and hardware configurations.</p>

Development View

Project Management

With the scale of this project and the sizable number of developers on our team, it was especially crucial for our team to work closely together with clearly defined roles.

Members and duties

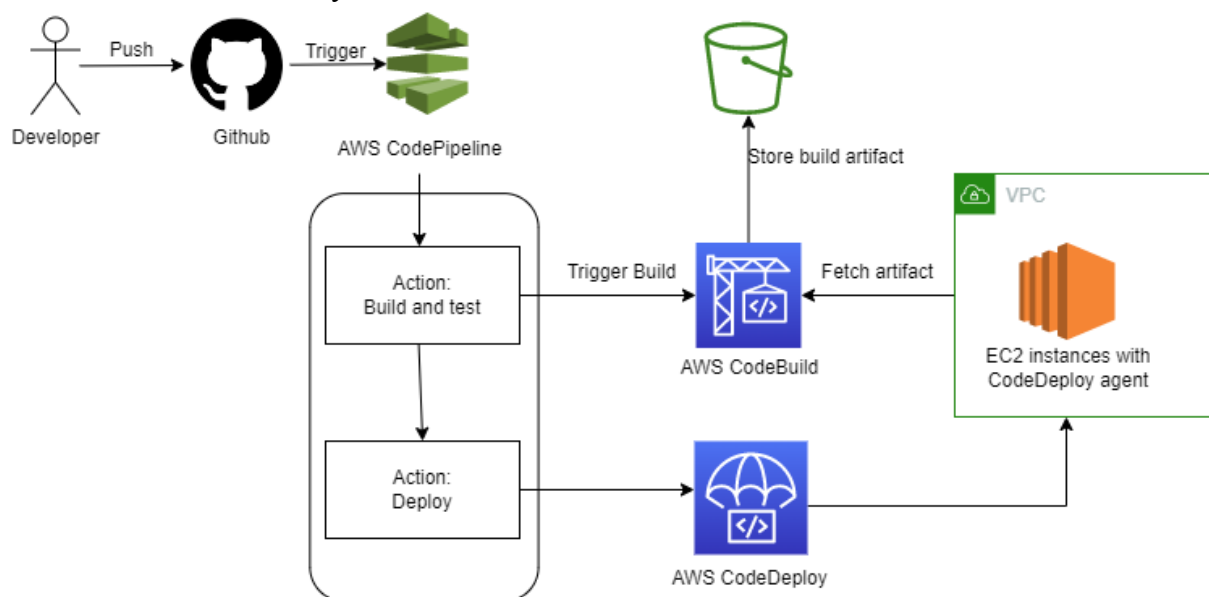
Team Member	Key Responsibilities
Seah Pei Ming	CI-CD and Cloud Architecture
Darien Tan Shi Feng	Frontend development
Lee Shuoan	Frontend development, CI-CD
Ng Jun Hng Aloysius	Backend (Bank SSO) development
Royston Lek Chun Keat	Backend (Hosted Login) development
Myo Min Tun	Backend (Enrolment) development

SCRUM Project Management

With the scrum framework and the usage of Notion platform, we held weekly scrum meetings to ensure that our project was on track. Doing this from the beginning was crucial such that everyone would be updated on the various cloud services that would be used during the development phase and for deployment on the cloud.

Continuous Integration Continuous Deployment (CICD)

For the deployment of our backend into EC2 instances, we made use of AWS CodePipeline, with CodeBuild and CodeDeploy to automatically deploy our changes to our development environment. For deployment of our frontend into S3, we made use of Github Actions to automatically deploy our changes to our S3 bucket. This enabled us to rapidly deploy and test our code for our overall system.



Note: CodeDeploy for backend not fully functional. Passes up till CodeBuild.

