System Design Document for the KudoMessage Project (SDD)

Contents

Table of Contents

System Design Document for the KudoMessage Project (SDD) **1**

1 Introduction **2**

1.1 Design goals **2**

1.1.1. System architecture 2

1.1.2. GUI 2

1.2 Definitions, acronyms and abbreviations **2**

2 System design **3**

2.1 Overview **3**

2.1.1 The webpage – Frontend and backend 3

2.1.2 The central server 4

2.2. Software decomposition **4**

2.2.1. General 4

2.2.2. Decompositions into subsystems 4

2.2.3. Layering 4

2.2.4. Dependency Analysis 4

2.3. Concurrency issues **8**

2.4. Persistent data management **8**

2.5. Access control and security **8**

2.6. Boundary conditions **8**

3. References .... Error! Bookmark not defined.

Version: 2.0

Date: 13-05-26

Author: Group 21

This version overrides all previous versions.

1 Introduction

1.1 Design goals

1.1.1. System architecture

One of the design goals with the application is to minimize the data traffic between the clients and the central server. The main reason for this is to avoid potential operating costs related to hosting of a high-traffic central server.

Another goal, in addition to what's mentioned above, is skip the handling of user data and saving of user messages. This responsibility is instead handed over to Google, where the messages are stored under a label named "KudoMessage SMS" at the user's Gmail account. The user authentication is handled via OAuth 2.0. The contacts are handled in a similar way, the application retrieves them from Google Contacts.

1.1.2. GUI

The design should be easy to grasp for a beginner and aesthetically pleasing. The GUI should follow accepted design conventions to eliminate excise for the user, and the program in general should have a jointly chosen design. For the android server the design doesn't have to be all that extensive, all it should be able to do is to allow the user to log in with their Google account and input the address to the central server, if they choose to use another one but ours.

1.2 Definitions, acronyms and abbreviations

- GUI, graphical user interface
- Java EE, platform independent programming language (EE stands for Enterprise Edition)
- Glassfish, a web server to host java files
- JSF, (Java Server Faces) The HTML pages with connection to java classes.
- Hustler, the push server
- OAuth 2.0, gives us the possibility to access server resources on behalf of the resource owner
- GCM (Google Cloud Messaging), a service that helps developers to send data from servers to their Android applications on Android devices.

2 System design

2.1 Overview

The application will use a MVC model, even though it won't have a comprehensive model. The frontend in the web application will be built using HTML and CSS, with Java as backend.

2.1.1 The webpage – Frontend and backend

Both the frontend and backend are developed using Java EE, it is this part of the application that renders and takes care of the graphical user interface. The visual parts are represented with XHTML and CSS, and the backend, including all the functionality, are developed in Java. Each and every HTML page has a connection to one or many Java classes, it is these classes that provides the ability to send and receive messages.

The conversations for a specific user are stored in a HashMap in the class ConversationsHolder. The key for the HashMap is a phone number and the value is the class Conversation that contains a list of messages. A message in the application is represented through the class KudoMessage, and every KudoMessage contains a message content, an ID, origin and a list of receivers.

When logging in to the application, the user is first redirected to Google where they have to log in with their Google account. Google, in turn, sends the user back to the application together with an access code. The class OAuthController is runned and takes the access code from Google and converts it to a so called "access token", which later is used when the application needs to access the user's Gmail, Contacts and email adress.

As soon as the application is opened the class BackendBean is loaded, which is the class that connects to the central server using sockets. When this is done the open socket is passed on to a new instance of the class SocketHandler. SocketHandler starts of by telling the central server whether it's a client or a gateway, followed by requesting the user's email, contacts and the first X number of messages. Afterwards it awaits new messages from the central server, when a new message is received it adds these to ConversationsHolder.

When a user writes a message and sends it through the GUI a class called MessageModel is called, which requests a reference to the socket's output. IT then sends the message, as a JSON object, via the socket to the central server.

Globals stores references to the PortableRender which is used to redraw the GUI, the users email address, which PushGroup the render uses, to idenentify whenever a message is sent or received (by checking the message origin) and an access token which is set by the OAuthController and is used in the authentication process with the central server.

CONSTANTS stores default and constant values used by the application such as API-keys.

2.1.2 The central server

When the application starts, the class Main is loaded which listens for incoming connections on the socket. When a new connection opens an instance of the class ConnectionHandler is created by sending a reference to the connection to the constructor.

ConnectionHandler is then loaded and begins to listen for the newly received message from the sender, which tells the class whether the new connection belongs to a new client or a gateway. If it's a client the class sends the reference of the connection to a new instance of the class ClientHandler, and if it's a gateway the reference is, conveniently, sent to the class GatewayHandler.

The GatewayHandler handles the communication between the gateway and the central server.

The gateway can register istelf as a gateway and then push out messages to the clients using this class. On every new connection the gateway is required to provide an access token in order to identify which user the gateway is conneted to.

