System design document for HabitHets

Elin Nilsson, Jacob Messinger, Tina Samimian, Norbets Laszlo, Oscar Helgesson

2019-10-11
1.1

1 Introduction

Give an introduction to the document and your application.

1.1 Definitions, acronyms, and abbreviations

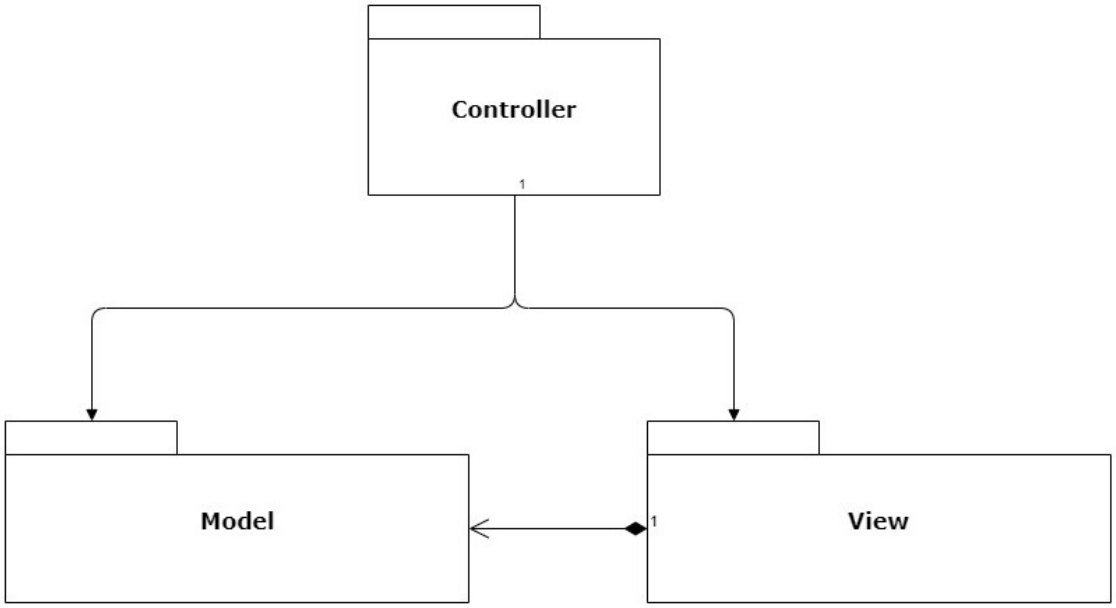
Definitions etc. probably same as in RAD

