|  |  |
| --- | --- |
|  |  |

Technical Product Description for Plate Recognizer

# About This Document

## Purpose

This document provides a high level technical solution description on Ericsson Parking Plate Recognizer system.

## Target Group

The intended target groups of this document are the following:

* Installation engineer
* System administrator
* Maintenance
* OpenCV and/or mobile developers

## Assumptions

It is assumed that users of this document are familiar with Android Studio and Java.

# Product Overview

* Key functionality

Plate Recognizer is an app designed to help the Ericsson parking staff travel the most effective route for recording parking while recording the data to be used in various ways. Parking data has an enormous potential to be used in IOT for its multiple usages in finding empty parking spaces, revolutionizing paid parking, car accountability, etc.

* Plate Recognizer is an Android app which uses:
* Couchbase with it synchronizing databases with NoSQL format.
* EasyPR which is the Chinese version of license plate recognizing software OpenCV.
* Android APK using Java
* Capacity
  + Currently capped at 100MB on the couchbase side. Can be increased and decreased(100MB is the lower limit) on demand.
  + Dependent on the mobile phone being used to host the app.

# System Overview

## System Architecture

Figure 1 shows the high level system architecture of the app-database relationship.

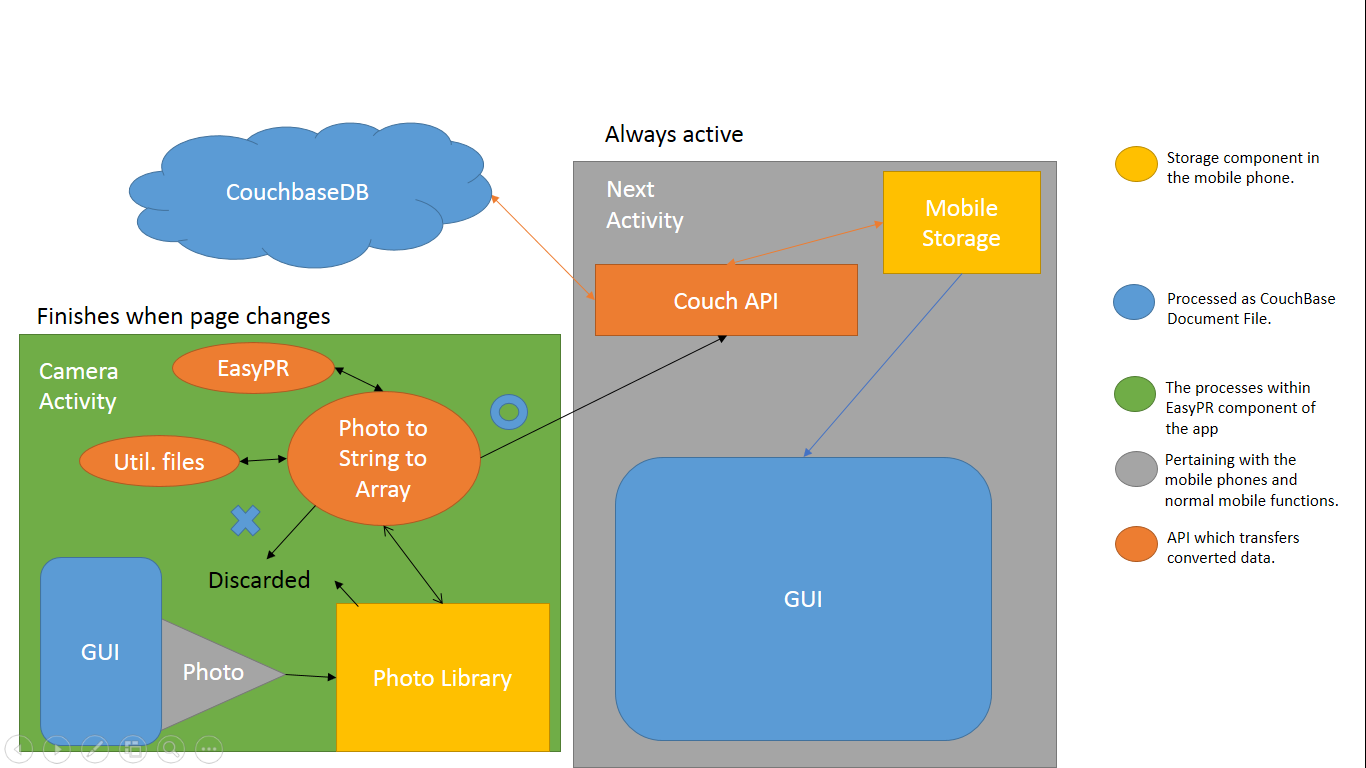


Figure High Level Architecture of app-database relationship

The Plate Recognizer application makes use of a cloud server, local mobile storage as well as the phone photo library to store information. The data is first starts out as a jpeg file which is broken down to bytes which are then used by the EasyPR dependency to obtain a string which represents the plate number of the car taken. This string is then used in an array containing the current instance of camera with information on which parking is relative to and passed to the background Next Activity instance. This is then placed in the storage as a ‘document’ folder which is a JSON format file which contains information relative to the Couchbase dependency for synchronization purposes.

When connected to the cloud Couchbase server, the API will self-sufficiently call the push and pull replications through liveQuery.

The application would have an intense single thread in the Camera Activity with simple threads being called alongside it when buttons are pressed. The background thread of Next Activity will also be always active. However, when the user switches to the Next Activity, the intensive Camera Activity switches off.

## Main Activity Interfaces

### Data structure

Outlining the data structures used in the application.

|  |  |
| --- | --- |
| Interface Parameter | Parameter Description |
| Port | Port used for connecting to the server: 4895 |
