Perfect, Nani! Let’s replicate the **State Farm B2B Portal** as your own B2B Insurance Partner Portal project from scratch with a full technical setup—front end, back end, microservices, security, deployment, and cloud—all structured and detailed for implementation.

**✅ Project Name: Smart B2B Partner Portal**

**🔧 1. Project Setup Overview**

**A. Tech Stack**

| **Layer** | **Technology** |
| --- | --- |
| Frontend | React.js (with JSX) |
| Backend | Spring Boot (Java 17) |
| Microservices | Spring Cloud + Eureka + Config Server |
| Database | MySQL |
| Auth & Security | Spring Security + LDAP + Vault |
| Secrets Mgmt | HashiCorp Vault |
| CI/CD | GitLab Pipelines |
| Containerization | Docker |
| Orchestration | Kubernetes (EKS or ROSA) |
| Cloud Platform | AWS |

**📁 2. Backend Microservices Structure**

smart-b2b-portal/

├── config-server/

├── discovery-server/

├── api-gateway/

├── auth-service/

├── partner-service/ # Repair shops, hospitals, suppliers

├── billing-service/ # Medical/EFT billing

├── supplier-service/ # Coupa interactions

├── common-lib/ # DTOs, Utils, Enums

└── vault-config/

**✅ 3. Key Microservices**

**🔐 auth-service**

Handles LDAP login, TPAR checks, and JWT generation.

**Classes / Interfaces:**

* LdapAuthProvider.java
* TPARService.java (calls TPAR APIs)
* AuthController.java
* JwtUtil.java
* UserPrincipal.java

**DTOs:**

* LoginRequest, LoginResponse
* TPARRoleDto, UserDto

**Functionality:**

* Authenticates using LDAP
* Validates roles using TPAR
* Issues JWT on successful login

**👥 partner-service**

Manages role-based module access (Medical, EFT, SelectService, Supplier)

**Entities:**

* Partner
* PartnerRole
* AccessModule

**DTOs:**

* PartnerDto, ModuleDto

**Services:**

* PartnerAccessService
* ModuleService

**Functionality:**

* Get partner details by TIMS ID
* Map TPAR roles to modules
* Control UI visibility via API

**🧾 billing-service**

Handles medical billing and EFT processing.

**Entities:**

* EFTDetails
* MedicalBill

**DTOs:**

* EFTDto, MedicalClaimDto

**Functionality:**

* Submit medical bills with claim IDs
* View EFT history, setup account

**📦 supplier-service**

Manages invoice and procurement via Coupa.

**Entities:**

* Supplier
* Invoice

**Services:**

* CoupaIntegrationService
* InvoiceService

**Functionality:**

* Submit & track invoices
* Fetch order status from Coupa

**🔗 Shared Lib: common-lib**

Includes:

* DTOs
* Constants
* Enums
* Feign clients (for inter-service communication)

**🔐 4. Spring Security Setup**

**LDAP + Spring Security:**

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.ldapAuthentication()

.userDnPatterns("uid={0},ou=people")

.contextSource()

.url("ldap://localhost:8389/dc=example,dc=com");

}

**Vault Integration:**

spring:

cloud:

vault:

uri: http://localhost:8200

authentication: TOKEN

token: s.1234567890abcdef

Use @Value("${vault.key}") to fetch credentials at runtime.

**🌐 5. React Frontend Structure (JSX)**

src/

├── components/

│ ├── Login.jsx

│ ├── Dashboard.jsx

│ ├── MedicalBillingForm.jsx

│ ├── EFTSetup.jsx

│ └── SupplierModule.jsx

├── services/

│ ├── authService.js

│ ├── billingService.js

│ └── supplierService.js

├── routes/

│ └── AppRouter.jsx

└── App.jsx

**Key Features:**

* Role-based UI rendering
* Form validation
* Axios for backend calls
* Token stored in localStorage
* Private Routes (Guarded Routes)

**🔁 6. CI/CD with GitLab**

stages:

- build

- test

- deploy

build-job:

script:

- ./mvnw clean package -DskipTests

docker-build:

script:

- docker build -t registry/b2b-portal:latest .

deploy-prod:

script:

- kubectl apply -f k8s/deployment.yaml

**🐳 7. Docker Setup**

**Dockerfile (backend):**

FROM openjdk:17

COPY target/\*.jar app.jar

ENTRYPOINT ["java", "-jar", "/app.jar"]

**Docker Compose (local dev):**

version: "3"

services:

auth-service:

image: auth-service:latest

ports:

- "8081:8081"

partner-service:

...

