LEARMONTH Anders

l018160a@student.staffs.ac.uk

Parcel tracking system: Design, implementation, testing and Review

Contents

[Introduction 1](#_Toc60146700)

[Design and Build (300 words) (include references to back up my descriptions of tools) 1](#_Toc60146701)

[Choices made 1](#_Toc60146702)

[Database 1](#_Toc60146703)

[Model View Controller (MVC) 1](#_Toc60146704)

[Filter 2](#_Toc60146705)

[Design pattern 1: DTO pattern, data transfer object 2](#_Toc60146706)

[Design pattern 2: Table/view data gateway pattern 2](#_Toc60146707)

[Design pattern 3: Factory pattern 2](#_Toc60146708)

[Design pattern 4: Command pattern 2](#_Toc60146709)

[Design pattern 5: Façade pattern 2](#_Toc60146710)

[Alternative design strategies 3](#_Toc60146711)

[Session beans 3](#_Toc60146712)

[Java persistence API 3](#_Toc60146713)

[Active record? 3](#_Toc60146714)

[Data Mapper pattern? 3](#_Toc60146715)

[Testing & Results 3](#_Toc60146716)

[Future development opportunities (150 words) 4](#_Toc60146717)

[Improvements to existing components 4](#_Toc60146718)

[New features 4](#_Toc60146719)

[Design alterations 4](#_Toc60146720)

[Conclusion 5](#_Toc60146721)

[Appendices 5](#_Toc60146722)

[References 5](#_Toc60146723)

# Introduction

This report details the design, build, testing and review of a parcel tracking system which allows users of multiple pre-assigned roles to perform differing actions against parcels, orders, and their current stage in the delivery process. This could be used for an online shop looking to migrate to a single enterprise system with all data accessible in one location.

The application uses Java EE to allow interaction with a Java database and includes JUnit to enable automated testing at all stages of development and to safely add features to the system in the future without the risk of unknown breakages elsewhere.

# Design and Build (300 words) (include references to back up my descriptions of tools)

The design and build of this application are wholly consistent with the design document (fig.) which include consideration for use cases from the specification, activity diagrams, the analysis model, class diagram which also accurately represents the design patterns used, sequence diagrams and the entity relationship diagram. This document was produced early in the development process to ensure consistency within the application and to avoid deviating from the requirements.

## Choices made

Justification for the choices/selections of tools/things I used/did with this project

### Database

Db is complex but manageable and highly scalable allowing future additions to complement the overall system without negatively impacting the existing areas.

DB DESIGN MEETS MAX CRITERIA, how many foreign keys and how many relationships, how many tables, …

### Model View Controller (MVC)

View is the web tier with xhtml files and templates including CSS and other visual components.

Model contains the business logic to ensure business level rules are enforced within the application along with being responsible for communicating with and manipulating the database(s) attached to the system.

Controller is handles by Java Server Faces in this case where the action requested by the client will be handled automatically, processing the model actions and redirecting the user to the specified view without developer interaction. (Talk about why the front controller isn’t/cannot be used in this application here)

### Filter

A filter was added to handle user logins and restricts access by reviewing a white list of resources that can be accessed by users that do not satisfy the “is logged in” requirement, including the logo, CSS files, the xhtml pages and templates but to leave the remaining, un-specified files as restricted to keep unauthorised users outside of the system until they have logged in with valid credentials.

Which 5 design patterns did I use, why did I use them and how did I implement them?

### Design pattern 1: DTO pattern, data transfer object

This object handles the transfer of data for this specific object type within the system and enforces its validity of data and completeness.

### Design pattern 2: Table/view data gateway pattern

acts as an interface between the application and the database through use of CRUD methods and usually a series of additional find methods specific to the application. Typically, one table data gateway class exists per table or one view gateway class per one or more tables in the database and all SQL for the application should reside in these classes.

### Design pattern 3: Factory pattern

what is command factory? Although it is possible to use a single command factory for the entire application, the commands have been split into one factory per user type to allow greater readability and scalability for future development to remove irrelevant commands from the list being worked on, further reducing the chance of breakages elsewhere in the system. Although this requires more dependency injection as a trade-off where some beans may require access to multiple command factory files to complete their required tasks.

### Design pattern 4: Command pattern

Developers can append to the command factory to insert new functionality into the system without affecting other, pre-existing features, keeping the system modular and highly scalable.