Learning Points

This project was certainly not without its challenges. Many concepts and principles encompassed in the project requirements were previously unknown to us, and we had to explore and learn a lot of these concepts through our research. An important part of our development strategy was also to consolidate learning points for our growth as developers! Here are a few brief

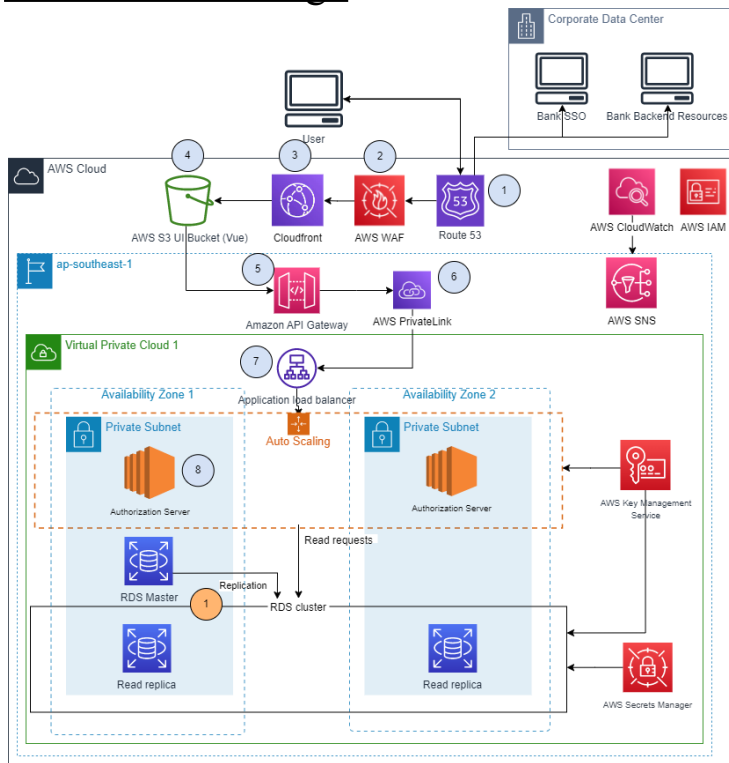
AAR pointers we gathered:

AAR pointers we gathered:

1. Allocate work better to distribute heavier programming work more evenly.
2. Better communication throughout the development process to catch issues early. Fail fast, recover fast
3. Making small incremental changes to our system instead of trying to set up the entire cloud infrastructure at once, making debugging very hard

Solution View (Maintainability)

Architecture Design



- 1 Route 53 DNS helps to route user requests to our internet facing applications
 - 2 Traffic is then routed to AWS WAF which helps to monitor, filter and block any malicious HTTP/S traffic travelling to the web application
 - 3 Cloudfront helps to speed up distribution of our static web content to 400+ points of presence worldwide
 - 4 Amazon S3 for static website hosting
 - 5 API Gateway works as a reverse proxy to direct traffic to our services
 - 6 AWS Privatelink establishes private connectivity between our API gateway and our VPC without exposing data to the internet
 - 7 Application load balancer distributes web traffic across instances in the different availability zones to increase availability and avoid a single point of failure
 - 8 The traffic is then received as a HTTP Request by our web server
- 1 RDS read replicas will be created in both availability zones, enhancing the availability of data

Routing configurations

VPC Network ACL

Rule Number	Type	Protocol	Port Range	Source Address	Allow/Deny
100	All IPv4 traffic	All	All	0.0.0.0/0	Allow
*	All IPv4 traffic	All	All	0.0.0.0/0	Deny

Application Load Balancer (Internal)

Type	Protocol	Source Address	Destination Port
HTTPS	TCP	0.0.0.0/0	443
HTTP	TCP	0.0.0.0/0	80
Custom TCP	TCP	10.0.0.0/16	8080

Authorization Server

Type	Protocol	Source Address	Destination Port
HTTP	TCP	10.0.0.0/16	80
Custom TCP	TCP	10.0.0.0/16	8080
SSH	22	0.0.0.0/0	22

Database

Type	Protocol	Source Address	Destination
Amazon RDS	TCP	10.0.0.0/16	3306

Integration Endpoints

Source System	Destination System	Protocol	Format	Communication Mode
Route 53	AWS WAF	HTTPS	JSON	Synchronous
AWS WAF	AWS CloudFront	HTTPS	JSON	Synchronous
AWS CloudFront	S3	HTTPS	JSON	Synchronous
S3	API Gateway	HTTPS	JSON	Synchronous

API Gateway	VPC Link	HTTPS	JSON	Synchronous
VPC Link	Application Load Balancer	HTTPS	JSON	Synchronous
Application Load Balancer	EC2	HTTP	JSON	Synchronous
EC2	RDS	TCP/IP	JSON	Synchronous

Software Design Principles

Our software design principles and patterns are chosen best suited to the business needs and requirements of the project description.

