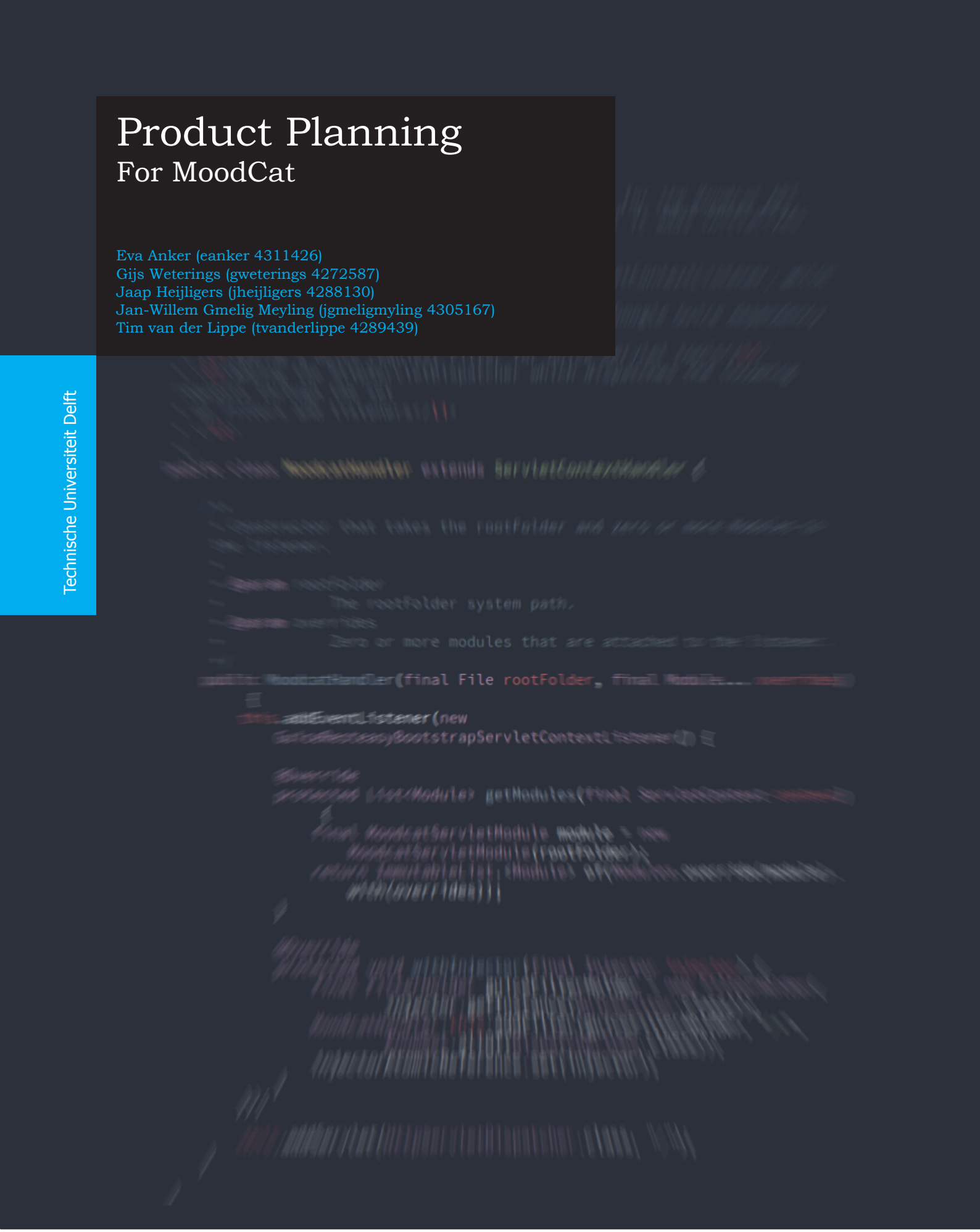
Product Planning
For MoodCat

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

Product Planning

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

Introduction

Purpose

The purpose of this document is setting a clear vision tasks and priorities for the project. This will be done, by giving general overview of how the task will be distributed over the weeks and setting the requirements for our end-product. With this document we have a solid, well defined base product. Any features extending past the scope of this document will be extra's that will be agreed upon with the stakeholders.

Eva Anker (eanker 4311426)
Gijs Weterings (gweterings 4272587)
Jaap Heijligers (jheijligers 4288130)
Jan-Willem Gmelig Meyling (jgmeligmyling 4305167)
Tim van der Lippe (tvanderlippe 4289439)
Delft, May 2015

1

Product

1.1. High-level product backlog

The MoSCoW model is used to structure the high-level product backlog. The items are listed according to the priority in relation to the other items in the backlog. The backlog is translated to an actual list of issues on GitHub (which is ordered using Waffle[1]).

