

CHALMERS TEKNISKA HÖGSKOLA
LSP310 Kommunikation och ingenjörskompetens

29MAY, 2016

System Design Document for IDGI (SDD)

Authors:

Tove Ekman, Jonathan Krän, Emil Logren, Tuyen Ngo, Allex Nordgren

Table of contents

1	Introduction	1
1.1	Design goals	1
1.2	Definitions, acronyms and abbreviations	1
2	System Design	1
2.1	Overview	1
2.1.1	Persistent data handling	1
2.1.2	Event handling	2
2.1.3	Internal representation of text	2
2.1.4	Lack of presenter class	2
2.1.5	Lists	2
2.2	Software decomposition	2
2.2.1	Unique IDs (keys)	2
2.2.2	General	2
2.2.3	Decomposition into subsystems	3
2.2.4	Layering	3
2.2.5	Dependency analysis	3
2.3	Concurrency issues	4
2.4	Persistent data management	4
2.5	Access control and security	4
2.6	Boundary conditions	4

1 Introduction

The application is written in Java for the Android platform using Android Studio as development environment. Running the application requires either a phone with the Android operating system or an Android phone emulator.

1.1 Design goals

The design must be divided into clearly defined packages with specialized purposes. These packages must not display any circular dependencies. Furthermore, the coding style has to be uniform throughout the packages.

1.2 Definitions, acronyms and abbreviations

- Activity: An android-specific class representing a single, focused thing that the user can do or interact with. Activities serve as both views and presenters in the application. An Activity is somewhat similar to JFrame in Swing.
- Fragment: A part of an activity that can be swapped depending on the situation. Fragments simplify the usage of modular components such as tabs.
- Event bus: An object that handles communication between objects to reduce dependencies.
- Firebase: A service for simple data storage on a remote server.
- RecyclerView: A View component that shows a list of items.
- Hat: Hats are rewards to the user for reaching certain amounts of points.

2 System Design

2.1 Overview

The application's main controlling classes are the activity classes which act both as view and presenter.

Decoupled communication between packages is accomplished by using the EventBus library.

The main point of entry is the StartActivity class. Alternatively, the class CreateLessonActivity will act as the entry activity if the application is started by sharing a video to IDGI through the YouTube application.

2.1.1 Persistent data handling

The application relies on Firebase for storing persistent data, which stores data as JSON. Firebase was chosen mainly due to its simplicity.

Handling data is done in one of two ways, depending on whether or not an Internet connection is available. When Internet access is available, the persistent data is loaded from the remote Firebase server. However, should Internet access not be available, predefined mock data (schools, subjects, etc.) will be loaded instead. Moreover, loading of mock data can also be chosen by specifying FirebaseMode in Config.java.

2.1.2 Event handling

Events are processed through the EventBus class. A global instance of this class is gotten from the ApplicationBus class. Posting an event is done through ApplicationBus.post(), and listening to events is done by annotating a method with @Subscribe. This method must have only one parameter, which determines which events the method will listen to.

2.1.3 Internal representation of text

Each text string in the application should be localized. This is done through the file strings.xml, and the text is then retrieved through the android R class.

2.1.4 Lack of presenter class

As of now, the Activity classes' responsibilities include those of both a view and a presenter. Given more time, a better approach would be to divide these areas of responsibility between two different classes.

2.1.5 Lists

The lists in the application are created through the usage of RecyclerViews. A RecyclerView makes use of two major components: One is the ViewHolder, which as the name suggests holds the view, or the look, of the list-item. The other one is the Adapter. Adapters provide views to the RecyclerView, and these views represent items in a set of data.


and an adapter, which function is to update the viewHolders with relevant information from a collection.

2.2 Software decomposition

2.2.1 Unique IDs (keys)

School, Account and Quiz objects all have unique keys associated with them. The School and Account keys are generated by the Firebase push() method, and is used to ensure that data is not overwritten if two users were to write to the database simultaneously. The Quiz key is used as a Map key to store how many points a user has earned for the quiz.

