Seal Wallet - Mini Money Transfer Application

Technical Presentation

Presented by: [Your Name]
Date: January 15, 2026

Slide 1: Project Overview

Seal Wallet Application

What I Built

- Mini money transfer application
- User registration and login
- Wallet balance management
- Money transfer between users
- Transaction history tracking

Technologies Used

- **Java 21** with **Spring Boot 3.5.9**
- **PostgreSQL** database
- **Docker** for containerization
- **JWT** for security

Slide 2: Database Design

Database Tables

Main Tables

- **users** - Store user information (phone, password)
- **wallets** - Store wallet balance for each user
- **transactions** - Record all money transfers
- **refresh_tokens** - Manage user sessions
- **otps** - For future OTP features

Relationships

- Each user has one wallet
- Each wallet can have many transactions
- Users can have multiple refresh tokens

Slide 3: Application Features

Core Functionality

User Management

- Register with phone number and password
- Login to get access tokens
- Secure logout

Wallet Operations

- Check wallet balance
- View wallet status

Money Transfer

- Send money to other users by phone number
- View transaction history
- See sent and received transactions separately

Slide 4: Security Features

Authentication & Security

JWT Token System

- **Access tokens** (15 minutes) - for API calls
- **Refresh tokens** (7 days) - to get new access tokens
- **Token rotation** - new tokens on each refresh

Password Security

- **BCrypt hashing** - passwords are encrypted
- **Phone-based login** - no usernames needed

Transaction Security

- **Balance validation** before transfer
- **Atomic transactions** - either complete success or complete failure

---

Slide 5: Technology Stack

What I Used and Why

Component	Technology	Version	Reason
Language	Java	21	Latest stable version
Framework	Spring Boot	3.5.9	Easy development
Database	PostgreSQL	11.22	Reliable for banking
Security	JWT + BCrypt	Latest	Industry standard

Build	Maven	3.9+	Standard build tool

Key Libraries

- **Spring Security** - Authentication
- **Spring Data JPA** - Database operations
- **HikariCP** - Database connections

Slide 6: Containerization

Docker Implementation

Why Docker?

- **Consistency** - runs the same everywhere
- **Easy deployment** - just run containers
- **Isolation** - app runs separately from system

What I Containerized

- **Application container** - Spring Boot app
- **Database container** - PostgreSQL
- **Docker Compose** - runs both together

Benefits

- Easy to start and stop
- No need to install Java or PostgreSQL locally
- Same environment for development and production

Slide 7: Multi-Container Setup

Docker Compose Architecture

Container Structure

```
Application Container (Port 8080)
    ↓ connects to
Database Container (Port 5434)
```

How They Work Together

- App container talks to database container
- Database stores all data persistently
- Both containers start together with one command
- App waits for database to be ready before starting

Slide 8: Live Demo

Working Application

Demo Steps

1. Start containers with Docker Compose
2. Login with test user (phone: 1234567890)
3. Check wallet balance (starts with 1000.00)
4. Transfer money to another user
5. Check transaction history
6. Show updated balances

Test Data Available

- **User 1:** Phone 1234567890, Balance 1000.00
- **User 2:** Phone 0987654321, Balance 1000.00

Slide 9: CI/CD Plan

Automation Strategy

Current Status

- Application is working and containerized
- Ready for automated deployment

Planned CI/CD Pipeline

1. **Jenkins** - Automation server
2. **SonarQube** - Code quality checking
3. **Automated building** - Build app when code changes
4. **Automated testing** - Run tests automatically
5. **Automated deployment** - Deploy to containers

Simple Pipeline Flow

```
Code Change → Jenkins → Build → Test → Deploy
```

Slide 10: Jenkins Setup Plan

Build Automation

What Jenkins Will Do

- **Watch for code changes** in Git
- **Build the application** automatically
- **Run tests** to make sure everything works
- **Create Docker images** for deployment
- **Deploy to containers** if everything passes

Benefits

- **No manual work** - everything happens automatically
- **Consistent builds** - same process every time
- **Quick feedback** - know immediately if something breaks

Slide 11: SonarQube Integration

Code Quality Checking

What SonarQube Checks

- **Code bugs** - find errors in code
- **Security issues** - find potential vulnerabilities
- **Code quality** - check if code is well written
- **Test coverage** - how much code is tested

Integration with Pipeline

- Jenkins runs SonarQube after building
- If code quality is poor, deployment stops
- Only good quality code gets deployed

Slide 12: Kubernetes Plan

Container Orchestration

Current State

- Application runs in Docker containers locally
- Works well for development and testing

Kubernetes Migration Plan

- **Minikube** - Local Kubernetes for learning
- **Container management** - Kubernetes manages containers
- **Multiple copies** - Run several app containers
- **Automatic restart** - If container fails, start new one

Why Kubernetes?

- **Better for production** - more reliable
- **Handles scaling** - can run more containers when busy
- **Industry standard** - used by most companies

Slide 13: Deployment Architecture

High-Level System Design

Current Setup (Docker)

```
Developer Machine
├── Application Container
├── Database Container
└── Docker Compose manages both
```

Future Setup (Kubernetes)

```
Kubernetes Cluster
├── Application Pods (multiple copies)
├── Database Service
├── Load Balancer (distributes traffic)
└── Kubernetes manages everything
```

Slide 14: Implementation Timeline

Next Steps

Phase 1: CI/CD (2 weeks)

- Set up Jenkins container
- Configure basic build pipeline
- Integrate SonarQube for code quality
- Test automated deployment

Phase 2: Kubernetes (2 weeks)

- Install Minikube locally
- Create basic Kubernetes configurations
- Deploy application to Kubernetes
- Test container management features

Phase 3: Integration (1 week)

- Connect Jenkins to Kubernetes
- Automated deployment to Kubernetes
- Basic monitoring setup

Slide 15: Current Status

What’s Ready Now

Completed

- **Working application** with all features
- **Database design** with proper relationships
- **Security implementation** with JWT tokens
- **Docker containerization** with multi-container setup
- **Documentation** and testing approach

In Progress

- CI/CD pipeline planning
- Kubernetes learning and setup
- Production deployment strategy

Next Steps

- Implement Jenkins automation
- Set up code quality checking
- Learn Kubernetes basics
- Create deployment pipeline

Slide 16: Thank You

Questions & Discussion

Key Achievements

- **Production-ready application** with banking features
- **Secure authentication** with modern JWT approach
- **Containerized deployment** ready for scaling
- **Clear roadmap** for enterprise deployment

Ready to Demonstrate

- Live application demo
- Docker container management
- Database operations
- Security features

Thank you for your time!

Technical Assessment Presentation
January 15, 2026