LogiWord

Brain training Android Application

Bachelor's degree in software engineering

Project report

Group:

Akos Faddi – 253992

David Kabaly – 253785

Krzysztof Majcher – 253784

Supervisor:

Kasper Knop Rasmussen

VIA University College

Characters with spaces 76276
Software Engineering
Bachelor
20-12-2019

Table of contents

List of figures and tables

Figures

Abstract

The purpose of this project is to create an application that can help the user increase the brain’s processing power, help people make more efficient and effective decisions. People tend to take the easy route, the ones that do not require much energy or brainpower, the ones that are just done by habit. If the brain does not get enough stimulus, its processing power will stagnate or decrease over time.(Nouchi et al., 2013)

This project intends to solve this problem by combining two well-known fields, mathematics and language-skills, to utilize a massive amount of mental ability, and force the user to come up with new and improved ways to solve different tasks and problems. (Oei and Patterson, 2013)

The main purpose of this project is to encourage the user to work with limited resources by creating an environment in which such task will be entertaining and competitive. The goal of the user is to combine a valid word from created letters. Those letters are transformed from the corresponding number in the alphabet. The number is a result of calculations performed by the user on a set of numbers and operations that are randomly selected. The performance of the user is later analysed and evaluated in the form of scores. For further feedback the personal performance is stored and compared with the results of other users.

The project was design with goal to provide maximum possible flexibility and scalability. The system is divided into single activity android application, web service based client server communication and dimension modelled relationship database to achieve modularity of the system.

The system was tested with black box test and unit test. All critical requirements and high priority requirements were fully implemented with exception of real time multiplayer system.

Introduction

Many people have problems with their logical thinking and brain capability. This project is aiming to make an application that would be capable to improve the logic of the specific person in a way that is enjoyable and competitive.

There are a few obstacles when it comes to creating the application. The program should be available for as many people as possible that is why it is developed to be playable on most android phones. It must be possible to be used by all age groups and that is why it has a minimalistic design to give the user a better and easier understanding of how the application should be used.

The program itself is using mathematical calculations to formulate words according to their position in the English alphabet. Based on the complexity of the calculation and the length of the word a score is given. The score is used for self-improvement in single-player match and provide challenges in multiplayer match to not just improve the logical thinking but to achieve something in a form of winning against someone else.

Rich Picture

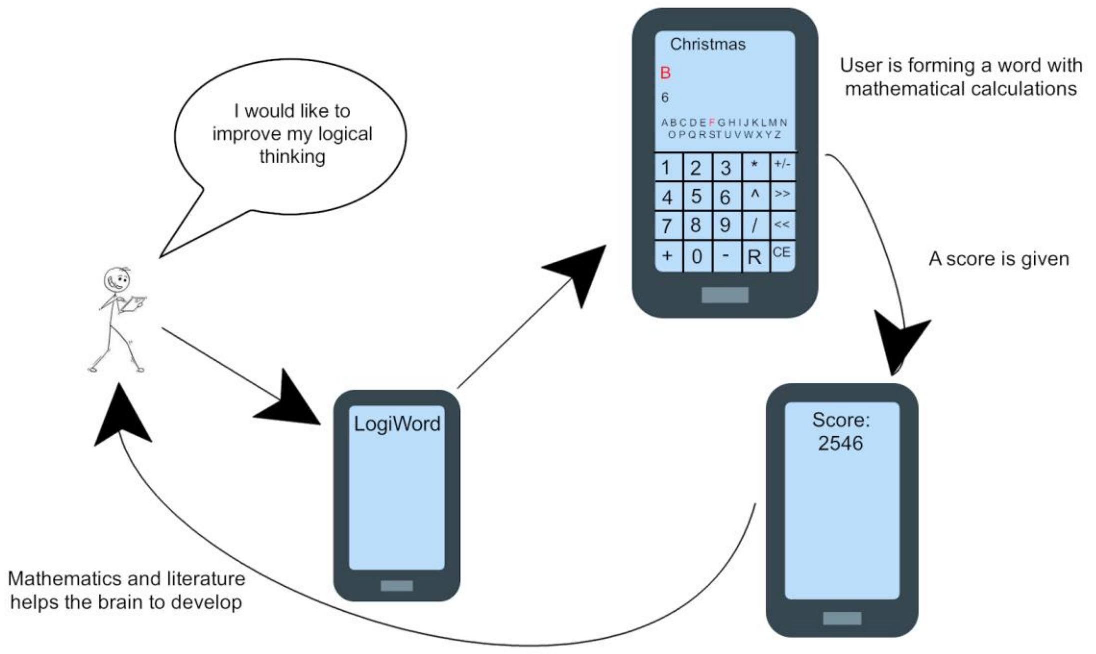

I would like to improve my logical thinking
LogiWord
Christmas
B
6
ABCDEFGHIJKLMN
OPQRSTUVWXYZ
1 2 3 * +/-
4 5 6 ^ >>
7 8 9 / <<
+ 0 - R CE
User is forming a word with mathematical calculations
A score is given
Score: 2546
Mathematics and literature helps the brain to develop

Figure 1 Rich Picture

Figure 1 Rich Picture illustrates a simplified version of the project. The main idea behind the system is to make an application which could interact with multiple users and can give the possibility to train their brain in an enjoyable and competitive way. The software that the project aims to make use of mathematical calculations and literature. The calculations are converted into a single letter that would be selectable in an optional order to form a word. When a grammatically correct English word is completed the user could choose to finish the game and then gets a score.

Background description

The idea of the project has been given from a company called SimpleNem. The company wanted to make a project that could help the people in their life to understand their everyday problems on a better, analytic way and find the best possible solutions. Unfortunately, the project was cancelled because the company had to focus on higher priority projects, but the team were given the chance to work on it individually without any help and make its own decisions.

Because of that reason the team started to research what could be a possible solution to this problem and how it could be improved. According to online sources logical thinking is the main factor when it comes to analysing situations and formulating solutions. (indeed, 2019)

These sources gave the idea to utilize the main tools of brain training, mathematics and literature, that could help people improve their logic. Only the choice of the platform was questionable but after some research it has been decided that an Android application would be the best approach since many people are using smartphones and Android is the most used operating system in the world right now.(statcounter, 2019)

Project structure

Requirements	The section will showcase the functional and non-functional requirements and describe them.
Analysis	This part will analyse the given problems based on the requirements.
Design	The design section will showcase the design patterns that have been used in the project and the choice of technologies.
Implementation	The section will contain code snippets that are important in the application and the explanation of them.
Test	This part will showcase the different types of tests that have been made in the project.
Results and discussion	This section will cover some of the results in the project and will discuss them in more detail.
Conclusions	The section will conclude on the project and describe the system regarding the problem statement.
Project future	This part will cover the future of the project in details regarding the planned features.

Table 1 Project structure

Acronyms and abbreviations

Through the report, many acronyms and abbreviations will be used. The table shows a list of the used terms with the related meaning or elaboration of it.

API	Application program interface (API) is a set of routines, protocols, and tools for building software applications. An API specifies how software components should interact. In this project API is referred to the Android API.
Spring framework	The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application. In the featured application is used to make the server.
Oracle Database	Oracle Database is a proprietary multi-model database management system produced and marketed by Oracle Corporation. It is a database commonly used for running online transaction processing, data warehousing and mixed database workloads.
MVC	Model-View-Controller
MVVM	Model-View-ViewModel
DAO	Data Access Object
CRUD	Create-Read-Update-Delete
POJO	Plain Old Java Object
OOP	Object Oriented Programming

JPA	Java Persistence API
URL	Uniform Resource Locator
HTTP	HyperText Transfer Protocol
OLAP	On-Line Analytical Processing
DBMS	Database Management System
ETL	Extract, Transform, Load
J2EE	Java 2 Platform Enterprise Edition

Table 2 Acronyms and abbreviations

Analysis

Firstly, the prioritized requirements, the use case diagram and some of the use case descriptions can be seen.

Secondly, this section will describe how the basic entities have been identified based on Figure 1 Rich Picture and the thought process of why other entities have been added to the domain. Later, it is described how these entities are going to interact with each other and a domain model is shown.

Finally, the system features can be seen that will describe each feature, show that which requirements belong to each component and the dependencies between these elements.

Requirements

The section will give information about the functional and non-functional requirements of the project. All requirements have a specified priority based on their importance.

MoSCoW prioritization

Requirement category	**Explanation**
Must(**M**)	Defines the highest priority requirement that must be satisfied in the project to be acceptable
Should(**S**)	Defines a high priority requirement that should be done if it is possible based on the time. They are not considered as time-critical cases.
Could(**C**)	Defines a low priority requirement that should be done only if the time and resources permitting it. The solution could be still accepted even if the functionality is not implemented.
Would(**W**)	Defines the lowest priority requirement that the stakeholder wants to have but agreed that it won't necessarily be implemented in the current version of the system.

Table 3 Moscow prioritization table

Prioritized requirements

To showcase the functional requirements a use case diagram with descriptions has been created.

In Table 4 Prioritized requirements, the functional requirements are shown in a prioritized order.

In Table 5 Non-functional requirements, the non-functional requirements are shown in the same way.

ID	Requirement	Priority
1	The application must be able to train the user's brain	M
2	The application must use mathematical calculations to make letters and then formulate words out of them.	M
3	The system must assign scores based on the length of the words and the complexity of the calculations.	M
4	The application must have a calculator layout with the capability to display the letters and later reorganize them in optional order to create a word.	M
5	The application must have the functionality to save all the data on the server.	M
6	The user must have unique identification for login purposes.	M
7	The user should be able to play against a random opponent in real-time.	S
8	The user should be able to manage friends within the application.	S

9	The friends should be challengeable in a match.	S
10	The users should be able to access the top scores of other users on a scoreboard.	S
11	The data should be accessible for the application at any time	C
12	The user statistics should be updated at any time when there is a change and the phone is connected to the internet.	S
13	The application should have daily word challenges for the users.	S
14	The application should have a tutorial what could teach the user how to use the functionalities.	C
15	An achievement system should be implemented in the game.	C
16	The application should be capable to help the user in the next step when the person is stuck during a match.	C
17	Multi Language support should be implemented for the application.	C
18	The main steps in each match should be checkable for the user.	C

Table 4 Prioritized requirements

ID	Requirements	Priority
1	The system should be capable to protect the data from hacking attempts	C

Table 5 Non-functional requirements

Use cases

The section covers the use cases related to the requirements.

This includes the use case diagrams with the related descriptions. The description contains the name of the use case, the actor, the actual description of the given case and a pre and a postcondition.

Not all the use cases are included, to see all the Use case descriptions go to the *Appendix A - Use case descriptions* section

Use case Diagram

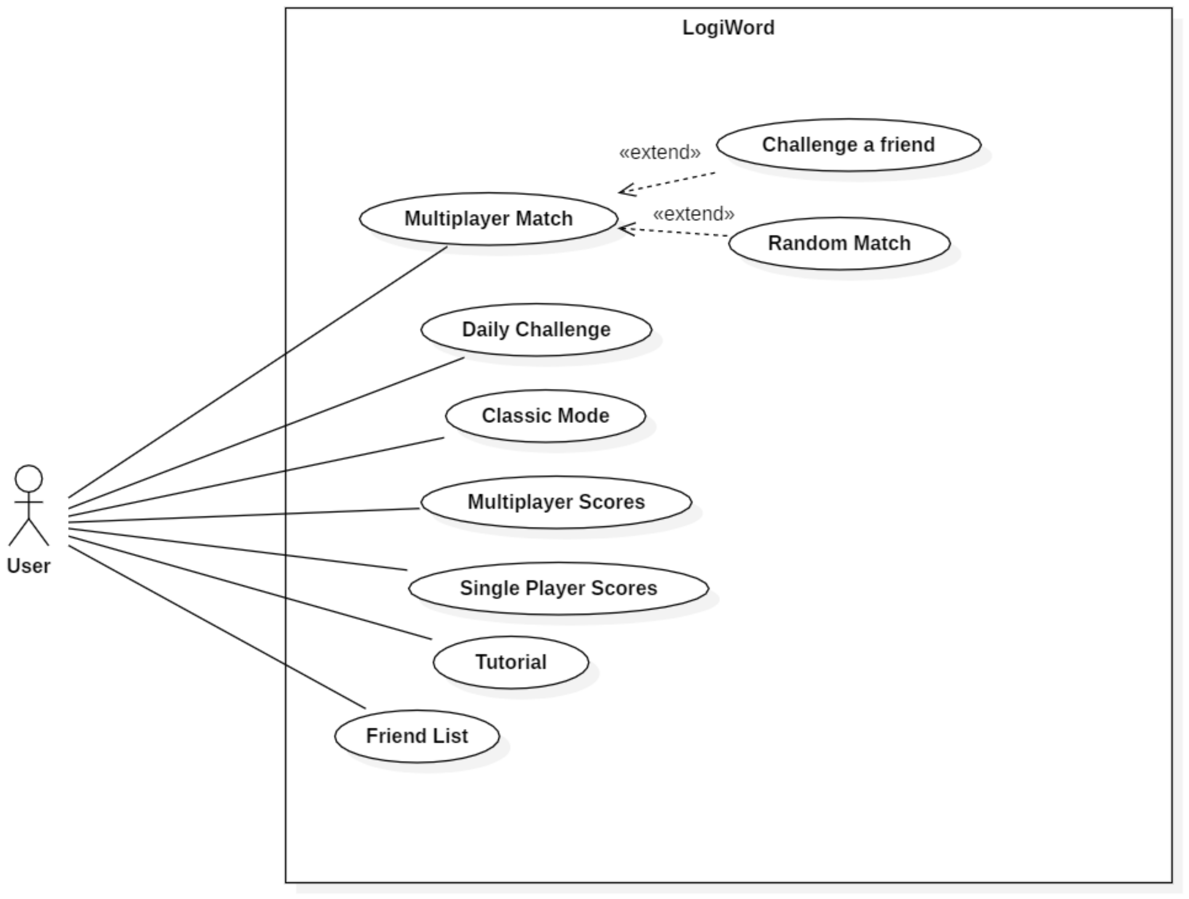


Figure 2 Use case model

Figure 2 Use case model illustrates the use case diagram of the system.

The actor is the user who using the application that has been created during this project. All the features can be reached through the android software.

It should be noted that the features are only reachable if the user has registered an account and logged in to the system. The player could reach most of the cases through the main menu.

Use case descriptions

Use case: Classic Single Player mode

In this use case the basic description of the single player match is shown.

Use Case	Classic Mode
Actor	User
Description	The user will get limited resources like few numbers and operations, and tries to get as many points as possible
Postcondition	The user played a single player match and gets a score

Use case: Challenge a friend for a multiplayer match

The use case describes how the user could play against a chosen opponent.

Use Case	Challenge a friend
Actor	User
Description	The user can select an opponent from the friend list to play a match against each other. They will get a word and should make the most points out of the word based on mathematical calculations.
Precondition	The user logged in and is online
Postcondition	The user played a multiplayer match against a friend

Use case: Play a random multiplayer match

In this use case the player can play a match against a random opponent.

Use Case	Random Match
Actor	User
Description	The user will get a random opponent to play a match against. They must earn the most points with limited mathematical signatures.
Precondition	The user logged in and is online
Postcondition	The user played a multiplayer match against a random opponent

Domain entities

On Figure 1 Rich Picture some of the app elements can be seen interacting with each other. This shows the core mechanics, but for a properly functioning application other functionalities had to be included to provide the necessary utilities for the user.

Based on the Rich picture, the entities that will take care of the basic mechanics of the app can be identified such as Match and Calculation. For the other entities, it is necessary to consider what would improve the user experience and provide useful services.

When the users get the scores, to compare these scores to the previous performances, a scoreboard entity would be necessary, where the highest scores can be seen. This could be extended, so that not just the current user's scores can be seen, but every other player's score is available.