2.2.2 General

The application is divided into the following structure: 

The android package is somewhat of a wrapper package, holding every other package related to the android framework.

- activity: Holds all the Activity classes as well as classes for dialog windows used by the activities.
- core: This is where the OO-model resides.
- event: Contains classes related to communication between other classes (EventBus).
- fragment: Contains fragments that are used by activities.
- recycleView: This package handles the RecyclerViews (Views that list things) in the application.
- service: Home of the database management.
- session: Any temporary information, like what lesson is currently being viewed, is stored here.
- widget: Contains subclasses of standard android views.

2.2.3 Decomposition into subsystems

The only service is the database.

```
public interface IDatabase {  
    IQuiz getQuiz(String key);  
    void retrieveSchools();  
    List<Subject> getSubjects(School school);  
    List<Lesson> getLessons(Course course);  
    List<Course> getCourses(Subject subject);  
    List<Comment> getComments(Lesson lesson);  
    void retrieveUsers();  
}
```

2.2.4 Layering

N/A

2.2.5 Dependency analysis

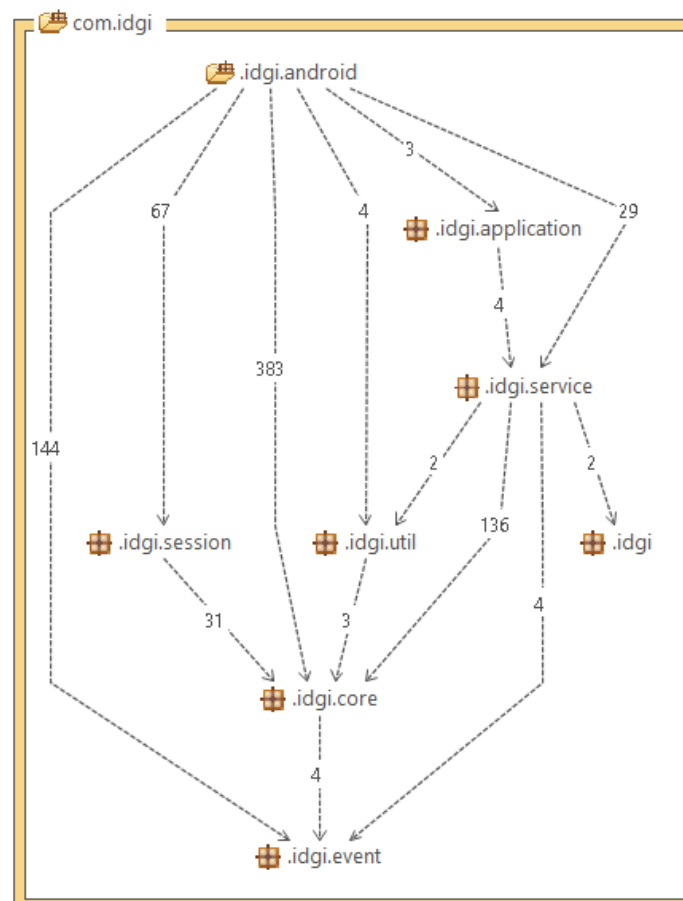


Fig. 1: Dependency analysis done in STAN

2.3 Concurrency issues

The choice of Firebase as the database implementation results in that cocurrency issues are not a problem. The framework includes ways to ensure that no data is mistakenly overwritten or otherwise lost. One of which is the push() method, that creates a unique key for an object as it is being written to the database. This way, even if two users were to simultaneously write say a School object with the same name, they would still both be saved correctly.

2.4 Persistent data management

Data is stored remotely on a database provided by Firebase. The persistent data consists of:

- Schools
- Accounts
- Hat templates, i.e. point requirement and image

All information about the objects are stored on the remote database. For a School object, that includes subjects, courses, etc.. all the way down to separate quizzes for the lessons.

2.5 Access control and security

The application is completely devoid of any security measures.

2.6 Boundary conditions

N/A