Gocashless Backend Microservices Design

This section elaborates on the design of each Spring Boot microservice, outlining their purpose, key entities, core functionalities (API endpoints), and interactions within the Gocashless system.

1. User Management Service (UMS)

- **Purpose:** The central authority for all user-related operations, including registration, authentication, authorization, and profile management for passengers, conductors, and bus company administrators.
- **Spring Boot Structure (with PostgreSQL):**

To implement the UMS with Spring Boot and PostgreSQL, you would typically set up the following:

  - **Maven Dependencies (pom.xml):**
  - <!-- Spring Boot Starter Web for REST APIs -->
  - <dependency>
  -     <groupId>org.springframework.boot</groupId>
  -     <artifactId>spring-boot-starter-web</artifactId>
  - </dependency>
  - <!-- Spring Boot Starter Data JPA for ORM and database interaction -->
  - <dependency>
  -     <groupId>org.springframework.boot</groupId>
  -     <artifactId>spring-boot-starter-data-jpa</artifactId>
  - </dependency>
  - <!-- PostgreSQL Driver -->
  - <dependency>
  -     <groupId>org.postgresql</groupId>
  -     <artifactId>postgresql</artifactId>
  -     <scope>runtime</scope>
  - </dependency>

- <!-- Lombok (Optional, for boilerplate reduction) -->
- <dependency>
-     <groupId>org.projectlombok</groupId>
-     <artifactId>lombok</artifactId>
-     <optional>true</optional>
- </dependency>
- <!-- Spring Boot Starter Security (for authentication/authorization) -->
- <dependency>
-     <groupId>org.springframework.boot</groupId>
-     <artifactId>spring-boot-starter-security</artifactId>
- </dependency>
- <!-- JJWT (for JWT token generation/validation) -->
- <dependency>
-     <groupId>io.jsonwebtoken</groupId>
-     <artifactId>jjwt-api</artifactId>
-     <version>0.11.5</version> <!-- Use a recent stable version -->
- </dependency>
- <dependency>
-     <groupId>io.jsonwebtoken</groupId>
-     <artifactId>jjwt-impl</artifactId>
-     <version>0.11.5</version>
-     <scope>runtime</scope>
- </dependency>
- <dependency>
-     <groupId>io.jsonwebtoken</groupId>
-     <artifactId>jjwt-jackson</artifactId>

-
- <version>0.11.5</version>
- <scope>runtime</scope>
- </dependency>

- **application.properties (or application.yml) for Database Configuration:**
- # PostgreSQL Database Configuration
- spring.datasource.url=jdbc:postgresql://localhost:5432/gocashless_ums_db
- spring.datasource.username=ums_user
- spring.datasource.password=ums_password
- spring.datasource.driver-class-name=org.postgresql.Driver
-
- # JPA/Hibernate Configuration
- spring.jpa.hibernate.ddl-auto=update # Use 'create' or 'create-drop' for development, 'none' or 'validate' for production
- spring.jpa.show-sql=true
- spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
-
- # JWT Secret (VERY IMPORTANT: Use a strong, securely stored secret in production)
- jwt.secret=yourSuperSecretKeyForJWTTokenGenerationAndValidationWhichShouldBeLongAndComplex
- jwt.expiration=3600000 # 1 hour in milliseconds

- **Package Structure:**
- com.gocashless.ums/
- ├── UmsApplication.java   // Main Spring Boot application class
- ├── config/

- | ├── SecurityConfig.java   // Spring Security configuration (JWT filters, etc.)
- | └── JwtUtil.java   // Utility for JWT token generation and validation
- ├── model/   // JPA Entities
- | ├── User.java
- | ├── BusCompany.java
- | └── ConductorProfile.java
- ├── repository/   // Spring Data JPA Repositories
- | ├── UserRepository.java
- | ├── BusCompanyRepository.java
- | └── ConductorProfileRepository.java
- ├── service/   // Business Logic
- | ├── UserService.java
- | ├── AuthService.java
- | └── BusCompanyService.java
- ├── controller/   // REST API Endpoints
- | ├── AuthController.java
- | ├── UserController.java
- | └── BusCompanyController.java
- └── dto/   // Data Transfer Objects (for request/response bodies)
-     ├── UserRegistrationRequest.java
-     ├── LoginRequest.java
-     └── UserResponse.java
-     └── ConductorRegistrationRequest.java

- **Example Classes:**

- **model/User.java (JPA Entity):**
- package com.gocashless.ums.model;
-
- import jakarta.persistence.*;
- import lombok.Data;
- import lombok.NoArgsConstructor;
- import lombok.AllArgsConstructor;
- import java.time.LocalDateTime;
- import java.util.UUID;
-
- @Entity
- @Table(name = "users") // Renamed to 'users' to avoid conflict with SQL 'user' keyword
- @Data
- @NoArgsConstructor
- @AllArgsConstructor
- public class User {
-   @Id
-   @GeneratedValue(strategy = GenerationType.UUID)
-   private UUID id;
-
-   @Column(unique = true, nullable = false)
-   private String username;
-
-   @Column(nullable = false)
-   private String passwordHash;

-
- @Column(unique = true)
- private String email;
-
- @Column(unique = true, nullable = false)
- private String phoneNumber;
-
- private String firstName;
- private String lastName;
-
- @Enumerated(EnumType.STRING)
- @Column(nullable = false)
- private Role role; // Enum: PASSENGER, CONDUCTOR, BUS_COMPANY_ADMIN, GOCASHLESS_ADMIN
-
- @Enumerated(EnumType.STRING)
- @Column(nullable = false)
- private UserStatus status; // Enum: ACTIVE, INACTIVE, BLOCKED
-
- private LocalDateTime createdAt;
- private LocalDateTime updatedAt;
-
- @PrePersist
- protected void onCreate() {
-     createdAt = LocalDateTime.now();
-     updatedAt = LocalDateTime.now();

- `    }`
-
- `    @PreUpdate`
- `    protected void onUpdate() {`
- `        updatedAt = LocalDateTime.now();`
- `    }`
- `}`
-
- `// Enums for Role and UserStatus`
- `enum Role {`
- `    PASSENGER, CONDUCTOR, BUS_COMPANY_ADMIN, GOCASHLESS_ADMIN`
- `}`
-
- `enum UserStatus {`
- `    ACTIVE, INACTIVE, BLOCKED`
- `}`

- **repository/UserRepository.java (Spring Data JPA Repository):**
- `package com.gocashless.ums.repository;`
-
- `import com.gocashless.ums.model.User;`
- `import org.springframework.data.jpa.repository.JpaRepository;`
- `import java.util.Optional;`
- `import java.util.UUID;`
-

- public interface UserRepository extends JpaRepository<User, UUID> {
-     Optional<User> findByUsername(String username);
-     Optional<User> findByPhoneNumber(String phoneNumber);
-     boolean existsByUsername(String username);
-     boolean existsByPhoneNumber(String phoneNumber);
- }

- **service/UserService.java (Service Layer):**
- package com.gocashless.ums.service;
-
- import com.gocashless.ums.model.User;
- import com.gocashless.ums.model.Role;
- import com.gocashless.ums.model.UserStatus;
- import com.gocashless.ums.repository.UserRepository;
- import com.gocashless.ums.dto.UserRegistrationRequest;
- import org.springframework.beans.factory.annotation.Autowired;
- import org.springframework.security.crypto.password.PasswordEncoder;
- import org.springframework.stereotype.Service;
-
- import java.util.Optional;
- import java.util.UUID;
-
- @Service
- public class UserService {
-
-     private final UserRepository userRepository;

```
▪     private final PasswordEncoder passwordEncoder;
▪
▪     @Autowired
▪     public UserService(UserRepository userRepository, PasswordEncoder passwordEncoder) {
▪         this.userRepository = userRepository;
▪         this.passwordEncoder = passwordEncoder;
▪     }
▪
▪     public User registerUser(UserRegistrationRequest request, Role role) {
▪         if (userRepository.existsByUsername(request.getUsername())) {
▪             throw new IllegalArgumentException("Username already exists.");
▪         }
▪         if (userRepository.existsByPhoneNumber(request.getPhoneNumber())) {
▪             throw new IllegalArgumentException("Phone number already registered.");
▪         }
▪
▪         User user = new User();
▪         user.setUsername(request.getUsername());
▪         user.setPasswordHash(passwordEncoder.encode(request.getPassword()));
▪         user.setEmail(request.getEmail());
▪         user.setPhoneNumber(request.getPhoneNumber());
▪         user.setFirstName(request.getFirstName());
▪         user.setLastName(request.getLastName());
▪         user.setRole(role);
▪         user.setStatus(UserStatus.ACTIVE); // Default status
▪
▪         return userRepository.save(user);
▪     }
▪
▪     public Optional<User> findUserById(UUID id) {
▪         return userRepository.findById(id);
▪     }
▪
▪     public Optional<User> findUserByUsername(String username) {
▪         return userRepository.findByUsername(username);
▪     }
▪
▪     // Other methods for updating user, managing status, etc.
▪ }
```

▪ **controller/UserController.java (REST Controller):**

```
▪ package com.gocashless.ums.controller;
▪
▪ import com.gocashless.ums.model.User;
▪ import com.gocashless.ums.model.Role;
▪ import com.gocashless.ums.service.UserService;
▪ import com.gocashless.ums.dto.UserRegistrationRequest;
▪ import org.springframework.beans.factory.annotation.Autowired;
▪ import org.springframework.http.HttpStatus;
▪ import org.springframework.http.ResponseEntity;
▪ import org.springframework.web.bind.annotation.*;
▪
▪ import java.util.UUID;
▪
▪ @RestController
▪ @RequestMapping("/api/v1/users")
▪ public class UserController {
▪
▪     private final UserService userService;
▪
▪     @Autowired
▪     public UserController(UserService userService) {
▪         this.userService = userService;
▪     }
▪
▪     @PostMapping("/register/passenger")
▪     public ResponseEntity<?> registerPassenger(@RequestBody UserRegistrationRequest request) {
▪         try {
▪             User newUser = userService.registerUser(request, Role.PASSENGER);
▪             return new ResponseEntity<>(newUser, HttpStatus.CREATED);
▪         } catch (IllegalArgumentException e) {
▪             return new ResponseEntity<>(e.getMessage(), HttpStatus.BAD_REQUEST);
▪         }
```

- }
-
- @GetMapping("/{id}")
- public ResponseEntity<User> getUserById(@PathVariable UUID id) {
- return userService.findUserById(id)
- .map(user -> new ResponseEntity<>(user, HttpStatus.OK))
- .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
- }
-
- // Add endpoints for updating users, managing conductors/bus companies, etc.
- }

- **dto/UserRegistrationRequest.java (DTO):**
- package com.gocashless.ums.dto;
-
- import lombok.Data;
-
- @Data
- public class UserRegistrationRequest {
- private String username;
- private String password;
- private String email;
- private String phoneNumber;
- private String firstName;
- private String lastName;

```
}
```

- **config/SecurityConfig.java (Spring Security Configuration - simplified):**

```
package com.gocashless.ums.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

```

- @Bean
- public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
- http
- .csrf(csrf -> csrf.disable()) // Disable CSRF for API endpoints
- .authorizeHttpRequests(authorize -> authorize
- .requestMatchers("/api/v1/users/register/**", "/api/v1/users/login").permitAll() // Allow public access to registration and login
- .anyRequest().authenticated() // All other requests require authentication
-);
- // You would add JWT filter here later
- return http.build();
- }
- }

- **config/JwtUtil.java (JWT Utility - simplified):**
- package com.gocashless.ums.config;
-
- import io.jsonwebtoken.Claims;
- import io.jsonwebtoken.Jwts;
- import io.jsonwebtoken.SignatureAlgorithm;
- import io.jsonwebtoken.security.Keys;
- import org.springframework.beans.factory.annotation.Value;
- import org.springframework.stereotype.Component;
-

```
import javax.crypto.SecretKey;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

@Component
public class JwtUtil {

    @Value("${jwt.secret}")
    private String secretString;

    @Value("${jwt.expiration}")
    private long expiration;

    private SecretKey key;

    // Initialize the key from the secret string
    // This should be called once, e.g., in a @PostConstruct method
    private SecretKey getSigningKey() {
        if (key == null) {
            key = Keys.hmacShaKeyFor(secretString.getBytes());
        }
        return key;
    }

    public String generateToken(String username) {
        Map<String, Object> claims = new HashMap<>();
        return createToken(claims, username);
    }

    private String createToken(Map<String, Object> claims, String subject) {
        return Jwts.builder()
                .setClaims(claims)
                .setSubject(subject)
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + expiration))
                .signWith(getSigningKey(), SignatureAlgorithm.HS256)
                .compact();
    }

    public Boolean validateToken(String token, String username) {
        final String extractedUsername = extractUsername(token);
        return (extractedUsername.equals(username) && !isTokenExpired(token));
    }

    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public Date extractExpiration(String token) {
```

        - return extractClaim(token, Claims::getExpiration);
        - }
        -
        - public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
        - final Claims claims = extractAllClaims(token);
        - return claimsResolver.apply(claims);
        - }
        -
        - private Claims extractAllClaims(String token) {
        - return Jwts.parserBuilder().setSigningKey(getSigningKey()).build().parseClaimsJws(token).getBody();
        - }
        -
        - private Boolean isTokenExpired(String token) {
        - return extractExpiration(token).before(new Date());
        - }
        - }

- **Key Entities/Data Models:**
    - **User:**
        - id (UUID)
        - username (String, unique)
        - passwordHash (String)
        - email (String, unique, optional)
        - phoneNumber (String, unique)

        - firstName (String)
        - lastName (String)
        - role (Enum: PASSENGER, CONDUCTOR, BUS_COMPANY_ADMIN, GOCASHLESS_ADMIN)
        - status (Enum: ACTIVE, INACTIVE, BLOCKED)
        - createdAt (Timestamp)
        - updatedAt (Timestamp)
    - **BusCompany:**
        - id (UUID)
        - name (String, unique)
        - address (String)
        - contactPersonId (UUID, foreign key to User with BUS_COMPANY_ADMIN role)
        - createdAt (Timestamp)
        - updatedAt (Timestamp)
    - **ConductorProfile:**
        - id (UUID)
        - userId (UUID, foreign key to User with CONDUCTOR role)
        - busCompanyId (UUID, foreign key to BusCompany)
        - employeeId (String, unique within company)
        - status (Enum: ACTIVE, INACTIVE)
        - createdAt (Timestamp)
        - updatedAt (Timestamp)
- **Core Functionality/APIs:**
    - POST /api/v1/users/register: Register a new user (passenger, conductor, or bus company admin).
    - POST /api/v1/users/login: Authenticate user and issue JWT token.

