Subscription Management System

Overview of the project, key deliverables with Spring Boot microservices

Spring Boot   Spring Cloud   Microservices

Table of Contents

1 System Overview
2 Architecture at a Glance
3 Microservices Approach
4 Team Responsibilities Overview
5 User & Plan Service
6 Subscription Service
7 Payment Service
8 Coupon Service
9 Device Service
10 Invoice Service
11 API Design & Integration Standards
12 End-to-End Flow & System Interactions
13 Deployment, Discovery & Testing
14 Q&A

System Overview

The Subscription Management System is a comprehensive backend solution that enables businesses to manage user subscriptions with flexibility and scalability.

Key Features

User Registration & Plan Selection
Streamlined signup process with multiple subscription options

Subscription Activation
Payment gateway integration with secure processing

Offer & Coupon Engine
Flexible discount application during signup or renewal

Plan Upgrade/Downgrade
Seamless transitions between subscription tiers

Microservices Architecture

Built with Spring Boot + Spring Cloud

Trial Period Logic
Automated 7-day trial management

Device Limit Management
Plan-based device restrictions and controls

Subscription History & Invoicing
Complete subscription record management

● Backend-only microservices with REST APIs

Architecture at a Glance

The Subscription Management System is built using a microservices architecture with Spring Boot and Spring Cloud, allowing independent development, deployment, and scaling of each component.