**☸️ 8. Kubernetes Setup**

**deployment.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: auth-service

spec:

replicas: 2

template:

spec:

containers:

- name: auth-service

image: registry/auth-service:latest

env:

- name: VAULT\_TOKEN

valueFrom:

secretKeyRef:

name: vault-token

key: token

**☁️ 9. AWS Deployment (EKS or ROSA)**

* Use **EKS** (Elastic Kubernetes Service) or **ROSA**
* Store secrets in **AWS Secrets Manager or Vault**
* Use **Route53** for DNS
* Load balancer via **Ingress Controller**
* Logs shipped to **CloudWatch/Splunk**

**🛠️ 10. Functional Workflow**

**Login Flow**

1. User logs in with TIMS ID → Authenticated via LDAP
2. JWT token generated → Frontend stores it
3. Calls /getUserModules → TPAR validated → Modules fetched
4. UI shows only allowed modules (Medical, EFT, etc.)

**Billing Flow**

1. Medical Provider submits form → Validated
2. Backend checks TPAR access → Encrypts data using Vault key
3. Sends data to claims-api → Claim stored

/\*

\* SMART B2B PARTNER PORTAL – CLASS STRUCTURE WITH TABLE DATA

\*/

// 1. USER ENTITY

@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String username;

private String password;

private String email;

@Enumerated(EnumType.STRING)

private Role role; // SUPPLIER, REPAIR\_SHOP, MEDICAL\_PROVIDER, LENDER

@OneToMany(mappedBy = "user")

private List<ModuleAccess> accesses;

}

// 2. MODULE ACCESS ENTITY

@Entity

@Table(name = "module\_access")

public class ModuleAccess {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String moduleName; // MEDICAL\_BILLING, EFT, SUPPLIER\_CENTER, etc.

@ManyToOne

@JoinColumn(name = "user\_id")

private User user;

}

// 3. MEDICAL BILL ENTITY

@Entity

@Table(name = "medical\_bills")

public class MedicalBill {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String claimNumber;

private String patientName;

private Double amount;

private LocalDate serviceDate;

@ManyToOne

@JoinColumn(name = "user\_id")

private User submittedBy;

}

// 4. EFT SETUP ENTITY

@Entity

@Table(name = "eft\_setups")

public class EFTSetup {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String bankName;

private String accountNumber;

private String routingNumber;

@OneToOne

@JoinColumn(name = "user\_id")

private User user;

}

// 5. SUPPLIER ENTITY

@Entity

@Table(name = "suppliers")

public class Supplier {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String supplierName;

private String contactPerson;

private String contactEmail;

private String status;

}

// 6. CLAIM LOOKUP ENTITY

@Entity

@Table(name = "claims")

public class Claim {

@Id

private String claimNumber;

private String policyHolderName;

private String status;

private LocalDate incidentDate;

private Double estimatedCost;

}

// 7. ENUM ROLE

public enum Role {

SUPPLIER,

REPAIR\_SHOP,

MEDICAL\_PROVIDER,

LENDER

}

/\*

\* EXAMPLE TABLE RECORDS (SIMPLIFIED VIEW)