2 System architecture

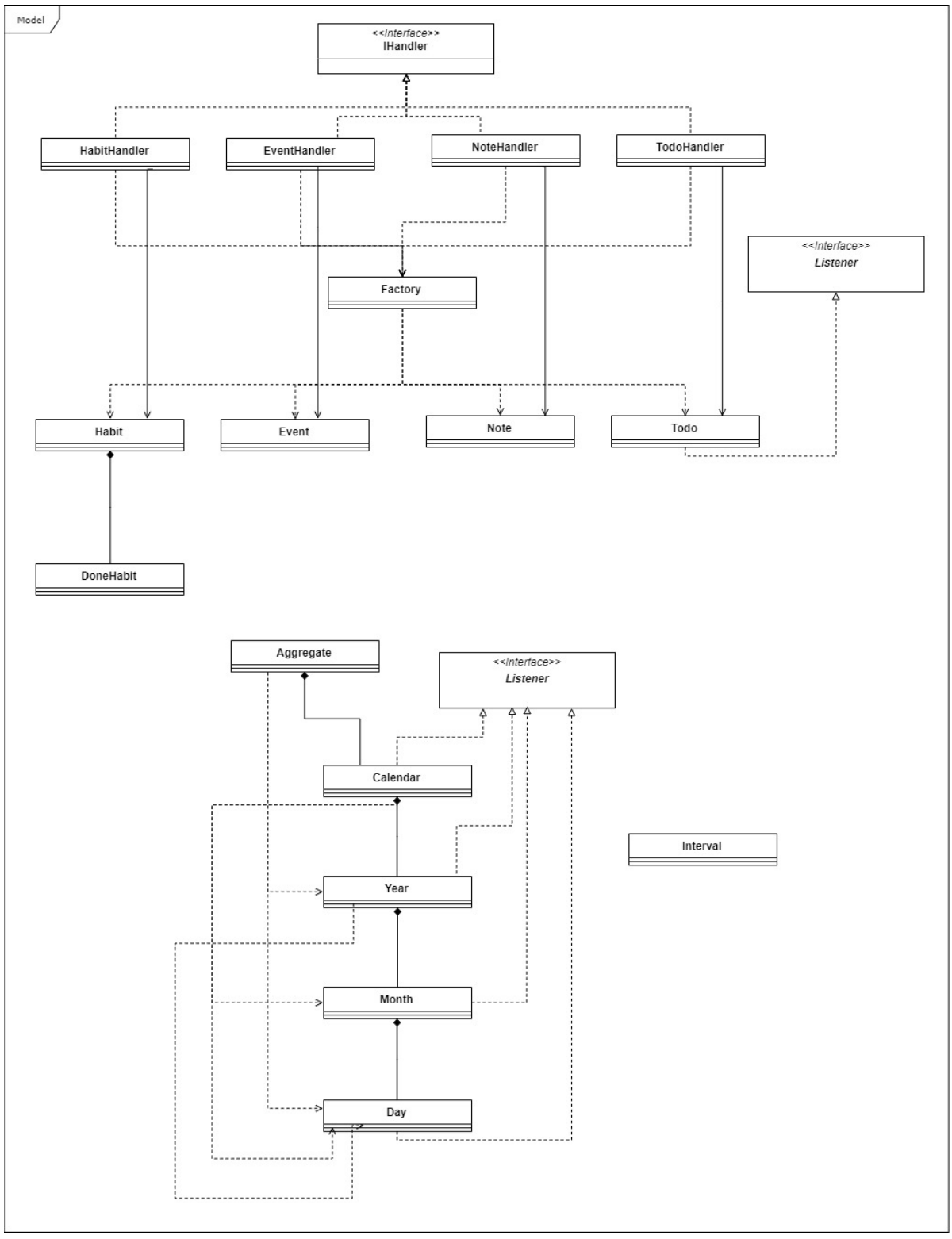
The most overall, top level description of your application. If your application uses multiple components (such as servers, databases, etc.), describe their responsibilities here and show how they are dependent on each other and how they communicate (which protocols etc.)

You will to describe the 'flow' of the application at a high level. What happens if the application is started (and later stopped) and what the normal flow of operation is. Relate this to the different components (if any) in your application.

3 System design



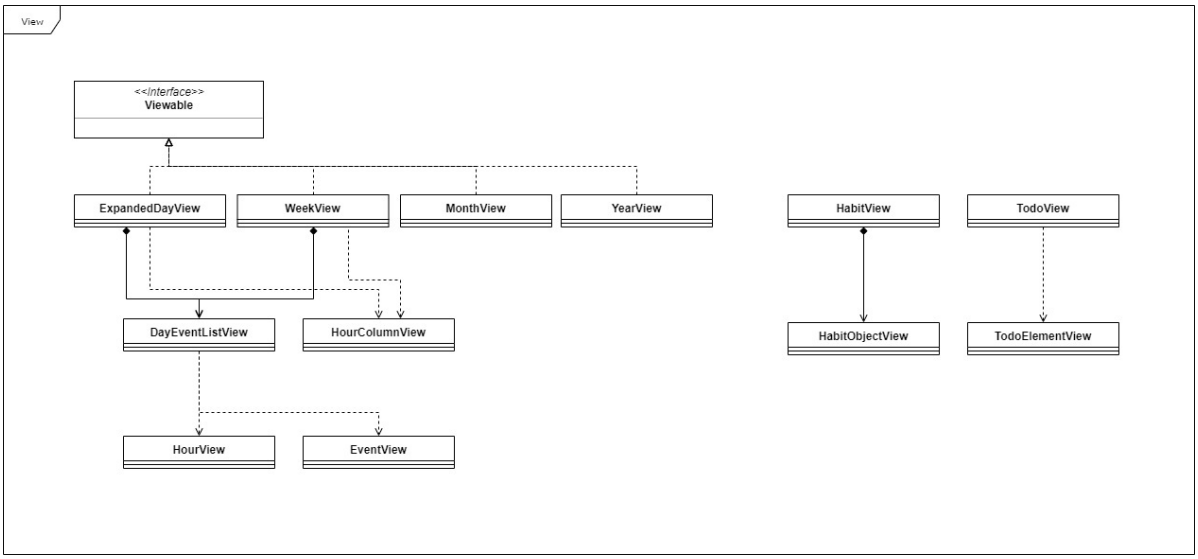
The application implements a classic MVC pattern to achieve a modular design on a high level. We separate the (View package) graphical representation and the (Model package) computing side that does the heavy work, in HabitHets case it is mostly Calendar logic and data handling.


