BANKING MICROSERVICES APPLICATION

Name: DANDE BALAJI

Email: balajidande7@gmail.com

UNDER THE GUIDANCE OF

Mr. RAMAKRISHNA

1. Introduction

The **Banking Microservices Application** is a **modern, secure, and scalable mobile banking platform** built using **microservices architecture**. It is designed to provide customers with seamless and reliable banking services while ensuring **high performance, security, and maintainability**.

This application supports a wide range of banking operations, including:

- **Customer onboarding & authentication** – Quick and secure registration and login for customers.
- **Account management** – Creation and management of Savings and Current accounts.
- **Money transfers** – Perform internal and external fund transfers efficiently.
- **Transaction history & statements** – View detailed transaction records and download statements.
- **Notifications** – Receive real-time alerts via SMS and Email for important activities.
- **Audit logging & reporting** – Track and report user actions for compliance and monitoring.

To ensure **scalability and resilience**, the platform uses:

- **Service Discovery** – Automatically detects available services.
- **Config Server** – Centralized management of configuration across environments.
- **API Gateway** – Provides secure and controlled access to all microservices.

This architecture makes the platform flexible, maintainable, and enterprise-ready, capable of handling high transaction volumes while providing real-time monitoring and observability.

2. Table of Contents

2. Technologies Used:

Tech Stack Overview (with Icons)

Below is the refined tech stack summary:

1. **Spring Boot** – Backend microservices framework
2. **Angular** – Frontend SPA (Single Page Application) framework
3. **MySQL** – Relational database for transactional data
4. **Apache Kafka** – Stream/event messaging backbone
5. **Bootstrap** – UI styling & responsive design (via Angular)
6. **JWT / OAuth2** – Security protocols for authentication & authorization

3. Key Features

- Service Discovery (Eureka/Consul)
- Centralized Config Server (Spring Cloud Config)
- API Gateway (Spring Cloud Gateway)
- Customer onboarding & KYC verification
- Account creation & lifecycle management
- Payment Service - Internal & external fund transfers
- Transaction history with statements download
- Real-time notifications (SMS/Email)
- Audit logging & compliance reporting

4. Spring Boot REST API Best Practices

1. API Design

- Use proper **HTTP methods**:
    - GET → Fetch resources
    - POST → Create resources
    - PUT → Update entire resource
    - PATCH → Partial update
    - DELETE → Remove resource
- Follow **RESTful naming conventions**:
    - Example: /api/customers

- Return **proper HTTP status codes**:
    - 200 OK → Successful request
    - 201 CREATED → Resource created
    - 400 BAD REQUEST → Invalid request data
    - 404 NOT FOUND → Resource not found
    - 500 INTERNAL SERVER ERROR → Unexpected errors

2. Data & DTO Handling

- Use **Entity classes** only for persistence (JPA/Hibernate).
- Use **DTOs (Data Transfer Objects)** for API requests/responses.
- Avoid exposing entities directly to the outside world.

3. Exception Handling

- Implement **Global Exception Handling** using:
    - @ControllerAdvice
    - @ExceptionHandler
- Create **custom exceptions** for business logic errors.
- Return consistent error responses (with error code, message, timestamp).

4. Validation

- Use **@Valid** with request DTOs.
- Add **Hibernate Validator annotations** (e.g., @NotNull, @Email, @Size).
- Create **custom validators** for business rules.

5. Logging & Monitoring

- Use **SLF4J with Lombok's @Slf4j** for logging.
- Avoid logging sensitive data (like passwords, tokens).
- Use **Spring Boot Actuator** for health checks & monitoring.
- Integrate **Micrometer + Prometheus + Grafana** for metrics visualization.

6. Security

- Use **Spring Security** with **JWT/OAuth2** for authentication & authorization.
- Always enforce **HTTPS** in production.
- Apply **role-based access control (RBAC)**.
- Secure endpoints behind an **API Gateway**.

7. Performance & Scalability

- Enable **pagination & sorting** for large collections (Pageable).
- Use **caching** (e.g., Spring Cache, Redis) for frequently accessed data.
- Make **asynchronous calls** where applicable.
- Optimize DB queries (indexes, batch operations).

8. Documentation

- Use **Swagger / OpenAPI 3.0** for interactive API documentation.
- Provide clear examples of requests/responses.
- Version APIs (/api/v1/...) to support backward compatibility.

9. Testing

- Use **JUnit 5 + Mockito** for unit testing.
- Use **Spring Boot Test** (@SpringBootTest) for integration testing.
- Maintain **Postman collections** for manual testing.

10. Code Quality & Structure

- Follow **layered architecture**:
    - **Controller → Service → Repository**
- Use **proper packaging structure**:
    - controller, service, repository, dto, config, exception, etc.
- Use **Lombok** (@Getter, @Setter, @Builder, @Slf4j) to reduce boilerplate code.

5. Security & Authentication

- Implemented with **Spring Security**, **JWT**, and **OAuth2**
- Role-based access control (Customer, Admin, Auditor)
- Encrypted inter-service communication
- Secured API endpoints via API Gateway

6. Notifications & Audit Logging

6.1 Notifications

- Implemented **real-time SMS and Email notifications** for:
  - Account creation & onboarding confirmation
  - Successful/failed money transfers
  - Balance updates & transaction alerts
- Notifications are sent **asynchronously** using **Apache Kafka** to avoid delays in core banking operations.
- **Kafka Topics** used:
  - transaction-events → triggers customer transaction alerts
  - account-events → account opening/closure notifications

6.2 Audit Logging

- All critical operations (logins, fund transfers, account updates) are **logged in an Audit Service**.
- Audit logs stored in **MongoDB** for:
  - Compliance reporting
  - User activity tracking
  - Fraud detection & anomaly analysis
- **Kafka Topics** used:
  - audit-events → streams every important action to Audit Service
- Ensures **non-blocking event-driven logging**, improving performance while maintaining traceability.

Role of Kafka (Asynchronous Communication)

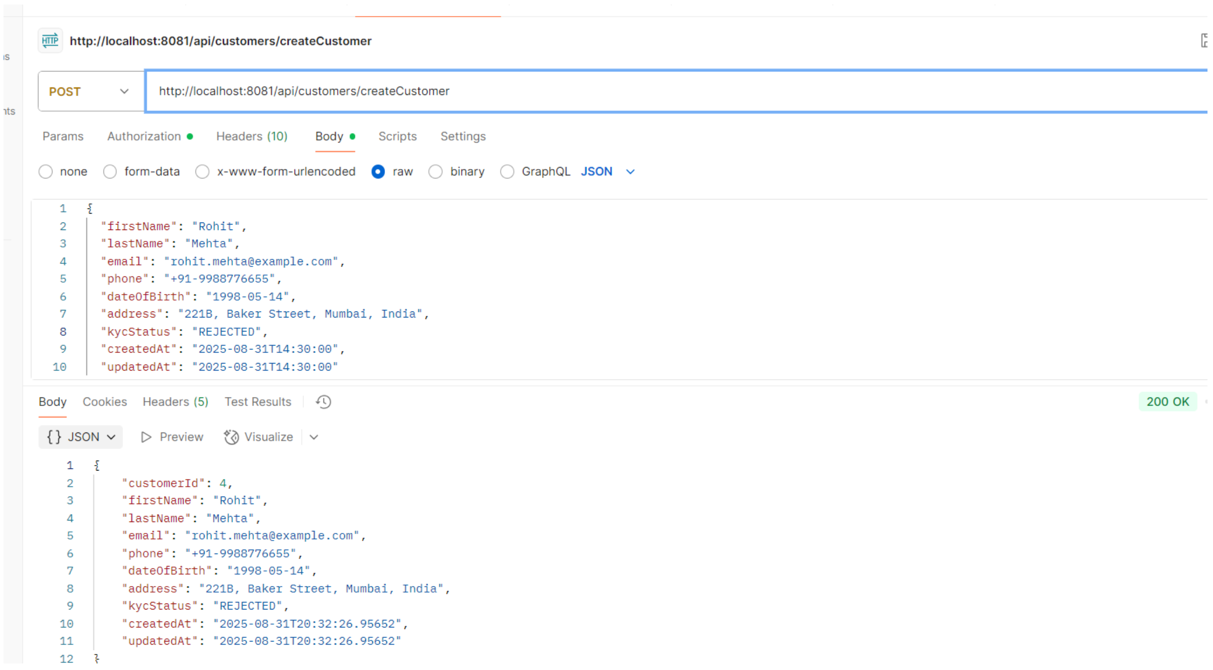
- **Decouples services** (e.g., Transaction Service doesn’t directly call Notification Service).
- Increases **scalability** → Notification/Audit services can scale independently.
- Ensures **reliability** → Events are not lost even if a service is temporarily down.
- Improves **user experience** → Core services respond quickly without waiting for notification/audit completion.

5.Micoservices Overview

- Customer Service
- Account Service
- Transaction Service
- Payment Service

5.1 **Customer Service end points:**

- Create Customer



http://localhost:8081/api/customers/createCustomer

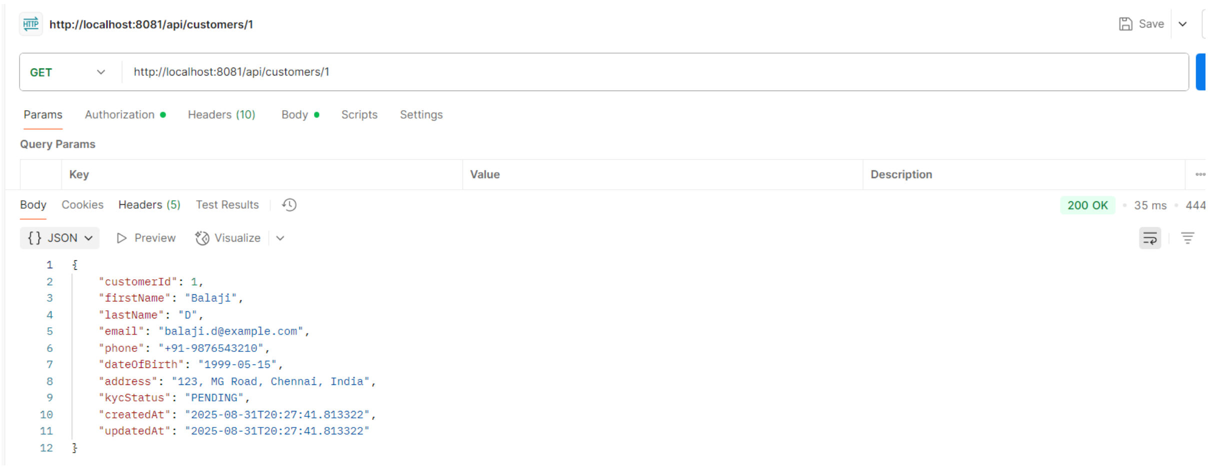
POST http://localhost:8081/api/customers/createCustomer

```
{
"firstName": "Rohit",
"lastName": "Mehta",
"email": "rohit.mehta@example.com",
"phone": "+91-9988776655",
"dateOfBirth": "1998-05-14",
"address": "221B, Baker Street, Mumbai, India",
"kycStatus": "REJECTED",
"createdAt": "2025-08-31T14:30:00",
"updatedAt": "2025-08-31T14:30:00"
```