1.1.1. Functional requirements

For the product MoodCat, the requirements regarding functionality and service are grouped under the *Functional requirements*. Within these functional requirements, four categories can be identified using the *MoSCoW* model [1] for prioritizing requirements:

Must haves

- The user shall interact with the product via a web interface.
- The web interface shall show a screen to log in to the application.
- The login mechanism shall be managed by SoundCloud OAuth keys (”Log in with SoundCloud”).
- The login mechanism shall show an error page when the user does not authorize MoodCat to have access to their SoundCloud data.
- The web interface shall show the user a page where they can select their current mood.
- The web interface shall show the following moods: Nervous, exciting, happy, pleasing, relaxing, peaceful, calm, sleepy, bored and sad. [2]
- The system will create and keep track of rooms, rooms consist of a currently played song and a chat.
- The system shall serve the user a list of rooms, ordered by how close they match the mood of the user.
- The web interface shall present the user with a room when it is chosen, which consists of a chat and music player.
- The platform should be able to stream music to the client.
- The system shall use an external library or API to predict the mood of a song.
- The system will use a recommender system to determine the next song to be played in each room.

[1] http://en.wikipedia.org/wiki/MoSCoW_method

[2] This list has been chosen, because it’s well spread over the grid. This is also a popular list in literature for example [**?**]

Should haves

- Users in the same room listen to the same song at the same point in time.
- A room shall contain a chat, connecting people with the same mood together.
- A room shall contain a now playing progress bar.
- The web interface shall give the user the option to mute or disconnect the current stream.
- The web interface shall notify the user if they made an illogical combination of moods, warning them results may vary.
- A room has a button to up-vote the current song, indicating the song is a good fit to the room.
- A room has a button to down-vote the current song, indicating the song is a bad fit to the room.
- When a song is down-voted the user gets buttons to tell how he would rate the song (as arousal/valence).
- The user shall be rewarded for up- and downvoting with ”treats”. These treats are achievement points.
- The user should have a button to vote to skip a song in the current room.
- The system shall retrieve more metadata about a song from AcousticBrainz[3].

Could haves

- The system shall provide a search engine that aids the user in finding a song to add.
- The web interface shall notify the user if they made an illogical combination of moods, warning them results may vary.
- If SoundCloud provides MoodCat with a purchase_url [4], This will be made available to the user.
- With enough achievement points the user can receive predefined perks, such as the ability to add a number to the room.
- The system shall periodically suggest the user to hop to another room to encourage more interaction and we can direct the user to a more preferred mood.
- If metadata is not present in the AcousticBrainz database the system shall run an analysis itself using the AcousticBrainz software.
- Visualisation for the current mood
- The web interface shall show a first-use tutorial upon logging in successfully for the first time.
- Show artist trivia information for the now playing song.

Would/Won’t haves

- We will focus on the development of a web interface, and will not build native clients for any type of device.
- The service won’t have social network integration.
- The chat won’t have a chatbot.
- The service won’t have a list of listeners.
- The service won’t have trending topics or moods.

[3] http://acousticbrainz.org/
[4] https://developers.soundcloud.com/docs/api/reference#tracks

1.1.2. Non-functional requirements

- For the project, the Scrum methodology shall be applied.
- Code shall be versioned using Git
- Git shall be used with the PR methodology: Every new feature will be developed in a branch, once completed it will be turned into a pull request. This will be reviewed and improved, and if it is accepted the branch gets merged into master.
- Frontend and Backend shall communicate using an API.
- The product shall consist of a separate frontend and backend repo, ensuring they only communicate through the API.
- The backend will be implemented in Java. The frontend web app will be built in HTML, CSS and JavaScript, using the AnuglarJS framework.
- The project shall use Continuous Integration to ensure the entire codebase is tested for every change. In this project, Travis CI will be the tool of choice.
- Code readability shall be safeguarded with the use of Checkstyle and FindBugs. Code formatting will also be agreed upon to prevent auto-formatting issues in git diffs.
- Core systems shall be unit tested with a coverage of at least 80%. In addition, integration tests can be implemented.
- Issues shall be tracked using waffle.io[5]. This ensures a Scrum style tracking of issues, while still having the overview on github.
- Every package, class and method shall be documented using JavaDoc. Javascript code will be documented using NgDoc.

1.2. Roadmap

The roadmap below shows a general overview of the goals for each SCRUM sprint in a week. This is an early planning and is expected to change.

