

BANKING-APPLICATION

This document describes the complete **microservices architecture** of the Banking Application, including Discovery and Config servers, API Gateway, Auth, Customer, Account, Payment, Audit, Feedback, and Notification services. It also explains Angular frontend integration, Kafka messaging, and deployment architecture.

GlobalSystemOverview:

The Banking Application is implemented as a microservices ecosystem: - Angular Frontend for user interface - API Gateway as a single-entry point - Discovery Server for service registry - Config Server for centralized configurations - Domain-driven microservices (Customer, Account, Payment, etc.) - Kafka and Zookeeper for messaging.

Discovery & Config Services- Discovery Service (Eureka): Registers and discovers microservices dynamically. - Simplifies API Gateway routing by avoiding hard-coded URLs. - Enables load balancing. Config Server: Centralized configuration management for all microservices. - Externalized properties for different environments (dev, QA, prod). - Integrated with Git for version control.

API Gateway- Acts as a single entry point for client (Angular) requests. - Performs authentication, routing, and load balancing. - Handles CORS, rate limiting, and logging.

Auth Service- Provides login, registration, and JWT token generation. - Integrates with API Gateway to enforce authorization policies. - Uses Spring Security for authentication.

Customer Service- Stores and manages customer profiles and KYC information. - Supports CRUD operations via REST endpoints. - Communicates with AuditService to log all profile updates.

Account Service- Manages account data including balances and account types. - Exposes APIs for fetching and updating balances. - Called by PaymentService for fund validation.

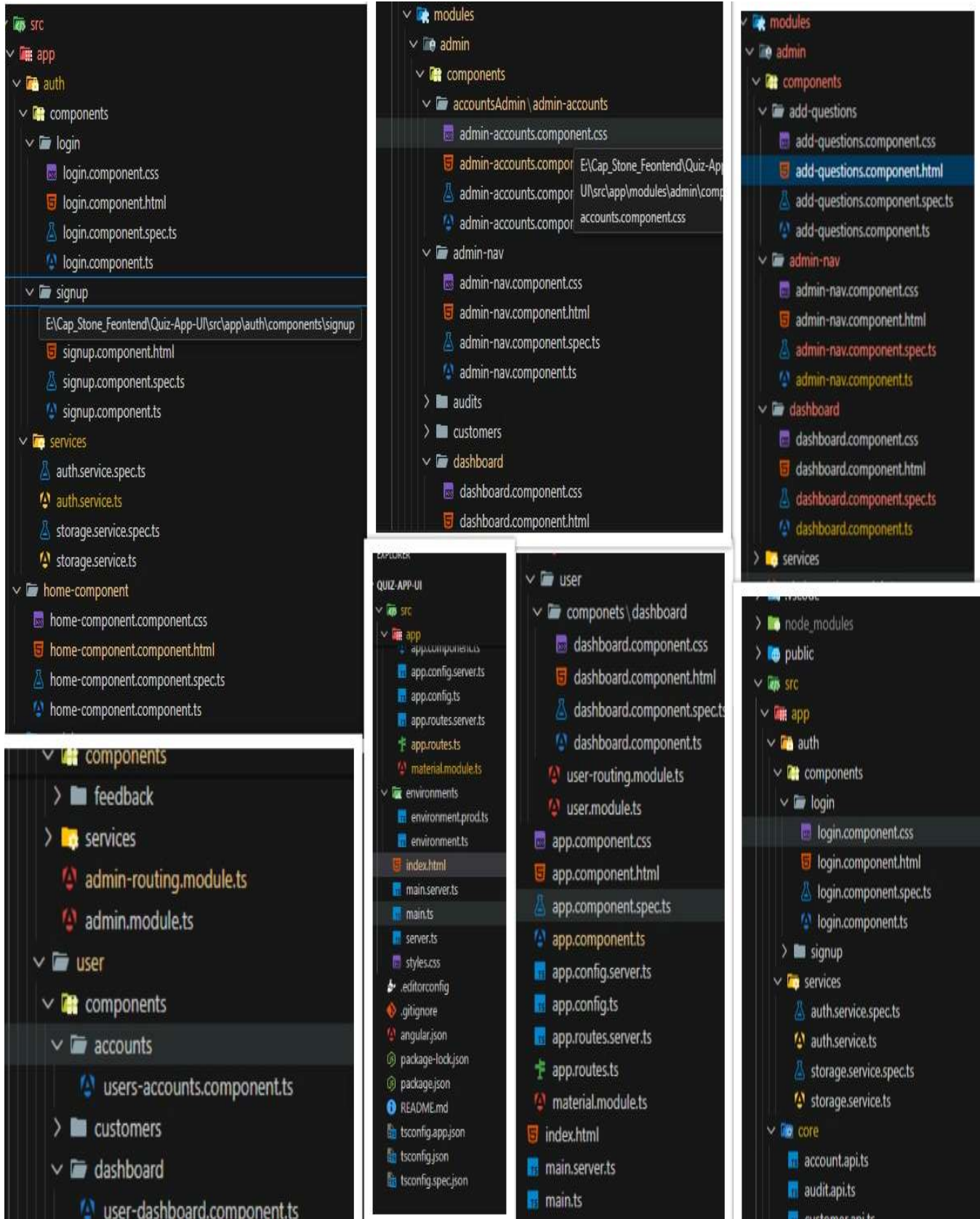
Payment Service- Handles fund transfers and payment workflows. - Communicates with AccountService, AuditService, and NotificationService. - Publishes Kafka events for asynchronous notifications.

