**INTRODUCTION**

Parking facilities are a major expense to society and parking conflicts are among the most common problems facing infrastructure planners. These problems can be most often described either in terms of supply or in terms of management. Parking management describes the process of optimizing the use of parking policies while making use of policies and programs that are applicable to parking. A well-thought out parking strategy often helps reduce the number of parking spots required in a particular situation and provides a variety of socio-economic and environmental benefits. When all factors are taken into consideration, improved management is often the best solution to parking problems.

In metropolitan areas, parking management inﬂuences drivers search time and cost for parking spaces, parking revenue, and traffic congestion. The wide deployment of wireless parking meters with sensing and communications capabilities allows the parking authority to monitor the state of each parking space in real time and optimize the parking management.

To understand, what is Multi-Storey Car Park? A multi-storey car-park (also called as a parking garage, parking structure, parking ramp, parkade, parking building or parking deck) is a building designed for car parking and where there are a number of floors or levels on which parking takes place. It is essentially a stacked car park.

**Figure 1.1: Exterior Car Park Figure 1.2: Interior Car Park**

As we can see, Figure 1 shows how a Multistorey car park looks from outside and Figure 2 shows an interior view of these kind of buildings. There are multiple levels called storey meant for parking of vehicles (cars) of various shapes and sizes. The cars are kept in a way that resembles a stack.

Movement of vehicles between floors can be effected by:

* interior ramps - the most common type
* exterior ramps - which may take the form of a circular ramp (colloquially known as a 'whirley-gig' in America)
* vehicle lifts - the least common
* automated robot systems - combination of ramp and elevator

In India, the most common way of loading the vehicles is ‘Vehicle lifts’ as seen in figure 3. In this kind, the cars are taken to the desired level or floor by a lift meant for cars. The functioning is same as that of the lifts meant for carrying persons.



**Figure 1.3: Vehicle Lifts**

In this thesis, we study state-of-the-art parking policies in smart parking systems, and show that the smart parking system needs to be “smarter”. Our design goals of the smart parking systems include:

(1) simplify the operations of parking systems

(2) improve drivers’ satisfaction

(3) increase parking revenue

(4) alleviate trafﬁc congestion.

Through analysis and simulations, we ﬁrst show that the proposed reservation-based parking policy has the potential to achieve the above goals. We then model the behavior of both service providers and drivers in smart parking systems, and explore the dynamic pricing scheme to achieve the goals in smart parking system design.

Furthermore, we design and implement a prototype of Reservation-based Smart Parking System (RSPS) that allows drivers to effectively ﬁnd and reserve the vacant parking spaces.

**ParkMeRight:**

ParkMeRight is a project meant for advance booking of parking spaces by vehicle owners so as to reduce the uncertainty of getting an available space for parking when required. This project is being made keeping in mind the Multistorey car park buildings that are recently introduced in India and that too in few metro cities like Mumbai, Chennai and Pune. It will be used for reserving spaces in these buildings. The present daily requirements by people are being considered in this project.

The user, while enquiring, has to provide information related to the car like the name or the model so that they can be told whether there is any space left for the parking for their vehicle(depending on its size) in the building or not. If available, they can book it and tell the time duration for which it would be required (The process is similar to the booking of tickets for movies). The payment can be made well in advance or at the time of leaving. All the details of booking and payment will be communicated to the user time to time.

This project will not only help the user to secure a parking space for their vehicle and make the roads less crowded but will also lead to an increase in the use of the multistorey park buildings by spreading awareness among people since it is user friendly.

**LITERATURE REVIEW**

According to the With about 130 million vehicles on its roads and about 2.7 million cars being added each year, India's parking woes are only going to get worse, especially as there is no organized industry or policy framework in the country.

In fact, IBM's first-ever Parking Index, a ranking of the emotional and economic toll of parking in 20 international cities, released in 2011,put New Delhi and Bangalore as the worst. In India, we have very few multi-level parking lots. Generally, it's open parking lots that are provided by the local authorities. India also has one of the lowest public parking rates in the world.

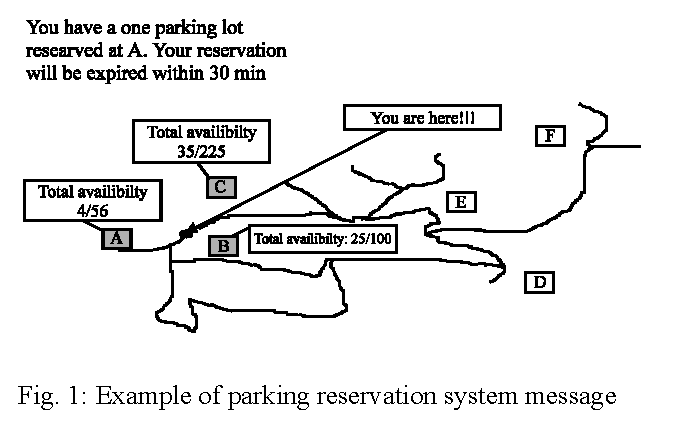
An advance booking online facility for parking space has been implemented in Chennai, India named uPark. Ramachandran, who is the company's chief executive, said an internal survey done by his company across eight Indian cities revealed that 93% of the respondents wanted to know parking availability in advance and did not mind paying for it. Car Owners are desperately looking for Parking Information and are often frustrated circling around the roads with no information on parking. Car owners often park haphazardly in the roads and grow tired of paying penalties and being towed.

The project team came across a research paper related to E-Parking System [1] while doing the survey. The study helped to find a proper solution to the mentioned traffic problems.

The smart parking system is considered beneficial for the car park operators, car park patrons as well as in environment conversation. For the car park operators, the information gathered via the implementation of the Smart parking system can be exploited to predict future parking patterns. Pricing strategies can also be manipulated according to the information obtained to increase the company’s profit. In terms of environment conservation, the level of pollution can be reduced by decreasing vehicle emission (air pollutant) in the air. This can be attributed to the fact that vehicle travel is reduced. As fuel consumption is directly related to vehicle miles travelled, it will be reduced as well.

**E-Parking System:**

It provides an alternative for patrons to enquire the availability and/or reserve a parking space at their desired parking facility to ensure the availability of vacant car park space when they arrive at the parking facility.System can be accessed via internet on WAP enabled Android smart phone.



**Figure 2.1: Example of parking system reservation message [1]**

The above figure, Figure 4, is an example of parking system reservation message which shows the number of available spaces out of the total spaces at different car parks. The message related to booked space is displayed to the user on their phones along with the current position and the nearest car park location.

But there are some limitations of implementing this system in India. The most important of them being, less number of multi-storey car park systems in India. So, smart parking systems can’t be implemented. Manual operation by an operator is required.

**Smart Payment System** **:**

The smart payment system is implemented in the effort to overcome the limitation of the conventional payment methods by revamping the method via parking meter and introduce new technologies. This is because the conventional methods cause delay and inconvenience. The smart payment system consists of contact method, contactless method and mobile devices. While the contact method involves the use of smart cards, debit cards and credit cards, the contactless method involves contactless cards, mobile devices as well as Automated Vehicle Identification tag whereby RFID technologies are utilized.

**METHODOLOGY**

**Introduction:**

Any software project follows the Software Development Life Cycle (SDLC). This final year project also used four major steps to implement project starting from requirement analysis, planning, implementing and testing. All the methods are used for finding and analyzing data regarding the project.

**Motivation:**

The main motivation of this application was to make a marketable product which when implemented in real world would have each and every characteristics of a usable and useful product. The task of engineer is to serve the society in the best way. This project of ours is an attempt made in that path. We feel motivated to find out or create those projects which would not only be profitable for us but is also beneficial for the society in some or the other way. Since, the parking woes is the prime problem faced by us in our fast day to day life, it came into our mind of at least reducing the overhead if not fully overcoming it. The new parking plaza named NIT Parking Plaza, Sitabuldi, Nagpur developed in 2012, was the reason this idea came in the mind.

**Requirement Analysis:**

Our project team visited the NIT Parking Plaza and analyzed some online booking web applications as to how they work and how the interface behaves. This helped the team to make the application more user friendly. The team had to think as the user as well as the stakeholder. All the functionalities were decided upon as to how and what facilities has to be provided. Keeping in mind the upsurge in use of smart phones used by people, the team decided to create an app that could be run on smart phones and tablets. This stage in project development focused on what actually is required by the user from any product and how to make it useful.

**Planning:**

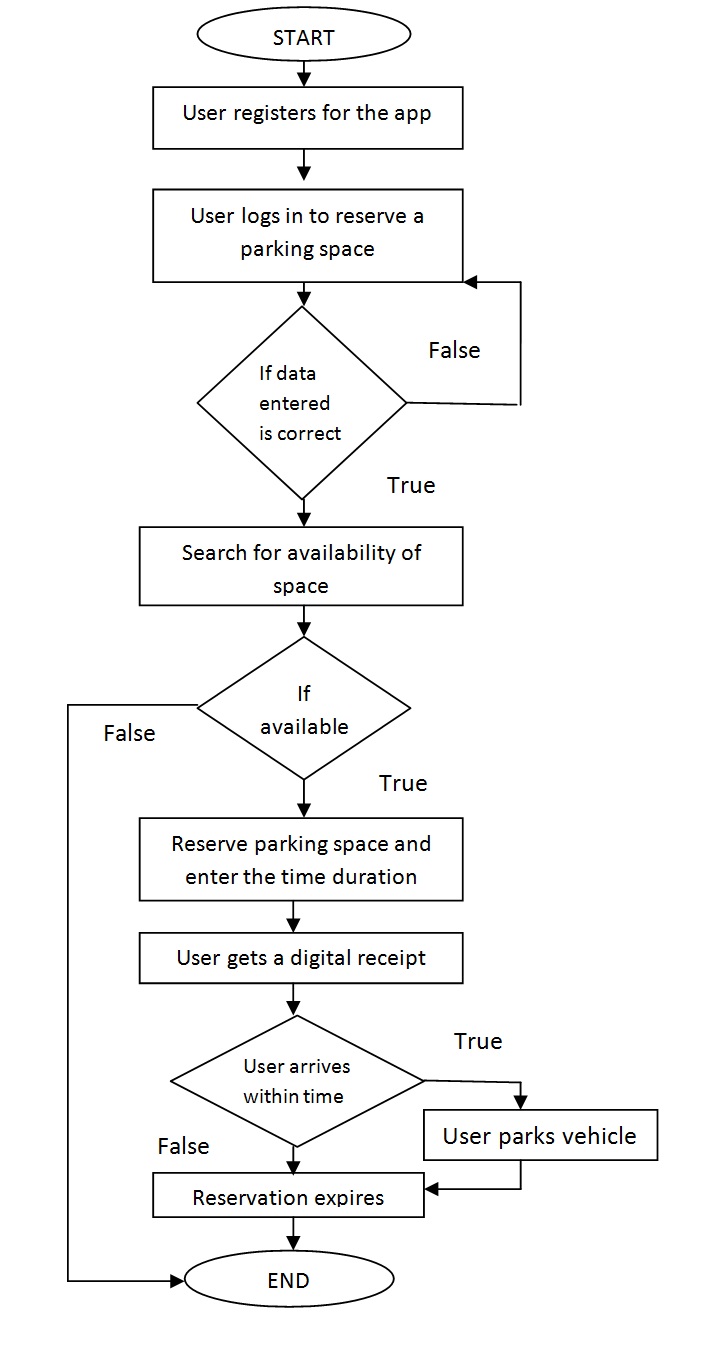
In order to identify all the information and requirement such as hardware and software, planning must be done in the proper manner. The planning phase has two main elements namely data collection and the requirements of hardware and software. This stage also required to develop a project plan as to how the team will proceed with the project and what all would be the milestones. The planning helped us to complete our project in the stipulated time and to deliver what was decided.

