Requirements and Analysis Document for FlashPig

Jesper Bergquist Wendy Pau Madeleine Xia
Salvija Zelvyte

October 16, 2020

Contents

1 Introduction **1**
- 1.1 Definitions, acronyms, and abbreviations . . . 1

2 Requirements **2**
- 2.1 Implemented User Stories . . . 2
  - 2.1.1 Functional Requirements . . . 2
  - 2.1.2 Non-functional requirements . . . 4
- 2.2 Unimplemented User Stories . . . 5
  - 2.2.1 Extra functional requirements . . . 5
- 2.3 Definition of Done . . . 6
- 2.4 User interface . . . 7

3 Domain model **12**
- 3.1 Class responsibilities . . . 13
  - 3.1.1 Card . . . 13
  - 3.1.2 Deck . . . 13
  - 3.1.3 Difficulty . . . 13
  - 3.1.4 Flashcard . . . 13
  - 3.1.5 GameLogic . . . 13
  - 3.1.6 Pairup . . . 13
  - 3.1.7 FakeDatabase . . . 14
  - 3.1.8 GridSpacingCardDecoration . . . 14
  - 3.1.9 MainActivityPairUp . . . 14
  - 3.1.10 MainActivityFlashcard . . . 14
  - 3.1.11 FlashcardFragmentStart . . . 14
  - 3.1.12 FlashcardFragmentEnd . . . 14
  - 3.1.13 PairUpFragmentStart . . . 14
  - 3.1.14 PairUpFragmentEnd . . . 14
  - 3.1.15 EditDeckFragment . . . 14
  - 3.1.16 DeckSpinnerAdapter . . . 14
  - 3.1.17 DeckRecyclerViewAdapter . . . 14
  - 3.1.18 PairUpViewModel . . . 14
  - 3.1.19 FlashcardViewModel . . . 14
  - 3.1.20 EditDeckViewModel . . . 14
  - 3.1.21 CardViewModel . . . 14

4 References **14**

1 Introduction

Flashpig is an application that provides the opportunity to create flashcards and use them to study and learn in three different ways.

The first way is "Flashcard" which consists of two sided cards, where one side has a question while the other has an answer, both sides can be appeared as picture or text. After each card the user has to decide which degree of difficulty the card had (easy, medium or hard). This makes the learning progress visible and accessible for the user. Furthermore, depending on what difficulty the user chose the cards will be shown in different interval of time which is based on an algorithm. As a result the "flashcard progress" will be based on the chosen complexity and will be available on the homepage for the user to check their score.

In addition there is also two other games named "Memory" and "Pair up". "Memory" works exactly like the classic memory game where in this case you have to combine front- and backside. However, "Pair up" makes all the cards revealed and gives the user multiple possible answers to match with the questions. Therefore the front sides (questions) is located on the left column and backsides (answers) on the right, all cards are in random order to increase the difficulty for the user.

The application is mainly designed for students but will work for any other user that want to learn new information in a quick, effective and fun way.

1.1 Definitions, acronyms, and abbreviations

Flashcard
Consists of a two-sided card where each side holds information through either some text and/or images.

(("Flashcard" consists of two-sided cards where each side consist of information in either texts or images. The user can choose whether to show the front/backside of the card first and then flip the card. The user can for each card choose what difficulty (easy, medium, hard) they thought about the card. This information will be used to show a "Flashcard Progress" which will be accessible from the dashboard.))

Pair Up
Pairs two cards with each other from a deck as the player studies the information on the card.

Memory
Memorisation game that trains the players memory as well as studying the subject of the chosen deck.

Java
A platform independent programming language the application is made of.

GUI
Graphical user interface.

UML
Unified Modeling Language. UML is used for visualising the design of the system, both on high level and low level.

MVVM
Model-View-ViewModel. A software architectural pattern used in FlashPig.

Mobile app
Application for mobile devices. //BEHÖVS ENS DENNA?

User
Any user of the FlashPig application.

Flashcard Progress
Allows the user to follow the progress for a chosen deck.

Deck
A collection of cards that belongs together.

Spaced repetition
An algorithm to handle the frequency for a specific card to show up based on the users previous experience with the card.

2 Requirements

2.1 Implemented User Stories

The following list is the User Stories for the application which has been implemented.

