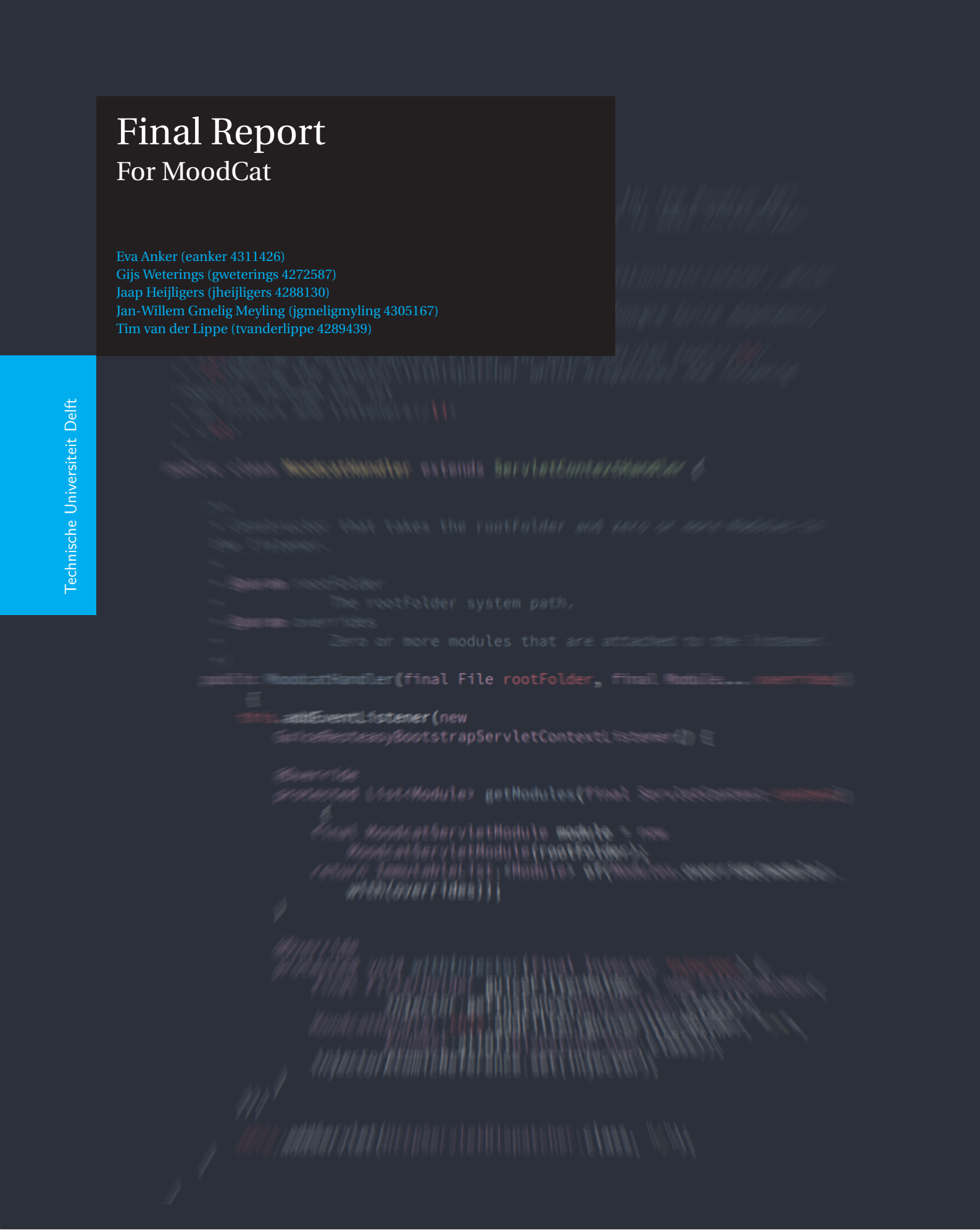
Final Report
For MoodCat

Eva Anker (eanker 4311426)
Gijs Weterings (gweterings 4272587)
Jaap Heijligers (jheijligers 4288130)
Jan-Willem Gmelig Meyling (jgmeligmyling 4305167)
Tim van der Lippe (tvanderlippe 4289439)