Given the design objectives of smart parking systems that require the coordination among multiple parties, we summarize the main design considerations as follows:

* Performance v.s. Overhead: In order to complete parking space reservation, drivers have to communicate with the system. However, there is a tradeoff between the overhead to generate and convey control signals to drivers v.s. performance optimality. For example, if the system computes the ﬁne-grained control signals for individual drivers and adopts unicast to inform the drivers about parking system status and instructions, the overhead will be very large. Hence, the parking management system generates a uniform control signal (e.g., the utilization of parking lot and parking price), and broadcasts the control signal to drivers.
* Trade-off Between Beneﬁts to Drivers and Service Providers: Multiple parties (drivers and service providers) are involved in the parking system operation. The state of the system depends on their interaction with each other. To balance the needs of involved parties, we use parking price as the control signal to coordinate the involved parties.
* Differentiated Service for Large Scale Autonomous Drivers: Thousands of drivers make parking decision autonomously. They have different needs and budgets for parking and their interpretation of parking information is different. Providing differentiated service for drivers is important to satisfy individual users. In this sense, the service quality is determined by parking price.

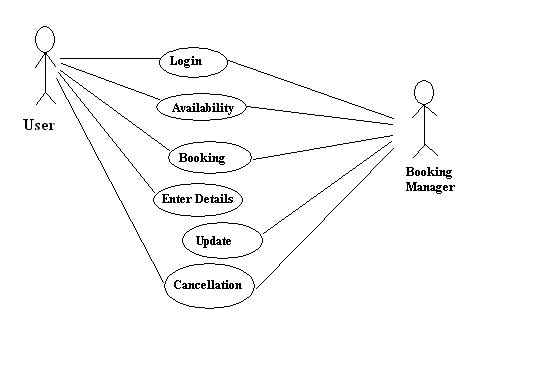
**Implementation:**

The building or vicinity or organization’s parking management maintains a website of their own that contains information related to parking spots available for vehicles of different sizes for a particular time on a given day. This website maintains a user database that contains the accounts of registered visitors and provision for adding new visitors. When the user wants to use the advance booking facility, they are prompted to download the mobile application on their cell phone using the Google play. They can then register themselves on the app for future use. Once the registration is done, the user is provided a unique username and password for logging into their account. Existing users can use their mobile app to book parking spaces remotely sitting anywhere prior to their visit. They have to specify the time of their visit, the duration for which they are expected to stay, and the size of the vehicle. The flowchart of the reservation system is as shown in the following figure:



**Figure 3.1: Flow Chart**

The team worked upon the use case diagram and on designing. Fig. below is the use case diagram.



**Figure 3.2: Use Case Diagram**

After the designing process, the focus shifted on coding and real implementation. The project was divided into different modules and each member was distributed with the even amount of work.

**Testing:**

As the modules started completing one by one, they were tested for any coding flaws such as syntax error or logical errors. Once a module was tested it was integrated with others to find how the integrated component reacts or performs. In this way, the whole project was completed without any major flaws or errors which would hamper its performance.

**DATA COLLECTION**

**Software:**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Requirements** | **Tools Used** |
| 1. | Operating System | Windows 7 |
| 2. | Language | Java |
| 3. | Database | MySQL |
| 4. | Tools | Eclipse IDE |
| 5. | Technologies | JSP, HTML, CSS, JQuery, Android |
| 6. | Server | Apache Tomcat Server |

**Table 4.1: Software Requirements**

**Hardware:**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Requirements** | **Hardware Used** |
| 1. | Processor | P IV or Higher |
| 2. | RAM | 256 MB |
| 3. | Space on Disk | Minimum 512 MB |

**Table 4.2: Hardware Requirements**

**Why Java Technology is used?**

Java is an object oriented, multi threading programming language developed by Sun Microsystems in 1991. This language was originally called OAK. It is designed to be small, simple and portable across different platforms as well as operating systems.The popularity of Java is due to its unique technology that is designed on the basis of three key elements. They are applets, powerful programming language constructs and a rich set of object classes. The easiest way to get a Java compiler runtime environment is Java Development Kit. This provides system input and output capabilities and other utility functions in addition to classes that support networking, common internet protocols and user interface toolkit functions.

**Features:**

* Platform Independence
* Object Oriented
* Compiler/Interpreter combination
* Robust

**Why Use MySQL?**

1. **Scalability and Flexibility**

The MySQL database server provides the ultimate in scalability, sporting the capacity to handle deeply embedded applications with a footprint of only 1MB to running massive data warehouses holding terabytes of information. Platform flexibility is a stalwart feature of MySQL with all flavors of Linux, UNIX, and Windows being supported. The open source nature of MySQL allows complete customization for those wanting to add unique requirements to the database server.

1. **High Performance**

A unique storage-engine architecture allows database professionals to configure the MySQL database server specifically for particular applications. With high-speed load utilities, distinctive memory caches, full text indexes, and other performance-enhancing mechanisms, MySQL offers all the right ammunition for today's critical business systems.

1. **High Availability**

MySQL offers a variety of high-availability options from high-speed master/slave replication configurations, to specialized Cluster servers offering instant failover, to third party vendors offering unique high-availability solutions for the MySQL database server.

1. **Robust Transactional Support**

MySQL offers one of the most powerful transactional database engines on the market. Features include complete ACID (atomic, consistent, isolated, durable) transaction support, unlimited row-level locking, distributed transaction capability, and multi-version transaction support where readers never block writers and vice-versa. Full data integrity is also assured through server-enforced referential integrity, specialized transaction isolation levels, and instant deadlock detection.

1. **Strong Data Protection**

Because guarding the data assets of corporations is the number one job of database professionals, MySQL offers exceptional security features that ensure absolute data protection. In terms of database authentication, MySQL provides powerful mechanisms for ensuring only authorized users have entry to the database server, with the ability to block users down to the client machine level being possible. SSH and SSL support are also provided to ensure safe and secure connections.

1. **Comprehensive Application Development**

One of the reasons MySQL is the world's most popular open source database is that it provides comprehensive support for every application development need. Within the database, support can be found for stored procedures, triggers, functions, views, cursors, ANSI-standard SQL, and more. For embedded applications, plug-in libraries are available to embed MySQL database support into nearly any application. MySQL also provides connectors and drivers (ODBC, JDBC, etc.) that allow all forms of applications to make use of MySQL as a preferred data management server. It doesn't matter if it's PHP, Perl, Java, Visual Basic, or .NET, MySQL offers application developers everything they need to be successful in building database-driven information systems.

**Java Server Pages**

Java Servlets and Java Server pages provide a secure, robust and platform independent technology for bringing the power of java to e-commerce and enterprise web computing. A Java Server Page is a template for a web page that uses java code to generate a HTML document dynamically. JSPs are run in a server side component known as JSP container; which translates them into equivalent Java Servlet. For this reason, JSP pages and Servlet are intimately related.

**Apache Tomcat Server**

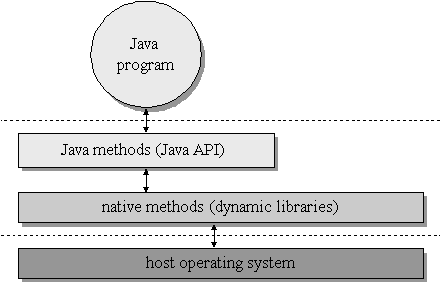
Apache tomcat server is a servlet container that is used in the official Reference implementation for the Java Servlet and Java Server Pages technologies. The Java Servlet and Java Server Pages are developed by sun under the java community process.

Apache Tomcat is developed in an open and participatory environment and released under the Apache Software License. Apache Tomcat is intended to be a collaboration of the best of breed developers from around the world. It powers numerous large scale, mission critical web applications across a diverse range of industries and organizations.

**What is Android?**

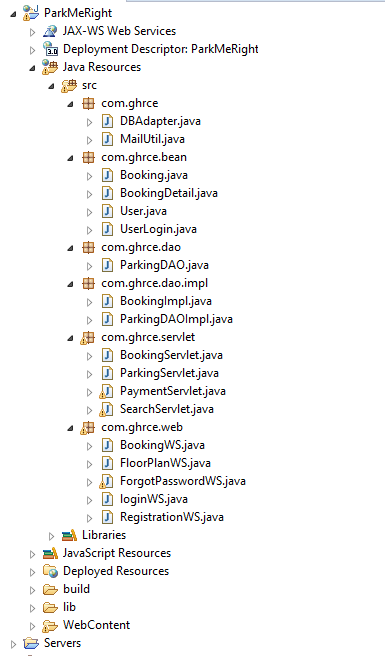
Android is a software stack for mobile device that includes an operating system, middleware and applications. Android is powered by Linux kernel, initially developed by Google and later the Open Handset Alliance. It allows developers to write managed code in java language, controlling the device via Google developed java library. Not like other famous rivals such as Microsoft window mobile or Symbian OS, android use developed java library because java is not just a programming language; it’s a complete dynamic platform offers powerful support for embedded devices that must maintain some form of dynamic behavior.

Moreover, java runtime environment can be integrated into almost any embedded device while java virtual machine includes interfaces that allow it to be readily integrated with RTOS and other native library. The RTOS supports multi-thread (scheduling), memory management, net working, and peripheral management for java VM.

