DALHOUSIE
UNIVERSITY

CSCI 5411

Advanced Cloud Architecting

Prof. Lu Yang

Final – Term Project

Milestone – 02

Banner ID: B00988337

Name: Patel Het Ghanshyambhai

Date: 29th June, 2025

Table of Contents –

01.	Introduction	03.
02.	Improved Architecture Diagram	04.
03.	Improved Data Sequence Diagram	05.
04.	Service Selection & Configuration Details	06.
05.	Infrastructure as Code (IaC)	14.
06.	Monitoring & Logging Solutions	18.
07.	Functional Requirements Demonstration	19.
08.	Non-Functional Requirements Demonstration	26.
09.	Security Measure at all Layers	29.
10.	Cost Analysis & Optimization Strategies	31.
11.	Lessons Learned & Future Improvements	32.
12.	Conclusion & References	33.

LoanSyncro: Loan Repayment Tracker

1. Introduction:

This report details the LoanSyncro project, a modern financial management application designed to simplify the tracking and optimization of personal and small business loans and their associated repayments. In an increasingly complex financial landscape, individuals and organizations often face challenges in maintaining a clear, consolidated overview of their various financial obligations. LoanSyncro addresses this critical need by providing an intuitive, centralized platform where users can effortlessly record new loans, meticulously log repayment transactions, and gain real-time, actionable insights into their overall financial standing.

LoanSyncro aims to demonstrate the practical application of cloud principles to build a robust, scalable, and secure solution. The core objective is to empower users with the tools necessary to take proactive control of their financial health, ensuring timely payments and a comprehensive understanding of their loan portfolios. This document will elaborate on the architectural design, implementation details, and how the deployed solution fulfils both functional and non-functional requirements, adhering to industry best practices and the AWS Well-Architected Framework.

2. Improved Architecture Diagram:

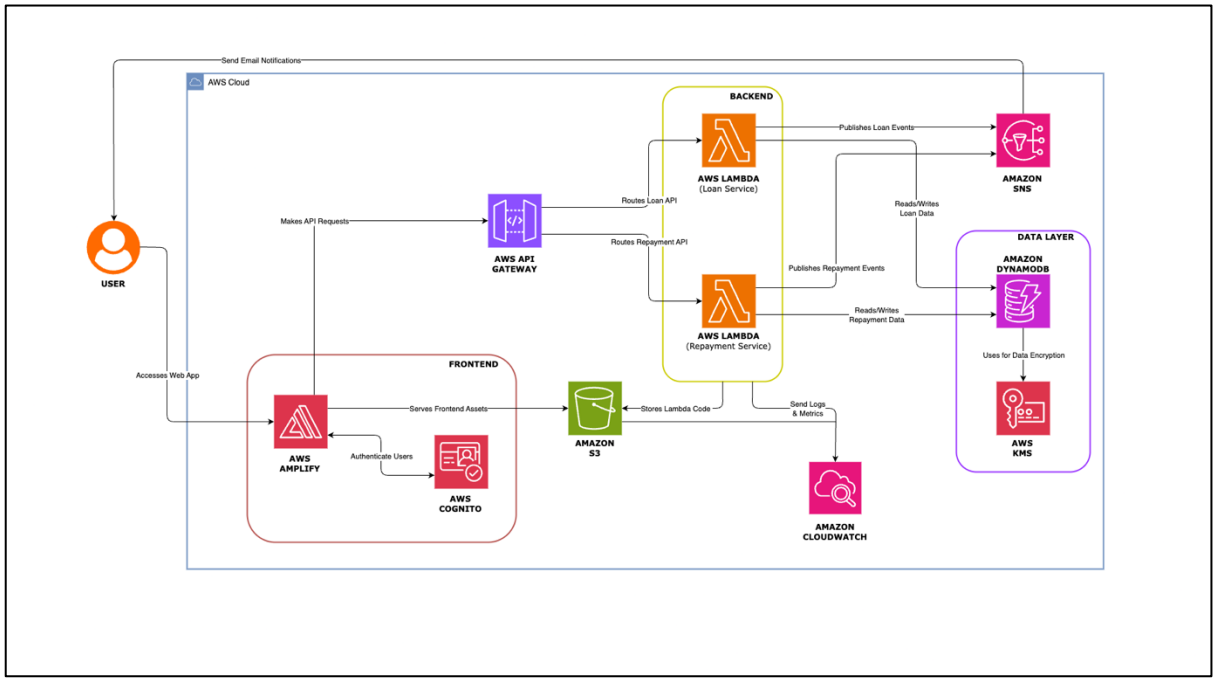

Fig. 1. Final Architecture Diagram

Link - https://app.diagrams.net/?title=Final-Architecture-Diagram-(M2)-B00988337.drawio&dark=auto#Uhttps%3A%2F%2Fdrive.google.com%2Fuc%3Fid%3D1s-wAW-YdFQRn2s7K6-nd7FFXTs8k9inl%26export%3Ddownload

3. Improved Data Sequence Diagram:

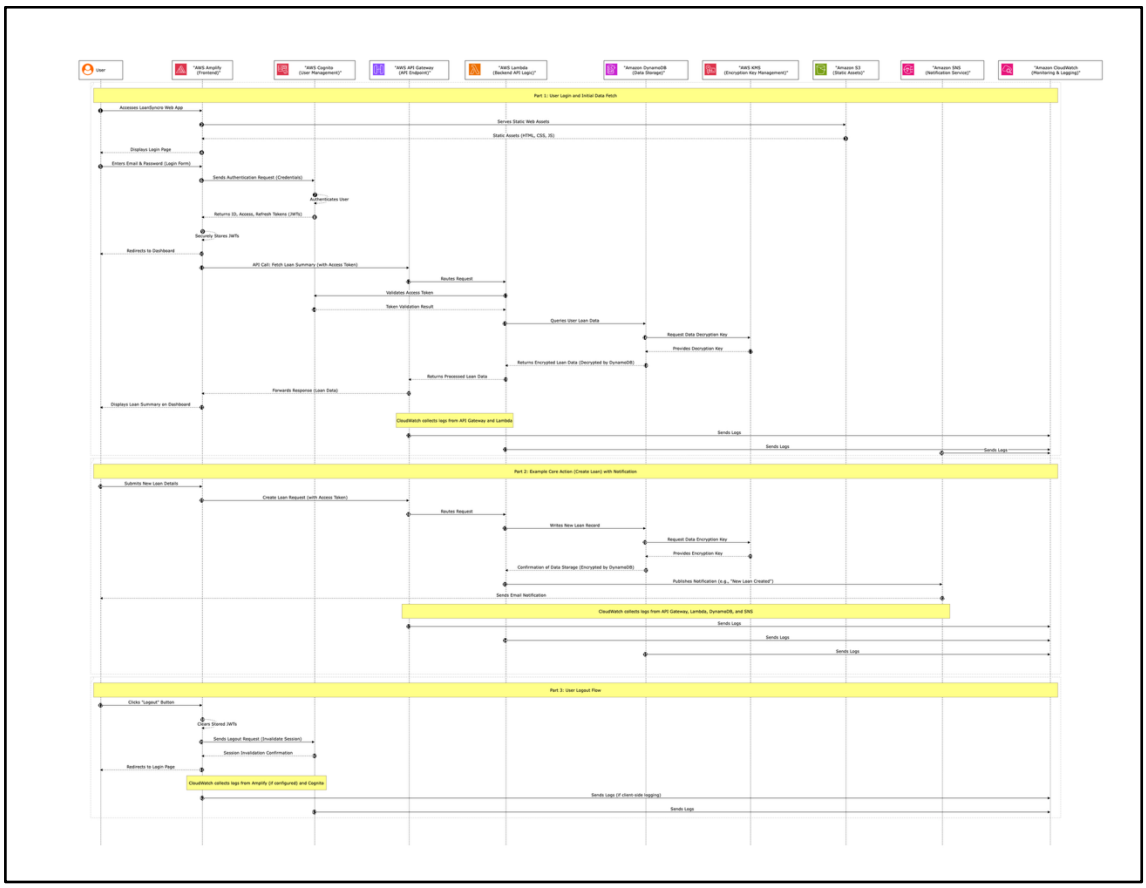

Fig. 2. Final Data Sequence Diagram (UML).

Link - https://app.diagrams.net/?title=Final-UML-Diagram-(M2)-B00988337.drawio&dark=auto#Uhttps%3A%2F%2Fdrive.google.com%2Fuc%3Fid%3D1iS3Ug6jKiiWgehDwQxEMeufWqzWo0YtA%26export%3Ddownload

4. Services Selection & Configuration Details:

I have selected a suite of AWS services that align with the project's functional and non-functional requirements, emphasizing well-managed architecture frameworks for scalable solutions.

1. AWS Amplify (Developer Tools)

Role: AWS Amplify is used for hosting the frontend React application. It provides a complete platform for web and mobile development, including continuous deployment, hosting, and backend integration.

Configuration:

- Connected to a GitHub repository for continuous deployment.
- Configured with rewrite rules for Single Page Application (SPA) routing.
- Environment variables are set to connect to the API Gateway endpoint and Cognito User Pool.

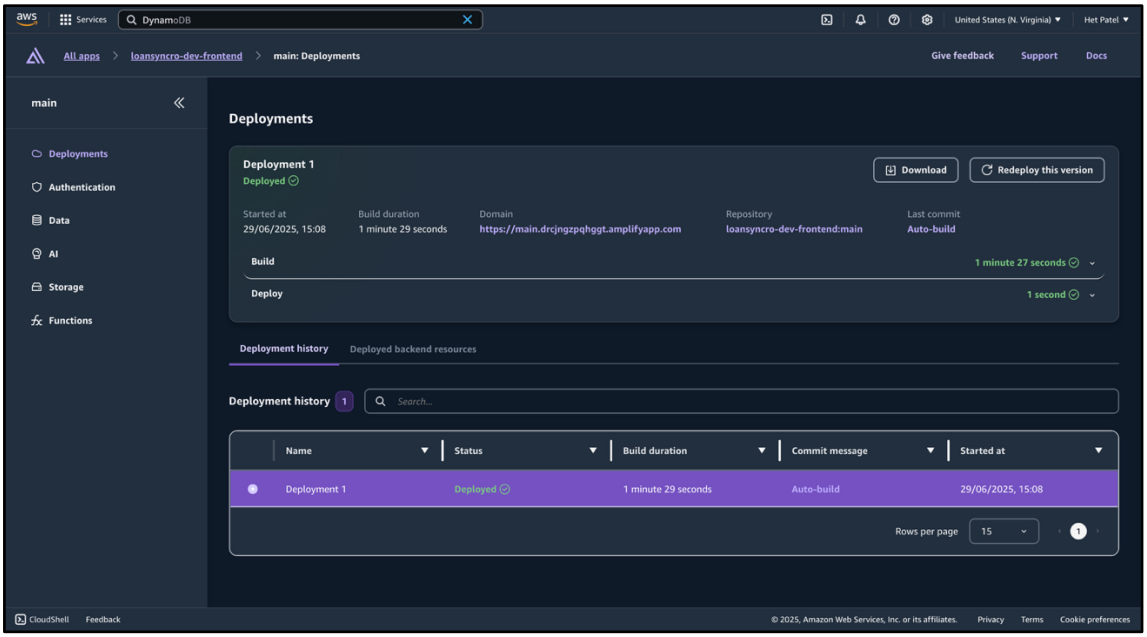

Fig. 3. AWS Amplify App Settings showing connected repository and build settings.