200 OK

```
{
    "customerId": 4,
    "firstName": "Rohit",
    "lastName": "Mehta",
    "email": "rohit.mehta@example.com",
    "phone": "+91-9988776655",
    "dateOfBirth": "1998-05-14",
    "address": "221B, Baker Street, Mumbai, India",
    "kycStatus": "REJECTED",
    "createdAt": "2025-08-31T20:32:26.95652",
    "updatedAt": "2025-08-31T20:32:26.95652"
}
```

getCustomerById/{id}



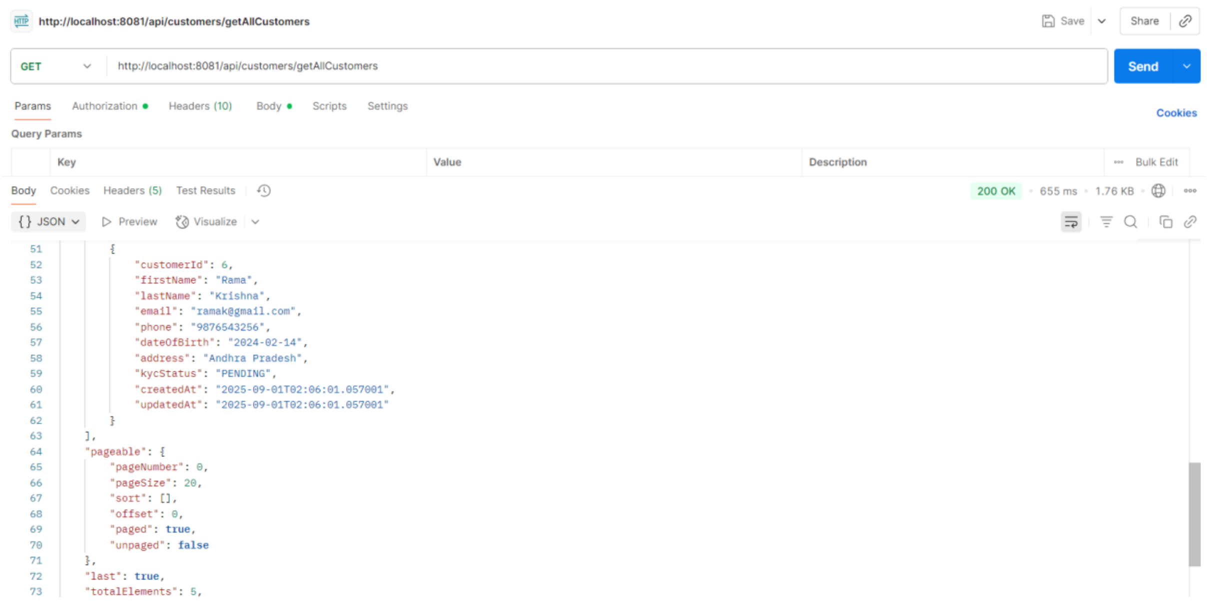
http://localhost:8081/api/customers/1

GET http://localhost:8081/api/customers/1

200 OK · 35 ms · 444

```
{
    "customerId": 1,
    "firstName": "Balaji",
    "lastName": "D",
    "email": "balaji.d@example.com",
    "phone": "+91-9876543210",
    "dateOfBirth": "1999-05-15",
    "address": "123, MG Road, Chennai, India",
    "kycStatus": "PENDING",
    "createdAt": "2025-08-31T20:27:41.813322",
    "updatedAt": "2025-08-31T20:27:41.813322"
}
```

--getAllCustomers


http://localhost:8081/api/customers/getAllCustomers

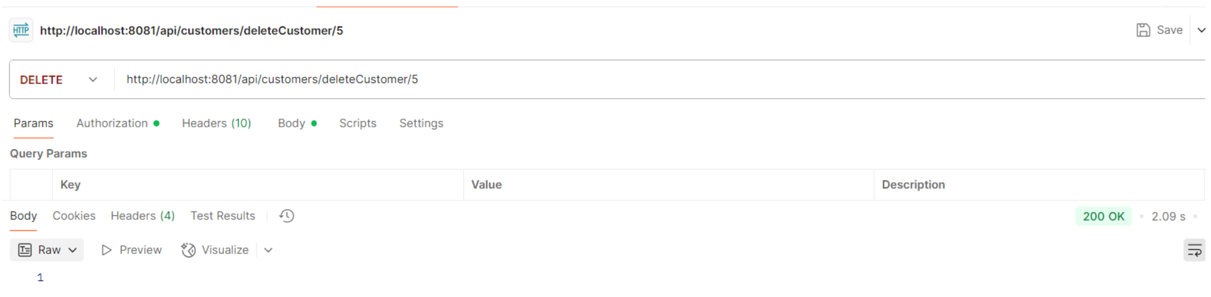
GET http://localhost:8081/api/customers/getAllCustomers

Query Params: Key | Value | Description

200 OK · 655 ms · 1.76 KB

```
        {
            "customerId": 6,
            "firstName": "Rama",
            "lastName": "Krishna",
            "email": "ramak@gmail.com",
            "phone": "9876543256",
            "dateOfBirth": "2024-02-14",
            "address": "Andhra Pradesh",
            "kycStatus": "PENDING",
            "createdAt": "2025-09-01T02:06:01.057001",
            "updatedAt": "2025-09-01T02:06:01.057001"
        }
    ],
    "pageable": {
        "pageNumber": 0,
        "pageSize": 20,
        "sort": [],
        "offset": 0,
        "paged": true,
        "unpaged": false
    },
    "last": true,
    "totalElements": 5,
```

--deleteCustomerById


http://localhost:8081/api/customers/deleteCustomer/5

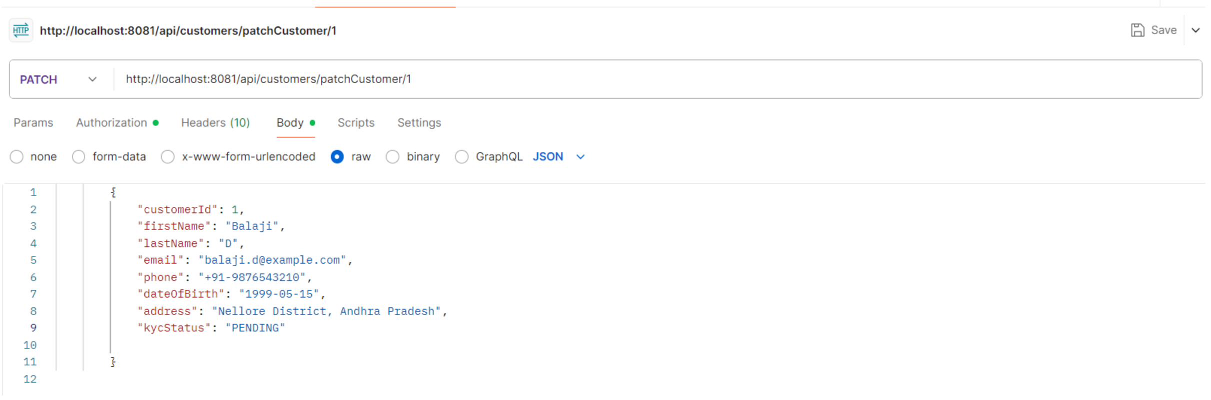
DELETE http://localhost:8081/api/customers/deleteCustomer/5

Query Params: Key | Value | Description

200 OK · 2.09 s

```
1
```

--patchCustomerById


http://localhost:8081/api/customers/patchCustomer/1

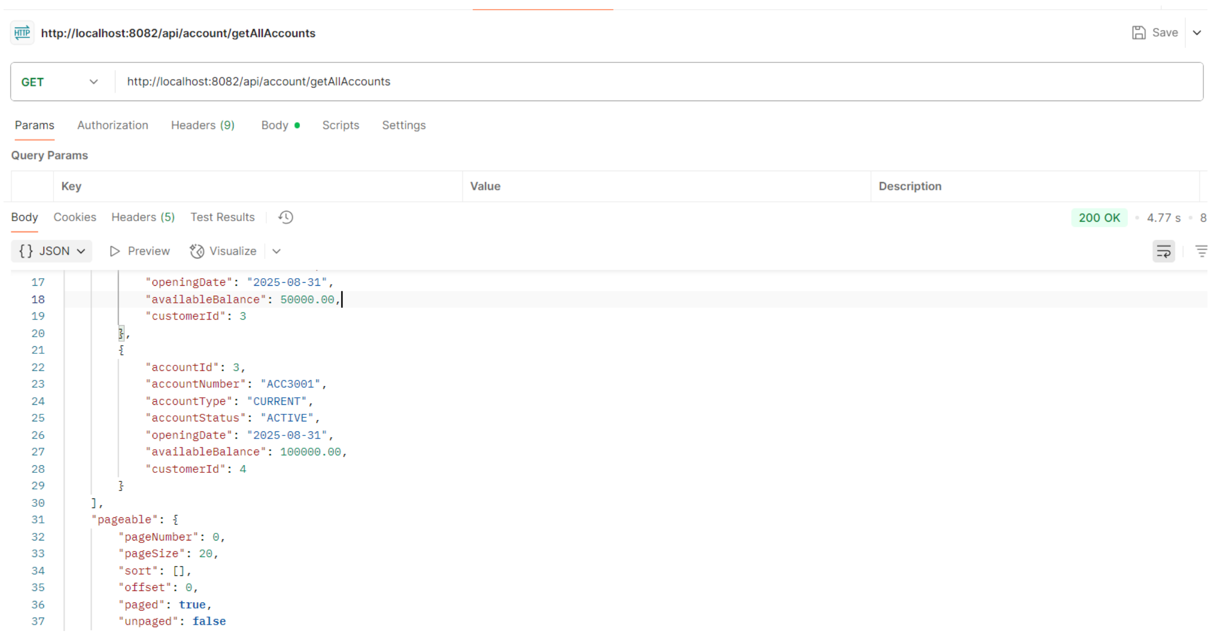
PATCH http://localhost:8081/api/customers/patchCustomer/1

```
{
    "customerId": 1,
    "firstName": "Balaji",
    "lastName": "D",
    "email": "balaji.d@example.com",
    "phone": "+91-9876543210",
    "dateOfBirth": "1999-05-15",
    "address": "Nellore District, Andhra Pradesh",
    "kycStatus": "PENDING"

}
```

5.2 Account Service

--getAllAccounts



http://localhost:8082/api/account/getAllAccounts

GET http://localhost:8082/api/account/getAllAccounts

Params Authorization Headers (9) Body Scripts Settings

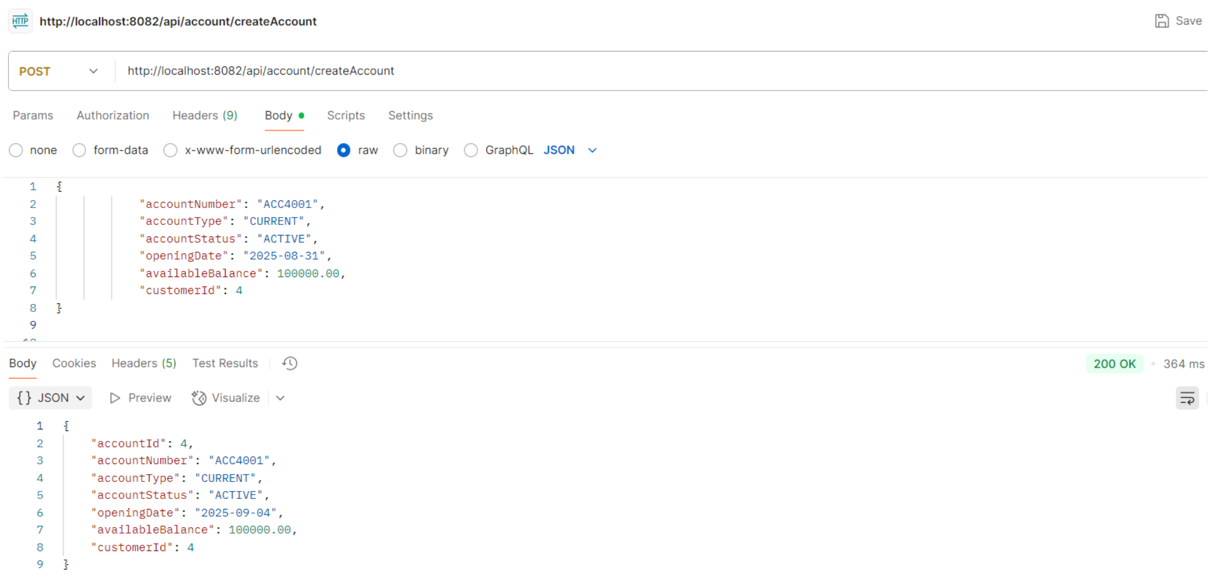
Query Params

Key	Value	Description

Body Cookies Headers (5) Test Results 200 OK • 4.77 s • 8

```
        "openingDate": "2025-08-31",
        "availableBalance": 50000.00,
        "customerId": 3
    },
    {
        "accountId": 3,
        "accountNumber": "ACC3001",
        "accountType": "CURRENT",
        "accountStatus": "ACTIVE",
        "openingDate": "2025-08-31",
        "availableBalance": 100000.00,
        "customerId": 4
    }
],
"pageable": {
    "pageNumber": 0,
    "pageSize": 20,
    "sort": [],
    "offset": 0,
    "paged": true,
    "unpaged": false
```

