**Software Design Specification**

**GameSAT**

Andris Docaj

Mostofa Adib Shakib

Ayush Dhumal

Scott McFarlane

Shree Shrestha

**Table of Contents**

1. **Project description3**
2. **System analysis and decomposition4**
3. **UML diagrams5**
   1. **Use Case diagrams5**
   2. **Class diagrams7**
   3. **Sequence diagrams12**
4. **Low fidelity prototype14**
5. **Implementation plan19**
6. **Project description**

GameSAT is an android device app which intends on making learning fun. This aim is achieved by transforming learning into an enjoyable play, where the user’s competitive aspects are challenged. The main intention of GameSAT is to prepare high school students for the verbal section of the SAT exam, or undergraduate students for the verbal section of the GRE exam.

Our app will first require the user to create an account so that their progress can be traced. Once an account has been created the user is then presented with the option of playing or training. The playing option allows the user to play a word game, which challenges the user’s vocabulary comprehension of words. In addition to this, the user can also play a passage game, where randomly chosen short passages, obtained from well known magazines such as Nautilus, and Quanta, are presented followed by questions. The questions will be related to the passage and will test the user’s reading comprehension. The play mode will start the user with 5 points and the maximum number of points that the game allows is 50. If the score reaches zero, the game is over. The game will have many levels, and each level will challenge the user by imposing a time limit on the answer window. This time limit will reduce as higher levels are reached. If the user misses a question they will be penalized, but a correctly found answer will lead them to a reward. In addition, if the user obtains five correct answers in a row, they will be rewarded a bonus point. The users will have the option to skip questions they find too difficult and avoid penalty.

In training mode no scoring is kept, and no time limit exists. The training is available for words individually but also for passages of text. The training mode is strictly designed for free learning, without any pressure. In this mode the questions are randomly chosen, but if a question is missed it will show up more often than others, until the user correctly selects for a particular number of times.

1. **System analysis and decomposition**

The following picture reveals our ideas regarding the system architecture.