  - GET /api/v1/users/{id}: Retrieve user profile by ID (authorized).
  - PUT /api/v1/users/{id}: Update user profile (authorized).
  - POST /api/v1/bus-companies: Register a new bus company (Gocashless Admin only).
  - GET /api/v1/bus-companies/{id}: Retrieve bus company details.
  - POST /api/v1/conductors: Register a new conductor (Bus Company Admin only).
  - GET /api/v1/bus-companies/{companyId}/conductors: List conductors for a specific bus company.
  - PUT /api/v1/conductors/{id}/status: Update conductor status (e.g., active/inactive).
- **Interactions:**
  - **Frontend Apps (Passenger, Conductor, Bus Company Web):** Interact directly for user registration, login, and profile management.
  - **QR Code Generation Service (QRGS):** May query UMS for conductor details to embed in QR codes.
  - **Transaction History Service (THS):** Queries UMS for user details (names, roles) to enrich transaction data for reporting.

2. Route & Fare Management Service (RFMS)

- **Purpose:** Manages all public transport routes, bus stops, and the dynamic fare matrix between different stops.
- **Spring Boot Structure (with PostgreSQL and Eureka Client):**

To implement the RFMS with Spring Boot, PostgreSQL, and integrate with Eureka, you would typically set up the following:

  - **Maven Dependencies (pom.xml):**
  - <?xml version="1.0" encoding="UTF-8"?>
  - <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  -         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

```xml
  <modelVersion>4.0.0</modelVersion>
  <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>3.3.1</version> <!-- Use the same Spring Boot version as other services -->
      <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.gocashless</sgroupId>
  <artifactId>route-fare-management-service</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>route-fare-management-service</name>
  <description>Gocashless Route &amp; Fare Management Service</description>

  <properties>
      <java.version>17</java.version>
      <spring-cloud.version>2023.0.2</spring-cloud.version> <!-- Ensure compatibility with Spring Boot 3.3.1 -->
  </properties>

  <dependencies>
      <!-- Spring Boot Starter Web for REST APIs -->
      <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-web</artifactId>
      </dependency>
```

```
o   <!-- Spring Boot Starter Data JPA for ORM and database interaction -->
o   <dependency>
o     <groupId>org.springframework.boot</groupId>
o     <artifactId>spring-boot-starter-data-jpa</artifactId>
o   </dependency>
o   <!-- PostgreSQL Driver -->
o   <dependency>
o     <groupId>org.postgresql</groupId>
o     <artifactId>postgresql</artifactId>
o     <scope>runtime</scope>
o   </dependency>
o   <!-- Lombok (Optional, for boilerplate reduction) -->
o   <dependency>
o     <groupId>org.projectlombok</groupId>
o     <artifactId>lombok</artifactId>
o     <optional>true</optional>
o   </dependency>
o   <!-- Spring Boot DevTools (Optional, for faster development) -->
o   <dependency>
o     <groupId>org.springframework.boot</groupId>
o     <artifactId>spring-boot-devtools</artifactId>
o     <scope>runtime</scope>
o     <optional>true</optional>
o   </dependency>
o   <!-- Spring Boot Starter Test -->
o   <dependency>
```

```
o           <groupId>org.springframework.boot</groupId>
o           <artifactId>spring-boot-starter-test</artifactId>
o           <scope>test</scope>
o       </dependency>
o
o       <!-- Spring Cloud Starter Netflix Eureka Client for Service Discovery -->
o       <dependency>
o           <groupId>org.springframework.cloud</groupId>
o           <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
o       </dependency>
o   </dependencies>
o
o   <dependencyManagement>
o       <dependencies>
o           <dependency>
o               <groupId>org.springframework.cloud</groupId>
o               <artifactId>spring-cloud-dependencies</artifactId>
o               <version>${spring-cloud.version}</version>
o               <type>pom</type>
o               <scope>import</scope>
o           </dependency>
o       </dependencies>
o   </dependencyManagement>
o
o   <build>
o       <plugins>
```

```
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

- **application.yml for Database and Eureka Configuration:**

```
server:
  port: 8081 # Unique port for RFMS, e.g., 8081

spring:
  application:
    name: route-fare-management-service # Name for Eureka registration
  datasource:
    url: jdbc:postgresql://localhost:5432/gocashless_rfms_db # Dedicated DB for RFMS
    username: rfms_user
    password: rfms_password
    driver-class-name: org.postgresql.Driver
  jpa:
    hibernate:
      ddl-auto: update # 'update' for development, 'none' or 'validate' for production
    show-sql: true
    properties:
      hibernate:
        dialect: org.hibernate.dialect.PostgreSQLDialect

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka # URL of your Eureka Server
    fetch-registry: true
    register-with-eureka: true
  instance:
    hostname: localhost
```

- **Package Structure:**

```
com.gocashless.rfms/
 ├── RfmsApplication.java       // Main Spring Boot application class
 ├── model/                     // JPA Entities
|    ├── Route.java
```

- | ├── BusStop.java
- | └── Fare.java
- ├── repository/ // Spring Data JPA Repositories
- | ├── RouteRepository.java
- | ├── BusStopRepository.java
- | └── FareRepository.java
- ├── service/ // Business Logic
- | ├── RouteService.java
- | ├── BusStopService.java
- | └── FareService.java
- ├── controller/ // REST API Endpoints
- | ├── RouteController.java
- | ├── BusStopController.java
- | └── FareController.java
- └── dto/ // Data Transfer Objects (for request/response bodies)
- ├── RouteRequest.java
- ├── BusStopRequest.java
- └── FareRequest.java
- └── RouteResponse.java

- **Example Classes:**
  - **RfmsApplication.java (Main Application Class):**
  - package com.gocashless.rfms;
  -
  - import org.springframework.boot.SpringApplication;

- import org.springframework.boot.autoconfigure.SpringBootApplication;
- import org.springframework.cloud.client.discovery.EnableDiscoveryClient; // Import for Eureka client
-
- @SpringBootApplication
- @EnableDiscoveryClient // Enable this application as a Eureka client
- public class RfmsApplication {
-
-     public static void main(String[] args) {
-         SpringApplication.run(RfmsApplication.class, args);
-     }
- }

- **model/Route.java (JPA Entity):**
- package com.gocashless.rfms.model;
-
- import jakarta.persistence.*;
- import lombok.Data;
- import lombok.NoArgsConstructor;
- import lombok.AllArgsConstructor;
- import java.time.LocalDateTime;
- import java.util.UUID;
-
- @Entity
- @Table(name = "routes")
- @Data

```java
@NoArgsConstructor
@AllArgsConstructor
public class Route {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;

    @Column(unique = true, nullable = false)
    private String name;

    private String description;

    @Column(nullable = false)
    private Boolean isActive = true; // Default to active

    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;

    @PrePersist
    protected void onCreate() {
        createdAt = LocalDateTime.now();
        updatedAt = LocalDateTime.now();
    }

    @PreUpdate
    protected void onUpdate() {
```

- updatedAt = LocalDateTime.now();
- }
- }

- **model/BusStop.java (JPA Entity):**
- package com.gocashless.rfms.model;
-
- import jakarta.persistence.*;
- import lombok.Data;
- import lombok.NoArgsConstructor;
- import lombok.AllArgsConstructor;
- import java.time.LocalDateTime;
- import java.util.UUID;
-
- @Entity
- @Table(name = "bus_stops")
- @Data
- @NoArgsConstructor
- @AllArgsConstructor
- public class BusStop {
- @Id
- @GeneratedValue(strategy = GenerationType.UUID)
- private UUID id;
-
- @Column(nullable = false)
- private String name;

```

    private Double latitude;
    private Double longitude;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "route_id") // Foreign key to Route
    private Route route; // If a stop is specifically tied to one route

    private Integer orderInRoute; // For sequential stops on a route

    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;

    @PrePersist
    protected void onCreate() {
        createdAt = LocalDateTime.now();
        updatedAt = LocalDateTime.now();
    }

    @PreUpdate
    protected void onUpdate() {
        updatedAt = LocalDateTime.now();
    }
}
```

- **model/Fare.java (JPA Entity):**

```java
package com.gocashless.rfms.model;

import jakarta.persistence.*;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;
import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.UUID;

@Entity
@Table(name = "fares")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Fare {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "route_id", nullable = false)
    private Route route;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "origin_stop_id", nullable = false)
```

```
▪     private BusStop originStop;
▪
▪     @ManyToOne(fetch = FetchType.LAZY)
▪     @JoinColumn(name = "destination_stop_id", nullable = false)
▪     private BusStop destinationStop;
▪
▪     @Column(nullable = false, precision = 10, scale = 2) // Precision for
    currency
▪     private BigDecimal amount;
▪
▪     @Column(nullable = false, length = 3)
▪     private String currency; // e.g., "ZMW"
▪
▪     private LocalDateTime validFrom;
▪     private LocalDateTime validTo; // Optional, for future fare changes
▪
▪     private LocalDateTime createdAt;
▪     private LocalDateTime updatedAt;
▪
▪     @PrePersist
▪     protected void onCreate() {
▪         createdAt = LocalDateTime.now();
▪         updatedAt = LocalDateTime.now();
▪     }
▪
▪     @PreUpdate
```

```java
    protected void onUpdate() {
        updatedAt = LocalDateTime.now();
    }
}
```

- **repository/RouteRepository.java (Spring Data JPA Repository):**

```java
package com.gocashless.rfms.repository;

import com.gocashless.rfms.model.Route;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;
import java.util.UUID;

public interface RouteRepository extends JpaRepository<Route, UUID> {
    Optional<Route> findByName(String name);
    boolean existsByName(String name);
}
```

- **repository/BusStopRepository.java (Spring Data JPA Repository):**

```java
package com.gocashless.rfms.repository;

import com.gocashless.rfms.model.BusStop;
import com.gocashless.rfms.model.Route;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;
import java.util.Optional;
```

- import java.util.UUID;
-
- public interface BusStopRepository extends JpaRepository<BusStop, UUID> {
-     Optional<BusStop> findByName(String name);
-     List<BusStop> findByRouteOrderByOrderInRouteAsc(Route route);
- }

- **repository/FareRepository.java (Spring Data JPA Repository):**
- package com.gocashless.rfms.repository;
-
- import com.gocashless.rfms.model.Fare;
- import com.gocashless.rfms.model.Route;
- import com.gocashless.rfms.model.BusStop;
- import org.springframework.data.jpa.repository.JpaRepository;
- import java.time.LocalDateTime;
- import java.util.Optional;
- import java.util.UUID;
-
- public interface FareRepository extends JpaRepository<Fare, UUID> {
-     // Find fare for a specific route, origin, and destination valid now
-     Optional<Fare> findByRouteAndOriginStopAndDestinationStopAndValidFromBeforeAndValidToAfterOrValidToIsNull(
-         Route route, BusStop originStop, BusStop destinationStop, LocalDateTime now1, LocalDateTime now2);
-

```
    // Find current fare for a specific route, origin, and destination (simplified)
    Optional<Fare> findFirstByRouteAndOriginStopAndDestinationStopOrderByValidFromDesc(
        Route route, BusStop originStop, BusStop destinationStop);
}
```

- **service/RouteService.java (Service Layer):**

```
package com.gocashless.rfms.service;

import com.gocashless.rfms.model.Route;
import com.gocashless.rfms.repository.RouteRepository;
import com.gocashless.rfms.dto.RouteRequest;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;
import java.util.UUID;

@Service
public class RouteService {

    private final RouteRepository routeRepository;

    @Autowired
public RouteService(RouteRepository routeRepository) {
this.routeRepository = routeRepository;
}

public Route createRoute(RouteRequest request) {
if (routeRepository.existsByName(request.getName())) {
throw new IllegalArgumentException("Route with this name already exists.");
}
Route route = new Route();
route.setName(request.getName());
route.setDescription(request.getDescription());
route.setIsActive(request.getIsActive() != null ? request.getIsActive() : true);
return routeRepository.save(route);
}

public List<Route> getAllRoutes() {
return routeRepository.findAll();
}

public Optional<Route> getRouteById(UUID id) {
return routeRepository.findById(id);
}

public Route updateRoute(UUID id, RouteRequest request) {
return routeRepository.findById(id).map(route -> {
```

```
▪           route.setName(request.getName());
▪           route.setDescription(request.getDescription());
▪           route.setIsActive(request.getIsActive());
▪           return routeRepository.save(route);
▪         }).orElseThrow(() -> new IllegalArgumentException("Route not found
  with ID: " + id));
▪     }
▪
▪     public void deleteRoute(UUID id) {
▪         routeRepository.deleteById(id);
▪     }
▪ }
```

- **service/BusStopService.java (Service Layer):**

```
▪ package com.gocashless.rfms.service;
▪
▪ import com.gocashless.rfms.model.BusStop;
▪ import com.gocashless.rfms.model.Route;
▪ import com.gocashless.rfms.repository.BusStopRepository;
▪ import com.gocashless.rfms.repository.RouteRepository;
▪ import com.gocashless.rfms.dto.BusStopRequest;
▪ import org.springframework.beans.factory.annotation.Autowired;
▪ import org.springframework.stereotype.Service;
▪
▪ import java.util.List;
▪ import java.util.Optional;
```

```java
import java.util.UUID;

@Service
public class BusStopService {

    private final BusStopRepository busStopRepository;
    private final RouteRepository routeRepository;

    @Autowired
    public BusStopService(BusStopRepository busStopRepository, RouteRepository routeRepository) {
        this.busStopRepository = busStopRepository;
        this.routeRepository = routeRepository;
    }

    public BusStop createBusStop(BusStopRequest request) {
        BusStop busStop = new BusStop();
        busStop.setName(request.getName());
        busStop.setLatitude(request.getLatitude());
        busStop.setLongitude(request.getLongitude());

        if (request.getRouteId() != null) {
            Route route = routeRepository.findById(request.getRouteId())
                    .orElseThrow(() -> new IllegalArgumentException("Route not found with ID: " + request.getRouteId()));
            busStop.setRoute(route);
            busStop.setOrderInRoute(request.getOrderInRoute());
        }

        return busStopRepository.save(busStop);
    }

    public List<BusStop> getAllBusStops() {
        return busStopRepository.findAll();
    }

    public Optional<BusStop> getBusStopById(UUID id) {
        return busStopRepository.findById(id);
    }

    public List<BusStop> getBusStopsByRoute(UUID routeId) {
        Route route = routeRepository.findById(routeId)
                .orElseThrow(() -> new IllegalArgumentException("Route not found with ID: " + routeId));
        return busStopRepository.findByRouteOrderByOrderInRouteAsc(route);
    }

    // Other update and delete methods
}
```

- **service/FareService.java (Service Layer):**

