**Parcel App Documentation**

**Timea Funtíková**

**2025**

Table of contents

[Table of contents 2](#_Toc194963393)

[1. Introduction 4](#_Toc194963394)

[1.1 Parcel Management App 4](#_Toc194963395)

[1.2 Target audience 5](#_Toc194963396)

[1.3 Key Benefits 5](#_Toc194963397)

[1.4 Technologies used 5](#_Toc194963398)

[1.5 Prerequisites 5](#_Toc194963399)

[1.6 Backend dependencies 5](#_Toc194963400)

[1.7 Frontend dependencies 6](#_Toc194963401)

[2. System architecture 7](#_Toc194963402)

[2.1 Overview 7](#_Toc194963403)

[2.2 Data Flow Summary 7](#_Toc194963404)

[2.3 Component Responsibilities 8](#_Toc194963405)

[2.4 Architecture Diagram 8](#_Toc194963406)

[2.5 System Design Goals 9](#_Toc194963407)

[3. User Interface (Frontend) 10](#_Toc194963408)

[3.1 Overview 10](#_Toc194963409)

[3.2 Key UI ElementS 10](#_Toc194963410)

[3.3 User Experience Features 10](#_Toc194963411)

[3.4 Accessibility And Usability 10](#_Toc194963412)

[3.5 Screenshots 10](#_Toc194963413)

[4. Backend & Data Flow 14](#_Toc194963414)

[4.1 Overview 14](#_Toc194963415)

[4.2 Responsibilities 14](#_Toc194963416)

[4.3 REST API Endpoints 14](#_Toc194963417)

[4.4 Data Flow Description 14](#_Toc194963418)

[4.5 Validation Logic (Backend) 14](#_Toc194963419)

[5. File Storage & Schema 15](#_Toc194963420)

[5.1 Storage Location 15](#_Toc194963421)

[5.2 File Naming Convention 15](#_Toc194963422)

[5.3 JSON Schema 15](#_Toc194963423)

[6. Deployment & Distribution 16](#_Toc194963424)

[6.1 Development Mode 16](#_Toc194963425)

[6.2 Production Mode 16](#_Toc194963426)

[7. User Instructions 17](#_Toc194963427)

[8. Security & Integrity 18](#_Toc194963428)

[9. Future Enhancements 19](#_Toc194963429)

[10. Appendices 20](#_Toc194963430)

[Appendix A – Sample JSON File 20](#_Toc194963431)

[Appendix B – References 20](#_Toc194963432)

1. Introduction

This document provides the technical specification and documentation for the Test Project – Parcel Distribution, developed in Java with a simple Angular-based UI. The project was created for Solver IT as part of their interview hiring process.

The primary goal was to deliver a complete and functional solution within 5 working days, with an emphasis on problem-solving, code quality, documentation, and technical proficiency.

The project is divided into two core components:

* **Input Application** – responsible for capturing and submitting parcel data via a user interface.
* **Processor Application** – responsible for handling, validating, organizing, and storing parcel data on the backend.

The implementation fully adheres to the original requirements and challenges as outlined in this documentation. The following sections detail each aspect of the solution.

* 1. Parcel Management App

The Application is designed to provide users with a streamlined interface for entering, validating, and securely storing parcel delivery information.

This solution focuses on local file-based storage, organizing data into structured JSON files grouped by postcode. Upon submission, each parcel is first stored in a central source JSON file and simultaneously categorized into separate files based on its destination postcode.

**Key goals** of this application include:

* Ensuring data accuracy through validation rules
* Preventing duplicate entries
* Automating the organization of parcel data for efficient access and processing

The app is particularly suited for small to mid-sized organizations looking to manage parcel-related operations **without the need for a complex database system**.

**Note:** While JSON storage is sufficient for development, testing, or small-scale usage, I would highly recommend using a proper database solution (e.g., PostgreSQL, MongoDB) for larger datasets. It is essential to ensure:

* Better data integrity and consistency
* Easier querying, indexing, and searching
* Scalability for future growth
* Improved performance under heavy load

This project intentionally demonstrates a simplified approach but is structured in a way that would make switching to a database straightforward in the future.