If other user's data has to be shown, it means that there has to be a way to distinguish between each user. This means that a user entity should be included and some form of authentication.

To keep the app interesting, more game modes could be introduced. One can be a daily challenge and now that there are other users, a versus mode could be available as well, either against random opponents or people the user knows.

These specific people could be accessible as friends, but for that it is necessary to manage these connections between users.

Some smaller entities could be included to provide smaller utilities, like a tutorial system to help new players get into the game, hints that could help players who are stuck at the game, and a history system where previous matches can be seen in great detail.

Domain model

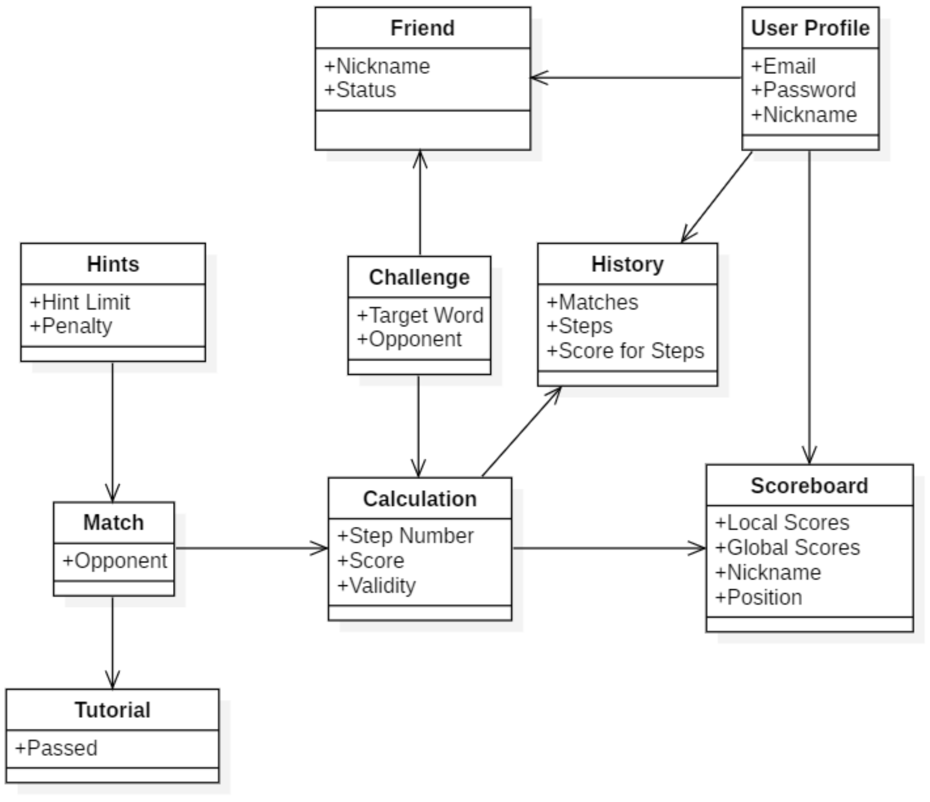


Figure 3 Domain model

Figure 3 Domain model shows how the entities mentioned in the Domain entities section could interact with each other.

When a match is finished, the data is sent to the Calculation to determine the amount of scores that the user achieved.

This score is stored, so it is possible for the user to access the previous matches and compare them. Because of that, it is necessary to identify the user, to have a user profile, so the system knows that which scores belong to which user.

From these profiles, users can interact with other users, add them as friends, and challenge them in a multiplayer game.

If the user prefers not to play against friends, or they are unavailable, random opponents can be challenged.

At last, users can access a tutorial section if they are new to the app, get a little help from hints if they are stuck, or check the details of their previous matches.

System features

This section describes the purpose, requirements and dependencies of system features in this project. The System features are based on the requirements.

1. **Core game**

The purpose of this feature is to provide the core game mechanics for the user and an intuitive UI to navigate within those mechanics.

- *Requirement id 1: The application must be able to train the user's brain*
- *Requirement id 2: The application must use mathematical calculations to make letters and then formulate words out of them.*
- *Requirement id 4: The application must have a calculator layout with the capability to display the letters and later reorganize them in optional order to create a word.*

Dependency: Independent

2. **Score calculator**

The purpose of this feature is to associate adequate amount of scores based on the user's performance by analysing the game results and the amount of operations performed by the user in order to complete the game.

- *Requirement id 3: The system must assign scores based on the length of the words and the complexity of the calculations.*
- *Requirement id 10: The users should be able to compete with each other on a scoreboard.*

Dependency:

- System feature: 1 Core game

3. **Server data storage**

The purpose of this feature is to create a centralized data storage location that fulfills the system requirements and is capable of further expansion, safe password storage and data analysis.

- *Requirement id 5: The application must have the functionality to save all the data on the server.*
- *Requirement id 6: The user must have unique identification for login purposes*
- *Requirement id 8: The user should be able to manage friends within the application.*
- *Requirement id 10: The users should be able to compete with each other on a scoreboard.*

Dependency: Independent

4. **Client authentication**

The purpose of this feature is to provide the ability to register a new user and identify existing ones.

- *Requirement id 6: The user must have unique identification for login purposes*

Dependency:

- System feature: 3 Server data storage

5. **Matchmaking system**

The purpose of this feature is to create a server-side matchmaking system that will match together online users that are looking for multiplayer games and host the games.

- *Requirement id 7: The user should be able to play against a random opponent in real-tim*e.

Dependency:

- System feature: 1 Core game
- System feature: 2 Score calculator
- System feature: 3 Server data storage
- System feature: 4 Client authentication

6. **Friend list**

The purpose of this feature is to allow the user to add other players to their friend list, access this friend list to see who is currently online and be able to send duel invitations to them.

- *Requirement id 8: The user should be able to manage friends within the application.*
- *Requirement id 9: The friends should be challengeable in a match.*

Dependency:

- System feature: 3 Server data storage
- System feature: 4 Client authentication

7. **Duel system**

The purpose of this feature is to extend the matchmaking system by allowing the user to send a game invitation to a chosen friend.

- *Requirement id 9: The friends should be challengeable in a match.*

Dependency:

- System feature: 1 Core game
- System feature: 2 Score calculator
- System feature: 3 Server data storage
- System feature: 4 Client authentication
- System feature: 5 Matchmaking system
- System feature: 6 Friend list

8. **Scoreboard**

The purpose of this feature is to provide a list of the highest-ranking players in single player games and allow the users to compare their results with other players. The users could access their previous scores in a different scoreboard.

- *Requirement id 10: The users should be able to compete with each other on a scoreboard.*

Dependency:

- System feature: 1 Core game
- System feature: 2 Score calculator
- System feature: 3 Server data storage
- System feature: 4 Client authentication

Design

The section covers design choices and various design patterns that have been used during the development. It includes the Architecture, Solid principles, UI related choices and Database information in the system.

To see the overall architecture, refer to the *Appendix F - Architecture Diagram* and to check how the system design looks like refer to *Appendix B – Client class diagram* and *Appendix C - Server class diagram* sections.

Architecture

This section is going to describe the architectural pattern that was used to implement the project. Architectural patterns give an overview that how some of the elements of the system are going to interact with each other.

MVC

MVC is an abbreviation for Model-View-Controller. It is a common architectural pattern used to design and structure the systems. (educba, 2019)

The system will be divided into three parts that depend and connected to each other. It is used to separate the way the data is shown from how it is accepted as an input to how it is displayed as an output.

The three different parts of the pattern are the model, the view and the controller.

The model is used to store data about the system. It is completely separated from how the data is going to be displayed to the user. The model and the user interface are independent of each other. In the project an example could be the FriendListModel class even if MVC has not been used the model classes are similar in MVVM and MVC.

The view is responsible for displaying the data that is in the model. It can use different approaches to show the data, like charts, tables and diagrams. The data that the application contains is visualized here.

The controller reacts to the user input, it can modify or convert it, and decide the right course of action based on the input. Changes to the model will be shown in the view, as the model updates the view.

MVVM

MVVM is a client-side architecture pattern, an alternative to MVC when utilizing data-binding. It clearly separates business logic and data-presentation logic. (Sachin Kumar, 2019)

As the client-side of this project is going to be an android application, MVVM is a recommended architectural design for the system.

MVVM stands for Model-View-ViewModel.

The model, just as in the MVC case, is responsible for the data that is used by the application. In the project most of the features are using a model class like the ScoreModel for receiving and processing any score related data.

The view is what the user interacts with, but compared to MVC, it can contain some logic about how to display the data to the user.

The view-model connects the model and view, as opposed to the MVC pattern, the model and the view is not connected directly. Any communication or binding goes through the view-model, which can change certain aspects of the data before passing it to the next element. In the project many ViewModel classes have been used for communication like the ScoreViewModel class.

This pattern provides high flexibility and as a result while someone is working on the design and the layout of the UI, the other person can implement the logic behind and process the data. It makes it easier to test the app, as automated UI testing is not necessary because the ViewModel can be tested with unit tests. By separating the code into different modules, it makes the code more readable and more flexible.

MVC vs MVVM

At first, the choice of pattern was MVC as it was well-known by all team members, but later, based on recommendation and research, it was changed to MVVM. (Pragati Singh, 2019)

For this project, MVVM seemed to have more advantages compared to MVC.

Because the components are loosely coupled, it is easy to swap the components for another one. For example, if two team members came up with two different designs for the view, it is simple to switch from one layout to the other.

Based on that the components are loosely coupled, if some changes have been made for the implementation of one of the components, the other modules are not concerned about the changes at all.

Implementation-wise, the work can be easily divided between the team members by splitting up the tasks between the components. For example, while one of the team-members is implementing the business logic, another member can work on the view without a problem.

For testing purposes, because of this clear separation, the components can be tested without any concern regarding the other elements.

During the project, the MVVM pattern was more advantageous because it can be used more efficiently with the development environment in Android since it is part of the recommended Android architecture components for better and clearer code.

It has also a good way to handle data and information between the View and the Model using LiveData that MVC does not have.

DAO Pattern

The Data Access Object pattern is a structural pattern that separates the business and the persistence layer from each other. It hides the complex CRUD operations from the other layers. This allows to scale, modify and replace the layers without them knowing about each other. (baeldung, 2019)

It consists of(tutorialspoint, 2019a):

- Data Access Object interface: This interface defines what operations are going to be performed on the model.
- Data Access Object implementation: It implements the above interface and will take care of performing operations on the database.
- Model: POJO, which contains some properties and provides access to those properties.

Advantages of DAO pattern

If the way data is acquired has changed, the other layers are not affected, and the same operations can be performed without any problem. For example, if the database is changed from SQL to NoSQL, only the persistence layer needs to be changed.

The layers are loosely coupled, so only the business/service layer has connection to the persistence layer, but even in this case, only through an interface and not through an actual implementation.

Because of the separate layers, it is easier to test each component.

The pattern emphasizes the usage of interfaces instead of concrete implementations, which is an important object-oriented programming concept(Shubham, 2019).

DAO pattern example in the project

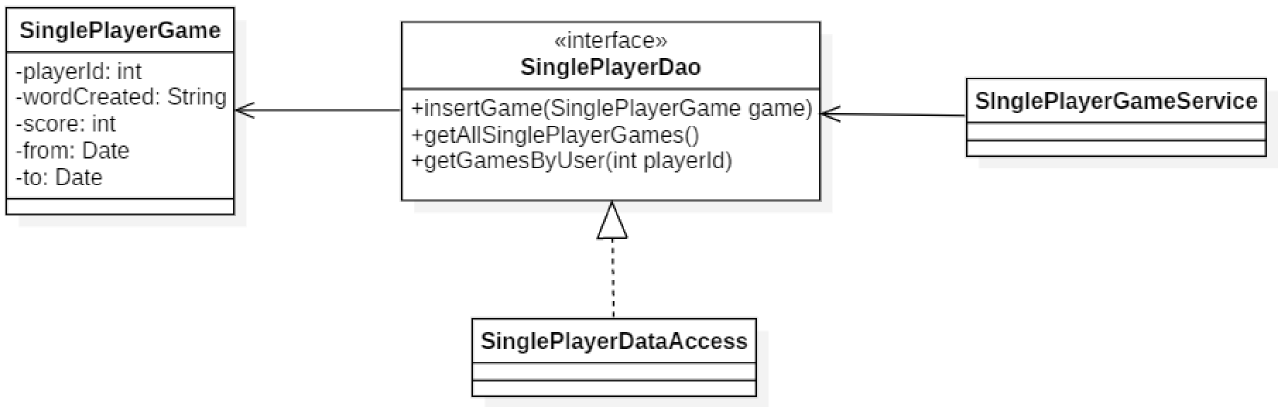


Figure 4 DAO pattern example in the project

Figure 4 DAO pattern example in the project shows the single player game management in the system. SinglePlayerGame is a model that contains the variables that are necessary to communicate with the database. In most cases it would directly mock the entities that are in the database, but there are two SinglePlayer tables in the database, because of that, this model will be disassembled into two sub-model classes that will represent those two tables.

SinglePlayerDao specifies the methods that are going to perform operations on the database.

SinglePlayerDataAccess is the actual implementation that performs the queries on the database. This class can be easily switched to another one and the SinglePlayerGameService, the one that calls these methods, would not notice the change in implementation.

Dependency Injection

Dependency injection is a design pattern used to create loosely coupled classes instead of classes that are heavily depend on each other. The dependant objects are created outside of the class and provided to that class in a different way. As a result, the creation and the binding of these objects are moved outside of the class that depends on them. (tutorialspoint, 2019b)

Types of dependency injections(Bhavya Karia, 2019):

- Constructor injection: the class constructor is used to provide the necessary dependencies.
- Setter injection: the dependencies are provided through an exposed setter method
- Interface injection: an injector method will inject the dependencies to the clients that have been passed to it. An interface with an exposed setter method is defined first, and the clients must implement this interface.

In this project, the Spring framework is used to take care of Dependency Injection. By using dependency injection, the code is going to be more readable because reading the dependencies of a component will be simpler. It is going to make it more testable as well, because while writing the tests, a mock object can be injected which will clearly separate the tested code from other parts of the software.

Singleton

Singleton design pattern is a pattern that takes care of restricting the ability of creating new instances of a class and provide global access to those instances. This project requires all words of a specific language, this list needs to be stored in a tree structure to provide fast search during word validation. Singleton design pattern was used to ensure that the file containing the set of words will be parsed only once to avoid repeating computation costly operations. On the client side a decision to use singleton instead of dependency injections was made because of time restrictions and more trivial technical needs. (Mark Townsend, 2012)

Adapter

Adapter pattern is a software design pattern which goal is to provide cooperation between two classes with incompatible interfaces. In other words, adapter can be seen as a bridge between two classes. The adapter design pattern is often used to allow new classes to work with existing ones without modifying the source code of existing classes. Adapter pattern can be very useful if the developer is not allowed to modify project dependencies due to technical or legal issues. This project required implementation of lists using RecyclerView. For that purpose, RecyclerView.Adapter class was extended in order to provide data binding between data and RecyclerView. (Alexander Shvets, 2019) (Google, 2019b) (Google, 2019c)

SOLID principle

The SOLID principles were used throughout the server development, because some of the principles are utilized by the DAO pattern and in general it is a good approach to follow them in OOP. By using them, the software is going to be easily readable, maintainable and scalable. In this section the principles are going to be discussed in general and examples are going to be shown from the project (Simon LH, 2019).

S - Single responsibility principle

A class should be responsible for only one functionality of the system.

All the controllers (Rest controllers) satisfy this principle, but to specify one, FriendController only defines an endpoint for the clients to reach and does not perform any actions other than passing the call to the FriendService.

O - Open/closed principle

Classes and modules should be open for extension but closed for modification.

