System design document for KwizGeeQ

Version: 1.0

Date: 28 may, 2017

Authors: Are Ehnberg, Anton Kimfors, Henrik Håkansson, Marcus Lindvärn

1. Introduction

KwizGeeQ (pronounced quiz-geek) is a quiz-app mainly intended to help students with learning or studying subjects of their choice. However, it can be used by anyone needing help to remember things. The user can create new quizzes, then answer the questions in a random sequence. The application also saves statistics globally as well as for individual quizzes.

1.1 Design goals

This document describes the construction of the KwizGeek application.
Some design goals:

- The design must be loosely coupled meaning we should have as few cross-package-dependencies as possible .
- The design must be testable i.e. it should be possible to isolate parts (modules,classes) for test.

1.2 Definitions, acronyms and abbreviations

Technical definitions:

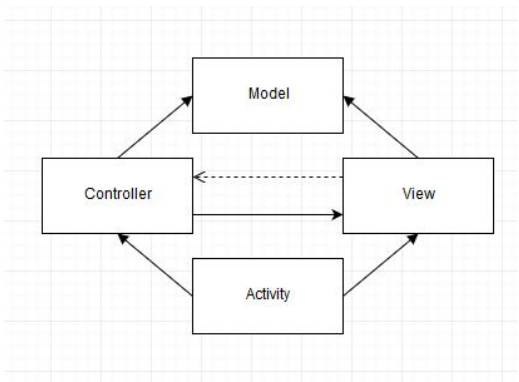
- API - Application Programming Interface. In our case Spotify's API, since it lets us use functionality from their systems.
- GUI - Graphical User Interface
- MVC, a way to partition an application with a GUI into distinct parts avoiding a mixture of GUI-code, application code and data spread all over.

Definitions and terms regarding the KwizGeeK application:

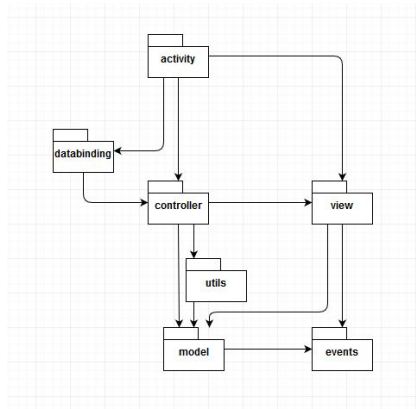
- UserQuiz - a user created Quiz with questions and answers created by the user
- SpotifyQuiz - a autogenerated quiz that tests the user in music related questions, partly tailored per user.

2 System Architecture

The application follows MVC structure, with some use of event buses. The application runs locally on a single android device. The application is decomposed into the following top level packages:



Basic MVC structure of a general Android application



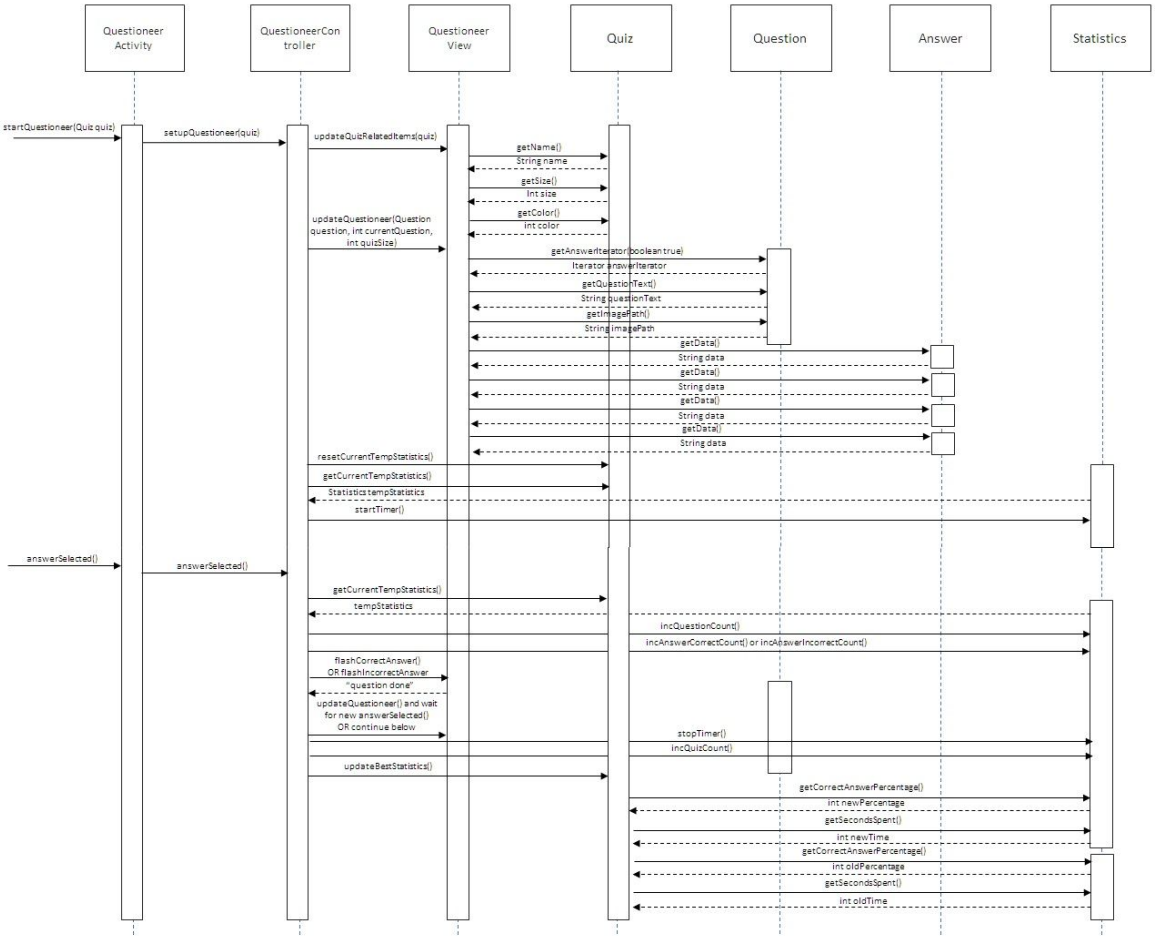
Structure of KwizGeeQ

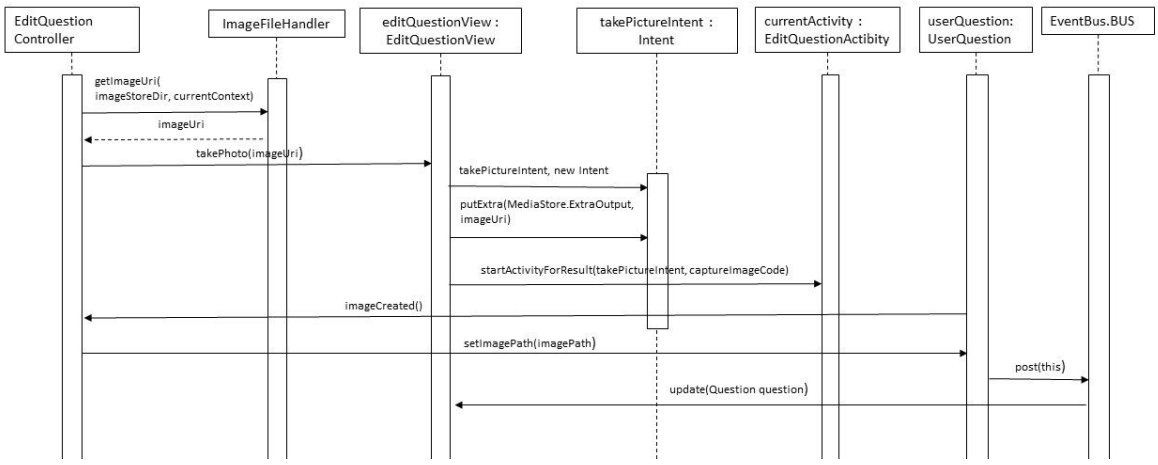
Package functionality:

- *Activity* - holds the android related activities, this is where the rest of the MVC is instantiated and bindings are created.
- *Databinding* - autogenerated classes, these transmit “on click”-events automatically from the activity to the controller.
- *Model* - holds the model for the application, which contains all the actual logic regarding saving, modifying, and getting data.
- *StorageUtilities* - holds classes related to storing data in different ways..
- *View* - stores all the classes that handle the views and their elements.
- *Controller* - holds the controller for each activity, this is where listeners are created, and holds methods controlling each activity.

Sequence Diagrams

UML sequence diagrams for 2 use cases from RAD: play a quiz and take photo for a question (originally Add media in RAD, but this part is the only one that is implemented)