Audit Service- Maintains logs for all critical actions and transactions. - Provides APIs for querying audit records. Integrated with PaymentService and CustomerService. Feedback Service- Collects and stores customer feedback. - Communicates with NotificationService to send alerts based on negative feedback. Notification Service- Sends notifications via email. - Listens to Kafka events (e.g., payment confirmation)

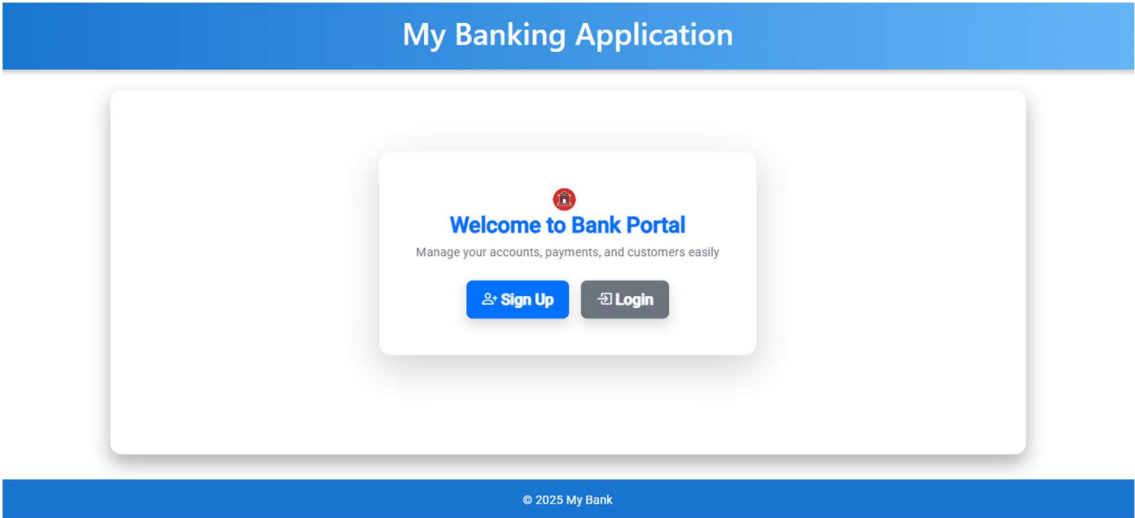
Angular-Backend Communication Flow 1. Angular sends authentication request via API Gateway. 2. JWT token is issued by AuthService and stored by Angular. 3. Angular makes secured requests to Gateway with token headers. 4. Gateway routes requests to respective services (Customer, Payment, etc.). 5. Kafka events notify NotificationService asynchronously.

ANGULAR VIEW OUTPUTS

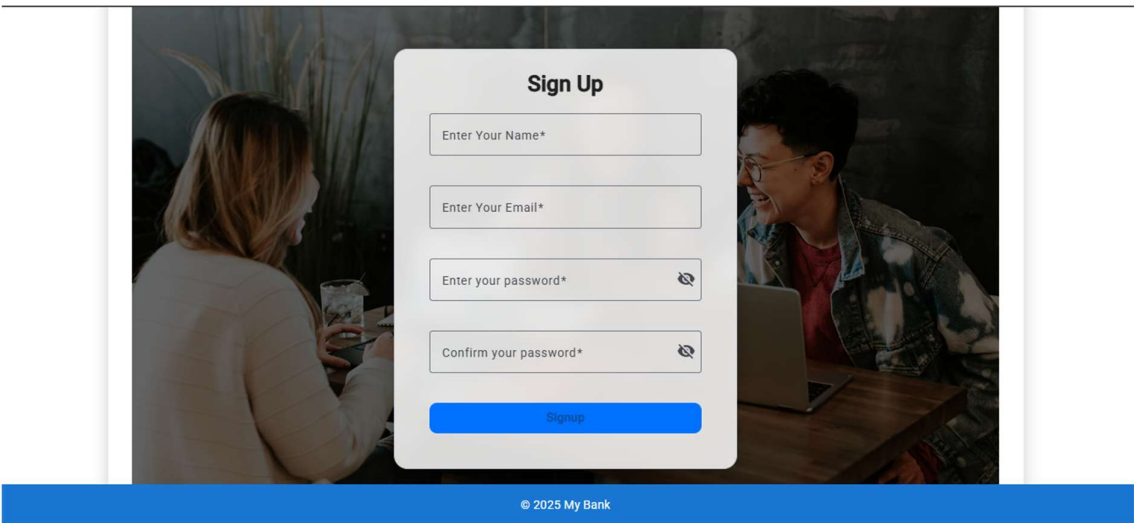
Folder Structure:



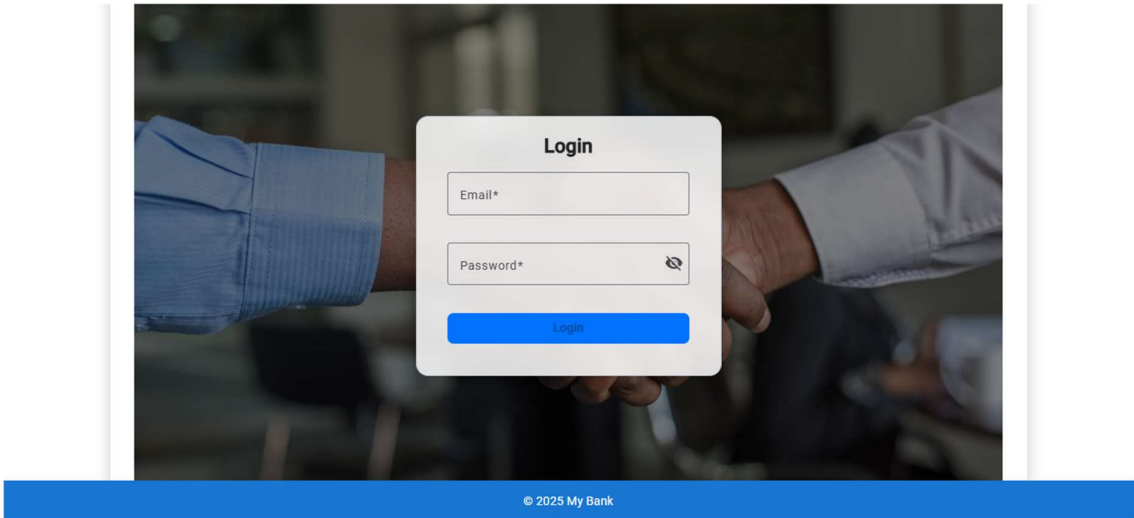
HOME PAGE:



SIGNUP PAGE:



LOGIN PAGE:



Dashboard

Accounts

Customers

Feedbacks

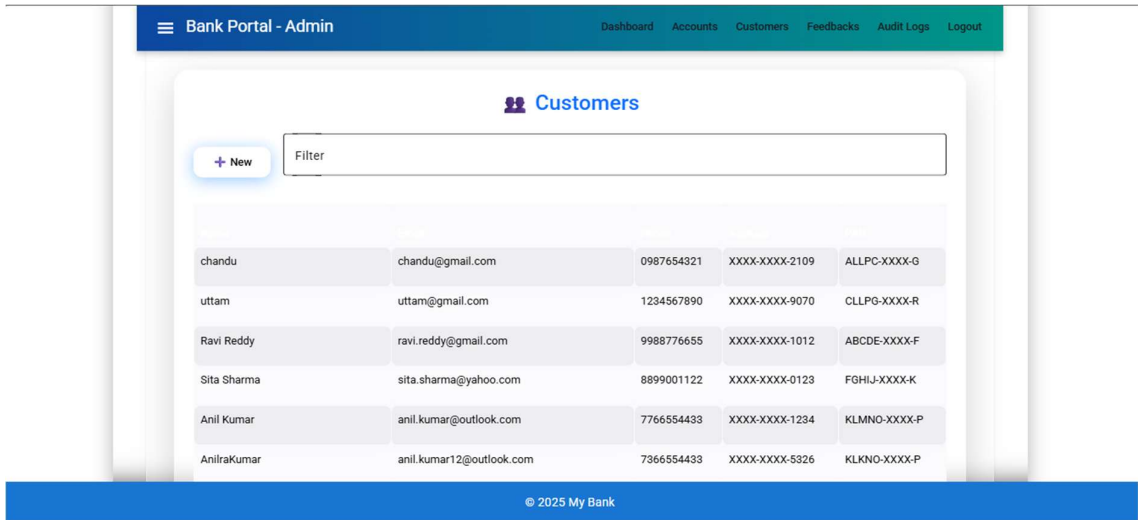
Audit Logs

Logout

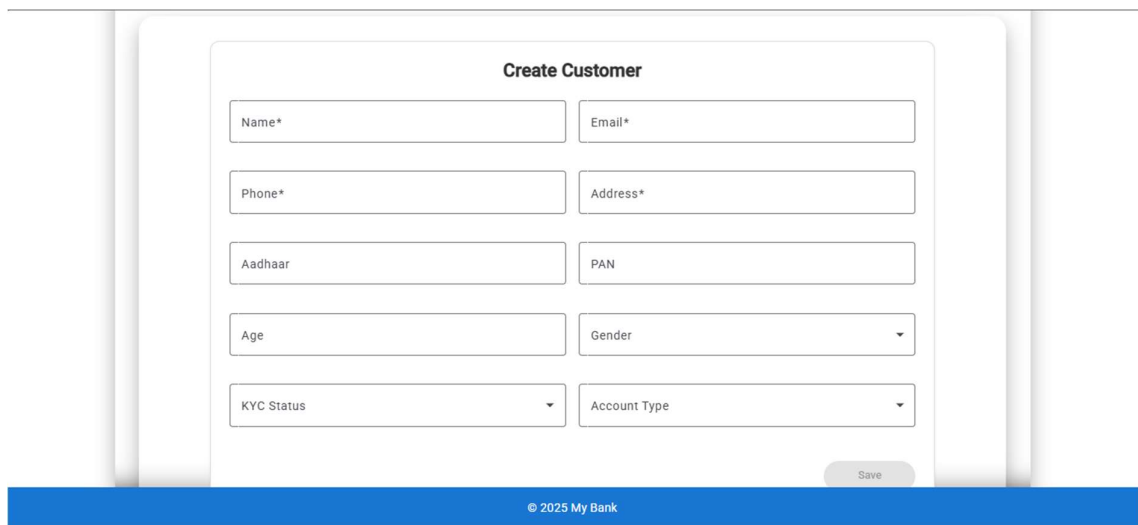
ALL BANK ACCOUNTS

ID	Name	PAN	Aadhaar	Account No	Bank Type	Balance	Loan	Address	Action
1	Uttam@Tss	CLLPN9488B	123456789012	3222312312	SAVINGS	₹175.00	₹1.00	tenalis	Edit
2	Chanduss	ASSDR7654G	098765432123	1234567890	SAVINGS	₹5,829.00	₹200.00	america	Edit
3	Ravi	QWERT1234Y	112233445566	9876543210	CURRENT	₹1,000.00	₹5,000.00	Hyderabad	Edit
4	Sita	ZXCVB5678U	223344556677	1234509876	SAVINGS	₹8,000.00	₹2,000.00	Bangalore	Edit
5	Anil	LKJHG9012P	334455667788	5678901234	CURRENT	₹5,000.00	₹0.00	palanadu	Edit
6	Anil	LKJHG9012P	334455667788	5678901234	CURRENT	₹5,000.00	₹0.00	palakollu	Edit

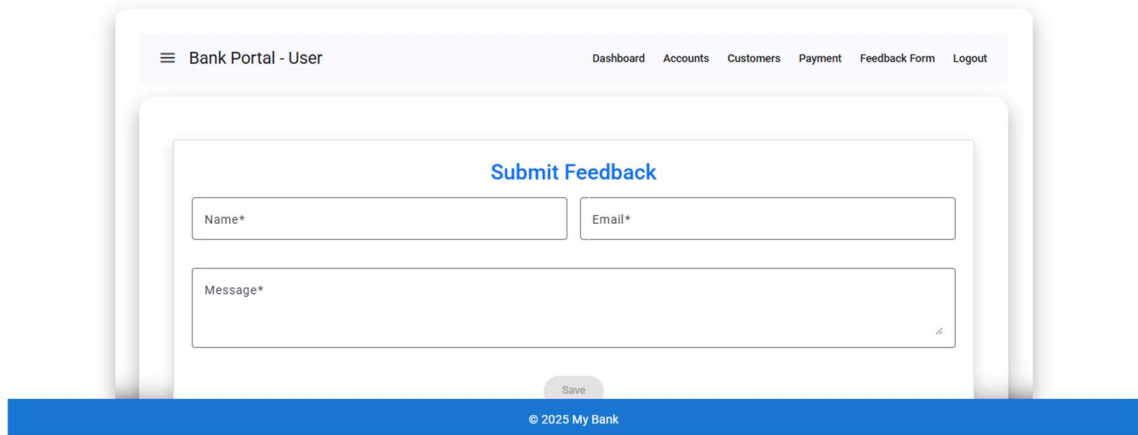
© 2025 My Bank



USER COMPONENTS VIEWS:



My Banking Application



My Banking Application

Bank Portal - User

Dashboard Accounts Customers Payment Feedback Form Logout

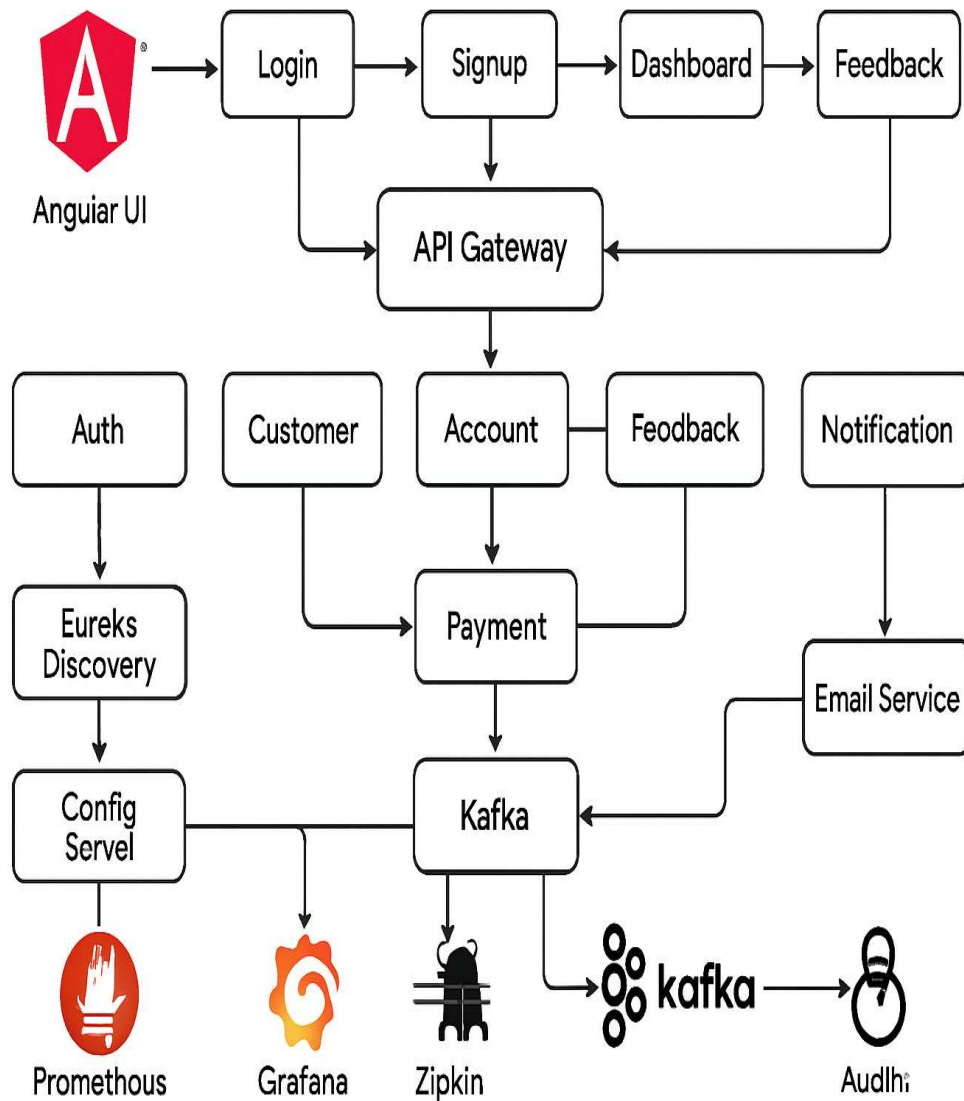
Payments

Sender ID* Receiver ID* Amount* 1

Transfer

© 2025 My Bank

ARCHITECTURE:



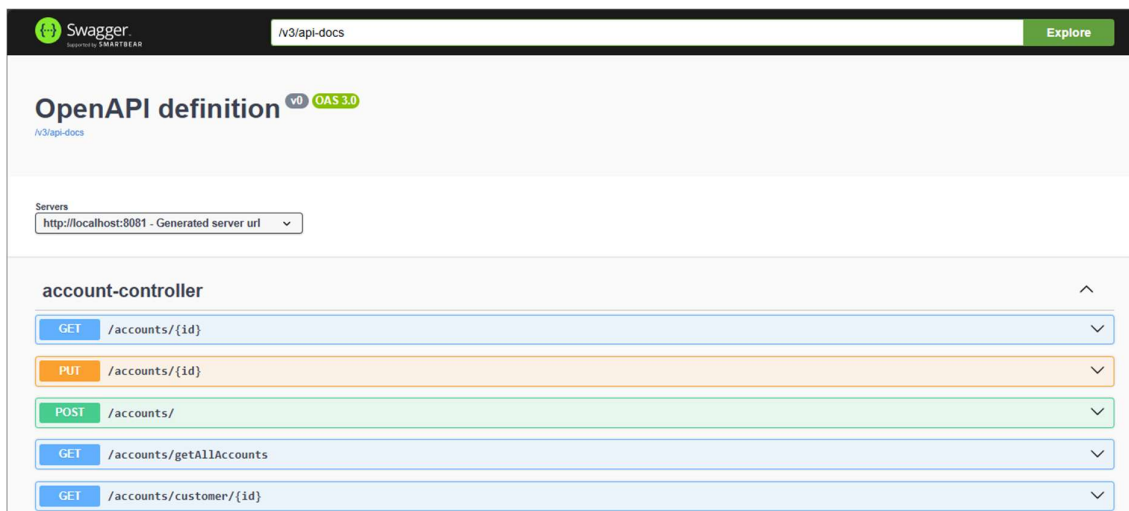
JAVA-SPRING BOOT MICROSERVICES:

SERVICES:

- AccountService [devtools]
- ApiGateway
- AuditService [devtools]
- ConfigServer
- CustomerService [devtools]
- DiscoveryServices
- FeedbackService [devtools]
- NotificationService
- PaymentService [devtools]
- UserAuthService [devtools]

