Requirements and Analysis document for AWME bokbytarapp

Ali Alladin, Magnus Andersson, William Hugo, Elias Johansson

2020-10-23

1.0

1 Introduction

The buying and selling of course literature between students at Chalmers University is today mainly done through different facebook groups. This android application is designed to be a better alternative to this. Since the product is free and non company affiliated, the stakeholders have been regarded as the end user. The application allows sellers to post their books for free and buyers to search for their desired books and contact sellers if they find something desirable. It also mediates contact between potential buyers and sellers through a messaging service in the app. All the messages are encrypted with a self made RSA cryptography.

1.1 Definitions, acronyms, and abbreviations

Listing: Refers to all the items that are being sold in the app.
RSA cryptography: Rivest–Shamir–Adleman cryptography.

2 Requirements

2.1 User Stories

User story - Done

Story Identifier: G001
Story Name: Buyer Search
Priority: High

Description

As a buyer, **I want to** be able to search for books by title **so that** I can find the books I am looking for.

Acceptance criteria

- Functional:
    - Can I search by title?
    - Can I interact with the search result?
    - Am I informed if my search returns no results?
- Non-functional:
    - Availability:
        - Can I search anytime of the day?

User story - Done

Story Identifier: G002
Story Name: Seller Post
Priority: High

Description

As a seller, **I want to** post a listing for my book **so that** other users can buy it

Acceptance criteria

- Functional:
    - Can I post my book?
    - Can I edit my listing after the fact?
    - Can I remove my listing?
    - Can I add a topic to my post?
- Non-functional:
    - Security:
        - Are unauthorised persons prevented from editing my listings?

User story - Done

Story Identifier: G003
Story Name: See Listings
Priority: High

Description

As a user, **I want to** be able to see an infinity list with listings, be able to scroll through it and click a card of interest to see more **so that** the app is easy to get started with

Acceptance criteria

- Functional:
    - Can I see a list?
    - Can I scroll the list?
    - Can I click a card in the list to see more about it?

User story - Done

Story identifier: G004
Story name: Login
Priority: Low

Description

As a user, **I want to** be able to log in to my account **so that** I can use the app securely

Acceptance criteria

- Functional:
    - Can I log in?
    - Can I log out

User story - Done

Story Identifier: Y001
Story Name: Buyer-Seller Contact
Priority: Low

Description

As a buyer, **I want to** be able to contact sellers **so that** I can buy the book

Acceptance criteria

- Functional:
    - Can I message the seller?
- Non-functional:
    - Availability:
        - Will my message reach a seller once they log in if they are not currently logged in?
        - Security: Will messages only be seen by me and the person I'm messaging?

User story - Done

Story Identifier: Y002
Story Name: Sort
Priority: Low

Description

As a user, **I want to** be able to sort books by subject/course **so that** I can find relevant listings

Acceptance criteria

- Functional:
    - Can I sort by price?
    - Can I sort by topic?
    - Can I sort by course?
    - Can I sort by date?

User story - Done

Story Identifier: Y003
Story Name: Selling List
Priority: Low

Description

As a seller, **I want to** be able to access a list of books that I'm selling **so that** I have an easy way to view all my listings

Acceptance criteria

- Functional:
    - Can I browse all of my listings?

User story - Unfinished

Story Identifier: Y004
Story Name: Selling notebooks
Priority: Low

Description

As a seller, **I want to** be able to post my notebooks from previous courses in the app **so that** I can sell them

Acceptance criteria

- Functional:
    - Can I post a notebook listing?
    - Can I specify which course the notebook was written for?

User story - Done

Story Identifier: O001
Story Name: Saved-list
Priority: Low

Description

As a buyer, **I want to** be able to save books **so that** they're easier to find in the future.

Acceptance criteria

- Functional:
    - Can I add a listing to my saved-list?
    - Can I access my saved-list easily?
    - Can I delete listings from my saved-list?

User story - Unfinished

Story identifier: R001
Story name: Seller Review
Priority: Low

Description

As a user, **I want to** be able to read reviews of a seller **so that** I know which seller to choose.

Acceptance criteria

- Functional:
    - Can I access a seller's reviews?
    - Can I review the seller?

User story - Done

Story Identifier: R002
Story Name: Extended Buyer Search
Priority: Low

Description

As a buyer, **I want to** be able to search for books by things other than titles, such as ISBN **so that** I can find the books I am searching for.

Acceptance criteria

- Functional:
    - Can I search by ISBN?
    - Can I interact with the search result?
    - Am I informed if my search returns no results?
- Non-functional:
    - Availability:
        - Can I search anytime of the day?

User story - Unfinished

Story identifier: R003
Story name: Google-Login
Priority: Low

Description

As a user, **I want to** be able to log in to my account with Google **so that** I don't have to use several password and use the app quickly

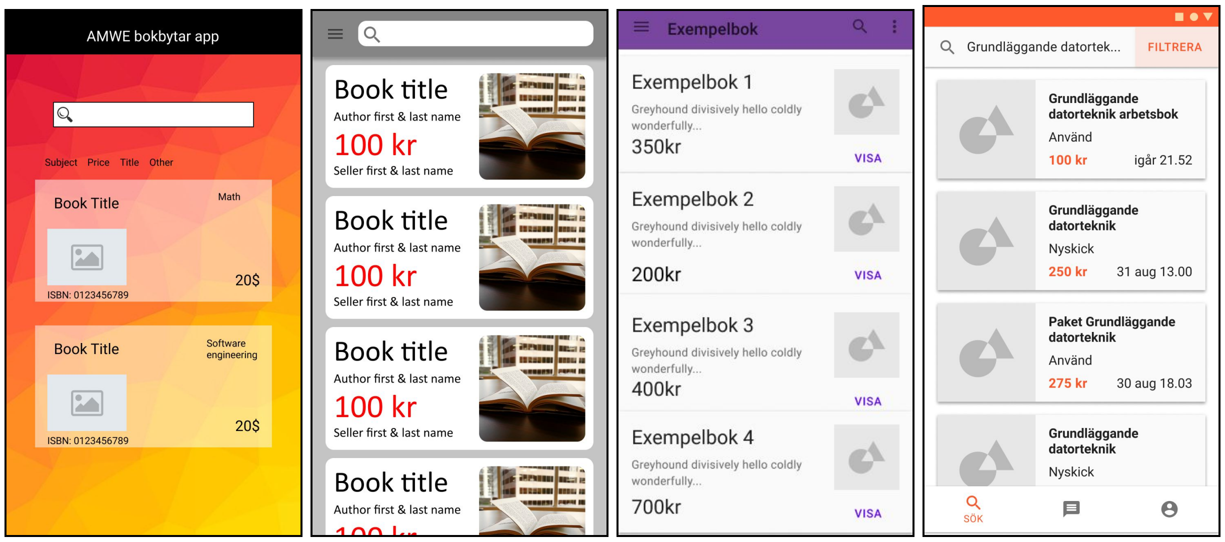
Acceptance criteria

- Functional:
    - Is the login handled by google?
    - Can I avoid using Google to sign in?