### Design pattern 5: Façade pattern

a single point of entry for requests that will be directed to its required action on the user’s behalf (why is this good?). In this application the Command (factory?) was used as the Façade …

Any troubles when building? Design patterns, libraries, dependency injection? …

## Alternative design strategies

Describe alternative methods available that could have been used in this project, including their pros and cons. And why I did not use them in this project, could they later be implemented to help solve a specific problem or improve scalability/maintainability/performance/reliability?

### Session beans

enterprise java beans (EJB)

### Java persistence API

Handles DB interactions

### Active record?

### Data Mapper pattern?

# Testing & Results

The black box testing approach was applied to this project, where the tester knows the inputs that will be provided and the expected outputs but does not care too much about what happens in-between. This is useful when unit testing to ensure functions are performing their intended action, while abstracting away from the inner workings of the system since this level of complexity would not provide added value (ref).

Automated testing should be used wherever possible and from as early in the project as is feasible to allow finding problems earlier, resulting in easier and usually much cheaper adjustments to correct. Using JUnit for this allows the initial developer(s) of an application to start a test suite which can be later appended by any successors in a way to ensure future additions or modifications to the system do not break any existing components (ref).

A comprehensive test plan including Unit, Integration and manual tests was derived directly from the use cases outlines in the specification document. By creating a series of test cases including normal, invalid, boundary and special values, it was possible to create an extensive test plan with almost 100 percent coverage of the application using 153 JUnit and 3 manual tests which were required to test the application filter, all of which passed and are evidenced in the Test Plan document (fig).

All boundary and invalid test cases are tested in isolation to ensure one error does not mask another and that all situations are accounted for prior to release to production. Each test is independent of all other tests without dependency to allow to be run out of sequence or in part, both saving time and reducing complexity, allowing the developer to isolate problem areas and fix bug much more easily.

It was expected that there would be problems when testing the Managed Beans that had dependency injection of other classes due to a known issue with JUnit (ref). However, this was not the case and all tests were able to run without third-party tools.

# Future development opportunities (150 words)

## Improvements to existing components

Although not required in the original specification, there should be consideration to include more checks using validators prior to actions taking place, for example, if a user or parcel already exists in the DB then it should warn the requestor to prevent duplicate records being added. To accompany this, there should be full implementation of validator message and messages outputs in the case of incorrect data entry/failed validation checks.

The use of a third-party testing tool may or may not be necessary but enhanced testing should be investigated using tools including JSFUnit or Mockito which will allow automated testing of classes which have injections by processing the files as if they are mocks of the real object instead of its literal representation within the system. This will allow more coverage of the entire system, increasing confidence that it functions correctly when initially releasing or applying updates, although the current test set up did not see the expected issues so this may not be required.

## New features

The order details page could be modified to perform a calculation of sub-total weight per line item and a total weight of the order to add a delivery charge based on overall weight using a database lookup with pre-set prices per weight range.

A converter class could be created to handle weight conversion from grams to pounds and a similar approach applied for converting from GBP to EUR and other currencies to enable a more pleasant user experience with future additions to the application.

## Design alterations

With more time to work on the project, the simple JSF application should be migrated to use session beans and the java persistence API …(why) (ref)

There should also be a discussion on whether to develop a native front-end to replace the web-based faces front end because …(why) (ref)

# Conclusion

The application in its current state is fit for purpose and fully meets the original specification including performing for all 22 use cases across the 4 user types outlined in the specification, successfully performing all the required tasks consistently and without issues. This is backed up by the comprehensive set of tests both automated and manual, with the results providing confidence that every aspect of the application functions as expected with both valid and erroneous data input attempts.

The sheer number of requirements derived from the original specification document should be limited for a project of this complexity requirement and timeframe restriction as it was quickly discovered that the work required to implement and test these increased exponentially. If this project were to be undertaken again, 12 use cases will be used rather than the 22 found in this solution.

Overall, the solution presents a database of 7 tables with 4 one-to-many relationships which is used in a Java EE application using JSF, managed beans and 5 design patterns including DTO, data gateway with CRUD operations, factory, command, and façade. With a full test suite of JUnit, manual tests and integrations tests based on the use cases, which fully satisfies the outlined specification requirements and is wholly consistent with the professional design document that accompanies the project (fig.).

# Appendices

# References

**There are no sources in the current document.**