2.2. Software decomposition

2.2.1. General

The application is decomposed into the following modules, see figure X

- Torsken (the web client), the GUI to interact with the users.
- Hustler (the central server), communicates with Google and the clients.
- Jessica (Android server), android application that is the user gateway. Pushes SMS to central server on receive and sends out messages received from the web client via the central server.

2.2.2. Decompositions into subsystems

The application is decomposed into three subsystems. We have the web client with the GUI for the users. We have the android push server which is an an android application that acts as a background service. Last but not least there is the central server, this module handles communication between all clients and retrieves user information from google.

2.2.3. Layering

The layering is as indicated in Figure below.

2.2.4. Dependency Analysis

Dependencies are as shown in Figure

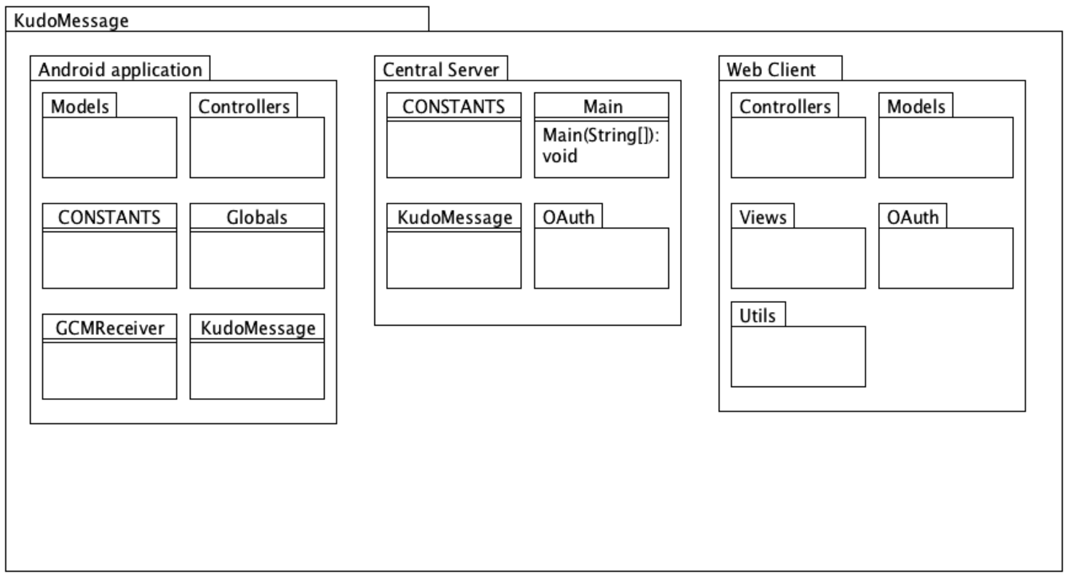

Figure 1: Layering

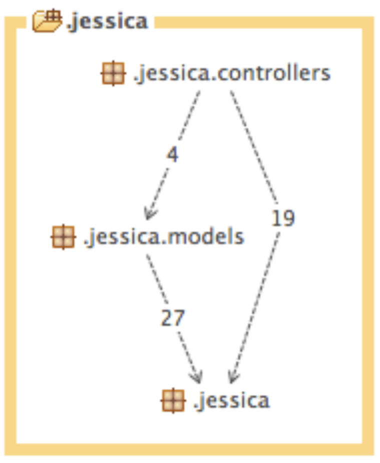

Figure 2: Layering and Dependency analysis of Jessica

