System design document for ShatApp

Benjamin Vinnerholt, Filip Andréasson, Jonathan Köre, Gustaf Spjut, Gustav Häger

2018-10-01
version 1

1 Introduction

This is the system design document for ShatApp, a chatting application created to be used in a company setting.

1.1 Definitions, acronyms, and abbreviations

Some definitions etc. probably same as in RAD

2 System architecture

The most overall, top level description of the system. Which (how many) machines are involved? What are the system components, and what are they responsible for? Show the dependency between the different system components. If there are more computing entities (machines) involved: show how they communicate. Describe the high level overall flow of some use stories. Describe how to start and stop the system.

Any general principles in application? Flow, creations, . . .
Only one machine is involved in running the application. The different system components are the Model, View, Controller and Infrastructure. The Model is responsible for the applications data and logic. It is essentially a model of how the program works "at its core". The Controller and View are part of the interface towards the user. The View presents the Model for the user, and the Controller manipulates the Model based on the users interaction with the View. The Infrastructure contains things outside of the domain model that are needed for things related to infrastructure, in this

case it contains the database. The data is stored and loaded from here. If the application had a server, it would have been in Infrastructure as well.

The View is dependent on the Model in order to get data. The Controller manipulates the Model based on user input from the View. The Model notifies the View through Observer Pattern. The Model knows of the Infrastructure through an interface.

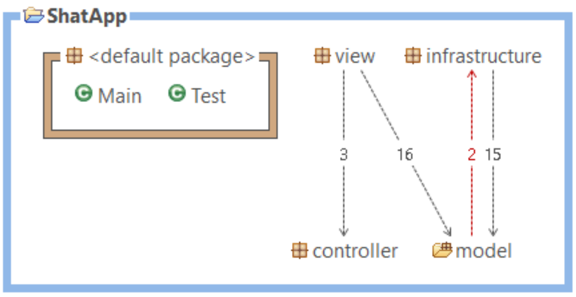

Figure 1: An overview of the system

2.1 Subsystem decomposition

Describe in this section each identified system component (that you have implemented).

2.2 Model

What is this component responsible for and what does it do. *Divide the component into subsystems (packages) and describe their responsibilities. Draw an UML package diagram for the top level. Describe the interface and dependencies between the packages. Try to identify abstraction layers. Think about concurrency issues.*

If your application is a standalone then:

- Describe how MVC is implemented
  The Model holds all the data, and has methods for manipulating it. The controller calls on the model when the user has made an input regarding data modification, or the view itself if the input is about changing a view. When data modification has occurred, a message will be broadcasted to the view, so that it can update itself appropriately.

- Describe your design model (which should be in one package and build on the domain model)
  The MainModel acts as a facade for the model, and makes it easy to access conversations, messages and users, as well as the fundamental functions such as sending a message. These important functions are part of an interface IMainModel. How all the data is stored is up to the IDataHandler, an interface for

storing and fetching data. In our implementation we use JsonHandler, which is using Gson to store the data. The data is represented by the classes Conversation, Message and User, and mainly holds primitive types. The Message and User does not hold any Users themselves, but rather IDs to other users. This is because it made the classes much easier to store, and because storing the contacts as a list of users would create an infinite loop, with a the first user having a second user as a contact, which has the first user as a contact and so on. Now, when a view needs to show info about a user, it can just request it from the MainModel.

- Give a class diagram for the design model.

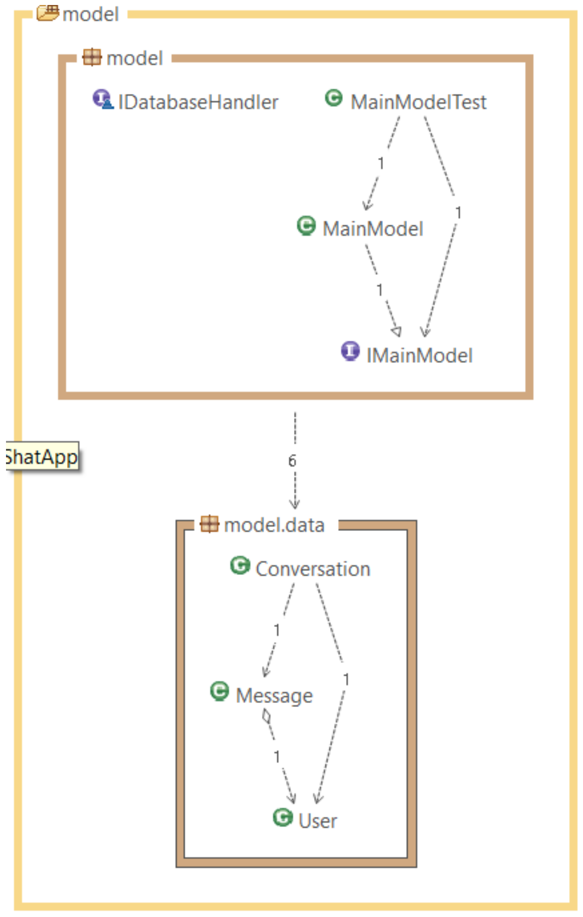

Figure 2: An overview of the model

Diagrams

- Dependencies (STAN or similar)
- UML sequence diagrams for flow. N/A

- Describe which design patterns you have applied. Iterator, Observer, Facade,

Quality

- List of tests (or description where to find the test)
- Quality tool reports, like PMD (known issues listed here)

NOTE: Each Java, XML, etc. file should have a header comment: Author, responsibility, used by ..., uses ..., etc.