--createCustomer



http://localhost:8082/api/account/createAccount

POST http://localhost:8082/api/account/createAccount

Params Authorization Headers (9) Body Scripts Settings

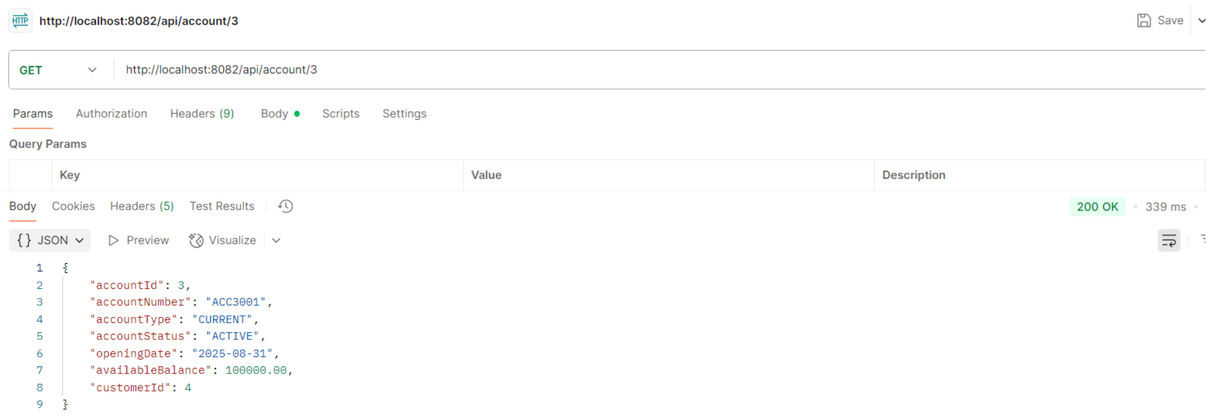
none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

```
{
        "accountNumber": "ACC4001",
        "accountType": "CURRENT",
        "accountStatus": "ACTIVE",
        "openingDate": "2025-08-31",
        "availableBalance": 100000.00,
        "customerId": 4
}
```

Body Cookies Headers (5) Test Results 200 OK • 364 ms

```
{
    "accountId": 4,
    "accountNumber": "ACC4001",
    "accountType": "CURRENT",
    "accountStatus": "ACTIVE",
    "openingDate": "2025-09-04",
    "availableBalance": 100000.00,
    "customerId": 4
}
```

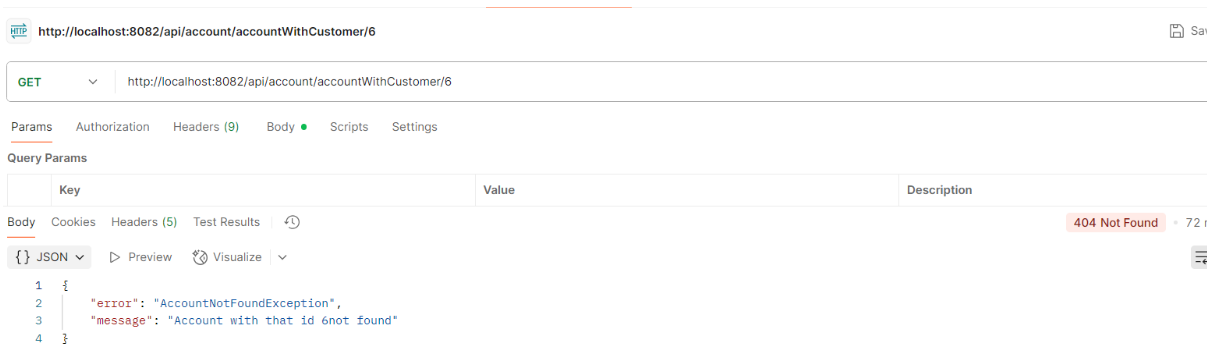
--getAccountById


http://localhost:8082/api/account/3

GET http://localhost:8082/api/account/3

```
{
    "accountId": 3,
    "accountNumber": "ACC3001",
    "accountType": "CURRENT",
    "accountStatus": "ACTIVE",
    "openingDate": "2025-08-31",
    "availableBalance": 100000.00,
    "customerId": 4
}
```

--customException

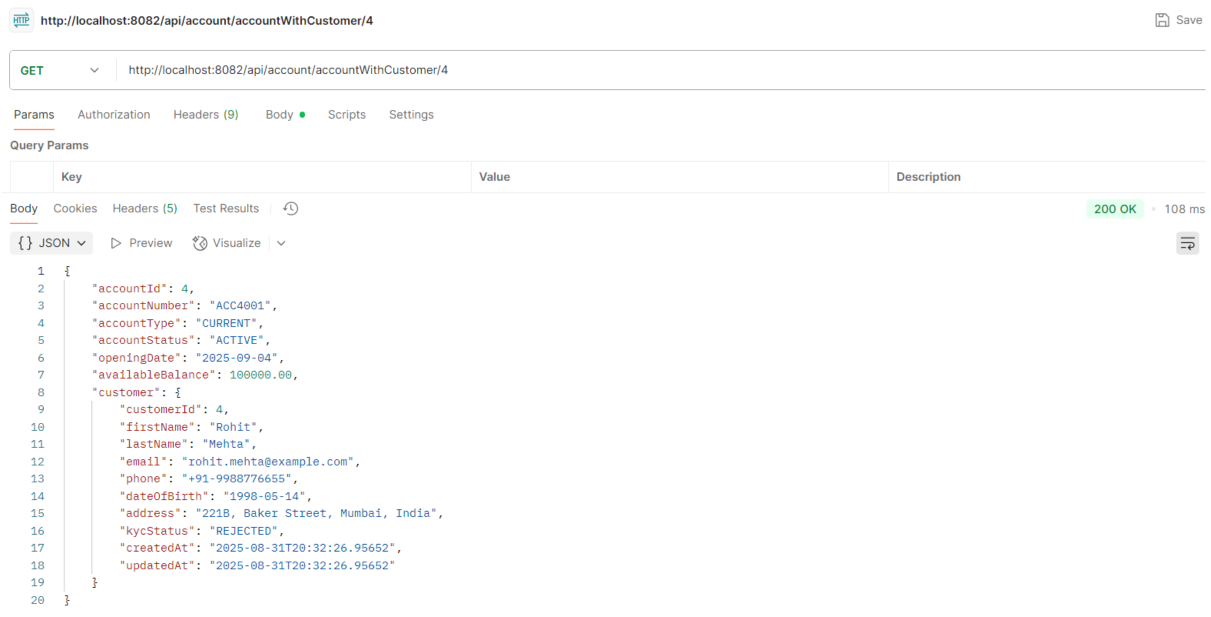

http://localhost:8082/api/account/accountWithCustomer/6

GET http://localhost:8082/api/account/accountWithCustomer/6

404 Not Found

```
{
    "error": "AccountNotFoundException",
    "message": "Account with that id 6not found"
}
```

--getAccountDetailsWithCustomerDetails


http://localhost:8082/api/account/accountWithCustomer/4

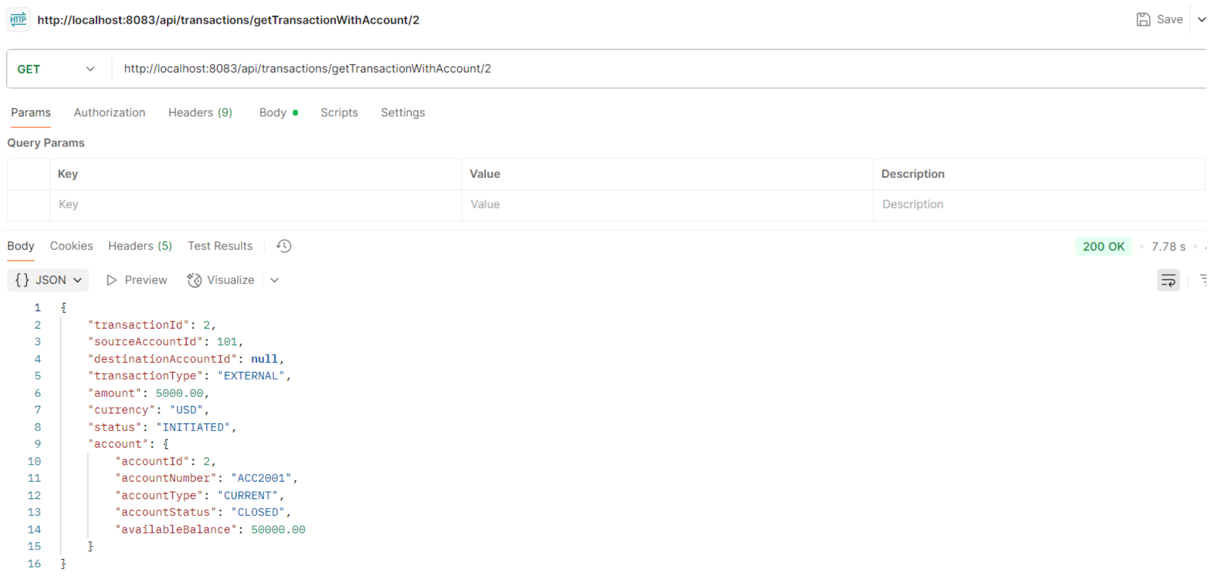
GET http://localhost:8082/api/account/accountWithCustomer/4

200 OK 108 ms

```
{
    "accountId": 4,
    "accountNumber": "ACC4001",
    "accountType": "CURRENT",
    "accountStatus": "ACTIVE",
    "openingDate": "2025-09-04",
    "availableBalance": 100000.00,
    "customer": {
        "customerId": 4,
        "firstName": "Rohit",
        "lastName": "Mehta",
        "email": "rohit.mehta@example.com",
        "phone": "+91-9988776655",
        "dateOfBirth": "1998-05-14",
        "address": "221B, Baker Street, Mumbai, India",
        "kycStatus": "REJECTED",
        "createdAt": "2025-08-31T20:32:26.95652",
        "updatedAt": "2025-08-31T20:32:26.95652"
    }
}
```

