HireLink

Implementation Documentation

Developer Guide & Technical Reference

Version 1.0
Graduate Academic Project
January 2025

Document Control

Document Title	HireLink Implementation Documentation
Version	1.0
Technology Stack	React, Spring Boot (Java), MySQL
Target Audience	Development Team, Technical Reviewers, Academic Evaluators

Table of Contents

1. Introduction

This Implementation Documentation provides a comprehensive guide for developers working on the HireLink platform. It covers development environment setup, coding standards, implementation details, and deployment procedures.

1.1 Purpose

This document serves as the primary technical reference for:

- Setting up the development environment
- Understanding the codebase structure and architecture
- Following coding standards and best practices
- Implementing new features and bug fixes
- Testing and quality assurance procedures
- Deployment and maintenance guidelines

1.2 Technology Stack Summary

Layer	Technology	Version
Frontend	React with Vite	React 18.x, Vite 5.x
UI Framework	Tailwind CSS + Material UI	Tailwind 3.x, MUI 5.x
Backend	Spring Boot (Java)	Spring Boot 3.x, Java 17+
Database	MySQL	MySQL 8.0+
ORM	Spring Data JPA / Hibernate	Hibernate 6.x
Authentication	Spring Security + JWT	jjwt 0.11.x
Build Tools	Maven / npm	Maven 3.9+, npm 10+

2. Development Environment Setup

This section provides step-by-step instructions for setting up the development environment.

2.1 Prerequisites

Ensure the following software is installed on your development machine:

Software	Version	Download URL
Java JDK	17 or higher (LTS)	adoptium.net
Node.js	18.x or higher (LTS)	nodejs.org
MySQL Server	8.0+	dev.mysql.com
Maven	3.9+	maven.apache.org
Git	Latest	git-scm.com
IDE (Recommended)	IntelliJ IDEA / VS Code	jetbrains.com / code.visualstudio.com

2.2 Project Setup Steps

2.2.1 Clone Repository

```
# Clone the main repository
git clone https://github.com/your-org/hirelink.git
cd hirelink

# The repository structure:
hirelink/
├── backend/            # Spring Boot application
├── frontend/           # React application
├── database/           # SQL scripts and migrations
├── docs/               # Documentation
└── docker/             # Docker configurations
```

2.2.2 Database Setup

```
# Login to MySQL
mysql -u root -p

# Create database and user
CREATE DATABASE hirelink_db CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
CREATE USER 'hirelink_user'@'localhost' IDENTIFIED BY 'your_secure_password';
GRANT ALL PRIVILEGES ON hirelink_db.* TO 'hirelink_user'@'localhost';
FLUSH PRIVILEGES;

# Run schema initialization
mysql -u hirelink_user -p hirelink_db < database/schema.sql
```

2.2.3 Backend Setup

```
# Navigate to backend directory
cd backend

# Copy environment configuration
cp src/main/resources/application.properties.example \
   src/main/resources/application.properties

# Edit application.properties with your database credentials
# spring.datasource.url=jdbc:mysql://localhost:3306/hirelink_db
# spring.datasource.username=hirelink_user
```

```
# spring.datasource.password=your_secure_password

# Build and run
mvn clean install
mvn spring-boot:run

# Backend will start on http://localhost:8080
```

2.2.4 Frontend Setup

```
# Navigate to frontend directory
cd frontend

# Install dependencies
npm install

# Copy environment configuration
cp .env.example .env.local

# Edit .env.local
# VITE_API_BASE_URL=http://localhost:8080/api/v1
# VITE_GOOGLE_MAPS_API_KEY=your_google_maps_key
# VITE_RAZORPAY_KEY_ID=your_razorpay_key

# Start development server
npm run dev

# Frontend will start on http://localhost:5173
```

3. Project Structure

This section details the organization of the codebase for both backend and frontend.