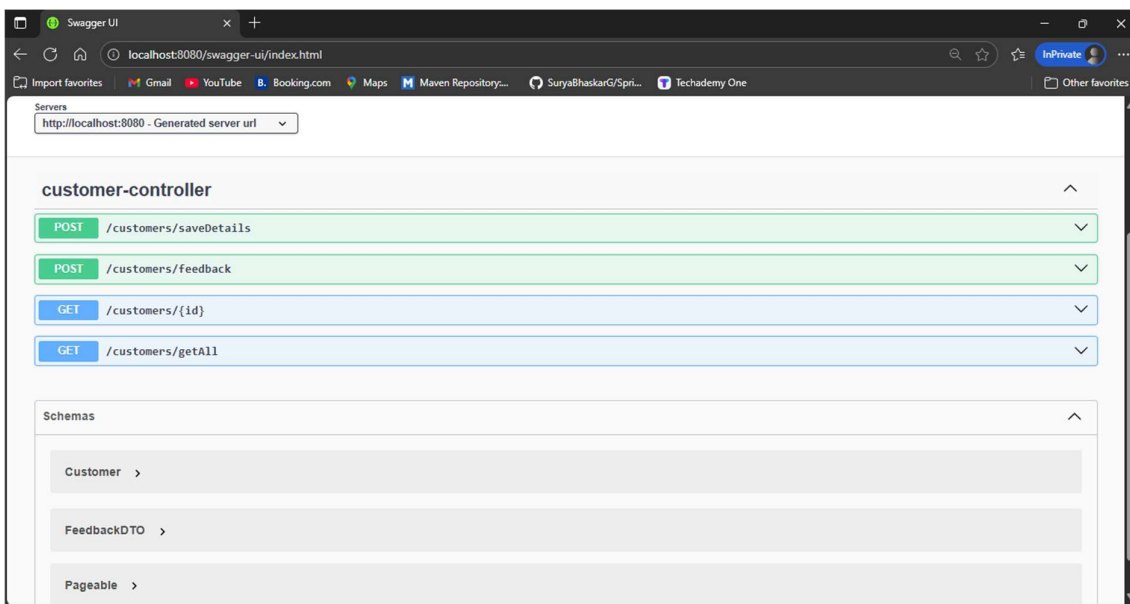
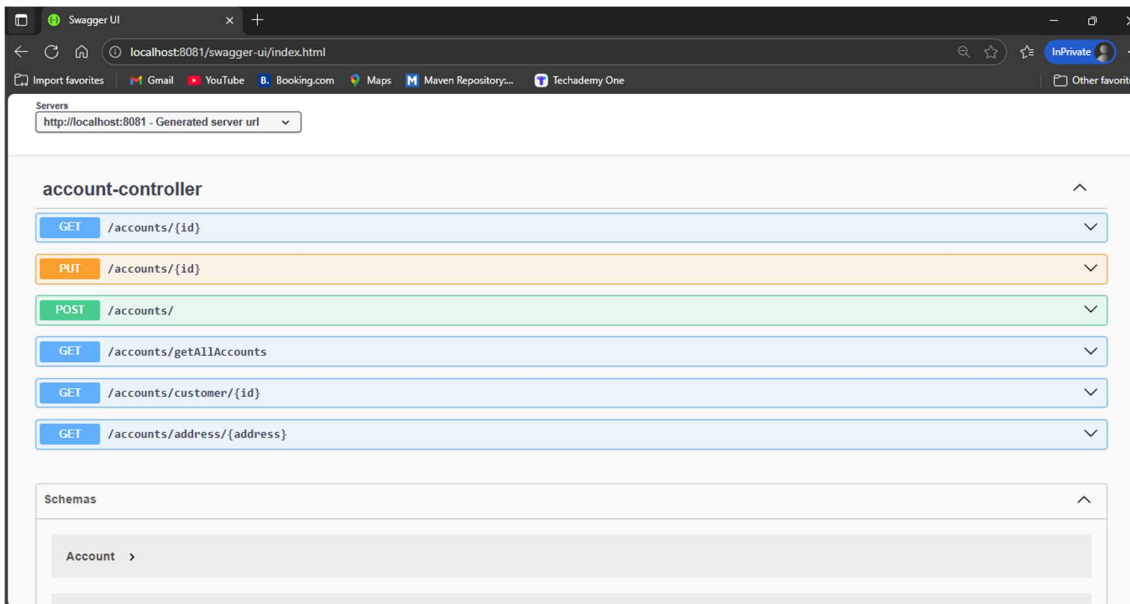
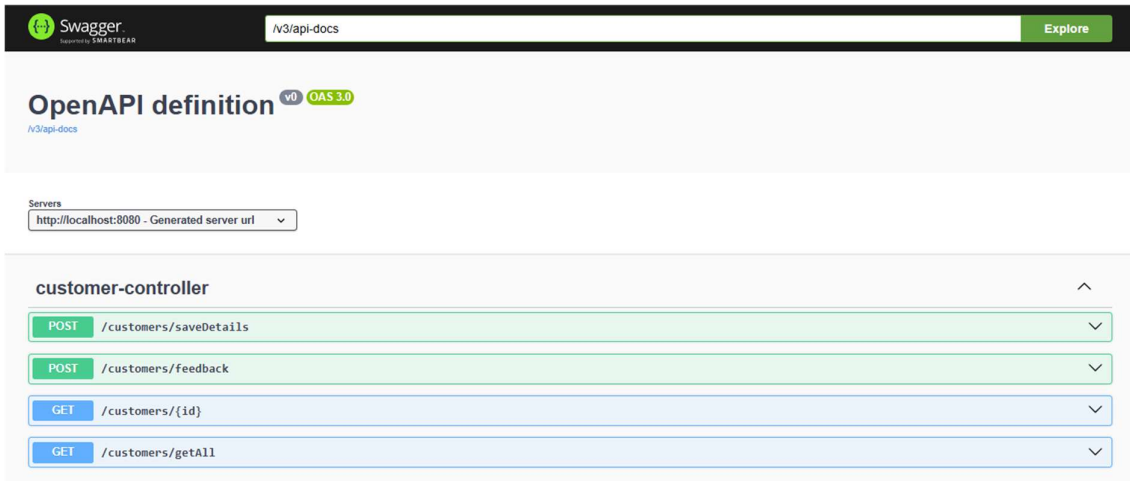
Service	Responsibility	Database
API Gateway	Routes client requests, handles load balancing	N/A
AuthService	User authentication, JWT issuance	AuthDB
AccountService	Account data and balance management	AccountDB
CustomerService	Customer profile and KYC data	CustomerDB
PaymentService	Handles payments, transaction workflow	PaymentDB
FeedbackService	User feedback storage and reporting	FeedbackDB
NotificationService	Async notifications via Kafka	NotificationDB
AuditService	Transaction logs, audit trails	AuditDB

SWAGGER and ZIPKIN and PROMETHEUS and GRAFANA :



The screenshot displays the Swagger UI for an API. At the top, there's a search bar with the text "/v3/api-docs" and an "Explore" button. Below this, the "OpenAPI definition" is shown, with a version indicator "v0 OAS 3.0". A "Servers" dropdown menu is set to "http://localhost:8081 - Generated server url". The main content area lists endpoints under the "account-controller" group:

- GET /accounts/{id}
- PUT /accounts/{id}
- POST /accounts/
- GET /accounts/getAllAccounts
- GET /accounts/customer/{id}



