CHAPTER 1

INTRODUCTION

OVERVIEW OF THE PROJECT

Parking is an expense process in terms of either money or the time and effort spent for the “free spot chasing”. Current studies reveal that a car is parked for 95 percent of its lifetime and only on the road 5 percent. If we take England in 2014 as an example, on average a car was driven for 361 hours a year according to the British National Travel survey yielding about 8404 hours in which a car would be parked. Now where would you park your car for these very long hours? Cruising for parking is naturally the first problem caused by the increase of car owners globally. On average, 30 percent of traffic is caused by drivers wandering around for parking spaces. In 2006, a study in France revealed estimation those 70 million hours was spent every year in France only in searching for parking which resulted in the loss of 700 million euros annually. In 2011, a global parking survey of IBM states those 20 minutes is spent average in searching for a covered spot. With these statistics, we can assume that a great portion of global pollution and fuel waste is related to cruising for parking.

Parking spaces are found to be more than plenty in some places and very rare to find in others. Pricing policies had played an important role in the overall parking availability for decades. Here comes the important question: do we need to have more parking spaces or do we need better parking management? We believe it is the later and thus the motivation behind this work is about better parking management with fair and profitable pricing policies.

The work presented in this paper combines parking reservation and pricing models to overcome the parking problems. On the parking reservation side, Mouskos et al. modeled the reservation process as a resource allocation problem.

Their model is based on MILP with the objective of minimizing driver cost. Their model offers real time reservations with fixed pricing. Geng et al. had extended their work by taking into account the user's cost in terms of pricing and walking distance. In addition, they had expanded the model by adding fairness constraints and applying extensive simulations. Although the system proposed is very good, their model is still limited by being suitable only for short-term reservations and the parking revenue was not considered.

CHAPTER 2

LITERATURE SURVEY

2.1 Cloud Based Smart Parking System Based Internet of Things

Author:Thanh Nam Pham

Things Technologies This paper provides a unique algorithm which increases the capability of the current cloud based smart parking system and it also develops a network architecture based on the Internet of Things technology. This system helps the users to find a free parking space with minimal cost based on new performance metrics which is automatic. This metrics will calculate the user spaces in each car park. To enhance the parking management, an intelligent parking system was developed which reduced the purpose of hiring people to maintain the parking system . In this paper it proposes an effective cloud-based Smart parking system based on the Internet of Things.

2.2 Smart Routing: A Novel Application of Collaborative Pathfinding to Smart Parking Systems

Author:Callum Rhodes

A Novel Application of Collaborative Pathfinding to Smart Parking Systems In this paper smart parking system provides guidance to the drivers to find available parking spaces to avoid increasing parking issue. Traffic authorities in many metropolitan cities have initiated parking guidance and information (PGI) systems providing drivers with up-to-date information on the available parking spaces and direct the drivers accordingly. The information is provided to the driver over the internet.

The systems provide the location of the available car park based on the driver's current location in intended area or his final destination.

2.3. A New "Smart Parking" System Based on Optimal Resource Allocation and Reservations :

Author:Yanfeng Geng

In this system a new smart parking system is implemented for cities. This system assigns and reserves a parking space for a user (driver) based on the users distance from the parking area and parking cost and also ensures that the overall parking capacity is effectively utilized. Their approach solves a Mixed Integer Linear Program (MILP) problem at each decision point in a time-driven sequence. For each MILP there is a solution which gives an optimal allocation based on user's current state information and also supports random events such as new user requests and parking space availability. The allocation is updated at the next decision point which ensures that there is no parking slot reservation conflict and that no user is ever assigned a parking slot with higher than the current cost function value. This mechanism ensures a better response from the system

2.4 The Research and Implement of the Intelligent Parking

Author:Cui Shiyao

With the increasing development of economy and city modernization level, traffic congestion and parking have become major social issue due to the increasing amount vehicle density. In order to overcome this parking issue a smart parking system has been proposed in this paper which is composed of ZigBee network which sends the user requested information to PC through a coordinator and further updates the database .

Using the internet, the parking information is provided to the application layer to make it convenient for the people seeking for the parking position with the help of web-services. The system consists of mobile client and server side parking lot**.**

2.5 One of the intelligent systems for car parking has beenn proposed by making use of Image processing.

Author:M.Ataur Rehman.

In this system, a brown rounded image on the parking slot is captured using camera and processed to detect the free parking slot. The information about the currently available parking spaces is displayed on the segment display. Initially, the image of parking slots with brown-rounded image is taken. The image is segmented to create binary images. The noise is removed from this image and the object boundaries are identified. The image detection module determines which objects are round, by determining each object's area and perimeter. Accordingly, the free parking space is allocated.

2.6 A vision based car parking system

Author:Patrick Sebastian

Developed which uses two types of images positive and negative to detect free parking slot. In this method, the object classifier detects the required object within the input. Positive images contain the images of cars from various angles. Negative images do not contain any cars in them. The co-ordinates of parking lots specified are used as input to detect the presence of cars in the region.

However, limitations may occur with this system with respect to the type of camera used. Also, the co-ordinate system used selects specific parking locations and thus camera has to be at a fixed location. Limited set of positive and negative images may put limitations on the system.

2.7 Number Plate Recognition technique for developing process the number plates of the vehicles.

Author:Norazwinawati Basharuddin

In this system, the image of the license number plate of the vehicle is acquired. It is further segmented to obtain individual characters in the number plate. Ultrasonic sensors are used to detect free-parking slots. Then the images of number plate are taken and analyzed. Simultaneously, the current timing is noted so as to calculate the parking fees. The LCD displays 'FULL' sign to indicate that a parking slot is not available. However some limitations with the system include background color being compulsorily black and character color white. Also, analysis is limited to number plates with just one row.

2.8 Smart Parking system designed proposed a mechanical model

Author:M.A.R.Sarkar.

The car would be parked with the use of lift at multiple levels. Also, image processing is used to capture the number plate and store in database for comparison to avoid illegal car entry. Thus, we aim to design a car parking system that represents a fully automated model with minimum human intervention and overcome the limitations of previous systems.

2.9 Smart parking reservation system using short message services.

Author:Noor HazrinHanyMoham and Hanif

Enhanced security due to password requirement. System can be used and applied anywhere due to ease of usage. Cost of implementation is high. GSM feature creates bottlenecks .The microcontroller will have to take a lot of load which can crash the system

2.10 ZigBee and GSM based secure vehicle parking management and reservation system

Author: .Ashwin Sayeeraman

Parking lot vacancy module uses ZigBee along with PIC. Security Feature: The exit password must be entered else the user is not allowed to get out of the parking bay as the barrier gate will not get open until correct exit password is entered .

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

In the past researches the researchers concentrates more on the costing side of parking system. On the pricing side, Shoup et al. introduced new concepts which led to the development of San Francisco Park (SFPark) in San Francisco which aims to overcome the traffic congestion by dynamically changing prices based on sensor historical data. In SFPark, sensors are deployed on the asphalt to gather parking information that are stored in a database and processed weekly or monthly. According to historical data, the prices are increased and decreased proportional to the expected utilization. Although dynamically changing parking prices shall balance the supply and demand for parking and increase overall utilization, it is based on historical data and statistics which may not be accurate enough to have the proper effect. But still there is a problem of users to waiting for the long queue without knowing the car total availability and waiting time to park their car in the respective lane. So a new methodology is required to park the car into the lane in an efficient manner.

3.2 PROPOSED SYSTEM

In this approach, we present a new smart car parking system, named iParker, with static resource scheduling, dynamic resource allocation and pricing models, to optimize the parking system for both parking managers and drivers. The contributions of our work include:

1) Increasing parking resource utilization,

2) Increasing parking revenue,

3) Improving parking experience of drivers by lowering cost, parking spot searching and walking times. Our work is different from the one in past approach, where a dynamic resource allocation model was proposed. The main limitations of that model are that only reservation for limited period of time (e.g., few minutes) was allowed, fixed price was used and revenue was not taken into account and only a single choice of destination was considered. Whereas our model allows a driver to reserve a parking space for any time in future, the revenue is considered and new pricing models are introduced. In addition, a parking solution with their individual journey planners is proposed. Our work in the pricing side also differs from Shop's work as we propose a real-time dynamic pricing in parking and prove its effectiveness. We design our proposed system with two powerful entities such as ANDROID and C# web service applications. So that the users can work on this system with both browser based as well as mobile based systems without any interruptions.

CHAPTER 4
SYSTEM ARCHITECTURE

4.1 FRAMEWORK

Our new concept is to combine real time reservations (RTR) with share time reservations (STR), thus a driver can reserve a spot while heading to it (e.g., few minutes away) and also can reserve it at any time earlier (e.g., many days away). RTR are achieved by performing dynamic resource allocation which is similar to skills based routing in call centers. In the case of RTR, drivers are constantly allocated the best parking spots available until they reach their destinations. Whereas STR are achieved by performing static resource allocation that is based on time scheduling where a driver can explicitly choose the preferred resource and the time frame at which it will be occupied at anytime in the future. Different pricing policies for both types of reservations that are fair for drivers and parking managers are proposed in this paper. In addition, a dynamic pricing engine which periodically updates the parking prices based on real time resource utilization by occupancy and reservations and other events is introduced. iParker features the normal and disabled parking spots and drivers are given the freedom of choosing multiple destinations and the system will assign the optimal resources according to their chosen destinations and circumstances.