2.2 Definition of Done

All user stories need to be Tested, under Version Control, reviewed and all criteria of the user story met.

2.3 User interface

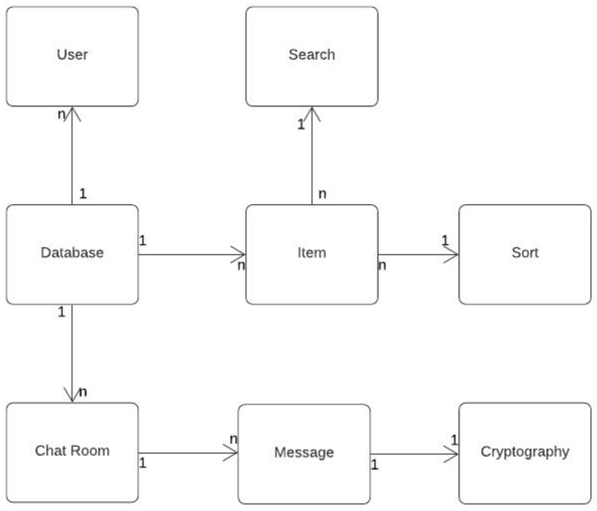


Some early sketches of the user interface.

The user interface is inspired by existing marketplace applications which would make it easy for new users to adapt to it. The whole user interface wasn't designed in advance but we had an overall goal that it should be as clean, easy and aesthetically appealing as possible. This was mainly achieved by following industry standard design patterns. Further Android's own Material design guidelines were also taken into account when the user interface was designed. All this has in conclusion given us a user interface that meets all requirements fully.

When the user starts the program they are met with a login screen. From there the user can either register or login if they have an account. When logged in the user is met with all the listings that are being sold in the program. If they click on one of them they will be taken to a page with information about the listing and also be given the opportunity to contact the seller and add it as a favourite. If the user chooses to contact the seller, the program will initialize a chat room where the potential buyer can write the seller a message. The user can also navigate by the bottom navigation bar to see all their different chat rooms and also their account page. On the account page the user can add their own listing, see their favorites and got to settings where they can change profile picture, password, email etc.

3 Domain model



Domain Model

3.1 Class responsibilities

Cryptography is a collection of classes used for encryption and decryption according to RSA cryptography.

Database handles all communication with the Firebase database and has various methods for adding new content and observing existing.

Item is an abstract class that all listings in the database inherits from.

Message is a class that represents the message stored in the database. It also has some behaviour to simplify encryption and interactions with the database.

Search handles a query in the form of a CharSequence from the user to match that sequence with the title of the Item in the database and then returns a List of items that match the query.

Sort is a common interface for all strategies used for sorting.

User is a conceptual class which is realised in the database rather than in our code.

Chat Room Is a conceptual class represented in the database as the collection of messages between two users.

3 References

List all references to external tools, platforms, libraries, papers, etc.

- Android Studio
- Android Documentation
- Firebase
- Firebase Realtime Database
- Firebase Authentication

System design document for AWME bokbytarapp

Ali Alladin, Magnus Andersson, William Hugo, Elias Johansson

2020-10-23

1.0

1 Introduction

This document is intended to present our system design for our application “Bokbytarapp”. The program functions as an online marketplace where users can sell used items that relate to their education in some form. No transactions are handled in the app as it only functions as a mediator. The program needs a minimum version of Android 7.0 to be installed and works on phones and tablets.

1.1 Definitions, acronyms, and abbreviations

Listing: Refers to all the items that are being sold in the app.
Activity: Activity is a class in Android that takes care of creating a full screen window for you in which you can place your UI components.
Firebase: Firebase is a platform developed by Google for creating mobile and web applications. The functions from Firebase used by us are Database and Authentication.
RSA cryptography: Rivest–Shamir–Adleman cryptography. A kind of asymmetric cryptography, meaning that there’s two keys, one public and one private, used for encrypting and decrypting strings. The private key cannot be derived from the public key, meaning that anyone can encrypt messages for a specific user using that user's public key, but only the user the message is for can decrypt the message using their private key.
Design Pattern: A typical solution to a common software design problem.

2 System architecture

When you start the application you have to login by entering Email and Password in the Login activity (See 8.1 Login). The credentials are then checked by Firebase Authentication. If the account exists and the credentials are correct it will start a websocket session with our database, which will persist throughout the whole run of the application and allow for real time updates to the items shown in the application, and messages sent and received.

If you don’t have an account you have the possibility to register one by clicking on the register button found just beneath the login button. This takes you to the Register activity (see 8.2 Register) which mainly consists of input fields where the user fills in their information. There are multiple conditions that need to be met before the account is registered. If the conditions are not met an error message is displayed with more information. When all the fields are filled and conditions are met the user is presented with a confirmation message and returned to the Login activity.

After you login you come to the MainActivity. MainActivity has a bottomNavigationView that the user can use to navigate between the three pages which are fragments. The first page that the user meets in MainActivity when he logs in is SearchPage (see 8.3 SearchPage) where all listings available in the database are displayed as cards, should another sale be added it will update on this screen. There is a search bar on top of this page where the user can search among all listings. Next to the search bar there is a button that opens up a popup menu. The listings on the SearchPage can be sorted in desired order based on the item-selected in the popup menu.