2. Amazon API Gateway (Networking & Content Delivery)

Role: API Gateway acts as the front door for the backend Lambda functions, handling all incoming HTTP requests, routing them to the correct Lambda, and managing CORS.

Configuration:

- REST API: A regional REST API is created.
- Resources & Methods: Defined resources for `/loans`, `/loans/{loan_id}`, `/repayments`, `/repayments/summary`, and `/repayments/loan/{loan_id}` with appropriate HTTP methods (GET, POST, PUT, DELETE).
- Integration: Configured as `AWS_PROXY` integration with Lambda functions, passing the entire request event to Lambda.
- CORS: OPTIONS methods are explicitly defined for all paths to handle Cross-Origin Resource Sharing, allowing the frontend to communicate with the API.
- Deployment & Stage: The API is deployed to a stage (e.g., `dev`) to make it accessible.

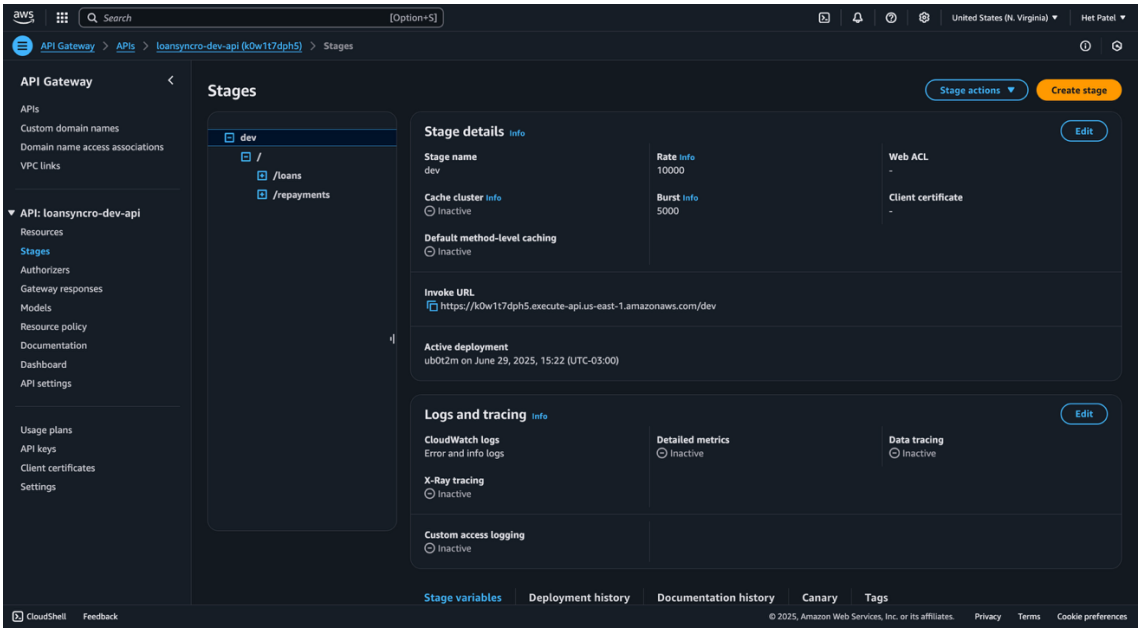

Fig. 4. AWS API Gateway Console showing defined resources and methods for /loans and /repayments.

3. AWS Lambda (Compute)

Role: Lambda functions serve as the compute layer for the backend logic, handling all business operations for loans and repayments. They are serverless, scaling automatically with demand.

Configuration:

- Runtime: Python 3.9.

- Handlers: Separate Lambda functions (`loans_handler`, `repayments_handler`) are created for managing loans and repayments, respectively.
- Environment Variables: Configured with environment variables for DynamoDB table names, S3 bucket name, Cognito User Pool IDs, KMS Key ID, and SNS Topic ARN.
- IAM Role: Assigned an IAM execution role with least-privilege permissions to access DynamoDB, S3, KMS, CloudWatch Logs, and SNS.
- Layers: A Lambda Layer is used to package Python dependencies (e.g., `PyJWT`, `requests`), reducing deployment package size and improving cold start times.

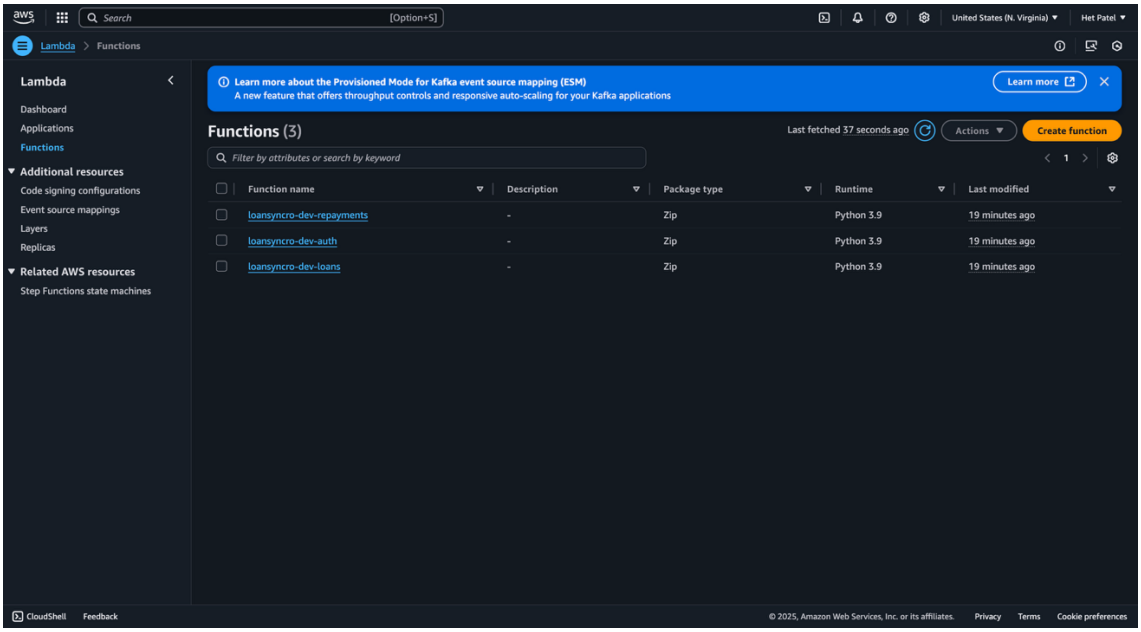

Fig. 5. AWS Lambda Console showing all the Lambda functions.

4. Amazon DynamoDB (Database)

Role: DynamoDB is a fully managed NoSQL database service used to store user profiles, loan details, and repayment records. Its key-value and document data models are well-suited for the application's structured but flexible data.

Configuration:

- Tables: Three tables are created: `users`, `loans`, and `repayments`.
- Billing Mode: Configured with `PAY_PER_REQUEST` billing mode, which is ideal for unpredictable workloads and cost optimization.
- Primary Keys: `id` as the hash key for all tables.

- Global Secondary Indexes (GSIs): `email-index` on `users` table, `user-id-index` on `loans` table, and `loan-id-index` on `repayments` table to enable efficient querying by non-primary key attributes.
- Encryption: Server-side encryption enabled using AWS KMS.
- Point-in-Time Recovery (PITR): Enabled for continuous backups.

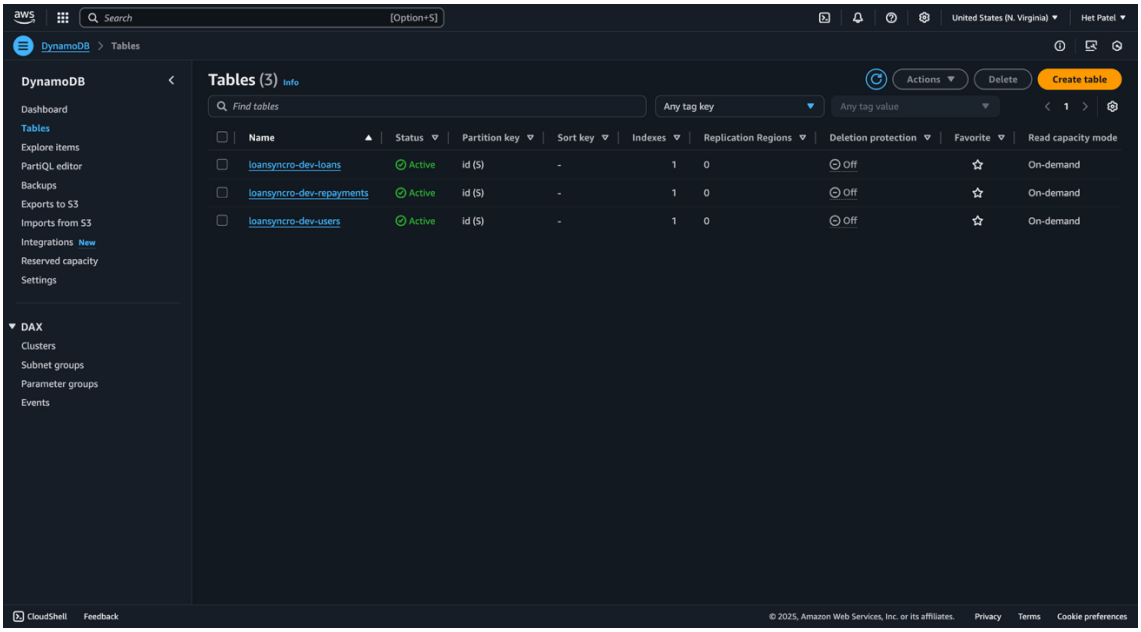

Fig. 6. AWS DynamoDB Console showing all the tables.

5. Amazon S3 (Storage)

Role: An S3 bucket is provisioned for general application storage, although currently not heavily utilized by the core loan/repayment logic. It provides highly durable and scalable object storage.

Configuration:

- Versioning: Enabled to protect against accidental deletions and allow recovery of previous object versions.
- Encryption: Server-side encryption enabled using AWS KMS.
- Public Access Block: All public access is blocked by default for security.
- Lifecycle Policy: Configured to transition objects to `STANDARD_IA` after 30 days and `GLACIER` after 90 days for cost optimization.