Throughout this paper, we will employ the term “parker” to refer to a driver or a car, “resource” to refer to a group of parking spots, “D-Type” to refer to a dynamic reservation, RTR or the type of a driver requesting a dynamic reservation and “S-Type” to refer to a static reservation, STR or the type of a driver requesting a static reservation, Architecture and Reaction Scheme.

iParker is a semi-distributed system as shown in Fig. 1: there are one central request center (CRC), one parking manager (PM) and multiple local smart allocation systems (SASs). The CRC receives parkers' requests, processes them and diverts them to the relevant local SAS. The request process is as following: parker chooses one to many destinations and if he/she is an S-type, preferred parking resource can be selected. Both types have to assign a weight parameter from 0-1 which reflects their desire between resource-destination proximity and resource price. Both types also set the maximum price and walking distance they can tolerate. For S-type, the spot occupancy interval has to be defined. For D-type, the GPS coordinates are measured and attached to the request. Finally the parker identifications (i.e., driver and car license IDs) are accompanied with the request. CRC also responds back to all parkers with reservation offers and in the same fashion notifies the local SAS with parker's response.

The PM is a central parking manager who is an interface among parking authorities, parking resource managers, SASs and local pricing engines. Parking authorities can use the PM to manually update the relevant pricing engine or data center. For instance, to fix pricing values for certain parking resource or update the data center with upcoming events near a relevant resource.

Below we describe the main components of a local smart allocation system:

Pricing Engine - Pricing engines are small applications that run a pricing model on web-servers. The duties of a pricing engine are to fetch parking utilization data and updates from parking authorities every predefined time interval and to set the new parking prices accordingly. The engine runs independent on the SAS, calculates the new prices and updates the data center.

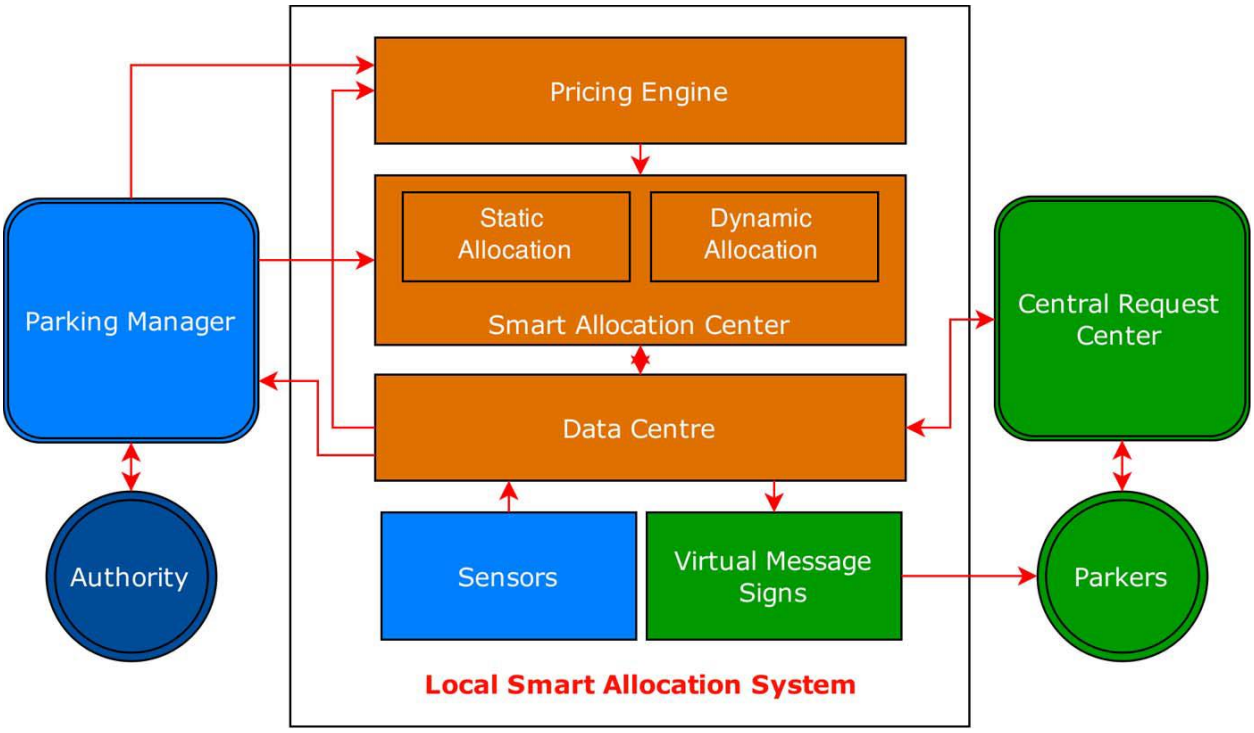


Fig. 4.1.1 iParker Framework.

Sensors - Every resource is occupied with a spot occupancy detection system. Ideally this system must provide accurate data on the utilization of the parking resource, deployed either indoors or outdoors. The detection system is normally composed of a wireless/wired sensor network that can provide occupation state of every parking spot, or alternatively composed of counter sensors at the entrance and exit of parking lots that is only capable of providing total utilization value. The later method can only work in controlled environments, thereforewe prefer to use sensor networks and a central processor that updates the data centre with the utilization values.

Data Centre - Holds all the information from all iParker components and store them in a structured data container. It's consisted of a pricing table which contains the up to date information on pricing per resource per minute, utilization table

which holds the utilization data, and finally authority table which stores other parameters that is set by parking authorities (e.g., events related). A Data Centre is also responsible for updating multiple types of virtual message signs and public devices of up to date pricing information and parking availability.

Smart Allocation Centre - A web service that runs a sophisticated MILP model that optimally and fairly assigns/reserves parking resources to the parkers. The assignment is based on key variables that are not limited to driver constraints, current resource utilization, up to date pricing information and events occurrences. The center provides nonstop parking reservation service to the parkers and is described in details in the next section.

Virtual Message Sign (VMS) - Updates parkers/public with up to date pricing and parking availability information. This is achieved by deploying numerous numbers of VMS panels across cities especially around on-street parking areas. For off-street parking lots, one VMS panel at the entrance is sufficient to inform arrivals of updated information. It is important to mention that a parker will only pay according to the price rate fixed in the reservation offer. If the parker is not using the service, he/she will pay according to the price rate displayed at the time of his/her parking. VMS is specially and critically important for non smart-phone.

4.2 SMART RESOURCE ALLOCATION

The problem addressed in this study combines the real time and share time reservation systems. Real time reservations are typically independent on the amount a parker will consume in a parking space, i.e., a parker can spend as much time as he/she needs without affecting the rest of the parkers.

On the other hand, share time reservations are dependent on the exact spot occupancy and spot leave times. Share time reservations are generally modeled as birth-death stochastic processes. In our model, dynamic reservations are real time and static reservations are share time.

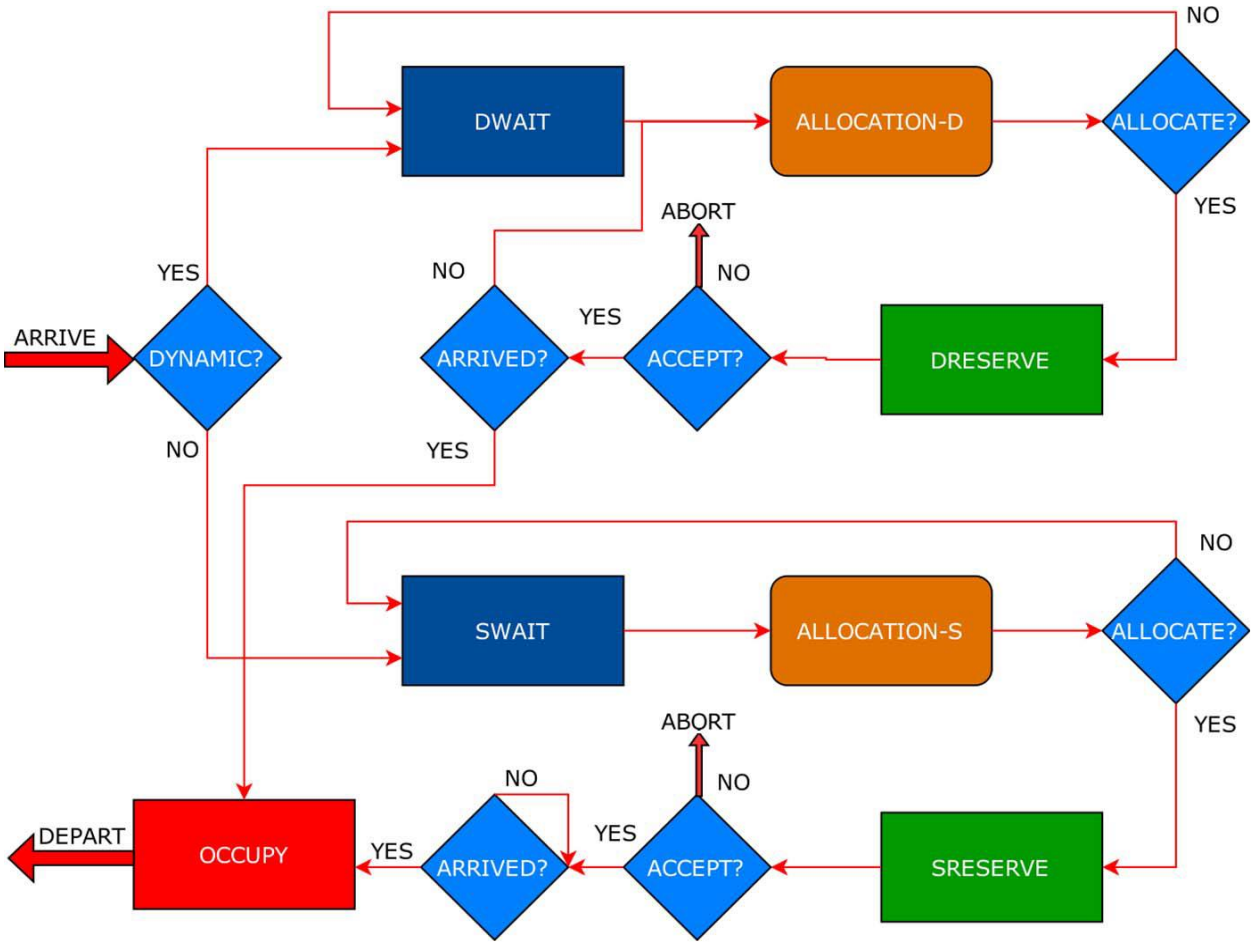


Fig.4.1 2. Queuing model flow.