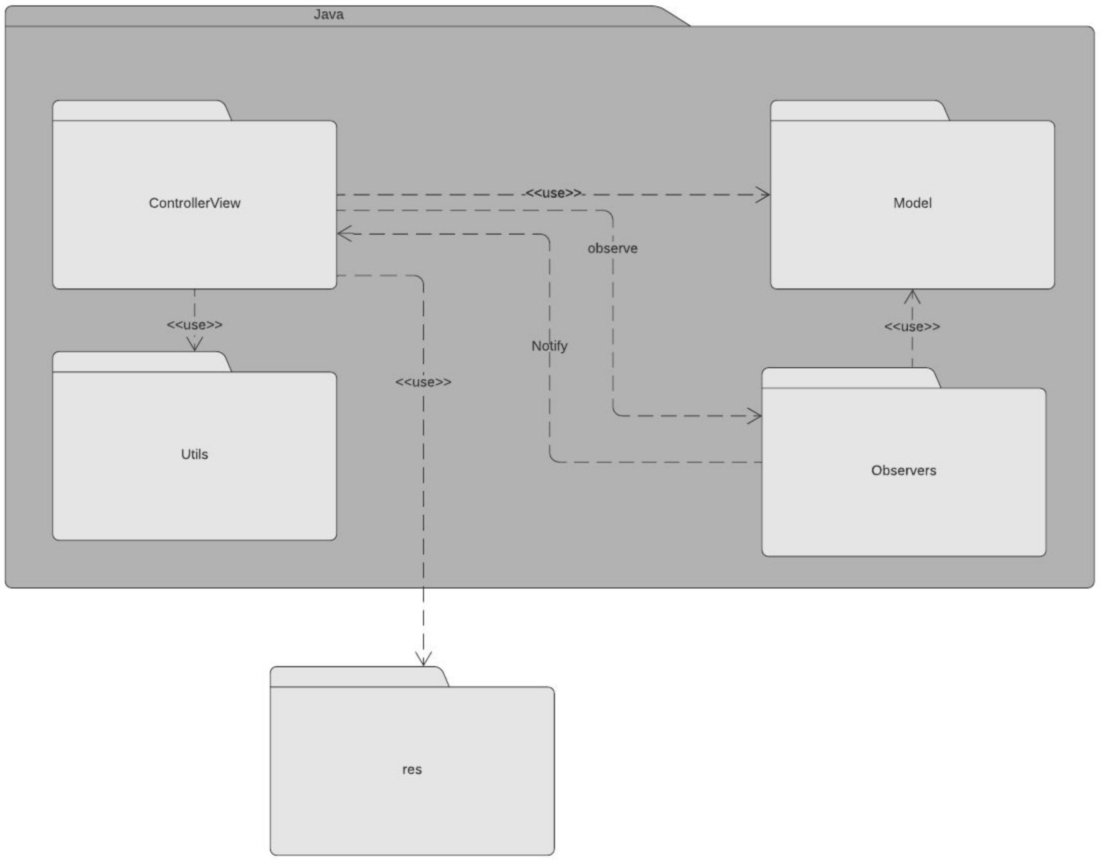
A listing (card) can be interacted with, which will take you to ListingPage (see 8.4 ListingPage) where you will see the full information about the listing. Based on whether the listing is uploaded by yourself or another user the ListingPage has some differences. If it's someone else's listing there's a button to save the listing as a "favourite". Further there's information about the seller at the bottom of the ListingPage. It's also from here you can contact the seller, but more about that will be mentioned later. If the listing on the other hand is your own it doesn't have the option to save the listing. Instead of the seller information there are two buttons displayed at the bottom, one to edit the listing that opens EditListing, and the other is to delete the listing.

Back in the MainActivity the second page after the SearchPage that can be accessed with the BottomNavigationBar is MessagePage (see 8.5 MessagesPage). On this page you can see all users you have contacted or been contacted by. The conversation holds information about the most recent message, time sent, profile picture and name. This functionality comes from ChatroomAdapter and MessagePage. Interacting with a user-card here takes you to a full log of messages, MessageListActivity (see 8.6 MessageListActivity). Here you'll be able to send messages that will be encrypted with RSA-encryption by the cryptography-related classes. It will then be sent to two chat rooms in the database so both users can decrypt it.

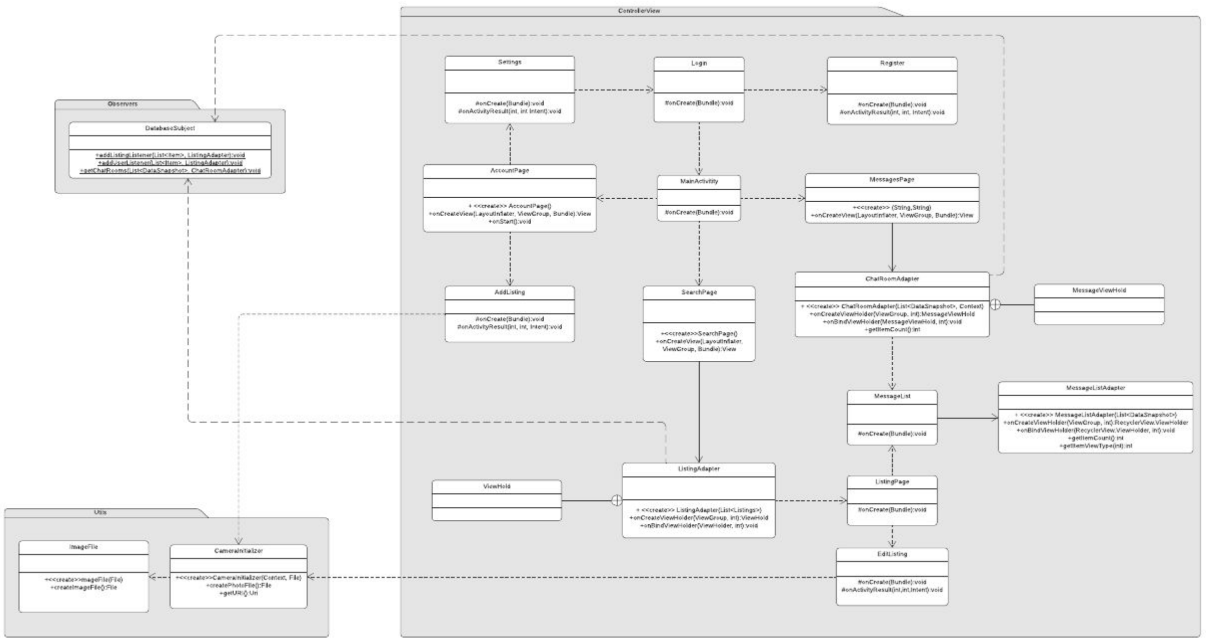
On the third and final page, which is AccountPage (see 8.7 AccountPage) you can see your own listings, see saved listings, post a new listing or go to settings. Both the lists on AccountPage contain the same ListingCards as SearchPage and therefore also take you to ListingPage. A new listing can be added by clicking on the big orange "Ny annons" button. This will take you to the AddListing activity (see 8.8 AddListing) where a new listing can be uploaded by filling all the fields of the form. Just like the register "form" there are some conditions that need to be met before the listing can be added. You will be informed if a condition isn't met. When everything is okay and the listing is successfully added a confirmation message is displayed and the user is redirected back to the AccountPage. The EditListing activity (see 8.9 EditListing) that was mentioned earlier is almost identical to AddListing with the only difference being that the EditListing has all the fields filled.
If one clicks on the "settings-button" the settings page (see 8.10 SettingsPage) appears. On this page the user's can change their own information and also have the option to log-out from the application. The changes made in this page are saved in the database and in Firebase Authentication.

If you are to close the application at any stage, you would break the connection to the database, and be greeted with the login screen upon the next launch.

