System Design Document for Qwalk

Version 2.0

2017-05-28

Chalmers Tekniska Högskola

Elina Olsson, Jonathan Kraft, Pia Westerbergh, Kevin Solovjov

This version overrides all previous versions.

1. Introduction

This document describes the overall structure and design of the code in the mobile application Qwalk and its subsystems. The goal is to design a system with low coupling that is easy to maintain. It should also be as loosely dependent on android as possible in order to make it possible to change platform easily.

1.1 Definitions, acronyms and abbreviation

- **Qwalk** - is the actual application but also another name for a quiz walk
- **Separation of concern principle** - A design principle: separate the code so that each part of the code only have one concern.
- **Open-closed principle** - A design principle which makes a program open for extension, but closed for modification.
- **Dependency inversion principle** - A design principle: depend on abstractions, not implementations.
- **MVP** - Model, View, Presenter is a design pattern commonly used to divide parts of the system into well-defined sections, while also following the separation of concern principle.
- **Java** - Multi-purpose programming language.
- **PHP** - Server-based programming language.
- **XML** - Programming language for graphical user interface.
- **Activity** - An Android specific class which contains logic for controlling the content of a graphical view.

2. System architecture

The main part of the application will run on an android device, but it also communicates with a database on a server. In this way, multiple users with different devices can send information to each other via the database. The application only requires one mobile device and one database, but it is possible to extend to any number of mobile devices (in theory).

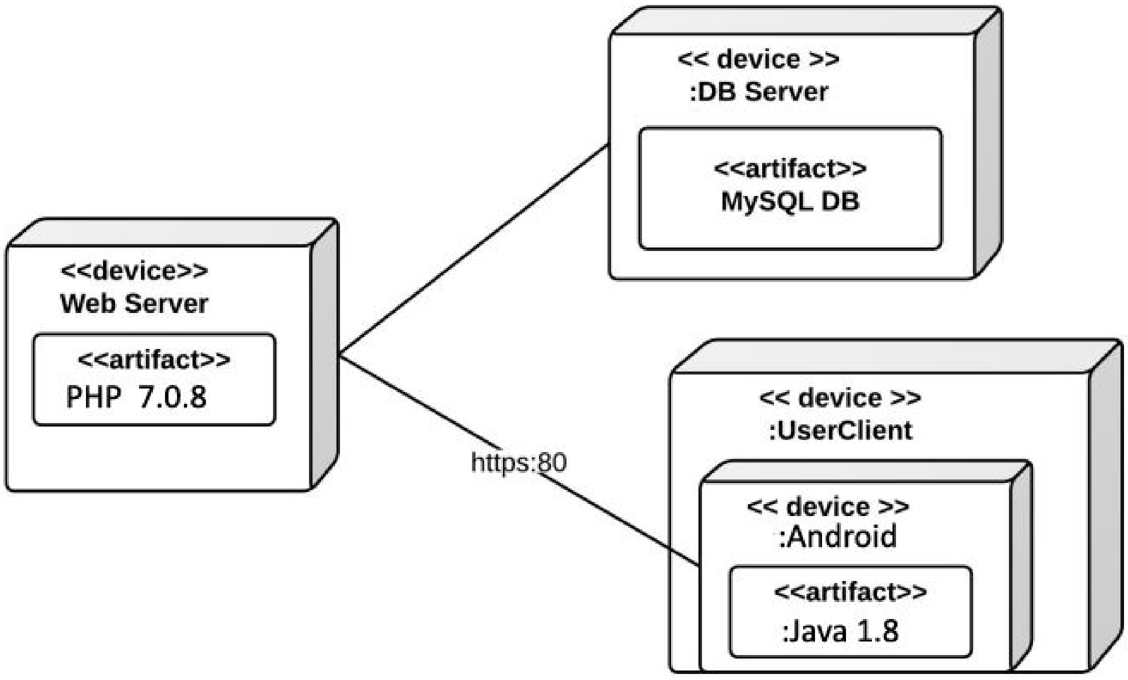
The android application is written mostly in Java 1.8 and XML 1.0 fifth edition, while the server-based part of the system is coded in PHP 7.0.8 and MySQL through InnoDB on MariaDB 10.1. Java is responsible for handling the Qwalk logic, data and user input while PHP is only used in communicating with the database.