2.1.1 Functional Requirements

1. **As a user, I want to be able to create new decks of cards to gather information that I want to learn.**

   Estimated time: 2 days.

   (a) Design a GUI to create new decks.

   (b) Be able to add how many cards the user wants.

(c) Give each deck an id and let the deck know how many cards it have.

Acceptance criteria:

- When the user can create a new deck and name it.

2. **As a user, I would like to edit my decks that I've already created.**

   Estimated time: 3 days.

   (a) Design a GUI to be able to edit decks.
   (b) Be able to delete decks.
   (c) Be able to change the cards text.
   (d) Be able to add/delete pictures in the cards.
   (e) Be able to create new cards to a deck.
   (f) Be able to change a decks name.

   Acceptance criteria:

   - When a deck is editable after its creation.

3. **As a user, I would like to play Flashcard so that I can study in an effective way.**

   Estimated time: 4 days.

   (a) Design an GUI for the Flashcard game.
   (b) Connect a deck to the game.
   (c) Ability to choose each cards difficulty (easy, medium, hard).
   (d) Be able to return to the game where the user left it.
   (e) Create a popup message to check if the user wants to leave the game.
   (f) Create a progress page where the user can see its performance in Flashcard.

   Acceptance criteria:

   - Can iterate through a deck.
   - Can flip the card to show the backside.
   - When the game saves after one round (even after unfinished round).

4. **As a user, I would like to play Memory so that I can train my memory.**

   Estimated time: 4 days.

   (a) Design an GUI for the Memory game.

(b) Connect a cards back/front-side with each other.

(c) Add a time taker.

(d) Create a logic where only 8 cards are shown in a time.

Acceptance criteria:

- The user can flip the card.
- The user can see how many pairs that's left under game.
- The user can see when the game is over.
- The user can see how much time it took for them to finish the game.

5. **As a user, I would like to play Pair Up to learn information in a playful way.**

   Estimated time: 4 days.

   (a) Design an GUI for the Pair up game.

   (b) Connect a cards back/front-side with each other.

   (c) Add a time taker.

   (d) Define when the game is over.

   Acceptance criteria:

   - The user can see when they clicked wrong or correct.
   - The user can see when the game is over.
   - The user can pair 2 cards together.

2.1.2 Non-functional requirements

1. **As a user, I would like the application to be beautiful and therefore give an better user experience.**

   Estimated time: NA.

   (a) Use suitable font, colours and shapes in the GUI.

   (b) Use uniform colours in the application.

   (c) Create Flashpig sprites.

   Acceptance criteria:

   - The colours are balanced.
   - The applications appearance are in line with the theory about how a good interface are.

2. **As a user, I would like the app to be user friendly to avoid spending time on navigating the application.**

   Estimated time: NA.

   (a) Implement help system designs or add "first time"-tutorials.
   (b) Make the design intuitive to use with help of icons, images etc.
   (c) Implement frequently used design patterns.

   Acceptance criteria:

   - Implements help systems.
   - Implements relevant design patterns.
   - The application acts as expected.

3. **As a user, I would like the application to be secure to use so that I can feel secure while using the application.**

   Estimated time: NA.

   (a) Be able to choose if a deck should be public or private,
   (b) **Ask about access from the user when the application access to the users camera, gallery etc. before use.**

   Acceptance criteria:

   - The application ask for permission to access to the users information beforehand.
   - When the application is optimised for security.

2.2 Unimplemented User Stories

The following list is the user stories for the application that are yet to be implemented. The concerning user stories are extended functionality for the application that improves the user experience.

2.2.1 Extra functional requirements

1. **As a user, I would like to be able to share my decks with other people.**

   Estimated time: 4 days.

   (a) The user shall be able to choose if they want to create public or private cards.
   (b) Create a downloadable link to a deck.

Acceptance criteria:

- The user can share its deck with other people.

2. **As a user, I would like to have access to others public decks.**

   Estimated time: 4 days.

   (a) Be able to see others public decks.
   (b) Be able to create a copy and save others decks as my own.

   Acceptance criteria:

   - The user have access to others public decks.
   - The user can copy/save others decks as their own.

3. **As a user, It would be fun to learn my flashcards through a mini quiz game.**

   Estimated time: 4 days.