The objective of our MILP model is to minimize the total monetary cost for parkers and ultimately maximize the total resource utilization to obtain the maximum revenue for parking managers. We will formulate our model based on the queuing model in Fig. 2.

There are N resources in which every resource j is split into P1 spots (number of normal parking spots for dynamic reservations), P2 spots (number of normal parking spots for static reservations) P3 and P4 (similar to P1 and P2 but for disabled people).

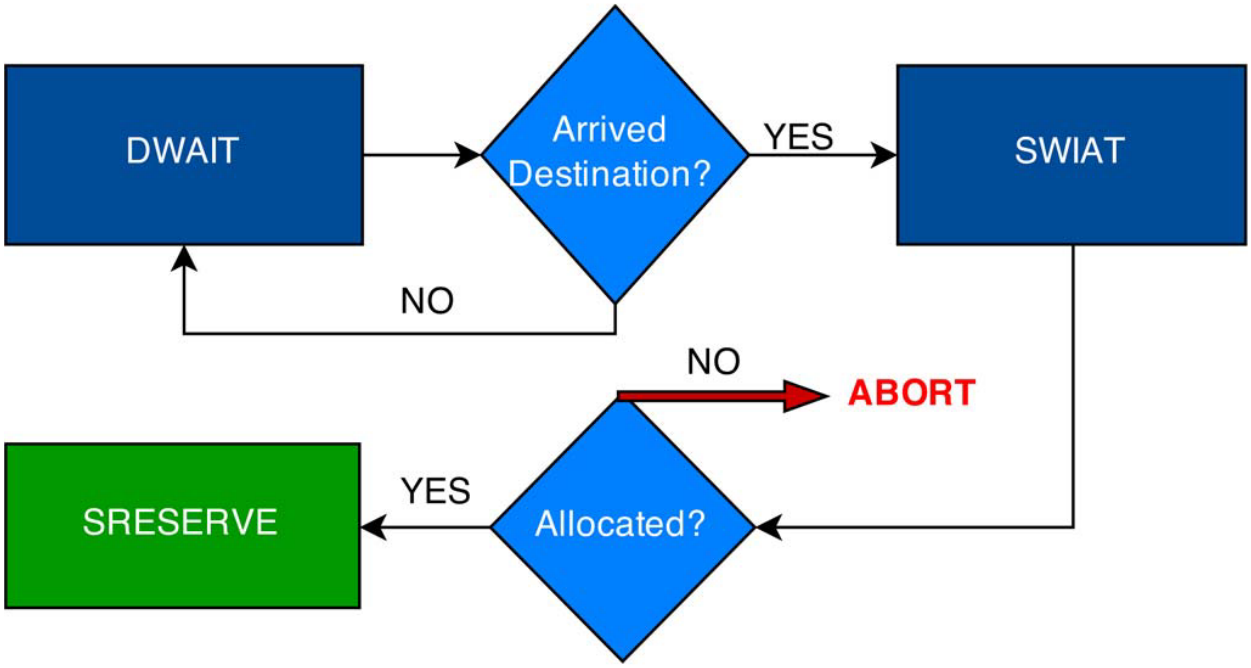


Fig.4.1.3. Static-dynamic reservations interface

The running time of the smart allocation centre is discretized into small time periods. We will denote each time period as a decision pointK. All parkers arrive to the allocation center randomly and independently joins the relevant WAIT queue. At each decision point, the allocation centre will allocate resources to dynamic and static parkers and move them to the relevant RESERVE queue. Parkers in the dynamic reserve queue (DRESERVE) will get re-allocated a better parking spot (if available) after each decision point until they reach a defined zone defined as their first destination. Parkers in the static wait queue (SWAIT) will only get allocated once and then move to the static reserve queue (SRESERVE).

When parkers arrive to their resources, they will be moved to the occupy queue and then P1, P2, P3 or P4 will be decremented by the number of parkers. When parkers leave the parking spaces, they will be completely removed from the system and again the count of spaces would be incremented.

To fully maximize the utilization of resources, we will initialize the system by making 50% of the resources to be “dynamic” and 50% to be “static.” Then we will follow the strategy in Fig. 3: when parkers in DWAIT queue reach their destination and fail to get allocated for the reason that P1 or P3 became zero (i.e., no free parking spaces for dynamic reservers), the system automatically diverts them to SWAIT queue, such that they get a chance for allocation for the “static” resources.