For example, the FriendDao interface is implemented by the FriendDataAccess, and this class can extend the defined functionalities with its own methods, like findPlayerId that would return the player id based on a player name.

L - Liskov substitution principle

Classes that inherit from the superclass or implement the same interface should be replaceable.

There is no concrete example for this principle in the project, but assume that the way data is accessed needs to be changed, then another class could implement one of the Dao interfaces, for example FriendDao, and the already existing one and the newly created class could be easily switched without breaking the application.

I - Interface segregation principle

A client should not be forced to depend on methods it does not use.

An example for this principle is the FriendDataAccess and the FriendDAO. The DAO is the interface that exposes some methods for the services to use, but when FriendDataAccess implements it, other utility methods have been implemented as well, which are only relevant for this class and should not be accessible from outside classes.

D - Dependency inversion principle

It is a way to decouple classes. Instead of using concrete implementations, abstractions should be used.

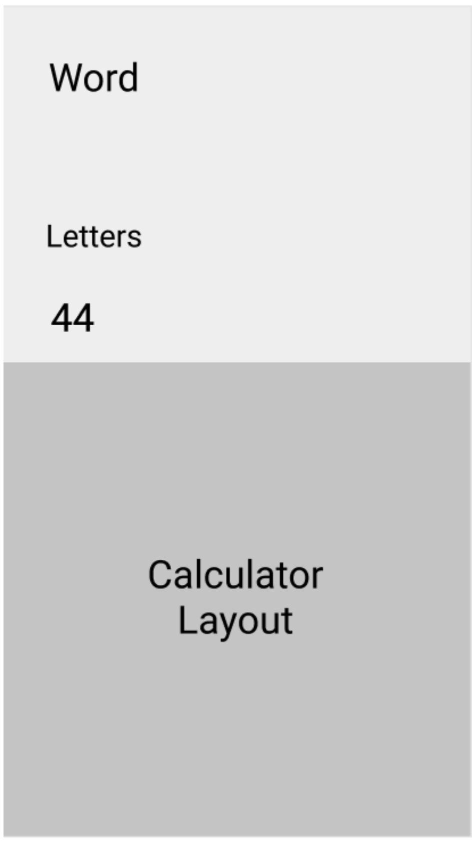
All services use the DAO interfaces instead of the actual implementations of the objects. For example, the FriendService has a dependency on the FriendDao interface instead of the FriendDataAccess actual implementation. This way, if another implementation would be prefered, it won’t require any changes on the service side.

UI Design

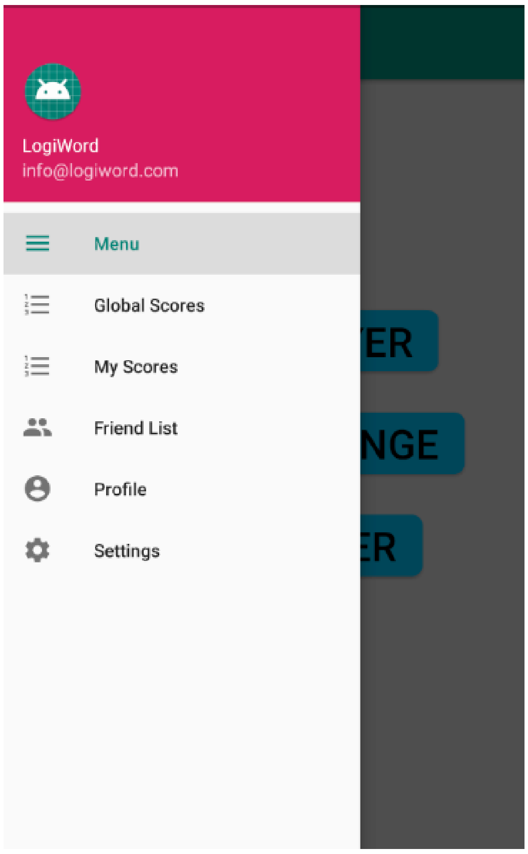
The user interface of the application is crucial and indispensable for a successful product. Unfortunately, the development team had skill limitations about design that is why it has been chosen to have a minimalistic style that could be easy to use and potentially reach more people.

Pre-plans have been made for design purposes with an online tool called “Figma”. The designs are more like guidelines for the developers and not the ones what are finalized in the product. The full pre-design pages can be found in the *Appendix G - Predesign Figma plans*.

The core of the game has an easy to use calculator layout with the related letters and word made with calculations. In most cases the colours are chosen to follow the colour themes by Apple (dark grey, light grey).



Other parts of the UI were made by following the android design principles. Therefore, a custom navigation drawer was implemented in the final version of the product even if it was not planned originally. However, some of the recommendations from the design principles were not used because the application did not need them in functionalities like a floating action button.



ER Description

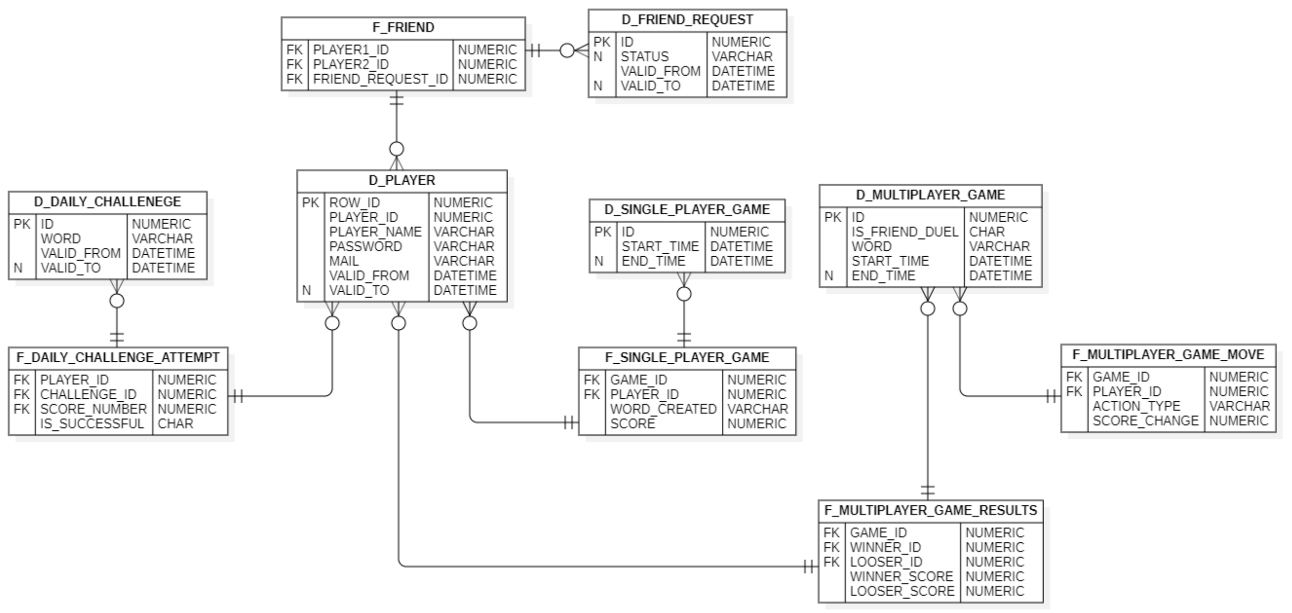


Figure 5 ER Diagram

The Figure 5 ER Diagram visualize a server-side data warehouse. This database uses the fact-dimension pattern which is purposely unnormalized in order to gain extra functionalities out of the relationship databases. Crow foot notation was used to create the ER diagram to make it simpler and easier to read. (Kimball et al., 2008)

This model allows to keep track of data changes, because it does not allow to remove data. In a standard normalized model if data is replaced by another data the information will be lost, however here the *valid_to* column will be replaced and if necessary new row will be inserted. This creates a big potential for further data analysis and allows to go back to the previous values if needed. Furthermore, this makes the database more user friendly especially for people with little experience with databases or who are unfamiliar with data models, since it requires less joins while querying. Finally, compared to the standard normalized model, it has a better performance especially with large amount of data. Tables on the top of the diagram starts with the letter "D" that means that they are dimensions.

Dimension is a structure that categorizes facts in order to answer business questions. They are characterized by few features, all of them have a primary key and additional two columns usually called *valid_from* and *valid_to*. Those columns store information when the data was inserted and starts to be valid or when it becomes outdated.

At the bottom of the diagram there are tables that starts with the letter “F”, these are facts. Fact is a measurement that consists of the set of foreign keys. As a contrast to dimensions it does not have a primary key, because it is identified by foreign keys. It also does not have *valid_from* and *valid_to* columns, because the data stored in the fact rows is never going to change. Some tables are facts and also dimensions like *single_player_game*. It is a way around the limitation of fact-dimensions model. In this case single player game needs a primary key, a start and end time, however it also needs a connection with *d_player*. Another table worth mention is *d_player* that have two id's, row id and player id. This table is called slowly changing dimension and it requires two separated identifiers in order to keep track of the data change.

The fact-dimension model was chosen for a few reasons. Firstly, this database should be designed to handle large amounts of data in case the application become popular so the database won’t require additional redesign after publishing it and will be able to generate profit instead of problems. Secondly, adding additional tables in this model is trivial and won’t require any changes in the previously created tables. Thirdly, this model opens opportunities for data analysis and further development, even if there is no feature like that planned for now, it is still a good practice to have it.

Dimensional modelling

The following section describes the reasoning behind the choice of dimensional modelling over third normalized form modelling.

Third normalized form databases are beneficial for highly transactional systems, in other words, systems in which there are many *inserts*, *delete*, *update* queries. For small systems 3NF models will provide small performance boost, however this project requires many *join* queries. Join queries are operationally costly, because they often require DBMS to perform nested loops. Dimensional modelling solves this problem by reducing the number of tables. (Oracle, 2019c)

Databases that implement dimensional modelling, due to their unnormalized nature require more memory and generate additional hardware cost. This could be problematic if the project was made in the 90's, however modern RAM and hard drives are relatively cheap. Furthermore, the lifespan of modern SSD drives that store data using semiconductor can potentially benefit from systems that are inserting new data instead of modifying existing ones. To fit the characteristic of dimensional modelling and minimize memory usage, the database in this project was tuned by setting *PCTFREE* parameter. Oracle DBMS which is used in this project by default leaves 20 percent of space in the block empty to prevent data migration to another block if data would change to something that requires more space. Since data in fact tables are never going to change this parameter was set to 0. (McCallum, 2019) (Oracle, 2019a)

From the cost of bigger database memory requirements, comes an advantage in the form of data change history. This project doesn’t include any data analysis or business intelligence, however in fact it doesn’t have to since the database is already compatible with many OLAP and data analysis tools. As a result, data analysis would be a matter of importing data to a tool like PowerBI and if there is a need for more complex analysis an ETL process would be much easier to implement compared to 3NF models. In its current state the project only analysing the most popular words made by users in single player in the last 7 days to estimate the word for daily challenge. Having *valid_from* and *valid_to* is convenient for purposes as such. The main reason why the decision to store data history was made is the value of the data. Applications with multiplayer should be structured that way that they could be supported for a long period of time and cheap to update even if there are no plans for future development, because the popularity of an application may be unpredictable. For future development this project will require feedback from the users and this feedback is provided by the dimensional modelled database and its data history. (A. Tsois, N. Karayannidis, n.d.)

An additional reason why the 3rd normal form model was not chosen because it required further development time. 3NF is more popular and commonly used which makes it more intuitive for developers that are not specialized in databases, however it requires more time to implement databases. Checking if every table fit all 3 normal forms is an additional step that serves very little purpose. Furthermore, making modification to the database will require developers to check all normal forms again and redesign many tables. In dimensional modelling modifying a table does not affect others, adding new information to store will require only the attachment of the new table to the existing ones. In the initial database design in this project there were no D_FRIEND_REQUEST table. During a sprint devoted to implementing the friend system, the developer team realized that it would be beneficial to not only store the fact that users have friends but also friend requests that are waiting and those ones that had been rejected. To implement this feature only adding additional dimension was required. In 3NF that modification would be more complex and might be abandoned due to lack of time. Overall simplicity of dimensional modelling reduces development time by making databases easier to query.

Choice of technologies

Collaboration

GitHub is a cloud-based service and a website that is aiming to help developers manage and store their code. It also helps to keep track of the development changes and in some cases revert the program back to the working version. It uses an open-source version control system called Git to take care of the code sharing.

During the project, it was used as a collaboration tool to work simultaneously on different parts of the application. On the client-side, the integrated collaboration tool in Android studio and on the server-side, a program called “Sourcetree” was used.

Spring

Spring is a framework that can be used to develop any Java application. Extensions can be installed for specific development plans, for example for web applications. Spring makes J2EE easier and enables POJO based programming models. (tutorialspoint, 2019c)

It includes packages that make development easier, for example make it simpler to define the API endpoints with just a few annotations, or with other annotations the connection to the database can be described.

All this functionality comes in packages, so the developer only needs to import those that are going to be used in the development.

JPA

The main concern of Java Persistence API is to make sure that objects outlive the processes that created them, for example storing them in a database. JPA allows the developer to define which objects and how these objects are going to be persisted. (Tyson, 2019)

JPA defines a set of concepts that can be used by any tool or framework, but by itself, JPA is not a framework. It works great with relational databases as it can map the model classes to database entities.

Implementation

The next section is going to cover important parts of the implementation process and highlight how previously discussed design patterns have been implemented.

At first the overall system interaction is shown in a simplified version. Then relevant code snippets and their descriptions can be seen from client to server. Finally, the system security is discussed.

Data workflow

To have a better grasp on the system, a simplified version of how the major components interact and communicate with each other is demonstrated. Figure 6 Data workflow shows a general diagram, as most of the features go through the same path while calling similar methods. In order to see an interaction within the system please refer to the *Appendix E - Sequence Diagram* section.

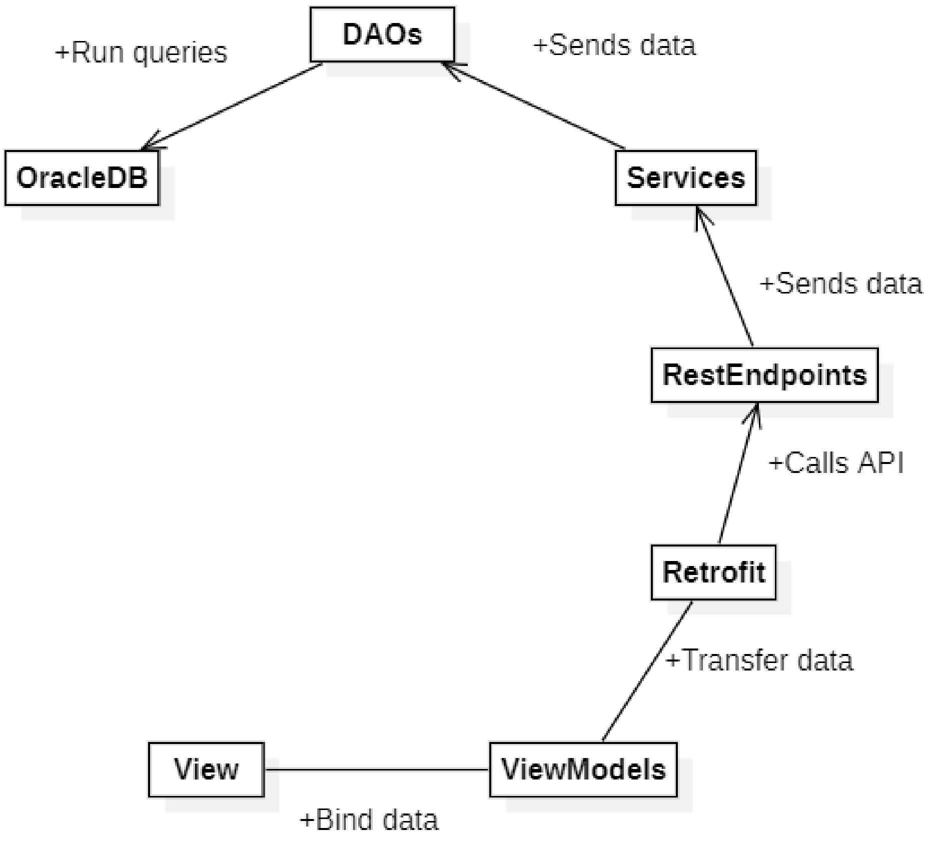


Figure 6 Data workflow

The flow of the system starts with the View. The view contains everything that the user will have on the screen. However, it must not contain any logic. In the project the view is made with fragments and not activities because of the capability to reuse them and build multi-pane interfaces.