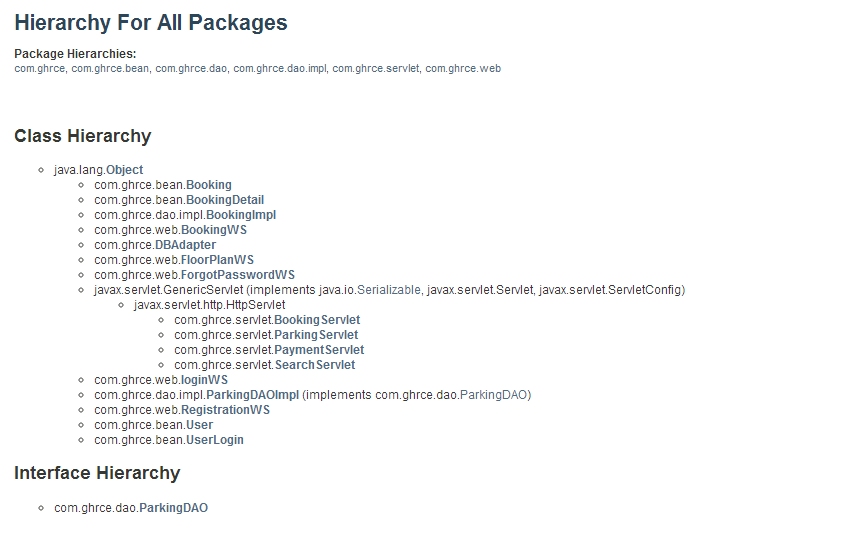


**Figure 4.1: Android Environment**

**IMPLEMENTATION**



**Fig 5.1: Project Structure**



**Fig 5.2 : Tree Hierarchy of Project**

**Database Connectivity**

For database connectivity, MySQL Connector is used. MySQL Connector/J is a native Java driver that converts JDBC (Java Database Connectivity) calls into the network protocol used by the MySQL database.Class.forName defines the type of database driver to be used. We need to provide the url of the mysql database location.

**DBAAdapter.java**

**public static** Connection getConnection(){

**if**(*connection* == **null**){

// This will load the MySQL driver, each DB has its own driver

**try** {

Class.*forName*("com.mysql.jdbc.Driver");

// Setup the connection with the DB

*connection* = DriverManager.*getConnection*(

"jdbc:mysql://localhost:3306/ParkMeRight","root", "admin");

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**return***connection*;

}

**Email Utility For Forgot Password**

JavaMail API is used for sending email to a recipient of a list of recipients using java. It provides a platform-independent and protocol-independent framework to build mail and messaging applications. JavaMail is a [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) [API](http://en.wikipedia.org/wiki/API) used to send and receive [email](http://en.wikipedia.org/wiki/Email) via [SMTP](http://en.wikipedia.org/wiki/SMTP), [POP3](http://en.wikipedia.org/wiki/POP3) and [IMAP](http://en.wikipedia.org/wiki/IMAP). JavaMail is built into the [Java EE](http://en.wikipedia.org/wiki/Java_EE) platform, but also provides an optional package for use in [Java SE](http://en.wikipedia.org/wiki/Java_SE).

**MailUtil.java**

**public boolean** sendMail(String[] recipients, String[] bccRecipients, String subject, String message) {

**try** {

Properties props = **new** Properties();

props.put("mail.smtp.host", SMTP\_HOST);

props.put("mail.smtp.auth", "true");

props.put("mail.debug", "false");

props.put("mail.smtp.ssl.enable", "true");

Session session = Session.*getInstance*(props, **new**SocialAuth());

Message msg = **new**MimeMessage(session);

InternetAddress from = **new**InternetAddress(FROM\_ADDRESS, FROM\_NAME);

msg.setFrom(from);

InternetAddress[] toAddresses = **new**InternetAddress[recipients.length];

**for** (**int**i = 0; i<recipients.length; i++) {

toAddresses[i] = **new**InternetAddress(recipients[i]);

}

msg.setRecipients(Message.RecipientType.*TO*, toAddresses);

InternetAddress[] bccAddresses = **new**InternetAddress[bccRecipients.length];

**for** (**int** j = 0; j <bccRecipients.length; j++)

{

bccAddresses[j] = **new**InternetAddress(bccRecipients[j]);

}

msg.setRecipients(Message.RecipientType.*BCC*, bccAddresses);

msg.setSubject(subject);

msg.setContent(message, "text/plain");

Transport.*send*(msg);

**return true**;

} **catch** (UnsupportedEncodingException ex) {

Logger.*getLogger*(MailUtil.**class**.getName()).log(Level.*SEVERE*, **null**, ex);

**return false**;

} **catch** (MessagingException ex) {

Logger.*getLogger*(MailUtil.**class**.getName()).log(Level.*SEVERE*, **null**, ex);

**return false**;

}

**Servlets**

**Booking Servlet:**

This servlet is called from Booking.jsp for fetching the number of slots for a particular floor. An ajax call is used to call this servlet.

**protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws**ServletException, IOException {

// **TODO** Auto-generated method stub

**try**{

String jsonData = request.getParameter("jsonData");

JSONObject js=**new** JSONObject(jsonData) ;

System.*out*.println(jsonData);

**int** slot=js.getInt("slotNo");

BookingImpl bk=**new** BookingImpl();

Booking booking = bk.getBookingDetails(slot);

Gson gson = **new** Gson();

String json = gson.toJson(booking);

System.*out*.println(json);

PrintWriter out = response.getWriter();

out.println(json);

}**catch** (JSONException | SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**Parking Servlet:**

This servlet is used to fetch the bookingid and the vehicle type corresponding to the slot selected. An ajax call is made to activate this servlet.

**protected void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws**ServletException, IOException {

// **TODO** Auto-generated method stub

ParkingDAOImpl dao = **new** ParkingDAOImpl();

String slotsJson = dao.getAllSlots();

System.*out*.println(slotsJson);

PrintWriter out = response.getWriter();

out.println(slotsJson);

}

**Payment Servlet:**

This servlet is used to do the final payment and slot booking of the selected slot. An ajax call is made from Booking.jsp page to activate this servlet.

// **TODO** Auto-generated method stub

**try**{

String jsondata = request.getParameter("jsondata");

JSONObject js=**new** JSONObject(jsondata) ;

Connection connection= (Connection) DBAdapter.*getConnection*();

Statement st= (Statement) connection.createStatement();

PreparedStatementps;

HttpSession ssn = request.getSession();

String name=ssn.getAttribute("userid").toString();

PrintWriter out = response.getWriter();

out.println(js);

**int** slot=(**int**) js.get("slot");

String sqlQuery="UPDATE floorplan SET `Available`='0' WHERE `Floor\_No`='1' and`Slot\_No`='"+slot+"'";

st.executeUpdate(sqlQuery);

DateFormat df=**new** SimpleDateFormat("dd/mm/yyyy");

Date d=**new** Date();

Calendar c=Calendar.*getInstance*();

ps=connection.prepareStatement("insert into booking values(?,?,?,?,?,?,?,?,?,?,?)");

System.*out*.println(name);

ps.setInt(1,(**int**) js.get("bookingId"));

ps.setString(2, name);

ps.setInt(3, 1);

ps.setInt(4,(**int**) js.get("slot"));

java.sql.DatesqlDate = **new** java.sql.Date(d.getTime());

ps.setDate(5, sqlDate );

**if**(js.get("vehicle").equals("SUV"))

ps.setObject(7, 'M', java.sql.Types.*CHAR*);

**else**

ps.setObject(7, 'S', java.sql.Types.*CHAR*);

ps.setString(8,(String) js.get("vehicleno"));

java.sql.TimesqlTime = **new** java.sql.Time(c.getTime().getTime());

ps.setTime(9, sqlTime);

ps.setTime(6,sqlTime);

ps.setInt(10, Integer.*parseInt*(js.get("duration").toString()));

ps.setFloat(11, (**float**) 0.0);

ps.execute();

}**catch** (JSONException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**Search Servlet:**

It is a servlet used to implement customer search capabilities. A call is made to this servlet from Booking.jsp via form.

**protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws**ServletException, IOException {

// **TODO** Auto-generated method stub

**try** {

Connection connection= (Connection) DBAdapter.*getConnection*();

Statement stmt = **null**;

ResultSetrs;

String email=request.getParameter("search");

String sql = "SELECT \* FROM booking WHERE Email='"+email+"'";

response.setContentType("text/html");

PrintWriter out = response.getWriter();

rs=(ResultSet) stmt.executeQuery(sql);

out.println("<div id='search'><section class='container'> <h2>Customer Search</h2><table class='order-table table'>");

out.println("<thead>");

out.println("<tr>");

out.println("<th>Booking-ID</th>");

out.println("<th>User\_ID</th>");

out.println("<th>Floor\_No</th>");

out.println("<th>Slot\_No</th>");

out.println("<th>Date</th>");

out.println("<th>Booking\_Time</th>");

out.println("<th>Vehicle\_Type</th>");

out.println("<th>Vehicle\_No</th>");

out.println("<th>Entry\_Time</th>");

out.println("<th>Duration</th>");

out.println("<th>Extension</th>");

out.println("</tr>");

out.println("</thead>");

out.println("<tbody>");

**if**(rs.next())

{

out.println("<tr>");

out.println("<td>"+rs.getString(1)+"</td>");

out.println("<td>"+rs.getString(2)+"</td>");

out.println("<td>"+rs.getString(3)+"</td>");

out.println("<td>"+ rs.getString(4)+"</td>");

out.println("<td> "+rs.getString(5)+"</td>");

out.println("<td> "+rs.getString(6)+"</td>");

out.println("<td>"+rs.getString(7)+"</td>");

out.println("<td> "+rs.getString(8)+"</td>");

out.println("<td> "+ rs.getString(9)+"</td>");

out.println("<td> "+ rs.getString(10)+"</td>");

out.println("<td> "+rs.getString(11)+"</td>");

out.println("</tr>");

}

**else**

{

out.println("No such booking exists");

}

out.println("</tbody>");

out.println("</table>");

out.println("</section>");

out.println("</div>");

out.println("<a href='Booking.jsp'>Back</a>");

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**Web Services:**

RESTEasy is a JBoss project that provides various frameworks to help you build RESTful Web Services and RESTful Java applications. It is a fully certified and portable implementation of the JAX-RS specification. JAX-RS is a new JCP specification that provides a Java API for RESTful Web Services over the HTTP protocol. Various Classes used are:

**BookingWS.java:**

*Url for the service:* http: //localhost:8080/WebService/getdetails/post

@POST

@Path("/post")

@Consumes("application/json")

**public** Response bookSlot(BookingDetailbt) **throws**SQLException

{

Connection connection;

Statement st;

connection=(Connection) DBAdapter.*getConnection*();

st=connection.createStatement();

**int**val = st.executeUpdate("insert into booking values

('"+bt.getBookingid()+"','"+bt.getUserid()+"'"+",'"+bt.getFloorno()

+"','"+bt.getSlotno()+"','"+bt.getDate()+"','"+bt.getBookingtime()

+"','"+bt.getType()+"','"+bt.getVehicleno()+"'"+",'"+bt.getEntrytime()

+"','"+bt.getDuration()+"','"+bt.getExtension()+"')");

st.close();

connection.close();

**if**(val==1){

**return** Response.*status*(201).entity("true").build();

}

**else**

{

**return** Response.*status*(201).entity("false").build(); }

}

**FloorPlanWS:**

*Url for the service:* http: //localhost:8080/WebService /floor/get

@GET

@Path("/get")

@Produces("application/json")

**public** String displayFloorPlan(String json)

{

String slotsJson;

ParkingDAOImpldao = **new**ParkingDAOImpl();

slotsJson = dao.getAllSlots();

**return** slotsJson;

}

**ForgotPasswordWS:**

*url:* http://localhost:8080/WebService /forgotpass/post

@GET

@Path("/{param}")

**public** Response emailCheck(@PathParam("param") String email)

{

Connection connection;

Statement st;

ResultSetrs;

**try**{

connection=DBAdapter.*getConnection*();

st=connection.createStatement();

rs=(ResultSet) st.executeQuery("select UserId from user\_info

where Email='"+email+"'");

**if**(rs.next())

{

rs.close();

st.close();

connection.close();

**return** Response.*status*(201).entity("true").build();

}

**else**

{

rs.close();

st.close();

connection.close();

**return** Response.*status*(401).entity("false").build();

}

}**catch** (SQLException e) {

// **TODO** Auto-generated catch block

((Throwable) e).printStackTrace();

}

**return null;**

}

**LoginWS:**

*url for the service:* http://localhost:8080/WebService/login /post

@POST

@Path("/post")

@Consumes("application/json")

**public** Response checkValidity(UserLoginul)

{

Connection connection;

Statement st;

ResultSetrs;

**try**{

connection=DBAdapter.*getConnection*();

st=connection.createStatement();

rs=(ResultSet) st.executeQuery("select userid from user

where Username='"+ul.getUsername()+"' ANDPassword='"+

ul.getPassword()+"'");

**if**(rs.next())

{

rs.close();

st.close();

connection.close();

**return** Response.*status*(201).entity("true").build();

}

**else**

{

rs.close();

st.close();

connection.close();

**return** Response.*status*(401).entity("false").build();

}

}**catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return null**;

}

@GET

@Path("/get")

@Produces("application/json")

**public**UserLogingetProductInJSON() {

System.*out*.println("get request called");

UserLoginuserLogin = **new**UserLogin();

userLogin.setPassword("abc");

userLogin.setUsername("xyz");

**return** userLogin; }

**RegistrationWS:**

*url for the service:* http://localhost:8080/WebService /register/post

@POST

@Path("/post")

@Consumes("application/json")

Connection connection;

Statement st;

ResultSetrs;

PreparedStatementpreparedStatement;

**int**val;

**try**{

String fname=user.getFname();

String lname=user.getLname();

String email=user.getEmail();

String mobile=user.getMobile();

String pass=user.getPassword();

connection=DBAdapter.*getConnection*();

st=connection.createStatement();

val=st.executeUpdate("insert into user values('"+email+"',

'"+pass+"')");

**if**(val==1)

{

rs=(ResultSet) st.executeQuery("select userid from user where Username='"+email+"' AND Password='"+pass+"'");

rs.next();

String userid=rs.getString("UserId");

String insertTableSQL = "INSERT INTO user\_info"

+ "(UserId, Fname, Lname, Phone,Email) VALUES"

+ "(?,?,?,?,?)";

preparedStatement = (PreparedStatement) connection.prepareStatement(insertTableSQL);

preparedStatement.setString(1, userid);

preparedStatement.setString(2, fname);

preparedStatement.setString(3, lname);

preparedStatement.setString(4, mobile);

preparedStatement.setString(5, email);

// execute insert SQL stetement

val=preparedStatement .executeUpdate();

rs.close();

st.close();

connection.close();

**if**(val==1)

**return** Response.*status*(201).entity("true").build();

**else**

**return** Response.*status*(401).entity("false").build();

}

**else**

{

**return** Response.*status*(401).entity("false").build();

}

}**catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return null**;

}

**Android App**

**MainActivity.java:**

By default, the android application includes an MainActivity.java source file having an activity class that runs when an app is launched using the app icon. This is the actual application file which ultimately gets converted to a Dalvik executable and runs the application.

public class MainActivity extends Activity implements RequestListener,

OnClickListener {

private Spinner spinner1, spinner2;

private Button button;

private String mFloorNo;

private String mSlotNo;

private String mBookingId;

private String mVehicleNo;

private String mEntryTime;

private String mDurationTime;

private String mVehType;

String selectedSlot;

private int mFlag;

List<String> slotAvailable = new ArrayList<String>();

// widgets instance

private EditText mBookingIdText;

private EditText mVehicleNoText;

private EditText mEntryTimeText;

private EditText mDurationTimeText;

private EditText mVehicleType;

private String userId;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

spinner1 = (Spinner) findViewById(R.id.spinner1);

spinner2 = (Spinner) findViewById(R.id.spinner2);

button = (Button) findViewById(R.id.button);

Bundle extras = getIntent().getExtras();

if (extras != null) {

userId = extras.getString("userId");

}

mBookingIdText = (EditText) findViewById(R.id.editText1);

mVehicleNoText = (EditText) findViewById(R.id.editText2);

mEntryTimeText = (EditText) findViewById(R.id.editText3);

mDurationTimeText = (EditText) findViewById(R.id.editText4);

mVehicleType = (EditText) findViewById(R.id.type\_text);

spinner2 = (Spinner) findViewById(R.id.spinner2);

spinner1 = (Spinner) findViewById(R.id.spinner1);

spinner1.setOnItemSelectedListener(spinner1Listener);

button.setOnClickListener(this);

}

OnItemSelectedListener spinner1Listener = new OnItemSelectedListener() {

@Override

public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {

if (spinner1.getSelectedItem().equals("Select Floor")) {

} else {

new SelectFloor().execute();

}

}

@Override

public void onNothingSelected(AdapterView<?> parent) {

}

};

// add items into spinner dynamically

@Override

public void onSuccess(JSONObject json) {

}

@Override

public void onFailure(int errorCode) {

Log.d("test", "" + errorCode);

}

public class SelectFloor extends AsyncTask<String, String, String> {

@Override

protected void onPostExecute(String result) {

super.onPostExecute(result);

ArrayAdapter dataAdapter = new ArrayAdapter(MainActivity.this, android.R.layout.simple\_spinner\_item, slotAvailable);

dataAdapter.setDropDownViewResource(android.R.layout.simple\_spinner\_dropdown\_item);

spinner2.setAdapter(dataAdapter);

spinner2.setOnItemSelectedListener(new OnItemSelectedListener() {

@Override

public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {

selectedSlot = (String) spinner2.getSelectedItem();

new SlotNo().execute();

}

@Override

public void onNothingSelected(AdapterView<?> parent) {

}

});

}

@Override

protected String doInBackground(String... params) {

// TODO Auto-generated method stub

HttpClient httpclient;

HttpParams httpParameters = new BasicHttpParams();

int timeoutConnection = 3000;

int timeoutSocket = 5000;

HttpConnectionParams.setSoTimeout(httpParameters,timeoutSocket);

httpclient = new DefaultHttpClient(httpParameters);

HttpResponse response;

try {

HttpGet httpget = new HttpGet( "http://10.0.2.2:8080/WebService/floor/get?floorNo=1");

response = httpclient.execute(httpget);

System.out.println("Inside PostData");

HttpEntity entity = response.getEntity();

Log.d("entity", "" + response.getEntity());

Log.d("params", "" + response.getParams());

Log.d("Stutus Line", "" + response.getStatusLine());

String data = EntityUtils.toString(entity);

Log.d("DATA : ", data);

JSONArray array = new JSONArray(data);

Log.d("Lenght", "" + array.length());

String[] available = new String[array.length()];

for (int i = 0; i < array.length(); i++) {

JSONObject json = array.getJSONObject(i);

Log.d("floorNo", json.getString("floorNo"));

Log.d("available", json.getString("available"));

Log.d("slotNo", json.getString("slotNo"));

available[i] = json.getString("available");

if (!available[i].equals("0")) {

slotAvailable.add(json.getString("slotNo"));

}

}

} catch (ClientProtocolException e4) {

e4.printStackTrace();

} catch (IOException e4) {

e4.printStackTrace();

} catch (ParseException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (JSONException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return null;

}

}

public class SlotNo extends AsyncTask<String, String, String> {

@Override

protected void onPostExecute(String result) {

super.onPostExecute(result);

mBookingIdText.setText(mBookingId);

mVehicleType.setText(mVehType);

mBookingIdText.setEnabled(false);

mVehicleType.setEnabled(false);

}

@Override

protected String doInBackground(String... params) {

HttpClient httpclient;

HttpParams httpParameters = new BasicHttpParams();

int timeoutConnection = 3000;

int timeoutSocket = 5000;

HttpConnectionParams.setSoTimeout(httpParameters,timeoutSocket);

httpclient = new DefaultHttpClient(httpParameters);

HttpPost httppost = new HttpPost(

"http://10.0.2.2:8080/WebService/getdetails/post");

System.out.println("Inside PostData");

JSONObject json = new JSONObject();

try {

json.put("slotno", selectedSlot);

StringEntity se = new StringEntity(json.toString());

se.setContentType(newBasicHeader(HTTP.CONTENT\_TYPE,"application/json"));

httppost.setEntity(se);

} catch (JSONException e) {

e.printStackTrace();

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

HttpResponse response;

try {

response = httpclient.execute(httppost);

HttpEntity entity = response.getEntity();

Log.d("Spinner 2", "" + mFlag);

Log.d("entity", "" + response.getEntity());

