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

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

TU Delft Delft University of Technology

Contents

Introduction

0.1. Purpose

The purpose of this document is setting a clear vision tasks and priorities for the project. 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 a first-use tutorial upon logging in succesfully for the first time.
- 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
  - Exiting
  - Happy
  - Pleasing
  - Relaxing
  - Peaceful
  - Calm
  - Sleepy
  - Bored
  - Sad
- The system shall serve the user a list of rooms, ordered by how close they match the mood of the user.

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

- The web interface shall present the user with a room when it is chosen.
- The platform should be able to stream music to the client
- The system shall use a multi-layer perceptron system 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.

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 slider.
- 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 upvote the current song, indicating the song is a good fit to the room.
- A room has a button to downvote the current song, indicating the song is a bad fit to the room. This will also vote to skip the song.
- A song shall be skipped in one of the following situations:
  - The song received $\frac{\$number\_of\_users\_in\_room}{3}$ votes to skip.
  - The song received at least 10 votes, and the ratio upvote:downvote is 1:2 or less.
- The multi-layer perceptron will be initialized by generating values from the metadata in our dataset.
- The upvotes and downvotes for a song change the weights of the multi-layer perceptron.
- The user shall be rewarded for up- and downvoting with "treats". These treats are achievement points.
- The system shall retrieve more metadata about a song from AcousticBrainz[2].

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 [3], 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
- Show artist trivia information for the now playing song.

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

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.

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 in a RESTful way, using an API.
- The product shall consist of a seperate frontend and backend repo, ensuring proper RESTful implementation is used.
- The backend will be implemented in Java. The frontend web app will be built in HTML, CSS and JavaScript, using the AngularJS framework.
- The project shall use Continuous Integration to ensure the entire codebase is tested for every change. In this project, Travic 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
- Issues shall be tracked using waffle.io[4]. 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. Angular controllers must 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.

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

- 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 APIs.
- Week 4: Sprint 3
  The first version of the music-matching algorithm will be created using Test-Driven Development (TDD) [**?**].
  - Choose songs to use as test data.
  - Create testcases to measure the performance of our algorithm.
  - Create initial version of algorithm.
- Week 5: Sprint 4
  The music-matching algorithm will be extended.
  - Assess the quality of the initial algorithm.
  - Incorporate user feedback in the algorithm.
- Week 6: Sprint 5
  The music-matching algorithm will be tweaked and extended. We also review our own code and test quality, since we're half way.
  - Review code and test quality.
  - Improve Algorithm
- 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.

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.

As a user,
When I select a room,
Then I should see a chat box.

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.

As a user,
When I join a room,
Then I should see the latest messages sent in this room.

2.2. USER STORIES OF DEFECTS (IF APPLICABLE)

As of the time of writing, no defects have been discovered.

2.3. USER STORIES OF TECHNICAL IMPROVEMENTS (IF APPLICABLE)

As of the time of writing, no technical improvements were made.

2.4. USER STORIES OF KNOW-HOW ACQUISITION

2.5. INITIAL RELEASE PLAN

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 succesfully.

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. Code is consistent, validated by the SIG test and documented.

4 Glossary

Bibliography

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

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