Technische Universiteit Delft



TU Delft Delft University of Technology

Challenge the future

FINAL REPORT

FOR MOODCAT

by

Eva Anker (eanker 4311426)
Gijs Weterings (gweterings 4272587)
Jaap Heijligers (jheijligers 4288130)
Jan-Willem Gmelig Meyling (jgmeligmyling 4305167)
Tim van der Lippe (tvanderlippe 4289439)

Supervisor:	Cynthia Liem	TU Delft
Teaching Assistants:	Friso Abcouwer,	TU Delft
	Abhishek Sen,	TU Delft

An electronic version of this document is available at https://github.com/MoodCat/MoodCat.me/.

TU Delft Delft University of Technology

Contents

1

INTRODUCTION

The context project is a project in which bachelor students at the TU Delft should develop a product in 10 weeks. Our clients from the business world want this product to be developed.

The task we were assigned to was to design the music services of tomorrow. Our group had to design an application that would allow users to listen to music of a certain mood and interact with people in the same mood.

The system should allow users select a mood and a room, whereafter they join a room according to the selected mood. When a user is logged in, he should be able to send messages to other users in the room. Additionally a user can classify a song according to mood of the song. Lastly there should be a ranking game where users can classify unranked songs. The user shall be rewarded with points for contributing to the system.

In this report we will explain MoodCat and how we have developed this system. First we will give an overview of the overall system and secondly reflect on the product and process. Thirdly we will explain the different functionalities of MoodCat After that we will elaborate the interaction between the user and our system. Then we will evaluate the functionalities and provide a list of implemented features. Lastly we will give an outlook of possible improvements for the system and how they could be implemented.

2

Overview of the developed and implemented software product

MoodCat is a website where people can listen to music together It combines enjoying your mood with social interaction. When entering MoodCat, the user can choose one or more moods from a given list. Based on the selected moods, a list of rooms sorted on relevance is shown, displaying the names and the song currently playing in the rooms. When entering a room the user can chat with other users that have joined the room.

The software of MoodCat consists of two main components: the Java based backend and the frontend which is a website, hosted at http://moodcat.me.

2.1. Frontend

The frontend uses Angular.js for managing its objects and connection with the backend and bootstrap for the layout and design. Angular.js is an MVVM framework written in Javascript. While it provides a nice way to split and integrate with models, views and controllers, it also simplifies managing and updating displayed data. Angular parses HTML elements that contain custom tag attributes. The attributes are interpreted by angular and replaced with actual data from the model. The model is updated through constant communication with the backend. The current song and the time of the music is synced with the backends in order to make sure the users in a room listen to the same song at the same time.

2.2. Backend

The backend is responsible for managing the underlying structure of the project, to connect loosely coupled with the frontend and to persist and retrieve data from our database. The backend connects through hibernate to a postgres database running on a server which is also hosted on the same server. The database contains the songs, artists, rooms and users. When the backend is started, it connects with the database and retrieves the current rooms. The REST API is then initialized, which can handle calls at http://moodcat.me/api/. The backend provides models, of which the fields are sent through the API as generated JSON. The models represent the inner content of the entities in the database.

2.3. Mapping moods

In order to compare rooms, moods and songs we use valence arousal vectors. Every mood is mapped by a 2 dimensional vector of valence and arousal, both of which have values between -1 and 1. Each room has its own vector which is used to determine which room to suggest based on the selection of moods. In order to queue up new songs for a room, the backend will fetch new songs that have vectors close to the room's vector. The database uses a RTREE spatial index to quickly calculate distances and order the songs accordingly. The songs are mapped to vectors through a user feedback system. When a user likes a song that plays in the room, the song can be upvoted. The system then knows this song fits in the room. When a user downvotes a song, the song apparently does not fit the room. The user will now be asked to provide a classification for the song. The user can choose a degree of both valence and arousal and the valence arousal vector is slightly tweaked accordingly.

