Getting Started with Healthcare SOA Project

This guide will help you set up and run the Healthcare Information Exchange SOA project from scratch.

Prerequisites

- Java 8+
- Maven 3.6+
- Docker and Docker Compose

- Git
- MuleSoft Runtime 3.8.0
- PostgreSQL 13+, MongoDB 4.4+, and Redis 6.2+ (containerized via Docker)

Initial Setup

1. **Clone the repository**

```
git clone https://github.com/your-org/healthcare-soa.git
cd healthcare-soa
```

2. **Set up environment variables**

Create a `.env` file in the project root with the following variables:

```
# Database Configuration
DB_HOST=postgres
DB_PORT=5432
DB_USER=postgres
DB_PASSWORD=postgres123456
DB_NAME=healthcaredb

# Enterprise DB Adaptation Settings
DB_USE_LEGACY_SCHEMA=true
DB_FEATURE_FLAGS_ENABLED=true

# MongoDB Configuration (Used by Appointment Service)
MONGO_HOST=mongodb
MONGO_PORT=27017
MONGO_USER=mongouser
MONGO_PASSWORD=mongopassword
MONGO_DB_NAME=appointmentdb

# Redis Configuration (Used by Appointment Service)
REDIS_HOST=redis
REDIS_PORT=6379
REDIS_PASSWORD=redispassword

# Kafka Configuration
KAFKA_BOOTSTRAP_SERVERS=localhost:9092

# MuleSoft ESB Configuration
MULE_ENV=local
MULE_VERSION=3.8.0

# Security
```

```
JWT_SECRET=your-jwt-secret-key
JWT_EXPIRATION_MS=86400000
```

3. **Start infrastructure services with Docker Compose**

```
docker-compose up -d
```

Project Structure

The project follows a modular structure with each service in its own directory:

```
healthcare-soa/
├── esb/                          # MuleSoft ESB configuration
│   ├── apps/                      # Mule applications
│   │   └── healthcare-integration-app/ # Main integration application
│   └── domains/                   # Shared domain configurations
│       └── default/               # Default domain
├── services/                      # Microservices
│   ├── patient-service/           # Patient management service
│   └── appointment-service/       # Appointment scheduling service
├── common/                       # Shared libraries and utilities
├── init-scripts/                 # Database initialization scripts
├── deployment/                   # Deployment scripts and configurations
│   ├── docker/                   # Docker configurations
│   └── kubernetes/               # K8s manifests
└── docs/                         # Project documentation
```

Building Services

To build all services:

```
./mvnw clean install
```

To build a specific service:

```
cd patient-service
../mvnw clean install
```

Running Locally

1. **Start infrastructure services**

```
docker-compose up -d postgres
```

2. **Start the ESB**

```
cd esb
../mvnw spring-boot:run
```

3. **Start individual services**

In separate terminals:

```
# Start the ESB (MuleSoft)
docker-compose up -d esb

# Start Patient Service
cd services/patient-service
./mvnw spring-boot:run -Dspring.profiles.active=local

# Start Appointment Service
cd services/appointment-service
./mvnw spring-boot:run -Dspring.profiles.active=local
```

Service Implementation Steps

1. Define Service Interface

Each service should define a clear interface in its API module:

```
// patient-service/patient-
api/src/main/java/com/healthcare/patient/api/PatientService.java
package com.healthcare.patient.api;

import java.util.List;
import java.util.Optional;
```

```
public interface PatientService {
    Patient createPatient(Patient patient);
    Optional<Patient> getPatientById(String id);
    List<Patient> findPatientsByName(String name);
    Patient updatePatient(String id, Patient patient);
    void deletePatient(String id);
}
```

2. Implement Domain Model

Create JPA entities and DTOs:

```
// services/patient-service/src/main/java/com/healthcare/patient/model/Patient.java
@Entity
@Table(name = "patients")
public class Patient {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(columnDefinition = "serial")
    private Integer id;

    @Column(nullable = false, name = "first_name")
    private String firstName;

    @Column(nullable = false, name = "last_name")
    private String lastName;

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

    @Column(name = "date_of_birth")
    private LocalDate dateOfBirth;

    // Additional fields, getters, setters
}
```