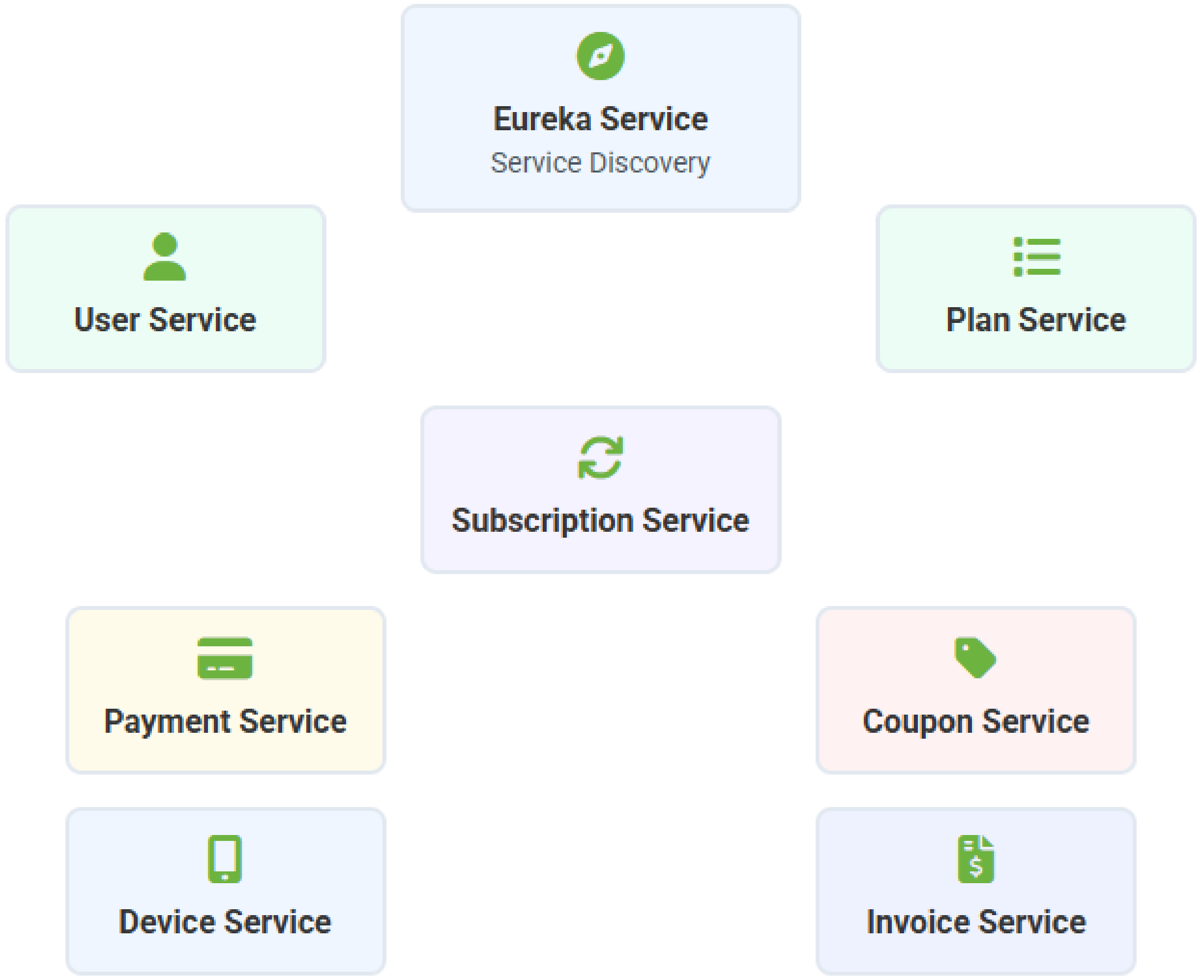
Key Architecture Benefits

Loose Coupling
Independent services with defined APIs

Scalability
Scale individual services based on demand

Database per Service
Each service has its own MySQL database

Service Discovery
Eureka enables automatic service registration



● All services communicate via REST APIs and Feign clients

Microservices Approach

Our subscription management system leverages microservices architecture with Spring Boot and Spring Cloud to achieve scalability, maintainability, and resilience.

Why Microservices?

Independent Development
Each team works on their service without affecting others

Independent Scaling
Scale individual components based on demand

Database per Service
Each service maintains its own data domain

Fault Isolation
System remains resilient when a service fails

Spring Boot + Spring Cloud

The perfect toolkit for microservices

Eureka Service Discovery
Automatic service registration and discovery

Feign Client
Simplified inter-service communication

Actuator Monitoring
Built-in health checks and metrics

OpenAPI/Swagger
Automatic API documentation generation

● Database per service pattern with REST APIs

User & Plan Service

Manage user accounts and subscription plans

Responsibilities:

- CRUD operations for Users
- CRUD operations for Plans (Monthly, Annual, Family)

`POST /users` `GET /users/{id}` `GET /plans`

Coupon Service

Manage promotional discounts and offers

Responsibilities:

- Manage flat discount coupons
- Validate coupon during subscription/payment

`POST /coupons` `POST /coupons/validate {code, subscriptionId}`

Subscription Service

Handle subscription lifecycle management

Responsibilities:

- Create subscription with userId + planId
- Manage status (TRIAL, ACTIVE, CANCELED)
- Simple trial flag + date field

`POST /subscriptions` `GET /subscriptions/{id}` `POST /subscriptions/{id}/end-trial`

Device Service

Track and control device access

Responsibilities:

- Manage devices for subscription (add/remove)
- Enforce plan device limit via Feign call to plan-service

`GET /devices?subscriptionId=` `POST /devices {subscriptionId, deviceId}` `DELETE /devices/{id}`

Payment Service

Process payments and activate subscriptions

Responsibilities:

- Mock payment confirmation API
- Call subscription-service via Feign to activate subscription

`POST /payments/confirm {subscriptionId, amount}`

Invoice Service

Generate and store billing records

Responsibilities:

- Generate invoice JSON record when payment succeeds
- Provide APIs to fetch invoices by subscription or ID

`POST /invoices {subscriptionId, amount}` `GET /invoices/{id}`

User & Plan Service

Responsibilities

- ✓ Create, Read, Update, Delete (CRUD) operations for Users
- ✓ CRUD operations for Plans (Monthly, Annual, Family)
- ✓ Registration with Eureka service discovery
- ✓ Swagger/OpenAPI documentation