```
package com.gocashless.rfms.service;

```

```
▪ import com.gocashless.rfms.model.Fare;
▪ import com.gocashless.rfms.model.Route;
▪ import com.gocashless.rfms.model.BusStop;
▪ import com.gocashless.rfms.repository.FareRepository;
▪ import com.gocashless.rfms.repository.RouteRepository;
▪ import com.gocashless.rfms.repository.BusStopRepository;
▪ import com.gocashless.rfms.dto.FareRequest;
▪ import org.springframework.beans.factory.annotation.Autowired;
▪ import org.springframework.stereotype.Service;
▪
▪ import java.time.LocalDateTime;
▪ import java.util.Optional;
▪ import java.util.UUID;
▪
▪ @Service
▪ public class FareService {
▪
▪     private final FareRepository fareRepository;
▪     private final RouteRepository routeRepository;
▪     private final BusStopRepository busStopRepository;
▪
▪     @Autowired
▪     public FareService(FareRepository fareRepository, RouteRepository routeRepository, BusStopRepository busStopRepository) {
▪         this.fareRepository = fareRepository;
▪         this.routeRepository = routeRepository;
▪         this.busStopRepository = busStopRepository;
▪     }
▪
▪     public Fare createFare(FareRequest request) {
▪         Route route = routeRepository.findById(request.getRouteId())
▪                 .orElseThrow(() -> new IllegalArgumentException("Route not found with ID: " + request.getRouteId()));
▪         BusStop originStop = busStopRepository.findById(request.getOriginStopId())
▪                 .orElseThrow(() -> new IllegalArgumentException("Origin Stop not found with ID: " + request.getOriginStopId()));
▪         BusStop destinationStop = busStopRepository.findById(request.getDestinationStopId())
▪                 .orElseThrow(() -> new IllegalArgumentException("Destination Stop not found with ID: " + request.getDestinationStopId()));
▪
▪         Fare fare = new Fare();
▪         fare.setRoute(route);
▪         fare.setOriginStop(originStop);
▪         fare.setDestinationStop(destinationStop);
▪         fare.setAmount(request.getAmount());
▪         fare.setCurrency(request.getCurrency());
▪         fare.setValidFrom(request.getValidFrom() != null ? request.getValidFrom() : LocalDateTime.now());
▪         fare.setValidTo(request.getValidTo());
▪
▪         return fareRepository.save(fare);
▪     }
```

```

    public Optional<Fare> getFareById(UUID id) {
        return fareRepository.findById(id);
    }

    public Optional<Fare> getFareForJourney(UUID routeId, UUID originStopId, UUID destinationStopId) {
        Route route = routeRepository.findById(routeId)
                .orElseThrow(() -> new IllegalArgumentException("Route not found with ID: " + routeId));
        BusStop originStop = busStopRepository.findById(originStopId)
                .orElseThrow(() -> new IllegalArgumentException("Origin Stop not found with ID: " + originStopId));
        BusStop destinationStop = busStopRepository.findById(destinationStopId)
                .orElseThrow(() -> new IllegalArgumentException("Destination Stop not found with ID: " + destinationStopId));

        // This method finds the most recently valid fare
        return fareRepository.findFirstByRouteAndOriginStopAndDestinationStopOrderByValidFromDesc(
                route, originStop, destinationStop
        );
    }

    // Other update and delete methods
}
```

- **controller/RouteController.java (REST Controller):**
- package com.gocashless.rfms.controller;
-
- import com.gocashless.rfms.model.Route;
- import com.gocashless.rfms.service.RouteService;
- import com.gocashless.rfms.dto.RouteRequest;
- import org.springframework.beans.factory.annotation.Autowired;
- import org.springframework.http.HttpStatus;
- import org.springframework.http.ResponseEntity;
- import org.springframework.web.bind.annotation.*;
-
- import java.util.List;
- import java.util.UUID;
-
- @RestController
- @RequestMapping("/api/v1/routes")
- public class RouteController {
-
-     private final RouteService routeService;
-
-     @Autowired
-     public RouteController(RouteService routeService) {
-         this.routeService = routeService;
-     }
-

```java
    @PostMapping
    public ResponseEntity<Route> createRoute(@RequestBody RouteRequest request) {
        try {
            Route newRoute = routeService.createRoute(request);
            return new ResponseEntity<>(newRoute, HttpStatus.CREATED);
        } catch (IllegalArgumentException e) {
            return new ResponseEntity(e.getMessage(), HttpStatus.BAD_REQUEST);
        }
    }

    @GetMapping
    public ResponseEntity<List<Route>> getAllRoutes() {
        List<Route> routes = routeService.getAllRoutes();
        return new ResponseEntity<>(routes, HttpStatus.OK);
    }

    @GetMapping("/{id}")
    public ResponseEntity<Route> getRouteById(@PathVariable UUID id) {
        return routeService.getRouteById(id)
                .map(route -> new ResponseEntity<>(route, HttpStatus.OK))
                .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    @PutMapping("/{id}")
```

- public ResponseEntity<Route> updateRoute(@PathVariable UUID id, @RequestBody RouteRequest request) {
- try {
- Route updatedRoute = routeService.updateRoute(id, request);
- return new ResponseEntity<>(updatedRoute, HttpStatus.OK);
- } catch (IllegalArgumentException e) {
- return new ResponseEntity(e.getMessage(), HttpStatus.NOT_FOUND);
- }
- }
-
- @DeleteMapping("/{id}")
- public ResponseEntity<Void> deleteRoute(@PathVariable UUID id) {
- routeService.deleteRoute(id);
- return new ResponseEntity<>(HttpStatus.NO_CONTENT);
- }
- }

- **controller/BusStopController.java (REST Controller):**
- package com.gocashless.rfms.controller;
-
- import com.gocashless.rfms.model.BusStop;
- import com.gocashless.rfms.service.BusStopService;
- import com.gocashless.rfms.dto.BusStopRequest;
- import org.springframework.beans.factory.annotation.Autowired;
- import org.springframework.http.HttpStatus;
- import org.springframework.http.ResponseEntity;

```java
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.UUID;

@RestController
@RequestMapping("/api/v1/bus-stops")
public class BusStopController {

    private final BusStopService busStopService;

    @Autowired
    public BusStopController(BusStopService busStopService) {
        this.busStopService = busStopService;
    }

    @PostMapping
    public ResponseEntity<BusStop> createBusStop(@RequestBody BusStopRequest request) {
        try {
            BusStop newBusStop = busStopService.createBusStop(request);
            return new ResponseEntity<>(newBusStop, HttpStatus.CREATED);
        } catch (IllegalArgumentException e) {
            return new ResponseEntity(e.getMessage(), HttpStatus.BAD_REQUEST);
        }
    }

    @GetMapping
    public ResponseEntity<List<BusStop>> getAllBusStops() {
        List<BusStop> busStops = busStopService.getAllBusStops();
        return new ResponseEntity<>(busStops, HttpStatus.OK);
    }

    @GetMapping("/{id}")
    public ResponseEntity<BusStop> getBusStopById(@PathVariable UUID id) {
        return busStopService.getBusStopById(id)
                .map(busStop -> new ResponseEntity<>(busStop, HttpStatus.OK))
                .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    @GetMapping("/by-route/{routeId}")
    public ResponseEntity<List<BusStop>> getBusStopsByRoute(@PathVariable UUID routeId) {
        try {
            List<BusStop> busStops = busStopService.getBusStopsByRoute(routeId);
            return new ResponseEntity<>(busStops, HttpStatus.OK);
        } catch (IllegalArgumentException e) {
            return new ResponseEntity(e.getMessage(), HttpStatus.NOT_FOUND);
        }
```

```
▪     }
▪ }
```

▪ **controller/FareController.java (REST Controller):**

```
▪ package com.gocashless.rfms.controller;
▪
▪ import com.gocashless.rfms.model.Fare;
▪ import com.gocashless.rfms.service.FareService;
▪ import com.gocashless.rfms.dto.FareRequest;
▪ import org.springframework.beans.factory.annotation.Autowired;
▪ import org.springframework.http.HttpStatus;
▪ import org.springframework.http.ResponseEntity;
▪ import org.springframework.web.bind.annotation.*;
▪
▪ import java.util.Optional;
▪ import java.util.UUID;
▪
▪ @RestController
▪ @RequestMapping("/api/v1/fares")
▪ public class FareController {
▪
▪     private final FareService fareService;
▪
▪     @Autowired
▪     public FareController(FareService fareService) {
▪         this.fareService = fareService;
```

```
    }

    @PostMapping
    public ResponseEntity<Fare> createFare(@RequestBody FareRequest request) {
        try {
            Fare newFare = fareService.createFare(request);
            return new ResponseEntity<>(newFare, HttpStatus.CREATED);
        } catch (IllegalArgumentException e) {
            return new ResponseEntity(e.getMessage(), HttpStatus.BAD_REQUEST);
        }
    }

    @GetMapping("/{id}")
    public ResponseEntity<Fare> getFareById(@PathVariable UUID id) {
        return fareService.getFareById(id)
                .map(fare -> new ResponseEntity<>(fare, HttpStatus.OK))
                .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    @GetMapping("/lookup")
    public ResponseEntity<Fare> getFareForJourney(
            @RequestParam UUID routeId,
            @RequestParam UUID originStopId,
            @RequestParam UUID destinationStopId) {
        try {
```

- Optional<Fare> fare = fareService.getFareForJourney(routeId, originStopId, destinationStopId);
- return fare.map(f -> new ResponseEntity<>(f, HttpStatus.OK))
- .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
- } catch (IllegalArgumentException e) {
- return new ResponseEntity(e.getMessage(), HttpStatus.BAD_REQUEST);
- }
- }
- }

- **dto/RouteRequest.java (DTO):**
- package com.gocashless.rfms.dto;
-
- import lombok.Data;
-
- @Data
- public class RouteRequest {
- private String name;
- private String description;
- private Boolean isActive;
- }

- **dto/BusStopRequest.java (DTO):**
- package com.gocashless.rfms.dto;
-
- import lombok.Data;

- import java.util.UUID;
-
- @Data
- public class BusStopRequest {
-     private String name;
-     private Double latitude;
-     private Double longitude;
-     private UUID routeId; // Optional, if a stop belongs to a specific route
-     private Integer orderInRoute; // Optional, for ordering stops on a route
- }

- **dto/FareRequest.java (DTO):**
- package com.gocashless.rfms.dto;
-
- import lombok.Data;
- import java.math.BigDecimal;
- import java.time.LocalDateTime;
- import java.util.UUID;
-
- @Data
- public class FareRequest {
-     private UUID routeId;
-     private UUID originStopId;
-     private UUID destinationStopId;
-     private BigDecimal amount;
-     private String currency;

        - private LocalDateTime validFrom;
        - private LocalDateTime validTo;
        - }

- **Key Entities/Data Models:**
    - **Route:**
        - id (UUID)
        - name (String, e.g., "Lusaka City Centre - Matero")
        - description (String)
        - isActive (Boolean)
        - createdAt (Timestamp)
        - updatedAt (Timestamp)
    - **BusStop:**
        - id (UUID)
        - name (String, e.g., "Kulima Tower")
        - latitude (Double)
        - longitude (Double)
        - routeId (UUID, foreign key to Route, if a stop belongs to a specific route)
        - orderInRoute (Integer, for sequential stops on a route)
        - createdAt (Timestamp)
        - updatedAt (Timestamp)
    - **Fare:**
        - id (UUID)
        - routeId (UUID)
        - originStopId (UUID)
        - destinationStopId (UUID)

        - amount (BigDecimal)
        - currency (String, e.g., "ZMW")
        - validFrom (Timestamp)
        - validTo (Timestamp, optional for future fare changes)
        - createdAt (Timestamp)
        - updatedAt (Timestamp)

- **Core Functionality/APIs:**
    - POST /api/v1/routes: Create a new route.
    - GET /api/v1/routes: List all routes.
    - GET /api/v1/routes/{id}/stops: Get all stops for a specific route.
    - POST /api/v1/bus-stops: Add a new bus stop.
    - GET /api/v1/bus-stops: List all bus stops.
    - POST /api/v1/fares: Set a new fare between two stops on a route.
    - GET /api/v1/fares/lookup: Get fare for a specific journey (using query parameters for route, origin, destination).
    - GET /api/v1/fares/route/{routeId}/origin/{originStopId}: (Not implemented in example, but would return all fares from an origin on a route)
- **Interactions:**
    - **Conductor App:** Queries RFMS to retrieve bus stops and corresponding fares for QR code generation.
    - **QR Code Generation Service (QRGS):** May validate fare information by calling RFMS before embedding it in the QR code.

3. QR Code Generation Service (QRGS)

- **Purpose:** Generates secure QR codes containing encrypted payment details based on conductor input and fare information.
- **Spring Boot Structure (with Eureka Client and ZXing):**

To implement the QRGS with Spring Boot, integrate with Eureka, and use ZXing for QR code generation, you would typically set up the following:

- **Maven Dependencies (pom.xml):**
- <?xml version="1.0" encoding="UTF-8"?>
- <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
-          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
-     <modelVersion>4.0.0</modelVersion>
-     <parent>
-         <groupId>org.springframework.boot</groupId>
-         <artifactId>spring-boot-starter-parent</artifactId>
-         <version>3.3.1</version> <!-- Use the same Spring Boot version as other services -->
-         <relativePath/> <!-- lookup parent from repository -->
-     </parent>
-     <groupId>com.gocashless</groupId>
-     <artifactId>qr-code-generation-service</artifactId>
-     <version>0.0.1-SNAPSHOT</version>
-     <name>qr-code-generation-service</name>
-     <description>Gocashless QR Code Generation Service</description>
-
-     <properties>
-         <java.version>17</java.version>
-         <spring-cloud.version>2023.0.2</spring-cloud.version> <!-- Ensure compatibility with Spring Boot 3.3.1 -->
-     </properties>
-
-     <dependencies>

```xml
<!-- Spring Boot Starter Web for REST APIs -->
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<!-- Lombok (Optional, for boilerplate reduction) -->
<dependency>
   <groupId>org.projectlombok</groupId>
   <artifactId>lombok</artifactId>
   <optional>true</optional>
</dependency>
<!-- Spring Boot DevTools (Optional, for faster development) -->
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-devtools</artifactId>
   <scope>runtime</scope>
   <optional>true</optional>
</dependency>
<!-- Spring Boot Starter Test -->
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-test</artifactId>
   <scope>test</scope>
</dependency>

<!-- Spring Cloud Starter Netflix Eureka Client for Service Discovery -->
```

```xml
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>

        <!-- ZXing Core for QR code generation -->
        <dependency>
            <groupId>com.google.zxing</groupId>
            <artifactId>core</artifactId>
            <version>3.5.3</version>
        </dependency>
        <!-- ZXing Java SE extensions for image processing -->
        <dependency>
            <groupId>com.google.zxing</groupId>
            <artifactId>javase</artifactId>
            <version>3.5.3</version>
        </dependency>
        <!-- Jackson for JSON processing (already included with web, but explicitly for clarity) -->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
        </dependency>
    </dependencies>