\*/

-- users

INSERT INTO users (id, username, password, email, role) VALUES

(1, 'repair\_john', 'pass123', 'john@autofix.com', 'REPAIR\_SHOP'),

(2, 'clinic\_mary', 'secure456', 'mary@clinic.com', 'MEDICAL\_PROVIDER'),

(3, 'bank\_tim', 'banking789', 'tim@lender.com', 'LENDER'),

(4, 'vendor\_amy', 'supplyme', 'amy@supplier.com', 'SUPPLIER'),

(5, 'repair\_mike', 'fixit', 'mike@fixcars.com', 'REPAIR\_SHOP'),

(6, 'dr\_sam', 'doctor1', 'sam@med.com', 'MEDICAL\_PROVIDER'),

(7, 'loan\_joe', 'escrow99', 'joe@loanbank.com', 'LENDER'),

(8, 'supplier\_kim', 'goods88', 'kim@vendors.com', 'SUPPLIER'),

(9, 'repair\_raj', 'garage123', 'raj@repairs.in', 'REPAIR\_SHOP'),

(10, 'clinic\_lisa', 'med101', 'lisa@clinicplus.com', 'MEDICAL\_PROVIDER');

-- module\_access

INSERT INTO module\_access (id, module\_name, user\_id) VALUES

(1, 'MEDICAL\_BILLING', 2),

(2, 'EFT', 2),

(3, 'SUPPLIER\_CENTER', 4),

(4, 'CLAIMS\_MANAGEMENT', 1),

(5, 'MORTGAGE\_TOOLS', 3),

(6, 'EFT', 6),

(7, 'CLAIMS\_MANAGEMENT', 5),

(8, 'SUPPLIER\_CENTER', 8),

(9, 'CLAIMS\_MANAGEMENT', 9),

(10, 'MEDICAL\_BILLING', 10);

-- medical\_bills

INSERT INTO medical\_bills (id, claim\_number, patient\_name, amount, service\_date, user\_id) VALUES

(1, 'CLM001', 'John Smith', 3000.00, '2024-11-01', 2),

(2, 'CLM002', 'Linda Bell', 1800.00, '2024-11-15', 6),

(3, 'CLM003', 'Mark Lee', 2200.00, '2024-12-05', 10),

(4, 'CLM004', 'Sophia Gomez', 2600.00, '2024-12-18', 2),

(5, 'CLM005', 'Tom Hardy', 3100.00, '2025-01-08', 6),

(6, 'CLM006', 'Emma Watson', 1750.00, '2025-01-20', 10),

(7, 'CLM007', 'Bruce Wayne', 1900.00, '2025-02-10', 2),

(8, 'CLM008', 'Clark Kent', 2800.00, '2025-02-25', 6),

(9, 'CLM009', 'Diana Prince', 3300.00, '2025-03-03', 10),

(10, 'CLM010', 'Peter Parker', 2400.00, '2025-03-12', 2);

-- eft\_setups

INSERT INTO eft\_setups (id, bank\_name, account\_number, routing\_number, user\_id) VALUES

(1, 'Bank of America', '123456789', '00112233', 2),

(2, 'Chase', '234567890', '00112234', 6),

(3, 'Citi Bank', '345678901', '00112235', 10);

-- suppliers

INSERT INTO suppliers (id, supplier\_name, contact\_person, contact\_email, status) VALUES

(1, 'ABC Supplies', 'Amy Lee', 'amy@abc.com', 'Active'),

(2, 'FastParts', 'Kim Rios', 'kim@fastparts.com', 'Pending'),

(3, 'AutoDepot', 'Jake White', 'jake@autodepot.com', 'Active'),

(4, 'RepairKing', 'Mike Jet', 'mike@repairking.com', 'Inactive'),

(5, 'VendorX', 'Nina Ray', 'nina@vendorx.com', 'Active'),

(6, 'ToolHub', 'Chris Blue', 'chris@toolhub.com', 'Active'),

(7, 'FixMart', 'David Fox', 'david@fixmart.com', 'Pending'),

(8, 'PartsExpress', 'Sam Green', 'sam@partsx.com', 'Active'),

(9, 'EquipZone', 'Olga Brown', 'olga@equipzone.com', 'Active'),

(10, 'MotorParts', 'Liam Cross', 'liam@motorparts.com', 'Inactive');