It is connected to the ViewModels through LiveData so there is no need to have a call every time when the data changes.

The next element is the ViewModel that is a link between the View and the Model. It is managing data links and conversions. This is where data binding is important because it is handled by the AndroidViewModel or ViewModel class. In the project it is connected to the Model and to Retrofit in order to gain information from them(Ampire, 2019).

The final element is the Retrofit on the client side. It is a Rest Client for Java and Android that makes it easy to upload and retrieve structured data like JSON based on web services(Lars Vogel, Simon Scholz, 2019)

In the project GSon was used to convert the data to Java objects from the next element in the flow that are the rest-endpoints.

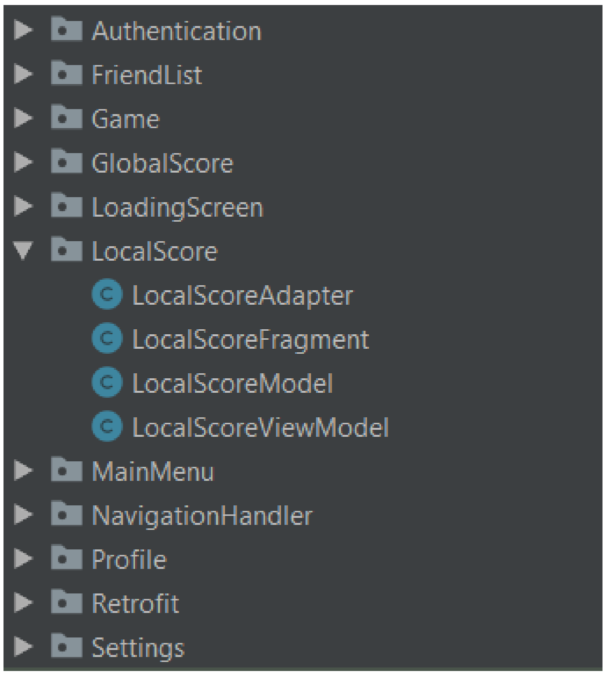
The rest-endpoints do not perform any operations on the sent data, only forward it to the appropriate service. The services are the ones that perform business logic operations on the sent data, but in the current state of the project, there is no actual processing in any of the services, they act more like a bridge and just call the DAO.

The DAO implementations contain the queries that will retrieve, insert or update the database. If the request was about getting data from the database, then the direction of data is reversed, database, DAO, service, etc. Currently if there is some error in the request, the server simply ignores it and do not provide any error message to the client.

Client Implementation

MVVM

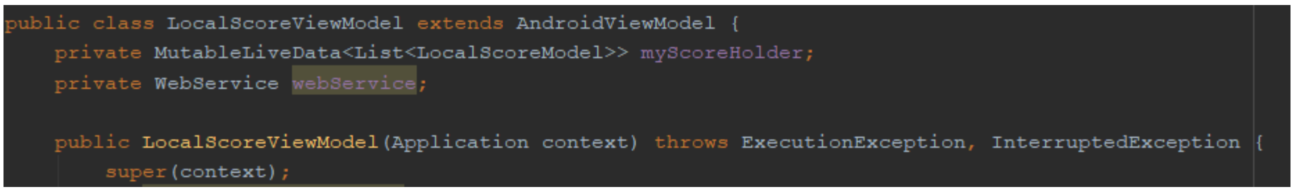
In order to make sure the system follows the MVVM design pattern the names of the classes are referring to their tasks within the pattern, like LocalScoreModel is a model that will be used by the connected ViewModel.



Snippet 1

This also made sure that no important runtime data will be removed by the garbage collector after the fragment had been destroyed.

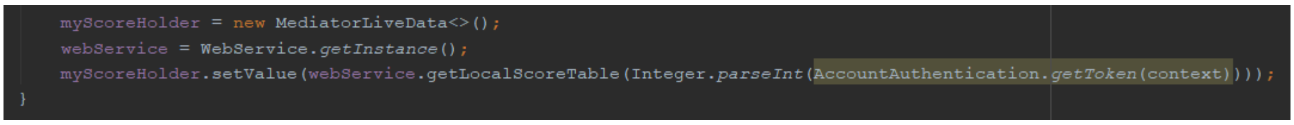
Android has two extendable classes that help the developer to use the design pattern, namely the ViewModel and AndroidViewModel classes.



```
public class LocalScoreViewModel extends AndroidViewModel {
    private MutableLiveData<List<LocalScoreModel>> myScoreHolder;
    private WebService webService;

    public LocalScoreViewModel(Application context) throws ExecutionException, InterruptedException {
        super(context);
```

Snippet 2

The difference between the two classes is that AndroidViewModel have a constructor that receives the Application as an argument since it extends the Context class and in the case of the ViewModel class, the Context of the application could be used (Google, 2019d).

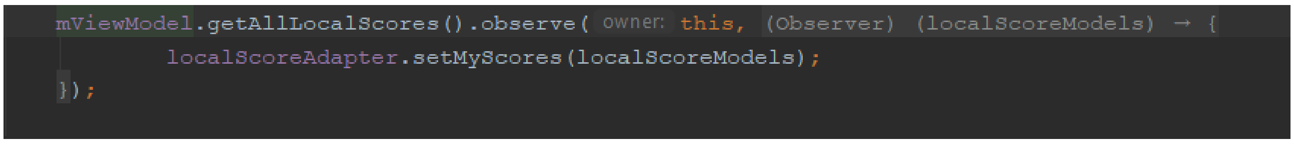


```
        myScoreHolder = new MediatorLiveData<>();
        webService = WebService.getInstance();
        myScoreHolder.setValue(webService.getLocalScoreTable(Integer.parseInt(AccountAuthentication.getToken(context))));
    }
```

Snippet 3

In some cases, the context was needed because the Authentication service has a function to retrieve the user token and it needs to get it to be functionable.

In order to have the most updated data on the UI, LiveData has been used that was being observed by the View.



```
    mViewModel.getAllLocalScores().observe( owner: this, (Observer) (localScoreModels) → {
            localScoreAdapter.setMyScores(localScoreModels);
    });
```

Snippet 4

Because of that, whenever a change happens in the data the new information will be automatically shown on the screen. As on Snippet 4 shown the LocalScores are being observed and when a change happens it sets the information in the adapter to the new updated ones.

Game mechanics

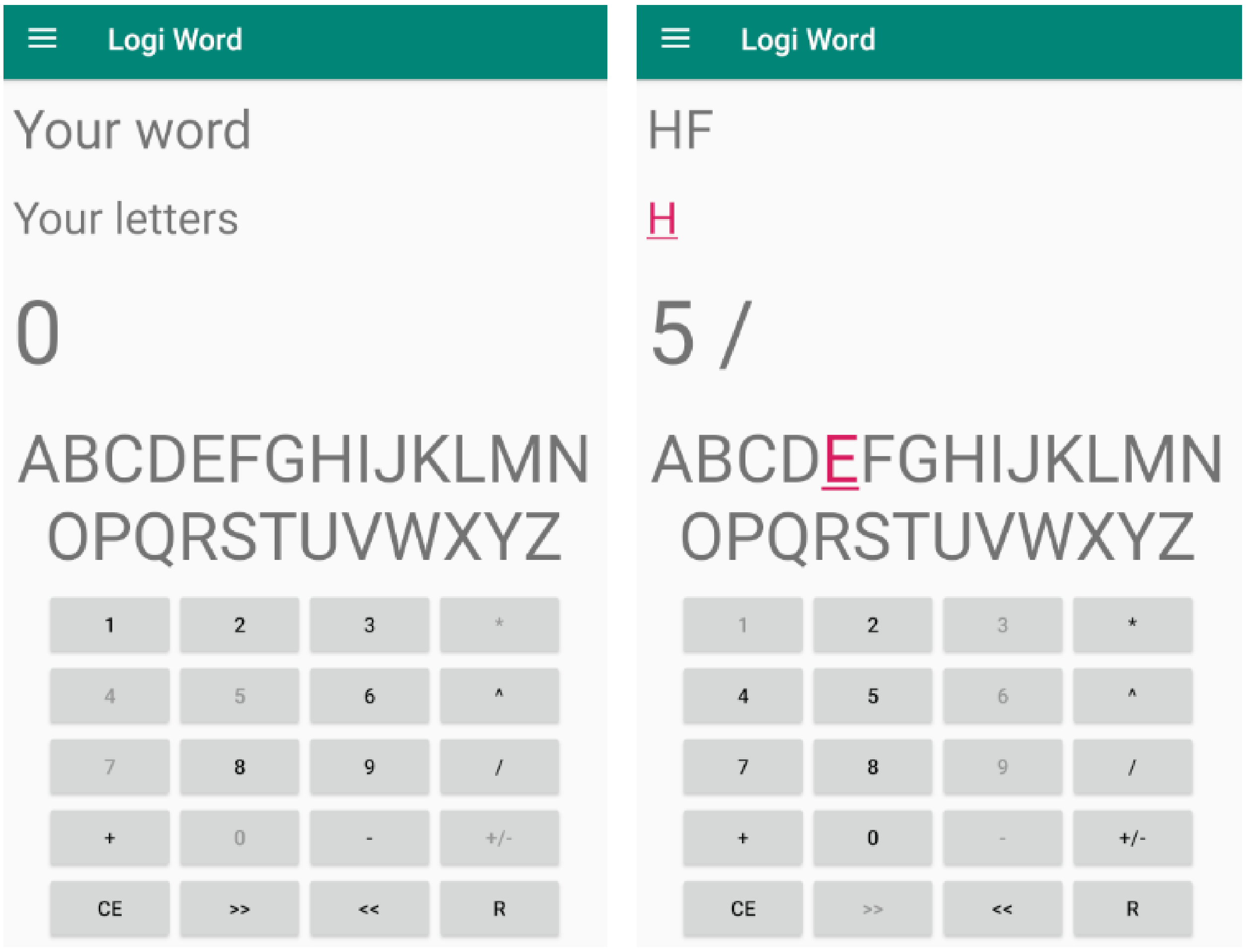


Figure 7 Game mechanics

Figure 7 Game mechanics shows the main game UI. This UI can be divided into five segments.

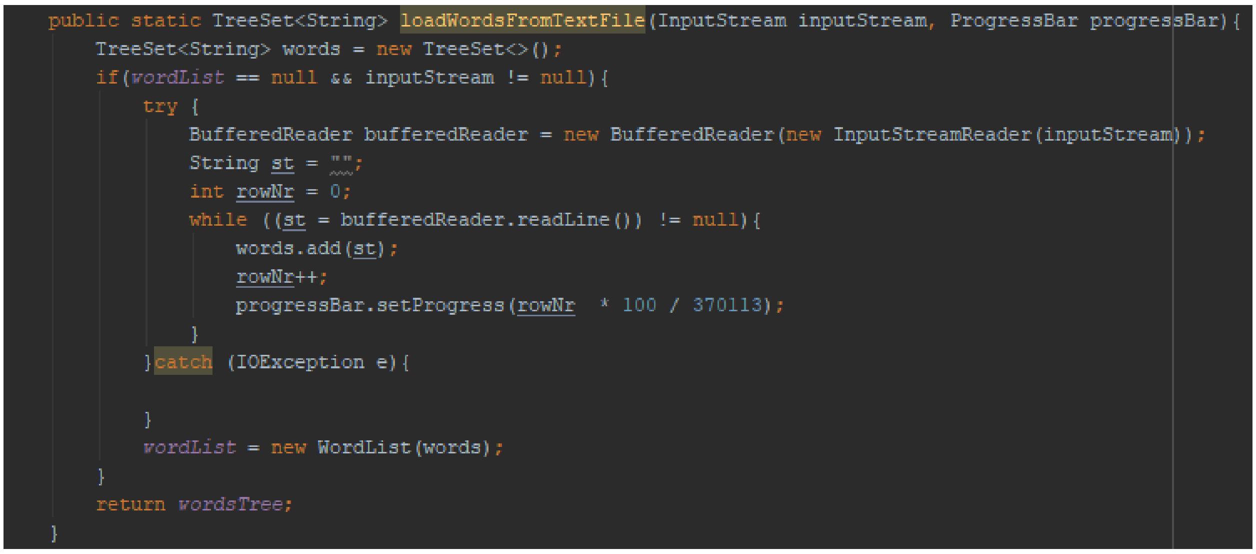
At the bottom of the screen there is a button layout similar to a calculator. Number buttons will set a value displayed above the segment that shows the alphabet if the current value is 0. It is not possible to replace value other then 0 or insert multiple digit values. If current value is not equal to 0, numbers can be used only as a coefficient of the calculation. Beside the number buttons, the available operations can be seen. If the user clicks on them, the corresponding symbol will be displayed on the value section, but operation will be performed only after the coefficient has been chosen. It is possible to change operation before choosing coefficient. These rules apply for the following operations: summary, subtraction, multiplication, division, and exponentiation. Other operations do not require coefficient and will be performed on click. Button “+/-“ will change the sign of the current value. Buttons “>>”, “<<” will remove right or left digit from current value if the value has more than one digit. Button “CE” will set the current value to 0 but it won’t restart the game and previously performed operations will affect the final score of the game. “R” button will be described later in this chapter. Buttons with grey text are not clickable. System will not react to the user trying to click on them. These buttons are selected at random and after an operation has been performed, unclickable buttons will be randomized again. The amount of unclickable buttons depend on the game difficulty.

Above the calculator section there is an alphabet. The main purpose of the alphabet is to visualize for the user to which letter the current value is mapped to and how far the user is to other values. It is also helpful for people that do not know the alphabet by heart or do not want to count letters in head. If the value is between 1 and 26 one of the letters will be highlighted and clickable. If the user decides to click on the letter this letter will be moved to the “Your letters” segment and the value will be set to 0.

The segment above the alphabet holds letters created by the user and have not been used yet. All of them can be clicked in any order. On click they will be moved to the “Your word” section. If the player wants to change the order of letter selection or not select already selected letters, the “R” button can restart “Your word” and move every letter back to “Your letters” section.

The section at the top of the screen holds currently selected letters in corresponding order. If the word created from the letters appears in the English dictionary, the word will be highlighted and clickable. If the user decides to click on the highlighted word, an alert dialog with the current score will be displayed. From that dialog, the user can decide whether or not the game should be finished. In case of the daily challenge mode, only the daily word will be considered valid. Additionally, the daily word will be displayed in the toolbar instead of the name of the application.

