Project 1 : CoPilot drive statistics

Design Description

December 1, 2016

Contents

1 Background

This project revolves around Volvo CoPilot software. The aim is to develop an Android application for their CoPilot hardware which will be able to measure parameters such as speed, direction or fuel to establish an eco-driving score. They want the application to be a game for the user and for all the results to be published and ranked on a web portal.

In this document, we will go over the high level design of the system that we are going to develop in accordance to Volvos requirements.

2 High-Level Description

The system to be developed is divided into two parts, it is an android application and a web application. It is going to be focused on eco driving. This means that it is going to help the operator of the machine with driving more efficiently. The idea is to make it into some kind of a game where drivers can compete against each other. The users will compete by trying to have the best score possible and this is calculated by (weight * distance) / (consumption * RPM * acceleration). These are the parameters that will be mentioned in the part below together with altitude.

The driver will start up the application, choose what kind of vehicle that is being used, identify him- or herself with an alias (which is later being swapped out for some kind of login, not required to be done in this project) and then choose how long the drive will be. When this is done the driver will get into the game screen of the application which supplies real time feedback on how efficient the drive is in the form of graphs and a high score. The graphs will be presented as line graphs which the driver can toggle between to get a specific view of how each parameter is affected. There will also be a radar graph that shows the correlation between all parameters, this graphs is what is going to be the main focus in the game screen. But while driving you do not want the drivers to get their eyes off of the path, so the feedback that is going to be most relevant while performing the actual drive is a change in color of the radar graph and a big presentation of the highscore. The game screen will also contain a timer on how much time the driver have left and a stop button that can terminate the drive if needed. While using the application there will also be an option to swipe the screen to access a list of the drivers past scores. All of these list items will contain the score for that drive and a saved version of all the graphs that was present while driving, that is all the line graphs and the radar graph, but they will only show for the list item that has been clicked.

The web portal is a one page application that will present users with information on the top 20 high scores that have been submitted. Similar to the score screen in the android application the items in this list will be clickable and expand to show the relevant graphs for that particular drive. The information that is being presented on the web portal is read from the same database that the android application uploads to.

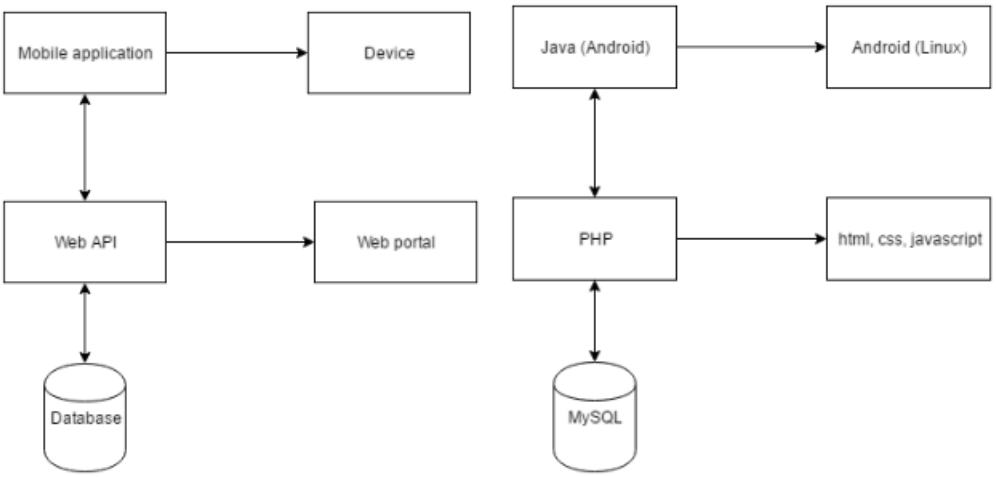
As mentioned in the introduction the android application will be run on a customized android tablet made for Volvo called the Co-pilot. This tablet can read data from sensors placed on the construction equipment via CAN busses and through custom libraries show this information to the driver.

3 Software Architecture

3.1 General Architecture

The overall system architecture can be described as a client-server structure. To be more precise, in this case we have fat client, where the android application do all the data processing and calculations, and thin server, where the Web API is responsible just for persisting data in centralized database.

The left image below represents an overview of how our system is structured and how different parts communicate with each other. On the right side we have the same diagram with different labels, the purpose of this diagram is to describe technologies used for parts of the system.



3.2 Android application

Every android application is created by combining component blocks. In android, there are four different types building components. The Volvo application is composed of two types, **Activities** and **Services**. An activity represents single screen with GUI. Every activity is an entity on its own and application can consist of one or more different activities. In order to be able to better understand design decision, we need to know how an activitys lifecycle looks like. The picture below gives us a detailed overview of the above mentioned process.