* 1. Target audience
* Small to medium businesses managing parcel records
* Administrative personnel entering delivery data
* Developers who may extend the system (REST API, DB, etc.)
  1. Key Benefits
* Simplified form interface for entering parcel information
* Automatic validation and data persistence
* Smart grouping of data into separate files by postcode
* Portable — runs as a standalone .jar or .exe file
* No database required — ideal for offline/local use
  1. Technologies used
* FrontEnd - Angular, HTML/CSS
* Backend – Java, SpringBoot
* Persistence – Local File System (JSON files)
* Utilities – GSON (JSON parsing), Lombok, Launch4j (.exe creation)
  1. Prerequisites

To successfully run and develop the Parcel Management App, the following software must be available on the system:

* **Java Development Kit (JDK) 21+** - required for compiling and running the Spring Boot backend.
* **Node.js & Angular CLI -** Node.js v18+ and Angular CLI are needed for building the frontend.
* **Maven -** used for managing backend dependencies and building the Spring Boot application.
* (Optional) Launch4j or JPackage -used for creating a Windows executable (.exe) from the packaged JAR.
* **Git** (optional but recommended)- for version control and collaboration.
  1. Backend dependencies

These libraries are automatically resolved by Maven but required at runtime:

* spring-boot-starter-web
* spring-boot-starter-data-jpa
* spring-boot-starter-test
* com.google.code.gson:gson
* org.projectlombok:lombok
* com.h2database:h2 *(dev only)*
* springdoc-openapi-starter-webmvc-ui
* org.slf4j:slf4j-api
* org.junit.jupiter *(test scope)*
* org.mockito:mockito-core *(test scope)*
  1. Frontend dependencies

Resolved via **package.json** and installed with *npm install*. Includes:

* @angular/core, @angular/forms, @angular/common, etc.
* (Optional) Bootstrap or any CSS framework used for UI
* RxJS for observable handling
* Angular CLI for building the frontend app

1. System architecture
   1. Overview

The Application is designed with a clean, modular architecture, promoting separation of concerns, maintainability, and future scalability. The system consists of the following core components:

**1. Frontend – Angular UI**

A standalone Angular application provides a responsive, user-friendly interface that enables users to:

* Input parcel delivery details via a structured form
* Perform real-time field validation
* View feedback such as parcel count and submission status

The frontend communicates with the backend via HTTP requests using Angular’s HttpClient. It also features internal services for managing form data, validating input, and displaying feedback.

**2. Backend – Spring Boot REST API**

The backend is powered by Spring Boot and acts as the main orchestration layer. It exposes RESTful endpoints for:

* Receiving parcel submissions from the frontend
* Returning parcel counts
* Validating incoming data and delegating the processing task to the core logic

This backend also handles configuration, logging, and launching the Angular app (pre-built and bundled in the final JAR/EXE). Its main component is a **Processor** (Core Logic Module) - a layer encapsulated within a dedicated service. It is responsible for:

* Validating incoming parcel data
* Storing parcels in a central source JSON file
* Organizing data into postcode-specific JSON files to improve file structure and retrieval

This logic is designed to be easily swappable with a database solution in the future.

* 1. Data Flow Summary
* User inputs parcel data in the Angular form.
* Angular submits the data to the Spring Boot backend via POST.
* Backend validates and delegates the parcel to the Processor.
* The processor service stores the data in the correct JSON structure (source + postcode-specific).
* Response is returned to the user with feedback.
  1. Component Responsibilities

**Frontend** is responsible for: Form UI, validation, parcel ID display, user interaction, success messages

**Backend** - REST API endpoints (GET/POST), receives data, calls Processor, handles CORS

**Processor Services** -: JSON schema validation, file creation, de-duplication, data grouping by postcode

* 1. Architecture Diagram

+-------------------+

| User / Browser |

+--------+----------+

|

+---------+----------+

| Angular UI |

| (Input Application) |

+---------+----------+

|

HTTP GET /api/parcels/count

HTTP POST /api/parcels/add

|

+---------+----------+

| Spring Boot Server |

| (Backend API) |

+---------+----------+

|

+-----------+-------------+

| Processor Services |

| (Validation + JSON I/O) |

+-----------+-------------+

|

+---------v----------+

| Local File Storage |

| JSON files per PSC |

+-------------------+

* 1. System Design Goals
* **Modularity** – easy to separate concerns and scale components independently
* **Portability** – everything runs locally, cross-platform compatible (.jar or .exe)
* **Maintainability** – logic isolated in Processor services for easy upgrades
* **Extensibility** – easily connect to a database later if needed

1. User Interface (Frontend)
   1. Overview

The App features a responsive, browser-based form designed with Angular. It allows users to submit parcel information quickly and intuitively with built-in validation and guidance.