Of all the design principles concerning coding, *Qwalk* aims to fulfill the following three in particular: s*eparation of concern*, *open/closed principle* and *dependency inversion* principle. One of the measures taken in order to achieve this goal is to implement the MVP pattern. By doing this, the components of the application get as few dependencies as possible, given the circumstances. See more at 3.4

2.1 Setup instructions

This section is a meant to aid in the deployment of the application *Qwalk* by providing instructions and explaining the inner workings of the program. The application has two main parts, the main program, written in Java, and the online database, made with MySQL and PHP.

2.1.1 Deployment Diagram



2.1.2 Java

The application requires an android device or equivalent emulator with android version 6.0 or higher. The application runs on Java 1.8 and uses XML 1.0 fifth edition as well as Gradle Version 3.3. If testing by compiling the code directly, a

compatible integrated development environment (IDE) must be chosen.
Android Studio is recommended for compiling and running the program in this case.

2.1.3 PHP

The PHP files can be found in the main directory of the project, in the folder titled "Webb-Server". These PHP files must be hosted on a PHP 7.0.8 compatible web-server that is accessible by the application through a URL. The application stores the URLs to every file in the "DataBasehandler" class, there the URLs are stored as strings and need to be changed if the host is moved. By default they are set to a version hosted on "www.000webhost.com", but can be changed to another location if desired, provided the files are hosted correctly.
The file "connectDB.php" contains the details needed to log into the database, including host URL, username, database and password. If the host is moved, make sure that these are updated as well with the new login credentials.

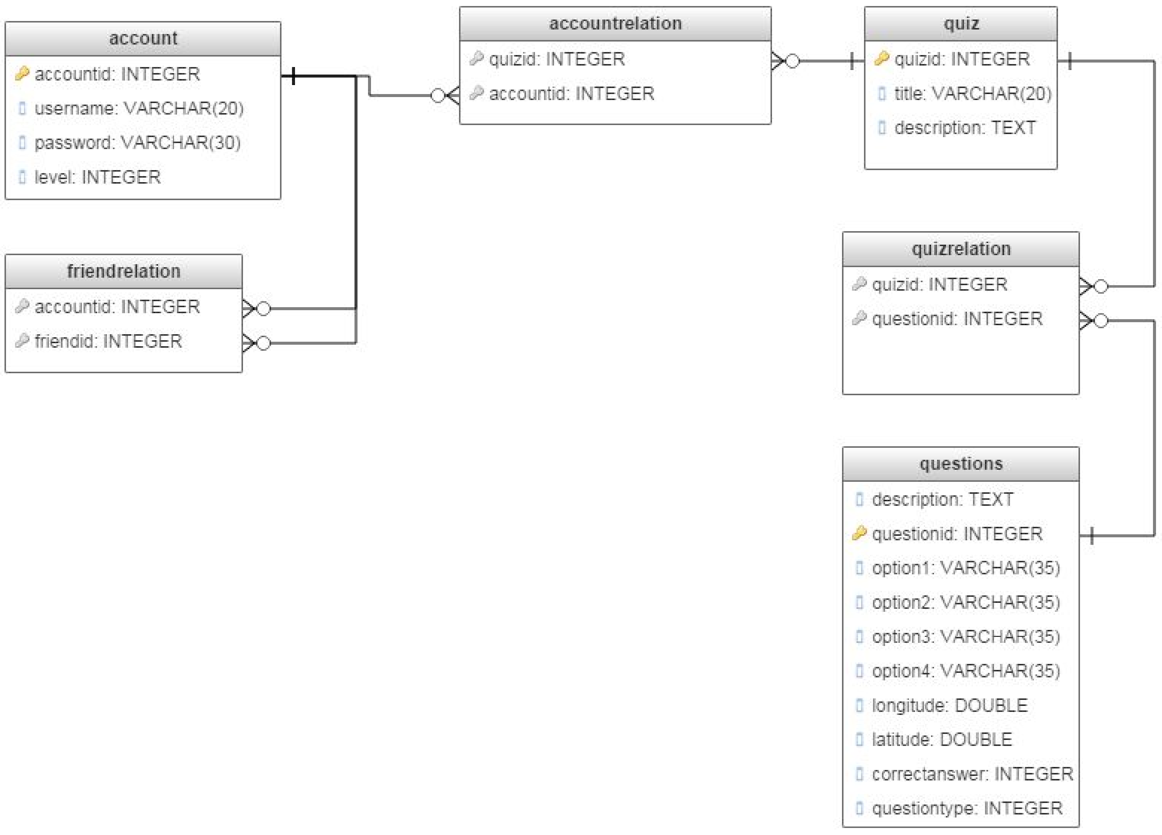
2.1.4 SQL

The database must be hosted as such that it can be accessed by the file "connectDB.php". If the database is hosted in the same place as as the PHP files, the url in "connectDB.php" may use the URL "localhost", otherwise an address must be given.