3.1 Backend Structure (Spring Boot)

```
backend/
├── src/
│   ├── main/
│   │   ├── java/com/hirelink/
│   │   │   ├── HireLinkApplication.java      # Main entry point
│   │   │   ├── config/                       # Configuration classes
│   │   │   │   ├── SecurityConfig.java       # Spring Security setup
│   │   │   │   ├── JwtConfig.java            # JWT token configuration
│   │   │   │   ├── CorsConfig.java           # CORS settings
│   │   │   │   └── SwaggerConfig.java        # API documentation
│   │   │   ├── controller/                   # REST API controllers
│   │   │   │   ├── AuthController.java       # Authentication endpoints
│   │   │   │   ├── UserController.java       # User management
│   │   │   │   ├── ProviderController.java   # Provider operations
│   │   │   │   ├── BookingController.java    # Booking management
│   │   │   │   ├── PaymentController.java    # Payment processing
│   │   │   │   └── ReviewController.java     # Reviews and ratings
│   │   │   ├── service/                      # Business logic layer
│   │   │   │   ├── AuthService.java          # Authentication logic
│   │   │   │   ├── UserService.java          # User operations
│   │   │   │   ├── ProviderService.java      # Provider operations
│   │   │   │   ├── BookingService.java       # Booking logic
│   │   │   │   ├── PaymentService.java       # Payment integration
│   │   │   │   ├── OtpService.java           # OTP generation/verification
│   │   │   │   ├── NotificationService.java  # SMS/Push notifications
│   │   │   │   └── AIMatchingService.java    # AI recommendation engine
│   │   │   ├── repository/                   # Data access layer
│   │   │   │   ├── UserRepository.java
│   │   │   │   ├── ProviderRepository.java
│   │   │   │   ├── BookingRepository.java
│   │   │   │   └── ReviewRepository.java
│   │   │   ├── entity/                       # JPA entities
│   │   │   │   ├── User.java
│   │   │   │   ├── ServiceProvider.java
│   │   │   │   ├── Service.java
│   │   │   │   ├── Booking.java
│   │   │   │   ├── Payment.java
│   │   │   │   └── Review.java
│   │   │   ├── dto/                          # Data Transfer Objects
│   │   │   │   ├── request/                  # Request DTOs
│   │   │   │   └── response/                 # Response DTOs
│   │   │   ├── exception/                    # Custom exceptions
│   │   │   │   ├── GlobalExceptionHandler.java
│   │   │   │   ├── ResourceNotFoundException.java
│   │   │   │   └── ValidationException.java
│   │   │   └── util/                         # Utility classes
│   │   │       ├── JwtUtil.java
│   │   │       └── ValidationUtil.java
│   │   └── resources/
│   │       ├── application.properties        # Configuration
```

```
│   │       └── messages.properties          # i18n messages
│   └── test/                                # Unit and integration tests
└── pom.xml                                  # Maven configuration
```