Singleton

We made use of Singleton design principles, creating one unique object of one class(SSOService) which can be used wherever required. This promotes code reusability and flexibility.

Don't Repeat Yourself (DRY)

To reduce the repetition of patterns and code duplication in favour of abstractions and avoiding redundancy, we made use of the DRY principle. This helps to divide the code into smaller segments, thus calling it whenever it is required.

Builder

SSO feature is based on the builder design principle which helps to simplify creation of objects, constructing the complex objects step by step. This enables us to gain better control over the construction process, improving its security construction and also the readability of code.

SOLID Design Principles

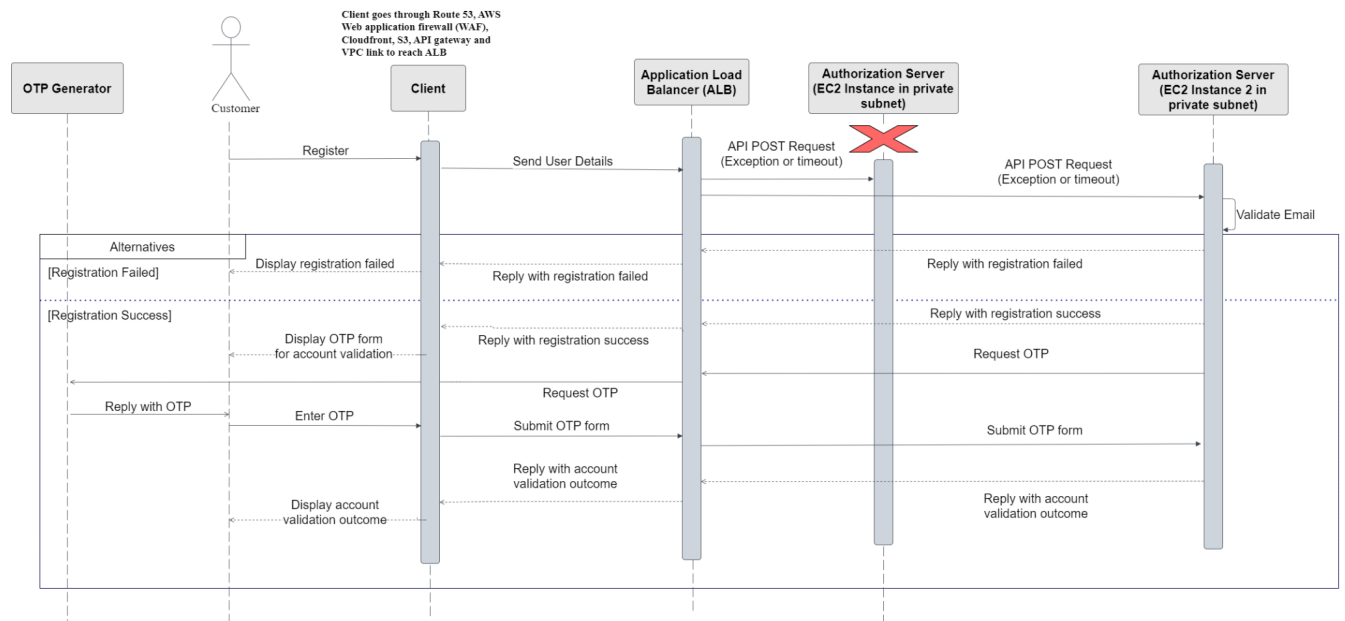
We adhered to SOLID design principles for object-oriented programming, having every class and interface to be documented with well-defined responsibilities, enabling us to identify any critical logical components.