    <dependencyManagement>
```

```
o           <dependencies>
o               <dependency>
o                   <groupId>org.springframework.cloud</groupId>
o                   <artifactId>spring-cloud-dependencies</artifactId>
o                   <version>${spring-cloud.version}</version>
o                   <type>pom</type>
o                   <scope>import</scope>
o               </dependency>
o           </dependencies>
o       </dependencyManagement>
o
o       <build>
o           <plugins>
o               <plugin>
o                   <groupId>org.springframework.boot</groupId>
o                   <artifactId>spring-boot-maven-plugin</artifactId>
o                   <configuration>
o                       <excludes>
o                           <exclude>
o                               <groupId>org.projectlombok</groupId>
o                               <artifactId>lombok</artifactId>
o                           </exclude>
o                       </excludes>
o                   </configuration>
o               </plugin>
o           </plugins>
```

```
o     </build>
o   </project>
```

- **application.yml for Eureka Configuration:**

```
o   server:
o     port: 8082 # Unique port for QRGS, e.g., 8082
o
o   spring:
o     application:
o       name: qr-code-generation-service # Name for Eureka registration
o
o   eureka:
o     client:
o       serviceUrl:
o         defaultZone: http://localhost:8761/eureka # URL of your Eureka Server
o       fetch-registry: true
o       register-with-eureka: true
o     instance:
o       hostname: localhost
o
o   # QR Code Configuration (optional, for default size/color)
o   qrcode:
o     width: 300
o     height: 300
o     format: PNG
o     charset: UTF-8
```

- **Package Structure:**
- com.gocashless.qrgs/
- ├── QrgsApplication.java   // Main Spring Boot application class
- ├── config/
- │   └── AppConfig.java   // Any general application configurations
- ├── model/   // DTOs for request/response payloads
- │   ├── QrGenerateRequest.java
- │   └── QrGenerateResponse.java
- ├── service/   // Business Logic
- │   ├── QrCodeService.java
- │   └── EncryptionService.java // For encrypting/signing QR payload
- ├── controller/   // REST API Endpoints
- │   └── QrCodeController.java
- └── util/   // Utility classes
-     └── QrCodeGenerator.java   // Wrapper for ZXing

- **Example Classes:**
  - **QrgsApplication.java (Main Application Class):**
  - package com.gocashless.qrgs;
  -
  - import org.springframework.boot.SpringApplication;
  - import org.springframework.boot.autoconfigure.SpringBootApplication;
  - import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
  -

- @SpringBootApplication
- @EnableDiscoveryClient // Enable this application as a Eureka client
- public class QrgsApplication {
-
-     public static void main(String[] args) {
-         SpringApplication.run(QrgsApplication.class, args);
-     }
- }

- **model/QrGenerateRequest.java (DTO for incoming request):**
- package com.gocashless.qrgs.model;
-
- import lombok.Data;
- import java.math.BigDecimal;
- import java.util.UUID;
-
- @Data
- public class QrGenerateRequest {
-     private UUID conductorId;
-     private UUID routeId;
-     private UUID originStopId;
-     private UUID destinationStopId;
-     // Add any other data needed for fare lookup or QR payload
- }

- **model/QrGenerateResponse.java (DTO for outgoing response):**

- package com.gocashless.qrgs.model;
-
- import lombok.Data;
- import java.util.UUID;
-
- @Data
- public class QrGenerateResponse {
-     private String qrCodeImageBase64; // Base64 encoded PNG image
-     private String transactionRef;   // Unique transaction ID generated for this QR
-     private String message;   // Optional message
- }

- **util/QrCodeGenerator.java (Utility for ZXing integration):**
- package com.gocashless.qrgs.util;
-
- import com.google.zxing.BarcodeFormat;
- import com.google.zxing.EncodeHintType;
- import com.google.zxing.WriterException;
- import com.google.zxing.client.j2se.MatrixToImageWriter;
- import com.google.zxing.common.BitMatrix;
- import com.google.zxing.qrcode.QRCodeWriter;
- import com.google.zxing.qrcode.decoder.ErrorCorrectionLevel;
- import org.springframework.beans.factory.annotation.Value;
- import org.springframework.stereotype.Component;
-

```
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.Base64;
import java.util.HashMap;
import java.util.Map;

@Component
public class QrCodeGenerator {

    @Value("${qrcode.width:300}")
    private int qrCodeWidth;

    @Value("${qrcode.height:300}")
    private int qrCodeHeight;

    @Value("${qrcode.format:PNG}")
    private String qrCodeImageFormat; // e.g., PNG, JPG

    @Value("${qrcode.charset:UTF-8}")
    private String qrCodeCharset;

    /**
     * Generates a QR code image as a Base64 encoded string.
     *
     * @param text The data to encode in the QR code.
     * @return Base64 encoded string of the QR code image (PNG format).
     * @throws WriterException If an error occurs during QR code generation.
     * @throws IOException If an I/O error occurs during image writing.
     */
    public String generateQrCodeImageBase64(String text) throws WriterException, IOException {
        Map<EncodeHintType, Object> hints = new HashMap<>();
        hints.put(EncodeHintType.CHARACTER_SET, qrCodeCharset);
        hints.put(EncodeHintType.ERROR_CORRECTION, ErrorCorrectionLevel.H); // High error correction
        hints.put(EncodeHintType.MARGIN, 1); // Less margin

        QRCodeWriter qrCodeWriter = new QRCodeWriter();
        BitMatrix bitMatrix = qrCodeWriter.encode(text, BarcodeFormat.QR_CODE, qrCodeWidth, qrCodeHeight, hints);

        ByteArrayOutputStream pngOutputStream = new ByteArrayOutputStream();
        MatrixToImageWriter.writeToStream(bitMatrix, qrCodeImageFormat, pngOutputStream);
        byte[] pngData = pngOutputStream.toByteArray();

        return Base64.getEncoder().encodeToString(pngData);
    }
}
```

- **service/EncryptionService.java (Service for payload encryption/signing):**
- package com.gocashless.qrgs.service;
-
- import com.fasterxml.jackson.databind.ObjectMapper;
- import org.springframework.beans.factory.annotation.Value;
- import org.springframework.stereotype.Service;
-
- import javax.crypto.Cipher;
- import javax.crypto.KeyGenerator;
- import javax.crypto.SecretKey;
- import javax.crypto.spec.SecretKeySpec;
- import java.nio.charset.StandardCharsets;
- import java.security.MessageDigest;
- import java.util.Base64;
- import java.util.UUID;
-
- /**
- * Service for encrypting and decrypting QR code payloads.
- * Uses AES for symmetric encryption and HMAC for integrity (simplified for example).
- * In a real system, you might use more robust key management and signing.
- */
- @Service
- public class EncryptionService {
-

- // This key should be securely managed (e.g., from environment variables, vault)
- // For demonstration, a simple hardcoded key. NEVER do this in production.
- @Value("${qr.encryption.secret-key:thisisasecretkeyforqrencryptionthatisatleast16byteslong}")
- private String secretKeyString;
-
- private SecretKey secretKey;
- private final ObjectMapper objectMapper = new ObjectMapper();
-
- // Initialize the secret key from the provided string
- private SecretKey getSecretKey() throws Exception {
-     if (secretKey == null) {
-         // Use SHA-256 hash of the secret string to get a 256-bit key for AES
-         byte[] keyBytes = MessageDigest.getInstance("SHA-256").digest(secretKeyString.getBytes(StandardCharsets.UTF_8));
-         secretKey = new SecretKeySpec(keyBytes, "AES");
-     }
-     return secretKey;
- }
-
- /**
- * Encrypts a JSON payload for the QR code.
- *
- * @param payload The object to encrypt (will be converted to JSON string).

```
     * @return Base64 encoded encrypted string.
     * @throws Exception If encryption fails.
     */
    public String encryptPayload(Object payload) throws Exception {
        String jsonPayload = objectMapper.writeValueAsString(payload);
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding"); // ECB is simple, consider CBC/GCM with IV for production
        cipher.init(Cipher.ENCRYPT_MODE, getSecretKey());
        byte[] encryptedBytes = cipher.doFinal(jsonPayload.getBytes(StandardCharsets.UTF_8));
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }

    /**
     * Decrypts a Base64 encoded encrypted string to a JSON payload.
     *
     * @param encryptedPayload Base64 encoded encrypted string.
     * @param valueType Class type to deserialize the JSON into.
     * @return Decrypted object.
     * @throws Exception If decryption fails.
     */
    public <T> T decryptPayload(String encryptedPayload, Class<T> valueType) throws Exception {
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.DECRYPT_MODE, getSecretKey());
        byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedPayload));
```

- String jsonPayload = new String(decryptedBytes, StandardCharsets.UTF_8);
- return objectMapper.readValue(jsonPayload, valueType);
- }
- }

- **service/QrCodeService.java (Main Service for QR generation logic):**
- package com.gocashless.qrgs.service;
-
- import com.gocashless.qrgs.model.QrGenerateRequest;
- import com.gocashless.qrgs.model.QrGenerateResponse;
- import com.gocashless.qrgs.util.QrCodeGenerator;
- import com.fasterxml.jackson.databind.JsonNode;
- import com.fasterxml.jackson.databind.ObjectMapper;
- import com.fasterxml.jackson.databind.node.ObjectNode;
- import org.springframework.beans.factory.annotation.Autowired;
- import org.springframework.stereotype.Service;
- import org.springframework.web.client.RestTemplate; // For inter-service communication
- import org.slf4j.Logger;
- import org.slf4j.LoggerFactory;
-
- import java.math.BigDecimal;
- import java.time.Instant;
- import java.util.UUID;
-
- @Service

```
public class QrCodeService {

    private static final Logger logger = LoggerFactory.getLogger(QrCodeService.class);

    private final QrCodeGenerator qrCodeGenerator;
    private final EncryptionService encryptionService;
    private final RestTemplate restTemplate; // For calling RFMS (Route & Fare Management Service)
    private final ObjectMapper objectMapper; // For building JSON payload

    @Autowired
    public QrCodeService(QrCodeGenerator qrCodeGenerator, EncryptionService encryptionService, RestTemplate restTemplate, ObjectMapper objectMapper) {
        this.qrCodeGenerator = qrCodeGenerator;
        this.encryptionService = encryptionService;
        this.restTemplate = restTemplate;
        this.objectMapper = objectMapper;
    }

    /**
     * Generates a QR code containing encrypted payment details.
     *
     * @param request The request containing conductor, route, and stop IDs.
     * @return QrGenerateResponse containing the Base64 encoded QR image and transaction reference.
```

- * @throws Exception If any error occurs during fare lookup, encryption, or QR generation.
- */
- public QrGenerateResponse generatePaymentQrCode(QrGenerateRequest request) throws Exception {
- // 1. Validate conductor (Optional - can be done by UMS or during authentication)
- // For now, assume conductorId is valid.
-
- // 2. Query RFMS to get the fare amount
- // Using Eureka service name for lookup
- String rfmsUrl = "http://ROUTE-FARE-MANAGEMENT-SERVICE/api/v1/fares/lookup";
- // Build query parameters for RestTemplate
- String fareLookupUrl = String.format("%s?routeId=%s&originStopId=%s&destinationStopId=%s",
- rfmsUrl, request.getRouteId(), request.getOriginStopId(), request.getDestinationStopId());
-
- logger.info("Calling RFMS for fare lookup: {}", fareLookupUrl);
- JsonNode fareResponse = restTemplate.getForObject(fareLookupUrl, JsonNode.class);
-
- if (fareResponse == null || fareResponse.get("amount") == null) {
- logger.error("Fare not found for the given route and stops. RFMS Response: {}", fareResponse);
- throw new IllegalArgumentException("Fare information could not be retrieved for the specified journey.");

- }
-
- BigDecimal fareAmount = new BigDecimal(fareResponse.get("amount").asText());
- String currency = fareResponse.get("currency").asText();
-
- // 3. Generate a unique transaction reference
- String transactionRef = UUID.randomUUID().toString();
-
- // 4. Construct the QR Payload (as a simple JSON object)
- ObjectNode qrPayload = objectMapper.createObjectNode();
- qrPayload.put("conductorId", request.getConductorId().toString());
- qrPayload.put("fareAmount", fareAmount);
- qrPayload.put("currency", currency);
- qrPayload.put("transactionRef", transactionRef);
- qrPayload.put("timestamp", Instant.now().toEpochMilli());
- // In a real system, you'd add a digital signature here for integrity verification
- // qrPayload.put("signature", generateSignature(qrPayloadJsonString));
-
- // 5. Encrypt the payload
- String encryptedPayload = encryptionService.encryptPayload(qrPayload);
- logger.info("Encrypted QR Payload: {}", encryptedPayload);
-
- // 6. Generate the QR code image from the encrypted payload

- String qrCodeImageBase64 = qrCodeGenerator.generateQrCodeImageBase64(encryptedPayload);
-
- QrGenerateResponse response = new QrGenerateResponse();
- response.setQrCodeImageBase64(qrCodeImageBase64);
- response.setTransactionRef(transactionRef);
- response.setMessage("QR Code generated successfully.");
- return response;
- }
- }

- **controller/QrCodeController.java (REST Controller):**
- package com.gocashless.qrgs.controller;
-
- import com.gocashless.qrgs.model.QrGenerateRequest;
- import com.gocashless.qrgs.model.QrGenerateResponse;
- import com.gocashless.qrgs.service.QrCodeService;
- import org.springframework.beans.factory.annotation.Autowired;
- import org.springframework.http.HttpStatus;
- import org.springframework.http.ResponseEntity;
- import org.springframework.web.bind.annotation.*;
- import org.slf4j.Logger;
- import org.slf4j.LoggerFactory;
-
- /**
- * REST Controller for QR Code Generation Service.