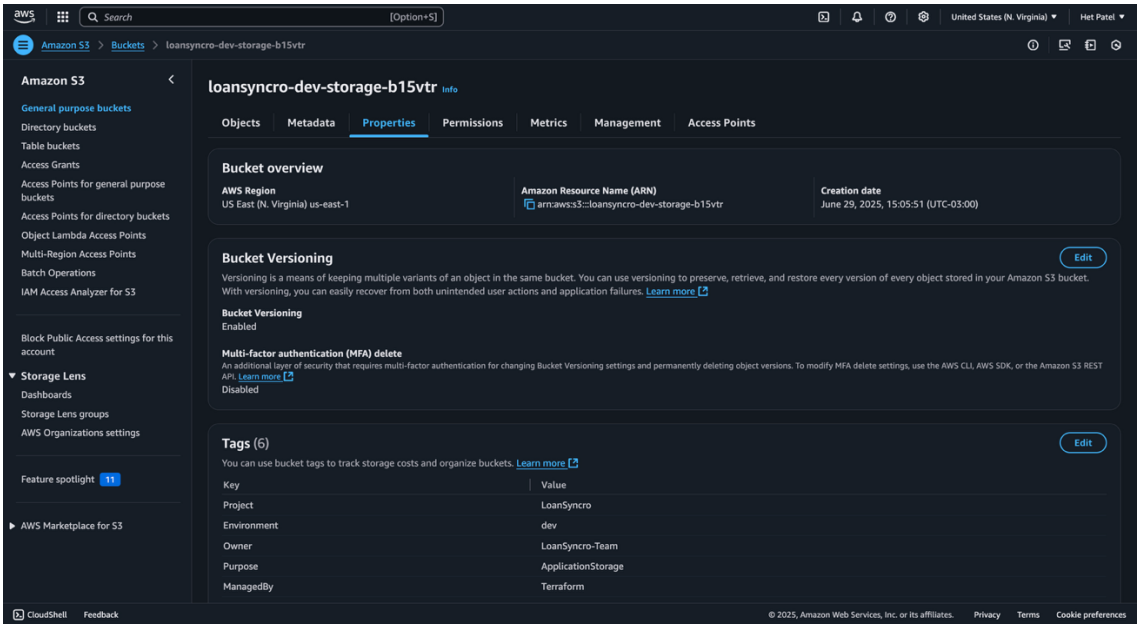

Fig. 7. AWS S3 Console showing bucket properties including versioning.

6. Amazon Cognito (Security, Identity, and Compliance)

Role: Cognito User Pools provide user authentication and management for the application, handling user registration, login, and token issuance.

Configuration:

- User Pool: Configured with email as the username attribute.
- Password Policy: Enforces strong password requirements (minimum length, uppercase, lowercase, numbers, symbols).
- Email Verification: Automatic email verification upon signup with a custom message template.
- User Pool Client: A client is created for the frontend application, configured with appropriate callback and logout URLs for local development and Amplify hosting.
- JWT Validation: Lambda functions validate JWTs issued by Cognito to authenticate API requests.

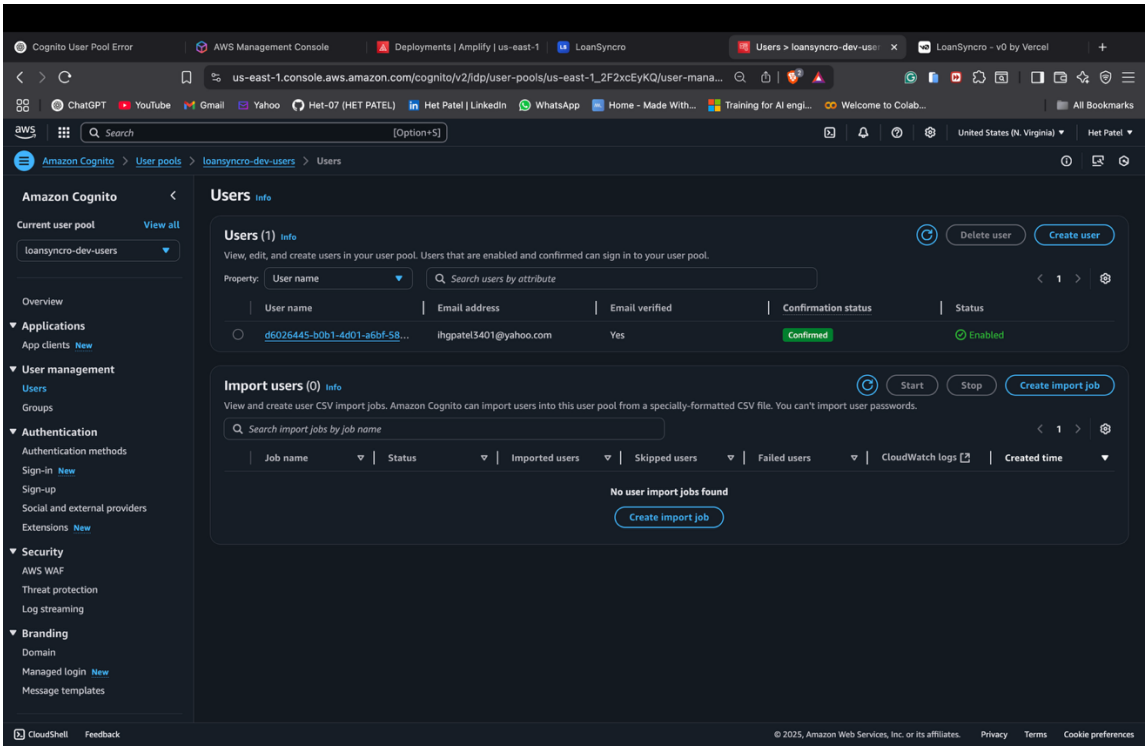

Fig. 8. AWS Cognito Console showing Users.

7. Amazon SNS (Application Integration)

Role: SNS is used for sending notifications and alerts, specifically for critical events like Lambda errors, API Gateway errors, and successful loan/repayment creations.

Configuration:

- Topic: An SNS topic (`loansyncro-dev-alerts`) is created.
- Subscription: An email subscription is added to the topic, ensuring that designated recipients receive email alerts.
- Integration: Lambda functions publish messages to this topic, and CloudWatch Alarms are configured to send notifications to it.

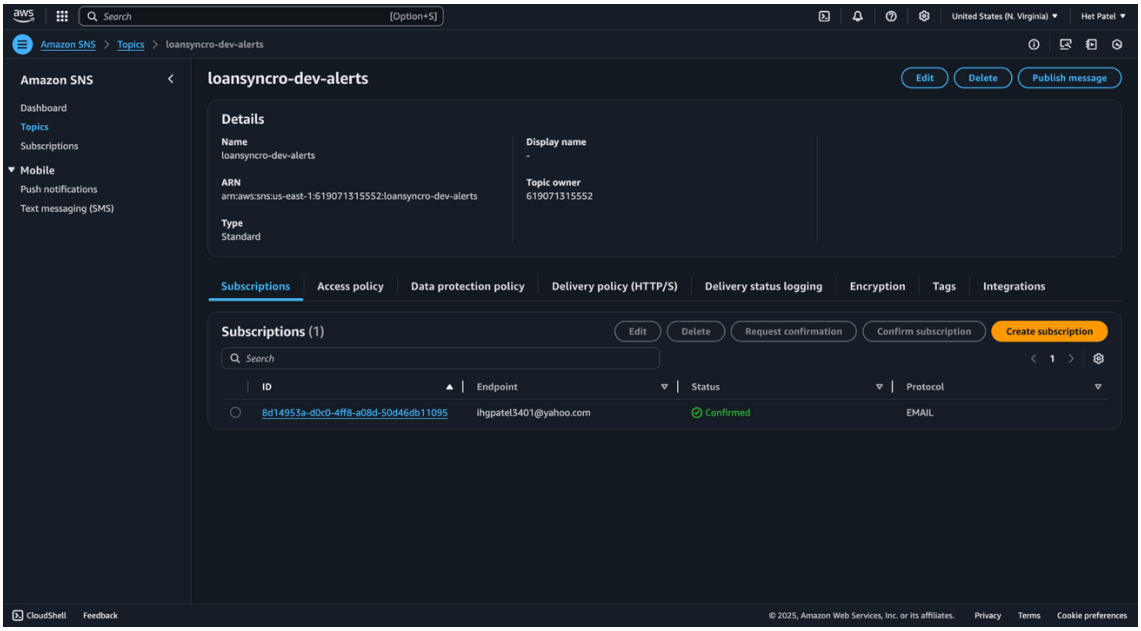

Fig. 9. AWS SNS Console showing the alerts topic and its email subscription.

8. AWS CloudWatch (Management and Governance)

Role: CloudWatch provides monitoring and logging capabilities for the application, collecting logs from Lambda and API Gateway, and enabling the creation of alarms based on metrics.

Configuration:

- Log Groups: Dedicated log groups are created for each Lambda function (`/aws/lambda/loansyncro-dev-loans`, `/aws/lambda/loansyncro-dev-repayments`) with a defined retention period.
- Metric Alarms:
  lambda_errors`: Triggers an SNS notification if any Lambda function reports errors.
  `api_gateway_errors`: Triggers an SNS notification if API Gateway experiences a high rate of 4XX errors.

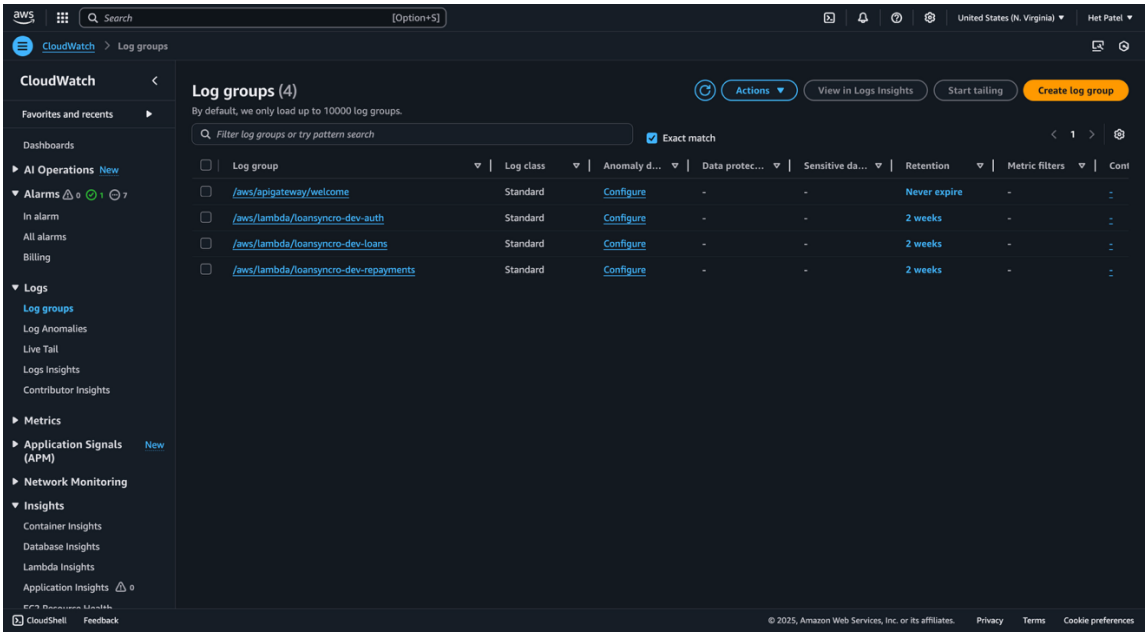

Fig. 10. AWS CloudWatch Console showing all the log groups.