Log.d("params", "" + response.getParams());

String data = EntityUtils.toString(entity);

Log.d("Slot Data : ", data);

JSONObject jsonSlot = new JSONObject(data);

mBookingId = jsonSlot.getString("bookingId");

mVehType = jsonSlot.getString("vehicle");

} catch (ClientProtocolException e4) {

e4.printStackTrace();

} catch (IOException e4) {

e4.printStackTrace();

} catch (JSONException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return null;

}

}

public class BookSlots extends AsyncTask<String, String, String> {

private String bookSuccess;

@Override

protected void onPostExecute(String result) {

super.onPostExecute(result);

if (bookSuccess == null) {

return;

}

if (bookSuccess.equals("true")) {

Toast.makeText(getApplicationContext(),

"Booking successfully done.",Toast.LENGTH\_SHORT).show();

MainActivity.this.finish();

} else {

Toast.makeText(getApplicationContext(),

"Check booking input.", Toast.LENGTH\_SHORT).show();

}

}

@Override

protected String doInBackground(String... params) {

HttpClient httpclient;

HttpParams httpParameters = new BasicHttpParams();

int timeoutConnection = 3000;

int timeoutSocket = 5000;

HttpConnectionParams.setSoTimeout(httpParameters,timeoutSocket);

httpclient = new DefaultHttpClient(httpParameters);

HttpPost httppost = new HttpPost(

"http://10.0.2.2:8080/WebService/booking/post");

System.out.println("Inside PostData");

JSONObject json = new JSONObject();

Calendar cal = Calendar.getInstance();

cal.add(Calendar.DATE, 1);

SimpleDateFormat format1 = new SimpleDateFormat("yyyy-MM-dd");

String formatted = format1.format(cal.getTime());

cal.getTime();

SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");

System.out.println(sdf.format(cal.getTime()));

try {

json.put("bookingid", mBookingId);

json.put("userid", userId);

json.put("floorno", "" + 1);

json.put("slotno", selectedSlot);

json.put("date", formatted);

json.put("bookingtime", "02:50:00");

json.put("type", "S");

json.put("vehicleno", mVehicleNo);

json.put("entrytime", mEntryTime);

json.put("duration", mDurationTime);

json.put("extension", "1");

StringEntity se = new StringEntity(json.toString());

se.setContentType(newBasicHeader(HTTP.CONTENT\_TYPE,"application/json"));

httppost.setEntity(se);

} catch (JSONException e) {

e.printStackTrace();

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

HttpResponse response;

try {

response = httpclient.execute(httppost);

HttpEntity entity = response.getEntity();

Log.d("entity", "" + response.getEntity());

Log.d("params", "" + response.getParams());

bookSuccess = EntityUtils.toString(entity);

Log.d("Slot Data : ", bookSuccess);

} catch (ClientProtocolException e4) {

e4.printStackTrace();

} catch (IOException e4) {

e4.printStackTrace();

}

return null;

}

}

@Override

public void onClick(View v) {

switch (v.getId()) {

case R.id.button:

mFloorNo = String.valueOf(spinner1.getSelectedItem());

mSlotNo = String.valueOf(spinner2.getSelectedItem());

mVehicleNo = mVehicleNoText.getText().toString();

mBookingId = mBookingIdText.getText().toString();

mEntryTime = mEntryTimeText.getText().toString();

mDurationTime = mDurationTimeText.getText().toString();

mVehType = mVehicleType.getText().toString();

new BookSlots().execute();

break;

default:

break;

}

}

}

**UserRegistrationActivity.java**

public class UserRegistrationActivity extends Activity implements

RequestListener {

private String mUserName;

private String mPassword;

private String mMobileNo;

private String mEmail;

private String mFirstName;

private String mLastName;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.user\_registration);

}

@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;

}

public void onClick11(View view) {

switch (view.getId()) {

case R.id.button1:

mUserName = ((EditText) findViewById(R.id.editText2)).getText()

.toString();

mPassword = ((EditText) findViewById(R.id.editText5)).getText()

.toString();

mMobileNo = ((EditText) findViewById(R.id.editText4)).getText()

.toString();

mEmail = ((EditText) findViewById(R.id.editText3)).getText()

.toString();

mFirstName = ((EditText) findViewById(R.id.editText1)).getText()

.toString();

mLastName = ((EditText) findViewById(R.id.EditText01)).getText()

.toString();

String reenter\_password = ((EditText) findViewById(R.id.EditText02)) .getText() .toString();

new RegistrationPage().execute();

}

}

public void reset(View view) {

System.out.println(view.getId());

System.out.println(view.getId() == R.id.button2);

switch (view.getId()) {

case R.id.button2:

Log.d("msg", "Inside click method");

Intent intent = new Intent(this, UserRegistrationActivity.class);

startActivity(intent);

break;

}

}

// get data from the server

@Override

public void onSuccess(JSONObject json) {

}

@Override

public void onFailure(int errorCode) {

Log.d("Error code", "" + errorCode);

}

public class RegistrationPage extends AsyncTask<String, String, String> {

private String register;

@Override

protected void onPostExecute(String result) {

super.onPostExecute(result);

if (register == null) {

return;

}

if (register.equals("true")) {

Toast.makeText(getApplicationContext(),

"User regitered successfully.", Toast.LENGTH\_SHORT).show();

UserRegistrationActivity.this.finish();

} else {

Toast.makeText(getApplicationContext(),

"Registered user.",Toast.LENGTH\_SHORT).show();

}

}

@Override

protected String doInBackground(String... params) {

HttpClient httpclient;

HttpParams httpParameters = new BasicHttpParams();

int timeoutConnection = 3000;

int timeoutSocket = 5000;

HttpConnectionParams.setSoTimeout(httpParameters,timeoutSocket);

httpclient = new DefaultHttpClient(httpParameters);

HttpPost httppost = new HttpPost(

"http://10.0.2.2:8080/WebService/register/post");

System.out.println("Inside PostData");

JSONObject json = new JSONObject();

try {

json.put("fname", mFirstName);

json.put("lname", mLastName);

json.put("email", mEmail);

json.put("mobile", mMobileNo);

json.put("password", mPassword);

StringEntity se = new StringEntity(json.toString());

se.setContentType(newBasicHeader(HTTP.CONTENT\_TYPE,"application/json"));

httppost.setEntity(se);

} catch (JSONException e) {

e.printStackTrace();

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

HttpResponse response;

try {

response = httpclient.execute(httppost);

HttpEntity entity = response.getEntity();

Log.d("entity", "" + response.getEntity());

Log.d("params", "" + response.getParams());

Log.d("Stutus Line :", response.getStatusLine().toString());

register = EntityUtils.toString(entity);

Log.d("Slot Data : ", register);

} catch (ClientProtocolException e4) {

e4.printStackTrace();

} catch (IOException e4) {

e4.printStackTrace();

}

return null;

}

}

}

**LoginActivity.java**

public class LoginActivity extends Activity implements RequestListener {

private static final String USER\_NAME = "vivek";

private static final String PASSWORD = "parking";

static JSONObject jObj = null;

static String json = null;

String username;

String password;

private String mLoginValid;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.login);

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

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

return true;

}

public void onClick11(View view) {

Intent intent = null;

switch (view.getId()) {

case R.id.button1:

username = ((EditText) findViewById(R.id.editText2)).getText()

.toString();

password = ((EditText) findViewById(R.id.editText1)).getText()

.toString();

if (username.length() != 0 && password.length() != 0) {

new LoginCheck().execute();

} else {

Toast.makeText(getApplicationContext(),

"Username and Password should not be empty.",Toast.LENGTH\_LONG).show();

}

break;

}

}

public void onClick(View view) {

System.out.println(view.getId());

System.out.println(view.getId() == R.id.signup\_button);

switch (view.getId()) {

case R.id.signup\_button:

Log.d("msg", "Inside click method");

Intent intent = new Intent(this, UserRegistrationActivity.class);

startActivity(intent);

break;

}

}

public void onClick1(View view1) {

System.out.println(view1.getId());

System.out.println(view1.getId() == R.id.forgotpassword\_button);

switch (view1.getId()) {

case R.id.forgotpassword\_button:

Log.d("msg", "Inside click method");

Intent intent = new Intent(this, ForgotPasswordActivity.class);

startActivity(intent);

break;

}

}

// get response from server and data in json format.

@Override

public void onSuccess(JSONObject json) {

}

@Override

public void onFailure(int errorCode) {

Log.d("error code :", "" + errorCode);

}

public class LoginCheck extends AsyncTask<String, String, String> {

@Override

protected void onPostExecute(String result) {

super.onPostExecute(result);

if (mLoginValid == null) {

return;

}

if (mLoginValid.equals("true")) {

Intent intent = new Intent(LoginActivity.this,MainActivity.class);

intent.putExtra("userId", username);

startActivity(intent);

} else {

Toast.makeText(getApplicationContext(),

"Invalid user.", Toast.LENGTH\_SHORT).show();

}

}

@Override

protected String doInBackground(String... params) {

HttpClient httpclient;

HttpParams httpParameters = new BasicHttpParams();

int timeoutConnection = 3000;

int timeoutSocket = 5000;

HttpConnectionParams.setSoTimeout(httpParameters,timeoutSocket);

httpclient = new DefaultHttpClient(httpParameters);

HttpPost httppost = new HttpPost(

"http://10.0.2.2:8080/WebService/login/post");

System.out.println("Inside PostData");

JSONObject json = new JSONObject();

try {

json.put("username", username);

json.put("password", password);

StringEntity se = new StringEntity(json.toString());

se.setContentType(newBasicHeader(HTTP.CONTENT\_TYPE,"application/json"));

httppost.setEntity(se);

} catch (JSONException e) {

e.printStackTrace();

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

HttpResponse response;

try {

response = httpclient.execute(httppost);

HttpEntity entity = response.getEntity();

Log.d("entity", "" + response.getEntity());

Log.d("params", "" + response.getParams());

mLoginValid = EntityUtils.toString(entity);

Log.d("responseText", mLoginValid);

} catch (ClientProtocolException e4) {

e4.printStackTrace();

} catch (IOException e4) {

e4.printStackTrace();

}

return null;

}

}

}

**AndroidManifest.xml**

Whatever component is developed as a part of the application, all its components must be declared in a manifest file called AndroidManifest.xml which resides at the root of the application project directory. This file works as an interface between Android OS and the application, so if the component is not declared in this file, then it will not be considered by the OS.

