System design document for Qwalk

Version: 1.1
Date: 2017-05-08
Author: Elina Olsson, Jonathan Kraft, 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.

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 - A specific class which contains logic for controlling the content of different graphical views.

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.

In more detail, the application is composed of a number of activities - where every activity has a 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, not following any industry standards, is portrayed in fig.1 below.

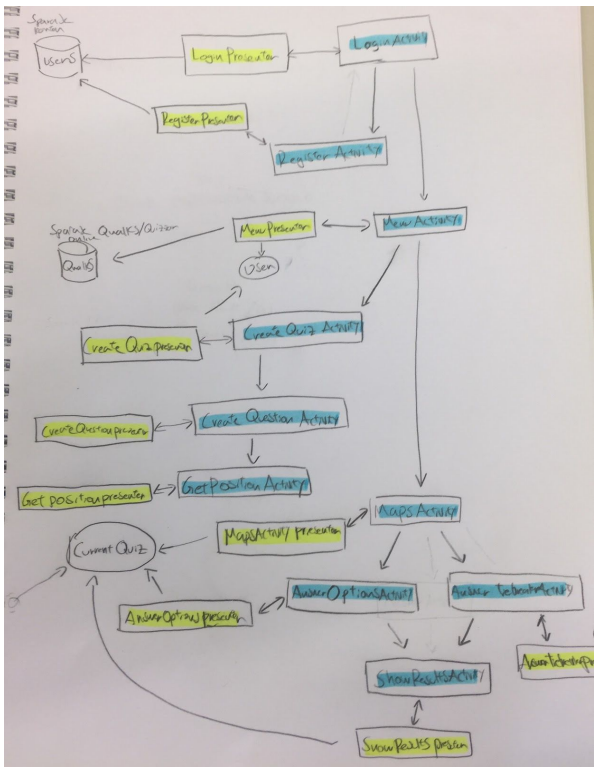

Fig. 1 - An illustration of all dependencies. Blue is an activity, yellow is a presenter, white is a model.

Some parts of the project contains references to even more software - such as MapsActivity and GetPositionActivity where Google Maps API is used.

An UML deployment diagram, possibly drawings and other explanations.

2.1 General observations

Denna rubrik fanns inte i mallen men Joachim hade skrivit med den i sitt exempel. Sjukt

3. Subsystem decomposition

For each identified software above (that we have implemented), describe it …

3.1 "…First software to describe" …

Recap: What is this doing (more detailed). Divide it into top level subsystems. An UML package diagram for the top level. Describe responsibilities for each package (subsystem). Describe interface. Describe the flow of some use case inside this software. Try to identify abstraction layers. Dependency analysis Concurrency issues.

If a standalone application

- Here you describe how MVC is implemented
- Here you describe your design model (which should be in one package and build on the domain model)
- A class diagram for the design model.

else

- MVC and domain model described at System Architecture

Diagrams

- Dependencies (STAN or similar)
- UML sequence diagrams for flow.

Quality

- List of tests (or description where to find the test)
- Quality tool reports, like PMD (known issues listed here)

NOTE: Each Java, XML, etc. file should have a header comment: Author, responsibility, used by.., uses …

3.2 "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.

3.3 "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".

4. Persistent data management

How does the application store data (handle resources, icons, images, audio, ...). When? How? URLs, paths, ... data formats... naming.

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

Different roles using the application (admin, user, ...)? How is this handled?

6. References