🚀 Inventory Management System - Complete Setup Guide

📋 Project Structure

```
inventory-management/
├── backend/                    # Spring Boot Backend
│   ├── src/
│   │   ├── main/
│   │   │   ├── java/com/inventory/
│   │   │   │   ├── InventoryManagementApplication.java
│   │   │   │   ├── model/
│   │   │   │   │   └── Product.java
│   │   │   │   ├── repository/
│   │   │   │   │   └── ProductRepository.java
│   │   │   │   ├── service/
│   │   │   │   │   └── ProductService.java
│   │   │   │   ├── controller/
│   │   │   │   │   └── ProductController.java
│   │   │   │   └── config/
│   │   │   │       └── CorsConfig.java
│   │   │   └── resources/
│   │   │       └── application.properties
│   │   └── test/
│   └── pom.xml
│
└── frontend/                   # React Frontend
    ├── src/
    │   ├── App.js
    │   ├── App.css
    │   └── index.js
    ├── public/
    │   └── index.html
    └── package.json
```

🛠️ Prerequisites

Required Software:

1. **Java JDK 17+** - Download
2. **Maven 3.6+** - Download

3. **Node.js 16+** - Download

4. **VS Code** - Download

5. **Git** - Download

Verify Installation:

```bash
java -version
mvn -version
node -version
npm -version
git --version
```

📦 Step 1: Setup Backend (Spring Boot)

1.1 Create Backend Project Structure

```bash
# Create main directory
mkdir inventory-management
cd inventory-management

# Create backend directory
mkdir backend
cd backend
```

1.2 Create Maven Project Structure

```bash
mkdir -p src/main/java/com/inventory/model
mkdir -p src/main/java/com/inventory/repository
mkdir -p src/main/java/com/inventory/service
mkdir -p src/main/java/com/inventory/controller
mkdir -p src/main/java/com/inventory/config
mkdir -p src/main/resources
mkdir -p src/test/java
```

1.3 Copy Backend Files

Copy these files into their respective directories:

- `pom.xml` → `backend/pom.xml`
- `application.properties` → `backend/src/main/resources/application.properties`
- All Java files from the first artifact to their respective packages

1.4 Build and Run Backend

```bash
cd backend

# Clean and build project
mvn clean install

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

Backend will run on: `http://localhost:8080`

1.5 Test Backend API

Open browser or use curl:

```bash
# Test health
curl http://localhost:8080/api/products

# View H2 Console (optional)
http://localhost:8080/h2-console
```

🎨 Step 2: Setup Frontend (React)

2.1 Create React App

```bash
```

```
# Go back to root directory
cd ..

# Create React app
npx create-react-app frontend
cd frontend
```

2.2 Install Dependencies

```bash
npm install axios
```

2.3 Copy Frontend Files

Replace these files:

- Copy `App.js` content → `frontend/src/App.js`
- Copy `App.css` content → `frontend/src/App.css`
- Update `package.json` with the provided version

2.4 Update index.js (if needed)

File: `frontend/src/index.js`

```javascript
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

2.5 Run Frontend

```bash
```

```
npm start
```

Frontend will run on: `http://localhost:3000`

🔗 Step 3: Connect Frontend to Backend

The frontend is already configured to connect to the backend through:

- Axios API calls to `http://localhost:8080/api/products`
- CORS is enabled in the backend

Test the Full Stack:

1. Keep backend running on terminal 1: `mvn spring-boot:run`
2. Keep frontend running on terminal 2: `npm start`
3. Open browser: `http://localhost:3000`
4. Try adding, editing, and deleting products!

🌐 Step 4: Deploy to GitHub

4.1 Initialize Git Repository

```bash
# In root directory (inventory-management)
cd ..
git init
```

4.2 Create .gitignore

Create `.gitignore` file:

```
# Backend
backend/target/
backend/.mvn/
backend/mvnw
backend/mvnw.cmd

# Frontend
frontend/node_modules/
frontend/build/
frontend/.env

# IDE
.vscode/
.idea/
*.iml

# OS
.DS_Store
Thumbs.db
```

4.3 Commit and Push

bash

```bash
git add .
git commit -m "Initial commit: Full-stack Inventory Management System"

# Create repository on GitHub first, then:
git remote add origin https://github.com/YOUR_USERNAME/inventory-management.git
git branch -M main
git push -u origin main
```

🚀 Step 5: Deploy Frontend to GitHub Pages

5.1 Install gh-pages

bash

```bash
cd frontend
npm install gh-pages --save-dev
```

5.2 Update package.json

Add these lines to `frontend/package.json`:

json

```json
{
  "homepage": "https://YOUR_USERNAME.github.io/inventory-management",
  "scripts": {
    "predeploy": "npm run build",
    "deploy": "gh-pages -d build",
    ...
  }
}
```

5.3 Deploy

bash

```bash
npm run deploy
```

Note: For production, you'll need to deploy backend separately (Heroku, Railway, AWS, etc.) and update the API URL in `App.js`.