3

REFLECTION ON THE PRODUCT AND PROCESS FROM A SOFTWARE ENGINEERING PERSPECTIVE

In this chapter, we discuss the software engineering perspectives of MoodCat. First, we discuss our developing process. Secondly, the general structure and architecture of the product is explained. Then we will take a closer look at the used and implemented design paterns. After that we will elaborate the choices to increase code readability and maintainability. Finally, the most important libraries and languages we used will be listed.

3.1. PRODUCT ARCHITECTURE

The architecture of MoodCat is very similar compared to existing websites. We're using a client-server architecture. The frontend (client) is a AngularJS web application, whereas the backend (server) is a Java API based on JAX-RS[1]. The backend communicates with a persistence server in the form of a PostgreSQL database.

3.2. DESIGN PATTERNS

Due to the usage of various libraries and frameworks, we use quite some design patterns. Additionally, we did design our backend system very methodically, using the model-view-controller (MVC) pattern. The API classes define the view of the application (in the form of JSON responses) which is fetched by the frontend to be used by the client. The "backend" package contains the controllers of the system. The entities in the "database" package define our models.

In the frontend we use a slightly different structure, most commonly referred to MVVM (Model, View, ViewModel). The API calls are the models obtaining the data. The controllers and services are the viewmodels, which not only show but also alter the content. And lastly the HTML templates are the views showing the website to the user.

A few of the design patterns we used are: Facades, Proxies, Singletons, Factories, Dependency Injection, MVC, MVVM, and resource pooling .

3.3. CODE READABILITY / MAINTAINABILITY

In our efforts to create a codebase that not only fit our needs, but is readable and maintainable by others as well, we valued clean, short code greatly. Methods are kept short and simple, fill only one purpose and have descriptive names.

We used the new technologies in Java 8 to it's full potential, substituting for loops and (nested) if statements for lambdas and anonymous classes wherever applicable.

In order to ease the communication with the database, we build the SQL commands using QueryDSL, which implements the builder pattern. Added benefit was the type checking capability of this pattern, de-

[1] The implementation we use is RestEASY, which can be found on http://resteasy.jboss.org/

creasing the risk of writing incorrect queries and making sure when making updates to entities, queries remain correct.

Google Guice allowed us to build our application like LEGO™: composition at its best. Guice is a Dependency Injection framework which allows looser coupling between classes and packages. Responsibilities are split better and rapid prototyping is possible without having to worry how to pass dependencies around in the project.

Lastly, the MVC-structure enabled us to change inner workings of the backend without breaking the API contract with the frontend and/or without breaking the database schema.

3.4. SCRUM

We worked with the SCRUM-methodology for the second time. Agile methodologies were known to most of us before this project, but no one had really used it consistently, aside from the basics of SCRUM during the SEM course. We did some research and found a really nice guide from Atlassian[2]. We learned about the different roles of SCRUM: the SCRUM Master, the Product Owner and the development team, as well as how to manage meetings in a proper manner. We decided to have daily SCRUM meetings at 09.00 o'clock every morning. During this meeting, we would use our Waffle[3] board to keep track of the groups progress. Each member of the group had to answer 4 questions:

1. What have you done yesterday?
2. What are you going to do today?
3. Are you on schedule?
4. Do you need help on anything?

This went surprisingly well! In our first sprint we had a little trouble guessing how long an issue would take. We discovered quickly in the following sprints that story points in Waffle helped us managing tickets and sprints. We decided that 1 story point equals 2 hours of work.

SCRUM meetings were useful to keep an overview of the system, as well as help others out when they were stuck. We did continue past 15 minutes a few times, but usually with a smaller group for a specific issue.

Overall, we're all convinced SCRUM is a great way to develop software, however we think that a sprint could better last 2 or 3 weeks instead of one. The sprint planning and sprint reflection took up a considerable amount of time in the sprint and the weekly demo forced us to put a difficult issue aside more than once.

3.5. USE OF LIBARIES AND LANGUAGES

We have used quite a lot of languages and libraries. To get a clear view of what belongs where, we split the system in our three components.