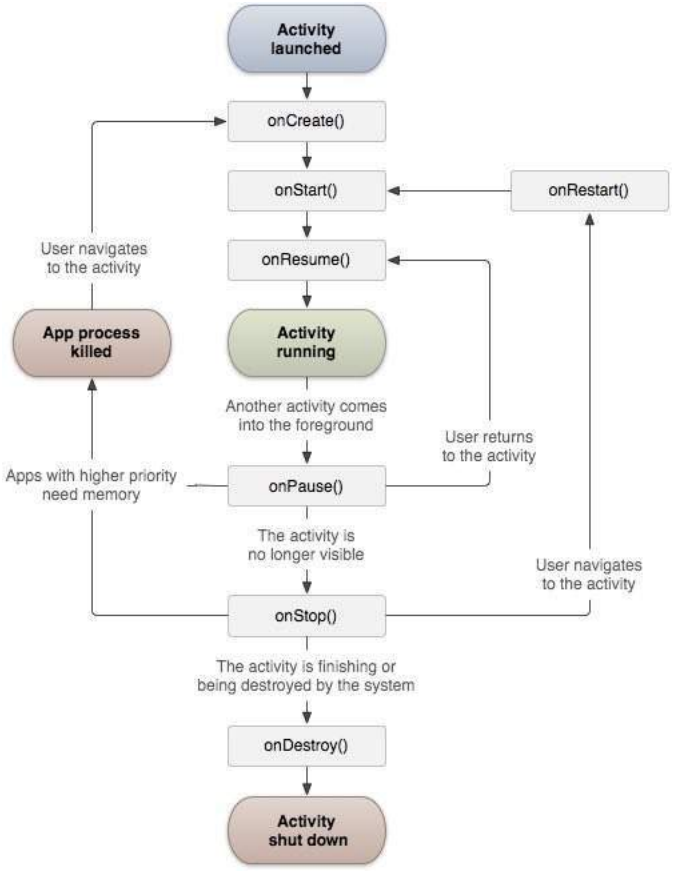


Figure 1: Image source: https://*www.tutorialspoint.com/android/android_acitivities.htm*

Its not visible from the image above but at one time only one activity can be active. In other words, when an activity is launched, the one previously active is **stopped (onStop())** and pushed to the activity stack, known as the back stack. The **back stack** is implemented in a LIFO (last in, first out) way, so when

we close current active activity (for example, press back button), the activity is popped out from the back stack, **destroy it (onDestroy())** and the previous activity **resumes (onResume())**.

nameOfActivity = CHANGE THIS IN THE NEXT VERSION TO MATCH THE CODE IMPLEMENTATION

All the application logic and user interface is divided into five different activities. They are:

1. Selecting the machine type - **nameOfActivity**
2. Enter alias - **nameOfActivity**
3. Choose time span - **nameOfActivity**
4. Main game view - **nameOfActivity**
5. Stats - **nameOfActivity**

Communication with the Web API is done through following methods:

1. create-new-game
2. end-session
3. register-event

(base url + method name)

Besides of the main functionality of the application to calculate high score and push it to the centralized database, there is also a requirement to provide an option to use the application even when the internet connection in not available. This task will be accomplished by implementing local storage, populating it with data, in case there is no internet connection and syncing it with the main database when the internet becomes available.

For persisting data that are stored locally inside the Volvo application we made a decision to use **SQLite**. Sql has multiple advantages over using other types of storage. By using SQLite, we can assure data wont get corrupted, also, its easy to perform a calculation on data using SQL.

3.3 Web portal

The web portal is designed in a way that all the communication with back end is done through the async calls (**AJAX**). The application requests are handled by **PHP** scripts that are located on the server. Furthermore, all the communication with database and calculations are done inside PHP scripts.

4 Concerns

Security may be a concern since Volvo is a large company that always strives to protect their technology, however we do not handle any critical information in this project and no major effort will be put into safety. One could argue that personal integrity should be prioritized between users but since we only use aliases this is more of a concern for the future of this product where user specific logins could be made.

Access to our application will not be any concern since its installed and used locally on the co-pilot tablet. The tablet also has internet connection so it can reach the web-portal if it is needed.

A concern that already has risen in our project, that may turn into a major one, is the time it takes for the co-pilot to operate. If the co-pilot is slow then getting a stable data stream may be difficult and without a stable data stream the accuracy of the results cant be verified. It is also less appealing for the user to use a slow semi-working product, but our focus will be for our application to affect the system in a rather small way. This will be done by creating an easy navigable GUI that doesnt require a lot of power to operate. This concern however is out of our reach to handle since receiving the correct and persistent data is depending on the sensors and the co-pilot hardware. The only thing we need to be concerned about is using and storing the data in a correct way, not how it is retrieved.