   (a) Create flashpig sprite.
   (b) Create the game world.
   (c) Create the quiz logic.

   Acceptance criteria:

   - The player can play the game.
   - The player can choose which deck to use.

2.3 Definition of Done

- The application should be original and not use others design without their permission.
- The code should be tested and thoroughly checked.
- The code should be runnable without bugs that ruins the users user experience.
- The application should have tested all the user stories.
- All the functions should work as the user expects it to be.

2.4 User interface

Start screen

Upon launching the screen, the screen will first show a splash screen (see fig.1) before taking the the user to the start screen/dashboard (see fig.2). Here, the user can see how many decks it has created, edit a deck and it has access to the FAQ and all the mini games. To be able to continue, the user needs to first choose a deck.

Figure 1: Splash screen

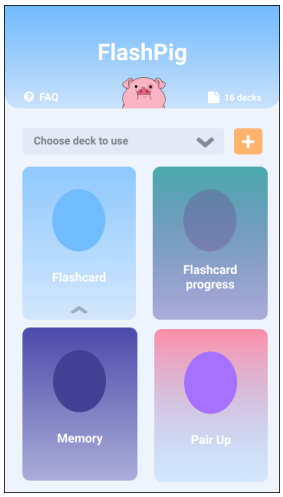

Figure 2: Dashboard

Create a new deck

To create a new deck the user needs to click on the distinct yellow plus button, where the user will then be taken to a new view (see fig.3). The user needs to first name the deck (see fig.3) and thereafter the user can create cards by choosing a picture and/or a text for the front and backside and later save the deck. (see fig.5).

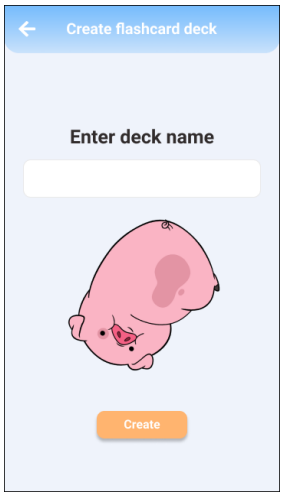

Figure 3: Create deck

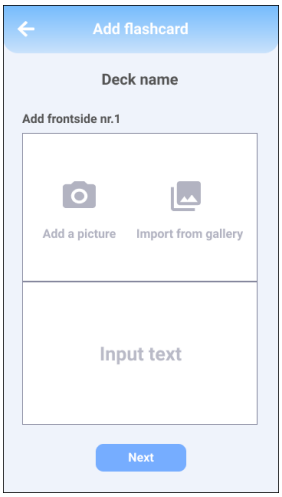

Figure 4: Create the cards front side

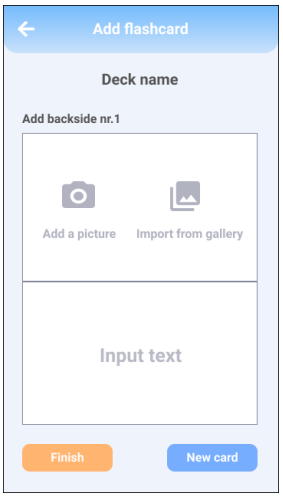

Figure 5: Create the cards front side

Edit deck

By clicking on the drop-down arrow in the combo box, further options are displayed for the user to use (see fig.6). By clicking on the edit-button the user are taken to the page for editing a deck and its cards (see fig.7). The prominent plus-button is now used to create a new card and each created card can be edited from this page (see fig.8). The user can also change which deck to edit from this page.