🔧 Step 6: Deploy Backend (Options)

Option A: Deploy to Railway.app (Recommended - Free)

1. Create account at railway.app
2. Create new project
3. Deploy from GitHub
4. Add PostgreSQL database
5. Update `application.properties` for production

Option B: Deploy to Heroku

bash

```
# Install Heroku CLI
heroku login
heroku create your-app-name

# Deploy
git subtree push --prefix backend heroku main
```

Option C: Deploy to AWS/Azure/GCP

Use their Java deployment services (Elastic Beanstalk, App Service, Cloud Run)

📝 Configuration for Production

Update Frontend API URL

File: `frontend/src/App.js`

```javascript
// Change this line:
const API_BASE_URL = 'http://localhost:8080/api/products';

// To your deployed backend URL:
const API_BASE_URL = 'https://your-backend.railway.app/api/products';
```

Update Backend for Production Database

File: `backend/src/main/resources/application.properties`

```properties
# PostgreSQL Configuration
spring.datasource.url=${DATABASE_URL}
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
```

✅ Testing Your Application

Backend Tests:

```bash
cd backend
mvn test
```

Frontend Tests:

```bash
cd frontend
npm test
```

Manual Testing Checklist:

- [ ] Add a new product
- [ ] Edit an existing product
- [ ] Delete a product
- [ ] Search products by name/SKU
- [ ] Filter by stock status
- [ ] Export to CSV
- [ ] Check statistics update correctly
- [ ] Test low stock alerts

🎯 Features Implemented

Backend (Spring Boot + Java):

- ✅ RESTful API with CRUD operations
- ✅ JPA/Hibernate for database operations
- ✅ H2 in-memory database (dev) / PostgreSQL (production)
- ✅ CORS configuration for frontend connection
- ✅ Search and filter endpoints
- ✅ Statistics calculation
- ✅ Exception handling

Frontend (React):

- ✅ Modern, responsive UI
- ✅ Real-time statistics dashboard
- ✅ Product CRUD operations

✅ Search and filter functionality
✅ Export to CSV
✅ Modal dialogs for add/edit
✅ Status badges (In Stock, Low Stock, Out of Stock)
✅ Error handling and loading states

🐛 Troubleshooting

Backend Issues:

Port 8080 already in use:

```bash
# Windows
netstat -ano | findstr :8080
taskkill /PID <PID> /F

# Mac/Linux
lsof -ti:8080 | xargs kill -9
```

Maven build fails:

```bash
mvn clean install -U
```

Frontend Issues:

Cannot connect to backend:

- Ensure backend is running on port 8080
- Check CORS configuration in backend
- Verify API_BASE_URL in App.js

npm install errors:

```bash
rm -rf node_modules package-lock.json
npm install
```

📚 API Documentation

Endpoints:

```
GET    /api/products              - Get all products
GET    /api/products/{id}         - Get product by ID
POST   /api/products              - Create new product
PUT    /api/products/{id}         - Update product
DELETE /api/products/{id}         - Delete product
GET    /api/products/search?query={term} - Search products
GET    /api/products/low-stock    - Get low stock products
GET    /api/products/out-of-stock - Get out of stock products
GET    /api/products/statistics   - Get inventory statistics
```

🎓 Next Steps

1. Add user authentication (Spring Security + JWT)
2. Add product categories management
3. Add inventory history tracking
4. Add email notifications for low stock
5. Add barcode scanner integration
6. Add reports and analytics
7. Add multi-warehouse support

📞 Support

If you encounter issues:

1. Check console logs (browser and terminal)
2. Verify all dependencies are installed
3. Ensure ports 3000 and 8080 are available
4. Check firewall settings

🎉 Congratulations!

You now have a fully functional full-stack Inventory Management System with:

- ✅ Java Spring Boot backend
- ✅ React frontend
- ✅ Database integration
- ✅ Ready for deployment
- ✅ Professional UI/UX

Happy coding! 🚀