4.3 DATA FLOW DIAGRAM

LEVEL – 0

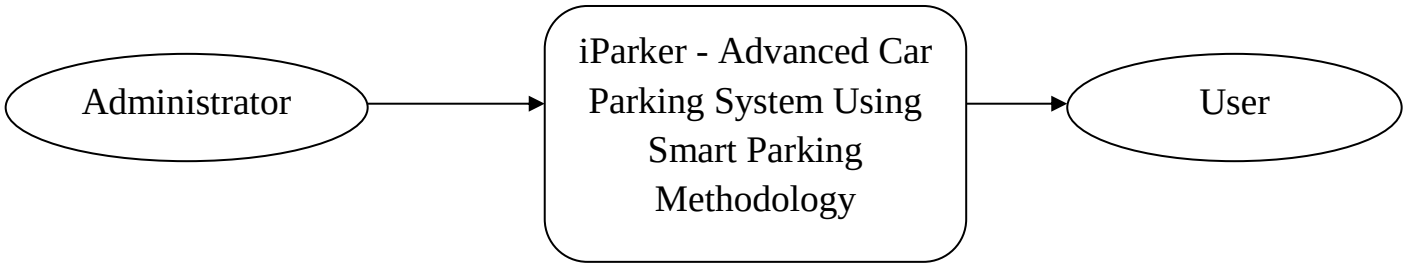


Fig. 4.3.1 Level 0

LEVEL – 1

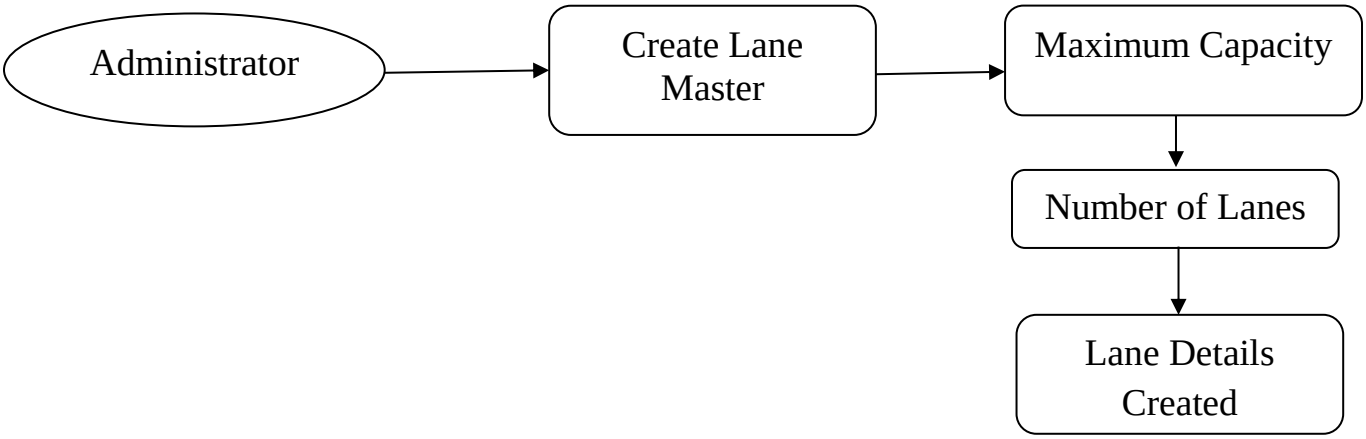


Fig. 4.3.2 Level 1

LEVEL - 2

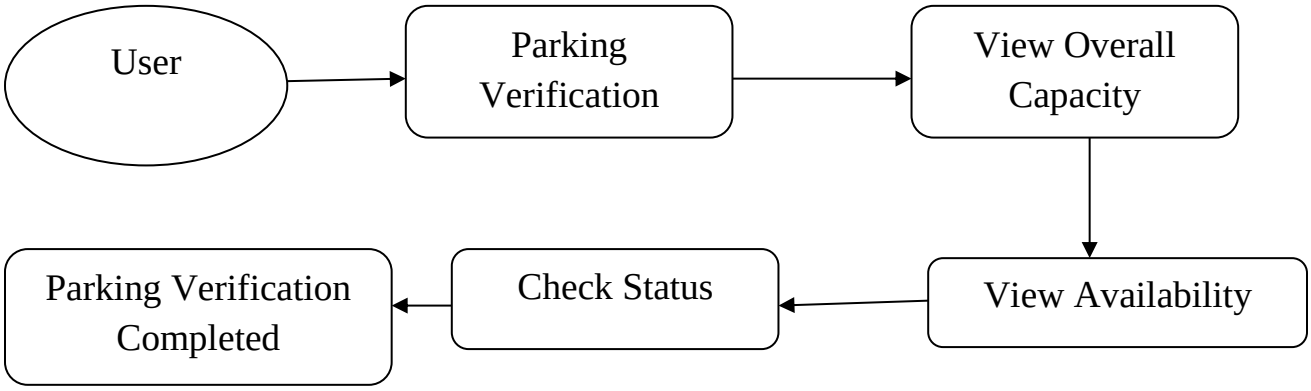


Fig. 4.3.3 Level 2

LEVEL – 3

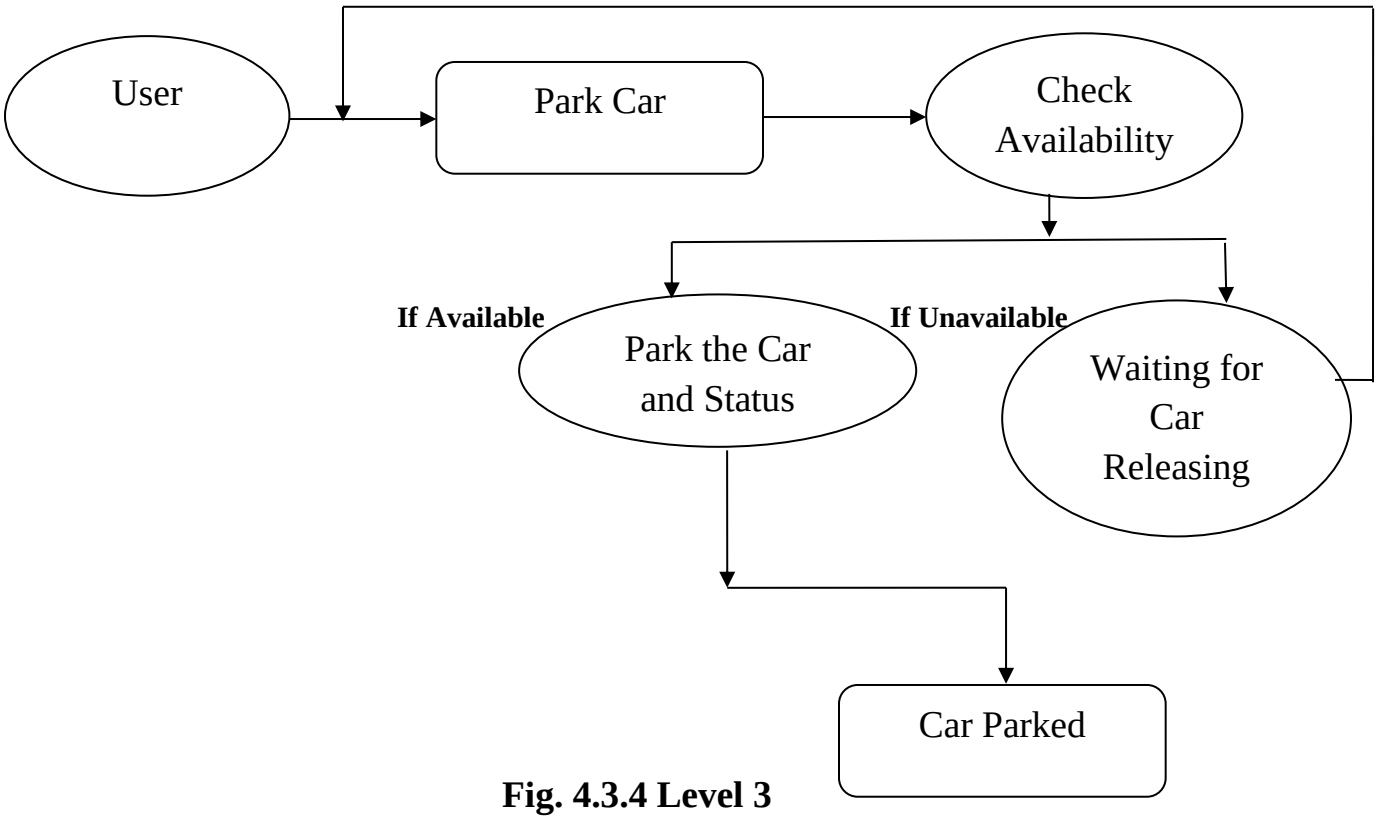


Fig. 4.3.4 Level 3

LEVEL – 4

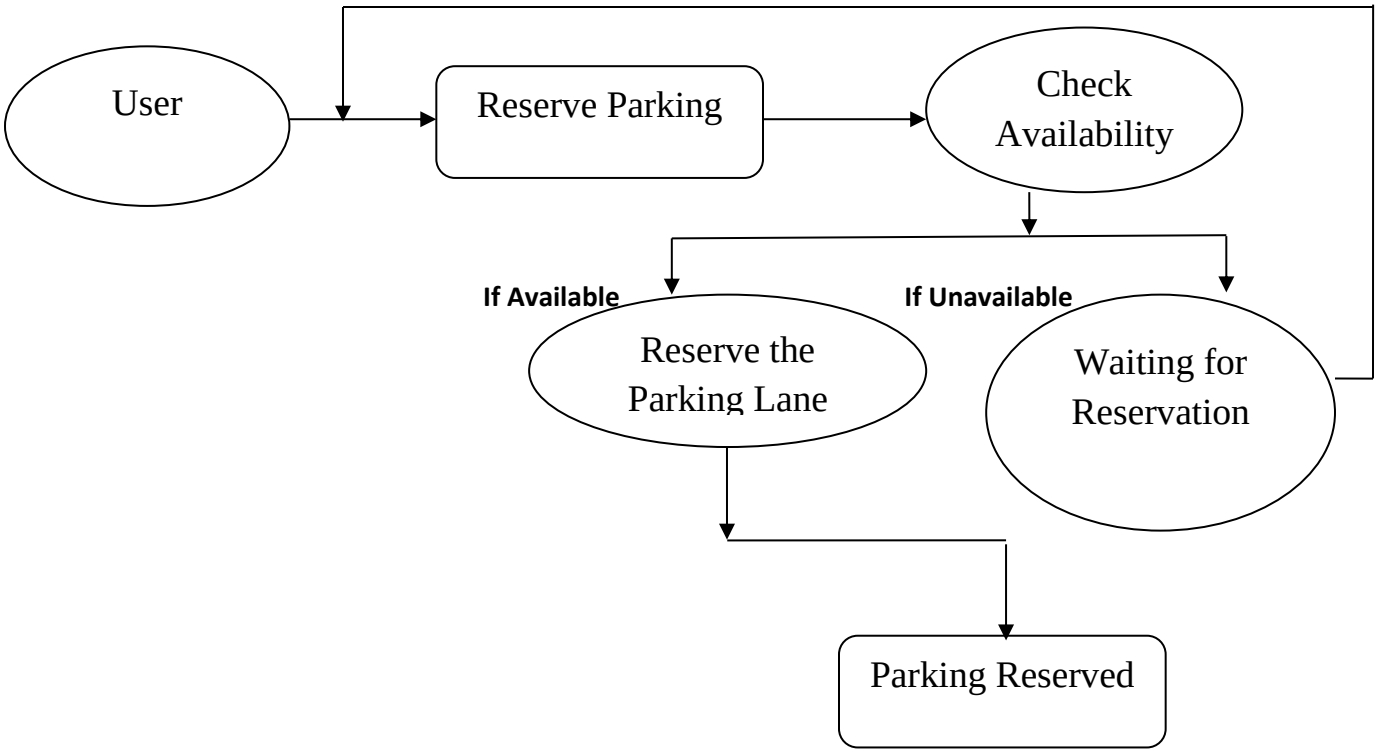


Fig. 4.3.5 Level 4

LEVEL - 5

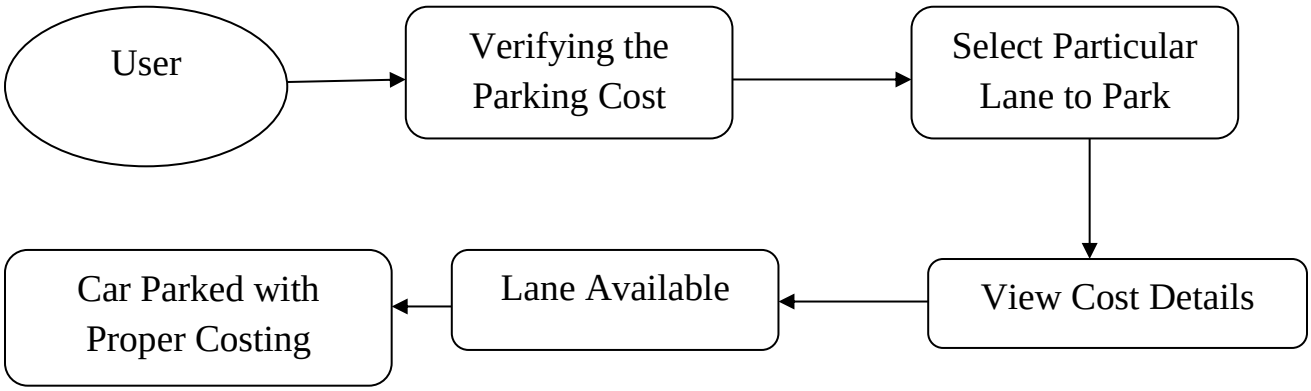


Fig. 4.3.6 Level 5

CHAPTER 5

LANGUAGE SPECIFICATION

MICROSOFT.NET FRAMEWORK

FEATURES OF .NET

Microsoft .NET is a set of Microsoft software technologies for rapidly building and integrating XML Web services, Microsoft Windows-based applications, and Web solutions. The .NET Framework is a language-neutral platform for writing programs that can easily and securely interoperate. There's no language barrier with .NET: there are numerous languages available to the developer including Managed C++, C#, Visual Basic and Java Script. The .NET framework provides the foundation for components to interact seamlessly, whether locally or remotely on different platforms. It standardizes common data types and communications protocols so that components created in different languages can easily interoperate.