Availability View

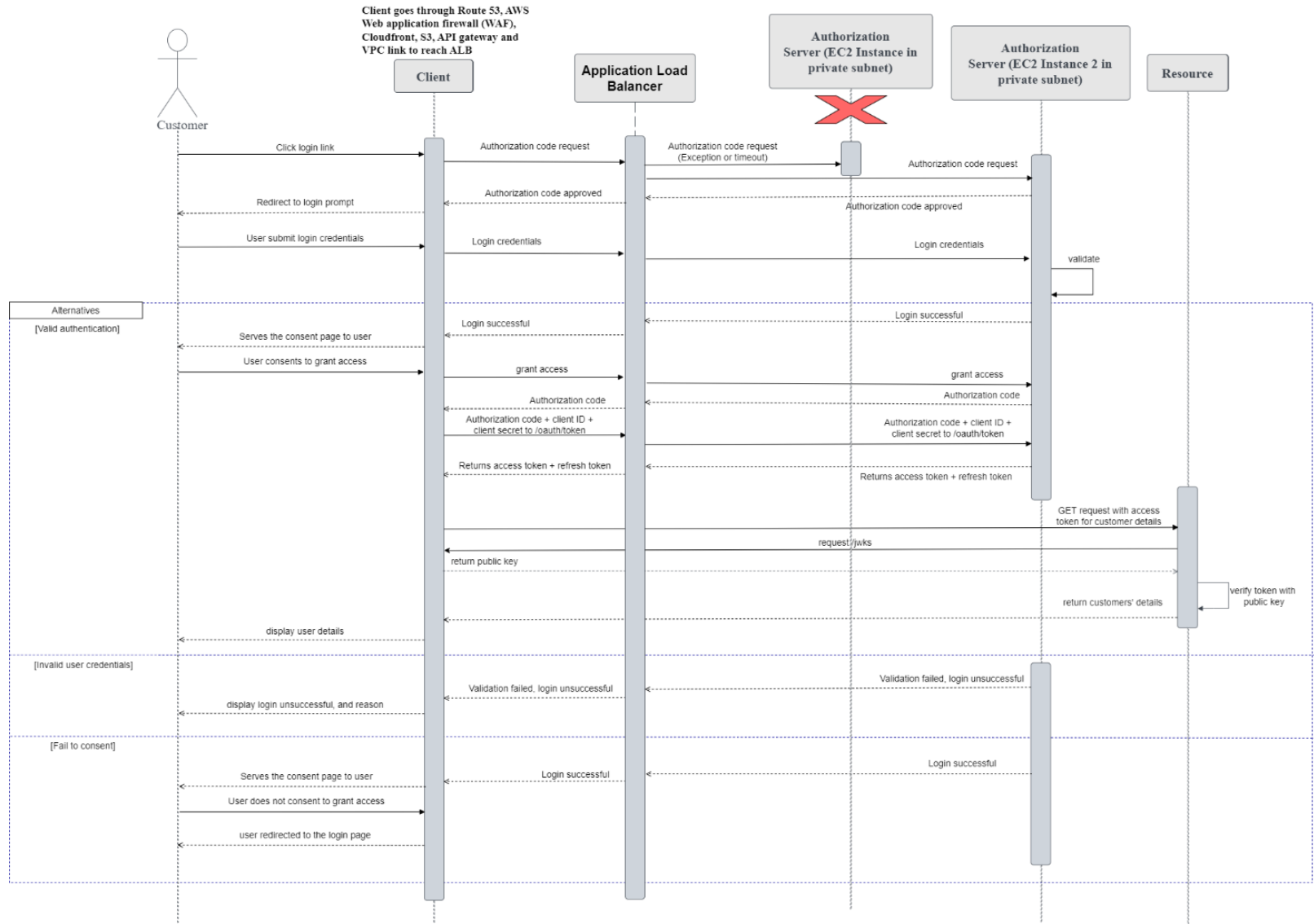
Node	Redundancy	Clustering	Replication
EC2 Instances	Horizontal Scaling	Node Config: Active-Active Failure Detection: Ping Failover: Application Load Balancer	Session State Storage: Client Sessions
Amazon RDS	Horizontal Scaling	Node Config: Active-Passive Failure Detection: Heartbeat Failover: Promotion of database Replica	DB Repl. Config: Master-Slave Repl. Mode: Asynchronous