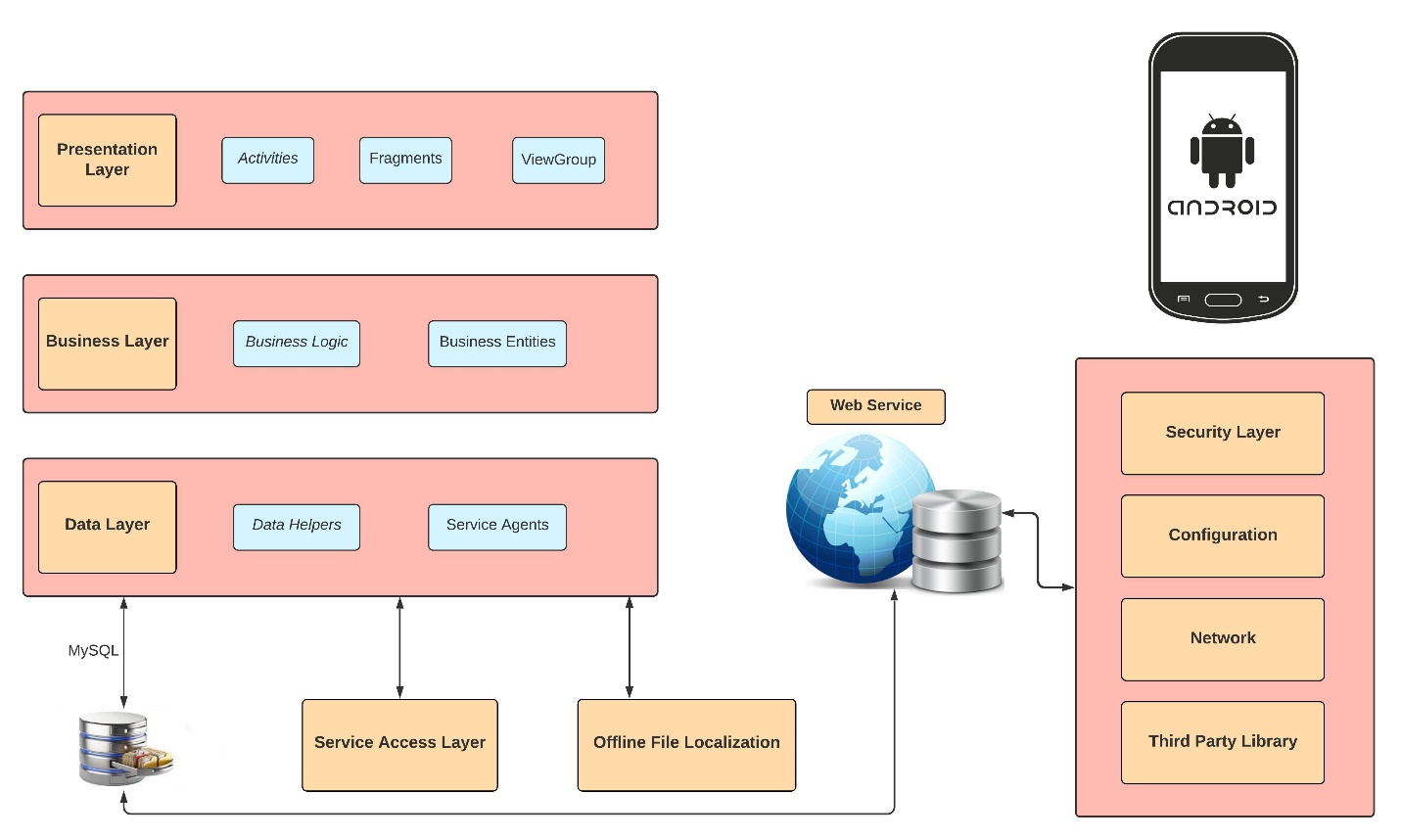


Figure 1: System architecture of GameSAT

The abstract representation of our system architecture starts with the presentation layer. This is the layer which will be presented to the user. The activities include the play mode or the game mode which the user can partake on. While Fragments handles possible options, the user can choose in play mode. For example, the user can choose to play a word game or a passage game. Similarly, in training mode the user can choose to train with words only, or with passages. The view group separates the different views which the user will be presented with. The view given to the user in game mode is different from the view given in train mode.

The business layer deals with the logic behind our app. The business entities are the various class objects which will be illustrated in the class diagrams. While the business logic contains the logic behind the interaction of the objects, and their operation.

The data layer enables our system with the ability to store data. Data helpers interact with the service access layer and determine if the user info should be stored or not. The offline file localization handles the data for various questions or missed questions.

The security layer in the phone app is enforced by the interface presented to the user. Also, the user will be required to register for an account. The configuration feature handles the possible options that the user can choose. For example, to play or to train, and within these options one can further choose words or passages.

The network feature is needed in order to have a live update of the 10 best scores, so that the user is aware of an accurate global score. Third party libraries will enable us to add features in an efficient way when we commence implementation.

1. **UML diagrams**
   1. **Use Case diagrams**

The first possible scenario the user of our app will experience is that of logging-in or creating a new profile. The use case diagram for this situation looks as follows,

Chart

Description automatically generated

Figure 2: Use Case for logging-in.

If the user is already registered, then they will access the **Log-in** functionality. On the other hand, if the user is new, the New User extension point activates the **Create Profile** functionality, which prompts the user to create a new profile.

The user upon logging in can decide to play a game, in which case the diagram capturing this scenario will look as follows,

Diagram, schematic

Description automatically generated

Figure 3: Use case for Game Mode.

In **Game Mode** the user has access to a **Help** functionality which provides the user with the rules of the game. The user can also **Skip** questions they find difficult. Moreover, a **Timer** exists for each question given to the user. The user will also be able to track their score through the **Game Score**. The **Level** which the user is in, will also be available. In addition to this, the user can see where their score ranks by observing the **Global Score**. In **Game Mode** the user can choose to play a **WordGame** or a **PassageGame**.

If the user selects to train instead of play, then the use case diagram will look as follows,

Diagram

Description automatically generated

Figure 4: Use Case for Train Mode.

In **Train Mode** the user can access the **Help** functionality which will provide the user with information regarding this aspect of the app. In addition, the user can choose to train word vocabulary by selecting **WordTrain**, or passage comprehension by selecting **PassageTrain**.