API Endpoints

`POST` `/users`

Create new user with profile information

`GET` `/users/{id}`

Retrieve user details by ID

`GET` `/plans`

List all available subscription plans

Database Structure

Users Table

id	long(PK)
email	VARCHAR(255)
name	VARCHAR(100)
phone	VARCHAR(15)NOTNULL
created_at	TIMESTAMP
updated_At	TIMESTAMP

Plans Table

id	long(PK)
name	VARCHAR(50)
sku	VARCHAR(50)
description	VARCHAR(50)
status	VARCHAR(50)
price	DECIMAL(10,2)
deviceLimit	INT DEFAULT 1
maxProfile	INT DEFAULT 1
billing_cycle	ENUM('MONTHLY', 'QUARTERLY', 'YEARLY') NOT NULL

Subscription Service

Responsibilities

Create Subscription
Link users to subscription plans and manage the entire lifecycle

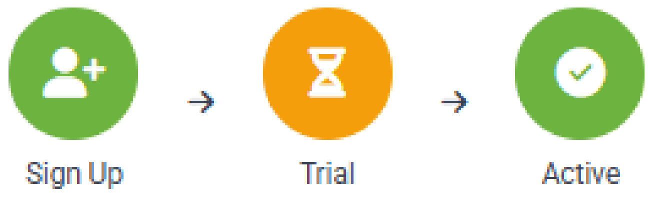
Trial Management
Implement 7-day free trial logic with automatic state transitions

Status Tracking
Monitor and transition between TRIAL, ACTIVE, and CANCELED states

Status Management

TRIAL	New subscriptions start with trial period
ACTIVE	After payment confirmation or trial completion
CANCELED	When subscription is terminated or expires

Subscription Lifecycle



API Endpoints

`POST` `/subscriptions`
`Create new subscription with userId and planId`

`GET` `/subscriptions/{id}`
`Retrieve subscription details and status`

`POST` `/subscriptions/{id}/end-trial`
`Manually end trial period and trigger status change`

Payment Service

The Payment Service team is responsible for processing payments and updating subscription status through inter-service communication.

Team Responsibilities

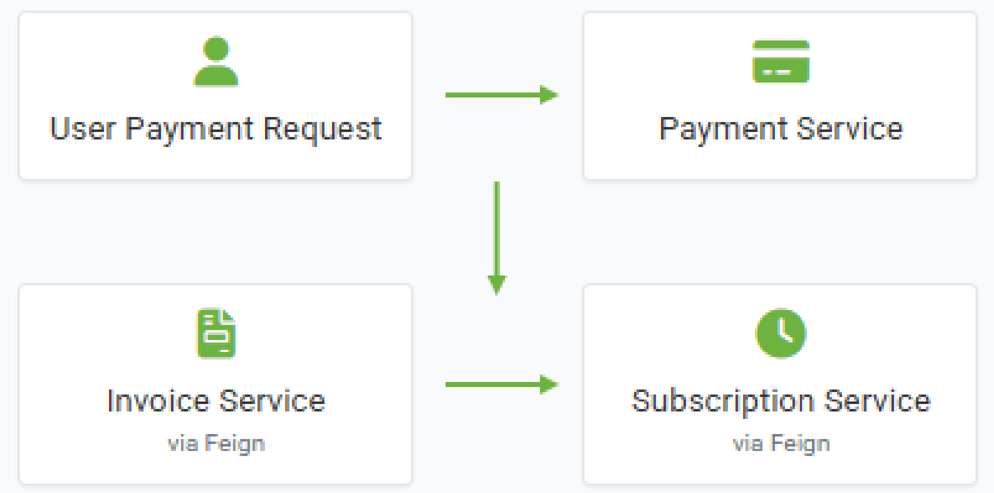
Mock Payment Integration
Simulate payment gateway interactions without real financial processing

Subscription Activation
On successful payment, activate subscription via Feign client

API Endpoint
`POST /payments/confirm {subscriptionId, amount}`

Health Monitoring
Actuator endpoints: /actuator/health and /info

Payment Flow



Payment Service uses `FeignClient` to communicate with other services

Coupon Service

Develops the Coupon Service that enables promotional codes and special offers to be applied during signup or subscription renewal.