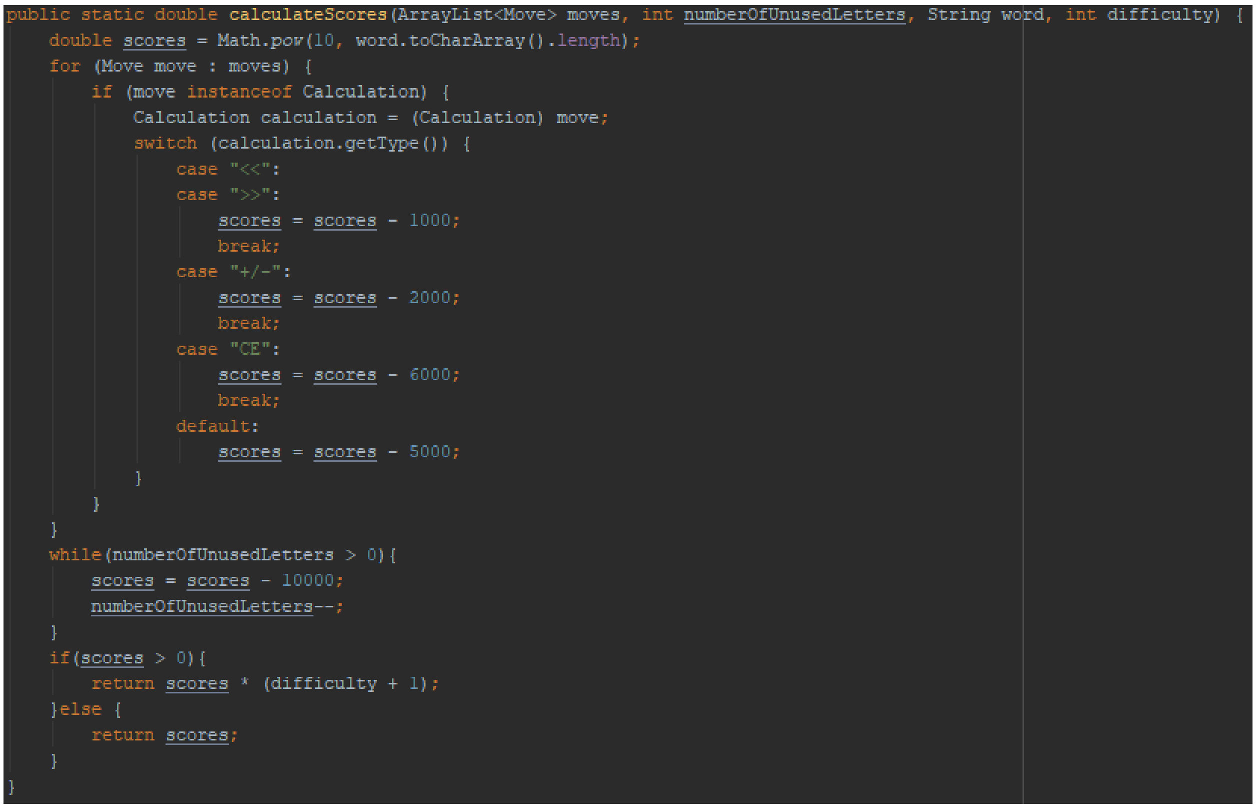
In order to optimize word validation, the English dictionary is stored in a java TreeSet collection. However, creating such collection is operationally costly. (Oracle, 2019d)



```java
public static TreeSet<String> loadWordsFromTextFile(InputStream inputStream, ProgressBar progressBar){
    TreeSet<String> words = new TreeSet<>();
    if(wordList == null && inputStream != null){
        try {
            BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(inputStream));
            String st = "";
            int rowNr = 0;
            while ((st = bufferedReader.readLine()) != null){
                words.add(st);
                rowNr++;
                progressBar.setProgress(rowNr  * 100 / 370113);
            }
        }catch (IOException e){

        }
        wordList = new WordList(words);
    }
    return wordsTree;
}
```

Figure 8 Load words from text file method

Figure 8 Load words from text file method shows the implementation of a method that converts the given file into TreeSet. English words are stored in a text file located in the assets of the application. The method is called in a separate thread in order to not block the main thread with time consuming operations. Additionally, the method takes a progress bar as an argument and updates it according to the current execution progress to inform the user that the application is still working and responding. Number 370133 is the number of total lines in the text file and it is a hardcoded value, because this value is not going to change and counting lines before parsing would extend execution time.



```java
public static double calculateScores(ArrayList<Move> moves, int numberOfUnusedLetters, String word, int difficulty) {
    double scores = Math.pow(10, word.toCharArray().length);
    for (Move move : moves) {
        if (move instanceof Calculation) {
            Calculation calculation = (Calculation) move;
            switch (calculation.getType()) {
                case "<<":
                case ">>":
                    scores = scores - 1000;
                    break;
                case "+/-":
                    scores = scores - 2000;
                    break;
                case "CE":
                    scores = scores - 6000;
                    break;
                default:
                    scores = scores - 5000;
            }
        }
    }
    while(numberOfUnusedLetters > 0){
        scores = scores - 10000;
        numberOfUnusedLetters--;
    }
    if(scores > 0){
        return scores * (difficulty + 1);
    }else {
        return scores;
    }
}
```

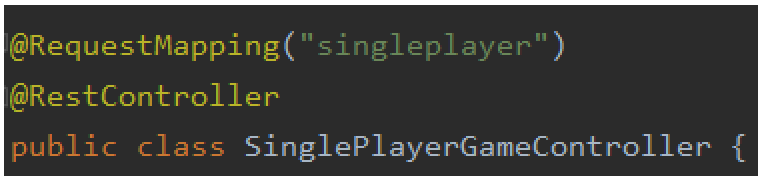
Figure 9 Calculate scores method

Figure 9 Calculate scores method shows the implementation of the method responsible for calculating scores. Score calculator is designed to reward users that are trying to spell longer words and use more difficult operations. The goal of a score system is to promote high risk – high reward behaviour in order to make the game more exciting. The user is getting point punishments for making many operations and creating unused letters. Additionally, if the score result is not negative it will be multiplied by game difficulty.

Server calls

API endpoints

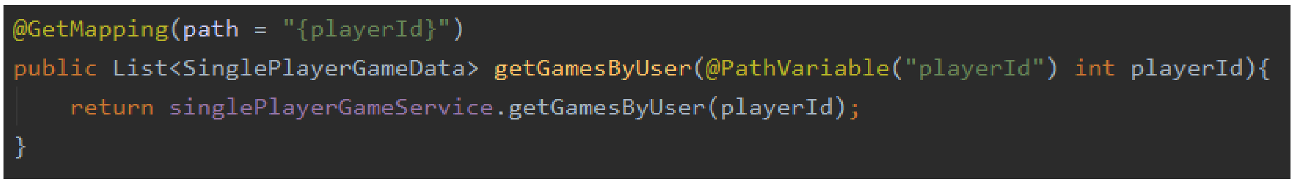
The endpoints are the methods that are going to be called by the clients. It is simple to set up an endpoint using Spring annotations. On Snippet 5 the SinglePlayerGameController is defined as a rest endpoint and the path that it can reached by the clients.



```java
@RequestMapping("singleplayer")
@RestController
public class SinglePlayerGameController {
```

Snippet 5

The annotations on Snippet 6 define the method type that this method is going to be reachable, like GET, POST, etc. and linking the path variables to the parameters. After that it just calls the service and passes the parameter to it.

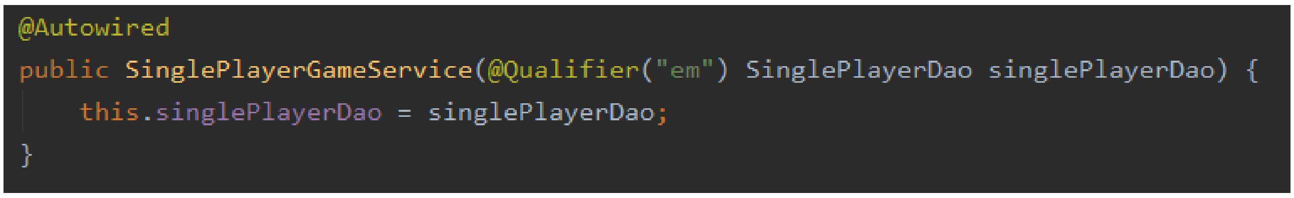


```java
@GetMapping(path = "{playerId}")
public List<SinglePlayerGameData> getGamesByUser(@PathVariable("playerId") int playerId){
    return singlePlayerGameService.getGamesByUser(playerId);
}
```

Snippet 6

Services

The constructor in SinglePlayerGameService uses dependency injection and that which implementation should be injected here, is defined by the string in the second annotation. The service could perform some modification on the passed data, but in this case data is just passed right to the DAO.

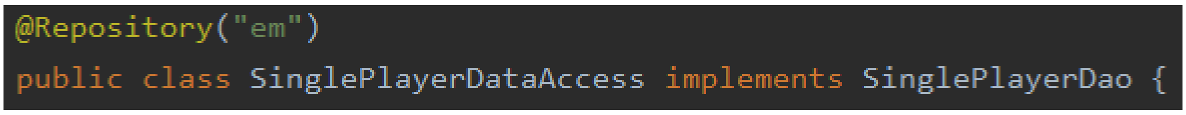


```java
@Autowired
public SinglePlayerGameService(@Qualifier("em") SinglePlayerDao singlePlayerDao) {
    this.singlePlayerDao = singlePlayerDao;
}
```

Snippet 7

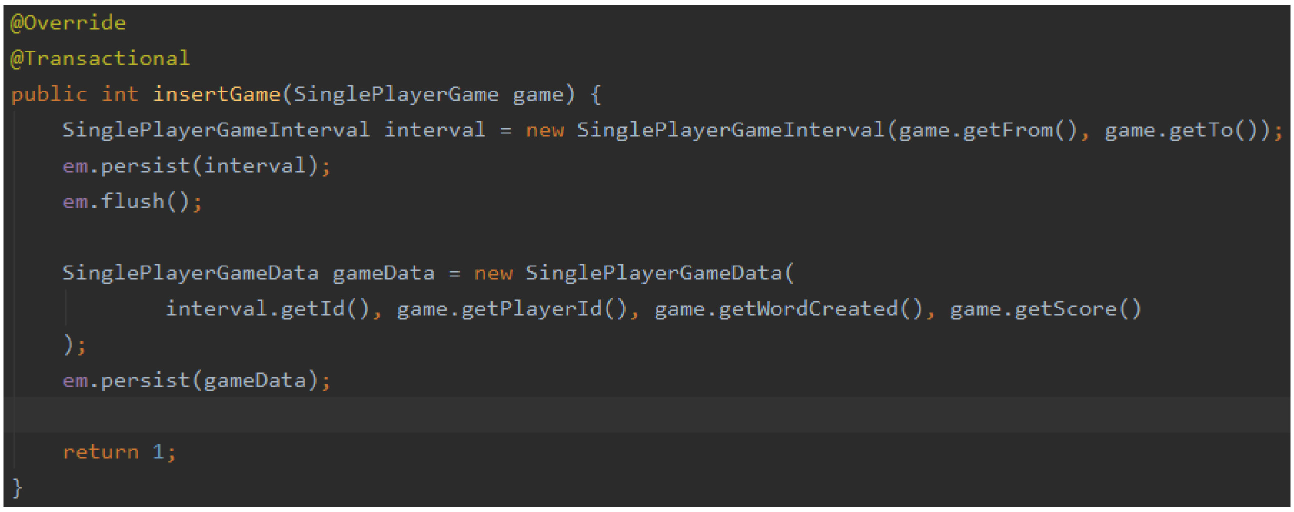
Data access

The string that identifies the implementation is used in the services can be seen in the annotation on Snippet 8.


```java
@Repository("em")
public class SinglePlayerDataAccess implements SinglePlayerDao {
```

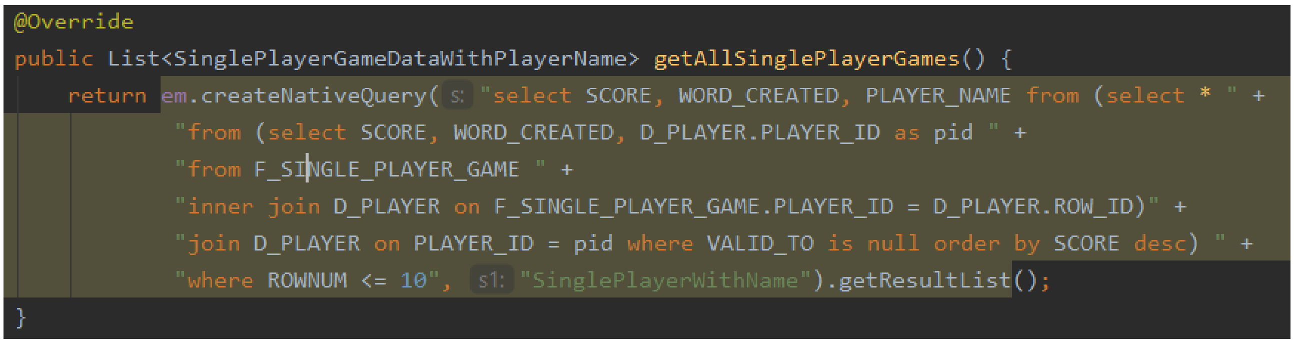
Snippet 8

During implementation, different approaches have been used to access data from the database. On Snippet 9 the JPA approach can be seen. At first, the object that needs to be stored is defined, then it is saved to the database. This step is necessary to create the next object, because the first insert's primary key will be generated by the database, and this primary key is going to be foreign key in the second object. When this key retrieved from the database, the second object can be created and saved to the database.


```java
@Override
@Transactional
public int insertGame(SinglePlayerGame game) {
    SinglePlayerGameInterval interval = new SinglePlayerGameInterval(game.getFrom(), game.getTo());
    em.persist(interval);
    em.flush();

    SinglePlayerGameData gameData = new SinglePlayerGameData(
            interval.getId(), game.getPlayerId(), game.getWordCreated(), game.getScore()
    );
    em.persist(gameData);

    return 1;
}
```

Snippet 9

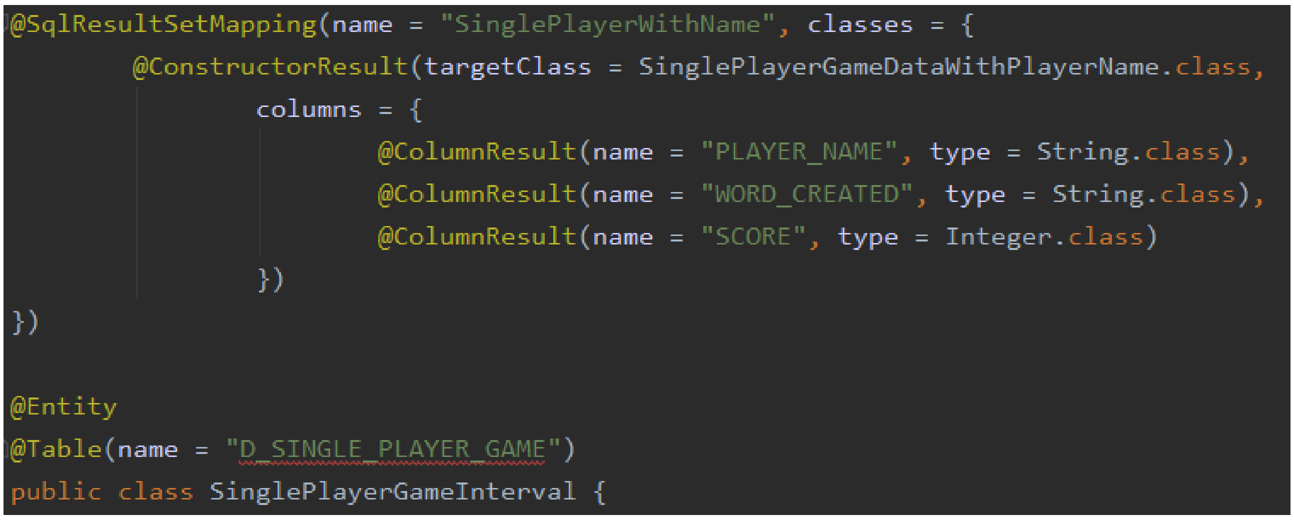
When the return type is a simple value, like int or String, or the returned data requires a more complex query, native queries have been used. On Snippet 10 the query will return the top 10 games globally. This required a complex query, which would have been difficult using JPA, but relatively simple using native queries.



```
@Override
public List<SinglePlayerGameDataWithPlayerName> getAllSinglePlayerGames() {
    return em.createNativeQuery( s: "select SCORE, WORD_CREATED, PLAYER_NAME from (select * " +
            "from (select SCORE, WORD_CREATED, D_PLAYER.PLAYER_ID as pid " +
            "from F_SINGLE_PLAYER_GAME " +
            "inner join D_PLAYER on F_SINGLE_PLAYER_GAME.PLAYER_ID = D_PLAYER.ROW_ID)" +
            "join D_PLAYER on PLAYER_ID = pid where VALID_TO is null order by SCORE desc) " +
            "where ROWNUM <= 10", s1: "SinglePlayerWithName").getResultList();
}
```

Snippet 10

This, and other similar queries return an object that has been made from using data from different tables, so it cannot be mapped to a java entity, even if an object with appropriate properties exist. On Snippet 11, a mapping had to be declared to map the result set of the query to a class. This mapping had to be declared in a class that is an entity in the database, Spring otherwise would ignore the annotation.