3.2 Frontend Structure (React)

```
frontend/
├── public/                          # Static assets
├── src/
│   ├── main.jsx                     # Application entry
│   ├── App.jsx                      # Root component
│   ├── components/                  # Reusable components
│   │   ├── common/                  # Shared components
│   │   │   ├── Button.jsx
│   │   │   ├── Input.jsx
│   │   │   ├── Modal.jsx
│   │   │   ├── Card.jsx
│   │   │   └── Loader.jsx
│   │   ├── layout/                  # Layout components
│   │   │   ├── Header.jsx
│   │   │   ├── Footer.jsx
│   │   │   ├── Sidebar.jsx
│   │   │   └── Navbar.jsx
│   │   ├── auth/                    # Authentication components
│   │   │   ├── LoginForm.jsx
│   │   │   ├── RegisterForm.jsx
│   │   │   └── OtpVerification.jsx
│   │   ├── provider/                # Provider-related components
│   │   │   ├── ProviderCard.jsx
│   │   │   ├── ProviderProfile.jsx
│   │   │   └── ProviderDashboard.jsx
│   │   ├── booking/                 # Booking components
│   │   │   ├── BookingForm.jsx
│   │   │   ├── BookingCard.jsx
│   │   │   └── BookingHistory.jsx
│   │   └── review/                  # Review components
│   │       ├── ReviewForm.jsx
│   │       └── ReviewList.jsx
│   ├── pages/                       # Page components
│   │   ├── Home.jsx
│   │   ├── Search.jsx
│   │   ├── ProviderDetails.jsx
│   │   ├── Booking.jsx
│   │   ├── UserDashboard.jsx
│   │   ├── ProviderDashboard.jsx
│   │   └── AdminDashboard.jsx
│   ├── hooks/                       # Custom React hooks
│   │   ├── useAuth.js
│   │   ├── useApi.js
│   │   ├── useGeolocation.js
│   │   └── useDebounce.js
│   ├── context/                     # React Context providers
│   │   ├── AuthContext.jsx
│   │   ├── ThemeContext.jsx
│   │   └── NotificationContext.jsx
│   ├── services/                    # API service modules
│   │   ├── api.js                   # Axios instance
```

```
│   │   ├── authService.js
│   │   ├── providerService.js
│   │   ├── bookingService.js
│   │   └── paymentService.js
│   ├── utils/                      # Utility functions
│   │   ├── validators.js
│   │   ├── formatters.js
│   │   └── constants.js
│   └── styles/                     # Global styles
│       └── globals.css
├── tailwind.config.js              # Tailwind configuration
├── vite.config.js                  # Vite configuration
└── package.json                    # npm configuration
```

4. Coding Standards

All developers must follow these coding standards to maintain code quality and consistency.

4.1 Java/Spring Boot Standards

4.1.1 Naming Conventions

Element	Convention	Example
Classes	PascalCase, nouns	UserService, BookingController
Methods	camelCase, verbs	createBooking(), findByEmail()
Variables	camelCase, descriptive	userId, providerList
Constants	UPPER_SNAKE_CASE	MAX_RETRY_COUNT, API_VERSION
Packages	lowercase, domain-based	com.hirelink.service

4.1.2 Code Structure Guidelines

1. Keep methods under 30 lines; extract complex logic into helper methods
2. Use constructor injection for dependencies (not field injection)
3. Follow Single Responsibility Principle - one class, one purpose
4. Use DTOs for API requests/responses; never expose entities directly
5. Add JavaDoc comments for all public methods and classes
6. Use Optional for potentially null return values

4.2 React/JavaScript Standards

4.2.1 Component Guidelines

1. Use functional components with hooks (no class components)
2. Component files: PascalCase (UserProfile.jsx)
3. Utility files: camelCase (formatDate.js)
4. Keep components under 200 lines; split if larger
5. Use PropTypes or TypeScript for prop validation
6. Extract reusable logic into custom hooks

4.2.2 State Management

- Use useState for local component state
- Use useContext for shared app-wide state (auth, theme)
- Consider React Query for server state management
- Avoid prop drilling beyond 2 levels; use Context instead

5. Database Implementation

This section covers database design decisions and implementation details.

5.1 Entity Relationship Overview

The database follows a normalized relational design with the following key relationships:

- User (1:1) ServiceProvider - A user can optionally be a service provider
- ServiceProvider (1:N) Services - A provider offers multiple services
- User (1:N) Bookings - A user can have multiple bookings
- Booking (1:1) Payment - Each booking has one payment record
- Booking (1:1) Review - Each completed booking can have one review

5.2 Key Table Implementations

5.2.1 Users Table

```
CREATE TABLE users (
    user_id BIGINT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    phone VARCHAR(15) NOT NULL UNIQUE,
    email VARCHAR(100) UNIQUE,
    password_hash VARCHAR(255),
    user_type ENUM('CUSTOMER', 'PROVIDER', 'ADMIN') NOT NULL DEFAULT 'CUSTOMER',
    profile_image_url VARCHAR(500),
    is_verified BOOLEAN DEFAULT FALSE,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    INDEX idx_phone (phone),
    INDEX idx_user_type (user_type)
);
```