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.ghrce.activity"

android:versionCode="1"

android:versionName="1.0" >

<uses-sdk

android:minSdkVersion="8"

android:targetSdkVersion="18" />

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />

<application

android:allowBackup="true"

android:icon="@drawable/ic\_launcher"

android:label="@string/app\_name"

android:theme="@style/AppTheme" >

<activity

android:name="com.ghrce.activity.LoginActivity"

android:label="@string/app\_name" >

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

<activity

android:name=".UserRegistrationActivity"

android:label="@string/app\_name" >

</activity>

<activity

android:name=".ForgotPasswordActivity"

android:label="@string/app\_name" >

</activity>

<activity

android:name=".MainActivity"

android:label="@string/app\_name" >

</activity>

</application>

</manifest>

**TESTING AND SUMMARY OF RESULTS**

Software testing can be stated as the process of validating and verifying that a computer program/application/product:

* meets the requirements that guided its design and development,
* works as expected,
* can be implemented with the same characteristics,
* and satisfies the needs of stakeholders.

Software testing, depending on the testing method employed, can be implemented at any time in the software development process. A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions.

**White-box testing**

It tests internal structures or workings of a program. While white-box testing can be applied at the unit, integration and system levels of the software testing process, it is usually done at the unit level.

When a particular module was completed it was tested to find out any kind of structural error in the code such as syntax error or logical error or any error in path coverage and decision making points.

**Black-box testing**

It treats the software as a "black box", examining functionality without any knowledge of internal implementation. The tester is only aware of what the software is supposed to do, not how it does it.

After completion of each module, the functionality it was supposed to perform was checked on and if found correct, it was integrated further with other components.

### Unit testing

This testing was used by us in order to test individual modules, so that the module which is to be integrated should be error free as far as possible. For this purpose, a dummy module was created to be involved in testing process.

**Integration Testing**

It is the testing of combined parts of our application to determine if they function together correctly. The ‘parts’ were code module, individual applications, client and server applications on a network, etc.

**Functional Testing**

Functional testing refers to activities that verify a specific action or function of the code. To check whether the project is performing its pre-determined functions correctly or are the functions performed correctly, this testing is done.

**System Testing**

System testing tests a completely integrated system to verify that it meets its requirements. In other words, this testing was done to verify if the project as a whole functions as defined and does not interrupt the other applications running on the system.

**Acceptance Testing**

This type of testing is done once the project is completed and is ready for delivery. It is done by the real time user to check if there is any difference in its usability from the expected behavior. The team worked with the project from the user point of view and came out with some flaws which the user might face.

**Usability testing**

Usability testing is needed to check if the user interface is easy to use and understand. It is concerned mainly with the use of the application. The project was tested for user friendliness.

**Summary of Results**

After performing all the above mentioned testing approaches and processes, the team came out with the following results and conclusion.

* The project was completed within the stipulated time and satisfied all the requirements.
* The project was found to be parallel with the pre mentioned requirements.
* There were some major as well as minor flaws which were removed during the course.
* ParkMeRight will be an effective parking system keeping in mind the present scenario of India, where the parking systems are not so smart i.e. they are not based on wireless sensor networks and are not fully automatic.
* Currently few smart parking systems have come up but they are limited to metro living crowd.
* This project will remove all backdrops and will work towards solving the problems faced by vehicle owners.
* The ParkMeRight makes it simpler to book the parking space in advance.
* Potential Market
  + Shopping Malls
  + Hotels
  + Hospitals
  + Railway Stations
  + Airports
  + Historical/Tourists Places

Some new innovations or enhancements can be done in this project if taken further in future which would make this project high in market potential. Those features can be considered in the future scope of this project as provided below.

* The app can be made as “One-for-all” which would let any user to book any available slot of any parking building nearer to them using the GPS system.
* There would be no use of any manual operator handling the basic details. The system would be connected to some sensors that would feed data about availability of slots.
* The feature of online transaction can be implemented for ease of use and advance payment.

**REFERENCES**

**[1] Bong, D.B.L. , K.C. Ting and K.C. Lai, “***Integrated approach in the design of car-park occupancy information system***”, IAENG Int. J. Comput. Sci., 35:1-8, 2008.**

**[2] Daniele Luttazzi**, “*Lepidezze postribolari”* (2007, Feltrinelli, p.275)

**[3] Hinze, D., 2000. “***Italians drive for smart card parking in major cities***”. Card Technol. Today, 2: 6-7**

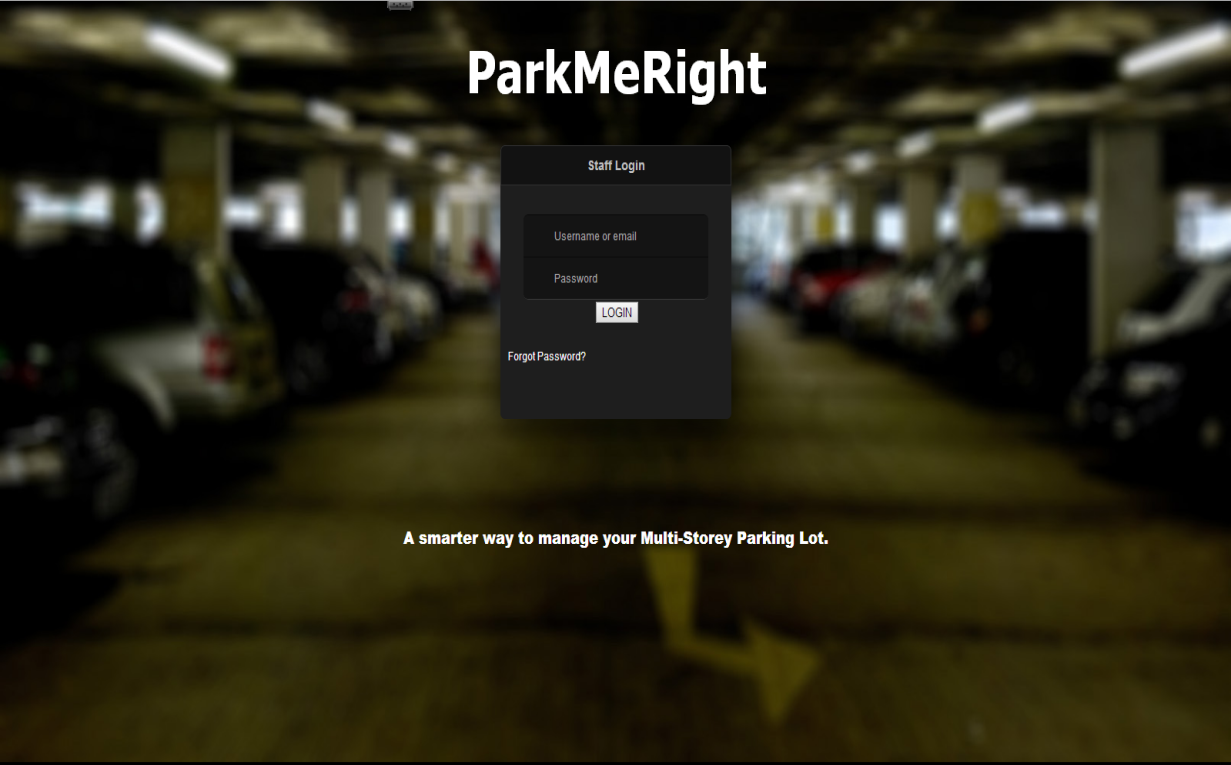
**[4] Idna, M.Y. and E.M. Tamil, 2007. “***Parking Information System Using GPS and Shortest Path Algorithm***”. Proceeding s of the SCOREED 2007, MAY 14-15, University Tenaga Nasional, Malaysia, pp: 1-7**

**[5] Inaba, K. , M. Shibui, T. Naganawa, M.Ogiwara and N.Yoshikai, 2001. “***Intelligent Parking Reservation service on the internet***”. SAINT W, Symposium on applications and the internet workshops (SAINT 2001 Workshops), Jan 8-12, San diego, CA USA,pp: 159-164**

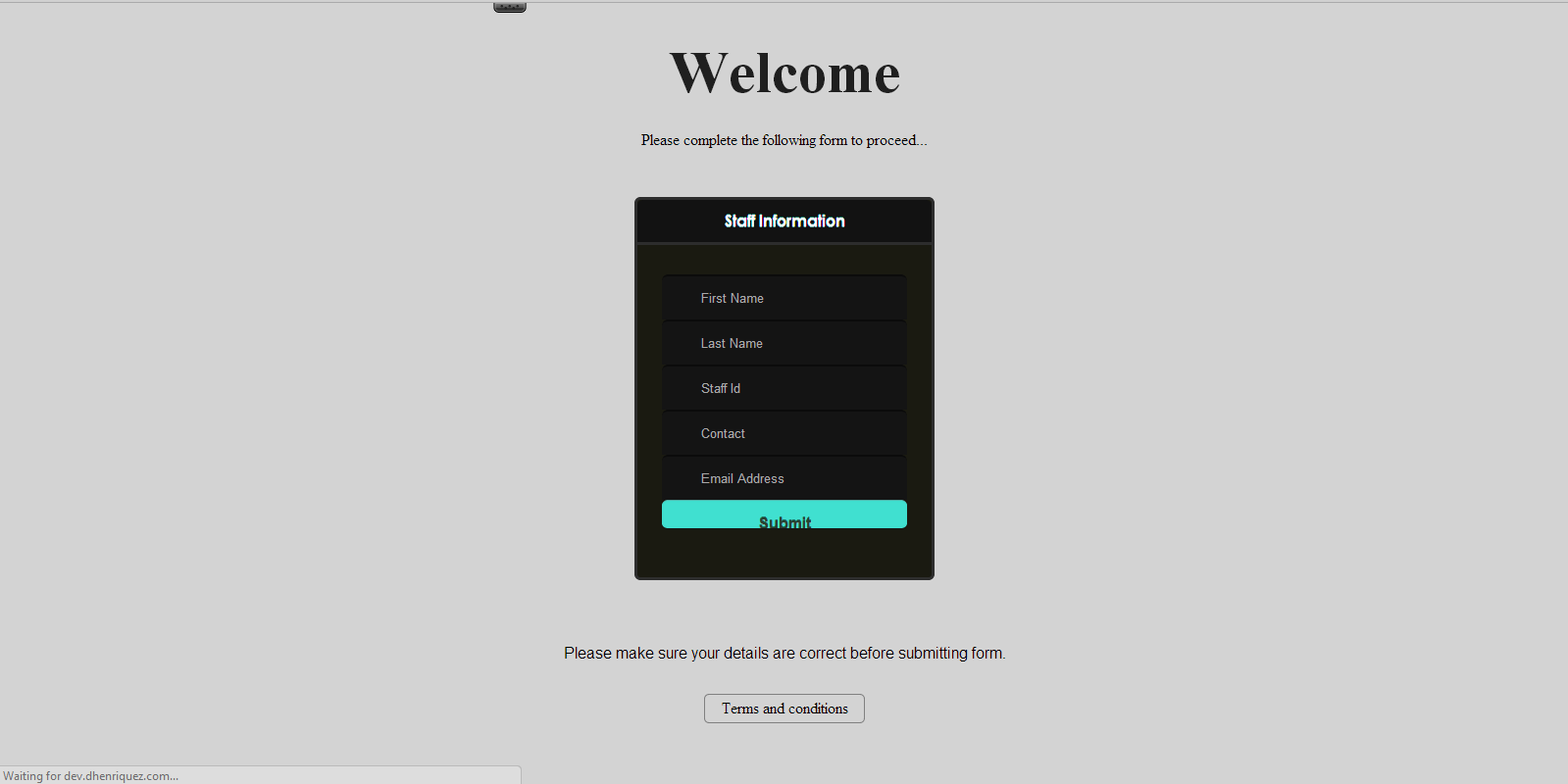
**[6] M.Y.I Idris, “***Car**Park System: A review of Smart parking System and Its technology****”*, Information Technology Journal 8(2), 101-113, 2009**