A file containing the structure of the database can be found in the folder "Webb-Server" in the main directory of the project. Most databases have an import function where this file can be uploaded to recreate the required structure. If this is unsatisfactory, there is a file in the same folder titled "structure.pdf" that describes every table in the database and the fields that it contains.

See the System Design Document (SDD) for an explanation of table relations.

2.1.5 Database Structure



3. Subsystem decomposition

3.1 Google Maps API

Google Maps API is a library handling a lot of functionality surrounding mobile map applications. More specifically it contains code about interactive maps, markers on the map, showing buildings in 3D, street view and user location to name a few. When developers are using this API, the interface they are presented with is only available in code. However, it suffices as an acceptable interface because it’s methods are clear and well-defined. Example of this is OnUserLocationChanged(Location) and checkLocationPermission().

Generally, the quality of the packages Qwalk is implementing from Google Maps API is very well developed, since it has a *range* of functionality but still few bugs.

The Google Maps Services package is a very large one with around one hundred different subpackages. The ones (currently) in use by Qwalk are some data implementations in it's *Model* subpackage, as well as packages containing code for interactive maps, camera updates, the map itself, location services and the Google Api Client package.

3.2 PHP & MySQL

PHP (Acronym for PHP: Hypertext Preprocessor) is a server-side scripting language most commonly used for the development of dynamic websites but is also used for general-purpose programming.

PHP is often used alongside MySQL, an open-source relational database management system. MySQL can be coupled with an user-friendly interface called "PHPMyAdmin" to give the user a simple way of creating databases, tables and structuring information so that it can later be interacted with through PHP. MySQL together with PHP form components of an "AMP" stack. AMP is an acronym for "Apache, MySQL, Perl/PHP/Python" and describes a complete system for the development of web-applications.

PHP has built in support for MySQL which allows it to easily communicate with the database to read, write, update and delete data. This makes it a fitting tool to use to save our data online by sending information from Java to PHP and letting the server handle the information without our application really knowing what is happening "behind the curtains".

We use an online host to act as our web-server and upload our PHP code there. We use the same service to also host our MySQL database. This allows us to easily access the stored data and is useful for testing on multiple devices as opposed to a local apache-server.

Our PHP code is fairly straight-forward and for the most part is only there to act as a bridge between our application and the online database. The functions found in the various files usually receive some form of information from the application, be it a simple number like "0" or "3" to determine an action, or a set of fields needed to create a quiz or question, and then return a string with information back that the application acts upon. In this way, the PHP code functions much like a method in Java, in that it receives a parameter, performs a simple task and then returns an answer.

We could have expanded the functionality of the back-end but preferred not to as we decided that it would be easier to keep most of the code locally in Java, especially when the goal is to be as object-oriented as possible.

3.3 Android

The basis for android applications are so called “Activities”. These are view classes that provide what the user sees on various parts of the application. Every new window/screen/menu etc. is a new activity. In the MVP model, these activities are very “dumbed down” and mostly just forward all user input to a presenter, which is “smarter”. The presenter has access to the model which contains all data and methods of calculation and contains it’s own special methods to handle user input before sending issuing commands back to the view for the user to see.

Another quirk of android programming is how you change between activities. Android uses something called “intents” which can be described as a set of instructions for an an action. These are, among other things, used to launch new activities, and can carry data with them for sending information to a new activity. An example of this in Qwalk is when a question is created in createOptionQuestionActivity and needs to be sent to CreateQuizActivity in order to build a quiz.

In the Qwalk application, activities and intents are just two examples of android functionality. Other used functionality include colour, showing *toast* messages, graphical user interface parts and some listeners for these parts.

3.4 Qwalk

3.4.1 Overall structure

The application is composed of a number of activities - where every activity has a corresponding presenter class and an XML-document containing the layout and design of the user interface. The presenter, in turn, handles the communication between the activity and application model classes. An illustration of this, as some pictures from a STAN analysis, is portrayed in the figures below.