Sequence Diagram

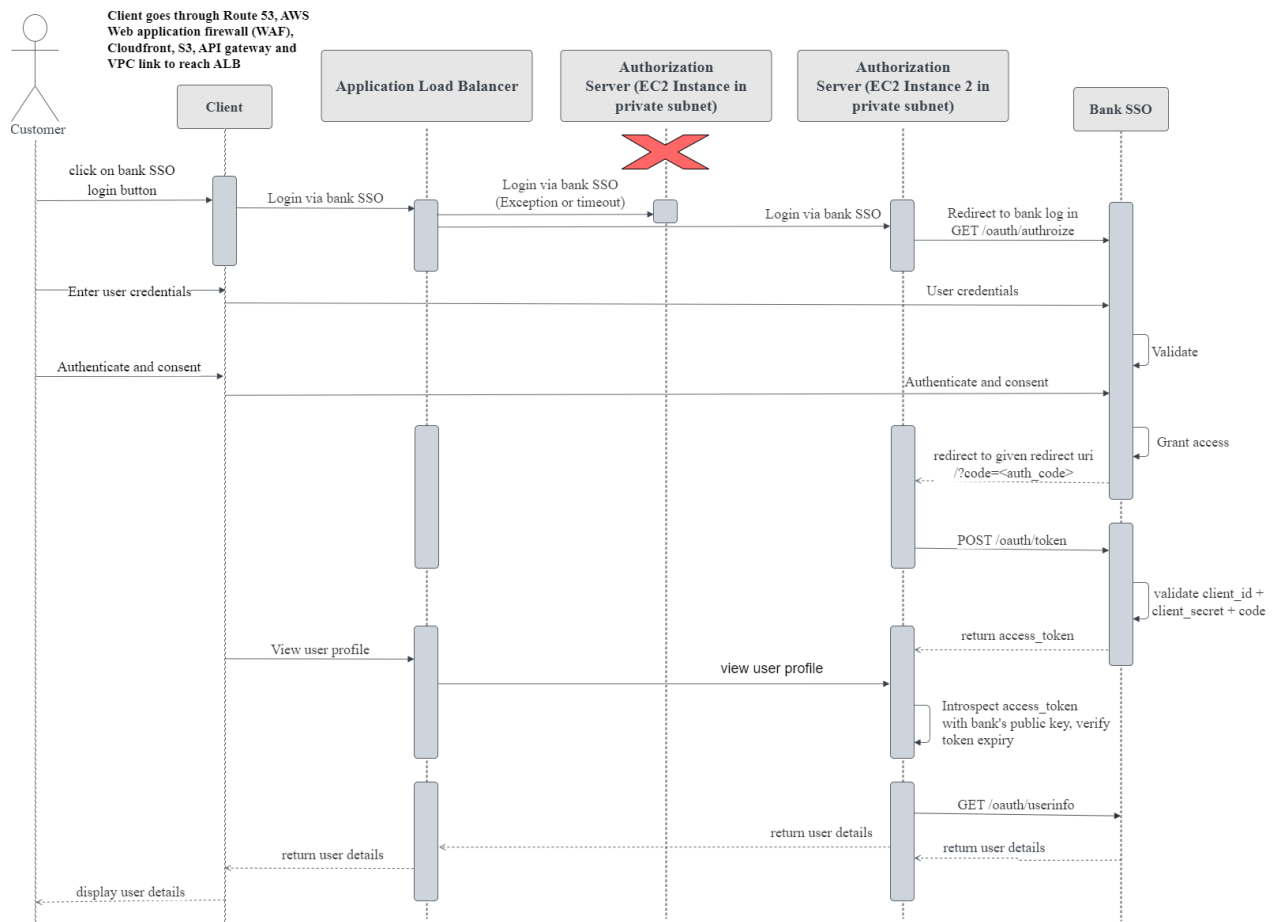
Loss of availability of EC2 Authorization Server during registration



Loss of availability of EC2 Authorization Server during hosted login



Loss of availability of EC2 Authorization Server during bank SSO



Our EC2 Authorization Server consists of hosting all of our functionalities, MFA registration, hosted login, bank SSO and the web server. Upon the loss of availability of the EC2 Authorization Server, the application load balancer will route traffic automatically to another functional EC2 instance. Through the health check, unhealthy instances will be terminated and would automatically replace the non-functional instance.