".NET" is also the collective name given to various software components built upon the .NET platform. These will be both products (Visual Studio.NET and Windows.NET Server, for instance) and services (like Passport, .NET My Services, and so on).

THE .NET FRAMEWORK

The .NET Framework has two main parts:

1. The Common Language Runtime (CLR).
2. A hierarchical set of class libraries.

The CLR is described as the "execution engine" of .NET. It provides the environment within which programs run. The most important features are

- Conversion from a low-level assembler-style language, called Intermediate Language (IL), into code native to the platform being executed on.
- Memory management, notably including garbage collection.
- Checking and enforcing security restrictions on the running code.
- Loading and executing programs, with version control and other such features.
- The following features of the .NET framework are also worth description:

MANAGED CODE

The code that targets .NET, and which contains certain extra Information - "metadata" - to describe itself. Whilst both managed and unmanaged code can run in the runtime, only managed code contains the information that allows the CLR to guarantee, for instance, safe execution and interoperability.

MANAGED DATA

With Managed Code comes Managed Data. CLR provides memory allocation and Deal location facilities, and garbage collection. Some .NET languages use Managed Data by default, such as C#, Visual Basic.NET and JScript.NET, whereas others, namely C++, do not. Targeting CLR can, depending on the language you're using, impose certain constraints on the features available. As with managed and unmanaged code, one can have both managed and unmanaged data in .NET applications - data that doesn't get garbage collected but instead is looked after by unmanaged code.

COMMON TYPE SYSTEM

The CLR uses something called the Common Type System (CTS) to strictly enforce type-safety. This ensures that all classes are compatible with each other, by describing types in a common way. CTS define how types work within the runtime, which enables types in one language to interoperate with types in another language, including cross-language exception handling. As well as ensuring that types are only used in appropriate ways, the runtime also ensures that code doesn't attempt to access memory that hasn't been allocated to it.

COMMON LANGUAGE SPECIFICATION

The CLR provides built-in support for language interoperability. To ensure that you can develop managed code that can be fully used by developers using any programming language, a set of language features and rules for using them called the Common Language Specification (CLS) has been defined. Components that follow these rules and expose only CLS features are considered CLS-compliant.

LANGUAGES SUPPORTED BY .NET

The multi-language capability of the .NET Framework and Visual Studio .NET enables developers to use their existing programming skills to build all types of applications and XML Web services. The .NET framework supports new versions of Microsoft's old favorites Visual Basic and C++ (as VB.NET and Managed C++), but there are also a number of new additions to the family.

Visual Basic .NET has been updated to include many new and improved language features that make it a powerful object-oriented programming language. These features include inheritance, interfaces, and overloading, among others. Visual Basic also now supports structured exception handling, custom attributes and also supports multi-threading.

Visual Basic .NET is also CLS compliant, which means that any CLS-compliant language can use the classes, objects, and components you create in Visual Basic .NET.

Managed Extensions for C++ and attributed programming are just some of the enhancements made to the C++ language. Managed Extensions simplify the task of migrating existing C++ applications to the new .NET Framework.

C# is Microsoft's new language. It's a C-style language that is essentially "C++ for Rapid Application Development". Unlike other languages, its specification is just the grammar of the language. It has no standard library of its own, and instead has been designed with the intention of using the .NET libraries as its own.

Microsoft Visual J# .NET provides the easiest transition for Java-language developers into the world of XML Web Services and dramatically improves the interoperability of Java-language programs with existing software written in a variety of other programming languages.

Active State has created Visual Perl and Visual Python, which enable .NET-aware applications to be built in either Perl or Python. Both products can be integrated into the Visual Studio .NET environment. Visual Perl includes support for Active State's Perl Dev Kit.

C#.NET is also compliant with CLS (Common Language Specification) and supports structured exception handling. CLS is set of rules and constructs that are supported by the CLR (Common Language Runtime). CLR is the runtime environment provided by the .NET Framework; it manages the execution of the code and also makes the development process easier by providing services C#.NET is a CLS-compliant language. Any objects, classes, or components that

created in C#.NET can be used in any other CLS-compliant language. In addition, we can use objects, classes, and components created in other CLS-compliant languages in C#.NET .The use of CLS ensures complete interoperability among applications, regardless of the languages used to create the application.

ANDROID

Android is a Linux-based operating system for mobile devices such as smart phones and tablet computers. It is developed by the Open Handset Alliance led by Google. Google purchased the initial developer of the software, Android Inc., in 2005. The unveiling of the Android distribution in 2007 was announced with the founding of the Open Handset Alliance, a consortium of 86 hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices. Google releases the Android code as open-source, under the Apache License. The Android Open Source Project (AOSP) is tasked with the maintenance and further development of Android. Android was listed as the best-selling Smartphone platform worldwide.

FOUNDATION

Android, Inc. was founded in Palo Alto, California, United States in October, 2003 by Andy Rubin (co-founder ofDanger), Rich Miner (co-founder of Wildfire Communications, Inc.), Nick Sears (once VP at T-Mobile), and Chris White (headed design and interface development at WebTV). Acquisition by Google acquired Android Inc. on August 17, 2005, making Android Inc. a wholly owned subsidiary of Google Inc. Key employees of Android Inc., including Andy Rubin, Rich Miner and Chris White, stayed at the company after the acquisition. Not much was known about Android Inc. at the time of the acquisition, but many assumed that Google was planning to enter the mobile phone market with this move.

OPEN HANDSET ALLIANCE

On November 5, 2007, the Open Handset Alliance, a consortium (A consortium is an association of two or more individuals, companies, organizations or governments (or any combination of these entities) with the objective of participating in a common activity or pooling their resources for achieving a common goal.) of several companies which include Broadcom Corporation, Google, HTC, Intel, LG, Marvell Technology Group, Motorola, Nvidia, Qualcomm, Samsung Electronics, Sprint Nextel, T-Mobile and Texas Instruments unveiled itself. The goal of the Open Handset Alliance is to develop open standards for mobile devices. On the same day, the Open Handset Alliance also unveiled their first product, Android, a mobile device platform built on the Linux kernel version 2.6

LINUX KERNEL

The Linux kernel is the operating system kernel used by the Linux family of Unix-like operating systems. It is one of the most prominent examples of free and open source software version 2.6.

KERNEL

In computing, the kernel is the main component of most computer operating systems, it is a bridge between applications and the actual data processing done at the hardware level. On December 9, 2008, 14 new members joined, including ARM Holdings, Atheros Communications, Asustek Computer Inc, Garmin Ltd, Huawei Technologies, Packet Video, Softbank, Sony Ericsson, Toshiba Corp, and Vodafone Group Plc.

ANDROID OPEN SOURCE PROJECT

The Android Open Source Project (AOSP) is led by Google, and is tasked with the maintenance and development of Android. According to the project "The goal of the Android Open Source Project is to create a successful real-world product that improves the mobile experience for end users." AOSP also maintains the Android Compatibility Program, defining an "Android compatible" device "as one that can run any application written by third-party developers using the Android SDK and NDK", to prevent incompatible Android implementations. The compatibility program is also optional and free of charge, with the Compatibility Test Suite also free and open-source.

DESIGN

Android consists of a kernel based on the Linux kernel, with middleware, libraries and APIs written in C and application software running on an application framework which includes Java-compatible libraries based on Apache Harmony. Android uses the Dalvik virtual machine with just-in-time compilation to run Dalvik dex-code (Dalvik Executable), which is usually translated from Java byte code. The main hardware platform for Android is the ARM architecture. There is support for x86 from the Android x86project, and Google TV uses a special x86 version of Android.

CHAPTER 6

REQUIREMENT SPECIFICATION

6.1 System Requirements

6.1.1 Hardware Requirements

System	: Pentium IV 2.4 GHz
Hard Disk	: 40 GB
Ram	: 4 GB

6.1.2 Software Requirements

Operating system	: Windows XP
Technology Used	: Microsoft .NET and ANDROID
Coding Language	: C# and Java
Backend Used	: SQL Server

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

In this system we have proposed iParker, a new smart parking system which is based on MILP model that yields optimal solution for dynamically and statically allocating parking resources to parkers—providing flexible reservation options. The new concepts introduced in this paper are the combination of real-time reservations with share-time reservations, dynamically performing system decisions (reservation time constraints and pricing) according to real-time utilization information, and offering the drivers the choice of choosing multiple destinations and reservation type. We also have proposed pricing policies for both static and dynamic reservations that maximize the profit from parking. Extensive simulation results indicate that the proposed system significantly cuts the total effective cost for all parkers by as much as 28%, maximizes the total utilization by up to 21% and increases the total revenue for parking management up to 16% as compared to the non-guided parking system. Finally we proposed a dynamic pricing scheme and by integrating it to iParker's model, we found by simulations that it balances the utilization across all the parking resources and thus assist in eliminating the overall traffic congestion caused by parking.

7.2 FUTURE WORK

Currently, the research focuses on a new parking sensing infrastructure and an indoor navigation service for car parking. In the future, we aim to evaluate our system using real-time data and greater number of resources and destinations. In addition, a scalability analysis is to be performed to examine the efficiency of the proposed scalability techniques.