```
 * Exposes endpoints for generating payment QR codes.
 */
@RestController
@RequestMapping("/api/v1/qr")
public class QrCodeController {

    private static final Logger logger = LoggerFactory.getLogger(QrCodeController.class);

    private final QrCodeService qrCodeService;

    @Autowired
    public QrCodeController(QrCodeService qrCodeService) {
        this.qrCodeService = qrCodeService;
    }

    /**
     * Endpoint to generate a payment QR code.
     * This would be called by the Conductor App.
     *
     * @param request The request containing details for QR code generation (e.g., conductor, route, stops).
     * @return ResponseEntity with the Base64 encoded QR image and transaction reference.
     */
    @PostMapping("/generate")
    public ResponseEntity<?> generateQrCode(@RequestBody QrGenerateRequest request) {
        logger.info("Received QR code generation request: {}", request);
        try {
            QrGenerateResponse response = qrCodeService.generatePaymentQrCode(request);
            return ResponseEntity.ok(response);
        } catch (IllegalArgumentException e) {
            logger.error("Invalid QR generation request: {}", e.getMessage());
            return ResponseEntity.badRequest().body(e.getMessage());
        } catch (Exception e) {
            logger.error("Error generating QR code: {}", e.getMessage(), e);
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Failed to generate QR code: " + e.getMessage());
        }
    }
}
```

- **config/AppConfig.java (Configuration for RestTemplate):**

```
package com.gocashless.qrgs.config;

import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

```

```
@Configuration
public class AppConfig {

  /**
   * Configures a LoadBalanced RestTemplate for inter-service communication.
   * This allows using Eureka service names (e.g., "ROUTE-FARE-MANAGEMENT-SERVICE")
   * instead of direct host:port when making HTTP calls to other microservices.
   */
  @Bean
  @LoadBalanced
  public RestTemplate restTemplate() {
    return new RestTemplate();
  }
}
```

- **Key Entities/Data Models:** (Primarily processes data; minimal persistence needed for QR codes themselves, as they are ephemeral)
  - **QR Payload Structure:** (Internal DTO)
    - conductorId (UUID)
    - fareAmount (BigDecimal)
    - currency (String)
    - transactionRef (String, UUID generated by QRGS or PPS)
    - timestamp (Long)
    - signature (String, for integrity verification)

- **Core Functionality/APIs:**
  - POST /api/v1/qr/generate:
    - **Request Body:** {"conductorId": "...", "routeId": "...", "originStopId": "...", "destinationStopId": "..."}
    - **Process:**
      1. Validate conductorId with UMS (optional, for real-time check).
      2. Query RFMS to get the fareAmount for the given route/stops.
      3. Generate a unique transactionRef (UUID).
      4. Construct the QR Payload Structure.
      5. Encrypt and sign the payload (e.g., using AES for encryption and HMAC for signing, or RSA as discussed for PIN, but for QR data, a symmetric approach might be more efficient).
      6. Generate the QR code image (Base64 encoded string or direct image stream).
    - **Response:** {"qrCodeImage": "base64encodedImageString", "transactionRef": "..."}
- **Interactions:**
  - **Conductor App:** Calls QRGS to get the QR code image to display.
  - **Payment Processing Service (PPS):** The Passenger App will send the *parsed* QR payload to PPS for payment initiation. QRGS doesn't directly interact with PPS, but its output is consumed by PPS.
  - **User Management Service (UMS):** (Optional) To validate conductor ID.
  - **Route & Fare Management Service (RFMS):** To retrieve fare details.

4. Payment Processing Service (PPS)

- **Purpose:** Orchestrates the mobile money payment flow, interacting with the Airtel MoMo API, handling payment initiation, callbacks, and status updates.
- **Spring Boot Structure (with PostgreSQL and Eureka Client):**

To implement the PPS with Spring Boot, PostgreSQL, and integrate with Eureka, you would typically set up the following:

- **Maven Dependencies (pom.xml):**
- <?xml version="1.0" encoding="UTF-8"?>
- <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
-          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
-     <modelVersion>4.0.0</modelVersion>
-     <parent>
-         <groupId>org.springframework.boot</groupId>
-         <artifactId>spring-boot-starter-parent</artifactId>
-         <version>3.3.1</version> <!-- Use the same Spring Boot version as other services -->
-         <relativePath/> <!-- lookup parent from repository -->
-     </parent>
-     <groupId>com.gocashless</groupId>
-     <artifactId>payment-processing-service</artifactId>
-     <version>0.0.1-SNAPSHOT</version>
-     <name>payment-processing-service</name>
-     <description>Gocashless Payment Processing Service</description>
-
-     <properties>
-         <java.version>17</java.version>
-         <spring-cloud.version>2023.0.2</spring-cloud.version> <!-- Ensure compatibility with Spring Boot 3.3.1 -->
-     </properties>
-
-     <dependencies>

```
o   <!-- Spring Boot Starter Web for REST APIs -->
o   <dependency>
o     <groupId>org.springframework.boot</groupId>
o     <artifactId>spring-boot-starter-web</artifactId>
o   </dependency>
o   <!-- Spring Boot Starter Data JPA for ORM and database interaction -->
o   <dependency>
o     <groupId>org.springframework.boot</groupId>
o     <artifactId>spring-boot-starter-data-jpa</artifactId>
o   </dependency>
o   <!-- PostgreSQL Driver -->
o   <dependency>
o     <groupId>org.postgresql</groupId>
o     <artifactId>postgresql</artifactId>
o     <scope>runtime</scope>
o   </dependency>
o   <!-- Lombok (Optional, for boilerplate reduction) -->
o   <dependency>
o     <groupId>org.projectlombok</groupId>
o     <artifactId>lombok</artifactId>
o     <optional>true</optional>
o   </dependency>
o   <!-- Spring Boot DevTools (Optional, for faster development) -->
o   <dependency>
o     <groupId>org.springframework.boot</groupId>
o     <artifactId>spring-boot-devtools</artifactId>
```

- <scope>runtime</scope>
- <optional>true</optional>
- </dependency>
- <!-- Spring Boot Starter Test -->
- <dependency>
- <groupId>org.springframework.boot</groupId>
- <artifactId>spring-boot-starter-test</artifactId>
- <scope>test</scope>
- </dependency>
-
- <!-- Spring Cloud Starter Netflix Eureka Client for Service Discovery -->
- <dependency>
- <groupId>org.springframework.cloud</groupId>
- <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
- </dependency>
-
- <!-- Jackson for JSON processing (already included with web, but explicitly for clarity) -->
- <dependency>
- <groupId>com.fasterxml.jackson.core</groupId>
- <artifactId>jackson-databind</artifactId>
- </dependency>
-
- <!-- For RSA encryption (if not already in JRE/JDK) -->
- <!-- You might need to add a Bouncy Castle dependency if using advanced crypto features not in default JDK -->
- <!-- For example:

```
o        <dependency>
o            <groupId>org.bouncycastle</groupId>
o            <artifactId>bcprov-jdk15on</artifactId>
o            <version>1.70</version>
o        </dependency>
o        -->
o    </dependencies>
o
o    <dependencyManagement>
o        <dependencies>
o            <dependency>
o                <groupId>org.springframework.cloud</groupId>
o                <artifactId>spring-cloud-dependencies</artifactId>
o                <version>${spring-cloud.version}</version>
o                <type>pom</type>
o                <scope>import</scope>
o            </dependency>
o        </dependencies>
o    </dependencyManagement>
o
o    <build>
o        <plugins>
o            <plugin>
o                <groupId>org.springframework.boot</groupId>
o                <artifactId>spring-boot-maven-plugin</artifactId>
o                <configuration>
```

```
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

- **application.yml for Database and Eureka Configuration:**

```
server:
  port: 8083 # Unique port for PPS, e.g., 8083

spring:
  application:
    name: payment-processing-service # Name for Eureka registration
  datasource:
    url: jdbc:postgresql://localhost:5432/gocashless_pps_db # Dedicated DB for PPS (optional, can use THS DB)
    username: pps_user
    password: pps_password
    driver-class-name: org.postgresql.Driver
  jpa:
```

```yaml
    hibernate:
      ddl-auto: update # 'update' for development, 'none' or 'validate' for production
    show-sql: true
    properties:
      hibernate:
        dialect: org.hibernate.dialect.PostgreSQLDialect

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka # URL of your Eureka Server
    fetch-registry: true
    register-with-eureka: true
  instance:
    hostname: localhost

# Airtel Mobile Money API Configuration
airtel:
  api:
    client-id: YOUR_AIRTEL_CLIENT_ID
    client-secret: YOUR_AIRTEL_CLIENT_SECRET
    disbursement-pin: YOUR_AIRTEL_DISBURSEMENT_PIN # Encrypted PIN
    country: ZM # Zambia
    currency: ZMW # Zambian Kwacha
    environment-mode: stagging # or production
```

- # Base URL will be derived from environment-mode in AirtelApiClient

- **Package Structure:**
- com.gocashless.pps/
- ├── PpsApplication.java // Main Spring Boot application class
- ├── config/
- │ ├── AppConfig.java // General application configurations (e.g., RestTemplate)
- │ └── AirtelApiConfig.java // Configuration properties for Airtel API
- ├── airtel/ // Airtel API integration classes
- │ ├── client/
- │ │ ├── AirtelApiClient.java // Handles raw HTTP calls to Airtel
- │ │ └── PinEncryptor.java // RSA PIN encryption utility
- │ └── dto/ // DTOs for Airtel API requests/responses
- │ ├── AirtelTokenResponse.java
- │ ├── AirtelPaymentRequest.java
- │ ├── AirtelPaymentResponse.java
- │ ├── AirtelDisbursementRequest.java
- │ └── AirtelDisbursementResponse.java
- ├── model/ // JPA Entities (if PPS stores its own data, e.g., webhooks)
- │ └── PaymentCallback.java // Example: to store raw Airtel callbacks
- ├── repository/ // Spring Data JPA Repositories
- │ └── PaymentCallbackRepository.java
- ├── service/ // Business Logic
- │ ├── PaymentProcessingService.java // Orchestrates payment flow

- | └── TransactionHistoryProxy.java // Client for THS
- | └── NotificationServiceProxy.java // Client for NOS
- ├── controller/ // REST API Endpoints
- | ├── PaymentController.java
- | └── AirtelCallbackController.java // Endpoint for Airtel webhooks
- └── dto/ // Data Transfer Objects (for internal requests/responses)
- ├── InitiatePaymentRequest.java
- └── InitiatePaymentResponse.java
- └── PaymentStatusResponse.java

- **Example Classes:**
  - **PpsApplication.java (Main Application Class):**
  - package com.gocashless.pps;
  -
  - import org.springframework.boot.SpringApplication;
  - import org.springframework.boot.autoconfigure.SpringBootApplication;
  - import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
  -
  - @SpringBootApplication
  - @EnableDiscoveryClient // Enable this application as a Eureka client
  - public class PpsApplication {
  -
  -     public static void main(String[] args) {
  -         SpringApplication.run(PpsApplication.class, args);
  -     }

```
}
```

- **config/AirtelApiConfig.java (Configuration Properties):**

```java
package com.gocashless.pps.config;

import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@Component
@ConfigurationProperties(prefix = "airtel.api")
@Data
public class AirtelApiConfig {
    private String clientId;
    private String clientSecret;
    private String disbursementPin;
    private String country;
    private String currency;
    private String environmentMode;

    // Derived property for base URL
    public String getBaseUrl() {
        return "production".equalsIgnoreCase(environmentMode) ?
                "https://openapi.airtel.africa" : "https://openapiuat.airtel.africa";
    }
```

- }

- **airtel/client/PinEncryptor.java (Utility for PIN encryption):**
- package com.gocashless.pps.airtel.client;
-
- import java.nio.charset.StandardCharsets;
- import java.security.KeyFactory;
- import java.security.PublicKey;
- import java.security.spec.X509EncodedKeySpec;
- import java.util.Base64;
- import javax.crypto.Cipher;
-
- /**
- * Utility class for encrypting the Airtel Money PIN using RSA PKCS1_v1_5 padding.
- * This corresponds to the 'Pin' class in the Python code.
- */
- public class PinEncryptor {
-
-     // Public key provided by Airtel for PIN encryption (Base64 encoded)
-     private static final String PUBLIC_KEY_BASE64 = "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCkq3XbDI1s8Lu7SpUBP+bqOs/MC6PKWz6n/0UkqTiOZqKqaoZClI3BUDTrSIJsrN1Qx7ivBzsaAYfsB0CygSSWay4iyUcnMVEDrNVOJwtWvHxpyWJC5RfKBrweW9b8klFa/CfKRtkK730apy0Kxjg+7fF0tB4O3Ic9Gxuv4pFkbQIDAQAB";
-
-     /**

```
▪      * Encrypts the provided PIN using RSA with PKCS1_v1_5 padding.
▪      *
▪      * @param pin The raw PIN string to encrypt.
▪      * @return The Base64 encoded encrypted PIN.
▪      * @throws Exception If an error occurs during key decoding or encryption.
▪      */
▪     public static String genPin(String pin) throws Exception {
▪         // Decode the Base64 public key
▪         byte[] keyBytes = Base64.getDecoder().decode(PUBLIC_KEY_BASE64);
▪
▪         // Create X509EncodedKeySpec for the public key
▪         X509EncodedKeySpec keySpec = new X509EncodedKeySpec(keyBytes);
▪         KeyFactory keyFactory = KeyFactory.getInstance("RSA");
▪         PublicKey publicKey = keyFactory.generatePublic(keySpec);
▪
▪         // Initialize Cipher for RSA encryption with PKCS1_v1_5 padding
▪         Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding"); // PKCS1Padding corresponds to PKCS1_v1_5
▪         cipher.init(Cipher.ENCRYPT_MODE, publicKey);
▪
▪         // Encrypt the PIN
▪         byte[] encryptedBytes = cipher.doFinal(pin.getBytes(StandardCharsets.UTF_8));
▪
```

```
        // Base64 encode the encrypted bytes
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }
}
```

- **airtel/client/AirtelApiClient.java (Airtel API Client):**

```
package com.gocashless.pps.airtel.client;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
import com.gocashless.pps.config.AirtelApiConfig;
import com.gocashless.pps.airtel.dto.*; // Import all DTOs for Airtel API
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Component;
import org.springframework.web.reactive.function.BodyInserters;
import org.springframework.web.reactive.function.client.WebClient;

import java.util.UUID;
import java.util.concurrent.ConcurrentHashMap;

/**
```

```
 * Client for making authenticated calls to the Airtel Mobile Money API.
 * Handles token generation, Collections (Request to Pay), and Disbursements.
 */
@Component
public class AirtelApiClient {

    private static final Logger logger = LoggerFactory.getLogger(AirtelApiClient.class);
    private final AirtelApiConfig airtelApiConfig;
    private final WebClient webClient;
    private final ObjectMapper objectMapper;

    // Cache for access token to avoid re-generating for every request
    private final ConcurrentHashMap<String, String> accessTokenCache = new ConcurrentHashMap<>();
    private long tokenExpiryTime = 0; // Epoch milliseconds

    @Autowired
    public AirtelApiClient(AirtelApiConfig airtelApiConfig, WebClient.Builder webClientBuilder, ObjectMapper objectMapper) {
        this.airtelApiConfig = airtelApiConfig;
        this.objectMapper = objectMapper;
        this.webClient = webClientBuilder.baseUrl(airtelApiConfig.getBaseUrl()).build();
    }