* 1. Key UI ElementS

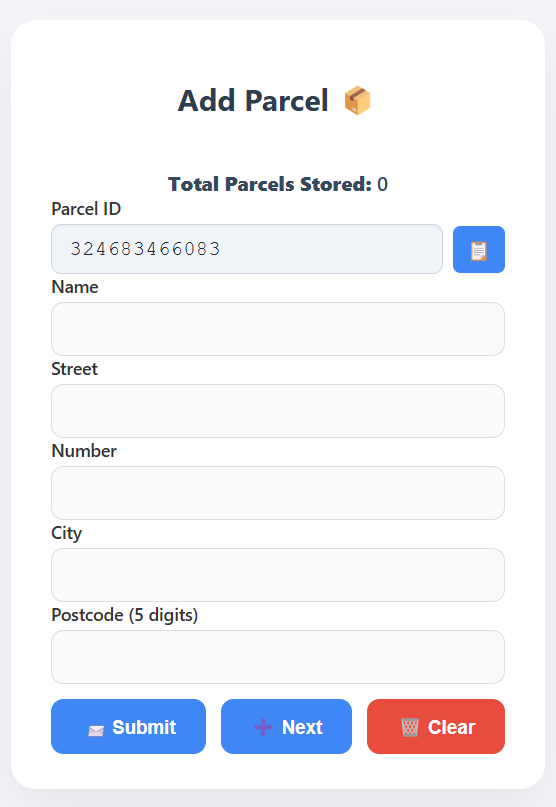
1. **Field Description:**
2. Parcel ID - auto-generated 12-digit number, not editable, visible for copy only
3. Name - recipient name, required
4. Street - street address (without house number), required
5. Number - house number - alphanumeric, / and - allowed
6. City - city name, required
7. Postcode - 5-digit postal code (no spaces allowed), required
8. **Button Functions:**
9. Submit - saves the parcel immediately, clears form, resets draft + buffer
10. Next - saves parcel, clears form for next entry (in buffer only, simulates queue)
11. Clear - wipes all input and removes draft from local storage

**3. Copy ID** - copies Parcel ID to clipboard with confirmation toast

* 1. User Experience Features
* Tab navigation between fields
* Success message (green toast) when parcel is saved
* Error messages appear inline below fields
* Sticky counter shows total parcels submitted
* “Copied to clipboard” message after copying Parcel ID
  1. Accessibility And Usability
* Form supports full keyboard navigation (Tab, Enter)
* Color contrasts meet accessibility guidelines
* All inputs include error feedback
  1. Screenshots

This part shows the demonstrating screenshots from the App.

1. **Initial empty form** where the 12-digit ID is automatically generated. It was required to let the user enter the ID, but for testing and convenient demonstrating purposes I intentionally pre-generated it. If the manual input is necessary, I would implement it similarly to the rest of the attributes of the form. Total Parcels Stored count is being requested from the server right after running – meaning currently in this case we have none yet.



1. **The form with validation errors:** if any of the fields do not fully meet the requirements, the form will inform the user. I also implemented only a 5-digit postcode, which does not allow any spaces. If more digits are required, the solution is easily expansible.

Obrázok, na ktorom je text, snímka obrazovky, číslo, písmo

Obsah vygenerovaný umelou inteligenciou môže byť nesprávny.

1. **“Parcel submitted” success toast**: if the request was successful, the parcel is added, and a notification is shown.

Obrázok, na ktorom je text, snímka obrazovky, písmo, logo

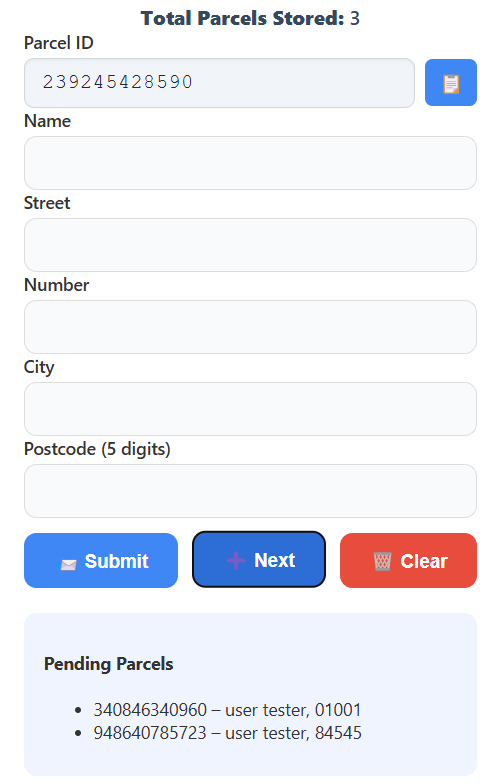
Obsah vygenerovaný umelou inteligenciou môže byť nesprávny.