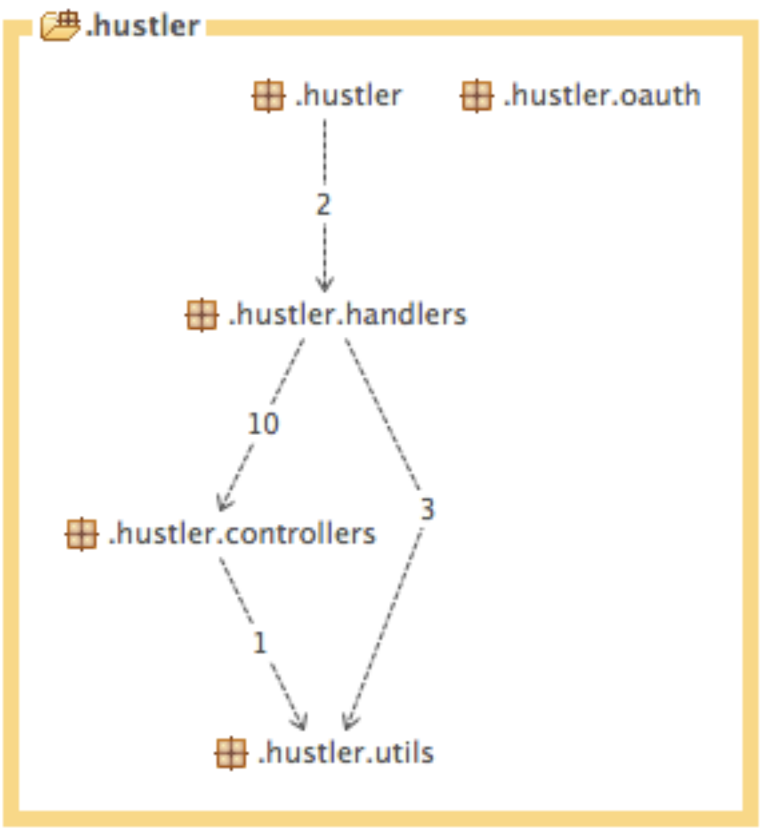

Figure 3: Layering and Dependency analysis of Hustler

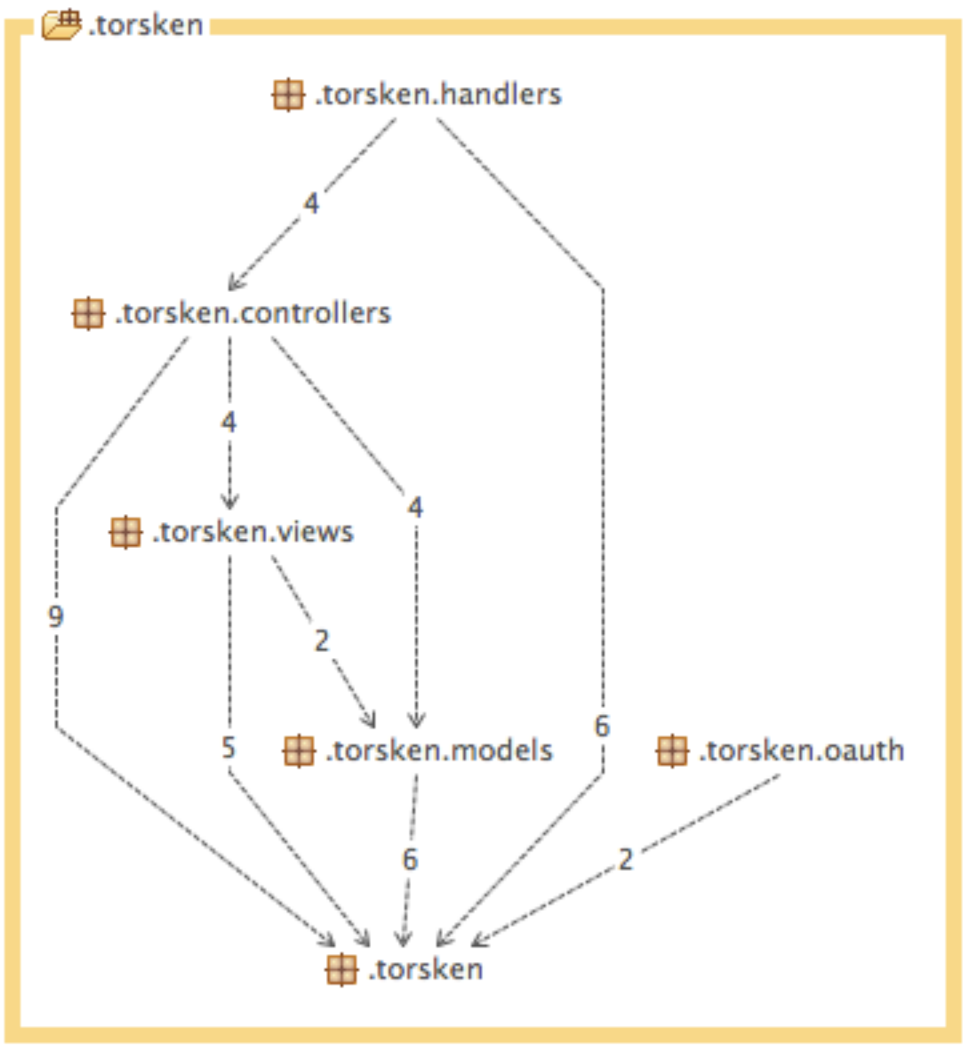

Figure 4: Layering and Dependency analysis of Torsken

2.3. Concurrency issues

NA. None that we are aware of.

2.4. Persistent data management

KudoMessage will handle SMS conversations from the user. It will not store the conversations between sessions though. Every time KudoMessage is started the central server will request the contacts and the SMS conversations from Google directly. This saves us alot of storage and is a more secure way to go because we dont need to save users SMS conversations or passwords of any type. Google takes care of everything.

2.5. Access control and security

As mentioned earlier we don’t store any SMS conversations, contacts or passwords. The users will login with their gmail account and give KudoMessage access to their contacts and e-mail. This will spare KudoMessage from saving and encrypting passwords. The main reason why we decided to do it like this is so there won't be any security leakage.

2.6. Boundary conditions

NA.