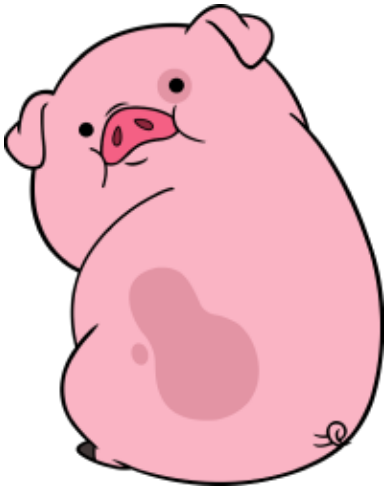
Requirements and Analysis Document for FlashPig

Jesper Bergquist, Wendy Pau, Madeleine Xia and Salvija Zelvyte

October 22, 2020



Contents

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 the game "Flashcard" which consists of two-sided cards, where one side holds a question while the other holds the answer. Both sides can be appeared as an image or text. After each card the user has to decide how difficult the card was (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 appear in the game in different intervals of time which is based on an algorithm, giving this study technique a spaced repetition.

Moreover, there are two other games named "Memory" and "Pair up". "Memory" works exactly like the classic memory game where in this case you have to combine the front- and backside. On the other hand, "Pair up" displays all the cards information and gives the user multiple possible answers to match the associated card. Therefore, the front sides (questions) are located on the left column and backsides (answers) on the right.

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. The user can choose whether to show the front- or backside of the card 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.

- **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.
- **UI** - User interface.
- **LiveData** - An observable data holder class. Unlike a regular observable, LiveData is lifecycle-aware, meaning it respects the lifecycle of other app components, such as activities, fragments, or services. This awareness ensures LiveData only updates app component observers that are in an active lifecycle state.

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 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 card's 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. (Unimplemented)

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).

3. **As a user, I would like to edit my decks that I have 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.

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

   Estimated time: 4 days.

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

   (b) Connect a cards back/frontside with each other.

   (c) 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 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. **As a user, I would like the app to be user friendly to avoid spending time on learning how to navigate in 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 beautiful and therefore give a 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.

2.2 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 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.

2. **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.

3. **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.

4. **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 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 Figure**??**) before taking the the user to the start screen/Dashboard (see Figure2). Here, the user can see how many decks has been created, edit a deck and 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

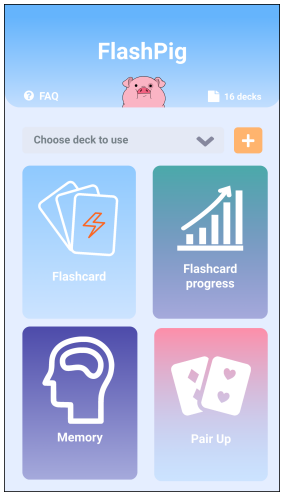


Figure 2: Dashboard

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. The user needs to first name the deck 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 Figure 3.