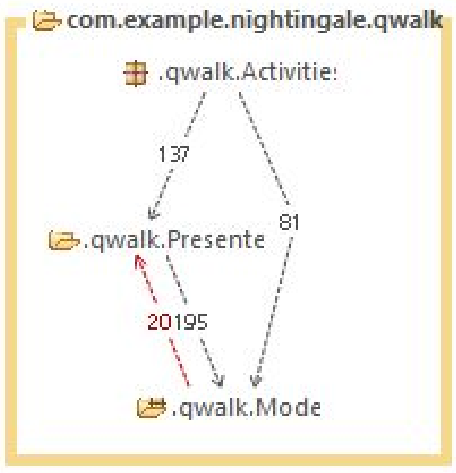


Figure 1 - Composition of the different packages

The graph pictured above shows the main components of the application, excluding the database and the server. The model contains classes for location, qwalks/quizzes, questions, communicating with the database, functionality for the bot and the player and the class QwalkGame.

QwalkGame is responsible for the flow of the application when the user is playing a qwalk. It is responsible for the twenty references to the presenter folder since it needs a lot of communicating between the map, user interface and itself.

Except QwalkGame, focus has been put on referencing as few as possible out-of-package classes, and making it not dependent on android. However, this goal has not fully been met since the location class contains functionality on converting different types of location classes into one another. Another part of why this goal was not met is because some classes implement *parcelable*, an android interface.

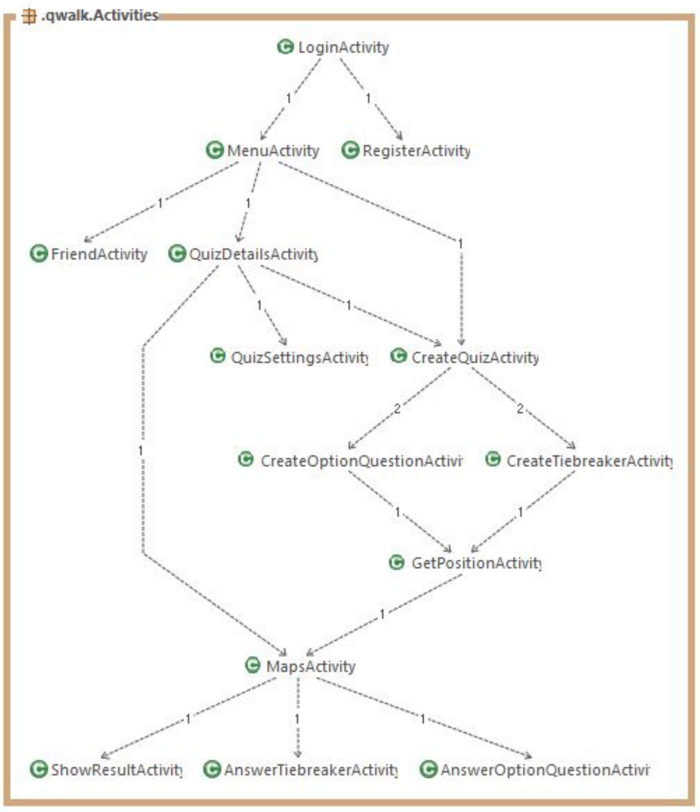


Figure 2 - Activity composition

Every activity has a class reference to the activities that it can open. Figure 2 also shows how the user might navigate through the application. The activities are responsible for almost all android implementation and user input, as well as communicating with the user interface.

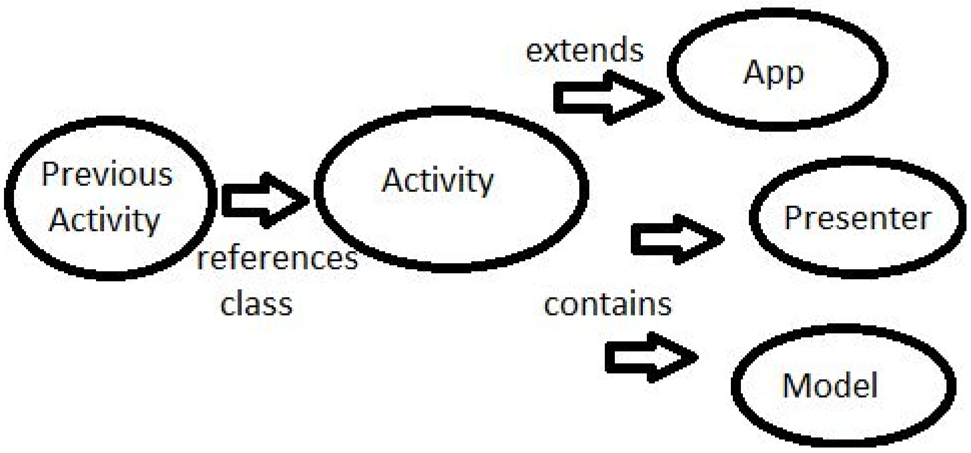


Figure 3 - An example activity

Almost all activities: are referenced by another one in order to initialize it, and contains model classes or references to them. Every activity has a corresponding presenter. The activities should only handle the android parts of the program and communicate user input to the presenters, as well as communicating from the presenter to the graphical interface. This is fulfilled in most of the classes.