- Week 1: Product setup
  This week we will setup our development environments and start brainstorming.
  - Set up GitHub project [2]
  - Create initial idea of MoodCat
- Week 2: Sprint 1
  This week we will focus on setting up a Graphical User Interface to let the user interact with the product. We also concretized our idea further, and wrote an initial product vision.
  - Set up initial version of frontend.
  - Write product vision draft.
- Week 3: Sprint 2
  This week we will set up a connection between the frontend and backend system.
  - Create various APIs on the backend.
  - Develop frontend to backend connection.
  - Develop interface on frontend to use API.

[5] http://www.waffle.io

- Week 4: Sprint 3
  The first version of the music-matching algorithm will be created. [6]
  - Choose songs to use as test data.
  - Create testcases to measure the performance of our algorithm.
  - Design the neural network.
  - You can vote how good or bad a song is.
- Week 5: Sprint 4
  The connection between the frontend and backend will be adjusted. We will get a basic overview of to to create an algorithm that will provide song suggestions for the room.
  - Make API's for the frontend (chat and general room).
  - Connection between API's and backend.
  - Make a user classification system.
  - Use nearest neighbour to find rooms that suits a mood.
- Week 6: Sprint 5
  The music-matching algorithm will be made. We also review our own code and test quality, since we're half way.
  - Review code and test quality.
  - Make the matching algorithm between users and rooms.
- Week 7: Sprint 6
  We will apply SIG feedback and implement an auto room creation system.
  - Apply SIG feedback.
  - Implement auto room creation system.
- Week 8: Sprint 7
  Finish documentation and start final report.
  - Finish architecture design document.
  - Create final report.
- Week 9: Sprint 8
  Finish final report and prepare presentation.
  - Finish final report.
  - Prepare and practise presentation.

[6] We found out making the music matching algorithm was not feasible. We discussed with our stakeholder and decided to let it go.

2

Product backlog

2.1. User stories of features

As a user,
When I start the app,
Then I should see a mood-selection page.

As a user,
When I select a mood,
Then I should see a list of rooms.

As a user,
When I select a room,
Then the current song should be played and I should see a chat box with the latest messages that have been sent in this room .

As a user,
When I type in the chat message input field,
And I click on 'Send message',
Then the message should be send to the backend.

As a user,
When a user in my current room sends a message to the backend,
Then I should see that message in the chat box.

As a user,
When I click on the 'thumbs up' or 'thumbs down' buttons,
Then the system should alter according to the feedback to enhance the song-selection algorithm.

As a user,
When I click on the 'mute' button,
Then the music should be muted.

2.2. Initial release plan

Each sprint has a corresponding milestone. For example, if a feature will be developed during sprint 3, the corresponding GitHub issue will have milestone 3. This makes sure that the roadmap outlines are closely followed and progress can be easily determined by using the GitHub tools.

3

Definition of Done

We need to set a definition of done. This entails that for every feature, sprint and the end product, we will define what we consider ”done”.

A feature is completed when it is implemented according to its specification and the code is well tested using unit tests. Furthermore, the public API (the public methods) of introduced classes should be well-documented using JavaDoc or NgDoc and follow our checkstyle and markup rules. Finally, the feature must be reviewed by at least one team member that was not directly involved in creating the code and tests. If all these prerequisites are met, the pull request can be merged.

We will use sprint iterations of one week. All stories assigned for a sprint should be done at the end of a sprint. If a story could not be completed in the sprint, it will be copied to the next sprint and this will be discussed in the sprint reflection in order to prevent planning failures in the future. The system also has to be manually tested for the main interaction with the system. Lastly, integration tests are created and ran successfully.

The final product is done when all requirements are fulfilled and the implementation matches the specification. The implementation should be well tested using unit tests and preferably integration / end-to-end tests as well. If a should-have requirement is not implemented, this has to be agreed upon by all team members with a valid reason, and the product should still be usable without the feature. The code base is formatted and documentated in a consistent manner and validated by the SIG test

Glossary

AcousticBrainz A database that crowd sources music to get the acoustic information from a song. 2
AnuglarJS A webapplication framework designed for applications with a single page. 3
API A set of routines, protocols, and tools for building software applications. 3

checkstyle A development tool that points out mistakes in Java code. 6
CSS Cascading Style Sheets is a style sheet language used for describing the look and formatting of a document written in a markup language. 3

GitHub An online revision control service for sharing and deploying code and thus software. 1

HTML HyperText Markup Language is the standard markup language used to create web pages. 3

JavaDoc A documentation generator for Java. 6
JavaScript A dynamic programming language. It is most commonly used as part of web browsers, whose implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed. 3

NgDoc A documentation generator for AngularJS. 6

SCRUM SCRUM is an agile software development method to manage product development. 3
SIG A company that reviews the quality of code. 4, 6

Travis CI A continuous integration service used to build and test projects. 3

Bibliography

[1] R. Software, *Waffle.io,* (2015).

[2] MoodCat, *Moodcat github repository,* .