3.5.1. FRONTEND

The frontend for Moodcat is the website loaded into the user's browser.

LANGUAGES

To create a website, we use the standard web languages. HTML for the DOM creation, javascript for client-side logic, DOM manipulation and communication with the backend API and CSS for styling.

LIBRARIES

We decided to use AngularJS to increase the productivity of developing features on the frontend. This is a framework built upon javascript and it deals exceptionally well with DOM updates (with 2-way data binding) and HTTP calls. It also enables re-usable code with directives, a mechanism to create an on the fly webcomponent. We actually extend AngularJS further with packages to suit our specific needs.

In order to generate CSS, we use preprocessor called SASS that eases the re-use of CSS rules and markup.

[2] https://www.atlassian.com/agile
[3] http://waffle.io

3.5.2. BACKEND

LANGUAGES

The backend is written in Java. We've chosen Java because everyone in the team was most experienced in this language.

LIBRARIES

To turn the application into a servlet and process HTTP requests, we use Jetty. The RESTful API is implemented using RestEASY, a library that implements the interfaces defined by JAX-RS. JAX-RS has a dependency injection module to ease the process of defining end-points. However, we also wanted a more complete dependency injection framework. Therefore we decided to use Guice for constructor injection, together with Guice-JPA for transaction support and Guice- assisted-inject for factories.

We're using Hibernate for ORM (Object-relational mapping). This allows us to transparently "store" Java objects in the database. Tables are generated according to the fields defined in the entities.

In order to quickly select songs or rooms from the database, based on their vector, we use a RTREE as spatial index. In order to enable RTREEs in our database and backend, we use Hibernate Spatial.

In order to ease the development process, we use two more dependencies: To simulate a postgres database during testing, we're using H2 (with Hatbox for spatial query support). Besides that, we're using Project Lombok for automatic generation of getters, setters, hashcode and equals methods.

4

DESCRIPTION OF THE DEVELOPED FUNCTIONALITIES

In this section we will discuss the main functionalities of our system. MoodCat can be split up into three main features. These functionalities are the ability to play music, chatting in the rooms and classifying a song. They will be described in the sections below. But before explaining that, we will give a small overview of the structure of the system.

4.1. INTRODUCTION TO MOODCAT

When a user enters the website he selects one or more moods. Then a list of rooms fitting that selection of moods will appear. When clicking on a room a user joins, and he can listen and chat with other people in the room. A user can vote if the song is appropriate for a song. If the song is not appropriate the user gets a small pop-up on screen to tell how the song should be rated.
MoodCat also has a gamification aspect where a user is served with songs not yet classified. The user then gets a snippet of the song (30 seconds tops) and is then asked for a rating.

4.2. LISTENING TO MUSIC

MoodCat is developed as a music application and thus the ability to play music is very important. Although, our music player is a bit different from a normal music player. MoodCat works as a streaming service, and because of that the music is synchronized with the room the player is in. So you can only mute the sound, not alter its progression. When the internet connection of the user goes down for a few seconds or the connection is slow, the music synchronises with the room as soon as possible.
In fact, a room in MoodCat can be compared to a radio station.

4.3. CHATTING

In Moodcat you can chat with other people in the room. In order to send a message the user needs to be logged in using their soundcloud account.
When a user is logged in, he can chat by typing a message in the "Type a message" bar. The user sees the messages from other users, accompanied by their usernames. The name of a user is the name a user has set in SoundCloud as their real name.

4.4. CLASSIFYING

As earlier described, when a user disagrees with the classification of a song, the user can give his own classification. This is done with valence and arousal, which are terms to express the mood of a user. To classify a user gets different buttons, with an image of state. Theses images are designed in such a way that it only can be seen as one state. When a user plays the classifying game the user classifies the snippets with the same buttons as the normal classification.
The classification effects how the song is classified as a mood. When a song is downvoted by a lot of people

in the room, the song is not played in that room any more. It can also be played in a new room, according to the new classification of the song. This makes the music the user gets better at every classification that is submitted.
When a user classifies, the user earns points. The scores can be found on the leaderboard. For classifying an unrated song a user gets more points than a normal classification. This is done to stimulate the user to classify more songs, and thus grow MoodCat's collection.