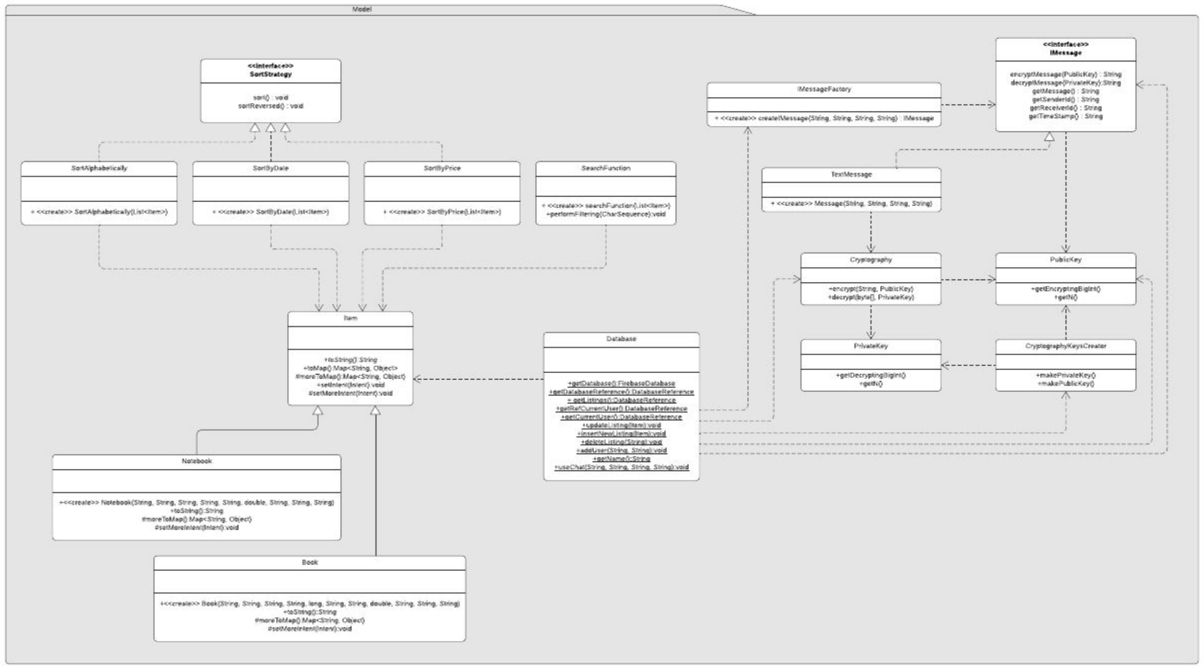
3 System design



Package diagram



Class diagram of ControllerView and Utils packages



Design Model

The MVC pattern is used by dividing our program into three packages: Model, ControllerView and res. Model represent all the model classes in our program. ControllerView are classes that function to some extent as classes that create views and ordinary controllers. This is because the Controller and View classes are hard to separate in Android. In a sense the package res is an extension of the view. Res contains the xml files and images in our project. There is also an additional package called utils that holds two classes that interact with the device's camera and is used by ControllerView. There is an additional package called observers which handle some communication between the database and ControllerView to avoid certain dependencies. ControllerView has adapter classes that use the model to update it and retrieve data from it. This is always done on the highest abstraction level.

We did not find a good way to represent our Res folders as a class diagram.

Module design pattern in form of MVC.
We make use of the **Template method** design pattern in our item hierarchy, because of methods with a majority of overlapping behaviour, but where some deviations are guaranteed.
For communicating with our database we have implemented an **Observer** pattern where the classes listingAdapter and chatroomAdapter observes the DatabaseSubject class. This has another benefit of containing our database, firebase, specific imports to the Database.
Factory-like pattern where IMessageFactory hides creation logic for IMessages,even though there will only be one type in this prototype, and generalizes the creation process.
Our sorting is structured with a **Strategy pattern** to follow the open/closed principle. New ways to sort our listings are almost guaranteed to come along as the application is extended, and this pattern is designed with that in mind.
Singleton - the Database class is a singleton since it is a shared resource used by several different parts in the program.

The domain model contains some conceptual objects that are not represented in our design model. This is because some objects like User or Chatroom are stored in our database. Other objects are represented in the design model such as Cryptography, Sort and Message but their technical implementation is not shown in the domain. There is also the case when two entities are shown as one in the domain. This is the case with the database, which has two classes related to it but it should be seen as an aggregate of both the classes and the database that is used.

4 Persistent data management

We use a database to store all items, messages and users in our program. Each item is stored as a dictionary of strings and numbers.
Books isbn numbers have to be stored as long integers because they are 13 digits long.
We store the price as doubles, to retain the decimals.
Images are stored as encrypted Base64 strings.
The encrypted messages are also stored as Base64 strings in the database.

Users are stored as a unique value generated by our database upon creation of the user, with attached name, and two lists of items, one that the user is selling, and one that the user favours.

Some icons are stored in our res folder.

5 Quality

We write jUnit tests, which gradle will run every time you build the application, and travis, the continuous integration service, will run every time someone submits code to our repo. These tests can be found at \bokbytarapp\app\src\androidTest\java\com\example\amwe and \bokbytarapp\app\src\test\java\com\example\amwe. AndroidTest can not run with coverage. The database related classes are not tested directly but indirectly with other tests. It would not be that useful to write a Mock-Database to see if objects are correctly uploaded and retrieved. Here is a link to our Continuous integration service, Travis, https://travis-ci.com/github/magnusGU/AMWE

Below is a list of all known issues

- Messages are not properly shown sometimes if emojis are involved.
- Messages take a while to load in when you scroll fast in a chatroom.
- Application crashes if one tries to back out from the phone's gallery when choosing profile picture.
- Photo gets rotated on some devices when it's taken through AddListing and EditListing.

Gradle handles both quality and dependency checks and reports. The following is a link to a gradle generated report, containing results of building, checking and generating a dependency tree. For the most in detail report check under the console log tab, where the whole "gradlew clean build app:dependencies" command can be followed step-for-step. https://scans.gradle.com/s/lgawvoymi5okc

5.1 Access control and security

To use our application you have to have a user account, or create one with a "semi-valid" email, (no verification mail is sent to the email, so does not have to be real). But there has to be an at-symbol '@' followed by some service dot some domain I.e name@mail.com.

These accounts, and their password security is handled by our database, Firebase.