APPENDIX 1

SAMPLE CODING

Adminlogin.java

```
package com.example.parking_vehicles;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
public class AdminLogin extends Activity {
private Button Login;
private TextView admin;
private TextView pass;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.adminlogin);
Login=(Button) findViewById(R.id.btnAdminLogin);
admin=(TextView) findViewById(R.id.txtAdminName);
pass=(TextView) findViewById(R.id.txtAdminPass);
Login.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
```

```
if (admin.getText().length()>0 && pass.getText().length()>0){

if(admin.getText().toString().equals("admin")&&pass.getText().toString().equals("
admin")){

admin.setText("");

pass.setText("");

Intent myIntent = new Intent(AdminLogin.this,AdminMain.class);

startActivity(myIntent);

}

else{

Toast.makeText(getApplicationContext(), "Please Check the Username and
Password", Toast.LENGTH_LONG).show();} }

else{

Toast.makeText(getApplicationContext(), "Please Enter Username and Password",
Toast.LENGTH_LONG).show();  }}}); }

@Override

public boolean onCreateOptionsMenu(Menu menu) {

// Inflate the menu; this adds items to the action bar if it is present.

getMenuInflater().inflate(R.menu.main, menu);

return true;}}
```

Administrative Main Page

```
package com.example.parking_vehicles;

import android.os.Bundle;

import android.app.Activity;

import android.content.Intent;

import android.view.Menu;

import android.view.View;

import android.widget.Button;
```

```
public class AdminMain extends Activity {
private Button VU;
private Button LM;
private Button BLM;
private Button CPS;
private Button BPS;
private Button Signout;
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.adminmain);
VU=(Button) findViewById(R.id.CmdViewUsersAM);
LM=(Button) findViewById(R.id.CmdLaneMasterAM);
BLM=(Button) findViewById(R.id.CmdLMBPAM);
CPS=(Button) findViewById(R.id.CmdCarParkingSummaryAM);
BPS=(Button) findViewById(R.id.CmdBikeParkingSummaryAM);
Signout=(Button) findViewById(R.id.CmdSignoutAM);
VU.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
Intent a=new Intent(AdminMain.this,viewusers.class);
startActivity(a);}});
LM.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
Intent a=new Intent(AdminMain.this,LaneDtlsMaster.class);
startActivity(a);}});
BLM.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
```

```
Intent a=new Intent(AdminMain.this,LaneDtlsMasterB.class);
startActivity(a); }});
CPS.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
Intent a=new Intent(AdminMain.this,viewbookadmin.class);
startActivity(a);}});
BPS.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
Intent a=new Intent(AdminMain.this,viewbikebookadmin.class);
startActivity(a);}});
Signout.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
Intent a=new Intent(AdminMain.this,mainpage.class);
startActivity(a);}});
}
@Override
public boolean onCreateOptionsMenu(Menu menu) {
// Inflate the menu; this adds items to the action bar if it is present.
getMenuInflater().inflate(R.menu.main, menu);
return true;
}
}
```

Lane master

```
package com.example.parking_vehicles;
import android.os.Bundle;
import android.app.Activity;
```

```
import android.content.ContentValues;
import android.content.Context;
import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
public class LaneDtlsMaster extends Activity {
SQLiteDatabase myDb;
String DB_NAME ="Parking.db";
private TextView LID;
private TextView LName;
private TextView Descr;
private TextView City;
private Button Submit;
private Button Back;
 int LIDs=0;
String LNames="";
String Descrs="";
String Citys="";
String alert="";
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
```

```
setContentView(R.layout.carlanemaster);
Submit=(Button) findViewById(R.id.CmdSubLM);
Back=(Button) findViewById(R.id.CmdBackLM);
LID=(TextView) findViewById(R.id.TxtLIDLM);
LName=(TextView) findViewById(R.id.TxtLaneLM);
Descr=(TextView) findViewById(R.id.TxtDescLM);
City=(TextView) findViewById(R.id.TxtCityLM);
        createTable();
        LID.setText(""+incID());
Submit.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
initdata(); // Read the Textbox Values
alert="";
if(LName.getText().toString().trim().length()<=0){
alert+="Enter Lane Name\n";}
SQLiteDatabase myDb = openOrCreateDatabase(DB_NAME,Context.MODE_PRIVATE, null);
Cursor c=myDb.rawQuery("SELECT * FROM CLaneDtls WHERE LName='"+LName.getText().toString()+"' And City='"+City.getText().toString()+"'",null);
if(c.moveToNext()){
alert+="Parking Lane Details Duplicated\n";
}
if(Descr.getText().toString().trim().length()<=0){
alert+="Enter Description\n";
}
if(City.getText().toString().trim().length()<=0){
```

```
alert+="Enter City\n";
}
if(!alert.trim().equals("")){
Toast.makeText(LaneDtlsMaster.this,""+alert,Toast.LENGTH_LONG).show(); }
                else{
saveData(LIDs,LNames,Descrs,Citys);
Cleardata();}}  });
Back.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
Intent a=new Intent(LaneDtlsMaster.this,AdminMain.class);
startActivity(a);
}});}
void createTable() {
SQLiteDatabase                    myDb                    =
openOrCreateDatabase(DB_NAME,Context.MODE_PRIVATE, null);
String MySQL = "create table if not exists CLaneDtls(LID Integer not null,LName
Text not null,Description TEXT not null,City TEXT not null);";
myDb.execSQL(MySQL);
myDb.close();}
public int incID(){
int ID=0;
try{
SQLiteDatabase                    myDb                    =
openOrCreateDatabase(DB_NAME,Context.MODE_PRIVATE, null);
Cursor c = myDb.rawQuery("Select MAX(LID) From CLaneDtls",null);
if(c.moveToNext()){
ID=c.getInt(0);
```

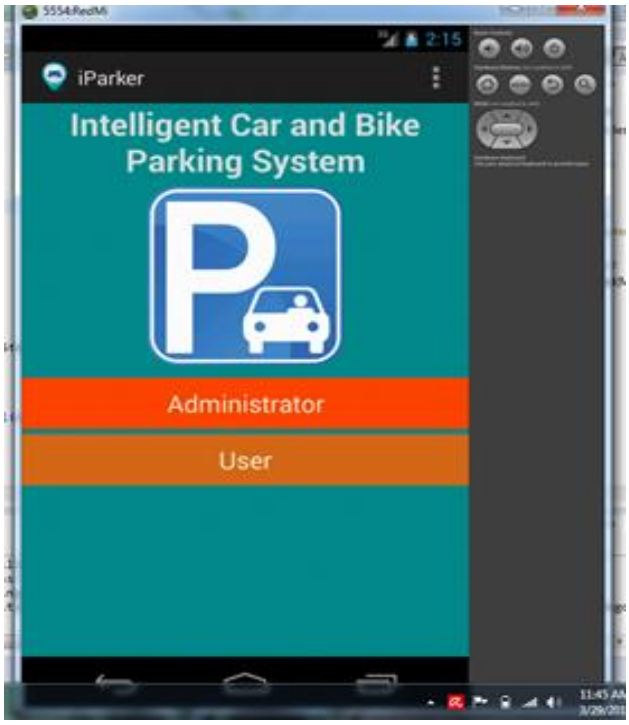
```
if(ID==0){
ID=1;}
        else{
        ID=ID+1;
}}}
catch(Exception e){
ID=1;}
return ID;}
void Cleardata(){
LID.setText(""+incID());
LName.setText("");
Descr.setText("");
City.setText("");
}
void initdata(){
try{
LIDs=incID();
LNames=LName.getText().toString();
Descrs=Descr.getText().toString();
Citys=City.getText().toString();}
catch(Exception e){
Toast.makeText(LaneDtlsMaster.this,"Loaded...",Toast.LENGTH_LONG).show();
}}
void saveData(int IDd,String LNamed,String Descrd,String Cityd) {
SQLiteDatabase myDb = openOrCreateDatabase(DB_NAME,Context.MODE_PRIVATE, null);
```

```
ContentValues newValues = new ContentValues();
newValues.put("LID", IDd);
newValues.put("LName", LNamed);
newValues.put("Description", Descrd);
newValues.put("City", Cityd);
long ret = myDb.insert("CLaneDtls", null, newValues);
if (ret > 0){
Toast.makeText(LaneDtlsMaster.this,"Parking Lane Details Created Successfully",Toast.LENGTH_LONG).show();}
myDb.close();}
public boolean onCreateOptionsMenu(Menu menu) {
// Inflate the menu; this adds items to the action bar if it is present.
getMenuInflater().inflate(R.menu.main, menu);
return true;
}}
```