3. Implement Repository Layer

Create Spring Data repositories:

```
// services/patient-
service/src/main/java/com/healthcare/patient/repository/PatientRepository.java
@Repository
public interface PatientRepository extends JpaRepository<Patient, Integer> {
    List<Patient> findByLastNameContainingIgnoreCase(String lastName);
    Optional<Patient> findByMrn(String mrn);
}
```

4. Implement Service Layer

Implement business logic:

```
// patient-service/patient-
service/src/main/java/com/healthcare/patient/service/PatientServiceImpl.java
@Service
@Transactional
public class PatientServiceImpl implements PatientService {
    private final PatientRepository patientRepository;
    private final PatientMapper patientMapper;

    // Constructor injection

    @Override
    public Patient createPatient(Patient patient) {
        validatePatient(patient);
        PatientEntity entity = patientMapper.toEntity(patient);
        PatientEntity saved = patientRepository.save(entity);
        return patientMapper.toDto(saved);
    }

    // Other methods
}
```

5. Implement REST Controllers

Create RESTful endpoints:

```
// services/patient-
service/src/main/java/com/healthcare/patient/controller/PatientController.java
@RestController
@RequestMapping("/api/patients")
public class PatientController {
    private final PatientService patientService;

    @Autowired
    public PatientController(PatientService patientService) {
        this.patientService = patientService;
    }

    @PostMapping
    public ResponseEntity<Patient> createPatient(@Valid @RequestBody Patient
patient) {
        Patient created = patientService.createPatient(patient);
        return ResponseEntity
            .created(URI.create("/api/patients/" + created.getId()))
            .body(created);
    }

    @GetMapping("/{id}")
    public ResponseEntity<Patient> getPatientById(@PathVariable Integer id) {
        return patientService.getPatientById(id)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }

    // Other endpoints
}
```

6. Configure Messaging

Implement event publishing with Kafka:

```
// services/appointment-
service/src/main/java/com/healthcare/appointment/event/AppointmentEventPublisher.ja
va
@Component
public class AppointmentEventPublisher {
    private final RedisTemplate<String, Object> redisTemplate;

    @Autowired
    public AppointmentEventPublisher(RedisTemplate<String, Object> redisTemplate) {
        this.redisTemplate = redisTemplate;
    }

    public void publishAppointmentCreated(Appointment appointment) {
        AppointmentEvent event = new AppointmentEvent(
            "APPOINTMENT_CREATED",
            appointment.getId(),
            LocalDateTime.now(),
            appointment
        );

        // Publish to Redis for real-time updates
        redisTemplate.convertAndSend("appointment-events", event);

        // Store event in MongoDB for historical purposes
        // This will be handled by an event listener
    }
}
```

Testing

Unit Testing

```
./mvnw test
```

Integration Testing

```
./mvnw verify
```

End-to-End Testing

```
cd e2e-tests
../mvnw test -Pe2e
```

Deployment

Local Kubernetes Deployment

```
# Apply configuration
kubectl apply -f deployment/kubernetes/config-maps.yaml
kubectl apply -f deployment/kubernetes/secrets.yaml

# Deploy services
kubectl apply -f deployment/kubernetes/services/
```

AWS Deployment

```
# Configure AWS credentials
aws configure

# Deploy with Terraform
cd deployment/terraform
terraform init
terraform apply
```

Monitoring

- Access Prometheus: http://localhost:9090
- Access Grafana: http://localhost:3000
- Access Kibana: http://localhost:5601

Documentation

- API documentation: http://localhost:8080/swagger-ui.html
- Admin dashboard: http://localhost:8080/admin

Next Steps

1. Enhance the enterprise database adaptation strategy
2. Implement more comprehensive appointment features in Redis and MongoDB
3. Extend MuleSoft ESB with additional healthcare integration flows
4. Add FHIR compliance to both services
5. Implement security with OAuth2
6. Set up CI/CD pipeline with automated testing
7. Add comprehensive monitoring and alerting