5.3 Transaction Service

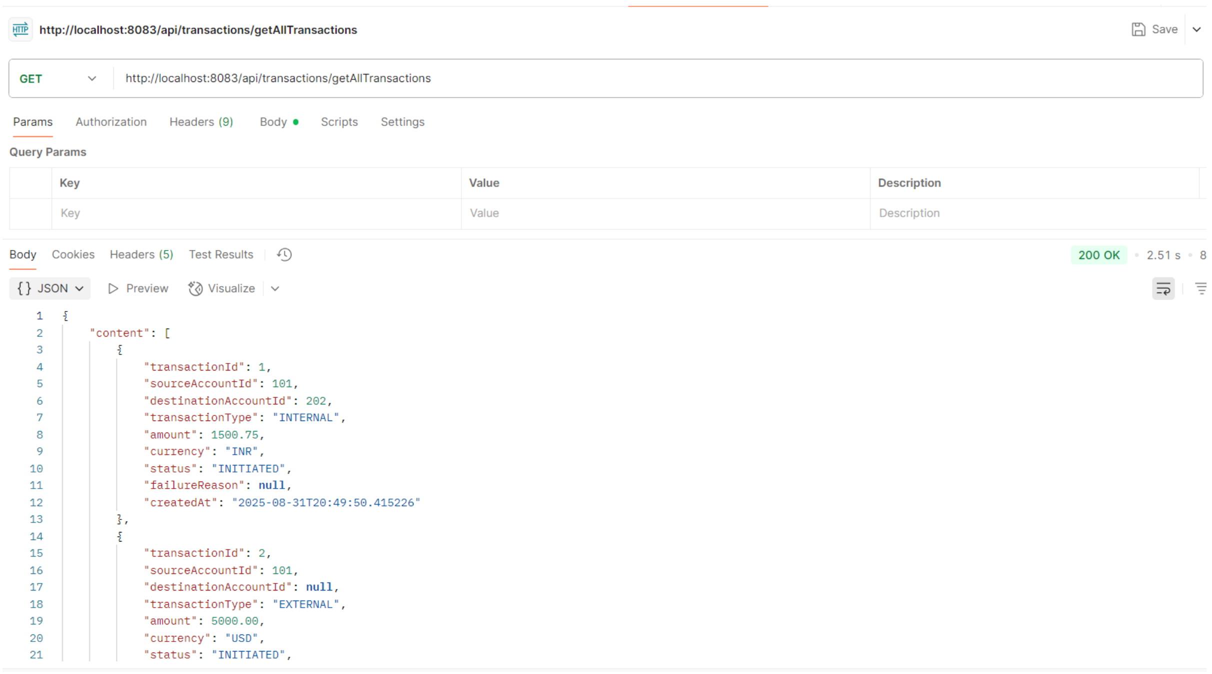
-- gettingTransactionsWithAccountDetails



GET http://localhost:8083/api/transactions/getTransactionWithAccount/2

```
{
    "transactionId": 2,
    "sourceAccountId": 101,
    "destinationAccountId": null,
    "transactionType": "EXTERNAL",
    "amount": 5000.00,
    "currency": "USD",
    "status": "INITIATED",
    "account": {
        "accountId": 2,
        "accountNumber": "ACC2001",
        "accountType": "CURRENT",
        "accountStatus": "CLOSED",
        "availableBalance": 50000.00
    }
}
```

--getAllTransactions with Pagination

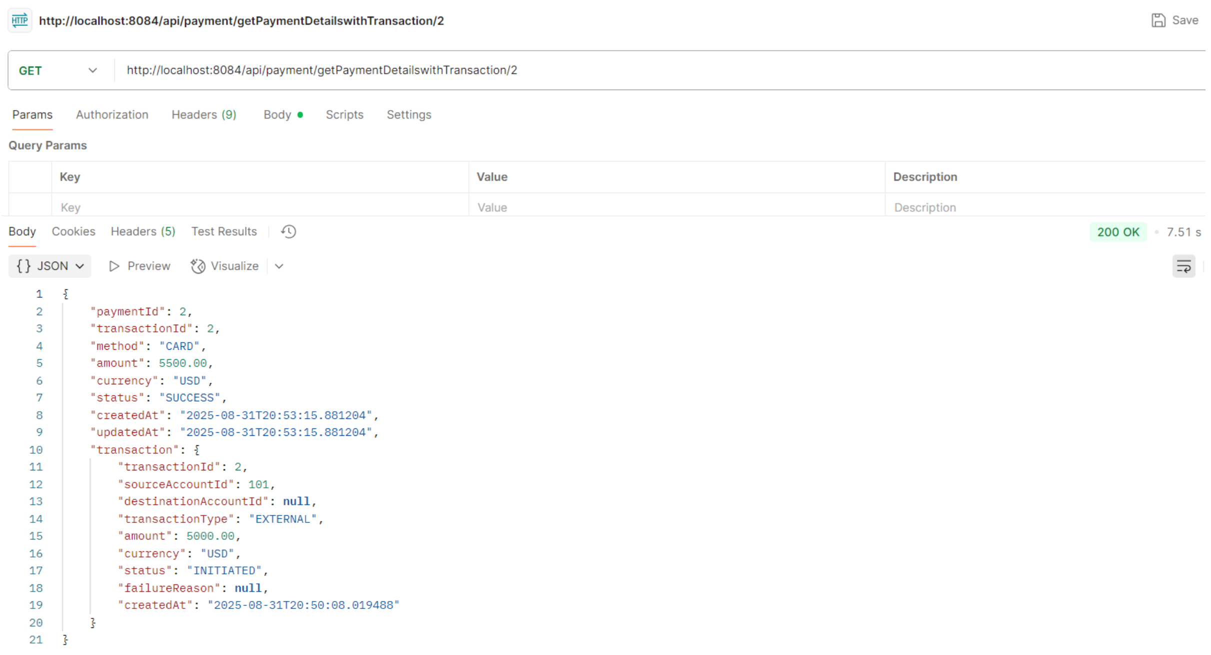


GET http://localhost:8083/api/transactions/getAllTransactions

```
{
    "content": [
        {
            "transactionId": 1,
            "sourceAccountId": 101,
            "destinationAccountId": 202,
            "transactionType": "INTERNAL",
            "amount": 1500.75,
            "currency": "INR",
            "status": "INITIATED",
            "failureReason": null,
            "createdAt": "2025-08-31T20:49:50.415226"
        },
        {
            "transactionId": 2,
            "sourceAccountId": 101,
            "destinationAccountId": null,
            "transactionType": "EXTERNAL",
            "amount": 5000.00,
            "currency": "USD",
            "status": "INITIATED",
```

5.4 Payment Service

--getPaymentDetailsWithTransactionDetails


http://localhost:8084/api/payment/getPaymentDetailswithTransaction/2

GET http://localhost:8084/api/payment/getPaymentDetailswithTransaction/2

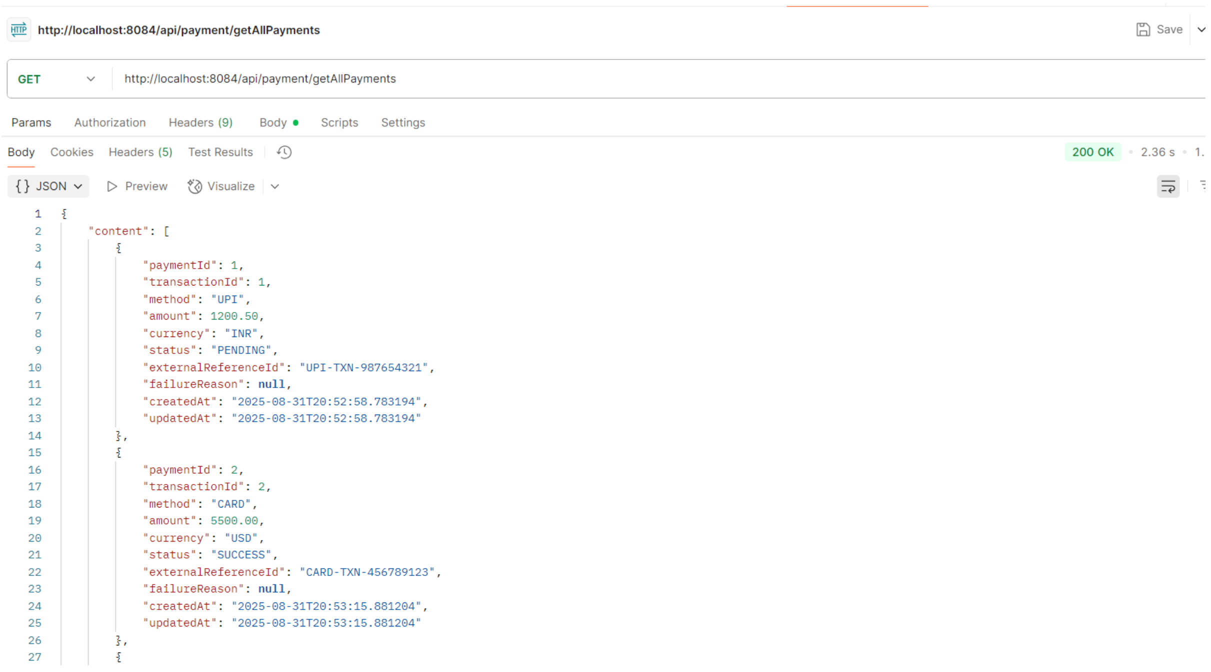
Query Params

Key	Value	Description
Key	Value	Description

200 OK · 7.51 s

```
{
    "paymentId": 2,
    "transactionId": 2,
    "method": "CARD",
    "amount": 5500.00,
    "currency": "USD",
    "status": "SUCCESS",
    "createdAt": "2025-08-31T20:53:15.881204",
    "updatedAt": "2025-08-31T20:53:15.881204",
    "transaction": {
        "transactionId": 2,
        "sourceAccountId": 101,
        "destinationAccountId": null,
        "transactionType": "EXTERNAL",
        "amount": 5000.00,
        "currency": "USD",
        "status": "INITIATED",
        "failureReason": null,
        "createdAt": "2025-08-31T20:50:08.019488"
    }
}
```

--getAllPayments


http://localhost:8084/api/payment/getAllPayments

GET http://localhost:8084/api/payment/getAllPayments

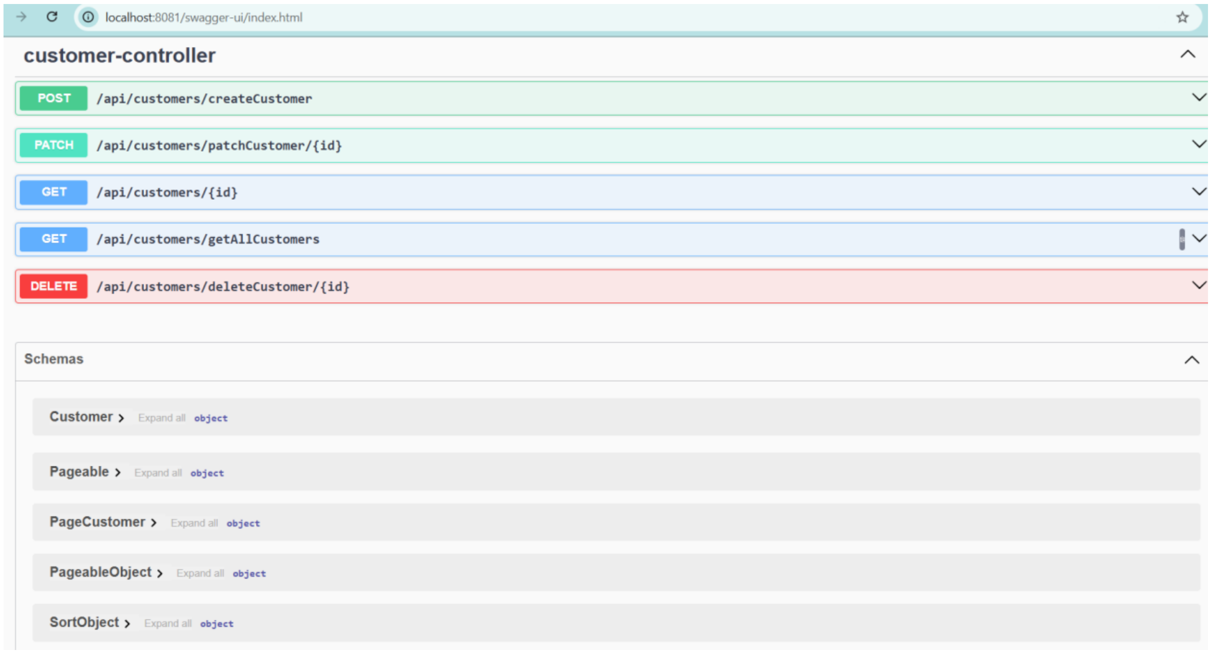
200 OK · 2.36 s

```
{
    "content": [
        {
            "paymentId": 1,
            "transactionId": 1,
            "method": "UPI",
            "amount": 1200.50,
            "currency": "INR",
            "status": "PENDING",
            "externalReferenceId": "UPI-TXN-987654321",
            "failureReason": null,
            "createdAt": "2025-08-31T20:52:58.783194",
            "updatedAt": "2025-08-31T20:52:58.783194"
        },
        {
            "paymentId": 2,
            "transactionId": 2,
            "method": "CARD",
            "amount": 5500.00,
            "currency": "USD",
            "status": "SUCCESS",
            "externalReferenceId": "CARD-TXN-456789123",
            "failureReason": null,
            "createdAt": "2025-08-31T20:53:15.881204",
            "updatedAt": "2025-08-31T20:53:15.881204"
        },
        {
```