-- claims

INSERT INTO claims (claim\_number, policy\_holder\_name, status, incident\_date, estimated\_cost) VALUES

('CLM001', 'John Smith', 'Open', '2024-10-15', 3200.00),

('CLM002', 'Linda Bell', 'Approved', '2024-10-20', 1800.00),

('CLM003', 'Mark Lee', 'Under Review', '2024-11-01', 2000.00),

('CLM004', 'Sophia Gomez', 'Denied', '2024-11-10', 2500.00),

('CLM005', 'Tom Hardy', 'Approved', '2024-12-01', 3000.00),

('CLM006', 'Emma Watson', 'Open', '2024-12-10', 1700.00),

('CLM007', 'Bruce Wayne', 'Approved', '2024-12-15', 1900.00),

('CLM008', 'Clark Kent', 'Under Review', '2025-01-01', 2800.00),

('CLM009', 'Diana Prince', 'Denied', '2025-01-15', 3300.00),

('CLM010', 'Peter Parker', 'Open', '2025-02-01', 2400.00);

* Users – with roles like Repair Shop, Medical Provider, Supplier, Lender
* Claims – associated with users
* MedicalBills – linked to medical provider users
* EFTPayments – linked to supplier users
* SupplierInvoices – from suppliers to State Farm

Great! You're working with a **modular microservices architecture** for the State Farm-style B2B Portal. Let's walk through **how each service communicates and what their responsibilities are**—especially when a **partner (like a repair shop or hospital) submits a claim**, and how the system processes that claim.

**🔄 High-Level Use Case: Claim Submission & Processing**

**Example:**  
A repair shop or hospital (partner) logs in and submits a claim for a policyholder.  
The claim goes through various states: **Pending → Approved/Denied**, depending on claim type, policy rules, and internal review.

**🔧 What Each Service Does and How They Interact**

**✅ auth-service**

* **Handles:** Authentication (e.g., TIMS login), LDAP-based user validation, role-based access via TPAR.
* **Works with:**
  + All services (for verifying tokens, roles).
  + Pulls user metadata (role: Repair Shop, Hospital, Supplier).
* **Responsibilities:**
  + Validate incoming request using JWT or session tokens.
  + Identify partner type and access level.

**✅ partner-service**

* **Handles:** Partner profiles like Repair Shops, Hospitals, Suppliers.
* **Works with:**
  + auth-service → Gets authenticated user role.
  + billing-service, supplier-service → Routes tasks like EFT/billing.
* **Responsibilities:**
  + Store/Fetch partner data (name, contact, TPAR id, permissions).
  + Provide dashboard-level data for partners.
  + Show available modules (Medical Billing, EFT, etc.) based on partner type.

**✅ billing-service**

* **Handles:** Claims and EFT billing data.
* **Works with:**
  + partner-service → Gets info on submitting partner.
  + common-lib → Uses shared DTOs like ClaimDTO, BillDTO.
  + vault-config → Fetches secure credentials for internal API calls.
* **Responsibilities:**
  + Accept claim submissions (e.g., accident report, hospital bill).
  + Validate policy type via external API (via api-gateway).
  + Set status: PENDING, then APPROVED, DENIED.
  + Assign reviewers from internal claims team.
  + Provide API for claim review UI (internal or via SFNet).

**✅ supplier-service**

* **Handles:** Interactions with external Coupa procurement system.
* **Works with:**
  + vault-config → Pull Coupa API credentials.
  + partner-service → Verify if current user is a supplier.
* **Responsibilities:**
  + Allow uploading invoices.
  + Track purchase orders & payments via Coupa API.

**✅ common-lib**

* **Handles:** Shared code: DTOs, Enums, Constants, Utils.
* **Used by:** Every other service.
* **Contains:**
  + ClaimDTO, PartnerDTO, EftDTO, ClaimStatus, PartnerType enums.
  + Date utils, file size validation, response wrapper classes.