* 1. **Class diagrams**

The general class diagram looks as follows,

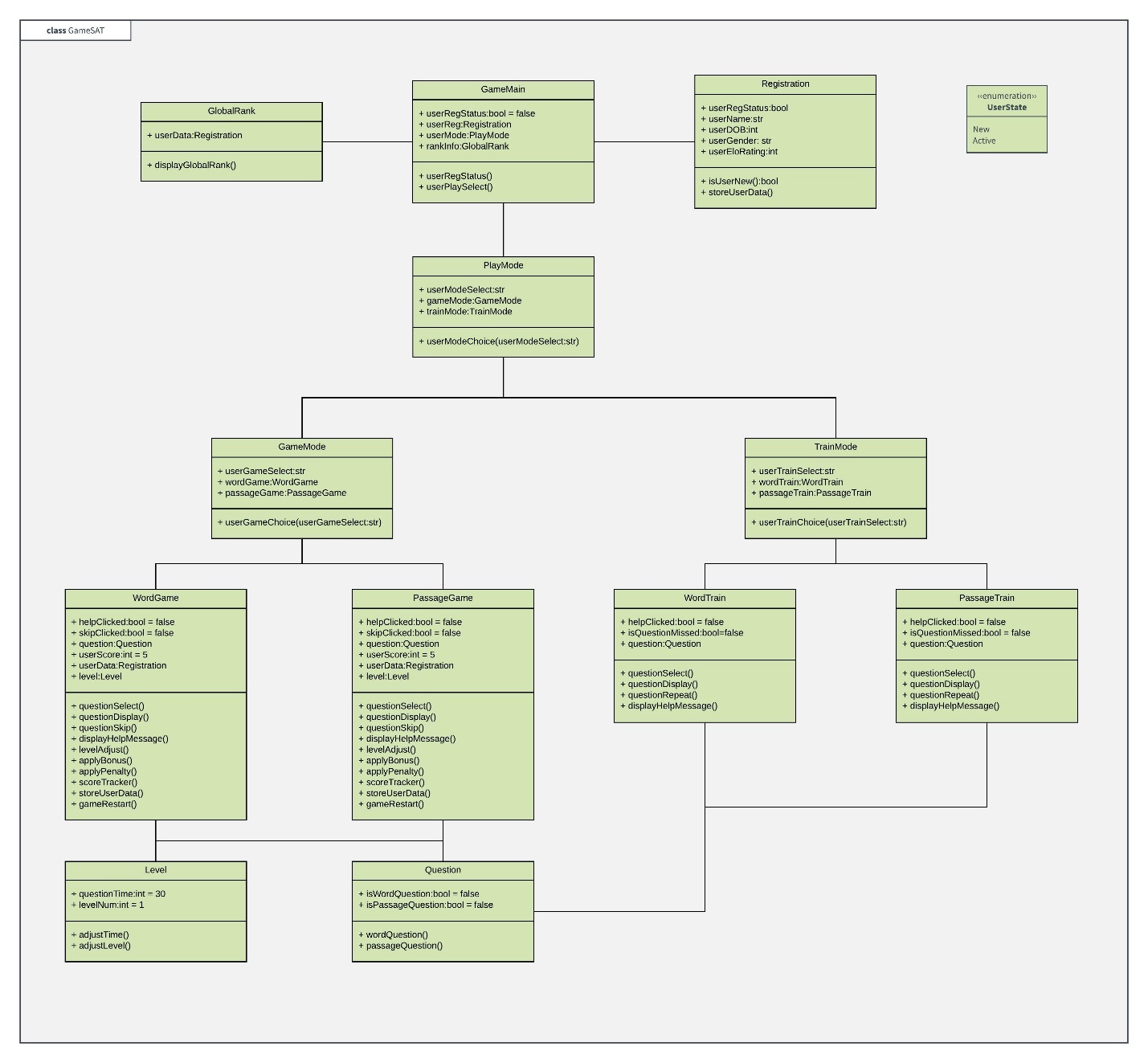


Figure 5: Class diagram for GameSAT.

All the objects are connected through association, so they know about each other’s presence. The user will communicate with the GameMain object, and the registration of the user’s data will look as follows,

A picture containing diagram

Description automatically generatedFigure 6: GameMain object.