6.Tools Used

6.1 API Documentation (Swagger & OpenAPI)

- Swagger is a tool for **documenting REST APIs**.
- It shows APIs in a **standard format** and allows testing in the browser.
- Provides **interactive API docs**.
- Reduces **manual documentation work**.
- Helps **frontend and backend teams** understand API contracts.
- Supports **API versioning and testing**.



localhost:8081/swagger-ui/index.html

customer-controller

POST /api/customers/createCustomer

PATCH /api/customers/patchCustomer/{id}

GET /api/customers/{id}

GET /api/customers/getAllCustomers

DELETE /api/customers/deleteCustomer/{id}

Schemas

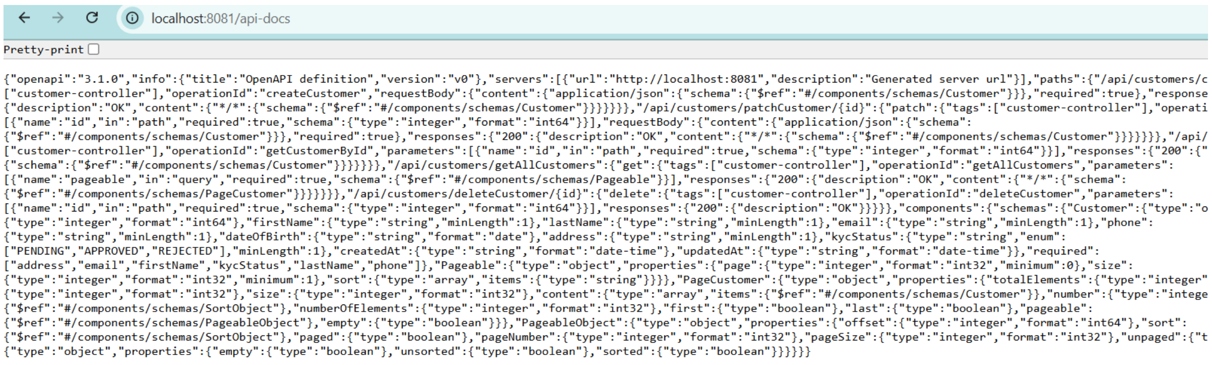
Customer > Expand all object

Pageable > Expand all object

PageCustomer > Expand all object

PageableObject > Expand all object

SortObject > Expand all object



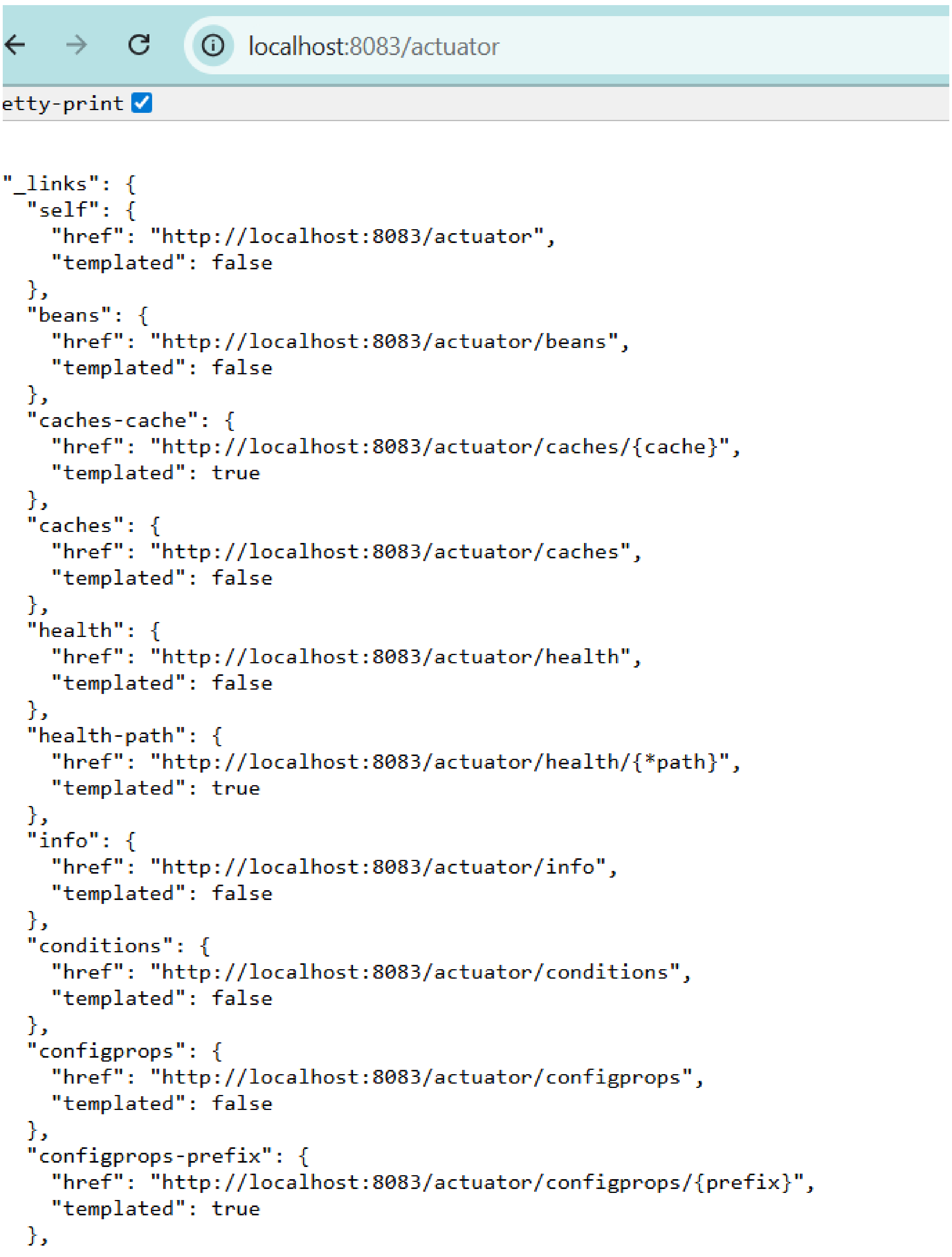
localhost:8081/api-docs

Pretty-print ☐

```
{"openapi":"3.1.0","info":{"title":"OpenAPI definition","version":"v0"},"servers":[{"url":"http://localhost:8081","description":"Generated server url"}],"paths":{"/api/customers/c
["customer-controller"],"operationId":"createCustomer","requestBody":{"content":{"application/json":{"schema":{"$ref":"#/components/schemas/Customer"}}},"required":true},"response
{"description":"OK","content":{"*/*":{"schema":{"$ref":"#/components/schemas/Customer"}}}}}}},"/api/customers/patchCustomer/{id}":{"patch":{"tags":["customer-controller"],"operati
[{"name":"id","in":"path","required":true,"schema":{"type":"integer","format":"int64"}}],"requestBody":{"content":{"application/json":{"schema":
{"$ref":"#/components/schemas/Customer"}}},"required":true},"responses":{"200":{"description":"OK","content":{"*/*":{"schema":{"$ref":"#/components/schemas/Customer"}}}}}}},"/api/
["customer-controller"],"operationId":"getCustomerById","parameters":[{"name":"id","in":"path","required":true,"schema":{"type":"integer","format":"int64"}}],"responses":{"200":{"
{"schema":{"$ref":"#/components/schemas/Customer"}}}}}}},"/api/customers/getAllCustomers":{"get":{"tags":["customer-controller"],"operationId":"getAllCustomers","parameters":
[{"name":"pageable","in":"query","required":true,"schema":{"$ref":"#/components/schemas/Pageable"}}],"responses":{"200":{"description":"OK","content":{"*/*":{"schema":
{"$ref":"#/components/schemas/PageCustomer"}}}}}}},"/api/customers/deleteCustomer/{id}":{"delete":{"tags":["customer-controller"],"operationId":"deleteCustomer","parameters":
[{"name":"id","in":"path","required":true,"schema":{"type":"integer","format":"int64"}}],"responses":{"200":{"description":"OK"}}}}},"components":{"schemas":{"Customer":{"type":"o
{"type":"integer","format":"int64"},"firstName":{"type":"string","minLength":1},"lastName":{"type":"string","minLength":1},"email":{"type":"string","minLength":1},"phone":
{"type":"string","minLength":1},"dateOfBirth":{"type":"string","format":"date"},"address":{"type":"string","minLength":1},"kycStatus":{"type":"string","enum":
["PENDING","APPROVED","REJECTED"],"minLength":1},"createdAt":{"type":"string","format":"date-time"},"updatedAt":{"type":"string","format":"date-time"}},"required":
["address","email","firstName","kycStatus","lastName","phone"]},"Pageable":{"type":"object","properties":{"page":{"type":"integer","format":"int32","minimum":0},"size":
{"type":"integer","format":"int32","minimum":1},"sort":{"type":"array","items":{"type":"string"}}}},"PageCustomer":{"type":"object","properties":{"totalElements":{"type":"integer"
{"type":"integer","format":"int32"},"size":{"type":"integer","format":"int32"},"content":{"type":"array","items":{"$ref":"#/components/schemas/Customer"}},"number":{"type":"intege
{"$ref":"#/components/schemas/SortObject"},"numberOfElements":{"type":"integer","format":"int32"},"first":{"type":"boolean"},"last":{"type":"boolean"},"pageable":
{"$ref":"#/components/schemas/PageableObject"},"empty":{"type":"boolean"}}},"PageableObject":{"type":"object","properties":{"offset":{"type":"integer","format":"int64"},"sort":
{"$ref":"#/components/schemas/SortObject"},"paged":{"type":"boolean"},"pageNumber":{"type":"integer","format":"int32"},"pageSize":{"type":"integer","format":"int32"},"unpaged":{"t
{"type":"object","properties":{"empty":{"type":"boolean"},"unsorted":{"type":"boolean"},"sorted":{"type":"boolean"}}}}}}
```

6.2 Actuator

- Health Monitoring: Check if microservices (Customer, Account, Transaction) are up.
- Metrics & Performance: Track memory usage, HTTP request counts, and response times.



```
"_links": {
  "self": {
    "href": "http://localhost:8083/actuator",
    "templated": false
  },
  "beans": {
    "href": "http://localhost:8083/actuator/beans",
    "templated": false
  },
  "caches-cache": {
    "href": "http://localhost:8083/actuator/caches/{cache}",
    "templated": true
  },
  "caches": {
    "href": "http://localhost:8083/actuator/caches",
    "templated": false
  },
  "health": {
    "href": "http://localhost:8083/actuator/health",
    "templated": false
  },
  "health-path": {
    "href": "http://localhost:8083/actuator/health/{*path}",
    "templated": true
  },
  "info": {
    "href": "http://localhost:8083/actuator/info",
    "templated": false
  },
  "conditions": {
    "href": "http://localhost:8083/actuator/conditions",
    "templated": false
  },
  "configprops": {
    "href": "http://localhost:8083/actuator/configprops",
    "templated": false
  },
  "configprops-prefix": {
    "href": "http://localhost:8083/actuator/configprops/{prefix}",
    "templated": true
  },
```

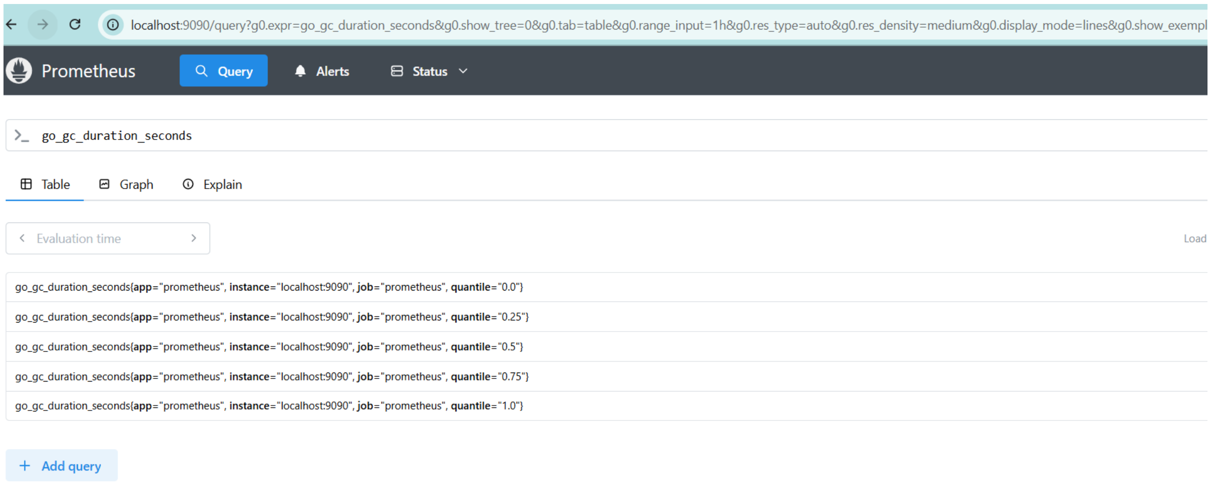
6.3 Prometheus & Grafana

1. Prometheus

- Prometheus is a monitoring and alerting tool.
- It collects metrics from applications, services, and servers.
- Metrics are stored in a time-series database.
- It helps track things like CPU usage, memory, HTTP requests, and application performance.

