Vendor Management System

A comprehensive Spring Boot application for managing vendors, their services, bookings, availability, and customer reviews.

Features

- **Vendor Management** - Register and manage vendor profiles
- **Service Management** - Add and manage services offered by vendors
- **Booking System** - Handle customer bookings for services
- **Availability Tracking** - Manage vendor availability schedules
- **Review System** - Collect and display customer reviews
- **RESTful API** - Fully functional REST API endpoints

Tech Stack

Spring Boot 3.3.2 **MySQL 8.0** **Spring Data JPA** **Hibernate** **Maven** **Java 17**

Prerequisites

Before running this application, ensure you have:

- Java JDK 17 or higher
- MySQL Server 8.0+
- Maven 3.6+
- Postman (for API testing)

Quick Start

1. Clone the Repository

```
git clone <your-repository-url>
cd vendor-management
```

2. Database Setup

```
CREATE DATABASE vendor_db;
USE vendor_db;
```

The application will automatically create tables using Hibernate's `ddl-auto=update`.

3. Configuration

Update `src/main/resources/application.properties` with your database credentials:

```
spring.datasource.url=jdbc:mysql://localhost:3306/vendor_db
spring.datasource.username=your_username
spring.datasource.password=your_password
spring.jpa.hibernate.ddl-auto=update
```

```
spring.jpa.show-sql=true
```

4. Build and Run

```
# Clean and compile
mvn clean compile

# Run the application
mvn spring-boot:run
```

The application will start on `http://localhost:8080`

API Endpoints

Vendors

- `GET /api/vendors` - Get all vendors
- `POST /api/vendors/register` - Register a new vendor
- `PUT /api/vendors/{id}` - Update vendor details

Services

- `POST /api/services/{vendorId}` - Add a service to a vendor
- `GET /api/services/vendor/{vendorId}` - Get services by vendor

Sample Requests

Register a Vendor:

```
POST http://localhost:8080/api/vendors/register
Content-Type: application/json

{
    "name": "John's Plumbing",
    "email": "john@plumbing.com",
    "password": "password123",
    "profileImage": "john.jpg"
}
```

Add a Service:

```
POST http://localhost:8080/api/services/1
Content-Type: application/json

{
    "name": "Pipe Repair",
    "description": "Emergency pipe repair service",
    "price": 99.99,
    "pricingTier": "Standard"
```

```
}
```

Database Schema

The application uses the following main tables:

- `vendors` - Vendor information
- `services` - Services offered by vendors
- `bookings` - Customer bookings
- `availability` - Vendor availability schedules
- `reviews` - Customer reviews and ratings

Project Structure

```
vendor-management/
├── src/main/java/com/vendorapp/
│   ├── entity/              # JPA Entities
│   │   ├── Vendor.java
│   │   ├── ServiceEntity.java
│   │   ├── Booking.java
│   │   ├── Availability.java
│   │   └── Review.java
│   ├── repository/          # Data Access Layer
│   │   ├── VendorRepository.java
│   │   ├── ServiceRepository.java
│   │   ├── BookingRepository.java
│   │   ├── AvailabilityRepository.java
│   │   └── ReviewRepository.java
│   ├── service/             # Business Logic Layer
│   │   ├── VendorService.java
│   │   └── ServiceService.java
│   ├── controller/          # REST Controllers
│   │   ├── VendorController.java
│   │   └── ServiceController.java
│   └── VendorAppApplication.java # Main Application
├── src/main/resources/
│   └── application.properties    # Configuration
└── pom.xml                  # Maven Configuration
```

Configuration Options

Key configuration properties in `application.properties`:

```
 # Server port
server.port=8080

# Database settings
spring.datasource.url=jdbc:mysql://localhost:3306/vendor_db
spring.datasource.username=root
spring.datasource.password=your_password

# JPA settings
spring.jpa.hibernate.ddl-auto=update
```

```
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

# Logging
logging.level.com.vendorapp=DEBUG
```

Testing

Using Postman

1. Import the Postman collection from `/postman` folder
2. Start the Spring Boot application
3. Test all endpoints sequentially

Sample Test Data

```
 INSERT INTO vendors (name, email, password, profile_image)
VALUES
('John Plumbing', 'john@plumbing.com', 'password123', 'john.jpg'),
('Sarah Electrical', 'sarah@electrical.com', 'electro123', 'sarah.jpg');
```

Troubleshooting

Common Issues:

1. Database Connection Error

- Verify MySQL is running
- Check credentials in `application.properties`

2. Port Already in Use

- Change `server.port` in application.properties
- Or kill process using port 8080

3. Entity Not Found

- Check package declarations in entity classes
- Verify `@Entity` annotations

Logs

Check application logs for detailed error information:

```
tail -f logs/application.log
```

Future Enhancements

- [ ] Authentication & Authorization (JWT)
- [ ] Email notifications
- [ ] File upload for vendor images
- [ ] Pagination and filtering
- [ ] Advanced search functionality
- [ ] Dashboard and analytics
- [ ] Mobile app interface

Contributing

1. Fork the repository
2. Create a feature branch (`git checkout -b feature/amazing-feature`)
3. Commit your changes (`git commit -m 'Add amazing feature'`)
4. Push to the branch (`git push origin feature/amazing-feature`)
5. Open a Pull Request

License

This project is licensed under the MIT License - see the LICENSE.md file for details.

Acknowledgments

- Spring Boot team
- MySQL community
- Open source contributors

Happy Coding!

For any questions or support, please open an issue in the GitHub repository.