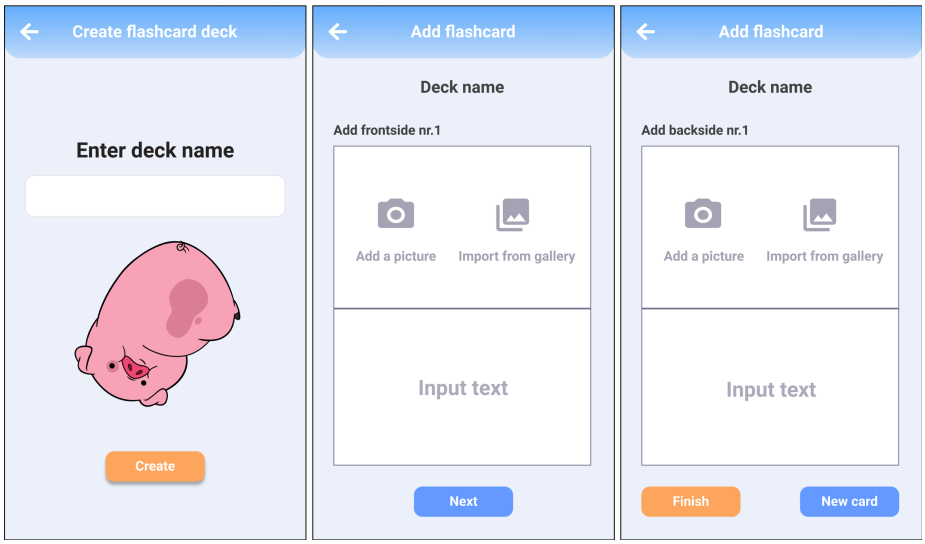


Figure 3: Views for creating a card

Edit deck

By clicking on the drop-down arrow in the combo box, further options are displayed for the user to use. By clicking on the edit-button the user are taken to the page for editing a deck and its cards. The prominent plus-button is now used to create a new card and each created card can be edited from this page. The user can also change which deck to edit from this page. See figure 4.

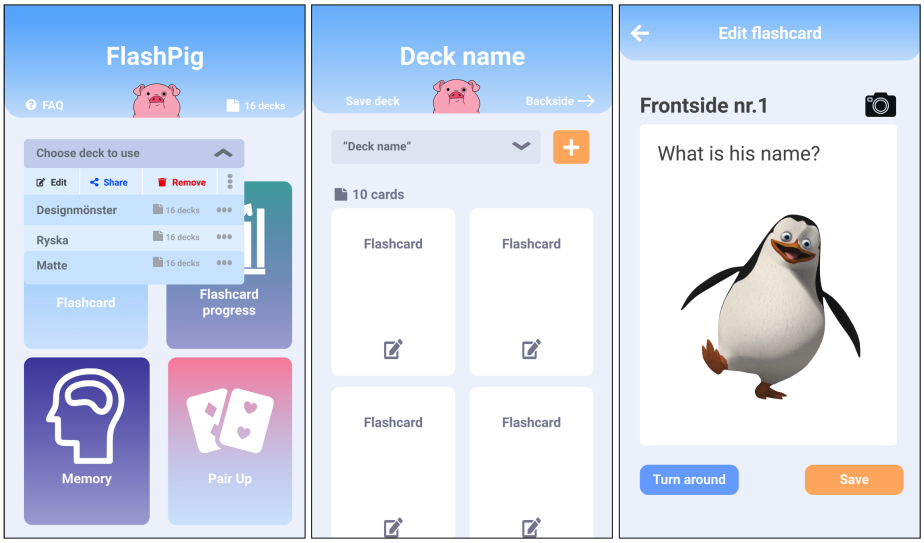


Figure 4: Edit a card

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, the card will flip and show the cards content on the back. 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 figure 5.

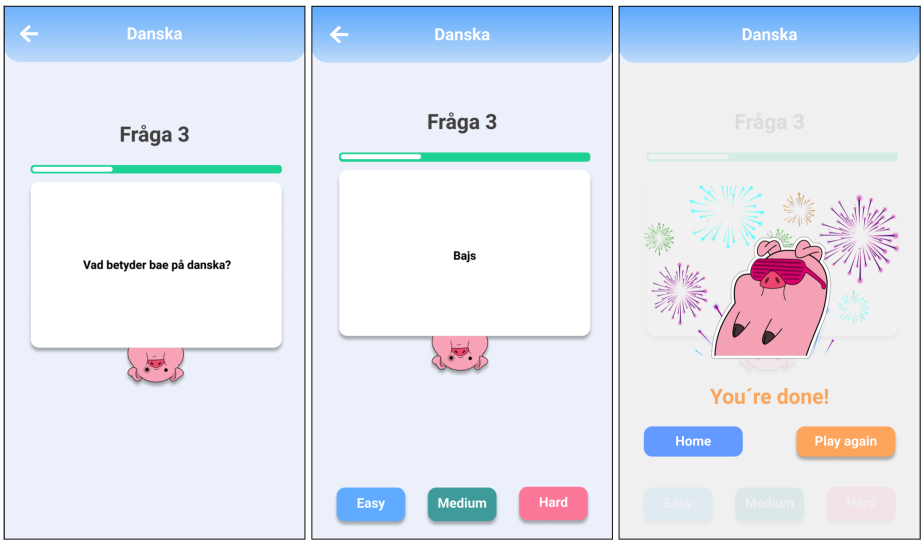


Figure 5: Flashcard game

When playing Memory, the user flip cards by 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. See Figure 6.