```

```
  /**
   * Generates an OAuth2 bearer token from Airtel Money API.
   * Caches the token and re-generates only when expired.
   *
   * @return The access token string.
   * @throws RuntimeException if token generation fails.
   */
  public String getAccessToken() {
      if (accessTokenCache.containsKey("airtel_token") && System.currentTimeMillis() < tokenExpiryTime) {
          return accessTokenCache.get("airtel_token");
      }

      logger.info("Generating new Airtel API access token...");
      String url = "/auth/oauth2/token";

      ObjectNode payload = objectMapper.createObjectNode();
      payload.put("client_id", airtelApiConfig.getClientId());
      payload.put("client_secret", airtelApiConfig.getClientSecret());
      payload.put("grant_type", "client_credentials");

      try {
          AirtelTokenResponse response = webClient.post()
                  .uri(url)
                  .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
                  .body(BodyInserters.fromValue(payload.toString()))
```

```
                    .retrieve()
                    .bodyToMono(AirtelTokenResponse.class)
                    .block(); // Blocking call for simplicity, consider reactive in a full WebFlux app

            if (response != null && response.getAccessToken() != null) {
                accessTokenCache.put("airtel_token", response.getAccessToken());
                // Set expiry time a bit before actual expiry for safety (e.g., 5 minutes before)
                tokenExpiryTime = System.currentTimeMillis() + (response.getExpiresIn() * 1000) - (5 * 60 * 1000);
                logger.info("Airtel API access token generated successfully. Expires in {} seconds.", response.getExpiresIn());
                return response.getAccessToken();
            } else {
                throw new RuntimeException("Failed to get access token: Response was null or missing token.");
            }
        } catch (Exception e) {
            logger.error("Error generating Airtel API access token: {}", e.getMessage(), e);
            throw new RuntimeException("Failed to generate Airtel API access token", e);
        }
    }

    /**
```

```
     * Initiates a merchant payment request (Collections - USSD Push) to a subscriber.
     * Corresponds to POST /merchant/v2/payments/
     *
     * @param request The payment request details.
     * @return AirtelPaymentResponse containing the API response.
     * @throws RuntimeException if the API call fails.
     */
    public AirtelPaymentResponse initiateCollectionPayment(AirtelPaymentRequest request) {
        String url = "/merchant/v2/payments/"; // Updated to v2 as per 1.txt

        try {
            String accessToken = getAccessToken();
            logger.info("Initiating Airtel Collection Payment to MSISDN: {}", request.getSubscriber().getMsisdn());

            // Note: x-signature and x-key are mentioned as required in 1.txt but not provided
            // in the original Python code. Implement message signing if Airtel requires it.
            // For now, they are omitted or can be added as empty strings if the API tolerates it.
            // Example: .header("x-signature", "")
            // Example: .header("x-key", "")

            return webClient.post()
                    .uri(url)
```

- .header(HttpHeaders.AUTHORIZATION, "Bearer " + accessToken)
- .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
- .header(HttpHeaders.ACCEPT, MediaType.APPLICATION_JSON_VALUE)
- .header("X-Country", airtelApiConfig.getCountry())
- .header("X-Currency", airtelApiConfig.getCurrency())
- .body(BodyInserters.fromValue(objectMapper.writeValueAsString(request)))
- .retrieve()
- .bodyToMono(AirtelPaymentResponse.class)
- .block();
- } catch (Exception e) {
- logger.error("Error initiating Airtel Collection Payment: {}", e.getMessage(), e);
- throw new RuntimeException("Failed to initiate Airtel Collection Payment", e);
- }
- }
-
- /**
- * Initiates a disbursement (transfer money) from the merchant account to a subscriber.
- * Corresponds to POST /standard/v3/disbursements
- *
- * @param request The disbursement request details.

```
     * @return AirtelDisbursementResponse containing the API response.
     * @throws RuntimeException if the API call fails.
     */
    public AirtelDisbursementResponse initiateDisbursement(AirtelDisbursementRequest request) {
        String url = "/standard/v3/disbursements"; // Updated to v3 as per 1.txt

        try {
            String accessToken = getAccessToken();
            logger.info("Initiating Airtel Disbursement to MSISDN: {}", request.getPayee().getMsisdn());

            // Encrypt the PIN
            String encryptedPin = PinEncryptor.genPin(airtelApiConfig.getDisbursementPin());
            request.setPin(encryptedPin); // Set the encrypted PIN in the request DTO

            // Note: x-signature and x-key are optional for disbursements in 1.txt.

            return webClient.post()
                    .uri(url)
                    .header(HttpHeaders.AUTHORIZATION, "Bearer " + accessToken)
                    .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
```

- .header(HttpHeaders.ACCEPT, MediaType.APPLICATION_JSON_VALUE)
- .header("X-Country", airtelApiConfig.getCountry())
- .header("X-Currency", airtelApiConfig.getCurrency())
- .body(BodyInserters.fromValue(objectMapper.writeValueAsString(request)))
- .retrieve()
- .bodyToMono(AirtelDisbursementResponse.class)
- .block();
- } catch (Exception e) {
- logger.error("Error initiating Airtel Disbursement: {}", e.getMessage(), e);
- throw new RuntimeException("Failed to initiate Airtel Disbursement", e);
- }
- }
-
- /**
- * Verifies the status of a previously initiated collection transaction.
- * Corresponds to GET /standard/v1/payments/{transactionId}
- * Note: The provided 1.txt only shows a Payments API for USSD Push, and a Disbursement API.
- * The verify transaction endpoint in the original Python was `/standard/v1/payments/{txn}`.
- * This might be a generic payment status check.
- *
- * @param transactionId The transaction ID to verify.

```
▪      * @return JsonNode containing the transaction verification details.
▪      * @throws RuntimeException if the API call fails.
▪      */
▪     public JsonNode verifyCollectionTransaction(String transactionId) {
▪         String url = "/standard/v1/payments/" + transactionId; // Assuming this endpoint is correct for verification
▪
▪         try {
▪             String accessToken = getAccessToken();
▪             logger.info("Verifying Airtel Collection Transaction ID: {}", transactionId);
▪
▪             return webClient.get()
▪                     .uri(url)
▪                     .header(HttpHeaders.AUTHORIZATION, "Bearer " + accessToken)
▪                     .header(HttpHeaders.ACCEPT, MediaType.APPLICATION_JSON_VALUE)
▪                     .header("X-Country", airtelApiConfig.getCountry())
▪                     .header("X-Currency", airtelApiConfig.getCurrency())
▪                     .retrieve()
▪                     .bodyToMono(JsonNode.class)
▪                     .block();
▪         } catch (Exception e) {
▪             logger.error("Error verifying Airtel Collection Transaction: {}", e.getMessage(), e);
▪             throw new RuntimeException("Failed to verify Airtel Collection Transaction", e);
```

```
        }
    }

    /**
     * Checks the balance of the Airtel Money account associated with the API credentials.
     * Corresponds to GET /standard/v1/users/balance
     *
     * @return JsonNode containing the account balance details.
     * @throws RuntimeException if the API call fails.
     */
    public JsonNode getAirtelBalance() {
        String url = "/standard/v1/users/balance";

        try {
            String accessToken = getAccessToken();
            logger.info("Fetching Airtel account balance.");

            return webClient.get()
                    .uri(url)
                    .header(HttpHeaders.AUTHORIZATION, "Bearer " + accessToken)
                    .header(HttpHeaders.ACCEPT, MediaType.APPLICATION_JSON_VALUE)
                    .header("X-Country", airtelApiConfig.getCountry())
                    .header("X-Currency", airtelApiConfig.getCurrency())
                    .retrieve()
```

- .bodyToMono(JsonNode.class)
- .block();
- } catch (Exception e) {
- logger.error("Error fetching Airtel account balance: {}", e.getMessage(), e);
- throw new RuntimeException("Failed to fetch Airtel account balance", e);
- }
- }
- }

- **airtel/dto/AirtelTokenResponse.java (DTO for Token API response):**
- package com.gocashless.pps.airtel.dto;
-
- import com.fasterxml.jackson.annotation.JsonProperty;
- import lombok.Data;
- import lombok.NoArgsConstructor;
- import lombok.AllArgsConstructor;
-
- @Data
- @NoArgsConstructor
- @AllArgsConstructor
- public class AirtelTokenResponse {
- @JsonProperty("access_token")
- private String accessToken;
- @JsonProperty("token_type")
- private String tokenType;

-     @JsonProperty("expires_in")
-     private long expiresIn; // in seconds
- }

- **airtel/dto/AirtelPaymentRequest.java (DTO for Collections API request - /merchant/v2/payments/):**
- package com.gocashless.pps.airtel.dto;
-
- import com.fasterxml.jackson.annotation.JsonProperty;
- import lombok.Data;
- import lombok.NoArgsConstructor;
- import lombok.AllArgsConstructor;
- import java.math.BigDecimal;
-
- @Data
- @NoArgsConstructor
- @AllArgsConstructor
- public class AirtelPaymentRequest {
-     private String reference;
-     private Subscriber subscriber;
-     private Transaction transaction;
-
-     @Data
-     @NoArgsConstructor
-     @AllArgsConstructor
-     public static class Subscriber {

-         private String country;
-         private String currency; // Optional as per 1.txt, but included for completeness
-         private String msisdn;
-     }
-
-     @Data
-     @NoArgsConstructor
-     @AllArgsConstructor
-     public static class Transaction {
-         private BigDecimal amount;
-         private String country; // Optional as per 1.txt, but included for completeness
-         private String currency; // Optional as per 1.txt, but included for completeness
-         private String id; // Partner unique transaction id
-     }
- }

- **airtel/dto/AirtelPaymentResponse.java (DTO for Collections API response):**
- package com.gocashless.pps.airtel.dto;
-
- import lombok.Data;
- import lombok.NoArgsConstructor;
- import lombok.AllArgsConstructor;
- import com.fasterxml.jackson.annotation.JsonProperty;

```
▪
▪ @Data
▪ @NoArgsConstructor
▪ @AllArgsConstructor
▪ public class AirtelPaymentResponse {
▪     private Data data;
▪     private Status status;
▪
▪     @Data
▪     @NoArgsConstructor
▪     @AllArgsConstructor
▪     public static class Data {
▪         private Transaction transaction;
▪     }
▪
▪     @Data
▪     @NoArgsConstructor
▪     @AllArgsConstructor
▪     public static class Transaction {
▪         private String id;
▪         private String status;
▪     }
▪
▪     @Data
▪     @NoArgsConstructor
▪     @AllArgsConstructor
```

-
  ```
      public static class Status {
  ```

-
  ```
          private String code;
  ```

-
  ```
          private String message;
  ```

-
  ```
          @JsonProperty("result_code")
  ```

-
  ```
          private String resultCode; // Deprecated, but included
  ```

-
  ```
          @JsonProperty("response_code")
  ```

-
  ```
          private String responseCode;
  ```

-
  ```
          private Boolean success;
  ```

-
  ```
      }
  ```

-
  ```
  }
  ```

- **airtel/dto/AirtelDisbursementRequest.java (DTO for Disbursements API request - /standard/v3/disbursements):**

-
  ```
  package com.gocashless.pps.airtel.dto;
  ```
-

-
  ```
  import com.fasterxml.jackson.annotation.JsonProperty;
  ```

-
  ```
  import lombok.Data;
  ```

-
  ```
  import lombok.NoArgsConstructor;
  ```

-
  ```
  import lombok.AllArgsConstructor;
  ```

-
  ```
  import java.math.BigDecimal;
  ```
-

-
  ```
  @Data
  ```

-
  ```
  @NoArgsConstructor
  ```

-
  ```
  @AllArgsConstructor
  ```

-
  ```
  public class AirtelDisbursementRequest {
  ```

-
  ```
      private Payee payee;
  ```

```
    private String reference;
    private String pin; // Encrypted PIN
    private Transaction transaction;

    @Data
    @NoArgsConstructor
    @AllArgsConstructor
    public static class Payee {
        private String msisdn;
        @JsonProperty("wallet_type")
        private String walletType; // e.g., "SALARY" or "NORMAL"
    }

    @Data
    @NoArgsConstructor
    @AllArgsConstructor
    public static class Transaction {
        private BigDecimal amount;
        private String id; // Payer unique transaction id
        private String type; // e.g., "B2C" or "B2B"
    }
}
```

- **airtel/dto/AirtelDisbursementResponse.java (DTO for Disbursements API response):**

```
package com.gocashless.pps.airtel.dto;
```

```
▪
▪ import com.fasterxml.jackson.annotation.JsonProperty;
▪ import lombok.Data;
▪ import lombok.NoArgsConstructor;
▪ import lombok.AllArgsConstructor;
▪
▪ @Data
▪ @NoArgsConstructor
▪ @AllArgsConstructor
▪ public class AirtelDisbursementResponse {
▪     private Data data;
▪     private Status status;
▪
▪     @Data
▪     @NoArgsConstructor
▪     @AllArgsConstructor
▪     public static class Data {
▪         private Transaction transaction;
▪     }
▪
▪     @Data
▪     @NoArgsConstructor
▪     @AllArgsConstructor
▪     public static class Transaction {
▪         @JsonProperty("reference_id")
▪         private String referenceId;
```

- @JsonProperty("airtel_money_id")
- private String airtelMoneyId;
- private String id;
- private String status;
- private String message;
- }
-
- @Data
- @NoArgsConstructor
- @AllArgsConstructor
- public static class Status {
- @JsonProperty("response_code")
- private String responseCode;
- private String code;
- private Boolean success;
- private String message;
- }
- }

- **model/PaymentCallback.java (JPA Entity for storing raw Airtel callbacks):**
- package com.gocashless.pps.model;
-
- import jakarta.persistence.*;
- import lombok.Data;
- import lombok.NoArgsConstructor;

```
import lombok.AllArgsConstructor;
import java.time.LocalDateTime;
import java.util.UUID;
import com.fasterxml.jackson.databind.JsonNode;
import com.vladmihalcea.hibernate.type.json.JsonBinaryType; // For PostgreSQL JSONB type
import org.hibernate.annotations.Type; // For Hibernate 6+ use @JdbcTypeCode(SqlTypes.JSON)

/**
 * Entity to store raw callback data received from Airtel Mobile Money.
 * This is crucial for auditing and debugging payment statuses.
 */
@Entity
@Table(name = "payment_callbacks")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class PaymentCallback {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;

    @Column(nullable = false)
    private String transactionRef; // Our internal transaction ID

    private String airtelMoneyId; // Airtel's transaction ID

    @Column(nullable = false)
    private String status; // Status reported by Airtel (e.g., "SUCCESS", "FAILED")

    @Column(columnDefinition = "jsonb") // Use jsonb for PostgreSQL
    @Type(JsonBinaryType.class) // Hibernate type for JSONB
    // For Hibernate 6+, use: @JdbcTypeCode(SqlTypes.JSON)
    private JsonNode rawCallbackData; // Store the entire raw JSON payload

    private LocalDateTime receivedAt;

    @PrePersist
    protected void onCreate() {
        receivedAt = LocalDateTime.now();
    }
}
```