**✅ api-gateway**

* **Handles:** Central routing of all external/internal API requests.
* **Responsibilities:**
  + Secure endpoints using Spring Cloud Gateway + JWT filter.
  + Route /claims/\*\* → billing-service.
  + Route /partners/\*\* → partner-service.

**✅ discovery-server**

* **Handles:** Service registration (Eureka).
* **Used by:** All services.
* **Responsibilities:**
  + Allow services to discover each other without hardcoded URLs.

**✅ config-server**

* **Handles:** Central config using Spring Cloud Config.
* **Responsibilities:**
  + Store all .yml configs (including Vault, DB, service ports).
  + Refresh service configs via /actuator/refresh.

**✅ vault-config**

* **Handles:** Secure secrets management via HashiCorp Vault.
* **Used by:** All services that call external/internal systems securely.
* **Responsibilities:**
  + Securely fetch DB creds, API keys.
  + Rotate secrets when needed.
  + Provide encryption/decryption for sensitive data (EFT, PII).

**💡 Example Flow: Repair Shop Submits Auto Repair Claim**

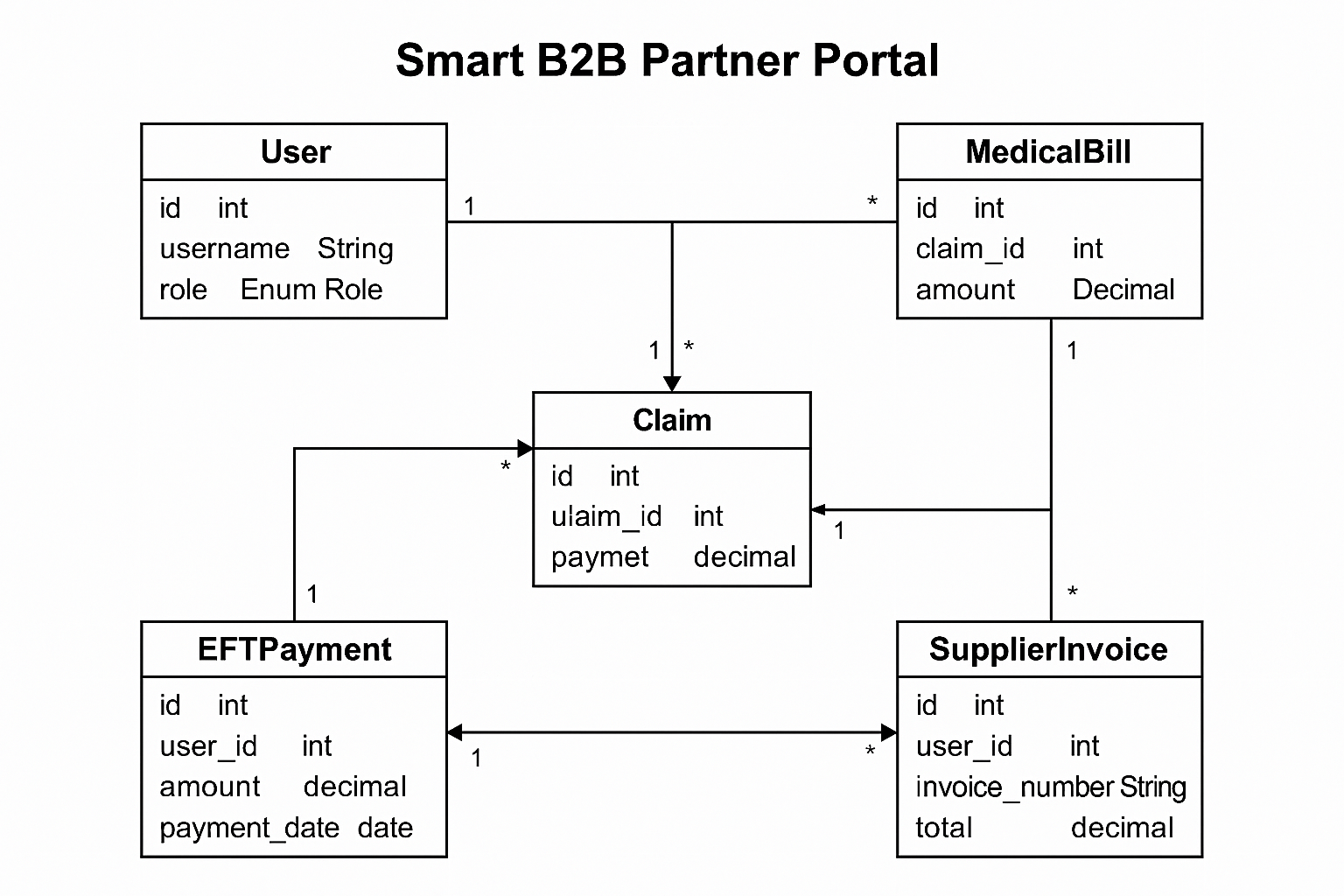
1. 🔐 **Login (via auth-service)**
   * User logs in via TIMS/LDAP
   * Token issued, role: REPAIR\_SHOP
2. 📋 **Partner Dashboard (partner-service)**
   * Fetch partner details from DB.
   * UI shows relevant modules based on partner type (e.g., “Select Service” claim submission).
3. 📝 **Submit Claim (billing-service)**
   * Repair shop submits damage report + cost estimate.
   * Claim is saved with status = PENDING
   * Partner info (TPAR ID, repair shop name) is stored.
4. 📡 **Backend Workflow**
   * Claim validated: VIN number, Policy type (auto/home), etc.
   * billing-service checks via API (e.g., claims API) if the policy is valid and active.
5. 🧑‍💼 **Claims Team Review**
   * Internal user sees the claim via reviewer dashboard.
   * Updates status to APPROVED or DENIED with reason.
6. 📩 **Notification**
   * Email or portal notification sent to partner.
   * Payment triggered via finance system (not part of this app).