5.2.2 Service Providers Table

```
CREATE TABLE service_providers (
    provider_id BIGINT PRIMARY KEY AUTO_INCREMENT,
    user_id BIGINT NOT NULL UNIQUE,
    business_name VARCHAR(200),
    bio TEXT,
    experience_years INT DEFAULT 0,
    latitude DECIMAL(10, 8),
    longitude DECIMAL(11, 8),
    service_radius_km INT DEFAULT 10,
    aadhaar_number_encrypted VARCHAR(500),
    pan_number_encrypted VARCHAR(500),
    is_kyc_verified BOOLEAN DEFAULT FALSE,
    avg_rating DECIMAL(2, 1) DEFAULT 0.0,
    total_reviews INT DEFAULT 0,
    total_bookings INT DEFAULT 0,
    is_available BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    INDEX idx_location (latitude, longitude),
    INDEX idx_availability (is_available, is_kyc_verified)
);
```

5.2.3 Bookings Table

```
CREATE TABLE bookings (
    booking_id BIGINT PRIMARY KEY AUTO_INCREMENT,
    user_id BIGINT NOT NULL,
    provider_id BIGINT NOT NULL,
    service_id BIGINT NOT NULL,
    scheduled_date DATE NOT NULL,
    scheduled_time TIME NOT NULL,
    address TEXT NOT NULL,
    pincode VARCHAR(6) NOT NULL,
    latitude DECIMAL(10, 8),
    longitude DECIMAL(11, 8),
    issue_description TEXT,
    status ENUM('PENDING', 'ACCEPTED', 'REJECTED', 'IN_PROGRESS',
                'COMPLETED', 'CANCELLED', 'DISPUTED') DEFAULT 'PENDING',
    estimated_amount DECIMAL(10, 2),
    final_amount DECIMAL(10, 2),
    cancellation_reason TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    FOREIGN KEY (provider_id) REFERENCES service_providers(provider_id),
    FOREIGN KEY (service_id) REFERENCES services(service_id),
    INDEX idx_user_bookings (user_id, status),
    INDEX idx_provider_bookings (provider_id, status),
    INDEX idx_scheduled (scheduled_date, scheduled_time)
);
```

5.3 Database Optimization

1. Indexes on frequently queried columns (phone, location, status)
2. Composite indexes for common query patterns
3. Use ENUM types for fixed-value columns to save storage
4. Encrypt sensitive data (Aadhaar, PAN) at rest using AES-256
5. Use connection pooling (HikariCP) for performance

6. API Implementation

This section covers RESTful API design and implementation guidelines.

6.1 API Design Principles

- Use RESTful conventions with appropriate HTTP methods
- Version all APIs (e.g., /api/v1/users)
- Use plural nouns for resource endpoints
- Return appropriate HTTP status codes
- Include HATEOAS links where beneficial
- Implement proper pagination for list endpoints

6.2 Authentication Flow

6.2.1 JWT Token Structure

```
// JWT Token Payload
{
  "sub": "user_123",           // User ID
  "name": "John Doe",          // User name
  "role": "CUSTOMER",          // User role
  "iat": 1704067200,           // Issued at
  "exp": 1704153600            // Expiry (24 hours)
}

// Token stored in Authorization header
Authorization: Bearer eyJhbGciOiJIUzUxMiJ9...
```

6.2.2 API Response Format

```
// Success Response
{
  "success": true,
  "message": "Operation successful",
  "data": { ... },
  "timestamp": "2025-01-01T12:00:00Z"
}

// Error Response
{
  "success": false,
  "message": "Validation failed",
  "errors": [
    { "field": "phone", "message": "Invalid phone format" }
  ],
  "timestamp": "2025-01-01T12:00:00Z"
}
```

6.3 Key API Endpoints

Method	Endpoint	Description
POST	`/api/v1/auth/register`	Register new user
POST	`/api/v1/auth/send-otp`	Send OTP to phone
POST	`/api/v1/auth/verify-otp`	Verify OTP and login
GET	`/api/v1/providers`	Search providers with filters
GET	`/api/v1/providers/nearby`	Get nearby providers by location
POST	`/api/v1/bookings`	Create new booking

PATCH	/api/v1/bookings/{id}/status	Update booking status
POST	/api/v1/payments/initiate	Initiate Razorpay payment
POST	/api/v1/reviews	Submit review for booking

7. Security Implementation

Security is paramount for the HireLink platform, especially handling sensitive user data and payments.