6 Potential improvements

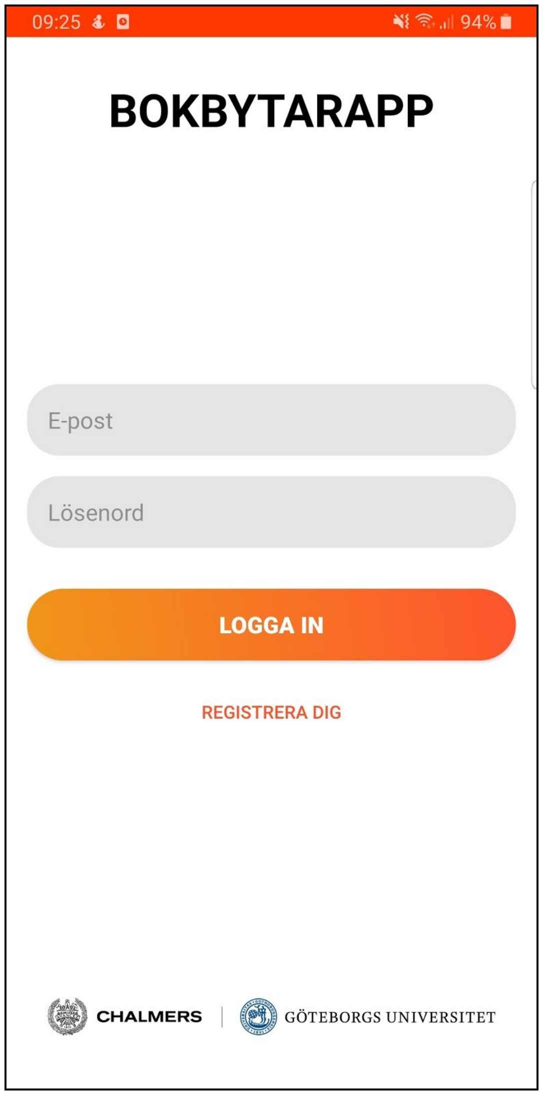
- Performance issues is something that would need to be addressed if the product would be a commercial one. Sometimes it takes time to load the GUI.
- In a commercial product the user should remain logged in after first time application usage and login.
- Messages should be shown in order and be marked as read when you read them. Store messages when you have downloaded them, store them instead of downloading them everytime.
- Messages can’t be deleted right now. This would of course not be
- Settings could be improved a lot. E.g. It’s currently very easy to change the password which is unsafe. Further it would be good if there was an option to delete the account.
- The handling of the database is sometimes not ideal. In several cases too much data is downloaded even though little of it is needed. A solution would be to refactor our database structure.

7 References

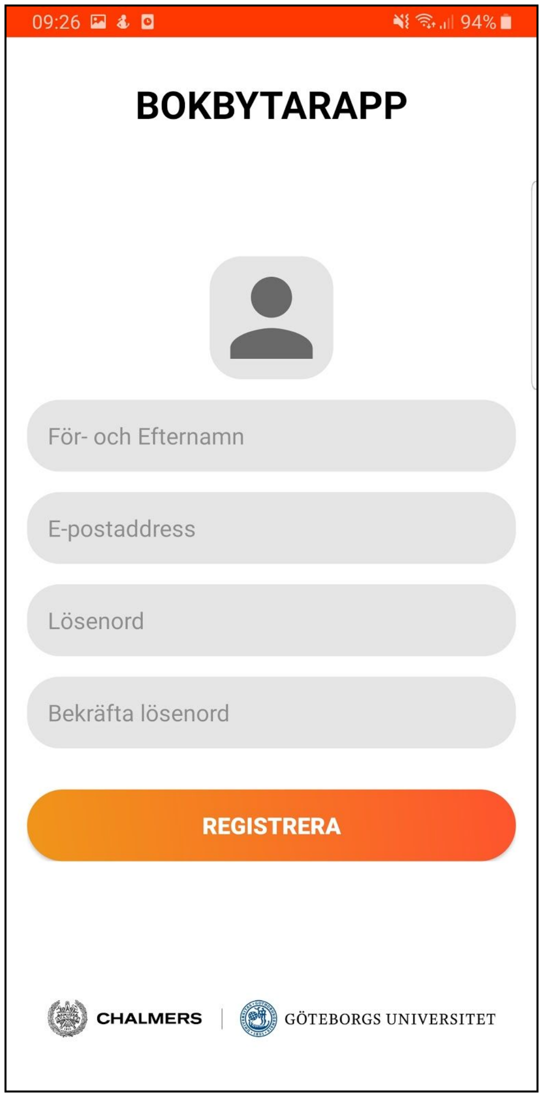
- Android
- Firebase
- Gradle
- Travis
- JUnit 4
- RSA cryptography

8 Appendices

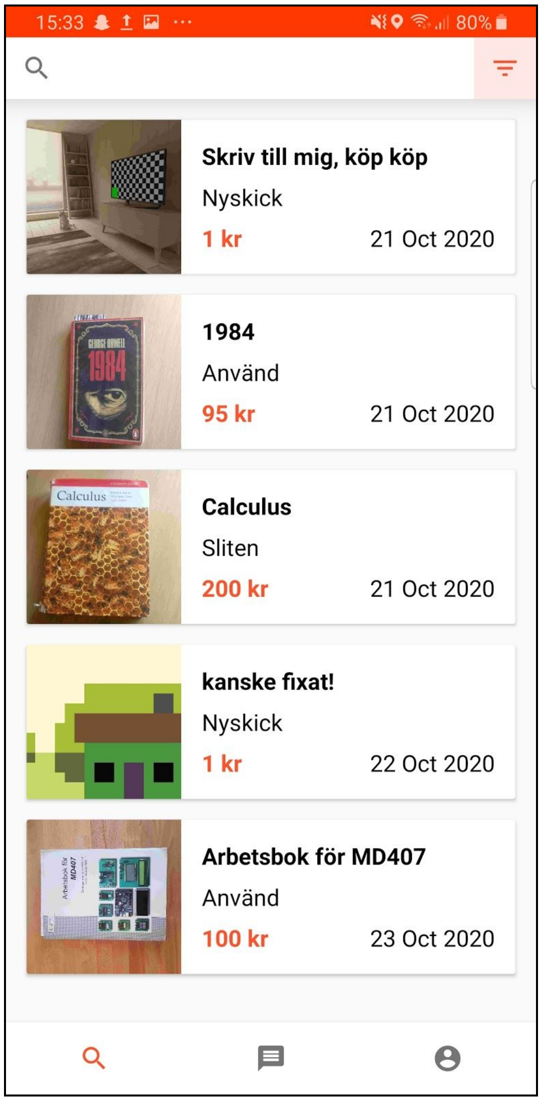
8.1 Login



8.2 Register


09:26
94%
BOKBYTARAPP
För- och Efternamn
E-postaddress
Lösenord
Bekräfta lösenord
REGISTRERA
CHALMERS
GÖTEBORGS UNIVERSITET