Security View

No	Asset/Asset	Potential Threat / Vulnerability pair	Possible Mitigation Controls
1	Servers	DDos Attack/Exploit Web server <ul style="list-style-type: none"> • Server outages and monetary loss • Unavailability of websites/servers • CORS attack 	Implementation of AWS WAF
2	Data in transit	MITM Attack/Exploit network between User and Server <ul style="list-style-type: none"> • Hijack user credentials • Post invalid data 	Enable SSL and HTTPS for the entire application
3	Data in database	SQL Injection/ Exploit APIs in Web Server <ul style="list-style-type: none"> • Leak/tamper user password • Change records in user table 	<ol style="list-style-type: none"> 1. Hash user passwords 2. Input validation (Not implemented) 3. Disable public access to database 4. Encrypt data with KMS

Personal Information

Customer information in Amazon RDS is logically segregated so users and customers will not be able to access resources not assigned to them. Amazon RDS encrypts data with the keys that we manage with AWS Key Management Service (KMS). All data including the replicas, backups and snapshots are encrypted at rest. To effectively remove user PII, we will implement a “Delete Account” feature so that users will be able to completely delete their information from their database.

Systems Security

AWS Cloudwatch will track the average CPU utilisation of the EC2 instance. Any anomaly will create a log in AWS Cloudwatch Events which can in turn trigger a message to AWS SNS to send notifications to the relevant stakeholders.

With AWS IAM, we will employ the principle of least privilege and ensure that users only get permissions for what they require, nothing more. With AWS WAF, our application is protected from common web exploits and bots as we can filter traffic based on rules that we have created. Using AWS VPC, we will create public and private subnets to ensure that instances in the private subnets are not accessible to the internet. With Amazon API Gateway, CORS protection is implemented, and requests are restricted to only valid clients/sources.

IAM Roles

To restrict and protect access to AWS sources and services, 3 Here are some of the main IAM

roles and their policies we implemented:

Role	AWSServiceRoleForRDS
Assumed By	RDS
Policy	Permissions
AmazonRDSServiceRolePolicy	<ul style="list-style-type: none"> Allow write to CloudWatch, list and write to CloudWatch Logs. Allow list and write to EC2 instances Allow read and write to Kinesis Allow write to RDS and SNS

Role	CodeDeployRole
Assumed By	EC2 Instances in g2t8-ec2-code-deploy
Policy	Permissions
AWSCodeDeployRole	<ul style="list-style-type: none"> Allow read and write to CloudWatch Allow list and write to EC2 Allow list and write to EC2 Auto Scaling Allow list and write to ELB Allow read to ELB v2 Allow read and write to Resource Group Tagging Allow write to SNS

Role	codebuild-g2t8-github-connection-service-role
Assumed By	g2t8-builder
Policy	Permissions
CodeBuildBasePolicy-g2t8-builder-ap-south-east-1	<ul style="list-style-type: none"> Allow write to CloudWatch Logs Allow write to CodeBuild Allow read and Write to S3 Allow read from Secrets Manager