Service Responsibilities

Flat Discount Management
Create, store, and manage coupon codes with fixed discount values

Coupon Validation
Validate coupon codes during subscription creation or renewal

Discount Calculation
Calculate final price after discount application

MySQL Database
Store coupon details, usage limits, and validity periods

API Endpoints

```
POST /coupons

Create new coupon with discount value

{
  "code": "SPRING25",
  "discountAmount": 25.00,
}
```

```
POST /coupons/validate

Validate coupon during checkout

{
  "code": "SPRING25",
  "subscriptionId": 12345
}
```

Integration Flow

Seamless integration with Payment Service via Feign client

Device Service

Managing devices linked to user subscriptions and enforcing device limits based on subscription plan rules.

Key Responsibilities

Device Management
Add, list, and remove devices linked to subscriptions

Plan Limit Enforcement
Validate device count against subscription plan limits

Service Integration
Communicate with Plan-Service via Feign client

REST APIs

`GET /devices?subscriptionId={id}`
List all devices for a subscription

`POST /devices`
Add a new device to subscription

```
{   "subscriptionId": "sub_123",   "deviceId": "vfgdr-324ddcf-ftf",
  "deviceName": "John's iPhone" }
```

`DELETE /devices/{id}`
Remove a device from subscription

Device Limit Flow

1. User requests to add new device
2. Service fetches current device count
3. Feign client calls Plan-Service to get limit
4. If limit not reached, device is added

Invoice Service

The Invoice Service, which generates and stores invoice records when payments are successfully processed.

Core Responsibilities

Invoice Generation
Create invoice JSON records upon successful payment

Retrieval APIs
Fetch invoices by subscription ID or invoice ID

Data Storage
Simplified invoice storage as JSON records in MySQL

API Endpoints

`POST /invoices`

Creates a new invoice record

```
{
  "subscriptionId": "sub_123",
  "amount": 99.00,
  "discount": 10.00,
  "total": 89.00
}
```

`GET /invoices/{id}`

Retrieves invoice by ID

Implementation Notes

- Invoice = simple DB record (JSON response, no PDF)
- Integration with Payment Service via Feign client
- Designed for future extensibility (e.g., PDF generation)
- Includes basic testing with @DataJpaTest and @WebMvcTest

API Design Standards

REST API Best Practices

Consistent Resource Naming
Use plural nouns for collections (/users, /subscriptions)

HTTP Methods
GET (read), POST (create), PUT/PATCH (update), DELETE (remove)

Standard Response Formats
Consistent JSON structure with proper status codes

Feign Client Communication

```
@FeignClient(name = "plan-service")
public interface PlanServiceClient {
    @GetMapping("/plans/{id}")
    PlanDTO getPlan(@PathVariable("id") Long id);
}
```

Benefits:

- Declarative REST client
- Automatic service discovery via Eureka
- Built-in load balancing

Swagger/OpenAPI Documentation

GET /subscriptions/{id} GET

Returns a subscription by its ID

Parameters:
id path Subscription ID (required)