8.3 SearchPage



15:33 80%

Skriv till mig, köp köp
Nyskick
1 kr 21 Oct 2020

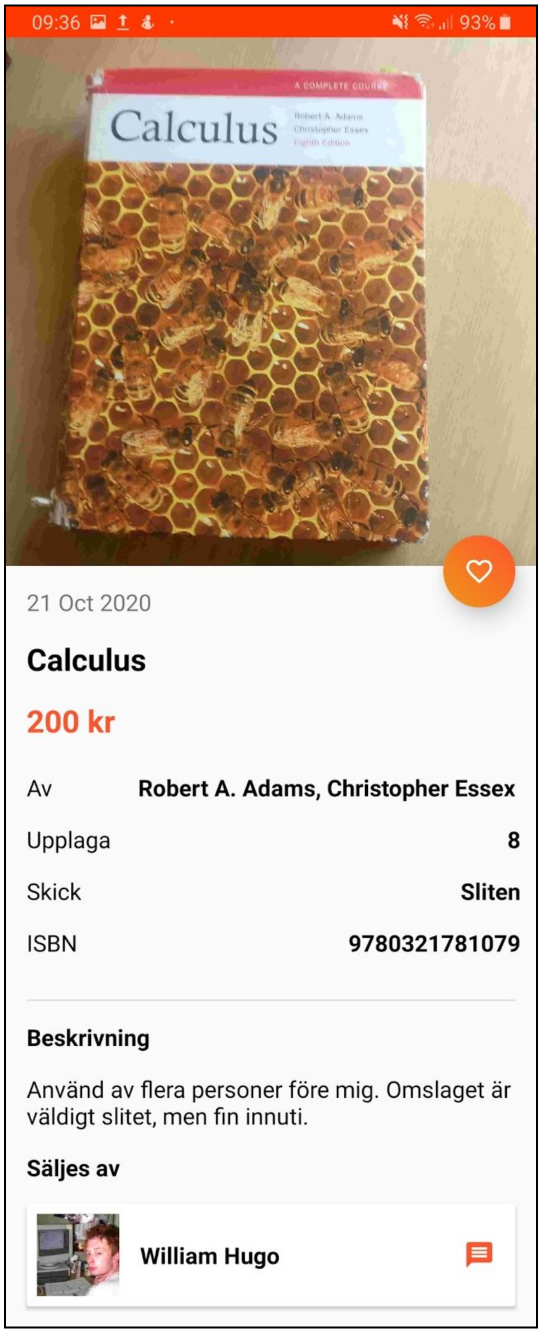
1984
Använd
95 kr 21 Oct 2020

Calculus
Sliten
200 kr 21 Oct 2020

kanske fixat!
Nyskick
1 kr 22 Oct 2020

Arbetsbok för MD407
Använd
100 kr 23 Oct 2020

8.4 ListingPage



21 Oct 2020

Calculus

200 kr

Av	**Robert A. Adams, Christopher Essex**
Upplaga	**8**
Skick	**Sliten**
ISBN	**9780321781079**

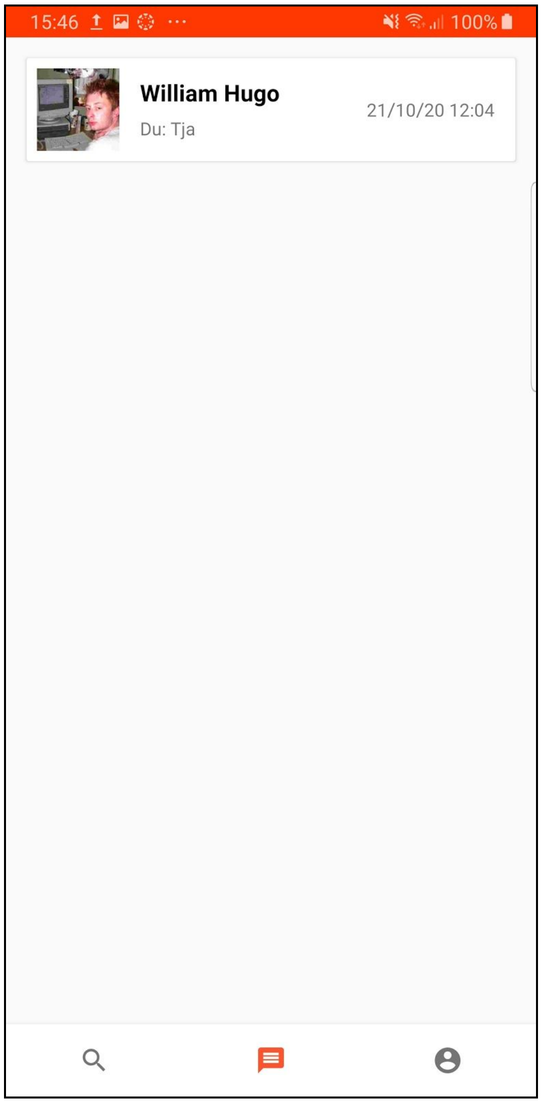
Beskrivning

Använd av flera personer före mig. Omslaget är väldigt slitet, men fin innuti.