Model
<<Interface>>
IHandler
HabitHandler
EventHandler
NoteHandler
TodoHandler
<<Interface>>
Listener
Factory
Habit
Event
Note
Todo
DoneHabit
Aggregate
<<Interface>>
Listener
Calendar
Interval
Year
Month
Day

HebitHets model package is divided into two main components. The first one from top is responsible for all the data related handling and logic, which falls into either one of the following 4 categories: Habit, Event, Note and Todo. The only way to access this data is through the factory class. Reason for that is to achieve a low coupling on a package level and high cohesion within the package.

The second part of the package is responsible for the calendar and it's logic that is used in the application. When the application is started this calendar generates days in an interval relative to today's date.

To make this calendar easily accessible it has an aggregate class with methods within that returns different parts of the calendar on call.



The View package is inspired to be as modular as possible to make it possible to reuse the different part again in other views. The applications calendar and main view can be represented in one of four ways:

- An expanded day view with more detailed information of that days upcoming events and presents the user with more functionality like the possibility to write notes that becomes connected to that day.
- A week view that summarises a whole week and acts more of a schedule.
- A month view that shows a months build in form of weeks and days.
- A year view the presents the user with a view of the years twelve months.

The week and expanded day view uses the same day components which in its turn uses multiple hour views an possible event views. In this way we maximize the reusability of the smaller views in the larger views.

All of these 4 views implements a Viewable interface which is responsible for the update of the content in the (calendar) views.

Apart from the calendar view HabitHets has two more views which are the Habit View and todo view. These views are entirely independent and does not rely on other views except their own elements that acts as the Habit and Todo objects.

4 Persistent data management

As far as persistent data goes we have three kinds of persistent data in the sense of not being deleted at restart of the program. These are Todos, Habits, Notes and Events. All data is being saved as list items in corresponding Handlers, responsible for holding logic for corresponding class in order to create a more modular design and to save data. In later iterations of the program those Handlers, to be more specific HabitHandler, TodoHandler, NotesHandler and EventHandler, will be responsible for sending the saved data to a database, and be able to fetch data back from the database.

5 Quality

As described in the Definiton of Done: ”To be considered done, all User Stories should first be unit tested to a sufficient degree of satisfaction.” This has led to a thorough testing of each implemented logical functionality.
Using JUnit as our testing platform, each method has been tested in such a way that every possible outcome has been tested for in a specific method. For example the Method ”IsCheckedToday” in Class ”TestHabit” has been tested in order to see if the habit has been checked or not. This specific method can be tested in various ways. For example when a habit is unchecked and the CheckedButton is pressed the first test-method tests if a habit then is checked today. The second test-method ”Checked-TodayTest” confirms that a checked habit that gets unchecked shows that the stack is empty. And a third test that confirms that missmatched data returns false.

Known issues include that when we switch between the views sometimes the wrong day gets displayed.

6 References

List all references to external tools, platforms, libraries, papers, etc. The purpose is that the reader can find additional information quickly and use this to understand how your application works.