5

HUMAN-COMPUTER INTERACTION

5.1. INTRODUCTION

This section describes the interaction between MoodCat and one of its users, and how we measured this. The user is very important for us, the user is the one that will use the application. Our system is build for interaction between users and the system is even learning from user interaction. The interaction between users is the chatroom that our music application is build around. Users are also involved in actually improving the system.

5.2. THE USER AND THEIR SITUATION

5.2.1. USE CASE SCENARIOS

In this chapter we will elaborate the use cases we have made for our product.

USER WANTS TO PLAY MUSIC ON THE WEBSITE MOODCAT.ME.

- The user goes to moodcat.me in his browser
- The user clicks on the mood he is in
- The user clicks next
- The user selects a room

WHILST IN A ROOM THE USER WANTS TO CHAT

If the user is not logged in yet:

- Click on "login" in the top right corner
- Click on the connect button

Everyone:

- The user clicks on the bar which says "type a message".
- The user clicks "send"

USER WANTS TO DO THE CLASSIFY GAME

- User clicks on "Rating game" in the header
- User clicks a song
- User listens to a song
- User selects the appropriate buttons for valence and arousal
- User clicks submit
- User gets 6 points
- User comes back to the menu to click a new song to classify

5.3. CONTEXT INQUIRY

We did a context inquiry with 5 other students a the TU Delft. The students joined the same chat room. They were separated by some free space, so they would not be influenced by each other. We just asked them to play with the system and we would see how they do. To see how they experienced the system is quite interesting, a user acts different in the first few minutes of system use, when they're not quite sure what MoodCat is. These are two very important aspects. First off all we want to bind the user to our system, so he will use it and then we want them to keep using the system. So, that the user comes back often, because he likes the system.
They could find almost all the features, they immediately found the rating of the song and the classify game. Only the leaderboard, where you come by clicking at the points was a bit too hidden away. This is something minor, but we want to change this in the future.

5.4. PRODUCT DESIGN AND CLAIM ANALYSIS

MoodCat is developed with the end-user as starting point, we tried to make the interface as intuitive as possible. The interface should not contain a lot of information, so that the user remains the overview.

5.4.1. CLAIM ANALYSIS

MoodCat is developed for people to enjoy music together with similar minded people. We were familiar with Twitch.tv, a streaming platform geared towards games. A big part of their success factor is the fact that users can interact with eachother while watching the stream. In our view this formula could also work with a music stream.

5.5. USABILITY EVALUATION

5.5.1. COGNITIVE WALKTHROUGHS

During the design of the product we did multiple usability evaluations. We have done quite a few Cognitive Walkthroughs, with changing possible end-users. We done this with the thinking out loud protocol, so we always knew there they where stuck or did not fully understand something. When you do not ask people to think out loud, they would sometimes hide some mistakes, which later on can be very important. We gave the users the task to explore the functionalities of the system. We got some feedback and found that very valuable. Mainly, because in an early stage you know what goes wrong and right. So, anticipating on the results is therefore a lot easier than later on.

5.5.2. GOMS

At the end of our project we did a GOMS analysis. We asked them to do some task and watched how quickly they preformed these tasks. These where the tasks of the Cognitive Walktrough, only split into multiple sections. We wanted to see where sometime went wrong. So, not that the user had some difficulties with finding a chatroom, but that they spent longer on the select a mood page than we would expect. These more specific failures gave us more information to work with.

6

EVALUATION OF THE FUNCTIONAL MODULES AND PRODUCT

Moodcat is roughly developed in four modules. For the backend we can distinguish the persistence layer, the backend layer and the API presentation layer. The frontend can be considered a module on itself. We will now describe how we reflect on the implementation of these modules.