**🧱 Policy Type Examples**

* **Auto Policy** → VIN must match repair report.
* **Health Policy** → Injury details must match medical billing.
* **Home Policy** → Only suppliers (roofing, electrical) can file this.

**🚀 Deployment Flow**

* Each service has a Dockerfile.
* Helm charts for each microservice.
* Deployed to Kubernetes (EKS or ROSA).
* Secrets mounted using Vault agents.
* CI/CD using GitLab pipelines:
  + Build → Test → Dockerize → Deploy to K8s



**// Smart B2B Partner Portal - Backend Full Setup**

// ─────────────────────────────────────────────────────────────

// COMMON MODULE: common-lib (DTOs, Enums, Utils)

// ─────────────────────────────────────────────────────────────

// 1. Enums

public enum ClaimStatus {

PENDING, APPROVED, DENIED;

}

public enum PolicyType {

HEALTH, AUTO, PROPERTY;

}

// 2. DTOs

public class PartnerDTO {

private Long id;

private String name;

private String type; // Hospital, Repair Shop, Supplier

private String email;

private String address;

}

public class ClaimDTO {

private Long id;

private Long policyId;

private String partnerName;

private ClaimStatus status;

private String reason;

}

// ─────────────────────────────────────────────────────────────

// AUTH SERVICE: auth-service

// ─────────────────────────────────────────────────────────────

@Entity

public class User {

@Id @GeneratedValue

private Long id;

private String username;

private String password;

private String role; // ADMIN, PARTNER, REVIEWER

}

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

Optional<User> findByUsername(String username);

}

@Service

public class AuthServiceImpl implements AuthService {

@Autowired private UserRepository userRepository;

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

return userRepository.findByUsername(username)

.orElseThrow(() -> new UsernameNotFoundException("User not found"));

}

}

// ─────────────────────────────────────────────────────────────

// PARTNER SERVICE: partner-service

// ─────────────────────────────────────────────────────────────

@Entity

public class Partner {

@Id @GeneratedValue

private Long id;

private String name;

private String type; // Hospital, Repair Shop, Supplier

private String email;

private String address;

@OneToMany(mappedBy = "partner")

private List<Claim> claims;

