System Design Document for Hyro

TDA367, Chalmers tekniska högskola

Eimer Ahlstedt, Emil Lindblad, Sebastian Kvaldén, Timothy Nilsson, Erik Larsson

10 oktober 2021

Table of contents

1 Introduction

The purpose of this document is to give insight into the technical side of the application Hyro. Descriptions of testing methodology, software architecture, system design and more will be given in the following chapters.

Hyro, the application in question, is a simple platform enabling its users to participate in a sharing economy, and thus save both time and money. Using Hyro, users can both list items they own as available for rent and browse the listings of other users. The platform is meant to offer a beginning to endsolution, including adding items for rent, browsing, booking and paying.

Through Hyro users will be able to save money by renting items at a lower price than from big companies, minimize their environmental impact by maximising the potential of already produced items and let their personal belongings generate passive income.

1.1 Definitions, acronyms and abbreviations

- **Det funkar på min branch"** - A way of saying that a bug has been fixed only locally, not avaliable to the rest of the developers.
- **Jira** - A tool used for Agile software management.
- **JSON** - JavaScript Object Notation. A human readable data format usually used for data interchange with webservers. We use it in our application for storing and persisting data between sessions.
- **JavaFX** - A Java framework used for creating graphical users interfaces.
- **GitHub** - A tool used for storing code repositories.

2 System Architecture

The application is designed around the MVC design pattern, which means that the structure can be broken down into three different components: Model, View and Controller. All these components, each with their own responsibility work together to

2.1 Model

The Model component is the central part of the application. It contains and manages all the data in the application. It receives instructions from the controller on what to do.

2.2 View

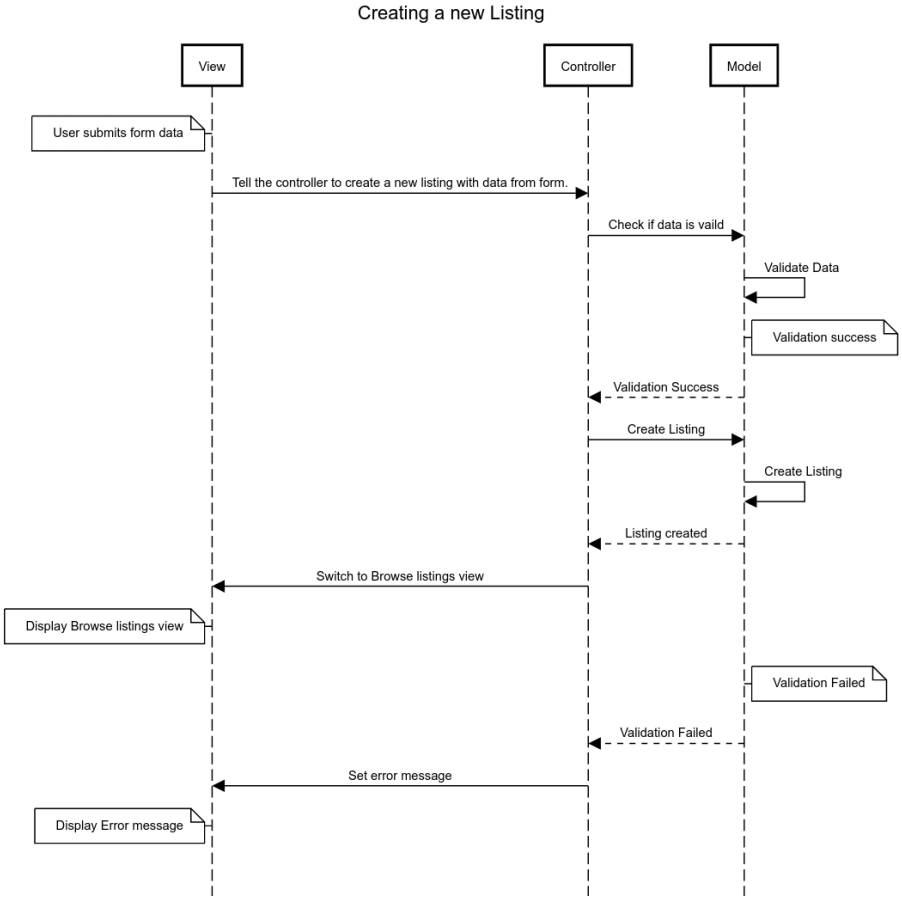
Anything that the end user can see and interact with is displayed and created by the View component.

2.3 Controller

Connection between the View and the Model is handled by the Controller component. The controller takes the user input from the View and decides what to get from the model and updates the view with the new data.

2.4 Application flow

A good way to describe how all these components work together is with a sequence diagram. In the example below (**??**), we will look at the flow though the application when creating a new listing.



Figur 1: A sequence diagram for creating a new listing

3 System Design

4 Persistent Data Management

4.1 User Data

The application uses a database to store information about users and their products. This functionality is the responsibility of the JSON package which consists of the classes JSONWriter and JSONReader. They both use Gson [1], a serialization/deserialization library, to convert Java Objects to and from the JSON file format. The .json files are stored in the resource folder JSONFiles and are written and read by the java.io Writer and FileReader respectively.

4.2 Images

In a future version, images for Products will simply be stored in a folder where they can be directly accessed by JavaFX.

5 Quality

5.1 Tests and Issues

We are using JUnit to write unit tests for our model classes. The tests are located in /src/test/java.

We use Travis CI for continuous integration. `https://app.travis-ci.com/github/emillindblad/17E-OOP-Project`

5.2 Analytics

5.2.1 Project Dependencies

Compile			
GroupId	ArtifactId	Version	Type
com.google.code.gson	gson	2.8.8	jar
org.openjfx	javafx-controls	16	jar
org.openjfx	javafx-fxml	16	jar
Test			
junit	junit	4.11	jar

5.2.2 Code Dependencies

Code - JDepend		
Class	Used by	Uses
Model.InputChecker	Controllers, View.Scenes	None
Model.Booking.Booking	TBD	Model.Listing.Listing, Model.UserPackage.User
Model.Booking.BookingHandler	TBD	Model.Listing.Listing, Model.UserPackage.User
Model.Listing.ListingState	Controllers, Model.Booking, View.Scenes	None

5.2.3 Quality - PMD

No major discrepancies between our code and best practices were found. Minor design flaws were flagged in Handlers, specifically the Law of Demeter. User and Listing are suspected to be Data Classes/Thin Classes. Finally, in every class there is missing or low amount of documentation.

Many JUnit assertions could be improved by including a message and simplified by changing assertion type.

5.3 Access Control and Security

To ensure that no one except the logged-in user can access their information everything passes through a "Handler-class where no information is accessible unless a successful login has been made. Furthermore the "handler"is structured in a way that there are no public methods to access the complete list of users and the only information that will be visible from the client side is that of the logged-in user and internal identification markers the client will use when sending calls to the "handleråhen an interaction with another user is made.

References

[1] URL: https://github.com/google/gson.