9. AWS Key Management Service (KMS) (Security, Identity, and Compliance)

Role: KMS is used to manage encryption keys, providing a secure and centralized way to encrypt data at rest for DynamoDB tables and S3 buckets.

Configuration:

- A customer-managed KMS key (`loansyncro-dev-kms-key`) is created.
- This key is explicitly used for server-side encryption in DynamoDB and S3.

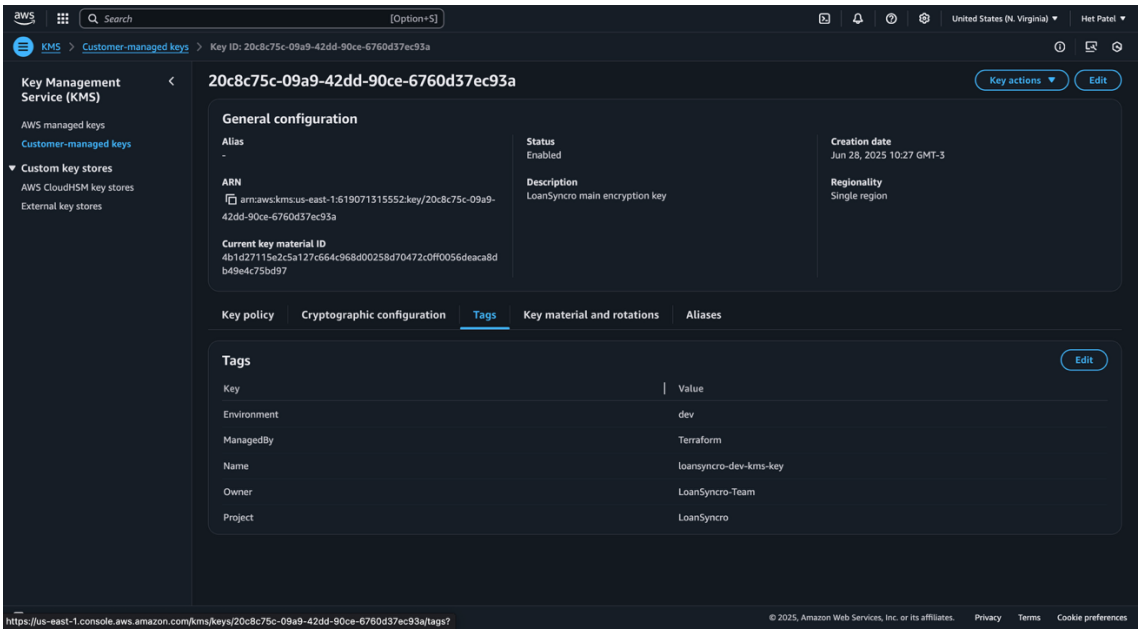

Fig. 11. AWS KMS Console showing general configurations.

5. Infrastructure as Code (IaC):

To ensure robust, scalable, and maintainable infrastructure, LoanSyncro leverages Terraform as its Infrastructure as Code (IaC) tool. Terraform allows us to define and provision our AWS resources in a declarative manner, ensuring consistency, repeatability, and version control over our cloud environment. This approach significantly reduces manual configuration errors and streamlines deployment processes.

The LoanSyncro project's infrastructure is organized into several Terraform files, each responsible for a specific domain of AWS services:

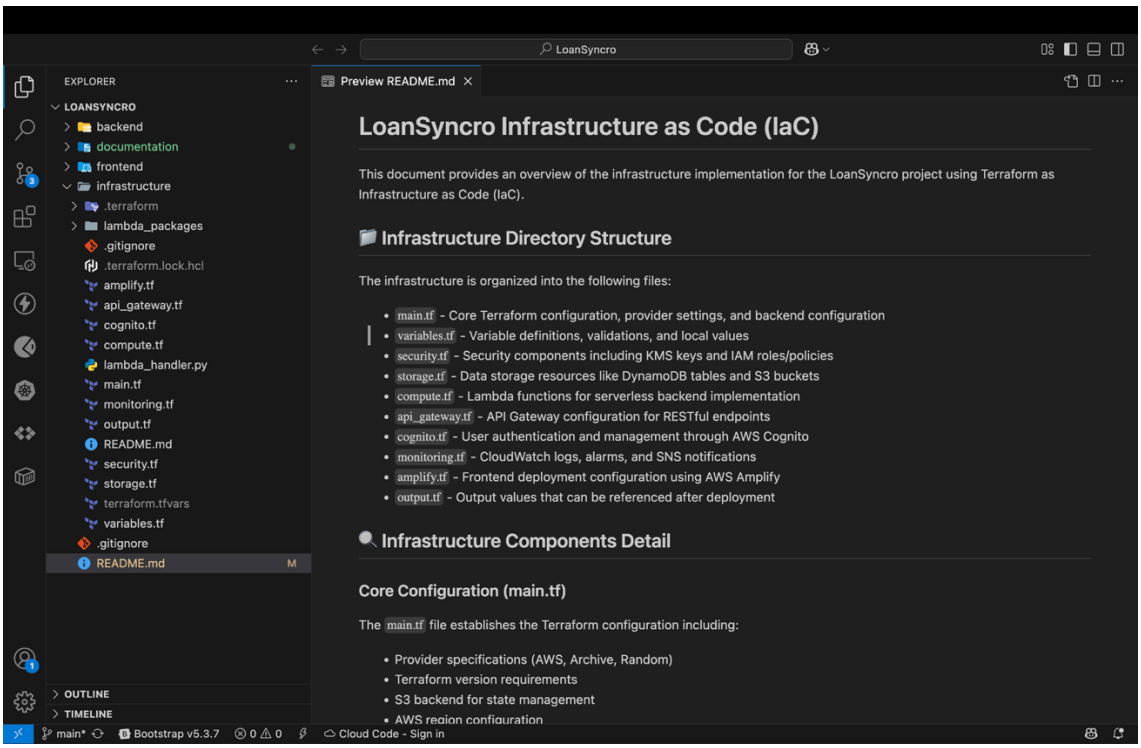

Fig. 12. Infrastructure as Code implementation files.

1. main.tf:

This is the core configuration file that defines the AWS provider, required Terraform versions and providers, and sets up the S3 backend for storing Terraform state. It also includes local values for consistent naming conventions across resources and retrieves current AWS account and region information. This file acts as the entry point for the entire infrastructure deployment.

2. variables.tf:

This file centralizes all configurable parameters for the infrastructure. It defines variables such as the AWS region, environment (dev, staging, prod), project name, notification email for alerts, GitHub repository details for Amplify, and optional

custom domains. This separation allows for easy customization and environment-specific deployments without modifying the core logic.

3. security.tf:

Dedicated to security-related configurations, this file defines the AWS Key Management Service (KMS) key for encryption at rest across various services (DynamoDB, S3). It also provisions the AWS Identity and Access Management (IAM) role and policy for Lambda functions, granting them necessary permissions to interact with CloudWatch Logs, DynamoDB tables, KMS, S3, and SNS. This ensures that Lambda functions operate with the principle of least privilege.

4. storage.tf:

This file manages the persistent data storage components of LoanSyncro. It defines three Amazon DynamoDB tables: `users`, `loans`, and `repayments`, each configured with on-demand billing, a hash key, and a global secondary index for efficient querying. All DynamoDB tables are enabled with server-side encryption using the KMS key and point-in-time recovery for data durability.
Additionally, it configures an Amazon S3 bucket for file storage, including versioning, server-side encryption, public access blocking, and a lifecycle policy for cost optimization by transitioning older data to lower-cost storage classes.

5. compute.tf:

This file is responsible for the serverless compute resources. It packages the Python Lambda handler code and creates a Lambda Layer for common Python dependencies (like `PyJWT`, `python-jose`, `requests`), ensuring efficient deployment and dependency management. It then defines two AWS Lambda functions: `loans_handler` and `repayments_handler`, which serve as the backend logic for managing loan and repayment data, respectively. Each Lambda function is configured with the appropriate IAM role, runtime, timeout, memory, and environment variables to connect to other AWS services.

6. api_gateway.tf:

This file sets up the Amazon API Gateway, which acts as the front door for the backend Lambda functions. It defines the REST API, resources (e.g., `/loans`, `/repayments`, `/repayments/summary`, `/repayments/loan/{loan_id}`), and methods (GET, POST, PUT, DELETE) for each resource. It integrates these methods with the corresponding Lambda functions using AWS_PROXY integration. Crucially, it also configures CORS (Cross-Origin Resource Sharing) for all endpoints, allowing the frontend application to securely interact with the API. A deployment and stage are also defined to make the API accessible.

7. cognito.tf:

This file handles user authentication and authorization through Amazon Cognito. It defines the Cognito User Pool, configuring password policies, email as the username attribute, automatic email verification, and account recovery settings. It also creates a Cognito User Pool Client, which the frontend application uses to interact with the user pool for signup, login, and token management. Callback and logout URLs are specified to support the frontend's authentication flow.

8. monitoring.tf:

This file establishes the monitoring and alerting infrastructure. It creates AWS CloudWatch Log Groups for the Lambda functions to capture their execution logs. It also defines an Amazon SNS (Simple Notification Service) topic for alerts and an email subscription to this topic, ensuring that critical notifications (e.g., Lambda errors, API Gateway 4XX errors) are sent to the designated email address. CloudWatch Metric Alarms are configured to trigger alerts based on predefined thresholds for Lambda errors and API Gateway client errors.

9. amplify.tf:

This file provisions the AWS Amplify application for the frontend. It connects to the specified GitHub repository and branch, using a GitHub token retrieved from AWS SSM Parameter Store for secure access. It defines the build settings, including pre-build and build commands for the React frontend, and specifies the base directory for artifacts. Environment variables required by the frontend (e.g., API URL, Cognito IDs, S3 bucket name) are passed to the Amplify build. Custom rewrite rules are also configured to handle single-page application (SPA) routing.