}

@Repository

public interface PartnerRepository extends JpaRepository<Partner, Long> {

List<Partner> findByType(String type);

}

@Service

public class PartnerServiceImpl implements PartnerService {

@Autowired private PartnerRepository partnerRepository;

public Partner createPartner(Partner partner) {

return partnerRepository.save(partner);

}

public List<Partner> getAllHospitals() {

return partnerRepository.findByType("Hospital");

}

public List<Partner> getAllPartners() {

return partnerRepository.findAll();

}

}

// ─────────────────────────────────────────────────────────────

// CLAIM SERVICE: billing-service

// ─────────────────────────────────────────────────────────────

@Entity

public class Claim {

@Id @GeneratedValue

private Long id;

@ManyToOne(fetch = FetchType.LAZY)

private Partner partner;

private Long policyId;

private ClaimStatus status;

private String reason;

}

@Repository

public interface ClaimRepository extends JpaRepository<Claim, Long> {

List<Claim> findByStatus(ClaimStatus status);

List<Claim> findByPartner\_Id(Long partnerId);

}

@Service

public class ClaimServiceImpl implements ClaimService {

@Autowired private ClaimRepository claimRepository;

@Autowired private PartnerRepository partnerRepository;

public Claim submitClaim(Long partnerId, Long policyId, String reason) {

Partner partner = partnerRepository.findById(partnerId)

.orElseThrow(() -> new RuntimeException("Partner not found"));

Claim claim = new Claim();

claim.setPartner(partner);

claim.setPolicyId(policyId);

claim.setReason(reason);

claim.setStatus(ClaimStatus.PENDING);

return claimRepository.save(claim);

}

public Claim approveClaim(Long claimId) {

Claim claim = claimRepository.findById(claimId)

.orElseThrow(() -> new RuntimeException("Claim not found"));

claim.setStatus(ClaimStatus.APPROVED);

return claimRepository.save(claim);

}

public List<Claim> getClaimsByPartner(Long partnerId) {

return claimRepository.findByPartner\_Id(partnerId);

}

public List<Claim> getPendingClaims() {

return claimRepository.findByStatus(ClaimStatus.PENDING);

}

public Map<String, Long> getClaimSummaryByPartner() {

return claimRepository.findAll().stream()

.collect(Collectors.groupingBy(

c -> c.getPartner().getName(),

Collectors.counting()

));

}

}

// ─────────────────────────────────────────────────────────────

// SUPPLIER SERVICE: supplier-service

// ─────────────────────────────────────────────────────────────

// Coupa or 3rd party data integration (mocked in code or via RestTemplate)

@Service

public class SupplierService {

public List<String> fetchApprovedSuppliers() {

return List.of("Supplier A", "Supplier B", "Supplier C");

}

}

// ─────────────────────────────────────────────────────────────

// GLOBAL EXCEPTION HANDLING

// ─────────────────────────────────────────────────────────────

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(RuntimeException.class)

public ResponseEntity<String> handleRuntimeException(RuntimeException ex) {

return ResponseEntity.status(HttpStatus.BAD\_REQUEST).body(ex.getMessage());

}

@ExceptionHandler(Exception.class)

public ResponseEntity<String> handleGenericException(Exception ex) {

return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body("Something went wrong");

}

}

// ─────────────────────────────────────────────────────────────

// STORED PROCEDURE (MYSQL EXAMPLE)

// ─────────────────────────────────────────────────────────────

-- Procedure: Fetch claims by status

DELIMITER //

CREATE PROCEDURE GetClaimsByStatus(IN status VARCHAR(20))

BEGIN

SELECT \* FROM claim WHERE status = status;

END //

DELIMITER ;

// Use it from JPA

public interface ClaimRepository extends JpaRepository<Claim, Long> {

@Procedure(name = "GetClaimsByStatus")

List<Claim> getClaimsByStatus(@Param("status") String status);

}

// Remaining services (API Gateway, Discovery, Config, Vault, React UI, Docker, K8s, Swagger) next...