1. Parcel ID + copy button: I added a little feature for the convenient UX. Sometimes it might be handy for the user to track the ID or enter it elsewhere.

Obrázok, na ktorom je text, snímka obrazovky, písmo, logo

Obsah vygenerovaný umelou inteligenciou môže byť nesprávny.

1. Pending parcels list (after clicking Next): after a next button is clicked, the parcel is sent to server and added to a “pending list”. Even though it is already processed on the server, it might be neat for the user to keep track of their parcels. Once the last parcel is submitted, the pending list is hidden. The last parcel must be entered and the submit button must be clicked to save the last parcel.



1. Backend & Data Flow
   1. Overview

The backend is developed using Java with Spring Boot, exposing REST API endpoints that receive, and process parcel data submitted from the frontend. It interacts with the processor module to validate input, prevent duplicates, and manage JSON file storage based on postcodes. Processor application is divided into two services, one responsible for structuring and reading the JSON file, the other one responsible for organizing the sub-files grouped by the postcode.

* 1. Responsibilities
* Receives and validates an incoming parcel record via the REST endpoints.
* Prevents duplicate entries based on Parcel ID.
* Delegates file handling to the processor services.
* Organizes data into JSON files by postcode.
* Launches the frontend in a browser after the app has started.
  1. REST API Endpoints

In this solution there are currently two endpoints:

1. */api/parcels/add* **POST -** submits a new parcel
2. */api/parcels/count* **GET -** retrieves all parcels as a count
   1. Data Flow Description

Prerequisites: the user fills in the parcel form and the frontend sends a POST request with a JSON body to the backend. The backend then:

* Validates the input fields.
* Checks for existing Parcel ID.
* Appends the new parcel to the respective JSON file.
* A success or error response is returned to the frontend.

**Note:** the count is being fetched once the application starts and then raised after a request for adding a parcel has been successful.

* 1. Validation Logic (Backend)
* Parcel ID must be unique (12 digits).
* All address fields must be filled.
* Postcode must be 5 digits (numeric only).
* House number must be alphanumeric (1–15 characters).

1. File Storage & Schema
   1. Storage Location

Parcel data is stored in JSON files in the current working directory of the application following the requirements.

* 1. File Naming Convention

For the organized sub-files, the following requested naming convention has been implemented:

*parcels\_<postcode>.json*

* 1. JSON Schema

For the source JSON the following schema for one parcel has been implemented (the same applies to organized files) – the complete one can be found in the appendix – or testing files directly in the source code.

*{*

*"parcelId": "string",*

*"address": {*

*"name": "string",*

*"street": "string",*

*"number": "string",*

*"city": "string",*

*"postcode": "string"*

*}*

*}*

1. Deployment & Distribution
   1. Development Mode

Frontend and backend can be run separately in development mode.

**Frontend (Angular):** *ng serve* command in the parcel-ui folder under frontend module. The Angular must be installed by npm -install.

**Backend (Spring Boot):** *mvn spring-boot:run*

* 1. Production Mode

Angular app is built (ng build) and copied into backend’s static/ folder.  
The backend is built with Maven to create a .jar: mvn clean package  
Running the app is possible by either entering the command*: java -jar parcel-app.jar –* or running it by double click.I created a RunApplication.bat file for convenient and easy use as well as parcel-app.exe file.

1. User Instructions

A user manual is added to the application for the user to follow the instructions.

1. Security & Integrity

To maintain the security of the application, I applied:

* Parcel ID field is read-only in the frontend.
* Parcel data is validated twice (FE and BE).
* Duplicate Parcel IDs are rejected.
* Data persisted immediately to avoid loss on crashes.

1. Future Enhancements

For this test project I can imagine a few nice features to make it look more user friendly such as:

* admin dashboard for reviewing stored parcels
* export to CSV or Excel
* optional authentication
* advanced filtering and searching
* previously mentioned database solution on backend side.

1. Appendices

Appendix A – Sample JSON File

[

{

"parcelId": "123456789012",

"address": {

"name": "John Doe",

"street": "Main Street",

"number": "12A",

"city": "Bratislava",

"postcode": "01001"

}

}

]

**Note:** sample JSON files are produced in the test folder. The application has been tested with 1000 sample data, for which a .js script is included as a part of the project.

Appendix B – References

* [Angular Docs](https://angular.dev/) – for FE solution
* [Spring Boot Docs](https://docs.spring.io/spring-boot/index.html) – for BE solution
* [GSON Library](https://google.github.io/gson/) – for parsing and JSON manipulation
* [Launch4j](https://sourceforge.net/projects/launch4j/files/launch4j-3/3.50/) – for creating .exe file