7.1 Authentication Security

Measure	Implementation
Password Hashing	BCrypt with cost factor 12 (Spring Security default)
JWT Signing	HS512 algorithm with 512-bit secret key
Token Expiry	Access token: 24 hours, Refresh token: 7 days
OTP Security	6-digit random, 5-minute expiry, max 3 attempts, rate limited
Session Management	Stateless JWT, token blacklist for logout

7.2 Data Protection

1. HTTPS/TLS 1.3 for all communications
2. AES-256 encryption for sensitive data at rest (Aadhaar, PAN)
3. Database credentials stored in environment variables, never in code
4. Input validation and sanitization on all endpoints
5. Parameterized queries to prevent SQL injection
6. XSS protection via Content Security Policy headers

7.3 Rate Limiting

```
// Rate Limiting Configuration (per IP/user)
Authentication endpoints:  5 requests/minute
OTP requests:              3 requests/5 minutes
API endpoints:             100 requests/minute
Search endpoints:          30 requests/minute
File uploads:              10 requests/hour
```

7.4 CORS Configuration

```
@Configuration
public class CorsConfig {
    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration config = new CorsConfiguration();
        config.setAllowedOrigins(Arrays.asList(
            "https://hirelink.com",
            "https://app.hirelink.com",
            "http://localhost:5173"  // Dev only
        ));
        config.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "PATCH",
"DELETE"));
        config.setAllowedHeaders(Arrays.asList("Authorization", "Content-Type"));
        config.setAllowCredentials(true);
        config.setMaxAge(3600L);
        // ... register configuration
    }
}
```

8. Testing Strategy

A comprehensive testing strategy ensures code quality and reliability.

8.1 Testing Pyramid

Test Type	Coverage Target	Tools
Unit Tests	80%+	JUnit 5, Mockito (Backend); Jest, React Testing Library (Frontend)
Integration Tests	70%+	Spring Boot Test, TestContainers, H2 Database
API Tests	100% endpoints	Postman/Newman, RestAssured
E2E Tests	Critical paths	Cypress, Playwright

8.2 Example Test Cases

8.2.1 Backend Unit Test

```
@ExtendWith(MockitoExtension.class)
class BookingServiceTest {
    @Mock private BookingRepository bookingRepository;
    @Mock private ProviderRepository providerRepository;
    @InjectMocks private BookingService bookingService;

    @Test
    void createBooking_ValidRequest_ReturnsBookingId() {
        // Arrange
        BookingRequest request = createValidBookingRequest();

when(providerRepository.findById(1L)).thenReturn(Optional.of(mockProvider));
        when(bookingRepository.save(any())).thenReturn(mockBooking);

        // Act
        BookingResponse response = bookingService.createBooking(request);

        // Assert
        assertNotNull(response.getBookingId());
        assertEquals(BookingStatus.PENDING, response.getStatus());
        verify(bookingRepository, times(1)).save(any());
    }
}
```