Responses:
200 Subscription details
404 Subscription not found

Implementation

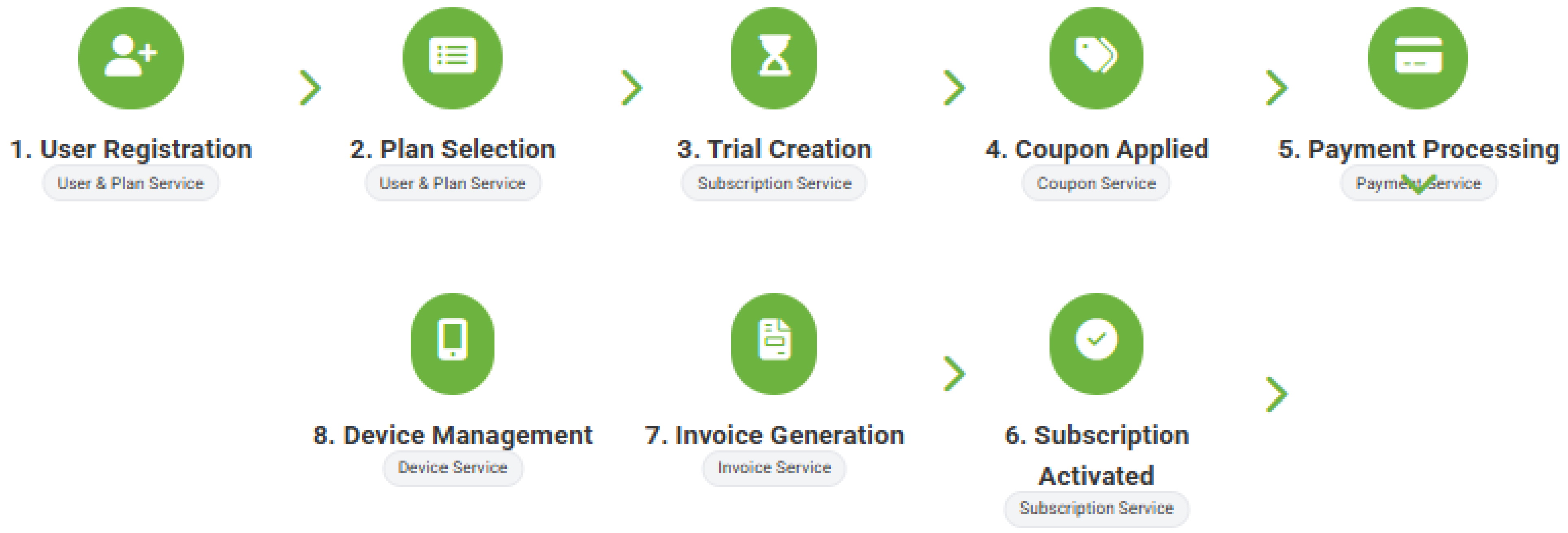
```
implementation 'org.springdoc:springdoc-openapi-ui:1.7.0'
```

Configuration
OpenAPI 3.0 configuration in Spring Boot

Annotations
@Operation, @ApiResponse, @Parameter

End-to-End Flow

Complete user journey through the Subscription Management System with all service interactions:



Service Communication Highlights

- Subscription Service calls User & Plan Service via Feign to validate plan details
- Payment Service activates subscription via Feign client call to Subscription Service
- Device Service enforces limits by calling Plan Service to check maximum allowed devices
- Invoice Service generates record when Payment Service confirms successful payment

Deployment, Discovery & Testing

Service Discovery

Eureka Registration

All microservices register with Eureka, enabling dynamic service discovery

```
@EnableEurekaClient
```

Spring Cloud Integration

Service-to-service communication via Feign clients using service names

```
@FeignClient(name = "subscription-service")
```

Monitoring

Spring Boot Actuator

Required endpoints:

- `/actuator/health` - Service health status
- `/actuator/info` - Service information

Database Strategy

Database Per Service

Each microservice maintains its own MySQL schema

- Data isolation and independence
- Prevents cross-service dependencies

Testing Requirements

Minimum Testing Standards

Each team must implement:

- `@DataJpaTest` - Repository layer testing
- `@WebMvcTest` - Controller layer testing

Team Deliverables

- Microservice registered in Eureka
- Swagger/OpenAPI docs at `/swagger-ui/index.html`
- Functional REST APIs with proper error handling
- README with setup steps & sample requests
- Passing minimal test suite

Thankyou