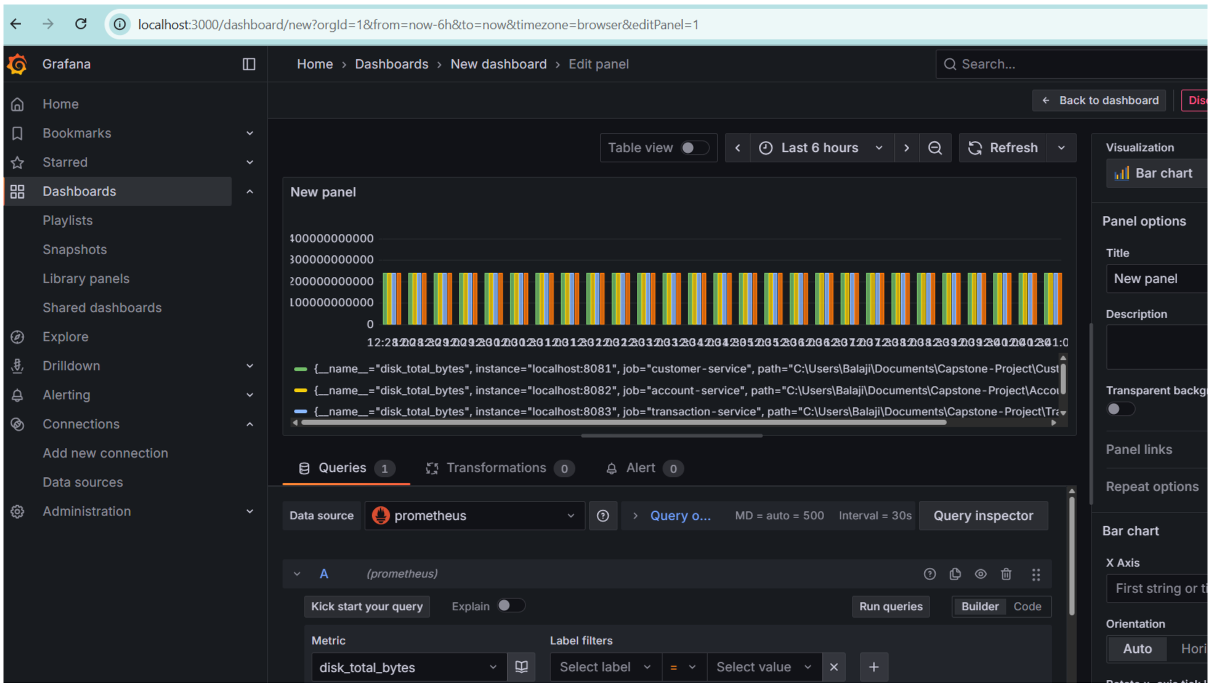
2. Grafana

- Grafana is a visualization tool for metrics.
- It connects to Prometheus (and other data sources) to create dashboards and graphs.
- Helps monitor services in real-time and identify issues quickly.


localhost:9090/query?g0.expr=go_gc_duration_seconds&g0.show_tree=0&g0.tab=table&g0.range_input=1h&g0.res_type=auto&g0.res_density=medium&g0.display_mode=lines&g0.show_exempl
Prometheus
Query
Alerts
Status
Table
Graph
Explain
Evaluation time
Load
Add query

```
go_gc_duration_seconds
```

```
go_gc_duration_seconds{app="prometheus", instance="localhost:9090", job="prometheus", quantile="0.0"}
go_gc_duration_seconds{app="prometheus", instance="localhost:9090", job="prometheus", quantile="0.25"}
go_gc_duration_seconds{app="prometheus", instance="localhost:9090", job="prometheus", quantile="0.5"}
go_gc_duration_seconds{app="prometheus", instance="localhost:9090", job="prometheus", quantile="0.75"}
go_gc_duration_seconds{app="prometheus", instance="localhost:9090", job="prometheus", quantile="1.0"}
```

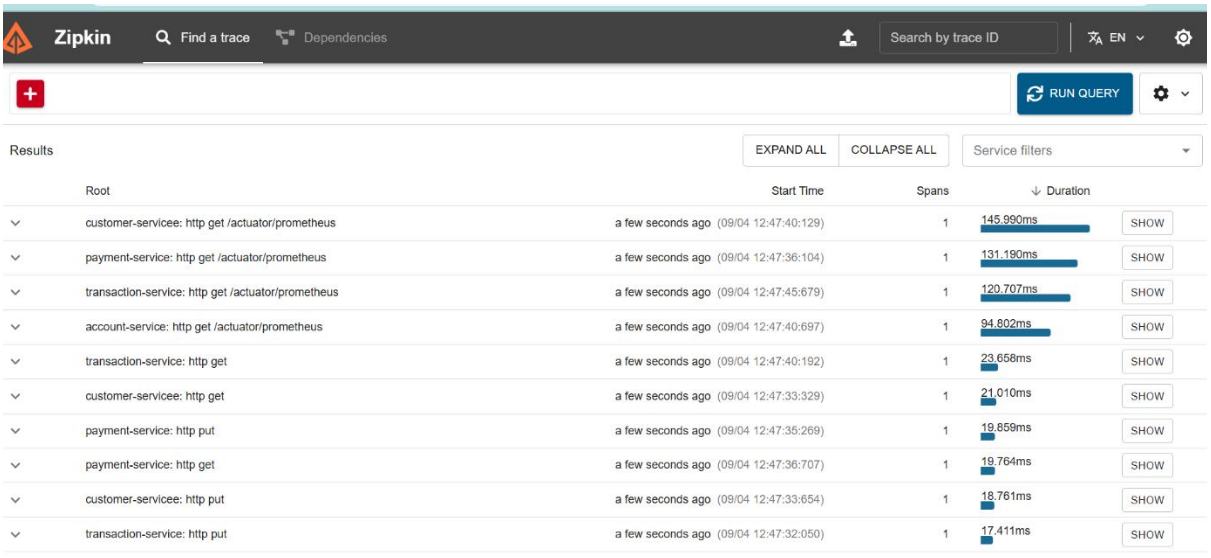


```
{__name__="disk_total_bytes", instance="localhost:8081", job="customer-service", path="C:\Users\Balaji\Documents\Capstone-Project\Cust
{__name__="disk_total_bytes", instance="localhost:8082", job="account-service", path="C:\Users\Balaji\Documents\Capstone-Project\Accou
{__name__="disk_total_bytes", instance="localhost:8083", job="transaction-service", path="C:\Users\Balaji\Documents\Capstone-Project\Tra
```

Data source: prometheus
Query o... MD = auto = 500 Interval = 30s
A (prometheus)
Kick start your query
Explain
Metric: disk_total_bytes
Label filters: Select label = Select value
Title: New panel
Description:
Transparent backg

6.4 Zipkin for tracing

- Zipkin is a distributed tracing system.
- It helps track requests as they flow through microservices.
- Useful for identifying latency, bottlenecks, and failures in complex systems.



Root	Start Time	Spans	Duration	
customer-servicee: http get /actuator/prometheus	a few seconds ago (09/04 12:47:40:129)	1	145.990ms	SHOW
payment-service: http get /actuator/prometheus	a few seconds ago (09/04 12:47:36:104)	1	131.190ms	SHOW
transaction-service: http get /actuator/prometheus	a few seconds ago (09/04 12:47:45:679)	1	120.707ms	SHOW
account-service: http get /actuator/prometheus	a few seconds ago (09/04 12:47:40:697)	1	94.802ms	SHOW
transaction-service: http get	a few seconds ago (09/04 12:47:40:192)	1	23.658ms	SHOW
customer-servicee: http get	a few seconds ago (09/04 12:47:33:329)	1	21.010ms	SHOW
payment-service: http put	a few seconds ago (09/04 12:47:35:269)	1	19.859ms	SHOW
payment-service: http get	a few seconds ago (09/04 12:47:36:707)	1	19.764ms	SHOW
customer-servicee: http put	a few seconds ago (09/04 12:47:33:654)	1	18.761ms	SHOW
transaction-service: http put	a few seconds ago (09/04 12:47:32:050)	1	17.411ms	SHOW

BANKING FRONTEND

Why Angular:

- Simplifies Front-End Development: Angular provides all the tools needed for front-end development in a single framework, making it easier to build dynamic and scalable applications.
- Component-Based Architecture: Angular's component-based structure helps break down large applications into smaller, manageable parts, making development and maintenance easier.
- Powerful Features: Angular offers powerful features such as two-way data binding, routing, and form validation that speed up the development process.
- Seamless Integration: Angular integrates well with RESTful APIs, making it ideal for building full-stack web applications and SPAs.
- Strong Community Support: With strong backing from Google and an active community, Angular is continuously updated and improved, ensuring long-term stability.