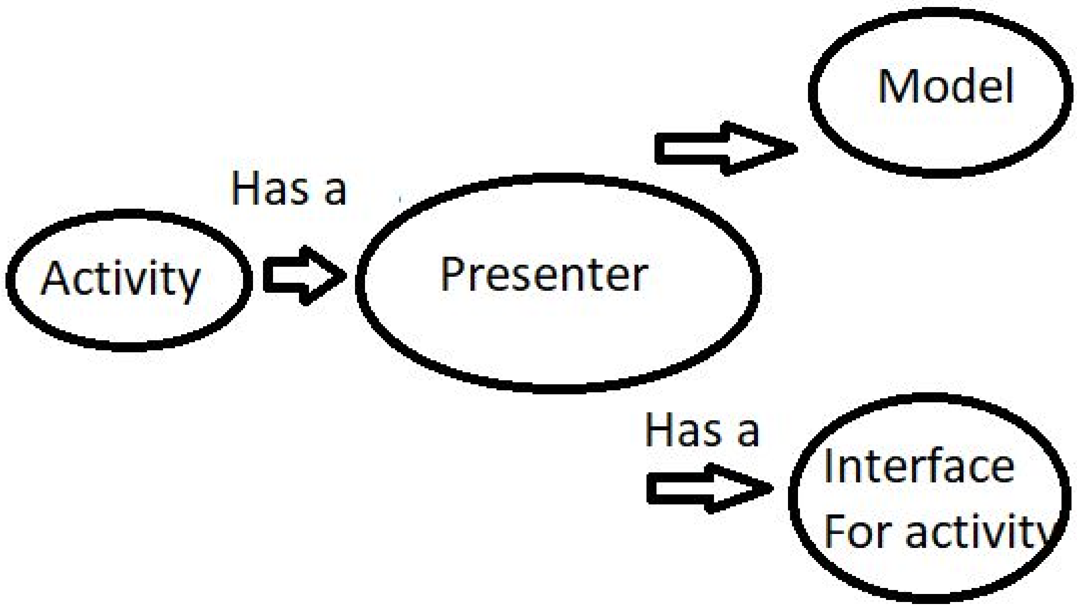


Figure 4 - An example presenter

The presenters have a layer of abstraction between them and their activities, so that we may change platform but still keep the presenter. No presenter references another. The presenters are smart, and contains the logic used to interpret user input and affect the graphical interface through its interface. It also uses and contains model classes.

3.4.2 Model classes

Qwalk contains a number of small and large model classes.

- QwalkGame

The class responsible for the flow when a user is playing a qwalk. Keeps track on what to do depending on different settings. Has the current quiz, the player, the bot, the game timer and the mapPresenter. Actually breaks the MVP pattern a little bit, an abstraction should be introduced between the presenter and this class.

- Actor

An interface for Ai and player. Tells them to have a location, and to save their answers.

- Ai

Contains the ai's location, answer and difficulty. Is also runnable, qwalkgame starts a thread with the ai to update its position on the map.

- Player

Contains the players answers and location.

- QLocation

A location class. Stores latitude and longitude

- Question

An abstract class with shared functionality for tiebreaker and optionquestion. Stores question title, which answer is right, a location and an ID in order to find it easily on the database.

- Tiebreaker

A type of question.

- OptionQuestion

A type of question.

- GameTimer

A timer class.

- Quiz

Contains a title and description of the qwalk/quiz. Also contains a list of questions, what difficulty is set and which settings are on.

- QuizSetting

An enum for qwalk settings.

- QuizDifficulty

An enum for qwalk difficulty.

- StandardQuizzes

A non-initializable class for storing the default quizzes/Qwalks

- DatabaseHandler

A static class for communicating with the database.

- Account

A user account class.

- MessageMediator

A class for communicating from the databasehandler to different presenters and activities.

- QwalkMarker

An android model class, for storing a marker on the map associated with a question and if it is in range or not.