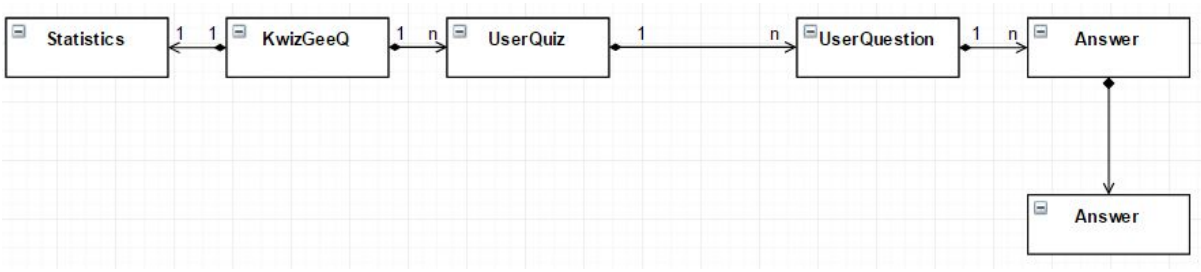




3. Subsystem decomposition

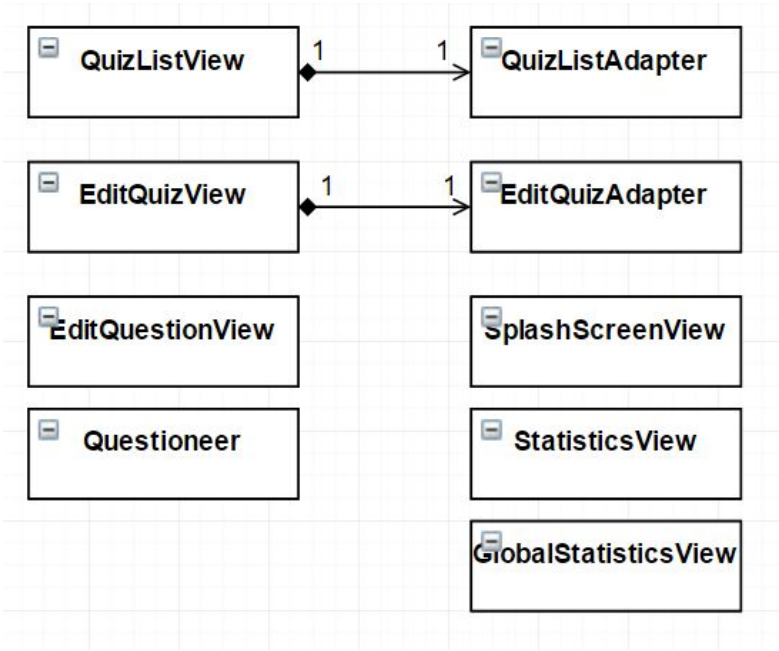
3.1 Model

The model package contains the design model and all logic related to the application.



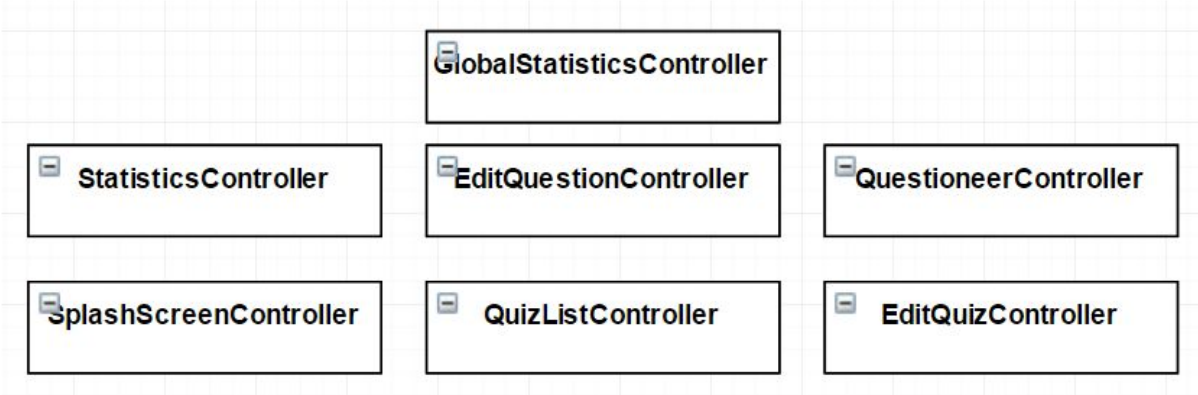
3.2 View

The view package is responsible for graphically visualising the model for the user. It is called by the Events package when there is a change in model that requires the view to be repainted, or from the controller. The view package also handles the actual logic for switching to and from new activites since this was considered to be part of the graphical interface.