Commands used for this project:

1. Install Node.js & npm

   --- node -v

   --- npm -v

2. Install Angular CLI

   --- npm install -g @angular/cli

3. Create Angular Project

   --- ng new banking-frontend

   --- cd banking-frontend

4. Run Development Server

   --- ng serve -o

5. Install Angular Material

   --- ng add @angular/material

6. Generate Modules

    --- ng g module transactions

    --- ng g module dashboard

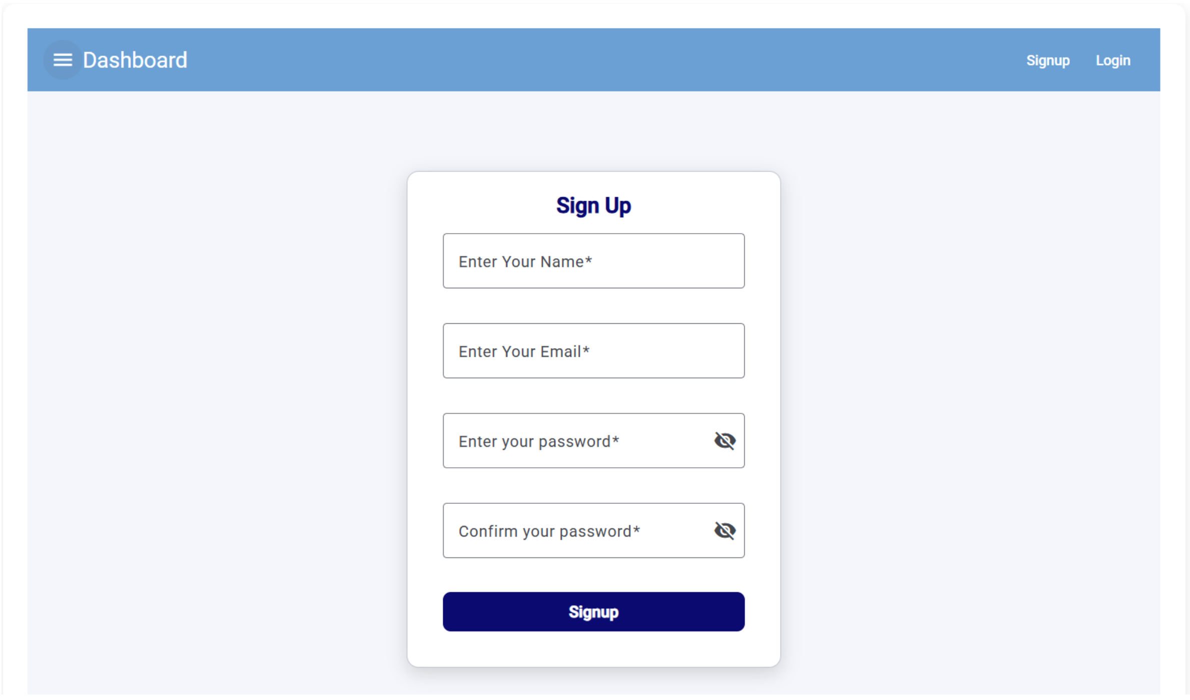
7. Generate Components

    --- ng g c component/account
    --- ng g c component/customer
    --- ng g c component/transactions
    --- ng g c component/payments

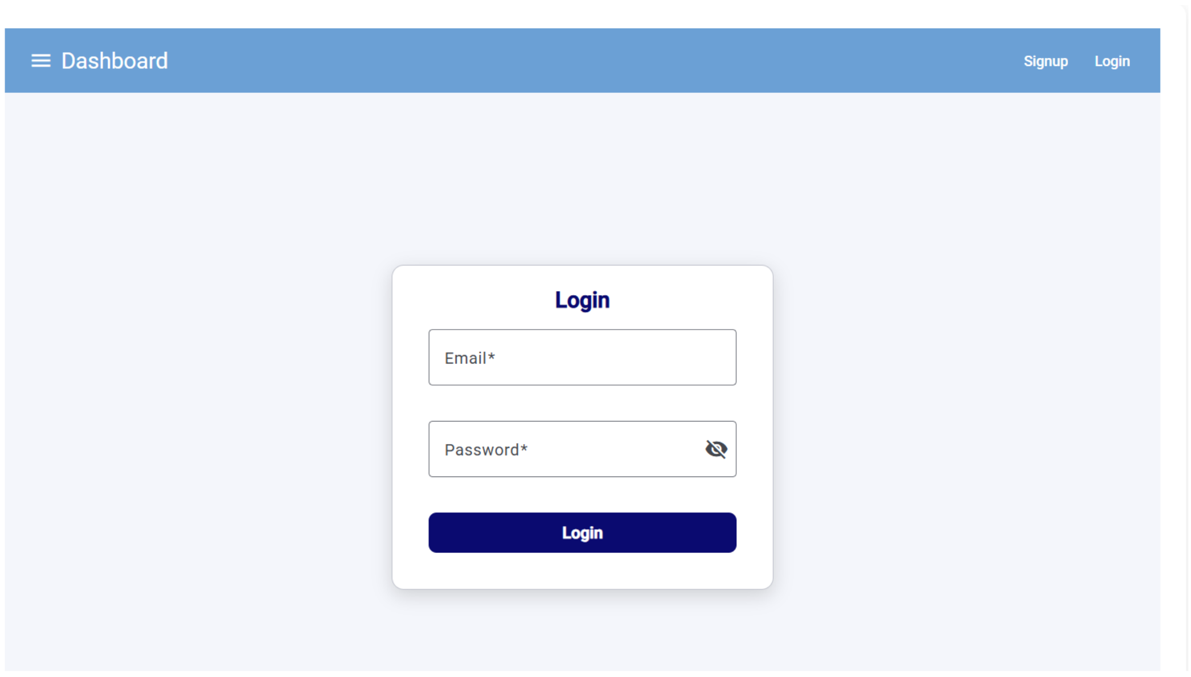
8. Generate Services

    --- ng g service services/auth
    --- ng g service services/account
    --- ng g service services/transaction
    --- ng g service services/customer
    --- ng g s services/payment

1.Signup page



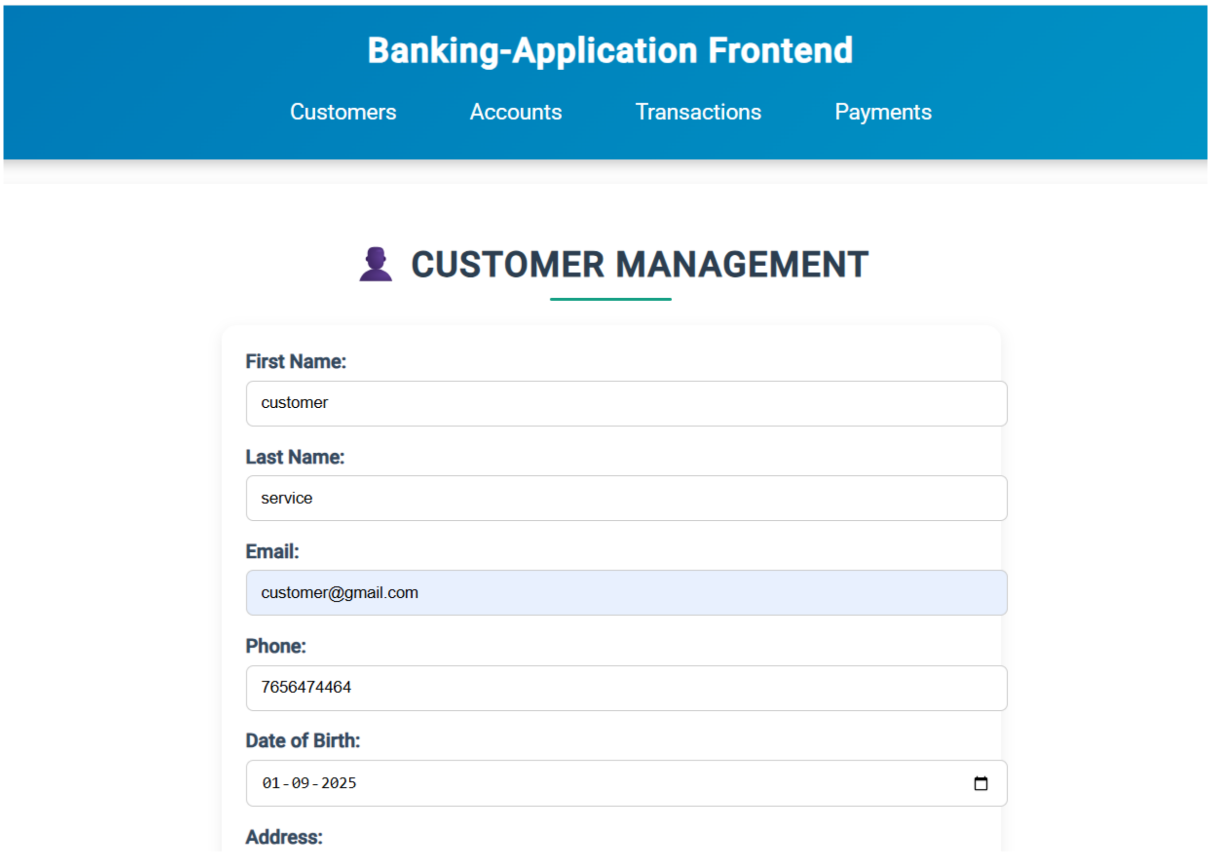
2.Login page



3.Customer Service Dashboard

Purpose: Manages all customer-related operations.
Typical Features:

- **View All Customers**: Displays a list of registered customers with details like name, email, phone, address, KYC status, etc.
- **Add New Customer**: A form to register new customers with personal and account details.
- **Search / Filter Customers**: By ID, name, or status.
- **Edit / Update Customer Details**: Patch/Update KYC status, phone number, or address.
- **Delete Customer**: Remove customer records if needed.
  Use Case Example: A bank employee uses this to onboard new customers or update KYC details.



First Name: customer

Last Name: service

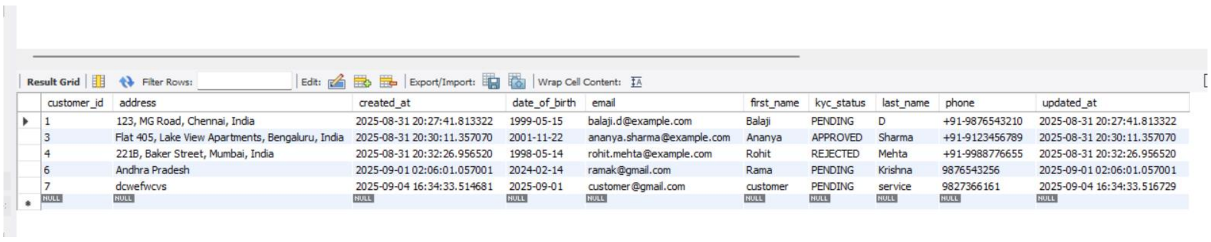
Email: customer@gmail.com

Phone: 7656474464

Date of Birth: 01-09-2025

Address:

Adding customer into database

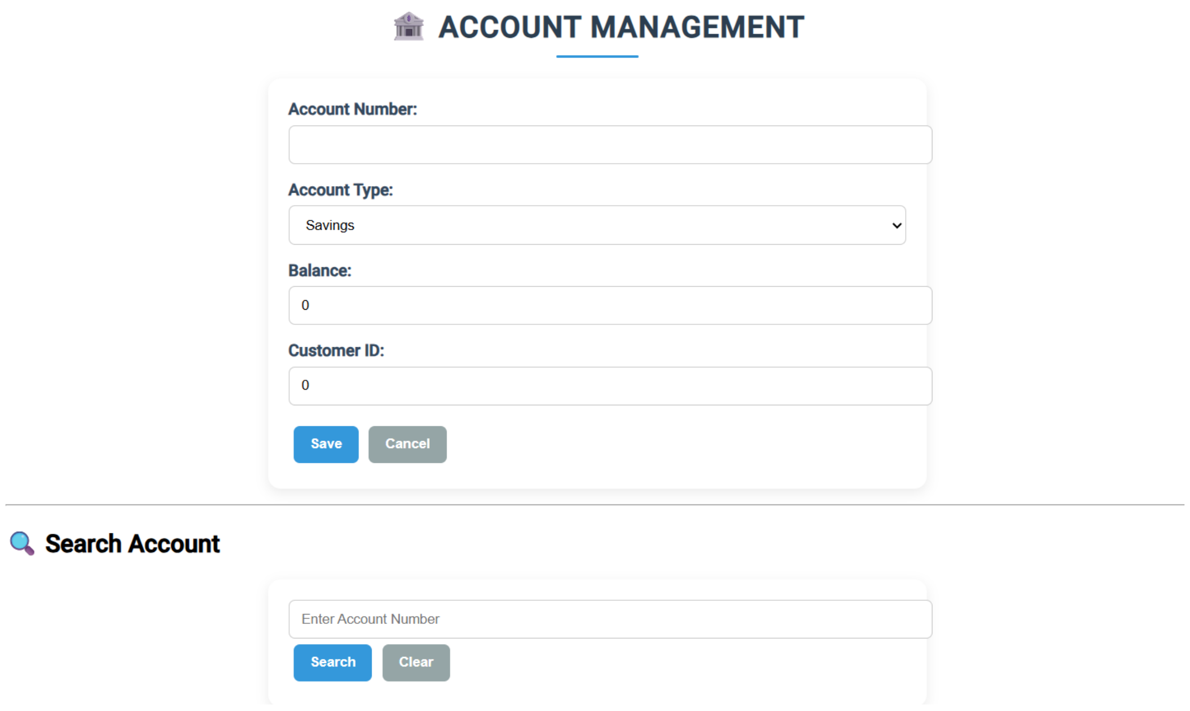


customer_id	address	created_at	date_of_birth	email	first_name	kyc_status	last_name	phone	updated_at
1	123, MG Road, Chennai, India	2025-08-31 20:27:41.813322	1999-05-15	balaji.d@example.com	Balaji	PENDING	D	+91-9876543210	2025-08-31 20:27:41.813322
3	Flat 405, Lake View Apartments, Bengaluru, India	2025-08-31 20:30:11.357070	2001-11-22	ananya.sharma@example.com	Ananya	APPROVED	Sharma	+91-9123456789	2025-08-31 20:30:11.357070
4	221B, Baker Street, Mumbai, India	2025-08-31 20:32:26.956520	1998-05-14	rohit.mehta@example.com	Rohit	REJECTED	Mehta	+91-9988776655	2025-08-31 20:32:26.956520
6	Andhra Pradesh	2025-09-01 02:06:01.057001	2024-02-14	ramak@gmail.com	Rama	PENDING	Krishna	9876543256	2025-09-01 02:06:01.057001
7	dcwefwcvs	2025-09-04 16:34:33.514681	2025-09-01	customer@gmail.com	customer	PENDING	service	9827366161	2025-09-04 16:34:33.516729
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

4.Account Service Dashboard

Purpose: Manages customer accounts (Savings, Current, etc.).
Typical Features:

- **Create Account**: Open a savings or current account for an existing customer.
- **View Accounts**: List of accounts with balance, account type, and status (active/inactive).
- **Account Linking**: Associate accounts with customers.
- **Account Actions**: Freeze/unfreeze account, close account, or modify limits.
  Use Case Example: When a new customer requests to open an account, the bank operator creates it here.