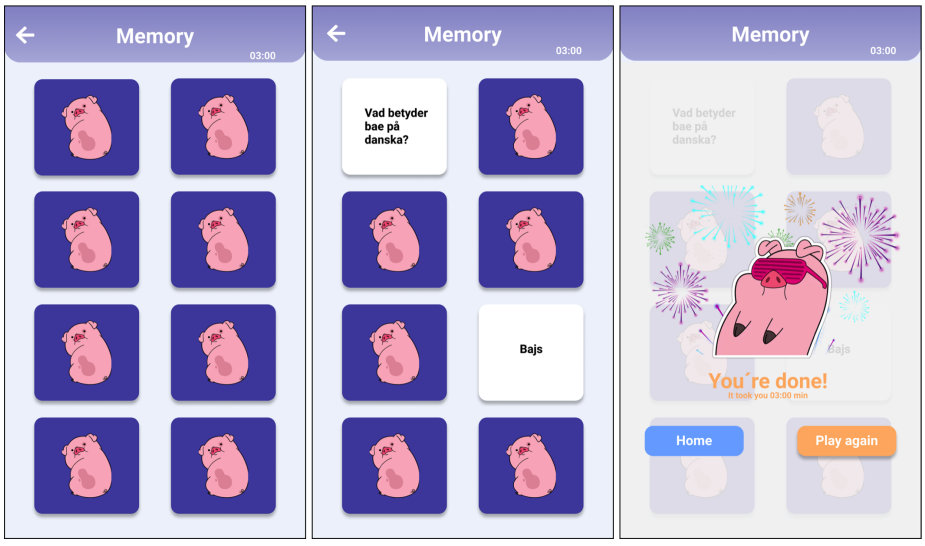


Figure 6: Memory game

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. See Figure 7.

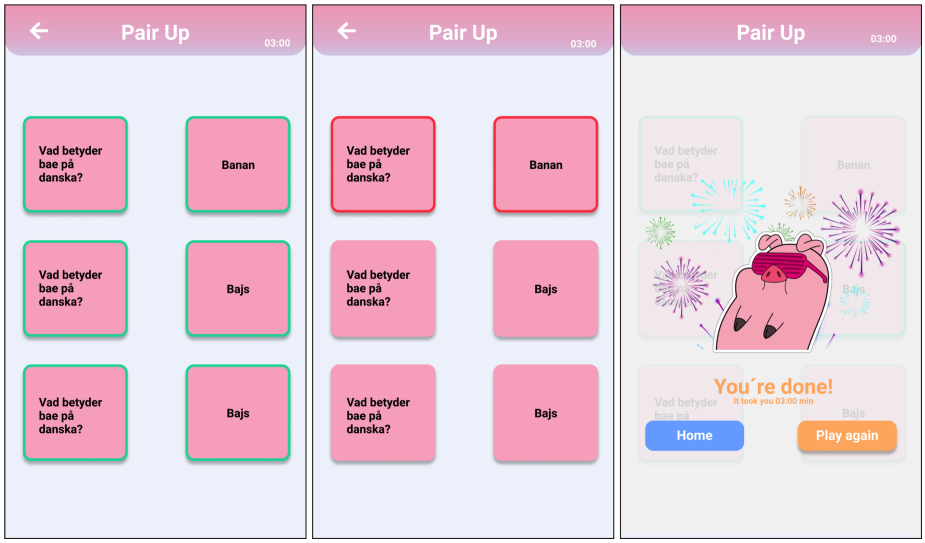


Figure 7: Pair Up game

3 Domain model

The following figures is the UML of the Domain model diagram in the first and last iteration.