10. output.tf:

This file defines the output values that are useful after Terraform applies the configuration. These outputs typically include important endpoints or identifiers, such as the API Gateway invoke URL, Cognito User Pool ID, and Cognito User Pool Client ID. These values are then consumed by the frontend application to connect to the backend services.

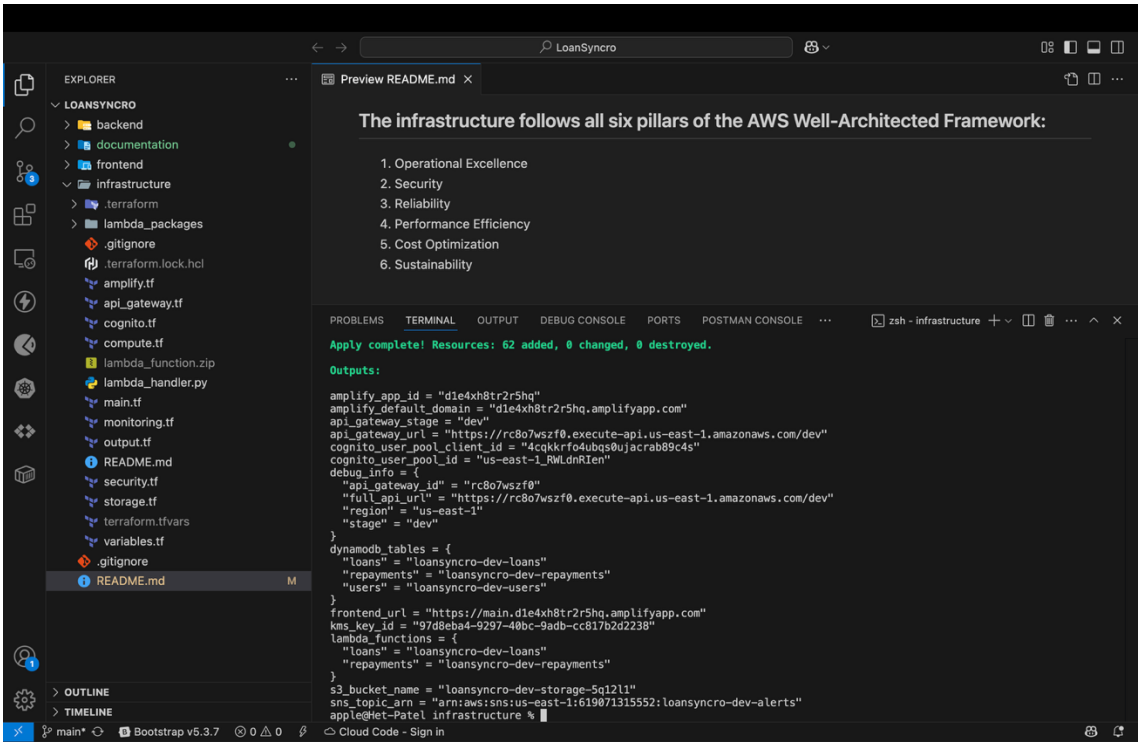

Fig. 13. Terraform Apply adding all the resources.

By segmenting the infrastructure into these logical Terraform files, LoanSyncro achieves a modular, understandable, and easily manageable cloud environment, facilitating future development, scaling, and troubleshooting.

6. Monitoring & Logging Solutions:

Monitoring and logging are crucial for maintaining the health, performance, and security of the LoanSyncro application.

1. Centralized Logging with CloudWatch Logs:

- All Lambda function invocations and API Gateway requests automatically send their logs to dedicated CloudWatch Log Groups.
- These logs capture execution details, errors, and custom print statements from the Lambda code, providing granular insights into application behaviour.
- Log retention policies are configured (e.g., 7 days) to manage storage costs while retaining sufficient historical data for debugging.

2. Proactive Alerting with CloudWatch Alarms and SNS:

- Lambda Error Alarm: A CloudWatch alarm monitors the `Errors` metric for Lambda functions. If the sum of errors exceeds 0 over a 10-minute period, an alert is triggered.
- API Gateway Error Alarm: An alarm monitors the `4XXError` metric for the API Gateway. If the count of 4XX errors exceeds a threshold (e.g., 10) over a 5-minute period, an alert is triggered.
- Both alarms publish notifications to the `loansyncro-dev-alerts` SNS topic, which then sends email notifications to the configured `notification_email`. This ensures that the development team is immediately aware of critical issues.

3. Application-Level Notifications:

- The Lambda functions themselves are instrumented to publish messages to the SNS alerts topic upon significant events, such as the successful creation of a new loan or repayment, or when a loan's status changes to 'paid'. This provides real-time operational awareness beyond just errors.

7. Functional Requirements Demonstration:

The LoanSyncro application successfully implements core functionalities:

1. User Authentication and Management:

User authentication and authorization are foundational to LoanSyncro's security model, managed entirely by AWS Cognito User Pools. Upon registration, new users provide their full name, email, and a strong password. Cognito handles the secure storage of user credentials and, critically, automatically verifies the user's email address upon successful signup, eliminating the need for a separate confirmation step for the user. This streamlined process ensures that only legitimate users can access the application.

Once registered, users can log in using their verified email and password. Cognito issues JSON Web Tokens (JWTs) upon successful authentication, which are then used by the frontend application to authorize subsequent API requests. On the backend, a custom authentication service within the Lambda functions intercepts these JWTs. It validates the token's signature against Cognito's public keys, verifies its expiration, and ensures it was issued by the correct user pool and client. This robust validation process guarantees that only authenticated and authorized users can interact with the backend services, preventing unauthorized data access or manipulation. Furthermore, upon a user's first successful login, their profile (including their unique Cognito `sub` ID, email, and full name) is automatically synchronized and stored in a dedicated DynamoDB `users` table. This ensures that user-specific data, such as loans and repayments, can be correctly associated and retrieved, providing a personalized experience.

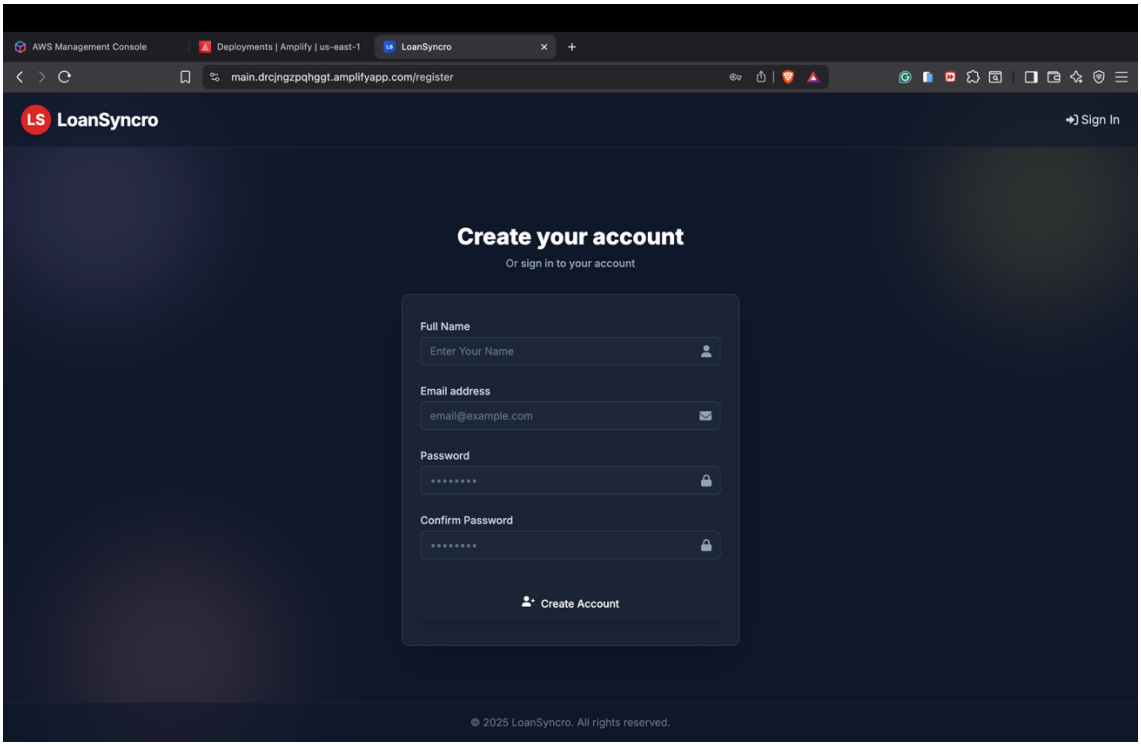

Fig. 14. User Registration Page.

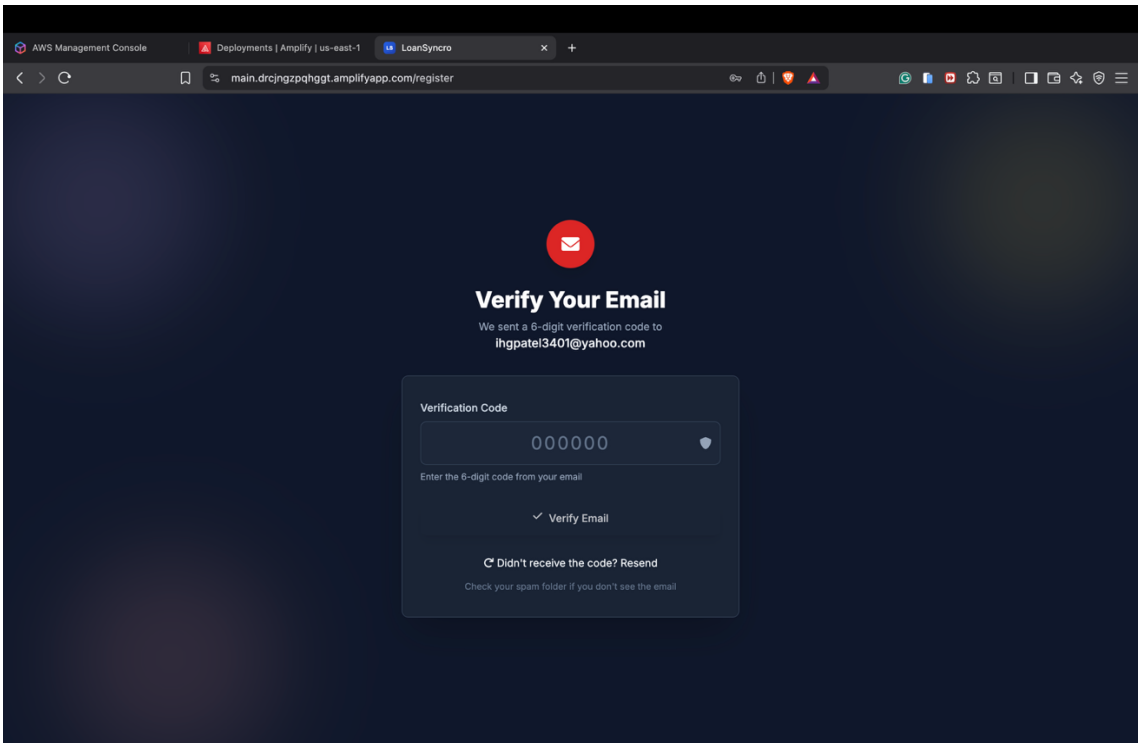

Fig. 15. Email Verification Page.