8.2.2 Frontend Component Test

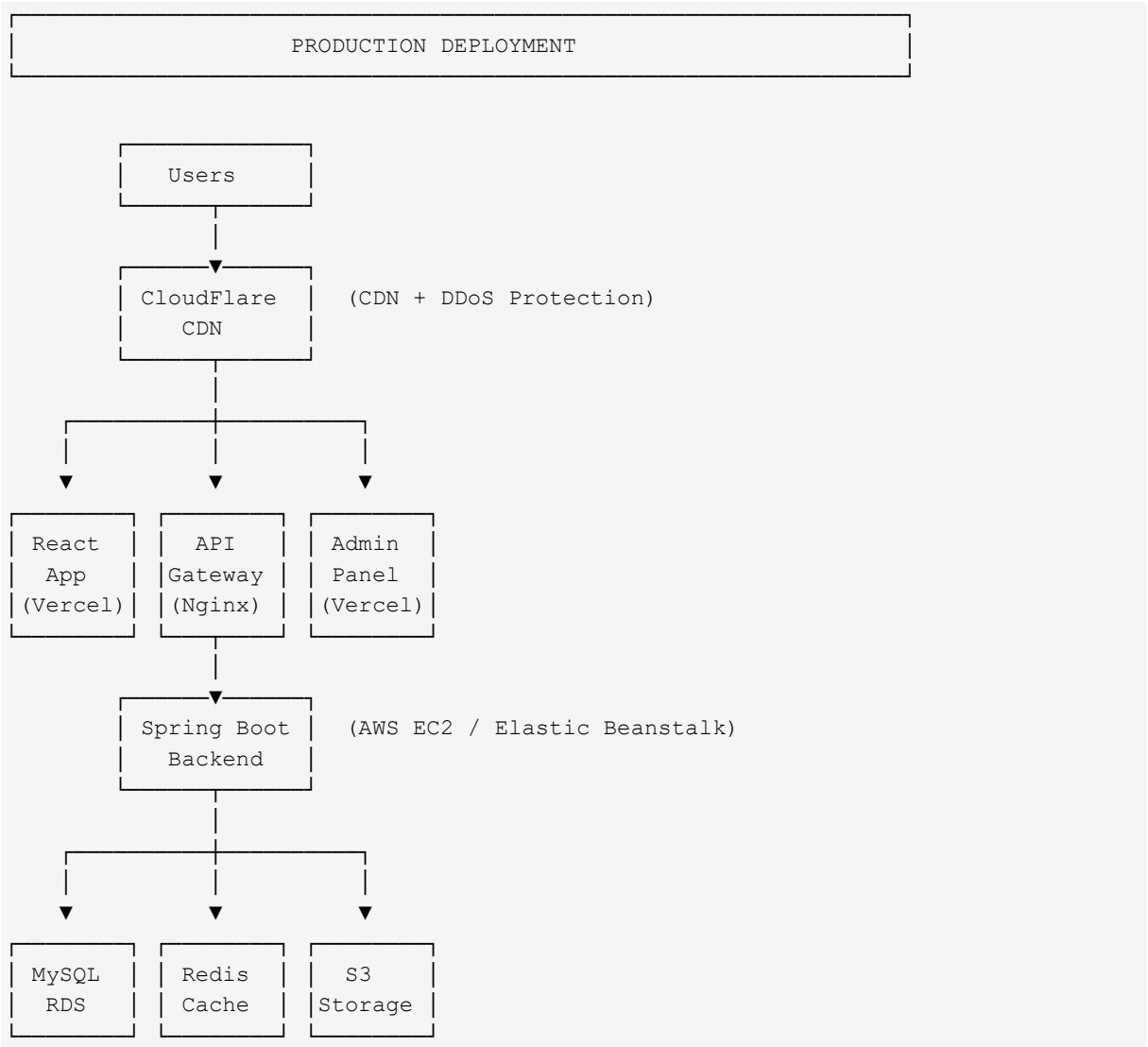
```
import { render, screen, fireEvent } from '@testing-library/react';
import { LoginForm } from './LoginForm';

describe('LoginForm', () => {
  it('validates phone number format', async () => {
    render(<LoginForm onSubmit={jest.fn()} />);

    const phoneInput = screen.getByLabelText(/phone/i);
    fireEvent.change(phoneInput, { target: { value: '123' } });
    fireEvent.blur(phoneInput);

    expect(await screen.findByText(/invalid phone/i)).toBeInTheDocument();
  });

  it('submits form with valid data', async () => {
```

```
    const mockSubmit = jest.fn();
    render(<LoginForm onSubmit={mockSubmit} />);

    fireEvent.change(screen.getByLabelText(/phone/i),
      { target: { value: '9876543210' } });
    fireEvent.click(screen.getByRole('button', { name: /send otp/i }));

    expect(mockSubmit).toHaveBeenCalledWith({ phone: '9876543210' });
  });
});
```

9. Deployment Guide

This section covers the deployment process for production environments.