| Database stream | Name of the database stream to connect to the cloud database: Parking1 |
| ‘Forward’ variable | A Boolean in the Camera Activity which is true from point A to point B and is false when the user wants to go from point B to point A. |
| Relevant data | Key thing to note is that the parking spots are separated into the key routes and sub-key pid. This is to create an optimal path from A to B going from route 1 to route 6. Route 1 could have pid 1 to 100 for example. If the forward Boolean is false, it starts from route 6 100 to 1 and accordingly, the route decreases from 6,5,4,3,2 to 1.  "lastCheck": "2016-11-28T16:24" -(String) The time the last car was last checked.  "parkSpot": "154" – (String)The specific parking spot number  "pid": 14 –(Int) The pid relative to the route number  "plateNum": "N92342" – (String)  "route": 6 –(Int) The route number  "state": 0 – (Int) The state of the car. 0 is unoccupied(grey),1 is occupied(green) and 2 is occupied(red). |
| View | Views are the sets in the stored database. There are 3 views in the app:   * sortByPid: sort the database with the keys being route and pid. * sortByParkingSpot: sort by parking spot number. |
| Queries | Queries are called to obtain specific sets of data from the views. There is a liveQuery which runs by itself to keep the main database updated when running with the cloud. It just queries sortByParkingSpot.  *CurrentCam* method runs sortByParkingSpot to obtain the current parking spot and fill the spot with the new license plate number and/or change the state number.  The sortByPid number is run in the *getNextSpot* method to obtain the next parkingspot, route and pid. If forward is true, pid and route naturally increases but it decreases when forward is false. |
| Document Template | The Document is a JSON document with a couchbase template format. An example is as such:  {"\_sync": {  "rev": "4-571c4078fb62a673525b01ff5f3170b3",  "sequence": 254,  "recent\_sequences": [254],  "history": {  "revs": ["1-be29b670d9c3825788537829bb889c4d"],  "parents": [-1],  "channels": [null]},  "time\_saved": " 2016-11-28T16:24:59.516Z"},  "lastCheck": "2016-11-28T16:24","parkSpot": "154","pid": 14,"plateNum": "N92342","route": 6,"state": 0} |

Table 1 Interface Parameter Description

### Camera Activity

* On app start-up (*OnCreate*)

Introduces the main buttons such as the left, right, capture as well as the title bar and time bar. Following the set-up, the interface starts the check to ensure the camera preview comes from the back camera or whether there is any camera at all using *getCameraInfo*(). The camera preview method starts. The intent introduced in binding to the cur\*\*\*(curPid,curRoute and curTime) variables are intents carried on from a previous Next Activity instance that ran. Background variable of ‘forward’ representing the route from A to B is true and is false for a route for B to A.