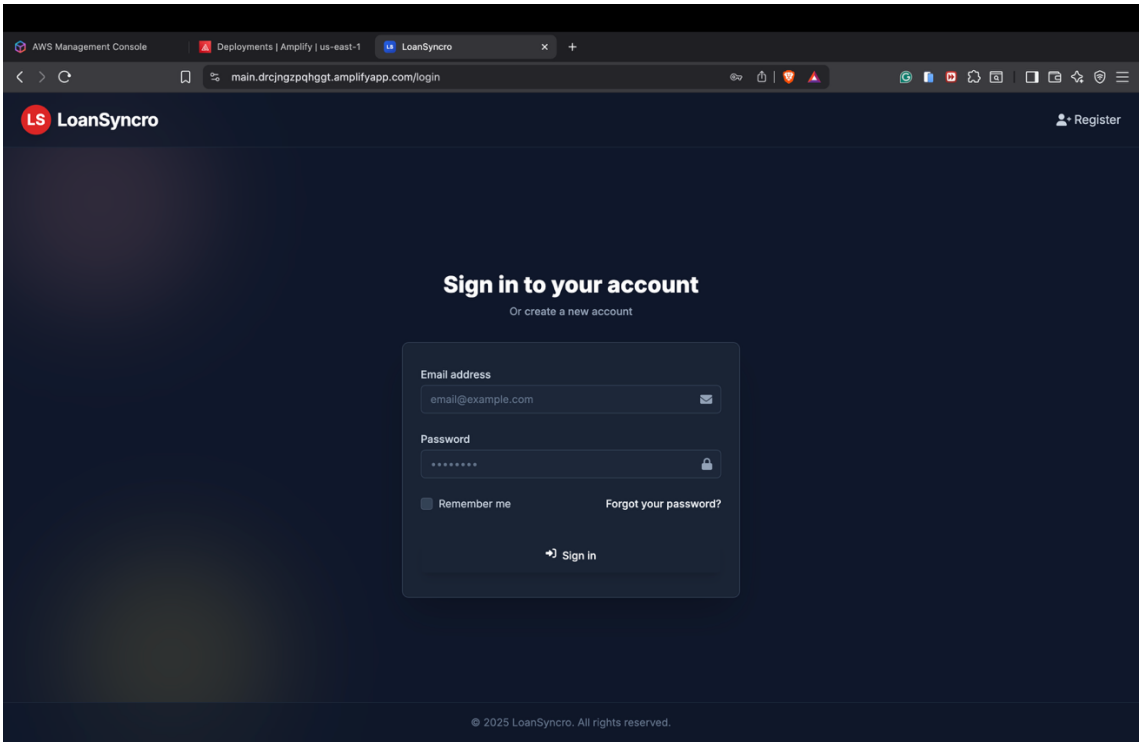

Fig. 16. Login Page.

2. Loan Management (CRUD):

LoanSyncro provides a complete suite of Create, Read, Update, and Delete (CRUD) functionalities for managing loans.

Creating a Loan: Users can initiate a new loan entry by providing essential details such as a title (e.g., "Home Mortgage," "Car Loan"), the principal amount, interest rate, and the loan term in months. A start date and an optional description can also be added. Upon submission, the backend Lambda function automatically calculates the monthly payment and the total amount to be repaid over the loan's lifetime, based on the provided principal, interest rate, and term. This automation reduces manual calculation errors and provides immediate financial clarity to the user. The newly created loan is then securely stored in the DynamoDB `loans` table, associated with the authenticated user's ID.

Viewing Loans: The application offers a dedicated "My Loans" page where users can view a comprehensive list of all their recorded loans. Each loan is displayed with key information such as its title, principal amount, interest rate, monthly payment, and current status (e.g., "active," "paid," "defaulted"). Users can also search for specific loans by title, allowing for quick navigation within a potentially large portfolio.

Viewing Loan Details: Clicking on an individual loan from the list navigates the user to a detailed view. This page presents all the information entered during creation, along with the calculated monthly payment and total amount. This granular view allows users to review specific terms and conditions of each loan.

Updating a Loan: Should any loan parameters change (e.g., a renegotiated interest rate or an extended term), users can easily edit existing loan entries. The update functionality allows modification of all fields, and the backend automatically recalculates the monthly payment and total amount based on the revised inputs. This ensures that the financial figures remain accurate and up-to-date.

Deleting a Loan: Users have the option to remove a loan entry from their portfolio. This action permanently deletes the loan and all associated repayment records from the DynamoDB tables, ensuring data integrity and allowing users to clean up their records as needed.

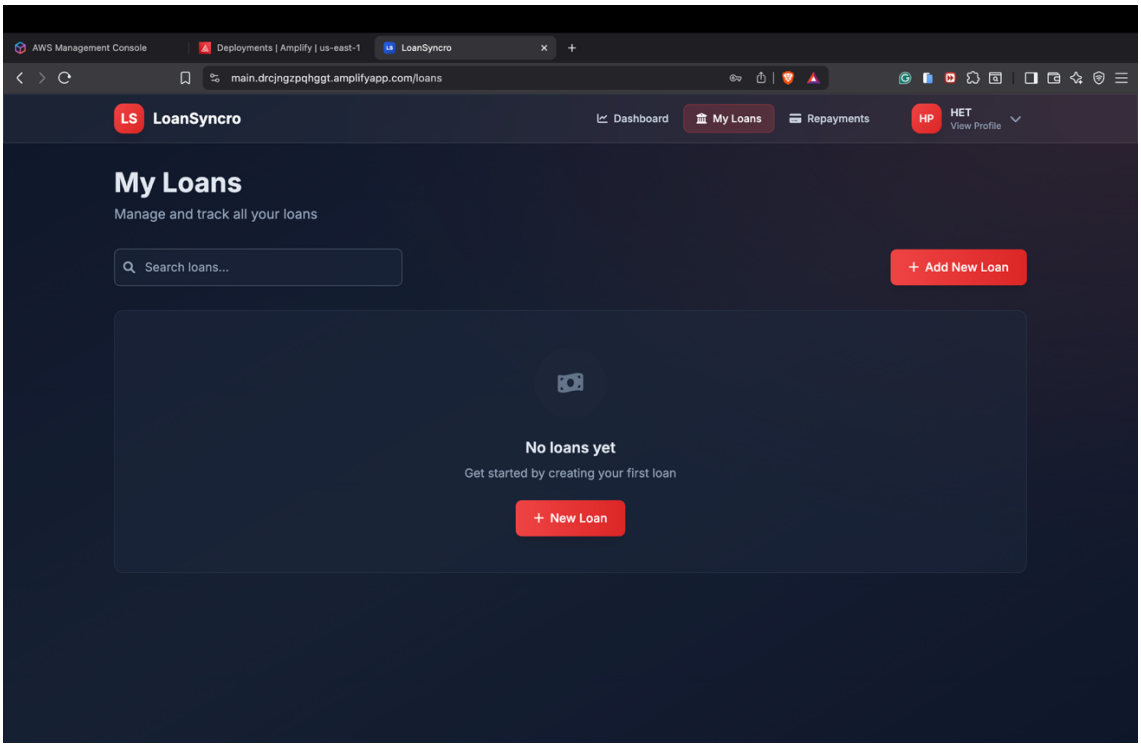

Fig. 17. Loan page to add loans.

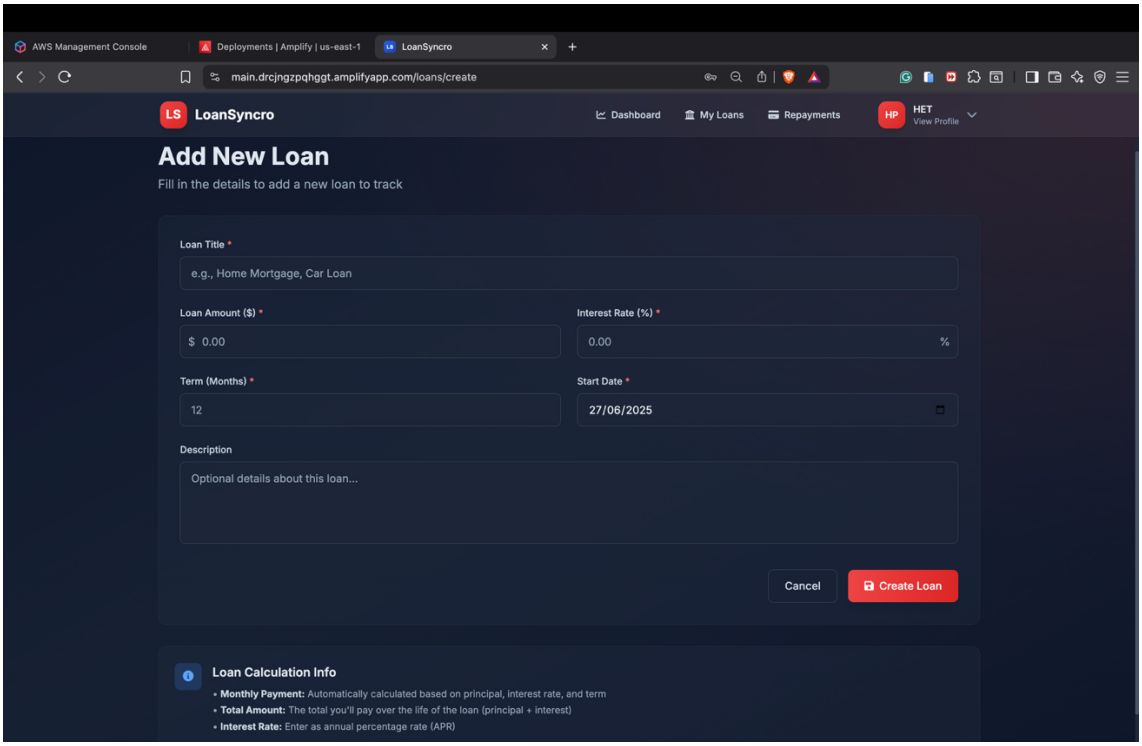

Fig. 18. Create new loan page.

3. Repayment Tracking (CRUD):

Beyond loan tracking, LoanSyncro facilitates the recording and management of individual repayments, providing a clear picture of payment history and outstanding balances.