3.1 Iteration 1

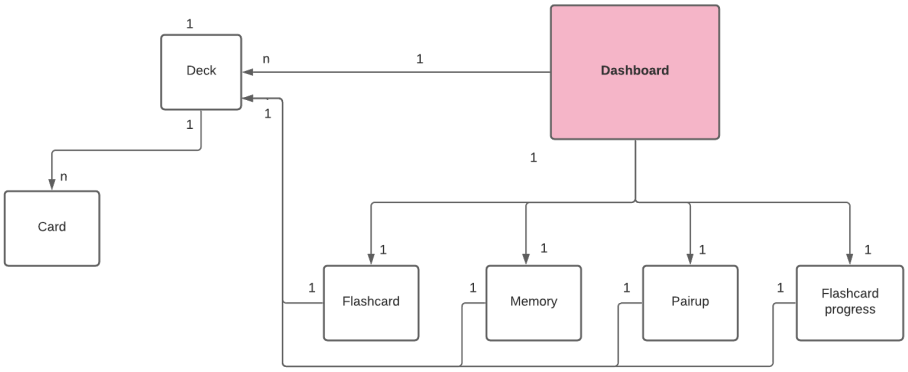


Figure 8: Domain Model iteration 1

3.2 Iteration 2

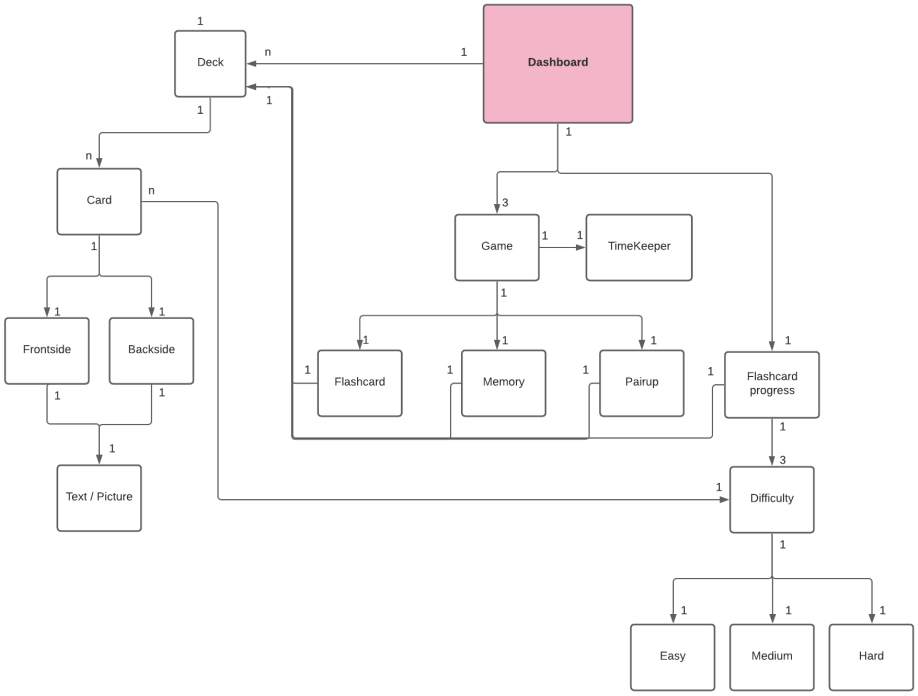


Figure 9: Domain Model iteration 2

4 Class responsibilities

4.1 Card

Card is the model class that holds information that belongs to card, a card has an id, a boolean called isFrontside to differ between sides of the card, a string for the front and a string for the back. The class have two bitmaps 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 parcelable.

4.2 Difficulty

Difficulty is a Enum-class that holds different difficulties for the cards: EASY, MEDIUM, HARD and NOTHING.

4.3 CardViewModel

ViewModel class for creating decks and cards. The purpose is to work as a bridge between the Card and Deck (Model classes) and CardFragment and CreateDeck-Fragment (View class). This class gets data from the users interaction with the views and updates the model classes with them.