Swagger UI

localhost:8080/swagger-ui/index.html

Servers

http://localhost:8080 - Generated server url

customer-controller

- POST /customers/saveDetails
- POST /customers/feedback
- GET /customers/{id}
- GET /customers/getAll

Schemas

- Customer >
- FeedbackDTO >
- Pageable >

Swagger UI

localhost:8082/swagger-ui/index.html

Servers

http://localhost:8082 - Generated server url

payment-controller

- POST /payments/transfer

Schemas

- PaymentRequest >
- PaymentResponseDTO >

Swagger UI

localhost:8083/swagger-ui/index.html

Servers

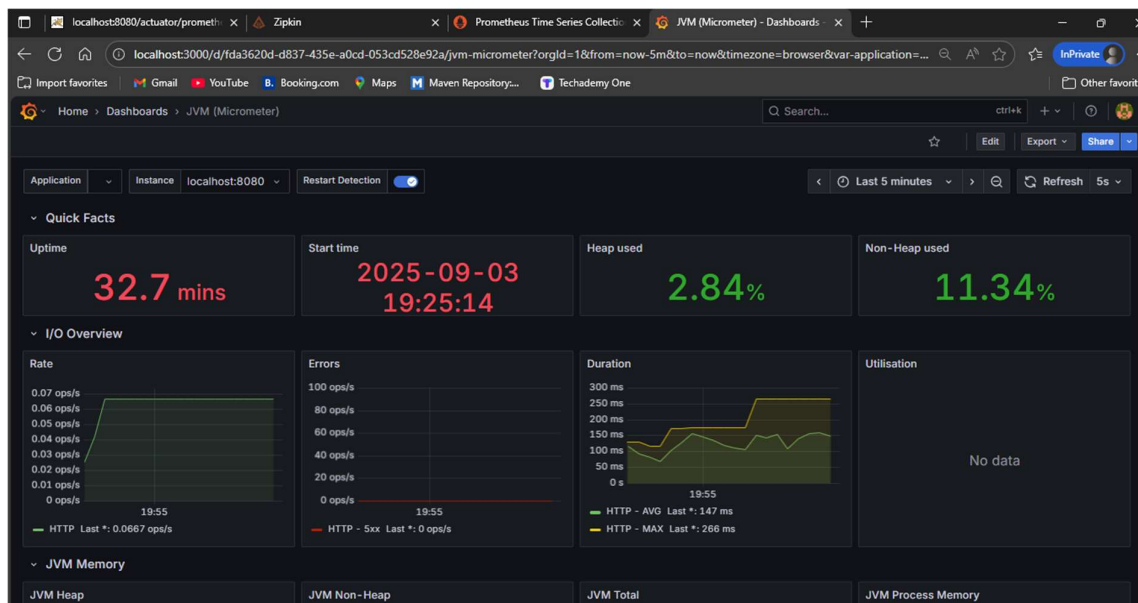
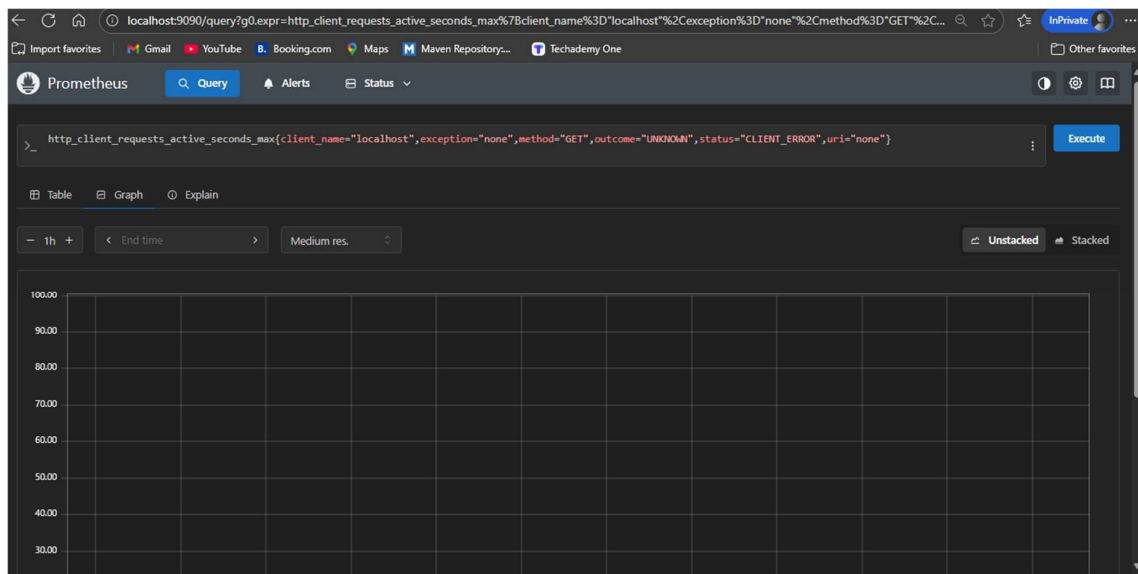
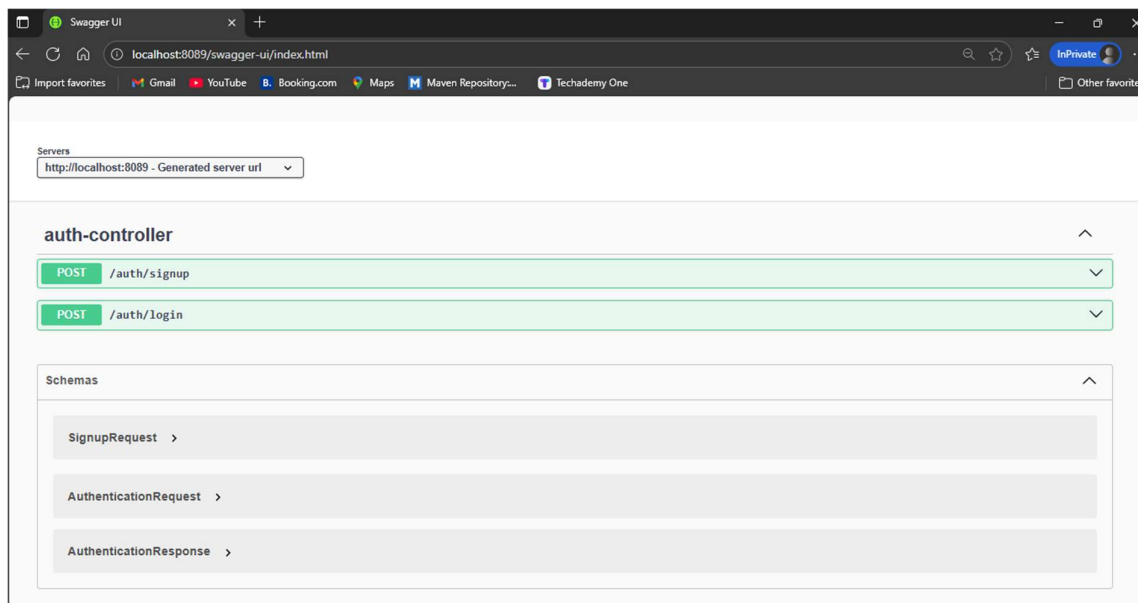
http://localhost:8083 - Generated server url