**[7] [Online] http://www.upark.in**

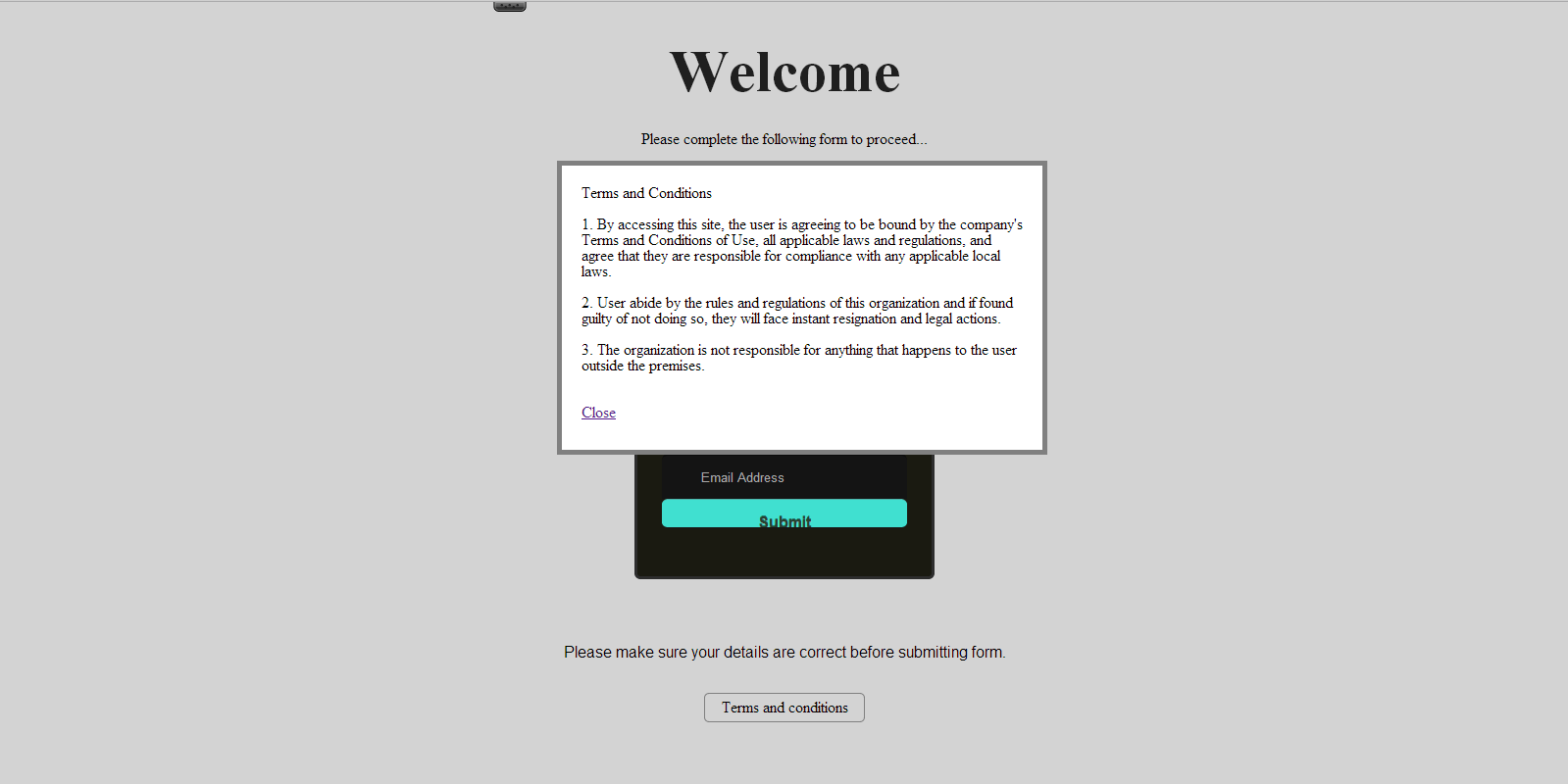
**IMAGES OF PROJECT**



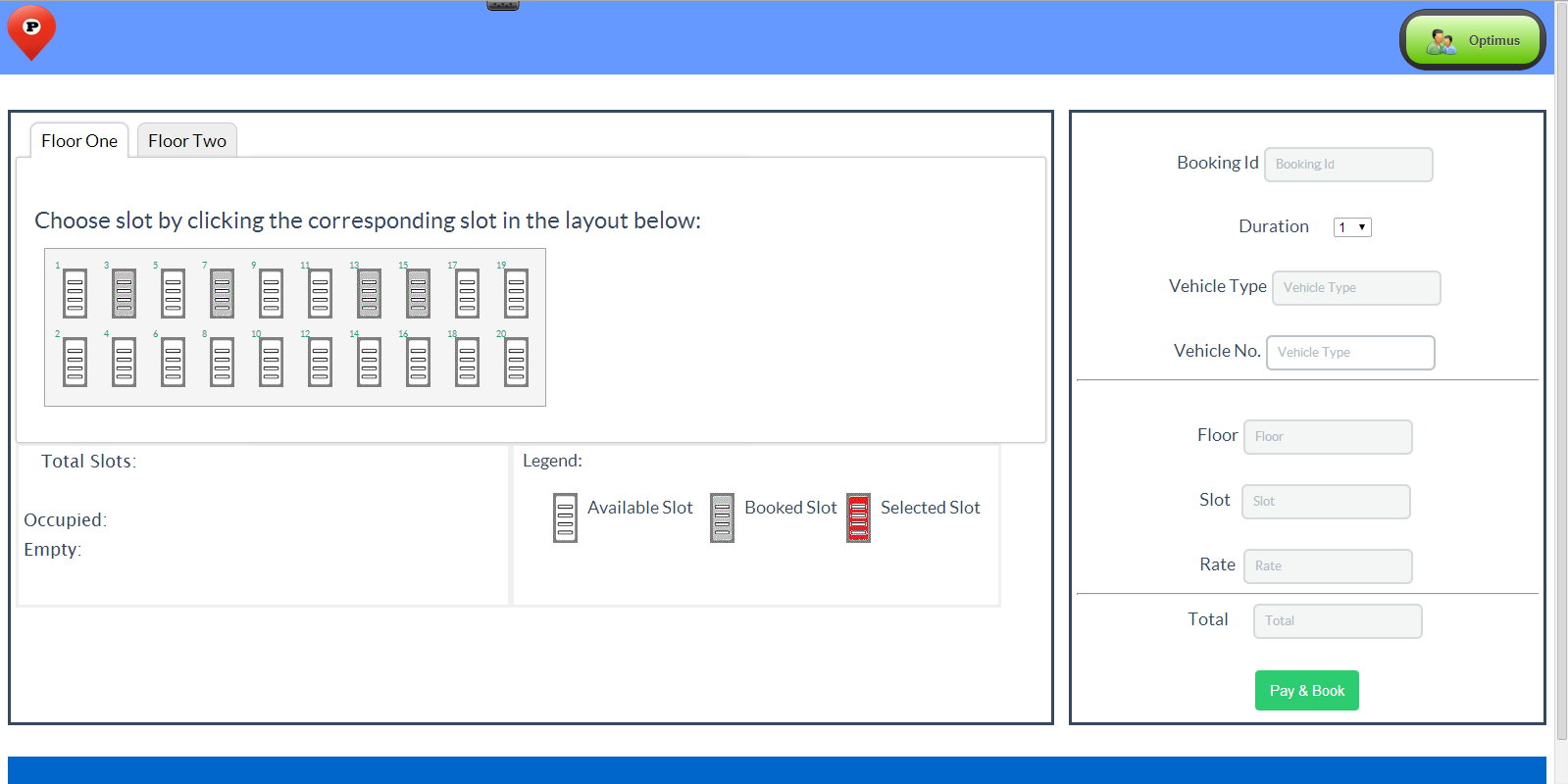
**Image 1: Login Page**



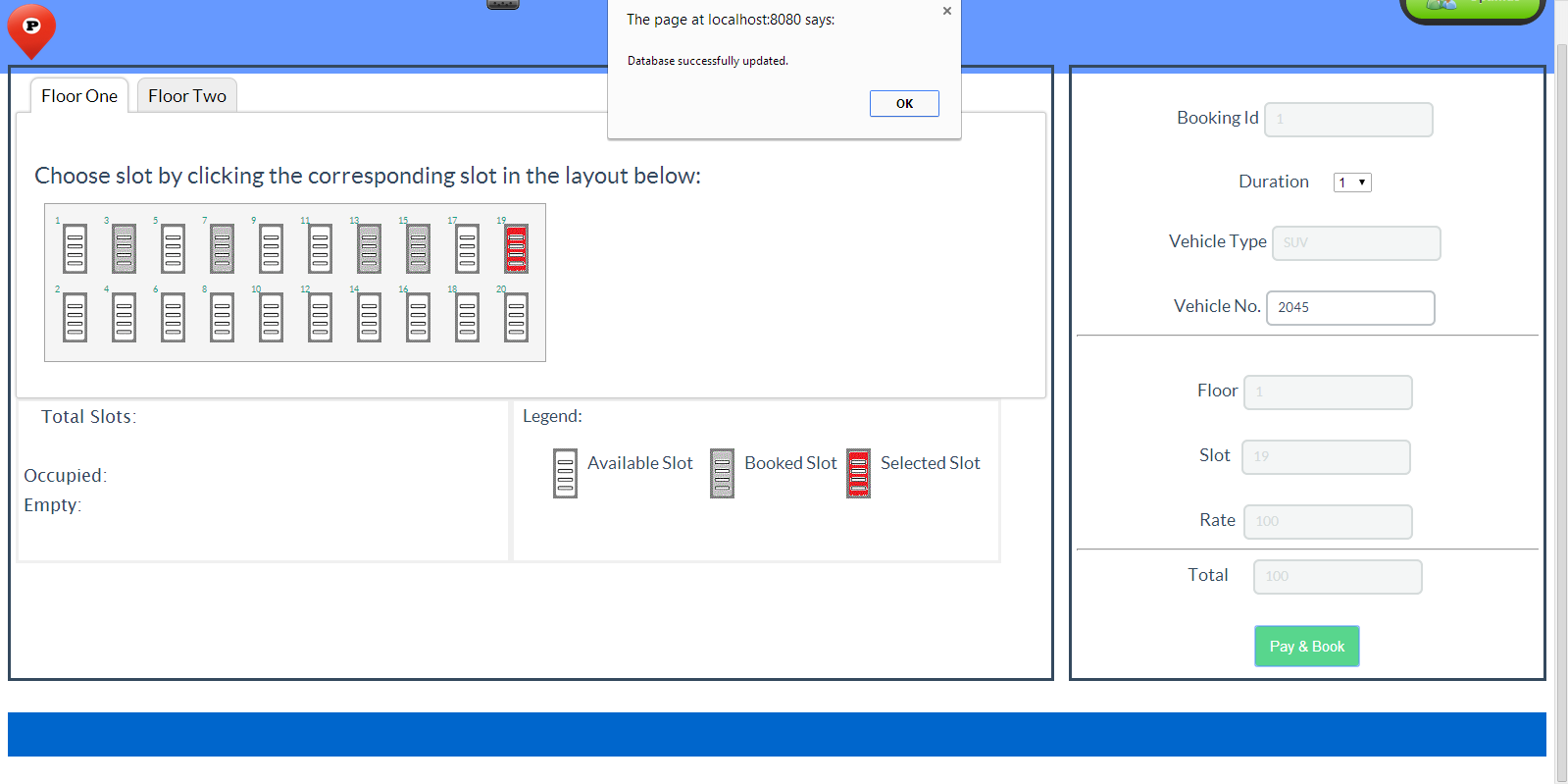