Recording a Repayment: Users can record a new repayment by selecting the associated loan, entering the payment amount, the date of payment, and optional notes. The backend processes this repayment, storing it in the DynamoDB `repayments` table. Crucially, after each repayment is recorded, the system automatically updates the status of the corresponding loan. If the total amount repaid for a loan reaches or exceeds its `total_amount` (principal plus interest), the loan's status is automatically changed to "paid." This real-time status update provides immediate feedback to the user on their progress towards debt freedom.

Viewing Repayments: The "Repayments" page provides a consolidated view of all repayments made across all loans. Repayments are grouped by loan, allowing users to easily see the payment history for each of their financial obligations. For each loan, a summary of total repaid amount, remaining balance, and payment progress is displayed, offering a quick overview of their financial standing. Users can also search for repayments by the associated loan title.

Viewing Loan-Specific Repayments: Within the detailed loan view, users can also see a list of all repayments specifically made for that particular loan. This provides a focused history of payments for a single financial obligation.

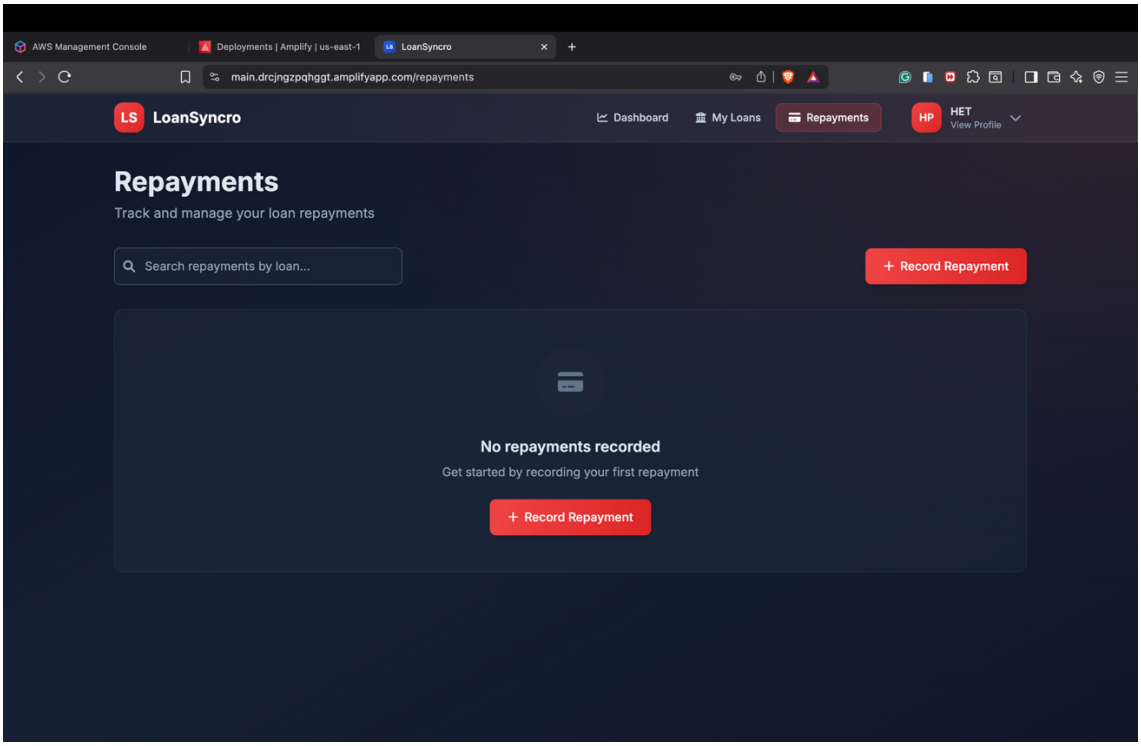

Fig. 19. Repayment page to add repaid record.

4. Dashboard and Summary

The LoanSyncro dashboard serves as the central hub for users, offering a high-level summary of their entire loan portfolio. This section aggregates data from both the `loans` and `repayments` DynamoDB tables to present key financial metrics at a glance. The dashboard prominently displays the total number of loans, the total amount borrowed across all loans, the cumulative amount repaid, and the current outstanding balance. These summary figures are dynamically updated as new loans are added or repayments are recorded, providing users with real-time insights into their financial position. The intuitive layout and clear presentation of these metrics empower users to quickly assess their financial health and identify areas requiring attention.

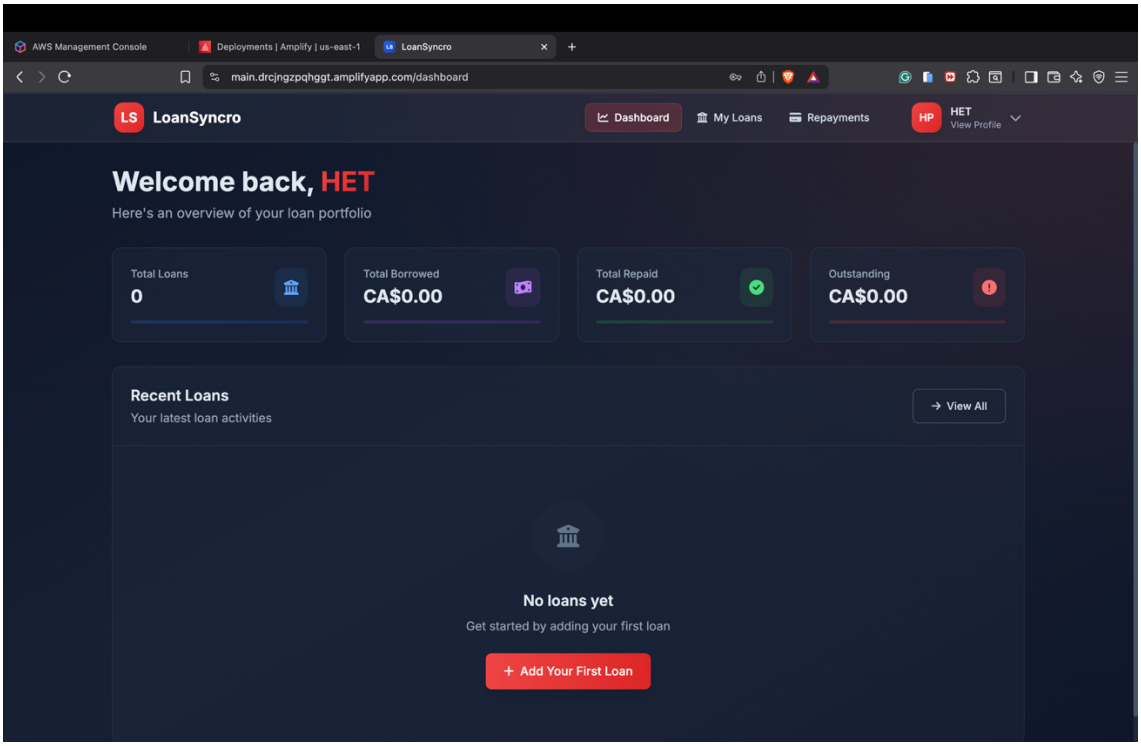

Fig. 20. Dashboard to track active loans and repaid amounts.

8. Non-Functional Requirements Demonstration:

Beyond core features, LoanSyncro is built with a strong emphasis on non-functional attributes, ensuring a reliable, secure, performant, and cost-effective solution.

1. Scalability

LoanSyncro is designed from the ground up to be highly scalable, leveraging the inherent elasticity of AWS serverless services. **AWS Lambda** functions, which power the backend API, automatically scale up or down based on the incoming request volume. This means the application can seamlessly handle a fluctuating number of users and API calls without requiring manual provisioning or de-provisioning of servers. Similarly, **Amazon API Gateway** acts as a fully managed entry point, capable of handling millions of concurrent requests and routing them efficiently to the appropriate Lambda functions. For data storage, **Amazon DynamoDB** is utilized in "On-Demand" capacity mode, which automatically scales throughput capacity to accommodate peak workloads without requiring capacity planning. This pay-per-request model ensures that the database can handle any load, from a single user to thousands, without performance degradation. The frontend, hosted on **AWS Amplify**, also provides automatic scaling for web hosting, distributing traffic globally to ensure low latency and high availability for users worldwide.

2. Reliability and High Availability

The architecture of LoanSyncro prioritizes reliability and high availability to ensure continuous service. All AWS services used, including Lambda, API Gateway, DynamoDB, S3, and Cognito, are inherently designed for high availability. They operate across multiple Availability Zones (AZs) within a region, providing built-in redundancy and fault tolerance. If one AZ experiences an outage, the services automatically failover to healthy AZs, minimizing downtime. **DynamoDB** further enhances reliability with automatic data replication across multiple AZs and offers Point-in-Time Recovery (PITR), enabling data restoration to any second within the last 35 days, protecting against accidental deletions or writes. **Amazon S3**, used for storing static assets and Lambda deployment packages, provides 99.999999999% (11 nines) of durability, ensuring data is virtually never lost. The use of **AWS Amplify** for frontend hosting also contributes to reliability by providing a globally distributed content delivery network (CDN) and automatic deployments, ensuring the application is always accessible and up-to-date.

3. Security

Security is paramount for a financial application, and LoanSyncro implements multiple layers of protection. **AWS Cognito** provides robust user authentication, handling password hashing, multi-factor authentication (if enabled), and secure token issuance. This offloads the complexity of user identity management, reducing the risk of common security vulnerabilities. Access to backend Lambda functions is secured via **API Gateway**, which acts as a secure front door. Within the Lambda functions, JWT validation ensures that only requests with valid, unexpired tokens from authenticated users are processed. **AWS Identity and Access Management (IAM)** roles are meticulously configured with the principle of least privilege, granting Lambda functions and other AWS resources only the minimum necessary permissions to perform their tasks. For data at rest, **AWS Key Management Service (KMS)** is used to encrypt sensitive data stored in DynamoDB tables and S3 buckets, providing an additional layer of protection. All communication between the frontend, API Gateway, and Lambda functions is encrypted in transit using HTTPS/SSL. Furthermore, the application's CORS (Cross-Origin Resource Sharing) policies are explicitly defined in API Gateway to prevent unauthorized cross-origin requests, enhancing overall application security.

4. Performance

LoanSyncro is optimized for performance to deliver a responsive user experience. **AWS Lambda** functions offer low-latency execution, processing API requests quickly. While cold starts can occur for infrequently used functions, the overall architecture is designed to minimize their impact. **Amazon API Gateway** provides efficient request routing and can be configured with caching (though not explicitly implemented in the current configuration, it's a readily available option) to further reduce latency for frequently accessed data. **Amazon DynamoDB** is a high-performance NoSQL database, offering single-digit millisecond latency for most read and write operations, ensuring that data retrieval and storage are fast. The frontend, hosted on **AWS Amplify**, benefits from global CDN distribution, which caches static assets closer to users, reducing load times and improving responsiveness regardless of geographical location.