* Interface interactions

The forward button on the bottom left changes the ‘forward’ variable from true to false and vice versa. On click on the *capturePhoto* button, it first checks the camera to ensure that the camera used is the one facing the back and then runs the method ‘*takePicture*’ to save the picture into the photo library.

There are two callbacks running in the *takePicture* method. The first *previewCallback* takes the preview photo and converts the area around the plate box into a bitmap for and determines whether it can be read. If there is an error in reading, the method ends. However, when it succeeds it uses the EasyPR dependency to determine what the license plate number is and it is placed in the plate variable. An alert window will pop up and ask the user if they want to use the license plate for the respective parking. If yes, then the string gets passed to Next Activity interfacing using a method, *currentCam*. The picture is also cropped before being sent to the next call back which sends the photo to the library.

The picture sent to the photo library is assigned a name and placed by the fileUtil.java methods. These methods run through the native android photo library/camera API and ensures that the photo is placed in the right place for subsequent retrieval. However, it is in the works to delete the photo outright instead of placing it in the photo library.

* Interactions with moving to a subsequent parking lots within route

The left, right and cameraCapture button all run both the *getNextSpot* method. The left button, however, does not run *currentCam* while the other two buttons do. The *currentCam* method saves the licence plate string (if any) and changes the state of the parking to occupied(green or red) or unoccupied(grey) via cameraCapture and right button respectively.

The *currentCam* and *getNextSpot* methods are called from the Next Activity so it needs to continue running from the background.

### NextActivity

* The sorting of items into a dataset

On the GUI of this activity, each item on the list is initialized by the TemplateActivity. The template item makes use of four variables: State, parkSpot, plateNum and timeCheck. The state is represented as a color picture and the plateNum is represented as the title of the item. On click, the item will take the user to the Camera Activity to start an instance relative to the parking spot selected. The title of the Camera activity is the parkSpot and the red text would be the timeCheck.

* LiveQuery and Synchronization

Live Query is a local change detection from the couchbase dependency. It works on its own preset intervals as well as on data change. It runs on creation of the activity. Synchronization is push and pull replication towards the SYNC\_URL. It will need to be manually called on to produce change. Hence, the *getSync* method is placed in *onCreate* and *CurrentCam*. This is the most efficient in the case of a steady internet connection but could be a subject of debate when the internet connection could be irregular.

As such, there is a sync button in place which runs the *getSync*() method while also allowing the user the set the database location if needed.



# Route customization

Figure two below shows the default route separated into parts which are sequentially numbered and is the default A to B route.