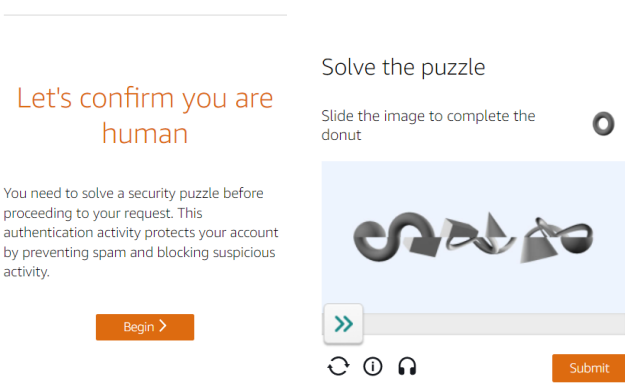
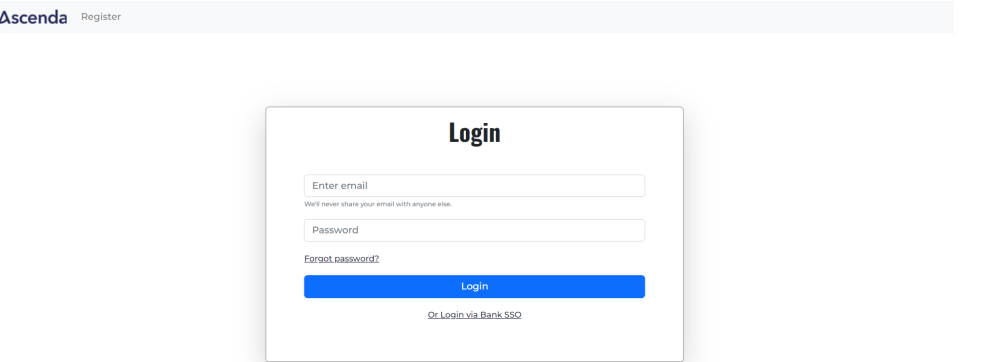
Role	AWSCodePipelineServiceRole-ap-southeast-1-g2t8-code-pipeline
Assumed By	g2t8-code-pipeline
Policy	Permissions

Performance View

No	Description of the Strategy	Justification
1	Auto Scaling group	We are expecting high traffic during specific times of the day. To cope with the additional demand, we rely on the Auto Scaling group to adjust the capacity.
2	Route read requests to database replica	To mitigate the computational workload on the primary database, read requests are routed to the replica instead.
3	Caching in Cloudfront	Static content will be retrieved from the closest point of presence. This allows for lower latency when serving static content

Our authorization servers are hosted on EC2 instances organised into separate autoscaling groups. Instances are scaled across multiple AZs, in response to the computational workload, maximising capacity. The autoscaling group has an autoscaling policy in place that will dynamically resize the autoscaling group based on the average CPU utilisation for the EC2 instances.

Website Screenshots

<p>Web Captcha</p>	
<p>Login</p>	

Register

The screenshot shows the Ascenda Login page. At the top left, there is a header with the Ascenda logo and a 'Login' link. The main content area features a white card titled 'New Account'. Inside the card, there are five input fields: 'First Name', 'Last Name', 'Email', a date field with a calendar icon and the placeholder 'dd/mm/yyyy', and 'Password'. Below these fields is a blue 'Register' button and a smaller 'Login' link.

User profile page

The screenshot shows the Ascenda Profile page. At the top left, there is a header with the Ascenda logo, a 'Profile' link, and a 'Log out' link. The main content area displays a list of user details in a table-like format:

Given Name	Ezequiel
Family Name	Ledner
Gender	Male
Date of Birth	1964-03-07
Email	ezequiel.ledner@robelInfo
Phone Number	+222 1-457-605-1253

Below the table is a blue 'Delete Account' button.

Admin login with only read role

The screenshot shows the Admin 'My Bank' page. At the top left, there is a header with the 'My Bank' logo and a 'Logout' link. The main content area is divided into two sections: 'Users' and 'Enrollment'. The 'Users' section contains a table of customers:

Email	Name	User ID	Status	Created At	Updated At	Actions
pagac_vince@yost.io	Vince Pagac	7eb073af-1377-409d-90ef-55db627852d2	enrolled			?
leeshuoan38@gmail.com	Shuoan Lee	7eb073af-1377-409d-90ef-55db627852d3	not enrolled			?
russel.stephan@kihn.name	Stephan Russel	8cecd1af-6c38-4186-9fe7-ba1cf15a7379	enrolled			?

Admin login with read and write roles

The screenshot shows the Ascenda Register page. At the top left, there is a header with the Ascenda logo and a 'Register' link. The main content area features a white card titled 'Login'. Inside the card, there are two input fields: one for the email address (containing 'russel.stephan@kihn.name') and one for the password (containing '*****'). Below these fields is a blue 'Login' button and a smaller 'Or Login via Bank SSO' link. A confirmation dialog box is overlaid on the card, asking: 'To login, Ascenda will need to fetch your personal information. Do you approve?' with 'Cancel' and 'OK' buttons.