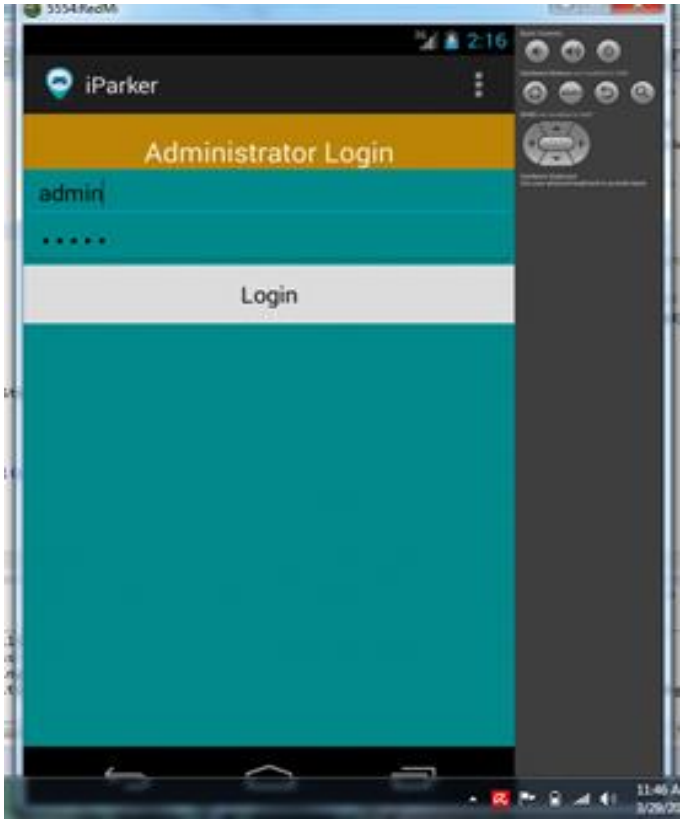
APPENDIX 2

SCREENSHOTS

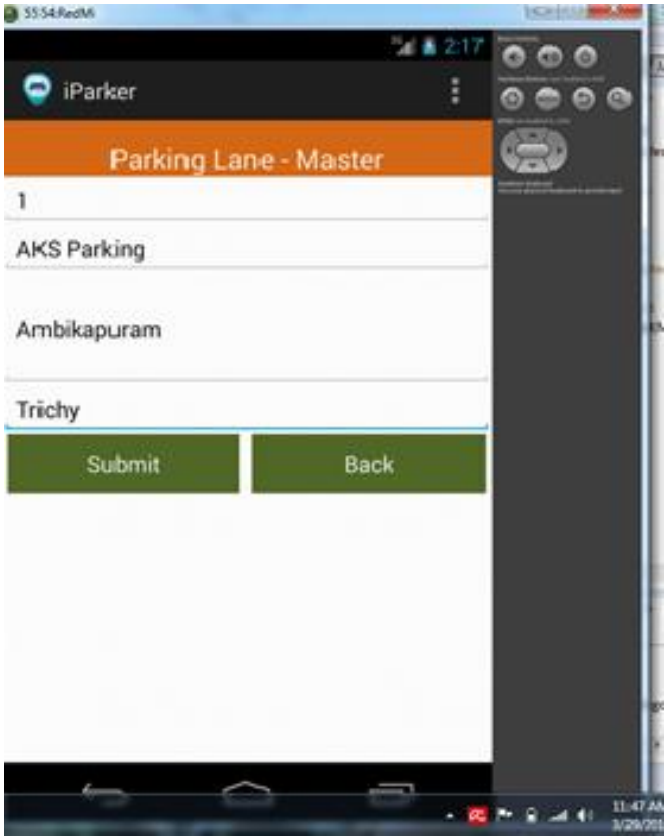
1. WELCOME PAGE



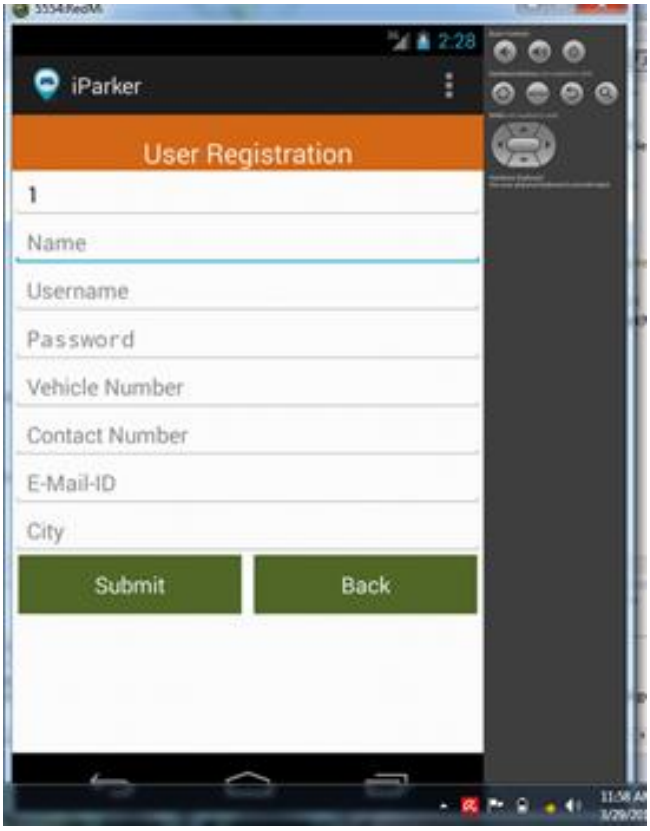
2. Administrative Login



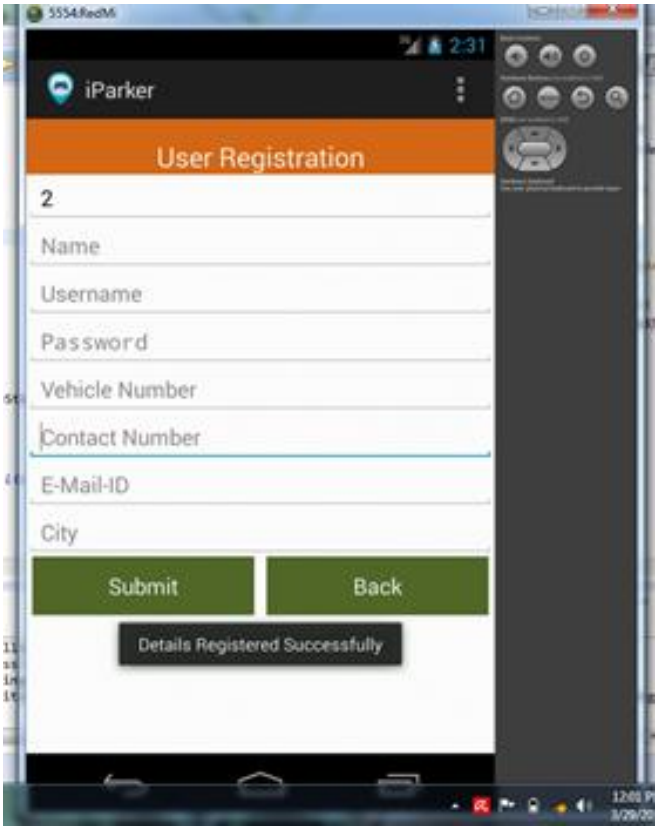
3. Creating Lane masters



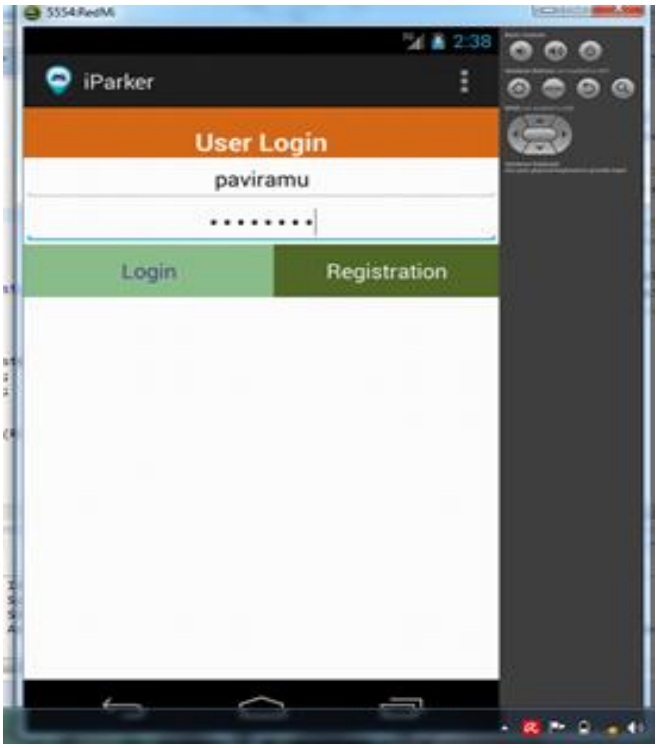
4. User registration



5. Confirmation



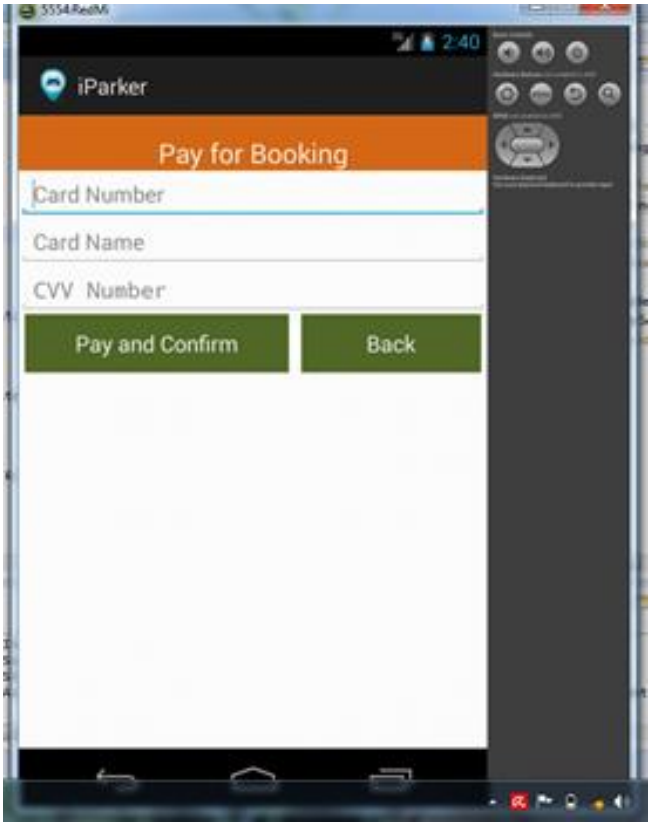
6. User login after registering



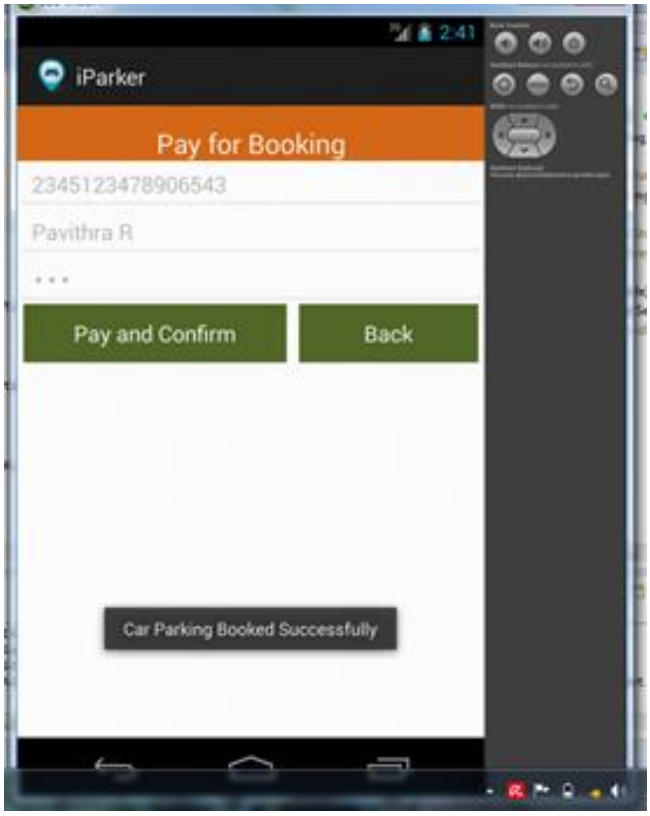
7. Booking for car parking

8. Slot booking

9. Making payment

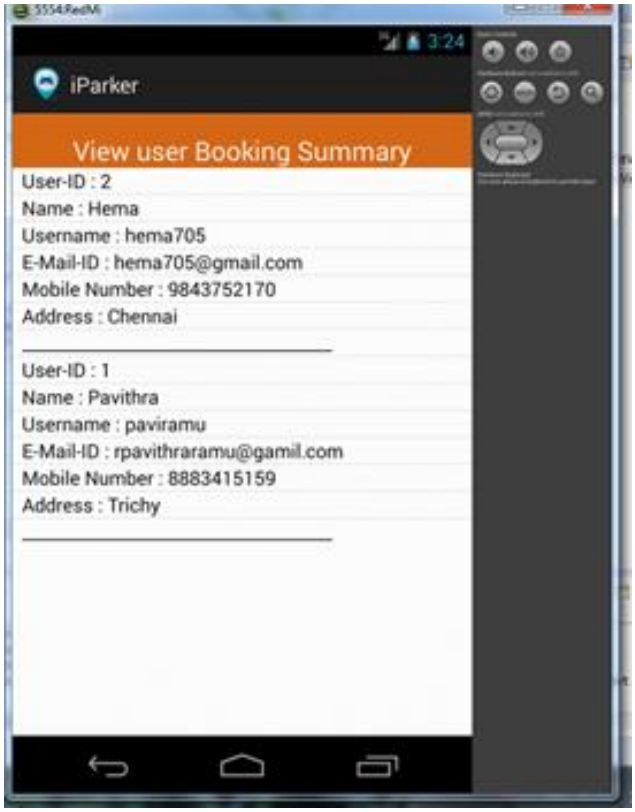


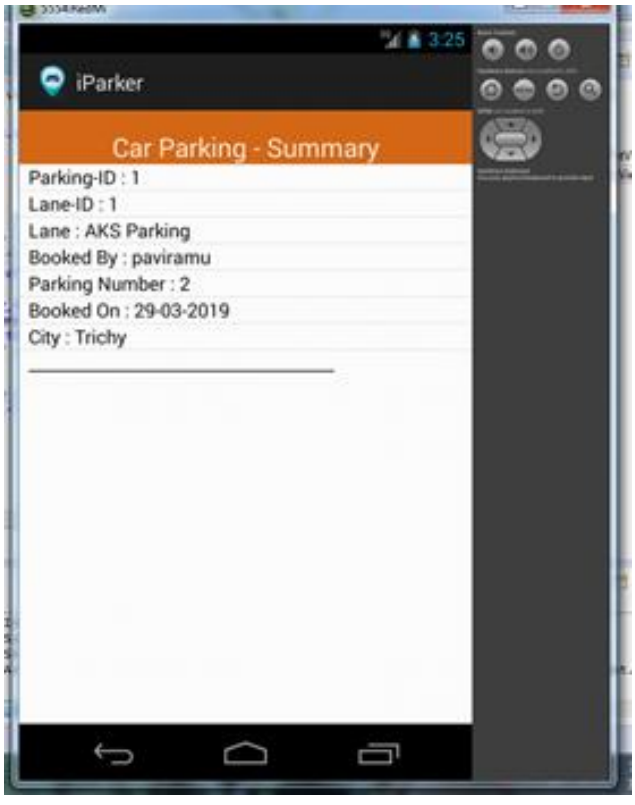
10. Confirmation of booking



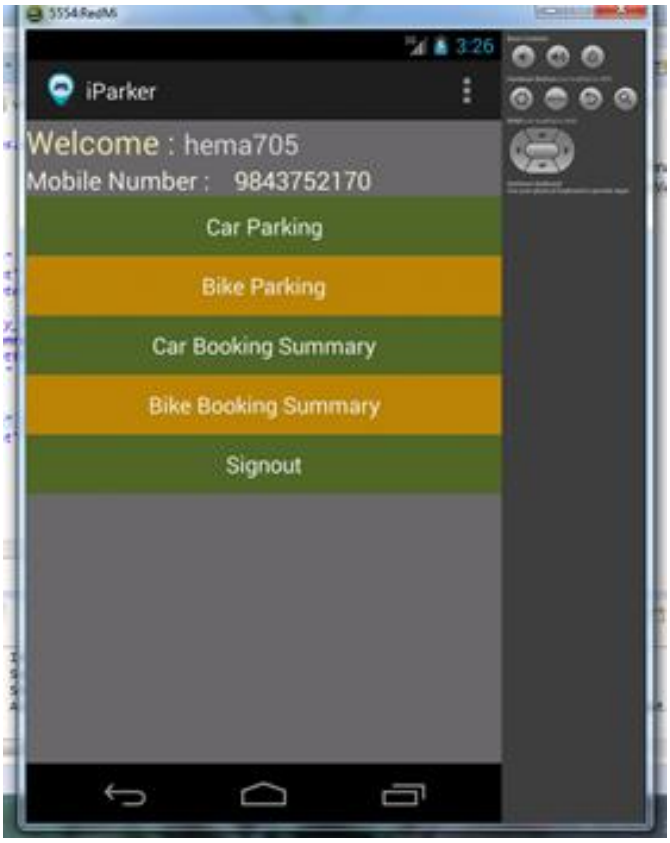
11.Summary of car parking

12. Admin viewing the summary of users and car parking





13.Bike parking



14. Booking a slot

15. Payment for booking

16. Administrator viewing the summary of parked car and bike

REFERENCES

[1]. Thanh Nam Pham1, Ming-Fong Tsai1, Duc Bing Nguyen1, Chyi-Ren Dow1 and Der-Jiunn Deng2. “A CloudBased Smart-Parking System Based on Internet-of-Things Technologies”. IEEE Access, volume 3, pp. 1581 – 1591, september 2015.

[2]. Callum Rhodes, William Blewitt, Craig Sharp, Gary Ushaw and Graham Morgan. “Smart Routing: A Novel Application of Collaborative Path-finding to Smart Parking Systems”. Business Informatics (CBI), 2014 IEEE Conference on volune 1, pp. 119-126, 2014.

[3]. Yanfeng Geng and Christos G. Cassandras. “A New Smart Parking System Based on Optimal Resource Allocation and Reservations”. IEEE Transaction on Intelligent Transportation Systems , volume 14, pp. 1129 - 1139, April 2013.