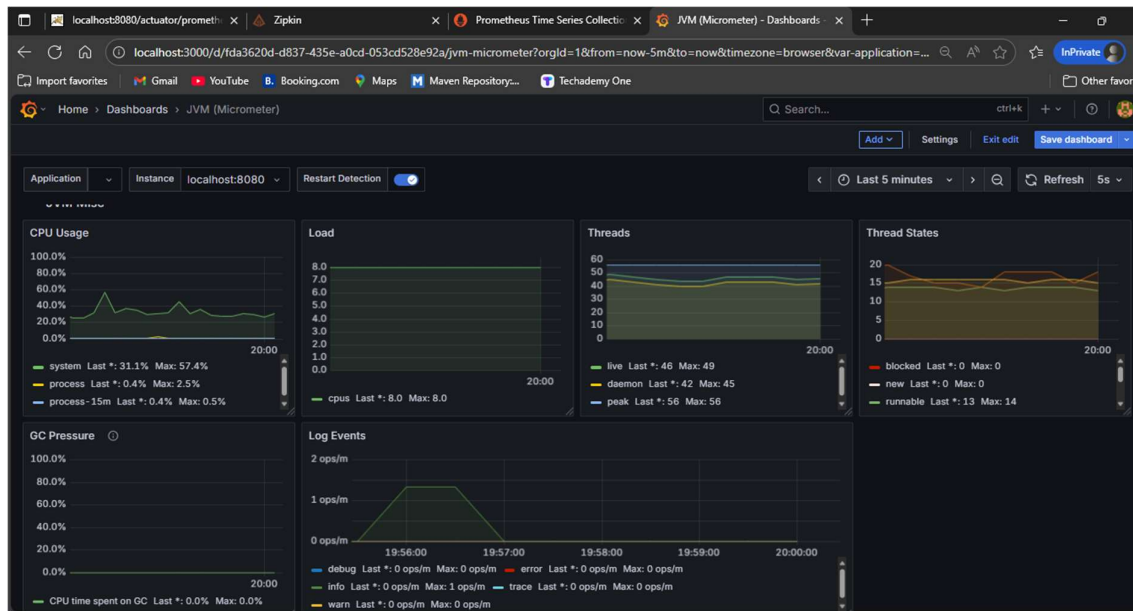
audit-log-controller

- POST /audits/log
- GET /audits/getAll

Schemas

- AuditLog >





Payment Notification Inbox x



usermergaa@gmail.com

to uttam

Payment of amount 4000 from user 6 to user 2 is SUCCESS

OVERVIEW:

The Banking Web Application project demonstrates a robust, microservices-based architecture built with Angular and Spring Boot. Core functionalities, including customer management, accounts, payments, feedback, audit, and notifications, are seamlessly integrated, with Kafka enabling asynchronous communication between Payment and Notification services and Feign Clients facilitating synchronous inter-service calls. Supported by API Gateway, Config Server, and Discovery Service, the system is secure, scalable, and maintainable. This project highlights modern enterprise-level development practices and provides a solid foundation for real-world banking solutions.

Acknowledgment / Conclusion:

This project has been completed successfully under the guidance of **Ramakrishna Sir**. I have applied my knowledge of **Angular, Spring Boot, and Microservices** to develop a scalable and secure Banking Web Application. I hope this project meets the objectives and expectations outlined at the start.

Submitted By:

Uttam Kumar Narra

uttamnarra1@gmail.com

**BATCH _II
ANGULAR**