Säljes av

William Hugo

8.5 ChatRoomsPage

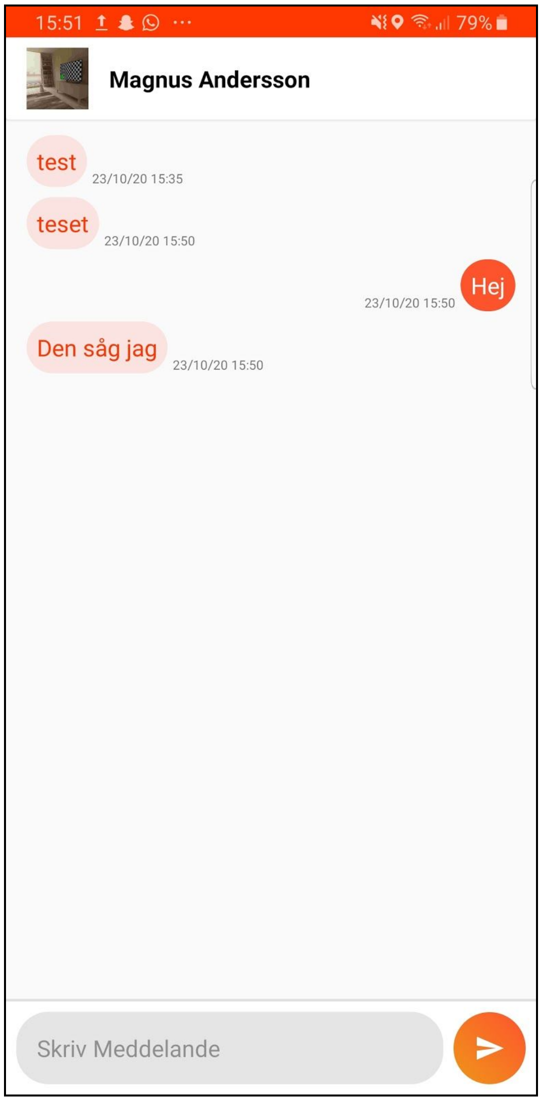

15:46 100%

William Hugo

Du: Tja

21/10/20 12:04

8.6 MessageList



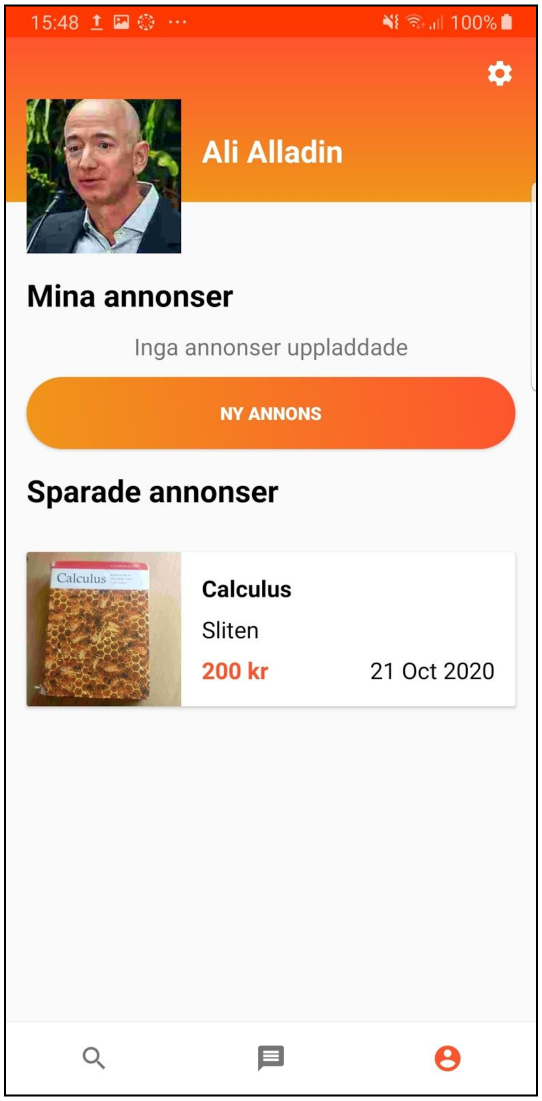
test 23/10/20 15:35

teset 23/10/20 15:50

23/10/20 15:50 Hej

Den såg jag 23/10/20 15:50

8.7 AccountPage



15:48 100%

Ali Alladin

Mina annonser

Inga annonser uppladdade

NY ANNONS

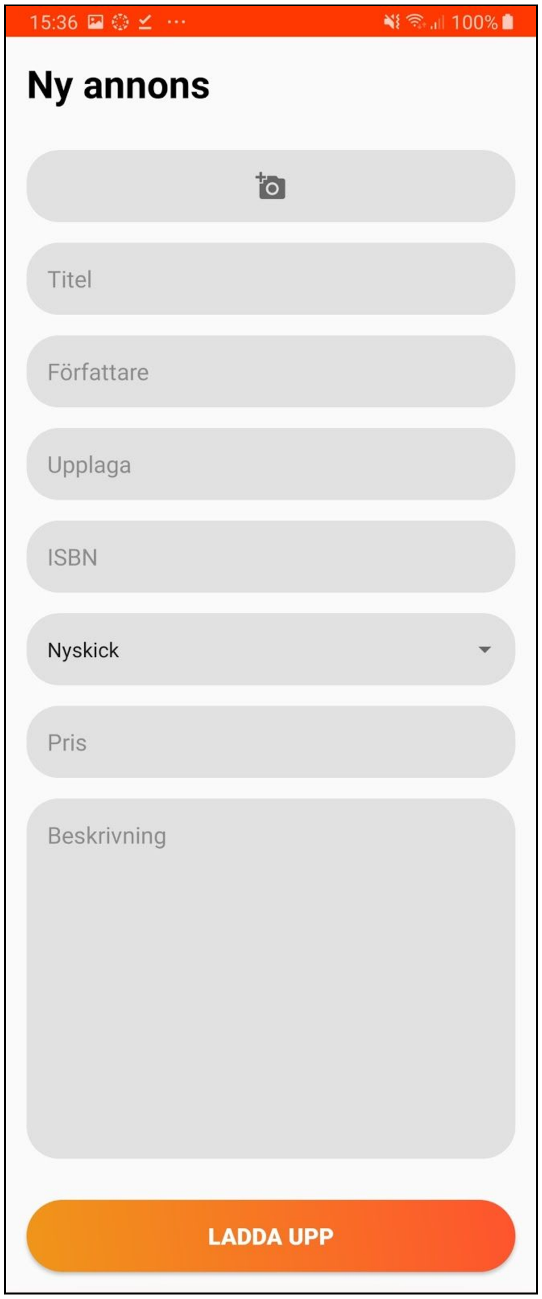
Sparade annonser

Calculus

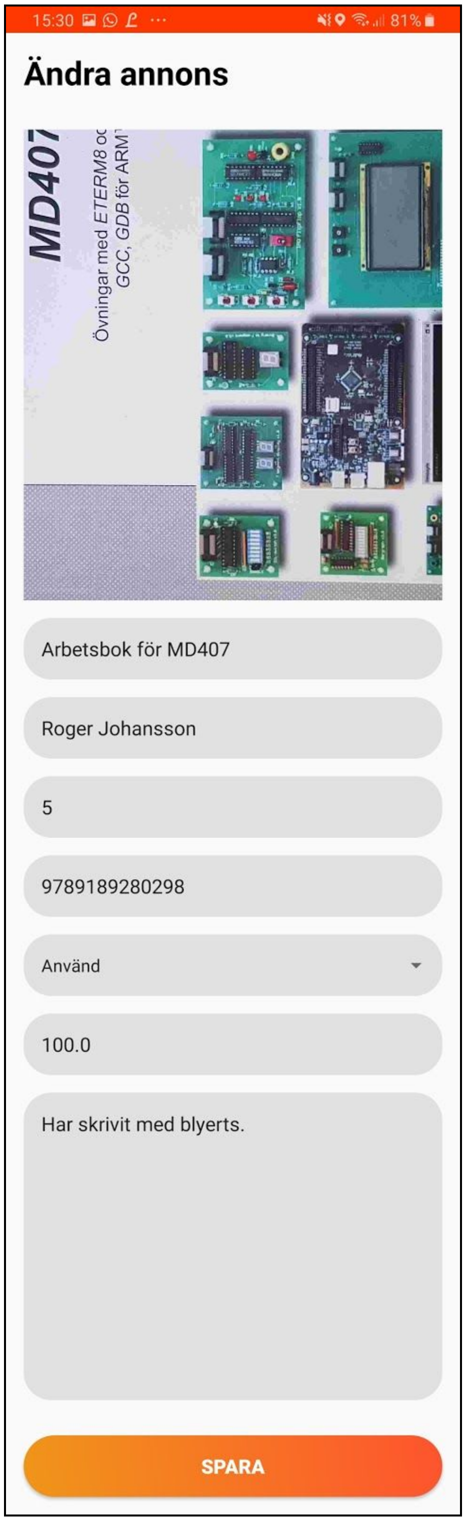
Sliten

200 kr 21 Oct 2020

8.8 AddListing



8.9 EditListing


15:30 81%

Ändra annons

Arbetsbok för MD407

Roger Johansson

5

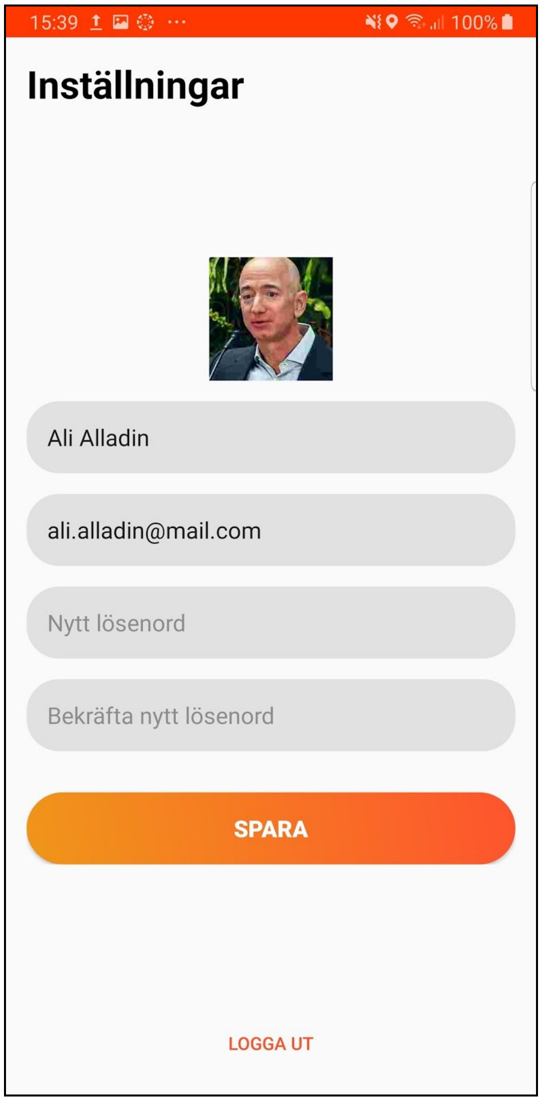
9789189280298

Använd

100.0

Har skrivit med blyerts.

8.10 Settings


15:39 100%

Inställningar

Ali Alladin

ali.alladin@mail.com

Nytt lösenord

Bekräfta nytt lösenord

Peer-review

Group 24 - Fyra eller högre

By group 10 - AMWE

We can tell that it is far from finished, terminal instead of GUI, and todo's exist in the code as well as the documents. But we like your intended GUI look. And think you have good abstractions overall.

You have a domain model that is not up to date and missing classes. Some arrows are missing direction, and the model package consists of only a single type of arrow (Is-A). In the UML both enums used are not titled by name of the enum, but by the different value names.

The User stories are not formatted in a correct fashion and "Start a new game" and "Play the game" are marked as implemented even though not all criteria are met.Furthermore, we think the project is lacking in the amount of user stories. Two user stories for a group of four over 8 weeks seems very thin.

The code submitted had one syntax error, in menyStyle.css, but the class in question is currently unused so the code compiles.

We note some contradicting coding choices in the same file. There are a lot of contradictions which seem to stem from the decision to make the application work in a terminal. A lot of methods not relevant for that have dead or unmaintained code. The group members might benefit from more communication instead of leaving code comments. Some parts are reusable, like Card and Player. But for the most part there are contradictions and dead code which can complicate reusability.

The game class does not follow SRP which means the code is harder to maintain because of possible unexpected side effects. To fix this the class needs to be divided into several classes so every class only has one reason to change.

Some ints in for-loops, while-loops and Lists, seen in Game and Leaderboard are hard coded. This means a change of the existing functionality would make the code harder to maintain. To solve this, use variables instead.