ACCOUNT MANAGEMENT

Account Number:

Account Type: Savings

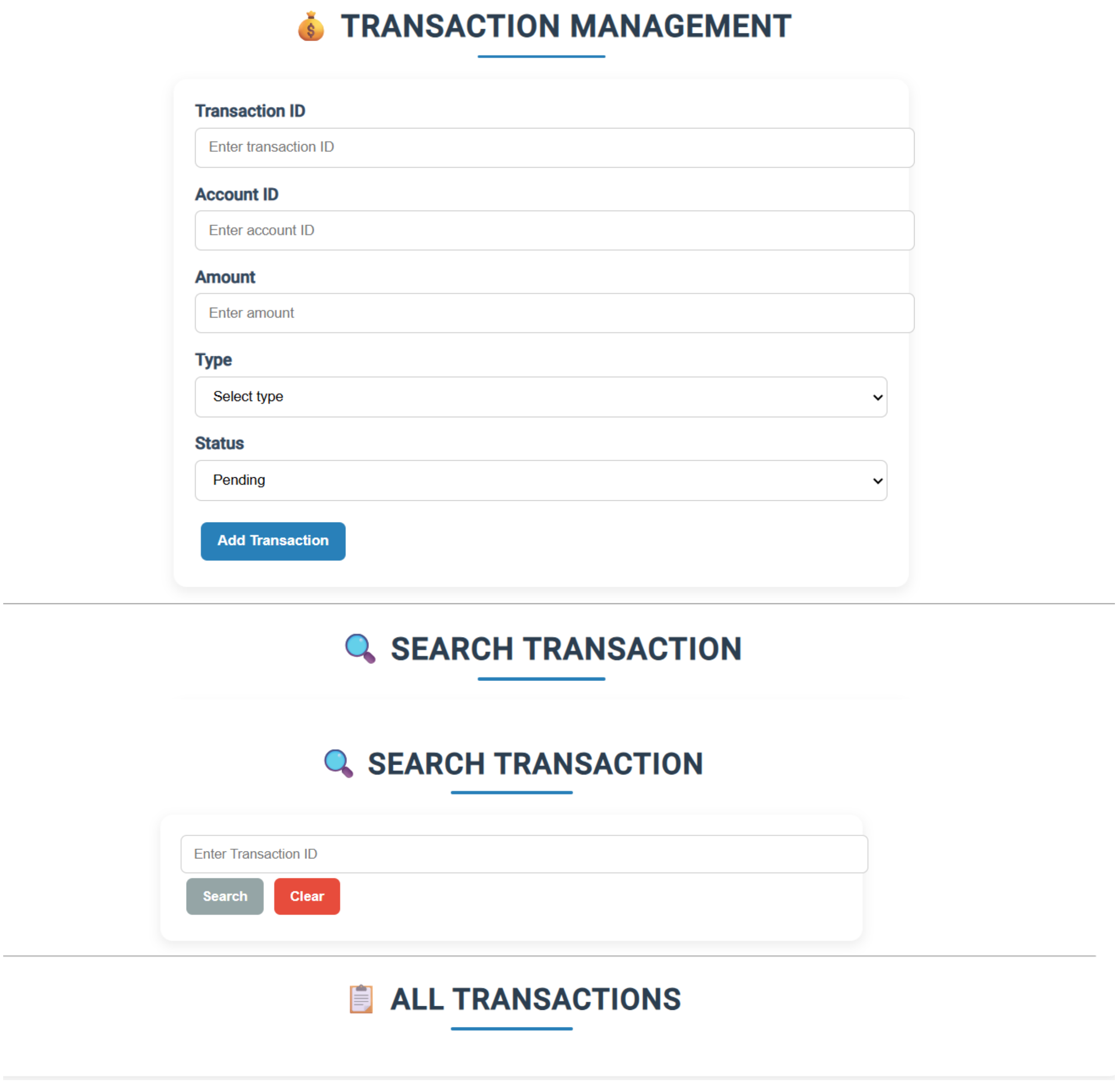
Balance: 0

Customer ID: 0

5.Transaction Service Dashboard

Purpose: Tracks and manages all banking transactions.
Typical Features:

- **View Transactions**: A table of credits/debits for each account (amount, date, description).
- **Search Transactions**: Filter by account ID, transaction ID, or date range.
- **Add Transaction**: Record manual adjustments (rare, but possible for testing/demo).
- **Transaction History Export**: Download statements (PDF/CSV).
  Use Case Example: A customer requests their last 10 transactions → Operator fetches from here, or the system provides via API.


💰 TRANSACTION MANAGEMENT

Transaction ID: Enter transaction ID
Account ID: Enter account ID
Amount: Enter amount
Type: Select type
Status: Pending

🔍 SEARCH TRANSACTION

🔍 SEARCH TRANSACTION

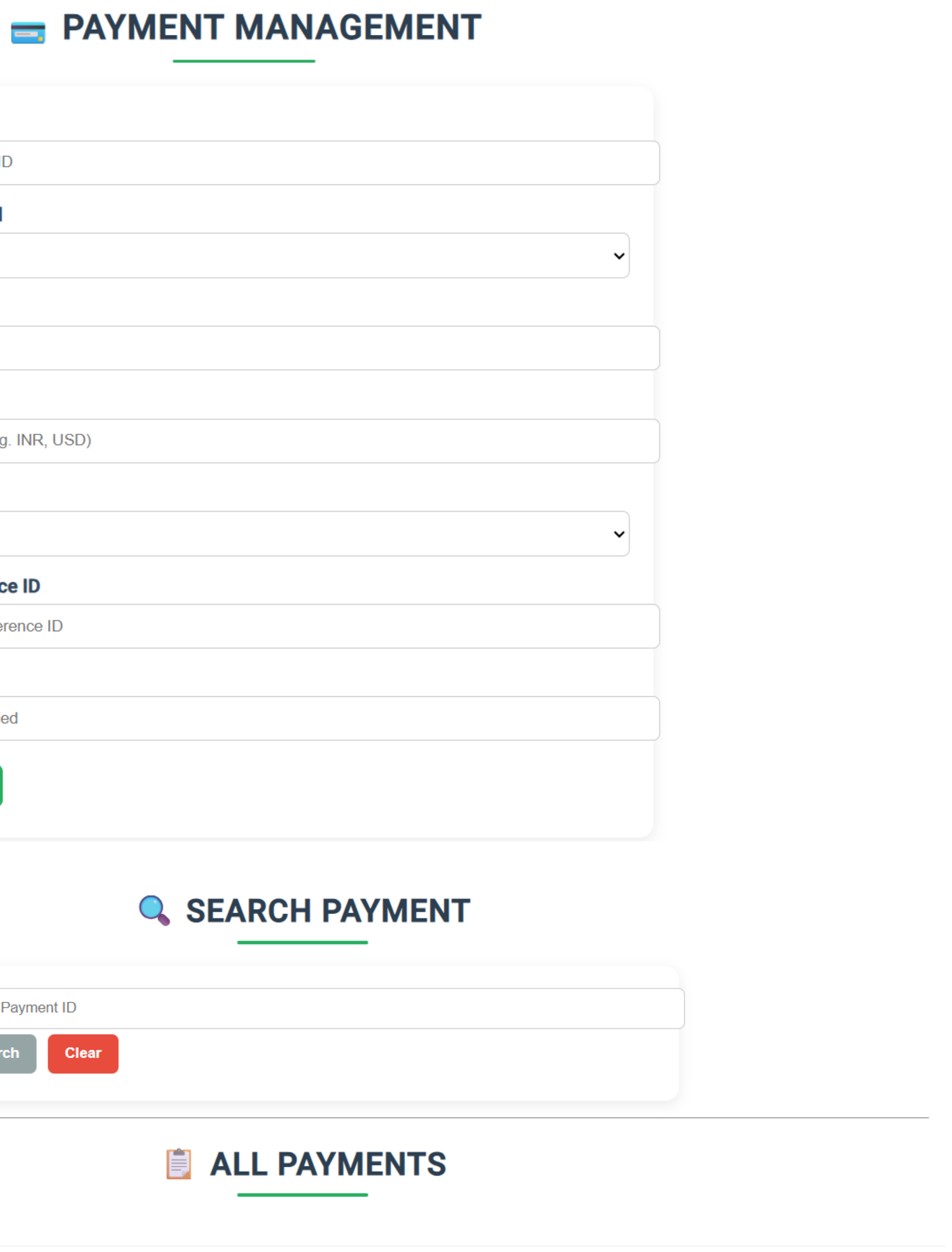
Enter Transaction ID

📋 ALL TRANSACTIONS

6.Payment Service Dashboard

Purpose: Handles fund transfers and payments.
Typical Features:

- **Internal Transfers**: Between accounts in the same bank (customer to customer).
- **External Payments**: Transfers to other banks (via UPI/NEFT/RTGS in real-world).
- **Bill Payments**: Utility bills, EMI, or card payments (optional).
- **Payment History**: Shows successful, pending, or failed payments.
- **Validation Rules**: Check sufficient balance, validate payee account, and OTP/security checks.
  Use Case Example: A customer transfers ₹5000 from their savings account to another user's account


PAYMENT MANAGEMENT

Transaction ID
Enter transaction ID

Payment Method
Select method

Amount
Enter amount

Currency
Enter currency (e.g. INR, USD)

Status
Pending

External Reference ID
Enter external reference ID

Failure Reason
Enter reason if failed

SEARCH PAYMENT

Enter Payment ID

ALL PAYMENTS

Conclusion:

The Banking Microservices Application developed with Angular (frontend) and Java Spring Boot (backend) successfully demonstrates how a modern banking platform can be built using microservices architecture. Each service dashboard — Customer, Account, Transaction, and Payment — provides a clear separation of concerns, allowing better scalability, maintainability, and security.

The frontend dashboards enable easy management of customers, accounts, transactions, and payments with a user-friendly interface, while the backend ensures secure authentication, validations, and smooth service-to-service communication.

By integrating API Gateway, Eureka Discovery, JWT Authentication, Kafka (for async notifications), Prometheus, Grafana, and Zipkin, the project showcases not only core banking operations but also real-world enterprise practices like monitoring, auditing, and distributed tracing.

In conclusion, this project provides a secure, scalable, and modular banking solution, which can be extended further with advanced features such as loan management, credit scoring, and AI-powered fraud detection.