5 Detailed Software Design

5.1 Android App

The Android application consists of three general sections: the game initialization, the game screen and the statistics/highscore list.

Game initialization

This section contains three Android Activities:

1. Selecting machine type
   - The user is presented with this activity if no previous selection has be done
   - If the Android SharedPreferences has a prefered selection stored the user if moved directly to Activity 2
   - If no previous selection has be done the user can select a machine type and the selection is stored in SharedPreferences
2. Selecting alias
   - The user is presented with text field and encouraged to enter an alias
   - After click the next button an Intent with the chosen alias is sent to open Activity 3
3. Selecting time interval to play game
   - This activity received the alias from Activity 2 and lets the user pick a time interval to play the game
   - On click next an intent containing both the alias and time interval is sent to open the game screen Activity

The user is presented with Activity 1 when starting the App and proceeds through Activity 2 and 3 until the user reaches the game screen Activity.

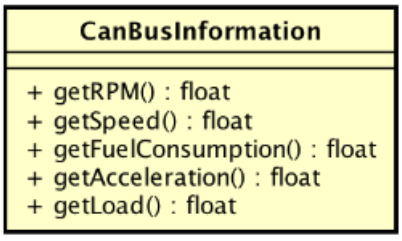
Game Screen Activity

The game screen user interface contains the following elements:

1. A radial graph showing live information from the vehicle (RPM, speed, distance, etc.)
2. A line graph showing the history of the live values presented in the 1
3. A set of buttons which toggle the line graph between the categories in the radial graph (RPM, speed, distance, etc.)
4. A stop button which ends the game before the time interval has ended
5. A progress bar which indicates how much time remains

Then there are the following underlying functions and classes which control the UI:

A. A class providing public functions which read the latest input from the machine CAN bus and provide them in the correct format for the app to use

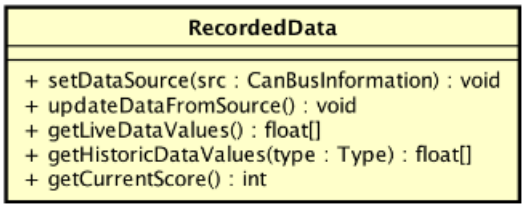


- Class CanBusInformation

  - Public getRPM()
  - Public getSpeed()
  - Public getFuelConsumption()
  - Public getLoad()
  - Public getAcceleration()

B. A class providing methods to store results in a statistics list (see statistics/highscore section)

C. A class which stores all recorded information from the can buss, calculates scores and provides live feedback via notification

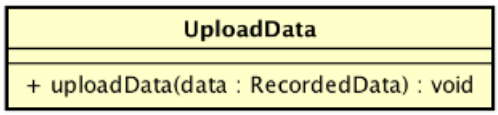


- Class RecordedData
  - Public setDataSource(CanBusInformation src)
  - Public updateDataFromSource()
  - Public float[] getLiveDataValues()
  - Public float[] getHistoricDataValues(Type type)
  - Public int getCurrentScore()

D. Functions which connect the UI to the data

E. A timer which ends the game when the time runs out

F. A class which takes data from RecordedData and uploads it to the web portal



- Class UploadData
  - Public uploadData(RecordedData data)

Statistics/highscore list

The statistics/highscore list contains a list with the most recent local game results. Each list item is expandable and shows information like alias, score, fuel consumption, etc.

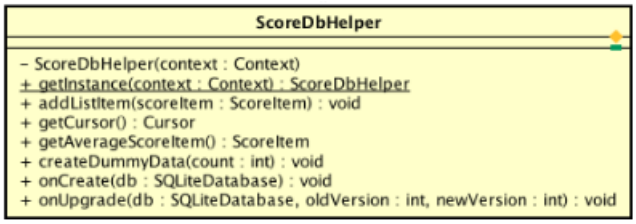
The activity read its content from a local SQLite database which contains one table collecting the following information:

- ID
- Alias
- Score
- Average fuel consumption
- Average acceleration
- Average speed

- Average load
- Average RPM
- Date

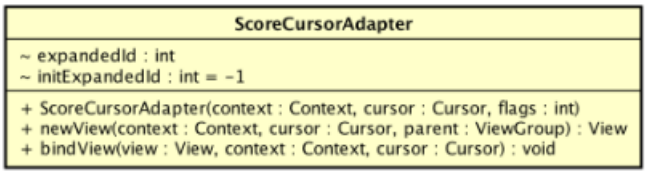
Two classes help to handle the database:

- ScoreDbHelper



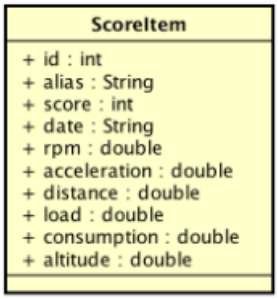
  - Class ScoreDbHelper extends SQLiteOpenHelper
    - public void addListItem(ScoreItem scoreItem)
    - public Cursor getCursor()
    - public ScoreItem getAverageScoreItem()
    - public void onCreate(SQLiteDatabase db)
    - public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)

- ScoreCursorAdapter



  - class ScoreCursorAdapter extends CursorAdapter
    - public ScoreCursorAdapter(Context context, Cursor cursor, int flags)
    - public View newView(Context context, Cursor cursor, ViewGroup parent)
    - public void bindView(View view, final Context context, Cursor cursor)

- ScoreItem



  - class ScoreItem
    - public ScoreItem(int id, String alias, int score, String date, double rpm, double acceleration, double distance, double load, double consumption, double altitude)
    - public String toString()

ScoreDbHelper creates the SQLite database, provides a cursor to access its data and provides methods to insert new data. ScoreCursorAdapter creates the ListVies in the activity and fills it with appropriate data from the database (bindView fill it with data, newView create a list item and its user interface). ScoreItem is used for data storage of database rows outside the database.

5.2 Web Portal

The web portal consists of a single page which lists the 20 highest game scores. Each list item is expandable and provides access to a radial graph and a line graph similar to the one in the game screen of the app.

The web portal also provides the option to filter results based on machine type in the left sidebar.

The website uses a set of JavaScript controllers to get and update information on the site:

- HighScoreController
- LineChartController
- RadialChartController

These controllers use PHP-APIs to connect to and communicate with the database server:

- Connection
- Create-new-game
- Query
- Register-event
- End-session

These APIs are also used by the Android App to upload new information to the server

The database server is MySQL and contains the following tables (see figure):

- ValueTime
- Session
- Alias

6 Graphical User Interface

Project consists of both the android application that executes on Co Pilot hardware and web portal which presents a list of top 20 high scores retrieved from the database.

First, we will focus on defining graphical user interface for android application as it provides main part of our project.

6.1 Android application

Android application has different activities which correspond to different steps that user must/should follow in order to use the application:

- First activity is selecting machine where user can swipe and choose machine on which he is operating at his job. This activity will be executed once and subsequent use of the machine wont require this activity to be processed. However, there is possibility to change the machine and overwrite saved settings.
- Second activity provides field for inserting your alias/name. This can also be perceived as a first activity when you are using the application after already setting the machine type previous time or by another user, and you dont want to change it.
- Following activity, after entering your name/alias is setting the period for which you are going to play, or use the main functionality of the application. At this point you have two options, to start/confirm the entered time or to return to previous activity Entering alias.
- After confirming you are moving to next activity which presents a main activity of game that includes information and graphs. This activity provides different user interaction features. You have a button stop. By clicking it you are interrupting the period you set previously and you are asked (by dialog), if you are going to confirm or reject the score. Your score is uploaded to database in both scenarios. The difference is that if you approve than you are going to be flagged so that your result may show on web portal if it is in top 20 high scores. If you reject your high score will be used only for calculation purposes. In the center of the application is radial graph where different parameters are shown. By clicking on any of these individually, the line graph, in the upper area, will change depending on selected parameter, and the selected parameter at radial graph will become highlighted. At the left corner, you can see the remaining time of selected period you had entered before.
- At main game screen, you have possibility to swipe and change the activity to historic data screen. At this screen, you can see different scores, made on that local machine, and all parameters that affected the score, individually.

The Graphical User Interface for android application is driven from ideas of the team to be both classy and professional in order to provide desirable effect, be easy to use and in that way to be accepted by the end users.

6.2 Web Portal

Web portal provides the ability to see the list of top 20 high scores and all individual data that affects the final score, and also graphs that represents the data in time and show more understandable image to the end-user.

At the web portal, interaction with the user includes:

- Changing machine on the left side of the web portal that influence the data (high score list), that will be shown. Default value is All, that shows high score list for all machine types. Beside these, user can choose other types such as excavator, wheel loader etc., individually. Also, on the left side of the application there is a client logo, at the top left corner.
- User select the score from the list. After that, the score will enlarge, focus and show more details about the score that includes all data that affects the score and graphs representing these data. At this point, user can click buttons in order to choose which data to show in line graph at the right side (FUEL, load etc.). Beside line graph there is a radial graph at the left side that shows all data and how they interact and affect the final score and in bottom right corner there is list of some parameters that presents sum values during the observed time.

The Graphical User Interface for web portal is driven from ideas of the team to be both classy and professional in order to provide desirable effect, be easy to use and in that way to be accepted by the end users.