- **repository/PaymentCallbackRepository.java (Spring Data JPA Repository):**

```
package com.gocashless.pps.repository;

import com.gocashless.pps.model.PaymentCallback;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;
import java.util.UUID;
```

-
- /**
- * Spring Data JPA Repository for managing PaymentCallback entities.
- * Provides standard CRUD operations and custom query methods.
- */
- public interface PaymentCallbackRepository extends JpaRepository<PaymentCallback, UUID> {
-     /**
-      * Finds the latest callback for a given internal transaction reference.
-      * This is useful for retrieving the most up-to-date status from callbacks.
-      * @param transactionRef The internal transaction reference ID.
-      * @return An Optional containing the latest PaymentCallback, or empty if not found.
-      */
-     Optional<PaymentCallback> findFirstByTransactionRefOrderByReceivedAtDesc(String transactionRef);
-
-     // You might add other query methods as needed, e.g.,
-     // List<PaymentCallback> findByAirtelMoneyId(String airtelMoneyId);
-     // List<PaymentCallback> findByStatus(String status);
- }

- **service/PaymentProcessingService.java (Core Business Logic Service):**
- package com.gocashless.pps.service;
-
- import com.fasterxml.jackson.databind.JsonNode;
- import com.fasterxml.jackson.databind.ObjectMapper;

```
import com.gocashless.pps.airtel.client.AirtelApiClient;
import com.gocashless.pps.airtel.dto.AirtelPaymentRequest;
import com.gocashless.pps.airtel.dto.AirtelPaymentResponse;
import com.gocashless.pps.airtel.dto.AirtelDisbursementRequest;
import com.gocashless.pps.airtel.dto.AirtelDisbursementResponse;
import com.gocashless.pps.dto.InitiatePaymentRequest;
import com.gocashless.pps.dto.InitiatePaymentResponse;
import com.gocashless.pps.dto.PaymentStatusResponse;
import com.gocashless.pps.model.PaymentCallback; // Import the new entity
import com.gocashless.pps.repository.PaymentCallbackRepository; // Import the new repository
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.util.UUID;

/**
 * Service responsible for orchestrating payment processing.
 * It interacts with the Airtel API client and will eventually
 * communicate with the Transaction History Service (THS) and
 * Notification Service (NOS).
 */
@Service
```

```
▪ public class PaymentProcessingService {
▪
▪     private static final Logger logger = LoggerFactory.getLogger(PaymentProcessingService.class);
▪
▪     private final AirtelApiClient airtelApiClient;
▪     private final ObjectMapper objectMapper; // To deserialize QR payload
▪     private final PaymentCallbackRepository paymentCallbackRepository; // Inject the new repository
▪
▪     // TODO: Inject proxies for THS and NOS here
▪     // private final TransactionHistoryProxy transactionHistoryProxy;
▪     // private final NotificationServiceProxy notificationServiceProxy;
▪
▪     @Autowired
▪     public PaymentProcessingService(AirtelApiClient airtelApiClient, ObjectMapper objectMapper,
▪                                     PaymentCallbackRepository paymentCallbackRepository) { // Add to constructor
▪         this.airtelApiClient = airtelApiClient;
▪         this.objectMapper = objectMapper;
▪         this.paymentCallbackRepository = paymentCallbackRepository; // Initialize
▪         // TODO: Initialize proxies
▪         // this.transactionHistoryProxy = transactionHistoryProxy;
▪         // this.notificationServiceProxy = notificationServiceProxy;
▪     }
```

```java

    /**
     * Initiates a payment request (Collection) from a passenger.
     * This method receives the decrypted QR payload.
     *
     * @param request The payment initiation request from the Passenger App.
     * @return InitiatePaymentResponse indicating the status of the request.
     */
    public InitiatePaymentResponse initiatePassengerPayment(InitiatePaymentRequest request) {
        logger.info("Initiating passenger payment for transactionRef: {}", request.getTransactionRef());

        try {
            // 1. Prepare Airtel Payment Request DTO
            AirtelPaymentRequest airtelRequest = new AirtelPaymentRequest();
            airtelRequest.setReference("Gocashless Payment for " + request.getTransactionRef());

            AirtelPaymentRequest.Subscriber subscriber = new AirtelPaymentRequest.Subscriber();
subscriber.setMsisdn(request.getPassengerPhoneNumber().startsWith("0") ?
                    request.getPassengerPhoneNumber().substring(1) : request.getPassengerPhoneNumber()); // Remove leading 0
```

- subscriber.setCountry("ZM"); // From config
- subscriber.setCurrency("ZMW"); // From config
- airtelRequest.setSubscriber(subscriber);
-
- AirtelPaymentRequest.Transaction transaction = new AirtelPaymentRequest.Transaction();
- transaction.setAmount(request.getAmount());
- transaction.setCountry("ZM"); // From config
- transaction.setCurrency("ZMW"); // From config
- transaction.setId(request.getTransactionRef()); // Use the unique ID from QR code
- airtelRequest.setTransaction(transaction);
-
- // 2. Call Airtel Collections API
- AirtelPaymentResponse airtelResponse = airtelApiClient.initiateCollectionPayment(airtelRequest);
- logger.info("Airtel Collection API response: {}", airtelResponse);
-
- // 3. Record initial transaction status in THS (PENDING, or based on immediate Airtel response)
- // TODO: transactionHistoryProxy.saveTransaction(request.getPassengerId(), request.getConductorId(),
- //   request.getAmount(), request.getCurrency(),
- //   request.getTransactionRef(), airtelResponse.getData().getTransaction().getId(),
- //   "PENDING_AIRTEL_PIN_CONFIRMATION");

```

            // 4. Return response to Passenger App
            if (airtelResponse != null && airtelResponse.getStatus().getSuccess()) {
                return new InitiatePaymentResponse(
                        request.getTransactionRef(),
                        airtelResponse.getData().getTransaction().getId(),
                        "PENDING", // Airtel sends USSD push, payment is pending user PIN
                        "Payment request sent. Please confirm on your mobile money app."
                );
            } else {
                String errorMessage = airtelResponse != null ? airtelResponse.getStatus().getMessage() : "Unknown error from Airtel.";
                return new InitiatePaymentResponse(
                        request.getTransactionRef(),
                        null,
                        "FAILED",
                        "Payment initiation failed: " + errorMessage
                );
            }

        } catch (Exception e) {
            logger.error("Error initiating passenger payment for transactionRef {}: {}", request.getTransactionRef(), e.getMessage(), e);
            // TODO: transactionHistoryProxy.saveFailedTransaction(...)
            return new InitiatePaymentResponse(
                request.getTransactionRef(),
                null,
                "FAILED",
                "An internal error occurred: " + e.getMessage()
            );
        }
    }

    /**
     * Handles the callback from Airtel Mobile Money API for Collections.
     * This is the endpoint Airtel will hit to notify about payment success/failure.
     *
     * @param airtelCallback The raw JSON payload from Airtel.
     * @return A response confirming receipt of the callback.
     */
    public JsonNode handleAirtelCollectionCallback(JsonNode airtelCallback) {
        logger.info("Received Airtel Collection Callback: {}", airtelCallback.toPrettyString());

        try {
            String transactionRef = airtelCallback.at("/data/transaction/id").asText(); // Partner's transaction ID
        String airtelMoneyId = airtelCallback.at("/data/transaction/airtel_money_id").asText(); // Airtel's ID (if available)
        String status = airtelCallback.at("/data/transaction/status").asText(); // e.g., "SUCCESS", "FAILED"
        String message = airtelCallback.at("/status/message").asText();

        // Save the raw callback data for auditing and debugging
        PaymentCallback callbackEntity = new PaymentCallback();
        callbackEntity.setTransactionRef(transactionRef);
        callbackEntity.setAirtelMoneyId(airtelMoneyId);
        callbackEntity.setStatus(status);
        callbackEntity.setRawCallbackData(airtelCallback);
        paymentCallbackRepository.save(callbackEntity);
        logger.info("Saved Airtel callback for transactionRef: {}", transactionRef);

        // Map Airtel status to Gocashless internal status
        String gocashlessStatus;
        if ("SUCCESS".equalsIgnoreCase(status) || "TS".equalsIgnoreCase(status)) { // "TS" is Transaction Successful from 1.txt
            gocashlessStatus = "SUCCESS";
        } else if ("FAILED".equalsIgnoreCase(status)) {
            gocashlessStatus = "FAILED";
        } else {
            gocashlessStatus = "UNKNOWN";
```

```
▪           }
▪
▪           logger.info("Processing Airtel callback for transactionRef: {}, Status: {}", transactionRef, gocashlessStatus);
▪
▪           // 1. Update transaction status in THS
▪           // TODO: transactionHistoryProxy.updateTransactionStatus(transactionRef, gocashlessStatus, airtelMoneyId, airtelCallback);
▪
▪           // 2. Notify relevant parties (Conductor, Passenger) via NOS
▪           // TODO: notificationServiceProxy.sendPaymentStatusNotification(transactionRef, gocashlessStatus, message);
▪
▪           // Return a success response to Airtel to acknowledge receipt
▪           ObjectNode response = objectMapper.createObjectNode();
▪           response.put("status", "RECEIVED");
▪           response.put("message", "Callback processed successfully");
▪           return response;
▪
▪       } catch (Exception e) {
▪           logger.error("Error processing Airtel Collection Callback: {}", e.getMessage(), e);
▪           // Log the error and return an appropriate response to Airtel
▪           ObjectNode errorResponse = objectMapper.createObjectNode();
▪           errorResponse.put("status", "ERROR");
▪             errorResponse.put("message", "Failed to process callback: " + e.getMessage());
▪             return errorResponse;
▪         }
▪     }
▪
▪     /**
▪      * Initiates a disbursement (e.g., for refunds or conductor payouts).
▪      *
▪      * @param phoneNumber The recipient's phone number.
▪      * @param amount The amount to disburse.
▪      * @param transactionId A unique ID for this disbursement.
▪      * @param walletType The type of wallet (e.g., "NORMAL", "SALARY").
▪      * @param transactionType The type of transaction (e.g., "B2C", "B2B").
▪      * @return AirtelDisbursementResponse from the Airtel API.
▪      */
▪     public AirtelDisbursementResponse initiateDisbursement(
▪             String phoneNumber, BigDecimal amount, String transactionId,
▪             String walletType, String transactionType) {
▪         logger.info("Initiating disbursement for phone: {} with amount: {}", phoneNumber, amount);
▪
▪         try {
▪             AirtelDisbursementRequest airtelRequest = new AirtelDisbursementRequest();
▪             airtelRequest.setReference("Gocashless Disbursement for " + transactionId);
▪
▪             AirtelDisbursementRequest.Payee payee = new AirtelDisbursementRequest.Payee();
▪             payee.setMsisdn(phoneNumber.startsWith("0") ? phoneNumber.substring(1) : phoneNumber); // Remove leading 0
▪             payee.setWalletType(walletType);
▪             airtelRequest.setPayee(payee);
▪
▪             AirtelDisbursementRequest.Transaction transaction = new AirtelDisbursementRequest.Transaction();
▪             transaction.setAmount(amount);
▪             transaction.setId(transactionId);
▪             transaction.setType(transactionType);
▪             airtelRequest.setTransaction(transaction);
▪
▪             return airtelApiClient.initiateDisbursement(airtelRequest);
▪         } catch (Exception e) {
▪             logger.error("Error initiating disbursement for transaction {}: {}", transactionId, e.getMessage(), e);
▪             throw new RuntimeException("Failed to initiate disbursement", e);
▪         }
▪     }
▪
▪     /**
▪      * Retrieves the status of a payment/transaction.
```

```
     * This could be used for polling or manual verification.
     *
     * @param transactionRef The internal transaction reference ID.
     * @return PaymentStatusResponse with the current status.
     */
    public PaymentStatusResponse getPaymentStatus(String transactionRef) {
        logger.info("Getting payment status for transactionRef: {}", transactionRef);
        // In a real system, this would query THS first, and only call Airtel if THS status is PENDING/UNKNOWN
        // For now, directly calling Airtel for demonstration.

        try {
            // First, try to get status from our stored callbacks
            Optional<PaymentCallback> latestCallback = paymentCallbackRepository.findFirstByTransactionRefOrderByReceivedAtDesc(transactionRef);
            if (latestCallback.isPresent()) {
                PaymentCallback callback = latestCallback.get();
                String status = callback.getStatus();
                String message = callback.getRawCallbackData().at("/status/message").asText(); // Extract message from raw data

                String gocashlessStatus;
                if ("SUCCESS".equalsIgnoreCase(status) || "TS".equalsIgnoreCase(status)) {
```

- gocashlessStatus = "SUCCESS";
- } else if ("FAILED".equalsIgnoreCase(status)) {
- gocashlessStatus = "FAILED";
- } else {
- gocashlessStatus = "PENDING"; // Default for other statuses
- }
- logger.info("Status for transactionRef {} retrieved from local callback: {}", transactionRef, gocashlessStatus);
- return new PaymentStatusResponse(transactionRef, gocashlessStatus, message);
- }
-
-
- // If not found in local callbacks, or if status is still PENDING/UNKNOWN, call Airtel API
- logger.warn("No local callback found for transactionRef {} or status is not final. Calling Airtel API.", transactionRef);
- JsonNode airtelResponse = airtelApiClient.verifyCollectionTransaction(transactionRef);
- String status = airtelResponse.at("/data/transaction/status").asText();
- String message = airtelResponse.at("/status/message").asText();
-
- String gocashlessStatus;
- if ("SUCCESS".equalsIgnoreCase(status) || "TS".equalsIgnoreCase(status)) {
- gocashlessStatus = "SUCCESS";
- } else if ("FAILED".equalsIgnoreCase(status)) {

- gocashlessStatus = "FAILED";
- } else {
- gocashlessStatus = "PENDING"; // Default for other statuses
- }
-
- // Optionally, save this new status from Airtel to your local callback repository
- // This helps keep your local data consistent with Airtel's latest status
- PaymentCallback newCallback = new PaymentCallback();
- newCallback.setTransactionRef(transactionRef);
- newCallback.setAirtelMoneyId(airtelResponse.at("/data/transaction/airtel_money_id").asText());
- newCallback.setStatus(status);
- newCallback.setRawCallbackData(airtelResponse);
- paymentCallbackRepository.save(newCallback);
- logger.info("Updated local callback with status from Airtel for transactionRef: {}", transactionRef);
-
-
- return new PaymentStatusResponse(transactionRef, gocashlessStatus, message);
- } catch (Exception e) {
- logger.error("Error getting payment status for transactionRef {}: {}", transactionRef, e.getMessage(), e);
- return new PaymentStatusResponse(transactionRef, "UNKNOWN", "Failed to retrieve status: " + e.getMessage());