Functionality cannot easily be removed or added, because of exposed class variables and inconsistencies. Example: the Game class both assigns health to the player's life and calls a method decLife() from the player class to change the int.

When it comes to design patterns we only found one at the current iteration of the program is MVC but without real use at the moment since View or Controller are not used. Observer and Facade design patterns are not yet implemented even though it says so in RAD.

The code is to some extent documented but not all methods or classes that need them. Furthermore the documentation does not follow the conventions of JavaDoc which is industry standard. Some methods have comments that don't describe their behaviour, only describe when they're called. Comments also vary between English and Swedish which is not consistent.

Choices of names in the application are overall good with some exceptions. E.g. the method name equals() in Card.java is maybe not optimal and should be changed to something more telling of its behaviour.
The code uses proper abstractions where needed. Like ArrayList generalized to List in method parameters.

The tests of the application cover 70% of classes and 54 % of methods in models. Which is reasonable at this stage of development, but still leaves room for improvement.
Some issues, both performance and security, were found. Performance issues are hard to judge at the moment since the program only uses the terminal at the moment. But one specific example which hinders the performance is using TimeUnit.Seconds.Sleep() to regulate game flow. Players instance variables are not private and are even accessed directly by code in Game, which is a security issue and does not follow Open Closed Principle. It is easily addressed by making them private.

The code is quite hard to read at the moment since there are a lot of comments and system.out.println statements inside methods. Since the Game class has too many behaviours it is also hard to understand.

There isn't isolation in the app currently, the model is currently both printing messages to view/terminal and handling user inputs. But this issue should be solved when the application catches up with the domain model, and implements MVC.

The way Board and Card are dependent on the Enums Shape and Color could probably be improved. It is less than ideal that both classes have that dependency since Board already has a dependency with Card.

The enums Shape and Color work well at the moment and might do so for the production cycle of the program. Adding behaviour to a Shape or Color would be hard. It would follow Open Closed Principle if they were separate classes instead of an Enum but this should be seen as a very minor critique since we as a group don't see why behaviour would be needed.

We think that your plan to implement the Facade, Observer and Strategy design pattern, is well thought out and would help the design of your program.