5. Cost Optimization

The serverless architecture of LoanSyncro inherently promotes significant cost optimization. With **AWS Lambda** and **Amazon API Gateway**, the "pay-per-use" model means costs are incurred only when the services are actively processing requests, eliminating the need to provision and pay for idle server capacity. **Amazon DynamoDB** in "On-Demand" mode also follows a pay-per-request model, where users are billed for the actual read and write requests and stored

data, rather than provisioned throughput. **Amazon S3** offers tiered storage classes and lifecycle policies, automatically moving less frequently accessed data to more cost-effective storage (e.g., Standard-IA, Glacier) after a defined period, further reducing storage costs. By minimizing operational overhead and leveraging managed services, LoanSyncro achieves a highly efficient cost profile, making it suitable for both small-scale personal use and larger deployments.

6. Observability (Monitoring and Logging)

Comprehensive monitoring and logging are integrated into LoanSyncro to ensure operational excellence and facilitate troubleshooting. **Amazon CloudWatch Logs** automatically collects logs from all Lambda function invocations and API Gateway requests. These logs provide detailed insights into application behaviour, errors, and performance metrics. Log groups are configured with a 14-day retention policy to balance historical data availability with cost efficiency. To proactively identify and respond to issues, **Amazon CloudWatch Alarms** are configured to monitor critical metrics. Specifically, alarms are set to trigger when Lambda functions report errors (e.g., any `Errors` metric greater than 0) or when API Gateway experiences a high rate of client-side errors (e.g., `4XXError` count exceeding a threshold). When an alarm state is reached, a notification is sent to an **Amazon SNS topic**, which then dispatches alerts to the designated email address. This robust observability setup ensures that the development and operations team are immediately aware of any anomalies, enabling rapid diagnosis and resolution of potential issues.

9. Security Measures at all Layers:

Security is paramount for a financial application, and LoanSyncro implements a robust, multi-layered defense strategy across its entire architecture to protect sensitive user data and ensure operational integrity.

1. Frontend (AWS Amplify):

The client-side application, hosted on AWS Amplify, is served exclusively over HTTPS/SSL, guaranteeing that all data exchanged between the user's browser and the server is encrypted in transit. Amplify's managed hosting inherently includes DDoS protection and leverages Content Delivery Network (CDN). This not only improves performance by caching content closer to users but also enhances security by providing an additional layer of defence against various web attacks and reducing the direct attack surface on the origin server. User authentication tokens (JWTs) are securely managed by the Amplify SDK, which handles their storage in browser-appropriate mechanisms (e.g., local storage or session storage) to prevent direct exposure.

2. API Gateway:

Amazon API Gateway serves as the secure and controlled entry point for all backend API requests. It strictly enforces HTTPS endpoints, ensuring that all communication with the backend is encrypted. CORS (Cross-Origin Resource Sharing) headers are meticulously configured to permit requests only from the authorized frontend domain, effectively mitigating cross-site scripting (XSS) and other cross-origin attacks. API Gateway also provides built-in mechanisms for throttling and rate limiting, which are crucial for preventing abuse, brute-force attacks, and denial-of-service (DoS) attempts by controlling the volume of incoming requests.

3. Lambda Functions:

The core business logic resides within AWS Lambda functions, which are inherently secure due to their ephemeral and stateless nature, minimizing the attack surface. Each Lambda function operates under a precisely defined IAM Role that adheres to the principle of least privilege. This means the function is granted only the absolute minimum permissions necessary to interact with other AWS services (e.g., read/write to specific DynamoDB tables, publish to specific SNS topics), thereby limiting the blast radius in case of a compromise. Crucially, every incoming API request is subjected to rigorous JWT validation within the Lambda function itself. This involves verifying the token's signature against Cognito's public keys, checking its expiration, and confirming its

issuer and audience, ensuring that only authenticated and authorized users can execute backend operations. Sensitive configurations, such as database table names and Cognito user pool IDs, are securely passed to Lambda functions as environment variables, which are not directly accessible from the public internet.

4. **DynamoDB:**

   All data stored in Amazon DynamoDB tables (for users, loans, and repayments) is encrypted at rest using AWS Key Management Service (KMS). This ensures that data is encrypted before it's written to disk and automatically decrypted when accessed, providing robust protection against unauthorized access to the underlying storage. Data in transit between Lambda functions and DynamoDB is automatically encrypted via SSL/TLS by the AWS SDK. Access to DynamoDB tables is strictly controlled through the IAM roles assigned to the Lambda functions, ensuring that only the authorized backend services can perform read and write operations.

5. **S3:**

   The Amazon S3 bucket used for storing static frontend assets and Lambda deployment packages is configured with "Block Public Access" settings, preventing any unauthorized direct public access to its contents. Data stored within S3 is also encrypted at rest using AWS KMS, similar to DynamoDB. Granular S3 Bucket Policies and IAM Policies are applied to grant specific, limited access only to necessary AWS services (e.g., AWS Amplify for deploying frontend assets, Lambda for retrieving deployment packages). Versioning is enabled on the S3 bucket, providing a critical mechanism for data recovery in the event of accidental deletion or modification.

6. **Cognito:**

   AWS Cognito provides a fully managed and highly secure identity service. It offloads the burden of sensitive operations like password hashing and storage from the application code, handling them securely according to industry best practices. Cognito enforces strong password policies (e.g., minimum length, complexity requirements) to encourage users to create robust passwords. Email verification ensures that user accounts are tied to legitimate email addresses. Furthermore, Cognito supports Multi-Factor Authentication (MFA), which can be enabled to add an extra layer of security for user logins, requiring a second verification step beyond just a password.

By meticulously implementing these security measures across all layers, LoanSyncro is designed to provide a robust and trustworthy platform for managing sensitive financial information.

10. Cost Analysis & Optimization Strategies:

Services	Usage Estimation	Cost
AWS Lambda	1.5M invocations, 128 MB, 100 ms average	$0.70
AWS API Gateway	1.2M HTTP API calls	$1.00
AWS DynamoDB	3 GB storage, 300K reads/writes	$2.20
AWS S3	7 GB storage, 30K GET requests	$0.50
AWS SNS	200 SMS, 600 email notifications	$3.75
AWS Amplify	4 GB hosting, 200 build minutes	$1.50
Amazon Cognito	300 monthly active users	$2.00
Amazon CloudWatch	3 GB logs, basic monitoring	$3.00
AWS KMS	30,000 requests	$0.03
Total		**$14.68**

Table. 1. Final Cost Analysis of LoanSyncro

Optimization Strategies:

- Serverless First: The primary cost optimization is the choice of serverless services, eliminating fixed infrastructure costs.
- DynamoDB On-Demand: Avoids over-provisioning capacity, paying only for actual usage.
- S3 Lifecycle Policies: Automatically move data to cheaper storage tiers (Standard-IA, Glacier) as it ages.
- Lambda Memory Optimization: Continuously monitor Lambda function memory usage and adjust it to the lowest possible value that doesn't impact performance, as billing is based on GB-seconds.
- Log Retention: Configure appropriate log retention periods in CloudWatch to avoid accumulating unnecessary log data.
- Alerting Thresholds: Fine-tune CloudWatch alarm thresholds to avoid excessive SNS notifications, which incur small costs per message.
- Amplify Build Minutes: Optimize frontend build processes to reduce build times and associated costs.

11. Lessons Learned & Future Improvements:

Developing LoanSyncro using a serverless architecture on AWS provided invaluable insights into modern cloud application development. The project reinforced the immense benefits of serverless computing, particularly in terms of automatic scalability, reduced operational overhead, and inherent cost efficiency due to the pay-per-use model, allowing the team to focus on application logic rather than infrastructure management. While powerful, integrating diverse AWS services effectively (e.g., Cognito with API Gateway and Lambda, Lambda with DynamoDB and SNS) required a deep understanding of each service's nuances and how they interact, highlighting the critical need for careful configuration of IAM roles, environment variables, and API Gateway routes. Furthermore, the implementation of secure user authentication and authorization using AWS Cognito and JWT validation in Lambda was a significant learning curve, emphasizing the importance of proper token handling, public key management, and adherence to security best practices to protect sensitive user data. Finally, utilizing Terraform for defining and deploying the entire infrastructure proved indispensable, ensuring consistency, repeatability, and version control for the cloud environment, making deployments predictable and reducing manual configuration errors.

Improvements –

LoanSyncro, while functional, has significant potential for expansion and enhancement to provide a more comprehensive and user-friendly experience:

- Advanced Loan Features: Implement detailed amortization schedules for each loan, showing the breakdown of principal and interest for every payment, and introduce support for variable interest rates to provide more granular financial tracking.
- Enhanced Repayment Management: Allow users to set up recurring repayment entries for automated tracking and accurately handle partial payments or overpayments, adjusting the remaining balance accordingly.
- Interactive Reporting & Analytics: Integrate more dynamic charts and graphs on the dashboard and loan detail pages to visually represent loan progress, interest vs. principal paid, and overall financial health, along with options to export data.
- External Integrations: Explore secure integrations with financial APIs (e.g., Plaid) to automatically import loan data and payment history from banks, significantly reducing manual data entry and enhancing data accuracy.

12. Conclusion & References:

LoanSyncro stands as a robust and scalable financial management application, successfully demonstrating the practical application of modern cloud architectural principles. By leveraging AWS serverless services such as Lambda, API Gateway, DynamoDB, S3, Cognito, and SNS, the project has delivered a solution that is inherently scalable, highly available, and cost-efficient. The implementation effectively addresses core functional requirements, enabling users to securely manage their loans and repayments with ease, while also fulfilling critical non-functional requirements related to security, performance, and observability. The journey of developing LoanSyncro has provided invaluable lessons in integrating diverse cloud services and adhering to best practices, laying a strong foundation for future enhancements and a deeper dive into advanced financial management features.

References –

[1] Amazon Web Services, Inc., "AWS Well-Architected Framework," [Online]. Available:https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html. [Accessed: 06-Jun-2025].

[2] "Build a Loan Management System | Features & Best Practices," Logic-Square, blog, Apr. 2025. [Online]. Available: https://logic-square.com/build-loan-management-system-features-best-practices/logic-square.com. [Accessed: 06-Jun-2025].

[3] Amazon Web Services, "Optimizing Enterprise Economics with Serverless Architectures," Amazon Web Services, n.d. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/optimizing-enterprise-economics-with-serverless/understanding-serverless-architectures.html [Accessed: 15-Jun-2025].

[4] Amazon Web Services, "AWS Key Management Service Developer Guide," Amazon Web Services, n.d. [Online]. Available: https://docs.aws.amazon.com/kms/latest/developerguide/overview.html [Accessed: 18-June-2025].