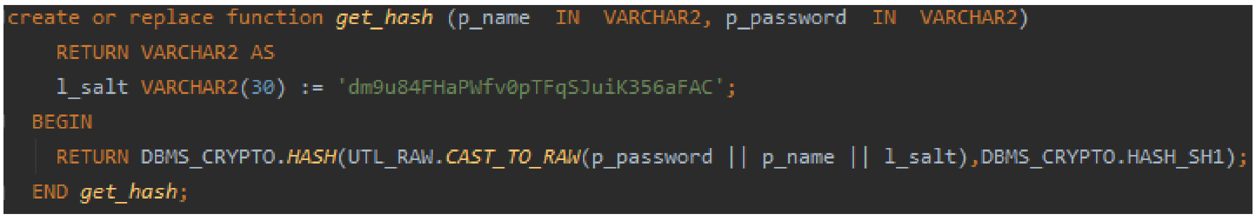
```
@SqlResultSetMapping(name = "SinglePlayerWithName", classes = {
        @ConstructorResult(targetClass = SinglePlayerGameDataWithPlayerName.class,
                columns = {
                        @ColumnResult(name = "PLAYER_NAME", type = String.class),
                        @ColumnResult(name = "WORD_CREATED", type = String.class),
                        @ColumnResult(name = "SCORE", type = Integer.class)
                })
})

@Entity
@Table(name = "D_SINGLE_PLAYER_GAME")
public class SinglePlayerGameInterval {
```

Snippet 11

Security

Due to the product owner's needs, data security in this project was not prioritized. That approach opens many potential threats of hacking attempts. However, some basic security was implemented as a matter of good practice.

In order to secure user passwords, this project uses Oracle DBMS hash function. This function will return an encrypted value in the form of a fourteen characters long hash. Popularity of this function means that some hashes are already decrypted and can be used to estimate user passwords. This is extremely dangerous for users that are using popular passwords or a password that contains real words or real words and numbers. For these reasons, the input of the DBMS hash function is a combination of user email address, password and noise. Figure 10 Get hash function shows the implementation of the get hash function. (Oracle, 2019b)



```
create or replace function get_hash (p_name  IN  VARCHAR2, p_password  IN  VARCHAR2)
    RETURN VARCHAR2 AS
    l_salt VARCHAR2(30) := 'dm9u84FHaPWfv0pTFqSJuiK356aFAC';
  BEGIN
    RETURN DBMS_CRYPTO.HASH(UTL_RAW.CAST_TO_RAW(p_password || p_name || l_salt),DBMS_CRYPTO.HASH_SH1);
  END get_hash;
```

Figure 10 Get hash function

Local sensitive data storage is handled by the Android AccountManager. AccountManager provides safe account detail storage and authentication token storage. However, authentication token is not implemented on the server side. Instead, database user Id is used directly. This leaves opportunity for hackers to login to any account by decompiling an apk file and hard code any lucky guessed user id. For that purpose, a decision to restrict functionalities of the application profile management was made. The user's email address won't be downloaded from the server and the user won't be able to change the email or password. This temporary solution should minimize damage in case of an account being stolen. (Google, 2019a)

One of the most common attack on systems that use SQL database is called SQL injection. Its popularity is caused by the simplicity of the attack. This attack assumes that user input is directly inserted into the query. For example in query "select * from user where user_name = *user_input"* instead of brand name user can insert " 'x'); drop database; --" or " 'x' or 1 = 1". Allowing the user to directly control the system database is a big gap in security. To avoid situation as such, prepared statements(Menon, n.d.) were used. Prepared statements will treat user input as value and not as part of the query. Preventing SQL injection attacks is not only a functionality of prepared statements and this is why the decision to use them was made even though security in this project has low priority. Before executing each query, Oracle DBMS prepares the execution plan according to the data stored in its dynamic performance views and in the static data dictionary. However, preparing an execution plan can be time-consuming for the query. In case of using a prepared statement, the database will keep the execution plan and reuse it, reducing the operation cost of the query. (Microsoft, 2019)

Additional problematic security issue is the fact that web services used on the server side are not using https protocols and that opens up the system to man-in-the-middle attacks, especially when the user is connected to a public Wi-Fi. (Callegati, Cerroni and Ramilli, 2009)

Test

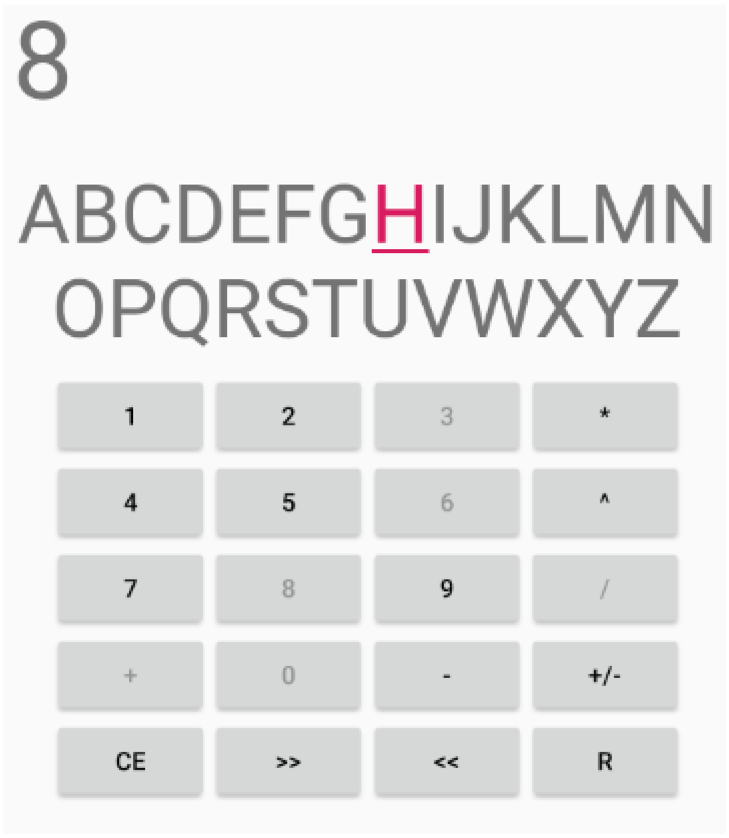
This section will cover different types of tests that were done during the test phase. The first type of test is a Use case test and later the unit tests are covered. Finally, different REST clients can be seen, and how they were used for testing in the early and later stages of the project.

Use case test

The use case that has been chosen for testing was the Classic Mode, because this contains the core mechanics of the app and everything else was built around it, so it should perform well for an enjoyable user experience.

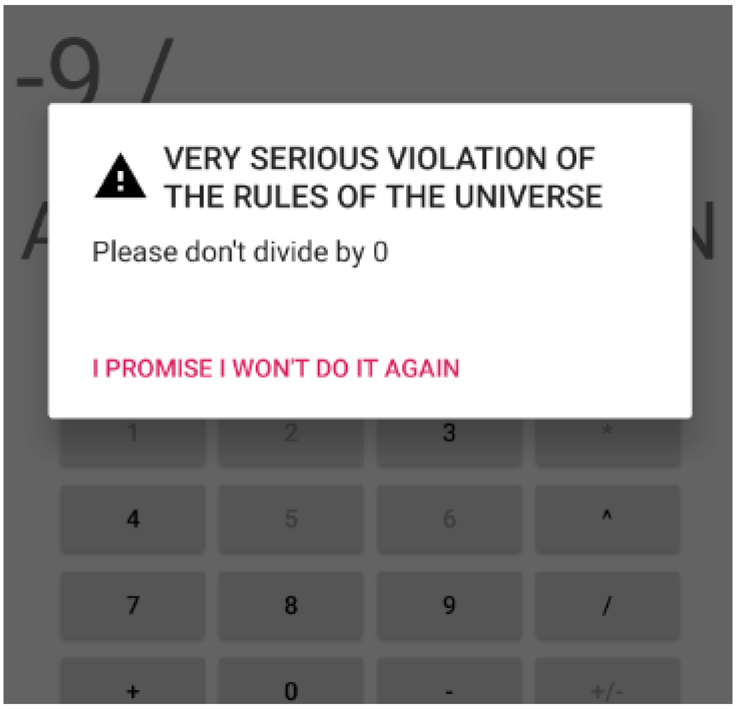
The end goal of this use case is to show the user the score that has been achieved, but to be more thorough considering the user experience, saving the score to the database and retrieving that information has been included as well.

Step 1 is to use the numbers and operations to create letters, and if the user is satisfied with the calculation, click on the highlighted letter to add it to the letter collection.



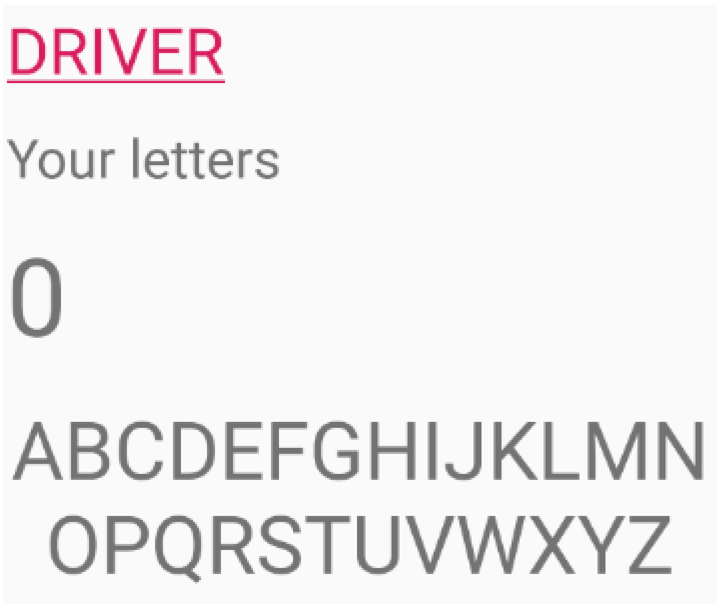
Snippet 12

If the user performs calculations that are not valid, those are handled by the application. For example, if the user tries to divide by 0, a pop-up is shown to the user saying that the current calculation is not acceptable.


-9 /
VERY SERIOUS VIOLATION OF THE RULES OF THE UNIVERSE
Please don't divide by 0
I PROMISE I WON'T DO IT AGAIN
1
2
3
*
4
5
6
^
7
8
9
/
+
0
-
+/-

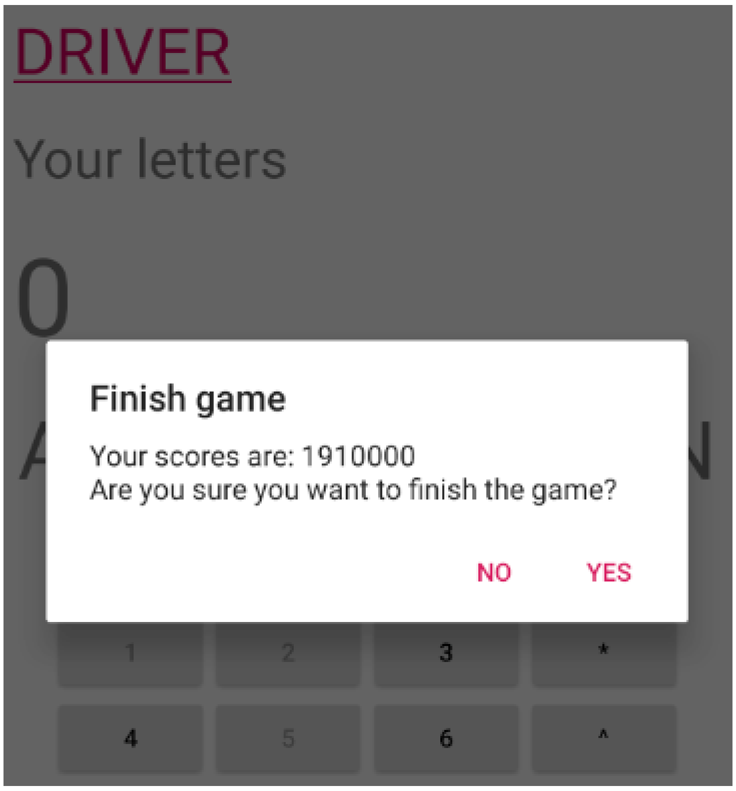
Snippet 13

Step 2 is to create more letters and put them in an order that forms an understandable word.



Snippet 14

Step 3, if the user is satisfied with the created word, then the word needs to be clicked and confirmed to finish the game.



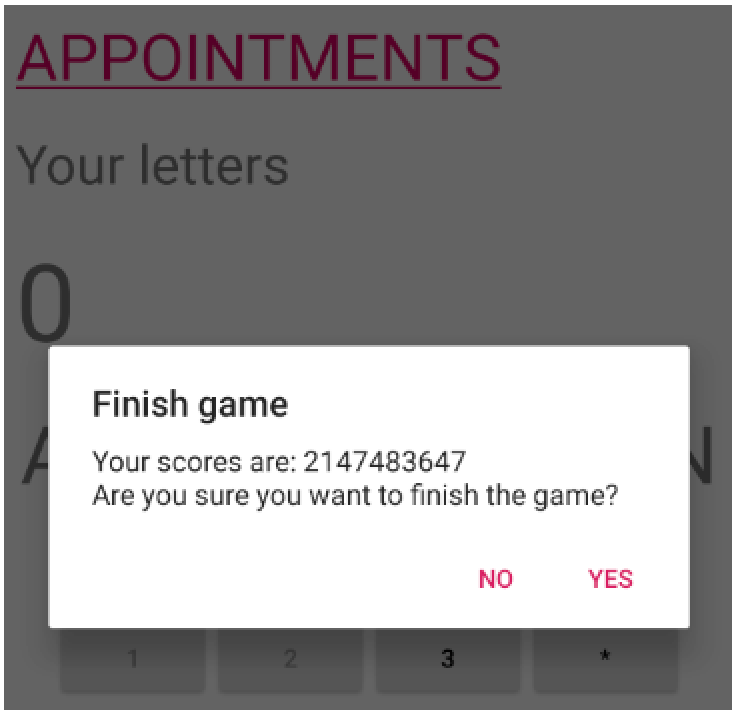
Finish game

Your scores are: 1910000
Are you sure you want to finish the game?

NO YES

Snippet 15

If the user reaches a really high score, the app is going to cap at the maximum value of a 32-bit integer.