One of the more controversial choices in our project is undoubtedly the relatively heavy usage of software dependencies and generated code (see section 3). The downside of this approach is that we all had to learn and adapt to the various libaries and frameworks. On the other hand, however, the used libaries and frameworks really set the bar on code quality and kickstarted our project. We would defenitely use JAX-RS, AngularJS and Hibernate again if we had to write a web system for future projects and we are glad that we took this project as the opportunity to work with these tools.

If we would have to pinpoint two failures in the development process, we can list two design problems. One failure was that we did not know our ORM framework (Hibernate) well enough. This led to quite some bugs which involved low level debugging in the framework. Eventually we got it fixed but it took relatively much time. Most of the problems were caused because we wanted to synchronize chat messages in the database so that another server would be able to take over in case of a failure or when scaling out. But in the end we can debate if this was such a good idea to implement something so complex for something that is "nice to have".

For the web interface there is still some room for improvement. Most importantly the unclear positioning of various elements and the lack of descriptive headers made some parts of the website not so intuitive. The only real failure we encountered in the UI layer was that the two ways of listening to music (the classification game and the room listening) had quite some conflicting logic which had to be dealt with. Because we designed these views / features separately, we did not plan out how transitions between the two modes would be handled well enough. This caused some bugs that had to resolved later on and the implementation of these fixes can still be improved using, for example, a state pattern.

Aside from these two failures, we we are really happy overall about the product we have built in the past few weeks. First of all, the product we have built is in a deployable state and almost all feedback from our current user group was positive. Secondly, we have managed to realize almost all must have requirements, and missed just one should have requirement. This of course besides the neural network, which we unfortunately had to drop in the development process. We however do think that the neural network that we designed would work in practice and that it would be a huge improvement both to the functioning and user experience of the system. We will further elaborate on this in the section Outlook.

Without the neural network, the system is suffering from a cold start problem. This will be solved over time through processing user input.

7

Outlook

During the project, we developed several improvements to the system. These improvements could not have been implemented due to time constraints, but the system designed in such a way that support for these features can be easily achieved.

Song and room suggesstions

Our initial selling point was a neural-network based suggestion system. A neural-network is a system that can classify items based on features and output in one (or few) dimensions. In order to train a neural-network, you also need pre-determined labels/expected values for the dimensions. In our case, the neural-network should output valence and arousal. However, existing datasets did not contain those values. How to obtain the valence and arousal of song-features is currently researched by various universities, but no method has been succesful yet. Therefore we were unable to train and use our network.

At the moment users can classify songs and provide the system the expected valence and arousal. The weights of the nodes in a network can be trained using the user-data gathered over time, which makes it possible to integrate a network in the future.

Message polling versus broadcasted events

Currently all room interactions are based on a status-poll structure between the frontend and backend. The backend provides the song that is currently playing and the latest chat messages which the frontend uses to quickly check their status. This unnecessarily stresses the network connection, as most calls do not contain additional information the frontend did not have yet.

To reduce the network usage, the polling-structure can be replaced with socket events. Therefore only when there are updates, packets are sent between the frontend and backend. This also makes changes more instant as the clients will immediately get notified, instead of on regular intervals.

User interface

There are various possible improvements to the user interface.

1. At the moment of writing, the room selection page contains a list of rooms. This list of rooms is sorted by relevance, but there is no indication on how relevant a certain room is.

   Our clients indicated a more suitable representation would be to display the select rooms in a 2D space. This space would implicitly represent the inner vector structure used in the backend. To indicate the selected moods and the corresponding vector, a point is marked in the space. Using this represenation, relevant rooms are placed more closely to the point, whereas less relevant rooms are placed further away. The room selection page can be easily changed to suit this new representation.

2. The up-vote and down-vote buttons are at the moment placed in the top bar. The initial philosophy was that while a song is playing, the user should be able to express his/her feelings. Since the currently played song is in the top bar, the natural decision was to place the ote buttons next to it.

   After gathering user feedback, one of the outcomes of the tests was that the purposes of the buttons was unknown. Some prototypes can be developed with different placements of buttons which can be tested on a set of users. Then the users can vote on the most intuitive solution.

GLOSSARY

arousal How calm or excited you are. 11

valence The amount of attraction you have towards a certain event. 11