4.4 Deck

Deck is a class that holds information that belongs to deck. It has 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 and holds a Random to generate a random id for the deck.

4.5 EditDeckFragment

Holds the logic for all the clickable objects in EditDeck.xml. The class also includes instances of the adapters: DeckRecyclerViewAdapter and DeckSpinnerAdapter. The DeckRecyclerViewAdapter is used to access all the required information about the card objects that populates the RecyclerView. Same goes for the DeckSpinnerAdapter, the instance gives all the needed access for the deck objects that populates the spinner.

4.6 DeckSpinnerAdapter

This adapter is used to inflate the spinner drop-down view and views with decks, amount of cards in each deck and buttons for removing or sharing a deck in the spinner. It also makes sure that the drop down view implements the row striping pattern by alternating colours on each row. The class includes also an instance of the interface onEditItemsClickListener which also is located inside the adapter. The interface is implemented by the two fragments EditDeckFragment and DaashboardFragment, it makes it possible for these classes to override methods that are used for the common spinner.

4.7 DeckRecyclerViewAdapter

This adapter is used to inflate the selected decks' cards in EditDeckFragment. Once a deck is selected in the spinner the decks list of cards deck will appear in the recyclerView below. The adapter includes an inner class called Edit-DeckViewHolder(), mainly used to populate the cardViews with all the required buttons, texts and images.

4.8 GridSpacingCardDecoration

This class has the main purpose to locate the cards in desired amount of columns and spacing between the objects. Mainly used in EditDeckFragment and PairUpFragment.

4.9 FakeDatabase

The FakeDatabase holds already created decks with cards in them that are used throughout the application.

4.10 Repository

Repository handles the data from the FakeDataBase and works as a wrapper. The wrapped data will be sent to whoever calls on it.

4.11 DashboardViewModel

ViewModel class for DashboardFragment and EditDeckFragment, This class is responsible for knowing which deck the user selects and updating the Card or Deck model classes if the user wants to edit their decks or cards. This is done through MutableLiveDatas which is observed by the mentioned Fragments. The class also separates the

4.12 GameLogic

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

4.13 PairUp

The model class for the Pair Up game. The class holds the logic for comparing two cards from a deck and checks if a game is done.

4.14 PairUpViewModel

The ViewModel class for PairUp game. The class gets input from user and controls if the cards are an pair, if the cards are the last cards that are shown or if the user has paired all the cards in the deck.

4.15 MainActivityPairUp

The class holds the logic for initialising the Pair Up activity by defining the UI and retrieving the widgets in the UI.

4.16 PairUpFragmentStart

The View class for the Pair Up game that shows six cards for the user to pair. If the game is over, the class will navigate the user to PairUpFragmentEnd.

4.17 Flashcard

The model class for the Flashcard game. This class is also responsible for creating a game deck which is a copy of the selected deck and setting the cards difficulty.

4.18 FlashcardViewModel

The ViewModel class consist of LiveDatas of the Flashcard class. The class is responsible for iterating through the selected deck and updating the Flashcard class of the users selection of difficulty. If the user chooses easy then the card will temporally be removed from the game deck for ....

4.19 MainActivityFlashcard

The class is responsible for setting the title for the toolbar and verifies if the user wants to quit the game. The class is also responsible for navigating through the flashcards fragments.

4.20 FlashcardFragmentStart

The view that observes the changes of the Flashcard model by the FlashcardViewModels' livedata and updates it views by the changes. The class also provides the FlashCardViewModel of the users selection of difficulty on a card.

4.21 FlashcardFragmentEnd and PairUpFragmentEnd

The view that shows up when the user has completed the game. Here, the user can chose to play again or return to the home page.

5 References

Figma: GUI
figma.com/file/FVwpn4oMpd4NVaKKtvg2qP/Startpage?node-id=0%3A1

FlashPig code: GitHub
github.com/Wendaaj/FlashPig

Lucidchart: Domain model