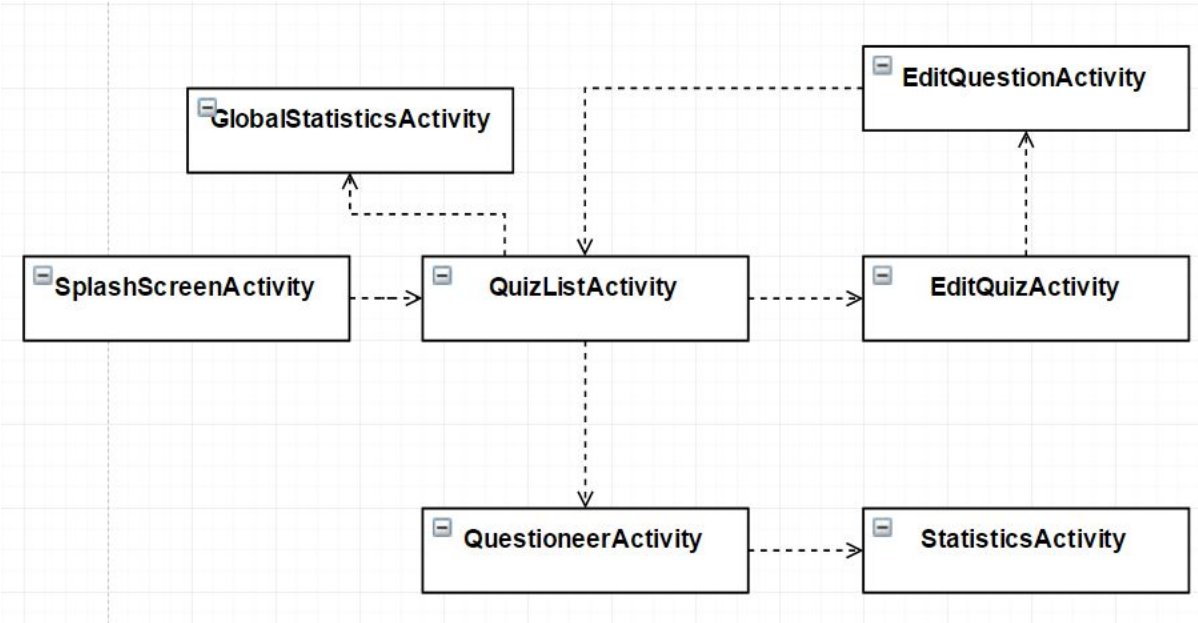
3.3 Controller

The controller package handles input from user and decides what should happen in the model or view based on this input.



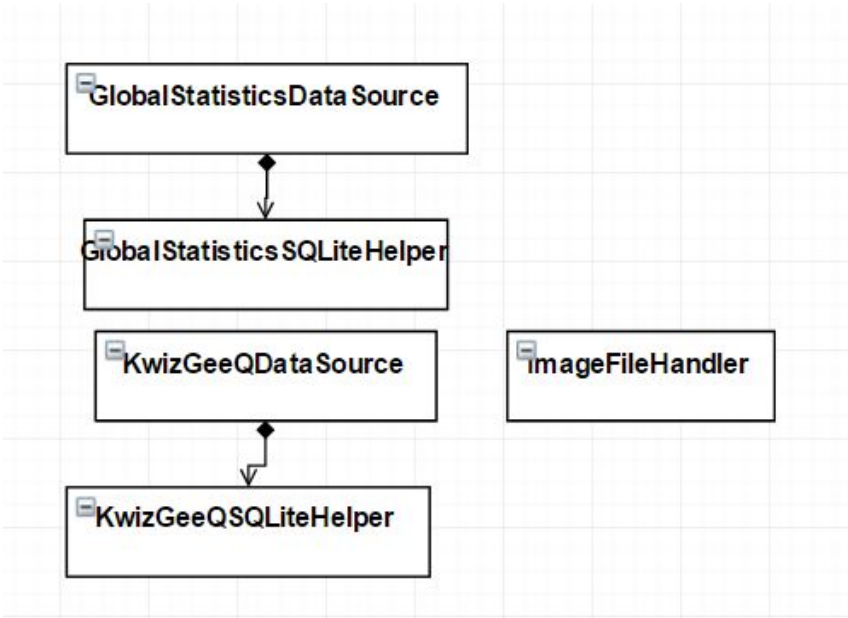
3.4 Activity

The activity package contains classes that in some way extend Android's Activity, which is needed for the application to run on an Android device. The package also constructs classes in controller and view package that is needed for the specific Activity.



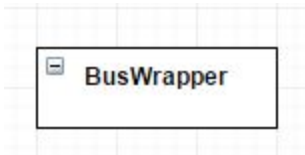
3.5 StorageUtilities

The StorageUtilities package handles storing data. Most of the classes deal with storing the data locally through SQL when the application is not running. The only class not doing this is classes the ImageFileHandler which handles creating URIs for images.



3.6 Events

The events package handles the events that originate in the Model, and notifies the View that these changes have occurred.



4. Persistent data management

The storageutilities package is handling all the data persistence with help of two different SQLite databases. Two classes, KwizGeeQDataSource and KwizGeeQSQLiteHelper, handles all the quizzes and together with ImageFileHandler they are able to save all images and URI:s to these. GlobalStatisticsDataSource and GlobalStatisticsSQLiteHelper handles the saving of global statistics.