2.3 View

The view uses JavaFX and consists of a number of FXML documents. These documents are linked to a controller class. This means that the view and controller are somewhat intertwined in one class, since both view related things, such as a text field, and the linked method to a send button is in the same document. To avoid dependencies we have created interfaces for both controllers and Views that the FXML controller class then implements. This makes the program modular in the sense that view and controller can be changed. You can however not change the controller class to something other than JavaFX if you are using JavaFX for the view since they are so closely related to each other.

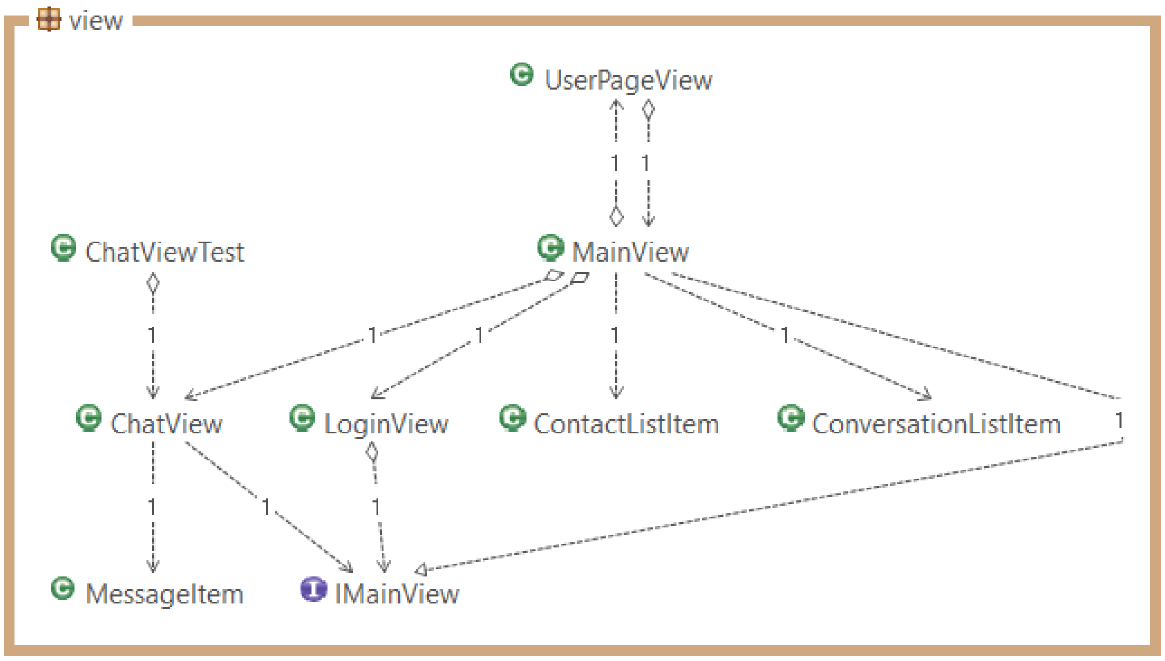

Figure 3: An overview of the view

What is this component responsible for and what does it do. *Divide the component into subsystems (packages) and describe their responsibilities. Draw an UML package diagram for the top level. Describe the interface and dependencies between the packages. Try to identify abstraction layers. Think about concurrency issues.*

If your application is a standalone then:

- Describe how MVC is implemented
  The View is comprised of classes that implement both controller interfaces and view interfaces. They also implement the observer interface, in order to observe the MainModel which is an Observable. The controller interfaces have methods that can modify the model based on input from the user through the methods implemented via the view interfaces.

- Describe your design model (which should be in one package and build on the domain model)
  The view is comprised of a main class called MainView that extends a JavaFX AnchorPane and implements the Initializable, IMainController, IMainView, and Observer interfaces. This then contains the other view-related items that are loaded upon starting the program. It houses the LoginView, ChatView, UserPageView and it also has a relation to the MainModel through the IMainModel interface. MainView is an Observer of MainModel.
  The ChatView loads messages from the MainModel which it then puts in MessageItem that are displayed in the chat area.

- Give a class diagram for the design model.

Diagrams

- Dependencies (STAN or similar)

- UML sequence diagrams for flow.

- Describe which design patterns you have applied.

Quality

- List of tests (or description where to find the test)

- Quality tool reports, like PMD (known issues listed here)

3 Persistent data management

How does the application store data (handle resources, icons, images, audio, ...). When? How? URLs, pathe's, ... data formats... naming..

The application serializes the data from the model into JSON using Gson in order to store it. It also uses Gson to deserialize the JSON into objects again. This is done by a

class, JsonHandler, implementing an interface, "IDataHandler". The JsonHandler and IDataHandler as well as the JSON files reside in the Infrastructure package. Users are saved into users.json and conversations to conversations.json. Since the Conversation holds a list of Message these need not be saved separately, instead the list is simply serialized as is.
There is a resource package in the main package called "resources" which houses the fxml-documents used for the view in a package called "fxml". The pictures used by the application by default are also stored in resources, in the package "pics".

4 Access control and security

Different roles using the application (admin, user, ...)? How is this handled?

5 References