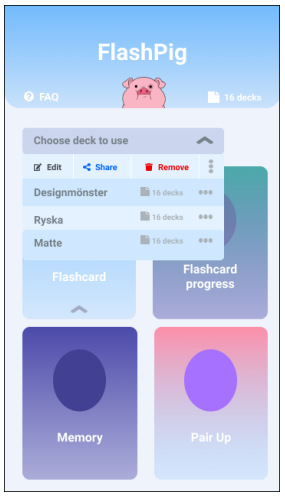

Figure 6: Options for deck

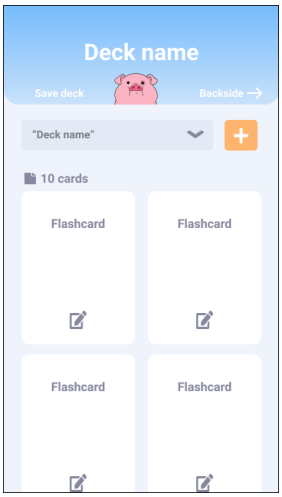

Figure 7: Page for editing a deck

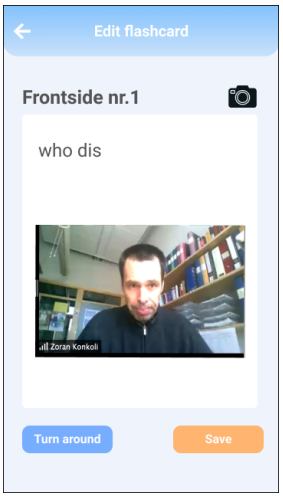

Figure 8: Page for editing a card

Mini-games

By clicking on one of the mini-games, a start button is shown. The user is taken to respective mini-game by clicking start and from there the user can either continue playing the game or return to the dashboard through the back-button.

When playing Flashcard, the user is shown a cards front side and upon clicking on the card (see fig.9), the card will flip and show the cards content on the back (see fig.10). The user can then choose how they thought the difficulty of the card is and thereafter, move on to the next card. If there are no more cards that the user thinks are hard, the round will end and the user can then either restart or go to the main screen (see fig.18).

Figure 9: Flashcard

Figure 10: Flashcard difficulty

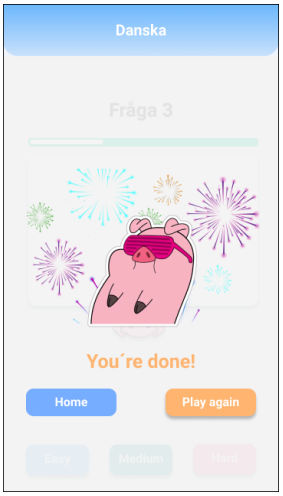

Figure 11: When the round is over

When playing Memory, the user flip cards through clicking on them and then if the cards do not match, it will turn over but if it is the other way, the card will stay as they are. A game is over when all of the cards have been matched and the user can choose the either play again or return to homepage.

Figure 12: Memory

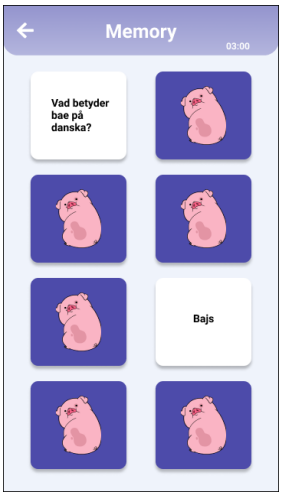

Figure 13: Memory flipped cards

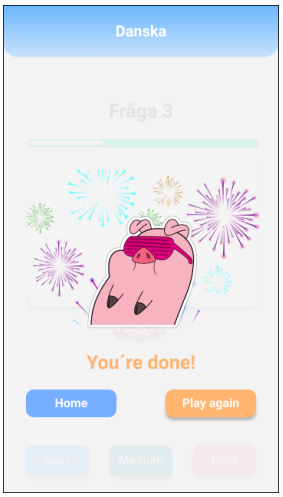

Figure 14: When the round is over

When playing Pair Up, the user will pair two cards on the board. If they are a pair, a green frame will appear. Otherwise, a red frame will appear. When all of the cards have been paired up, the game round is over and the user can choose to start again or return to homepage.