GameMain presents the user with a display and acquires the user’s information. If the user is new, then Registration stores the user’s data. The user can also observe the GlobalRank where the top 10 scores are recorded.

The user can further provide GameMain with the choice on whether to play or train. This case is illustrated below,

Diagram

Description automatically generated

Figure 7: Play Mode options.

The userModeChoice in play mode takes the user’s choice and determines whether to go in Game Mode or in Train Mode. If the Game Mode option is selected, we will have the following,

Diagram

Description automatically generated

Figure 8: Game Mode options.

In Game Mode the function userGameChoice applies the user’s preference to play word games or passage games. If the user selects WordGame, then questions determined by the Question object will be presented to the user. The Level object will keep track of the levels, and the time each level allows to answer a question. The WordGame object will also keep track of the score, penalties, and bonuses of the user. The same features exist if the user selected to play PassageGame, with the only difference being that the Question object would present the user with questions involving passages instead of singular words.

If the user decided to train instead of play, we would have the following diagram,

Diagram

Description automatically generated

Figure 9: Train Mode options.

In train mode the user is informed through the userTrainChoice function that they can train with words or with passages. If the user selects to train with words, then WordTrain will inform the Question object of this and provide the user with word questions. On the other hand, PassageTrain will provide the user, through its interaction with the Question object, with passage questions.

* 1. **Sequence diagrams**

The sequence diagram that illustrates the logging in of the user, will look as follows,

Diagram

Description automatically generated

Figure 10: Sequence diagram for logging in.

The user informs the GameMain object through the userRegStatus if they are new or active users. If the user is new, then the function isUserNew informs the Registration of this fact, and the user’s data is stored using storeUserData.

On the other hand, if the user decides to play a game the sequence diagram will look as follows,

Diagram

Description automatically generated

Figure 11: Sequence diagram for play mode.

The user makes their choice available to GameMain, and GameMain informs PlayMode of the user’s choice using userModeChoice. Afterwards, GameMain informs GameMode on the user’s preference of which game they like to play. If the user selects to play WordGame, then questions particular to this game will be selected and displayed. On the other hand, if the user decides to play a passage game, then PassageGame will select and display questions particular to this game.

The sequence diagram for the train mode,

Diagram

Description automatically generated

Figure 12: Sequence diagram for train mode.

is very similar to that of the GameMode. PlayMode is informed of user’s choice through userModeChoice, and afterwards GameMain informs TrainMode using userTrainChoice on the user’s preference for word training or passage training. If the user decides to train on words, then WordTrain will select and display word questions. If the user wants to train with passages, then PassageTrain will select and display passage questions.

1. **Low fidelity prototype**

The initial view the user will be presented with, will look to some extent like the following figure,

Graphical user interface, application

Description automatically generated

Figure 13: Initial view of our app.

The user will enter their data and afterwards will have the option to play the game or train, as shown below,

Application

Description automatically generated with low confidence

Figure 14: Game modes of our app.

If the user selects to play, then the options presented to them will look as follows,

Graphical user interface

Description automatically generated

Figure 15: Game Mode option display.

In this particular case the user is playing a word game, and the available options to quit the game, or skip the present question are immediately available. Also, the user will be aware of the level they are in, and how much time each question allows for an answer. The score of the user will also be displayed along with a help option.

If the user decides to train instead, then they will be presented with the following view,

Graphical user interface

Description automatically generated

Figure 16: Train Mode display.

In this particular case the user is training on word vocabulary, and the time and level options, along with the skip options are unavailable. The user can quit the game at any time and can ask for help if they need it.

1. **Implementation plan**

The programming language we will be using is Java. In addition to this, we intend on employing the functionality of android studio 4.1 or some other compatible version for our implementation needs. The android operating system we intend on using is the latest version, Android 12, but maybe other versions will be used also. If time permits, we will test various other android operating systems in order to produce a bug free and efficient app. The implementation of the code will mostly occur in Windows and maybe Linux. The testing that we intend on doing is black-box testing, and we will use Junit for most of our testing needs, which is provided by java. The database needs will be handled using MySQL. The project management will occur through Jira. In addition to this, we already have a GitHub repository which we are presently using to share our work.