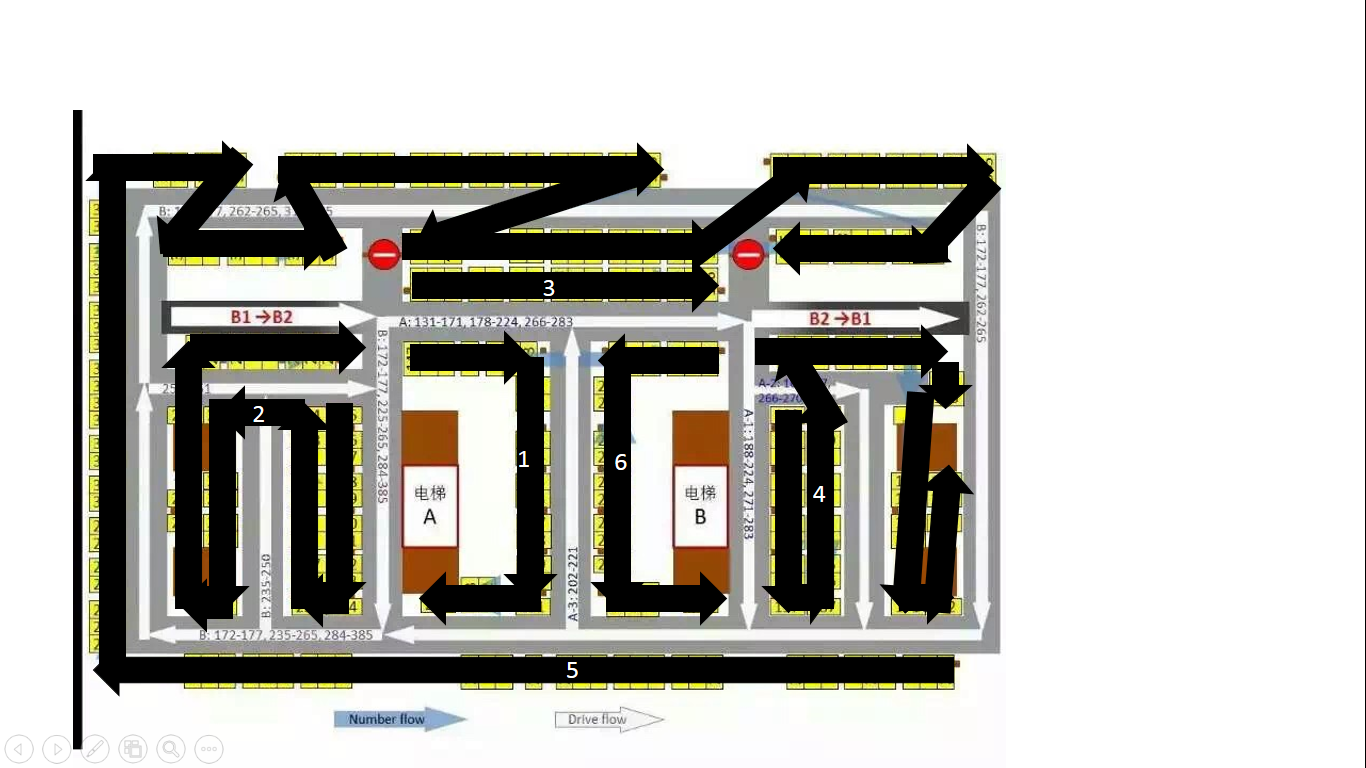


Figure 2 Map of the ET2 B2 carpark

## Route Explanations

|  |  |  |
| --- | --- | --- |
| Route | Function Description | Route parking |
| 1 | This route starts from elevator A. | 147-153 + 221-224 |
| 2 | This route is unchanged from the previous route planning | 225-261 |
| 3 | Short, straight and unchanged from previous planning | 131-146 |
| 4 | Combinations of two previous routes to make an efficient route | 188-197 +188 – 178 +160-167 +177-176 + 167-175 |
| 5 | The longest route and changed towards the end to allow the attendant to reach elevator B more efficiently | 262-364 + 374-385 +373-365 |
| 6 | Last route leading to elevator B | 198-210 + 154-159 |

# Relevant Android data

This section describes the current hardware and software of the phone used in the demo.

|  |  |
| --- | --- |
| Hardware | Information |
| Model | HTC M8St |
| Memory | 16GB |
| SDK | 21 |

Table Phone specs

# Dependencies

|  |  |  |
| --- | --- | --- |
| Dependency | Version | Description |
| EasyPR | 1.4 | Plate recognition software. |
| android.support | 23.4.0 | Source framework for all android apps |
| couchbase-lite | 1.3.1 | Allowes access to internal phone database as well as cloud |
| butterknife | 7.0.1 | a library of utilities to assist with bind variables to other components of the app. |

# Limitations

Currently, there is no option to create a custom route but it is in progress. Also synchronization can be unstable in areas with unstable connection. Synchronization has to work on demand even if the query is updated regularly on the phone.

Heavy use of the camera could be a huge usage of the battery. The EasyPR dependency has been known to place the pictures in different locations and has been modified by its author to fix this, but there could still be cases of photos not being found on the phone.