[4]. Cui Shiyao, Wu Ming, Liu Chen, Rong Na . “The Research and Implement of the Intelligent Parking Reservation Management System Based on ZigBee Technology”. Measuring Technology and Mechatronics Automation (ICMTMA) , pp. 741-744, January 2014.

[5]. K.Ashokkumar a, Baron Sam , R.Arshadprabhu , Britto. "Cloud Based Intelligent Transport System". Procedia Computer Science, volume 50, pp. 58-63, 2015.

[6]. Prof. D. J. Bonde , Rohit S. Shende, Ketan S. Gaikwad, Akshay S. Kedari,Amol U. Bhokre. “Automated Car Parking System Commanded by Android Application”, (IJCSIT) International Journal of Computer Science and Information Technologies, volume 5(3), pp. 1-4, 2014.

[7] M.Ataur Rehman, M.M. Rashid, A.Musa, A.Farhana and N.Farhana,“Automatic parking management and parking fee collection based On Number Plate Recognition”, International Journal of Machine Learning and Computing, vol. 2, no. 2, pp. 93-98, 2012.

[8] Patrick Sebastian, Hamada R.H. Al-Absi, Justin Dinesh Daniel Devraj and Yap Vooi Voon, “Vision based automated parking System”, 10th International conference on Information Science, Signal Processing and their Applications (ISSPA 2010), no. 1, pp. 757-760, 2010.

[9] Norazwinawati Basharuddin, R. Yusnita, Fariza Norbaya,“Intelligent Parking space detection system based on image Processing”, Internation Journal of Innovation, Management and Technology, vol. 3, no. 3, pp. 232-253, 2012.

[10] M.A.R. Sarkar, A.A. Rokoni, M.O. Reza, M.F. Ismail, “Smart Parking system with image processing facility”, I.J. Intelligent Systems and Applications, 2012, vol. 3, pp. 41-47.