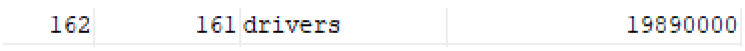


Finish game

Your scores are: 2147483647
Are you sure you want to finish the game?

Snippet 16

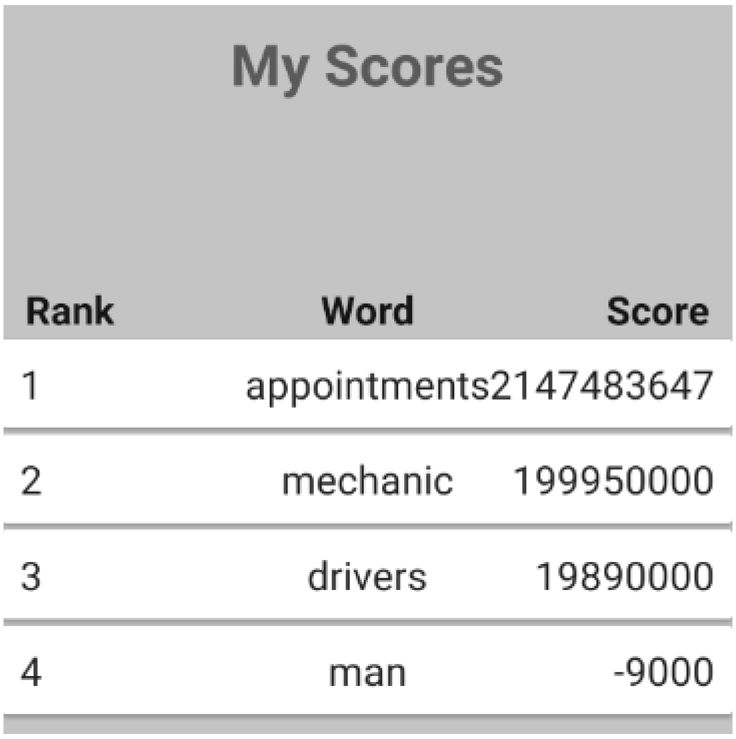
When the game is done, the data is sent to the server and saved to the database.



```
    162         161 drivers                          19890000
```

Snippet 17

These scores later can be retrieved from the database and displayed on the My score or Global scores pages.



My Scores

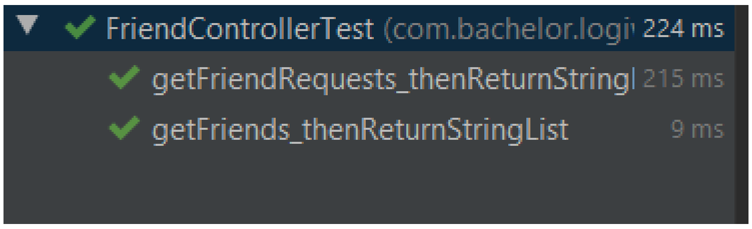
Rank	Word	Score
1	appointments	2147483647
2	mechanic	199950000
3	drivers	19890000
4	man	-9000

Snippet 18

Unit tests

Due to time restrictions, the testing phase of the project is not completed, integration tests, performance tests and some of the unit tests are missing from the current implementation of the project.

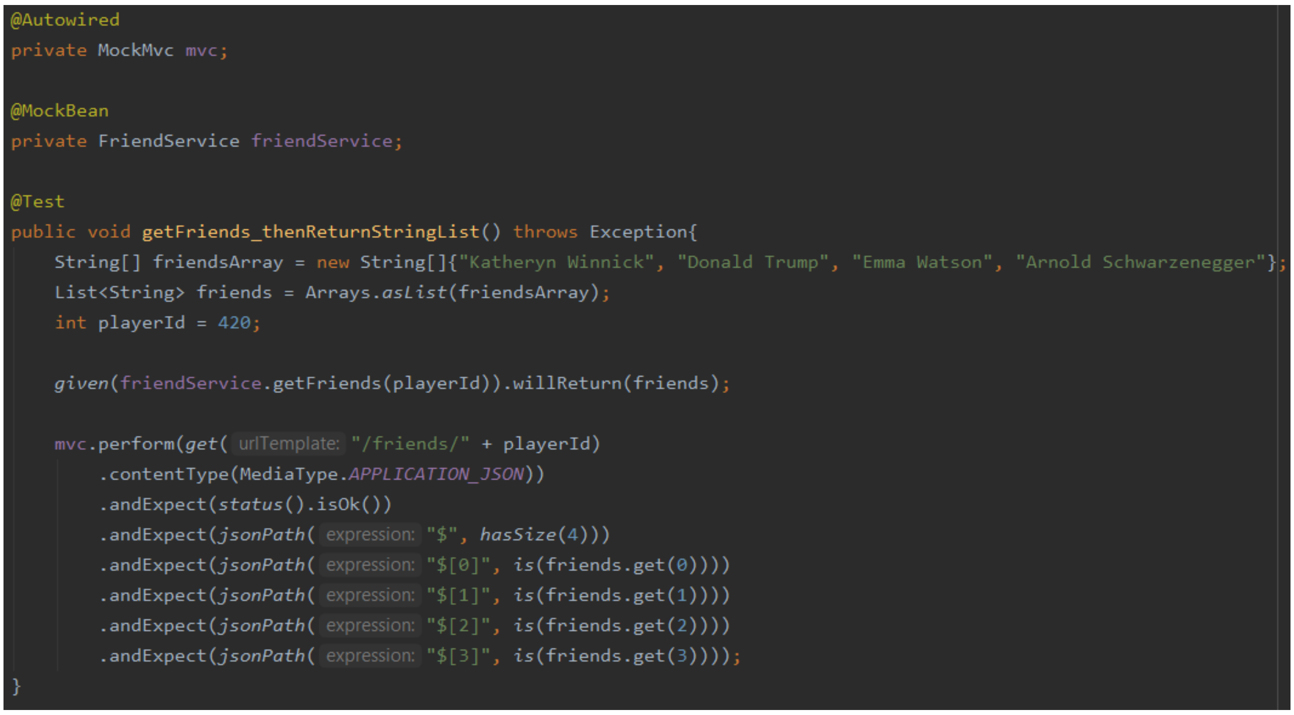
Unit tests have been started on the client and the server side as well, but in the future these tests will need to be expanded. On the server side, the FriendController was tested, which takes care of the API endpoint for friend management. It was chosen because out of all the endpoints, this one includes the most operations and it takes care of the whole friend management system.



```
▼ ✓ FriendControllerTest (com.bachelor.logi 224 ms
      ✓ getFriendRequests_thenReturnStringl 215 ms
      ✓ getFriends_thenReturnStringList       9 ms
```

Snippet 19

The performed tests on the controller were testing the GET methods, as these are the only ones in the controller that have return types. Error handling or notifying the user if something went wrong on the server is currently not implemented.



```
@Autowired
private MockMvc mvc;

@MockBean
private FriendService friendService;

@Test
public void getFriends_thenReturnStringList() throws Exception{
    String[] friendsArray = new String[]{"Katheryn Winnick", "Donald Trump", "Emma Watson", "Arnold Schwarzenegger"};
    List<String> friends = Arrays.asList(friendsArray);
    int playerId = 420;

    given(friendService.getFriends(playerId)).willReturn(friends);

    mvc.perform(get( urlTemplate: "/friends/" + playerId)
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath( expression: "$", hasSize(4)))
            .andExpect(jsonPath( expression: "$[0]", is(friends.get(0))))
            .andExpect(jsonPath( expression: "$[1]", is(friends.get(1))))
            .andExpect(jsonPath( expression: "$[2]", is(friends.get(2))))
            .andExpect(jsonPath( expression: "$[3]", is(friends.get(3))));
}
```

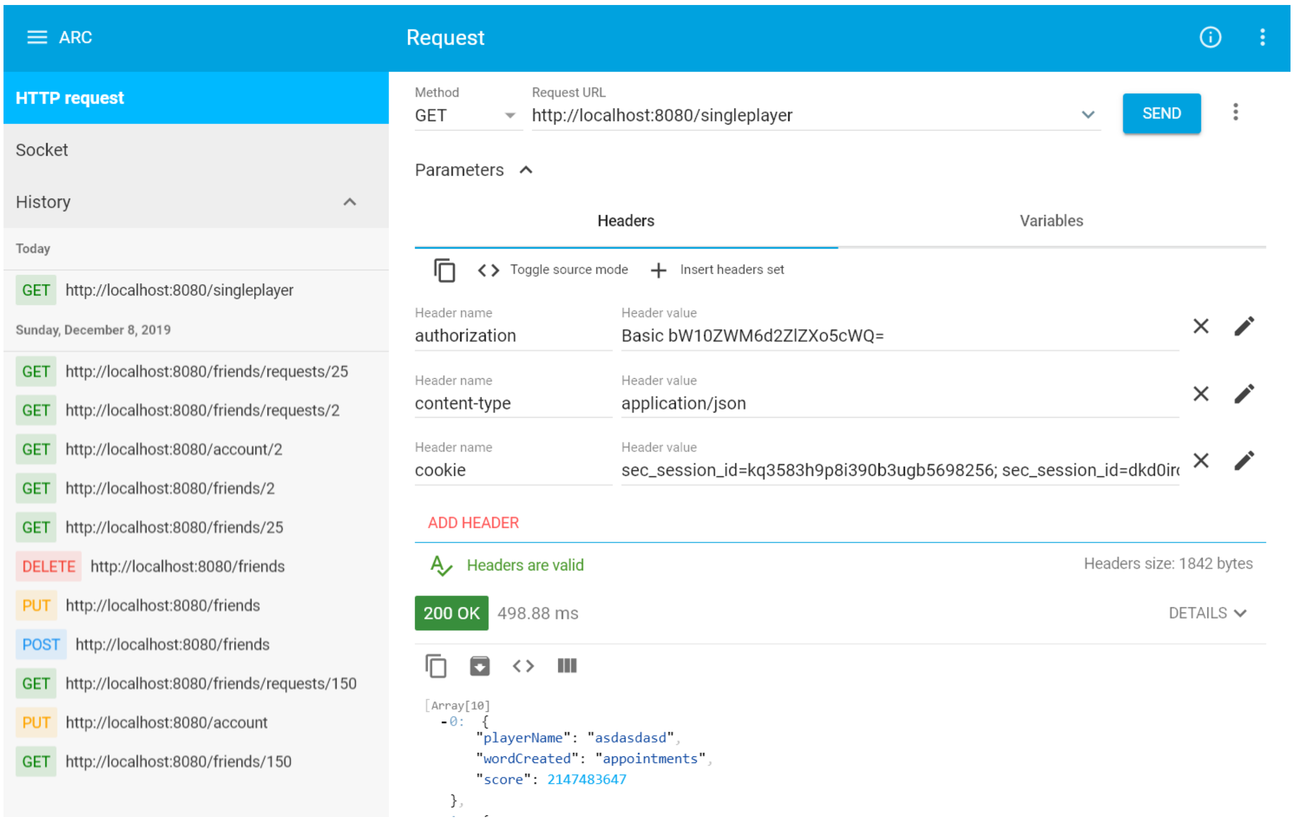
Snippet 20

On Snippet 20 the test that checks if the correct JSON values have been returned can be seen. The GET method tests are similar to each other, that's why only the getFriends test have been included. The FriendService has been mocked, so it will return dummy data, then the controller is checked if it returns the correct status code and the correct data.

Postman & Advanced REST client

At the early stages of the implementation phase, the client and the server side were developed separately. Because the REST calls have not been implemented on the client side early on, there was no way to check if the server returns the correct data. To still be able to progress with the back-end development, REST clients have been used to imitate an actual client's call.

These clients have great UIs, where at first the URL needs to be entered, the address where the server can be reached. Next, the HTTP methods can be selected, which will specify what type of operation should be performed on the server. If necessary, additional information can be sent in the header or in the body. All recent calls are easily accessible, so it is simple to change between calls.



Today

GET	http://localhost:8080/singleplayer

Sunday, December 8, 2019

GET	http://localhost:8080/friends/requests/25
GET	http://localhost:8080/friends/requests/2
GET	http://localhost:8080/account/2
GET	http://localhost:8080/friends/2
GET	http://localhost:8080/friends/25
DELETE	http://localhost:8080/friends
PUT	http://localhost:8080/friends
POST	http://localhost:8080/friends
GET	http://localhost:8080/friends/requests/150
PUT	http://localhost:8080/account
GET	http://localhost:8080/friends/150

Method: GET
Request URL: http://localhost:8080/singleplayer

Header name	Header value
authorization	Basic bW10ZWM6d2ZlZXo5cWQ=
content-type	application/json
cookie	sec_session_id=kq3583h9p8i390b3ugb5698256; sec_session_id=dkd0ir

Headers are valid
Headers size: 1842 bytes

200 OK 498.88 ms

```
[Array[10]
  -0:  {
      "playerName": "asdasdasd",
      "wordCreated": "appointments",
      "score": 2147483647
    },
```

Figure 11 Advanced REST client

The REST clients were used in the later stages of development as well, to compare the data between the REST clients and the actual client. This way it was easy to detect if the results differ and know that there are some problems on the client side.

Results and Discussion

The section will present the requirements in a table. The table will show if the requirements have been fulfilled or not. In the end the results will be discussed, even if they were not implemented.

Results

The section will cover the functional and non-functional requirements, but it will not discuss the results, only if they were fulfilled or not.

ID	**Requirement**	**Completed**
1	The application must be able to train the user's brain	Yes
2	The application must use mathematical calculations to make letters and then formulate words out of them.	Yes
3	The system must assign scores based on the length of the words and the complexity of the calculations.	Yes
4	The application must have a calculator layout with the capability to display the letters and later reorganize them in optional order to create a word.	Yes
5	The application must have the functionality to save all the data on the server.	Yes
6	The user must have unique identification for login purposes.	Yes
7	The user should be able to play against a random opponent in real-time.	No
8	The user should be able to manage friends within the application.	Yes

9	The friends should be challengeable in a match.	No
10	The users should be able to access the top scores of other users on a scoreboard.	Yes
11	The data should be accessible for the application at any time	No
12	The user statistics should be updated at any time when there is a change and the phone is connected to the internet.	Yes
13	The application should have daily word challenges for the users.	Yes
14	The application should have a tutorial what could teach the user how to use the functionalities.	No
15	An achievement system should be implemented in the game.	No
16	The application should be capable to help the user in the next step when the person is stuck during a match.	No
17	Multi Language support should be implemented for the application.	No
18	The main steps in each match should be checkable for the user.	No

Table 6 Functional requirement results

Table 6 Functional requirement results represents the functional requirements and if they are fulfilled or not. Some results will be discussed later in the *discussion* section.

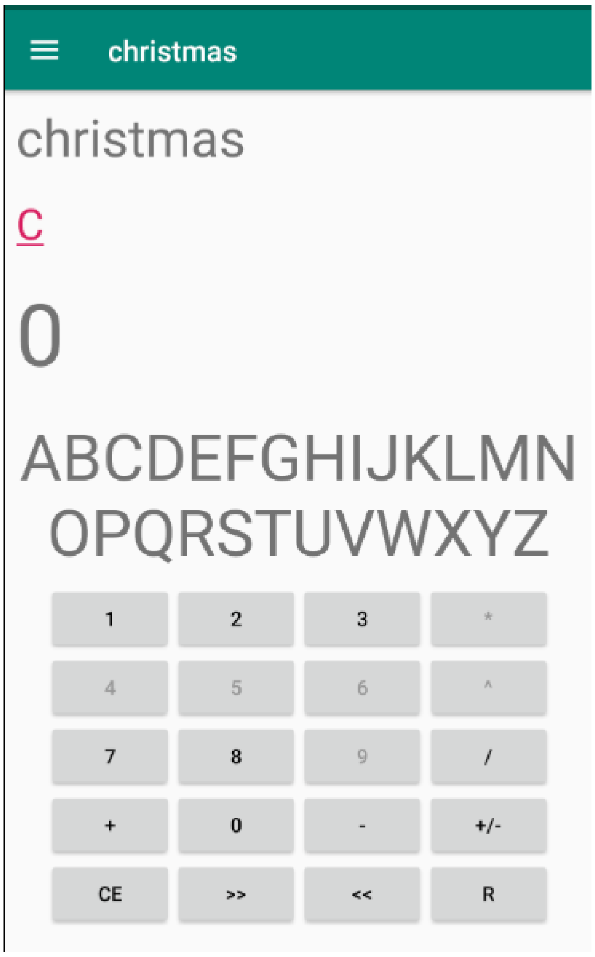
ID	Requirements	Completed
1	The system should be capable to protect the data from hacking attempts	No

Table 7 Non-functional requirements results

Table 7 Non-functional requirements results represents the result of the non-functional requirements.