-         }
-     }
-
-     // Placeholder for other service proxies
-     // public void setTransactionHistoryProxy(TransactionHistoryProxy transactionHistoryProxy) {
-     //   this.transactionHistoryProxy = transactionHistoryProxy;
-     // }
-     // public void setNotificationServiceProxy(NotificationServiceProxy notificationServiceProxy) {
-     //   this.notificationServiceProxy = notificationServiceProxy;
-     // }
- }

- **controller/PaymentController.java (REST Controller for Payments):**
- package com.gocashless.pps.controller;
-
- import com.gocashless.pps.dto.InitiatePaymentRequest;
- import com.gocashless.pps.dto.InitiatePaymentResponse;
- import com.gocashless.pps.dto.PaymentStatusResponse;
- import com.gocashless.pps.service.PaymentProcessingService;
- import org.slf4j.Logger;
- import org.slf4j.LoggerFactory;
- import org.springframework.beans.factory.annotation.Autowired;
- import org.springframework.http.HttpStatus;
- import org.springframework.http.ResponseEntity;
- import org.springframework.web.bind.annotation.*;

```

/**
 * REST Controller for handling payment initiation and status queries.
 * This is the primary interface for the Passenger App.
 */
@RestController
@RequestMapping("/api/v1/payments")
public class PaymentController {

    private static final Logger logger = LoggerFactory.getLogger(PaymentController.class);

    private final PaymentProcessingService paymentProcessingService;

    @Autowired
    public PaymentController(PaymentProcessingService paymentProcessingService) {
        this.paymentProcessingService = paymentProcessingService;
    }

    /**
     * Endpoint for Passenger App to initiate a payment.
     * The request payload is the decrypted data from the scanned QR code.
     *
     * @param request The payment request details (from QR code).
     * @return ResponseEntity with payment initiation status.
     */
    @PostMapping("/initiate")
    public ResponseEntity<InitiatePaymentResponse> initiatePayment(@RequestBody InitiatePaymentRequest request) {
        logger.info("Received payment initiation request from Passenger App for transactionRef: {}", request.getTransactionRef());
        InitiatePaymentResponse response = paymentProcessingService.initiatePassengerPayment(request);
        if ("FAILED".equals(response.getStatus())) {
            return new ResponseEntity<>(response, HttpStatus.INTERNAL_SERVER_ERROR);
        }
        return new ResponseEntity<>(response, HttpStatus.OK);
    }

    /**
     * Endpoint to get the current status of a payment.
     * Can be used by Passenger/Conductor App to poll for status updates.
     *
     * @param transactionRef The unique transaction reference ID.
     * @return ResponseEntity with the payment status.
     */
    @GetMapping("/status/{transactionRef}")
    public ResponseEntity<PaymentStatusResponse> getPaymentStatus(@PathVariable String transactionRef) {
        logger.info("Received request for payment status for transactionRef: {}", transactionRef);
```

- PaymentStatusResponse response = paymentProcessingService.getPaymentStatus(transactionRef);
- return new ResponseEntity<>(response, HttpStatus.OK);
- }
-
- // You might add an endpoint for initiating disbursements if needed for specific admin actions
- // @PostMapping("/disburse")
- // public ResponseEntity<?> disburseFunds(@RequestBody DisbursementRequest request) { ... }
- }

- **controller/AirtelCallbackController.java (REST Controller for Airtel Callbacks):**
- package com.gocashless.pps.controller;
-
- import com.fasterxml.jackson.databind.JsonNode;
- import com.gocashless.pps.service.PaymentProcessingService;
- import org.slf4j.Logger;
- import org.slf4j.LoggerFactory;
- import org.springframework.beans.factory.annotation.Autowired;
- import org.springframework.http.HttpStatus;
- import org.springframework.http.ResponseEntity;
- import org.springframework.web.bind.annotation.PostMapping;
- import org.springframework.web.bind.annotation.RequestBody;
- import org.springframework.web.bind.annotation.RequestMapping;
- import org.springframework.web.bind.annotation.RestController;

```

/**
 * REST Controller to handle incoming webhook callbacks from Airtel Mobile Money.
 * This endpoint should be configured in your Airtel developer portal as a callback URL.
 */
@RestController
@RequestMapping("/airtel/callback") // Specific path for Airtel callbacks
public class AirtelCallbackController {

    private static final Logger logger = LoggerFactory.getLogger(AirtelCallbackController.class);

    private final PaymentProcessingService paymentProcessingService;

    @Autowired
    public AirtelCallbackController(PaymentProcessingService paymentProcessingService) {
        this.paymentProcessingService = paymentProcessingService;
    }

    /**
     * Endpoint to receive payment status updates from Airtel Mobile Money.
     * Airtel will send a POST request to this URL when a payment status changes.
     *
     * @param airtelCallback The raw JSON payload from Airtel.
     * @return ResponseEntity acknowledging receipt of the callback.
     */
    @PostMapping("/collection")
    public ResponseEntity<JsonNode> handleCollectionCallback(@RequestBody JsonNode airtelCallback) {
        logger.info("Received Airtel Collection Callback.");
        JsonNode response = paymentProcessingService.handleAirtelCollectionCallback(airtelCallback);
        return new ResponseEntity<>(response, HttpStatus.OK);
    }

    // Add other callback endpoints if Airtel has separate ones for disbursements, etc.
    // @PostMapping("/disbursement")
    // public ResponseEntity<JsonNode> handleDisbursementCallback(@RequestBody JsonNode airtelCallback) { ... }
}
```

- **dto/InitiatePaymentRequest.java (Internal DTO for payment initiation from QRGS):**

```
package com.gocashless.pps.dto;

import lombok.Data;
import java.math.BigDecimal;
import java.util.UUID;

```

- /**
- * DTO representing the decrypted payload from the QR code,
- * sent by the Passenger App to PPS to initiate payment.
- */
- @Data
- public class InitiatePaymentRequest {
-     private UUID conductorId;
-     private String passengerPhoneNumber; // This would be the passenger's phone number from their app/profile
-     private BigDecimal amount;
-     private String currency;
-     private String transactionRef; // The unique ID generated by QRGS
-     // Add passengerId if available from UMS
-     private UUID passengerId;
- }

- **dto/InitiatePaymentResponse.java (Internal DTO for payment initiation response to Passenger App):**
- package com.gocashless.pps.dto;
-
- import lombok.AllArgsConstructor;
- import lombok.Data;
- import lombok.NoArgsConstructor;
-
- /**
- * DTO representing the response to the Passenger App after initiating a payment.

- */
- @Data
- @NoArgsConstructor
- @AllArgsConstructor
- public class InitiatePaymentResponse {
-     private String transactionRef;
-     private String paymentGatewayTransactionId; // Airtel's transaction ID, if available immediately
-     private String status; // e.g., "PENDING", "SUCCESS", "FAILED"
-     private String message;
- }

- **dto/PaymentStatusResponse.java (Internal DTO for payment status query):**
- package com.gocashless.pps.dto;
-
- import lombok.AllArgsConstructor;
- import lombok.Data;
- import lombok.NoArgsConstructor;
-
- /**
- * DTO representing the response for a payment status query.
- */
- @Data
- @NoArgsConstructor
- @AllArgsConstructor
- public class PaymentStatusResponse {

        - private String transactionRef;
        - private String status; // e.g., "PENDING", "SUCCESS", "FAILED", "UNKNOWN"
        - private String message;
        - }

- **Key Entities/Data Models:** (Primarily processes and forwards to THS)
    - **PaymentRequest (Internal DTO):**
        - passengerPhoneNumber (String)
        - conductorId (UUID)
        - amount (BigDecimal)
        - currency (String)
        - transactionRef (String, from QR code)
        - paymentGatewayTransactionId (String, from Airtel MoMo)
        - status (Enum: PENDING, SUCCESS, FAILED, REFUNDED)
        - timestamp (Timestamp)
- **Core Functionality/APIs:**
    - POST /api/v1/payments/initiate:
        - **Request Body:** {"phoneNumber": "...", "amount": "...", "transactionRef": "..."} (This is the parsed data from the QR code)
        - **Process:**
            1. Receive payment request from Passenger App.
            2. Call AirtelApiClient.initiateCollectionPayment() to initiate payment with Airtel MoMo.
            3. Record an initial PENDING transaction in THS.
            4. Return immediate response to Passenger App (e.g., "Payment request sent, awaiting PIN confirmation").

   - POST /airtel/callback/collection:
     - **Request Body:** Airtel MoMo webhook payload (contains transaction ID, status, etc.).
     - **Process:**
       1. Receive callback from Airtel MoMo.
       2. Parse callback data to get transactionRef and status.
       3. **Store the raw callback data in PaymentCallbackRepository.**
       4. Update the transaction status in THS (e.g., SUCCESS or FAILED).
       5. Notify NOS about the payment status change.
   - GET /api/v1/payments/status/{transactionRef}:
     - **Process: First, query PaymentCallbackRepository for the latest status.** If not found or status is not final, call AirtelApiClient.verifyCollectionTransaction() for real-time check with Airtel.
   - POST /api/v1/payments/disburse: (New endpoint for initiating disbursements)
     - **Request Body:** {"phoneNumber": "...", "amount": "...", "walletType": "...", "transactionType": "..."}
     - **Process:** Calls AirtelApiClient.initiateDisbursement() to transfer funds.
- **Interactions:**
   - **Passenger App:** Calls /api/v1/payments/initiate endpoint.
   - **Airtel MoMo API:** Direct communication for payment initiation and receiving callbacks (/airtel/callback/collection).
   - **Transaction History Service (THS):** Updates transaction records (TODO: implement proxy).
   - **Notification Service (NOS):** Triggers notifications based on payment status (TODO: implement proxy).

5. Transaction History Service (THS)

- **Purpose:** Stores and provides access to all payment transaction records for passengers, conductors, and bus companies.

- **Key Entities/Data Models:**
  - **Transaction:**
    - id (UUID)
    - transactionRef (String, unique, from QRGS/PPS)
    - paymentGatewayTransactionId (String, unique, from Airtel MoMo)
    - passengerId (UUID, foreign key to User)
    - conductorId (UUID, foreign key to User)
    - busCompanyId (UUID, foreign key to BusCompany)
    - routeId (UUID)
    - originStopId (UUID)
    - destinationStopId (UUID)
    - amount (BigDecimal)
    - currency (String)
    - status (Enum: PENDING, SUCCESS, FAILED, REFUNDED)
    - transactionType (Enum: PAYMENT, DISBURSEMENT, REFUND)
    - timestamp (Timestamp)
    - lastUpdated (Timestamp)
    - metadata (JSONB/Map for additional details from MoMo callback)
- **Core Functionality/APIs:**
  - POST /api/v1/transactions: Create/Update a transaction record (primarily called by PPS).
  - GET /api/v1/transactions/{id}: Retrieve a single transaction by ID.
  - GET /api/v1/transactions/passenger/{passengerId}: Get transaction history for a passenger.
  - GET /api/v1/transactions/conductor/{conductorId}: Get transaction history for a conductor.

    - GET /api/v1/transactions/company/{companyId}: Get aggregated transaction history for a bus company (for dashboard).
    - GET /api/v1/transactions/company/{companyId}/summary: Get daily/monthly/custom range summaries for bus companies.
    - GET /api/v1/transactions/search: Advanced search/filter for transactions (e.g., by date range, status, amount).
- **Interactions:**
    - **Payment Processing Service (PPS):** Writes and updates transaction records.
    - **Passenger App, Conductor App, Bus Company Web App:** Read transaction history for display and reporting.
    - **User Management Service (UMS):** Queries UMS to enrich transaction data with user/company names when needed for display.
    - **Route & Fare Management Service (RFMS):** Queries RFMS to enrich transaction data with route/stop names.

6. Notification Service (NOS)

- **Purpose:** Delivers real-time notifications to users (passengers, conductors) and potentially bus companies about payment status changes and other relevant events.
- **Key Entities/Data Models:**
    - **Notification:**
        - id (UUID)
        - userId (UUID)
        - type (Enum: PAYMENT_SUCCESS, PAYMENT_FAILURE, DISBURSEMENT_RECEIVED, NEW_MESSAGE, etc.)
        - message (String)
        - read (Boolean)
        - timestamp (Timestamp)
        - relatedEntityId (UUID, e.g., transaction ID)
- **Core Functionality/APIs:**

    - POST /api/v1/notifications/send:
        - **Request Body:** {"userId": "...", "type": "...", "message": "...", "relatedEntityId": "..."}
        - **Process:** Sends a notification. This might involve:
            - Storing the notification in its own database.
            - Publishing to a message broker (e.g., Kafka, RabbitMQ) for real-time delivery.
            - Integrating with a push notification service (e.g., Firebase Cloud Messaging for mobile apps).
            - Using WebSockets for real-time updates to connected clients.
    - GET /api/v1/notifications/user/{userId}: Retrieve unread/all notifications for a specific user.
    - PUT /api/v1/notifications/{id}/read: Mark a notification as read.
- **Interactions:**
    - **Payment Processing Service (PPS):** Triggers notifications upon payment success or failure.
    - **Other Services (UMS, RFMS, THS):** Could potentially trigger notifications for other events (e.g., new conductor registered, fare changes, suspicious activity).
    - **Passenger App, Conductor App:** Consume notifications in real-time or by polling.

General Backend Considerations:

- **API Gateway:** For a production microservices setup, an API Gateway (e.g., Spring Cloud Gateway, Netflix Zuul) would sit in front of these services to handle routing, load balancing, authentication, and rate limiting.
- **Service Discovery:** Services would register with a service discovery mechanism (e.g., Eureka, Consul) to find each other.
- **Centralized Logging & Monitoring:** Implement solutions like ELK stack (Elasticsearch, Logstash, Kibana) or Prometheus/Grafana for monitoring the health and performance of individual services.
- **Distributed Tracing:** Tools like Zipkin or Jaeger would be crucial for understanding requests flowing across multiple services.

- **Message Broker:** A message broker (Kafka, RabbitMQ) can facilitate asynchronous communication between services, improving resilience and scalability, especially for notifications and event-driven updates.
- **Database Choice:** While PostgreSQL/MongoDB are mentioned, each service could potentially have its own database instance (polyglot persistence) or share a database with separate schemas, depending on the data needs and team preferences. For simplicity, initially, a single shared database with distinct schemas per service can work.

This detailed design provides a solid foundation for implementing your Gocashless backend microservices.