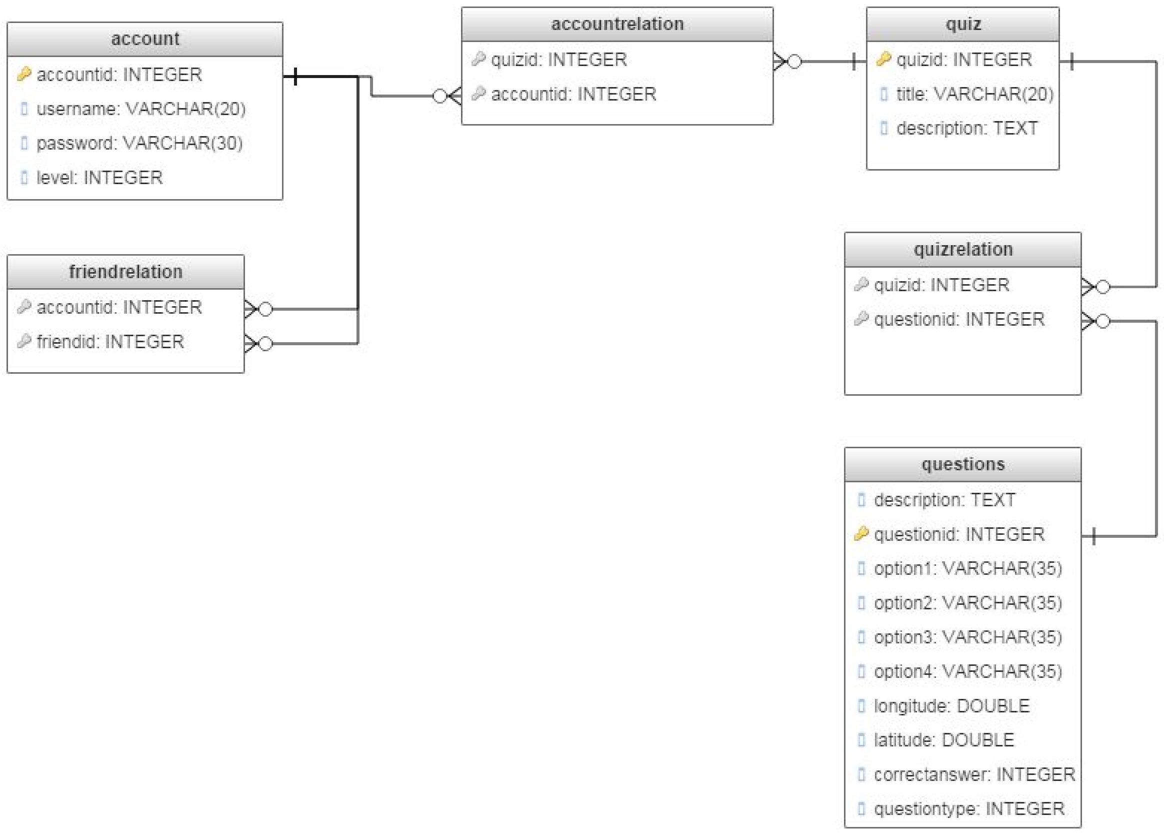
3.4.3 Code quality

The code has been analyzed using *PMD*, which found around six hundred errors in the code. After going through every error, the group reduced this number to about two hundred - where over one third consists of variables overshadowing existing variables. This mostly occurs in the constructors of the Qwalk classes. Another big group of errors are the *magic numbers*, which are numbers directly written in the code. Fixing mentioned errors was not prioritized.

Some JUnit tests have been used during the development. These can be found in the repository, in the test folder.

4. Persistent data management

Account information and quiz/question data is stored on a MySQL database that is hosted on https://www.000webhost.com, which is a free web hosting service. There are three separate tables (a series of columns and rows) for questions, accounts and quizzes and these are then connected through so called relation-tables. These relation-tables instead contain two columns with ID’s for different data to save which data is related to what, for example which questions belong to which quiz.



The data is saved by sending a JSON string with information from the Java application to a PHP file hosted online that then communicates with the database and passes the variables it received from the application.

To read the data, the application similarly sends a request to a PHP file that gets the information from the MySQL database and sends a JSON string as a response back to the android application, which can then be decoded and read.

5. Access control and security

At the moment, every user has as much access to the application as any other - if they have made an account. If the user is logged in as a guest, they can only play qwalks and not create, edit or delete a qwalk or add friends.

The server has functionality supporting an admin user, but there are no implementations of this in the application.