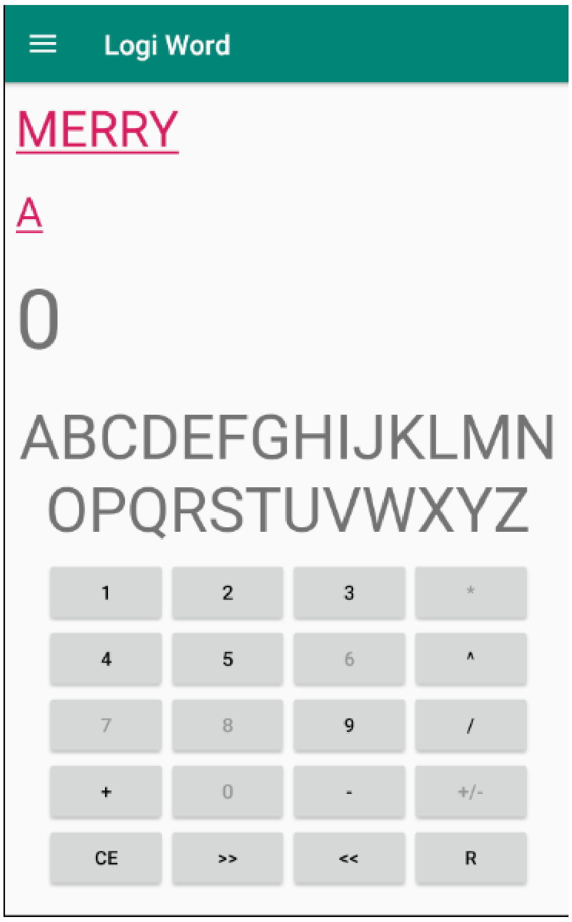
Product results

In this part of the documentation, the result of the final product can be seen.



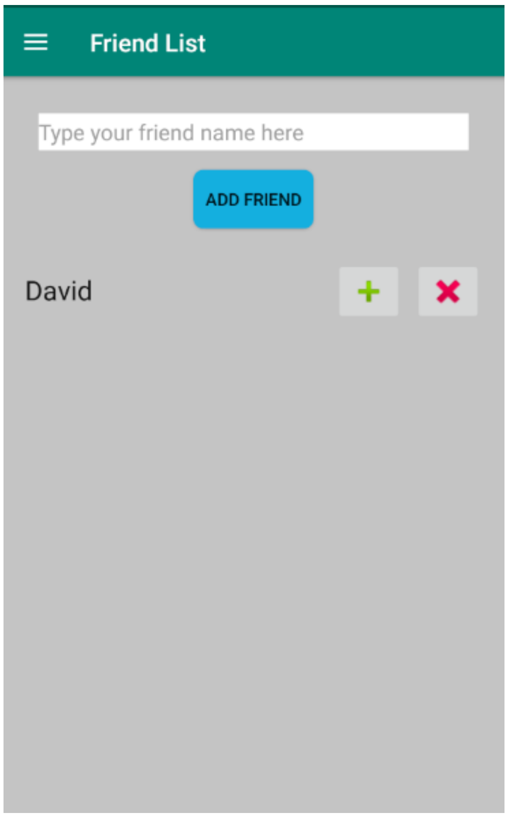
Snippet 21

Snippet 21 shows the result of the daily word challenge implementation. The user gets a word from the server and must spell it in order to succeed in the game and get a score.



Snippet 22

Snippet 22 shows the result of the classic single player implementation. The user can form any valid word and gets a score for it.



Snippet 23

Snippet 23 shows the result of the friend list implementation. The user can send friend requests, respond to other player's requests or later remove them from the list.

Discussion

This section will discuss some of the results that are presented in the *Results* section. Some results from the functional requirements will be discussed.

Functional results

Req-ID-3 the implementation for this requirement was estimated to take a long time. The difficulty of this requirement comes from how to retrieve the history of steps that are needed to give a fair score to the user. In the end, this requirement has been solved with scores that are representing how complex calculation the user could use and how long are the spelled words.

Req-ID-4 was a complicated requirement on the client side because the features were complex to implement. However, the requirement has been fulfilled and the player is capable to reorganize all the letters that are calculated and put them in an optional order. It is also possible to revert the letters so they can be reorganized again. The actual calculator layout was not hard to implement but because of the lot of elements on the screen and that all of them had to have functionalities, it was time consuming.

Req-ID-5 was one of the major functionalities that had to be done and needed a reliable communication between the client and the server. On the client side, the data is coming from the user's input and from the server. The back-end retrieves the information, processes them and then stores it in the Oracle Database for later usages.

Req-ID-6 was a key functionality to have, because the whole application was built on multiple users. The user is capable to make an account that will be stored on the server. The player gets a custom player ID that is used to identify the user and for communication with the back-end.

Conclusion

During the project period, the team worked hard to reach their final goal to create a system that will fulfil the product owner's requirements.

All must-have requirements of the system were fully implemented. Those requirements were considered critical and received the highest priority. Other requirements depend directly on them and future system development would not be possible, therefore the project couldn't be considered done.

Should-have requirements were also treated with high priority, however not fulfilling them would not be considered a failed project and they could be further developed in the future. Most of the should-have requirements were dependent on client-server communication, therefore fulfilling one of them would mean that it is easier to accomplish the other requirements. Due to time limitation, after discussions and considerations the developer team made the difficult decision to not implement two of them that were considered the lowest priority in the should-haves. Those requirements were based on real time multiplayer that after the implementation of the game mechanics and many experiments with those, the other requirements were considered more beneficial for the user.

The most important part of the system was to implement game mechanics that will satisfy the product owner needs. The downside of the unique idea behind this project was that it is difficult to place this application in any category that is not a logical or puzzle game, therefore there were no examples that the developer team could get inspiration from. Additionally, there was a risk that the application would be completely unintuitive or unentertaining for the users. In such case implementing other functionalities of the system will serve no purpose. Therefore, extra attention was invested in this part of the system and these requirements were marked as critical.

Additional challenge was to implement the back-end of the application. Most of the requirements are internet based, therefore implementing back-end of the application was an important part of this project. The server side of the application was considered critical. The future development of this project depends on the flexibility of the back-end and analysis of the stored data. The database was designed with the goal to contain data about user behaviour in the system and being compatible with data analysis tools. Therefore, the decision whether this application should be improved can be easily made based on the collected and analysed data. Furthermore, analysing the collected data can help find and prioritize problems or additional requirements.

In conclusion, the developer team was satisfied with the result of this project and is looking forward to continue improving it afterwards.

Project future

The system achieved many of the planned goals. Both technical and mathematical features were implemented, and these implementations could give a solid foundation for the future of the project. Some of the requirements were not implemented because of the lack of time during the project period but they give a good overview of what could happen with the application in the future.

One major part of the software would be to implement a multiplayer where the users could play against each other in a random match. A real time networking system has to be implemented between the clients in order to succeed with this plan, but it would make the application more enjoyable since the players could play with each other and not just compare their scores.

Moreover, the friend system could be developed further, because so far, the users are only allowed to accept and remove friends. When the actual multiplayer is implemented, this feature could have the possibility to invite a friend to a match and even see their online/offline status.

A local cache storage would be great for the user because the data would be accessible at any time, so the player does not have to be online to play with the single player game mode. Later, the scores and information could be synchronized with the server.

In order to reach more users, the application could have multi language support. The basics of this feature are implemented on the client side, however it is still far from an actual support for more languages. Also, many modifications have to be made on the back-end as well.

Security is an issue at the moment with the system. Even if it was not highly prioritized during the implementation of the product, later it would be important to solve this issue. Firstly, the http requests should be changed to https requests to counter the man-in-the-middle attack. Secondly, the client itself stores some raw data what should be solved later. Lastly, the log-in system could be optimized in a way that it won’t leak any data.

Sources of information

A. Tsois, N. Karayannidis, T.S., n.d. *MAC: Conceptual Data Modeling for OLAP*. [online] Available at: <http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-39/>.

Alexander Shvets, 2019. *Adapter Design Pattern*. [online] Available at: <https://sourcemaking.com/design_patterns/adapter> [Accessed 10 Dec. 2019].

Ampire, E., 2019. *MVVM on Android with the Architecture Components + Koin*. [online] Available at: <https://medium.com/swlh/mvvm-on-android-with-the-architecture-components-koin-f53c3c200363> [Accessed 15 Dec. 2019].

baeldung, 2019. *The DAO Pattern in Java | Baeldung*. [online] Available at: <https://www.baeldung.com/java-dao-pattern> [Accessed 15 Dec. 2019].

Bhavya Karia, 2019. *A quick intro to Dependency Injection: what it is, and when to use it*. [online] Available at: <https://www.freecodecamp.org/news/a-quick-intro-to-dependency-injection-what-it-is-and-when-to-use-it-7578c84fa88f/> [Accessed 15 Dec. 2019].

Callegati, F., Cerroni, W. and Ramilli, M., 2009. *Man-in-the-middle attack to the HTTPS protocol*. *IEEE Security and Privacy*.

educba, 2019. *What is mvc design pattern | How IT Works | Skills & Scope | Advantages*. [online] Available at: <https://www.educba.com/what-is-mvc-design-pattern/> [Accessed 15 Dec. 2019].

Google, 2019a. *AccountManager | Android Developers*. [online] Available at: <https://developer.android.com/reference/android/accounts/AccountManager.html> [Accessed 10 Dec. 2019].

Google, 2019b. *Create a List with RecyclerView | Android Developers*. [online] Available at: <https://developer.android.com/guide/topics/ui/layout/recyclerview> [Accessed 10 Dec. 2019].

Google, 2019c. *RecyclerView.Adapter | Android Developers*. [online] Available at: <https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.Adapter.html> [Accessed 10 Dec. 2019].

Google, 2019d. *ViewModel Overview | Android Developers*. [online] Available at: <https://developer.android.com/topic/libraries/architecture/viewmodel> [Accessed 15 Dec. 2019].

indeed, 2019. *The Best Ways to Strengthen Your Logical Thinking Skills | Indeed.com*. [online] Available at: <https://www.indeed.com/career-advice/career-

development/strengthen-logical-thinking-skills> [Accessed 15 Dec. 2019].

Kimball, R., Reeves, L., Ross, M. and Thornthwaite, W., 2008. *The Data Warehouse Lifecycle Toolkit*. *WILEY*.

Lars Vogel, Simon Scholz, D.W., 2019. *Using Retrofit 2.x as REST client - Tutorial*. [online] Available at: <https://www.vogella.com/tutorials/Retrofit/article.html> [Accessed 15 Dec. 2019].

Mark Townsend, 2012. Exploring the Singleton Design Pattern. [online] Available at: <https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee817670(v=pandp.10)?redirectedfrom=MSDN>.

McCallum, J.C., 2019. *Disk Drive Prices (1955-2019)*. [online] Available at: <https://jcmit.net/diskprice.htm> [Accessed 9 Dec. 2019].

Menon, R., n.d. *Statements and Prepared Statements*. [online] Available at: <https://www.oracle.com/technetwork/testcontent/jdbc-ch5-131209.pdf>.

Microsoft, 2019. *SQL Injection | Microsoft Docs*. [online] Available at: <https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms161953(v=sql.105)?redirectedfrom=MSDN> [Accessed 10 Dec. 2019].

Nouchi, R., Taki, Y., Takeuchi, H., Hashizume, H., Nozawa, T., Kambara, T., Sekiguchi, A., Miyauchi, C.M., Kotozaki, Y., Nouchi, H. and Kawashima, R., 2013. Brain Training Game Boosts Executive Functions, Working Memory and Processing Speed in the Young Adults: A Randomized Controlled Trial. *PLoS ONE*.

Oei, A.C. and Patterson, M.D., 2013. Enhancing Cognition with Video Games: A Multiple Game Training Study. *PLoS ONE*, [online] 8(3), p.e58546. Available at: <https://dx.plos.org/10.1371/journal.pone.0058546>.

Oracle, 2019a. *Data Blocks, Extents, and Segments*. [online] Available at: <https://docs.oracle.com/cd/B19306_01/server.102/b14220/logical.htm> [Accessed 9 Dec. 2019].

Oracle, 2019b. *DBMS_CRYPTO*. [online] Available at: <https://docs.oracle.com/database/121/ARPLS/d_crypto.htm#ARPLS65669> [Accessed 10 Dec. 2019].

Oracle, 2019c. *Optimization of Joins*. [online] Available at: <https://docs.oracle.com/cd/F49540_01/DOC/server.815/a67781/c20c_joi.htm#2437> [Accessed 9 Dec. 2019].

Oracle, 2019d. *TreeSet (Java Platform SE 7)*. [online] Available at:

<https://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html> [Accessed 10 Dec. 2019].

Pragati Singh, 2019. *Difference between MVC and MVVM. - mobidroid - Medium*. [online] Available at: <https://medium.com/mobidroid/difference-between-mvc-and-mvvm-456ec67181f6> [Accessed 15 Dec. 2019].

Sachin Kumar, 2019. *MVVM architecture using android architecture components*. [online] Available at: <https://medium.com/mindorks/mvvm-architecture-using-android-architecture-components-212af860a4bc> [Accessed 15 Dec. 2019].

Shubham, 2019. *DAO Design Pattern - JournalDev*. [online] Available at: <https://www.journaldev.com/16813/dao-design-pattern> [Accessed 15 Dec. 2019].

Simon LH, 2019. *SOLID Principles: Explanation and examples - ITNEXT*. [online] Available at: <https://itnext.io/solid-principles-explanation-and-examples-715b975dcad4> [Accessed 15 Dec. 2019].

statcounter, 2019. *Operating System Market Share Worldwide | StatCounter Global Stats*. [online] Available at: <https://gs.statcounter.com/os-market-share#quarterly-201903-201903-map> [Accessed 15 Dec. 2019].

tutorialspoint, 2019a. *Data Access Object Pattern - Tutorialspoint*. [online] Available at: <https://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm> [Accessed 15 Dec. 2019].

tutorialspoint, 2019b. *Dependency Injection*. [online] Available at: <https://www.tutorialsteacher.com/ioc/dependency-injection> [Accessed 15 Dec. 2019].

tutorialspoint, 2019c. *Spring Framework - Overview - Tutorialspoint*. [online] Available at: <https://www.tutorialspoint.com/spring/spring_overview.htm?fbclid=IwAR0Y63FGQCVomuXSDIFq25jDsg3AEDjt6-9G2vHT1laSembA3TLfUB-ZJOc> [Accessed 15 Dec. 2019].

Tyson, M., 2019. *What is JPA? Introduction to the Java Persistence API | JavaWorld*. [online] Available at: <https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html?fbclid=IwAR1rD9qHTjxgqJ77WRhBqercBHigLCV09CZ99TgD7KbmhNYK_o3tlbgdop8> [Accessed 15 Dec. 2019].