Figure 15: Pair Up

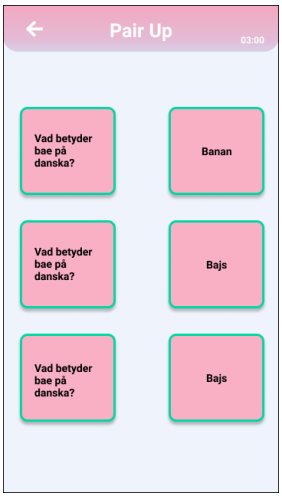

Figure 16: Pair Up right answer

Figure 17: Pair Up wring answer

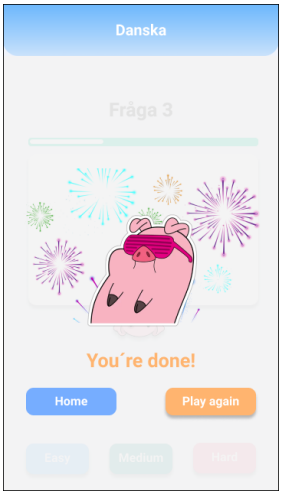

Figure 18: Pair Up round is over

3 Domain model

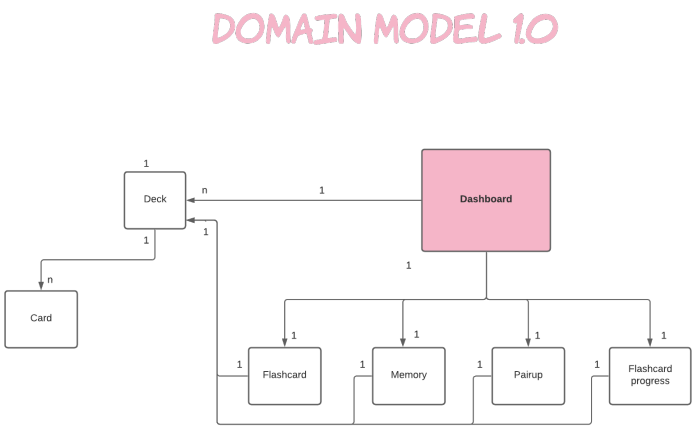

Figure 19: Iteration 1

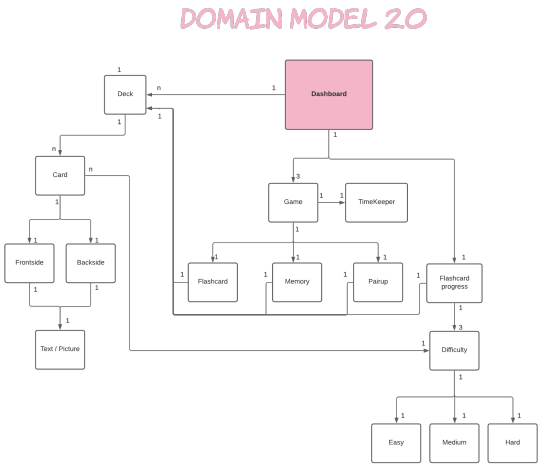

Figure 20: Iteration 2

3.1 Class responsibilities

3.1.1 Card

Card is the class that holds information that belongs to card, a card has id, a boolean called is frontside to differ between sides of the card, a string for the front and a string for the back. And two bitmap for the front and back to store the pictures. And a Enum difficulty to set a difficulty for the card. The class also holds a Random to generate a random id. The class is also made parceble.

3.1.2 Deck

Deck is a class that holds information that belongs to deck. it holds a int for the id, a list of cards, a string for the deck name, an int to hold the amount of cards in the deck and a boolean to check if the deck is empty or not. The class is also made parceble. The class also holds a Random to generate a random id.

3.1.3 Difficulty

Difficulty is a Enum-class that holds different difficulties for the cards. Easy, Medium, Hard and nothing.

3.1.4 Flashcard

Holds logic for the flashcard game.

3.1.5 GameLogic

Gamelogic is a abstract class that holds some values and methods that is needed commonly for the minigames.

3.1.6 Pairup

Holds the logic for comparing two cards from a deck and checks if a game is done.

3.1.7 FakeDatabase

3.1.8 GridSpacingCardDecoration

3.1.9 MainActivityPairUp

3.1.10 MainActivityFlashcard

3.1.11 FlashcardFragmentStart

3.1.12 FlashcardFragmentEnd

3.1.13 PairUpFragmentStart

3.1.14 PairUpFragmentEnd

3.1.15 EditDeckFragment

Holds the logic for all the clickable objects in EditDeck.xml. The class also includes instances of the adaptors: DeckRecyclerViewAdapter and DeckSpinner

3.1.16 DeckSpinnerAdapter

3.1.17 DeckRecyclerViewAdapter

3.1.18 PairUpViewModel

3.1.19 FlashcardViewModel

3.1.20 EditDeckViewModel

3.1.21 CardViewModel

4 References