9.1 Deployment Architecture



9.2 Deployment Checklist

1. Run all tests and ensure 100% pass rate
2. Update version numbers in pom.xml and package.json
3. Build production artifacts (mvn package -Pprod, npm run build)
4. Run database migrations if any schema changes
5. Update environment variables in production
6. Deploy backend to AWS/Heroku with zero-downtime strategy
7. Deploy frontend to Vercel/Netlify
8. Verify health endpoints and run smoke tests
9. Monitor logs and metrics for first 24 hours

9.3 Environment Variables

Variable	Description

DATABASE_URL	MySQL connection string
JWT_SECRET	Secret key for JWT signing (512-bit)
TWILIO_ACCOUNT_SID	Twilio account for SMS OTP
RAZORPAY_KEY_ID	Razorpay API key ID
RAZORPAY_KEY_SECRET	Razorpay API secret
AWS_S3_BUCKET	S3 bucket for file storage
GOOGLE_MAPS_API_KEY	Google Maps API for location services

10. Maintenance & Support

Guidelines for ongoing maintenance and support of the HireLink platform.

10.1 Monitoring & Logging

- Application logs: Centralized logging with ELK Stack (Elasticsearch, Logstash, Kibana)
- Performance monitoring: Application Performance Monitoring with New Relic or Datadog
- Error tracking: Real-time error alerts with Sentry
- Infrastructure: AWS CloudWatch for EC2/RDS metrics
- Uptime monitoring: Pingdom or UptimeRobot for endpoint health

10.2 Backup Strategy

Data Type	Frequency	Retention
Database (Full)	Daily	30 days (AWS RDS automated backups)
Database (Incremental)	Every 5 minutes	Point-in-time recovery (RDS)
User Files (S3)	Real-time	Cross-region replication, 90-day versioning
Configuration	On change	Git version control (infinite)

10.3 Incident Response

1. Detection: Automated alerts trigger on error spikes, latency issues, or downtime
2. Triage: On-call engineer assesses severity (P1-Critical, P2-High, P3-Medium, P4-Low)
3. Response: Follow runbooks for common issues; escalate if needed
4. Resolution: Fix deployed, verified in production
5. Post-mortem: Document root cause and preventive measures for P1/P2 incidents

10.4 Support Escalation Matrix

Level	Issues Handled	Response Time	Team
L1 Support	User queries, basic troubleshooting	< 4 hours	Customer Support
L2 Support	Technical issues, bug reports	< 24 hours	Dev Team
L3 Support	Critical system issues	< 1 hour	Senior Engineers

Appendix A: Useful Commands

A.1 Development Commands

```
# Backend
mvn spring-boot:run              # Run development server
mvn test                          # Run unit tests
mvn verify                        # Run all tests including integration
mvn clean package -DskipTests    # Build JAR without tests

# Frontend
npm run dev                       # Start dev server
npm run build                     # Production build
npm run test                      # Run tests
npm run lint                      # Run ESLint
npm run preview                   # Preview production build

# Database
mvn flyway:migrate               # Run database migrations
mvn flyway:info                  # Show migration status
```

A.2 Docker Commands (Optional)

```
# Build and run entire stack
docker-compose up --build

# Run only database
docker-compose up mysql

# View logs
docker-compose logs -f backend

# Stop all services
docker-compose down
```

Document Revision History

Version	Date	Author	Changes
1.0	January 2025	Project Team	Initial implementation documentation