**Image 2: Sign Up Page**



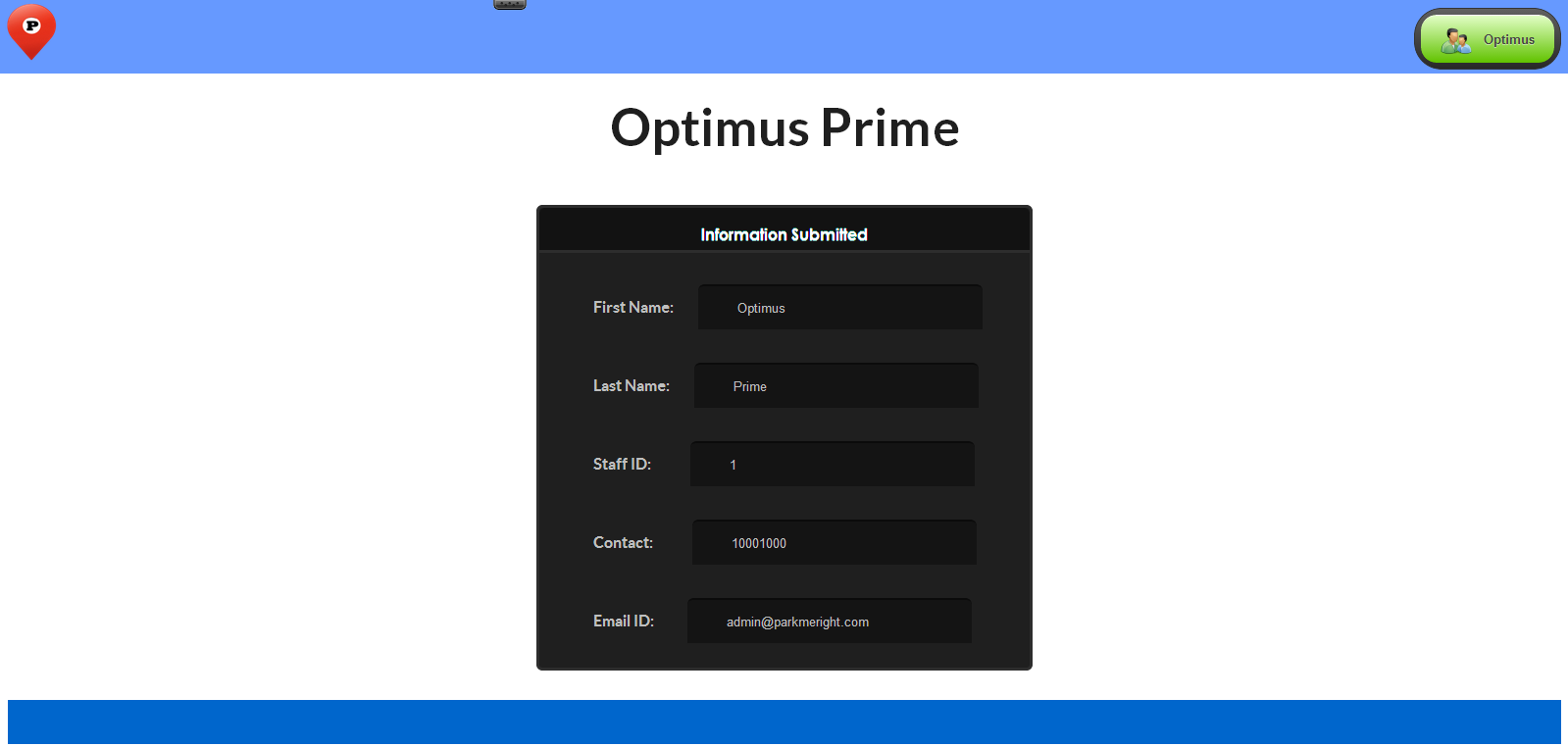
**Image 3: Terms and Conditions**



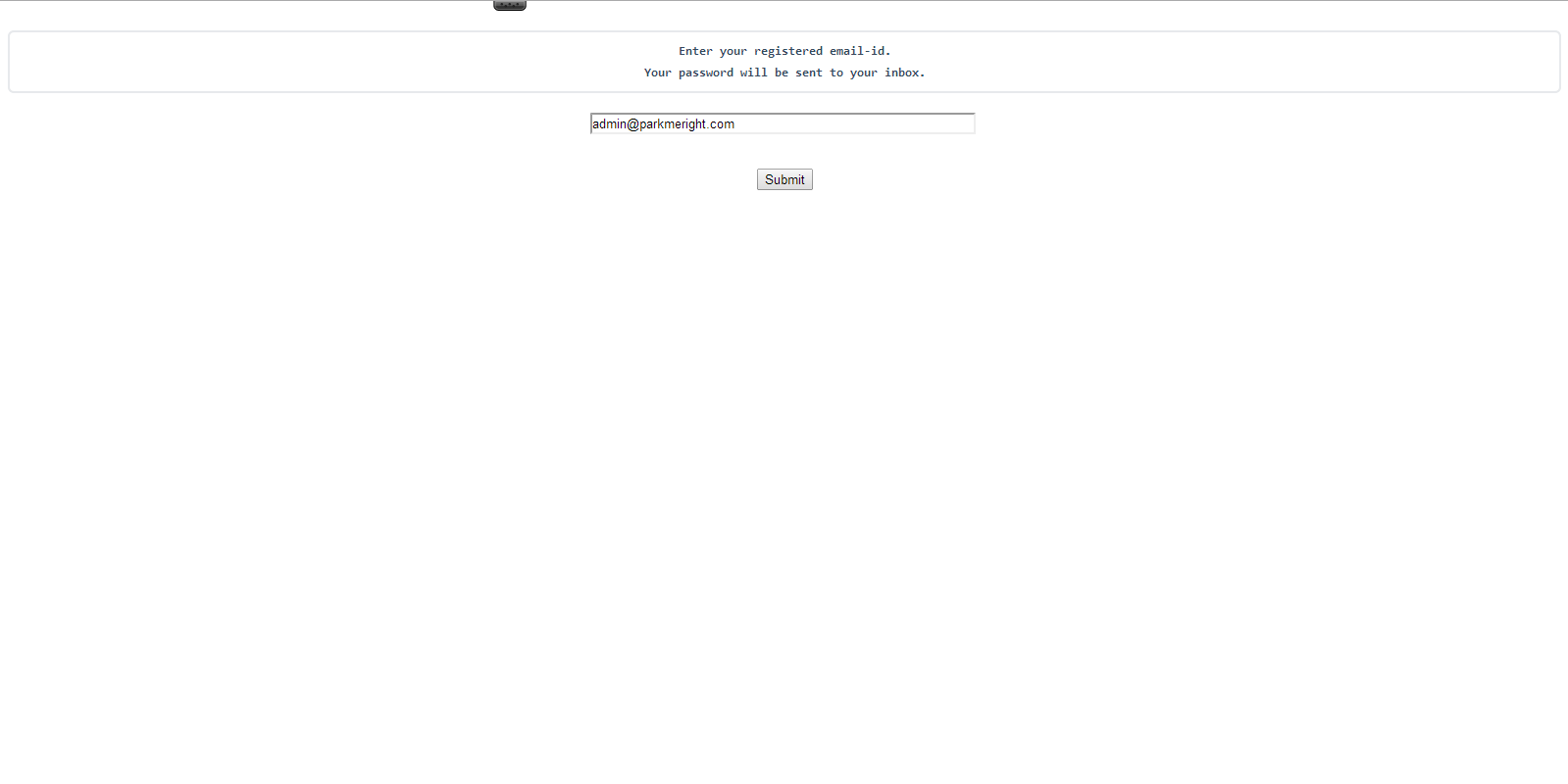
**Image 4: Main Page**



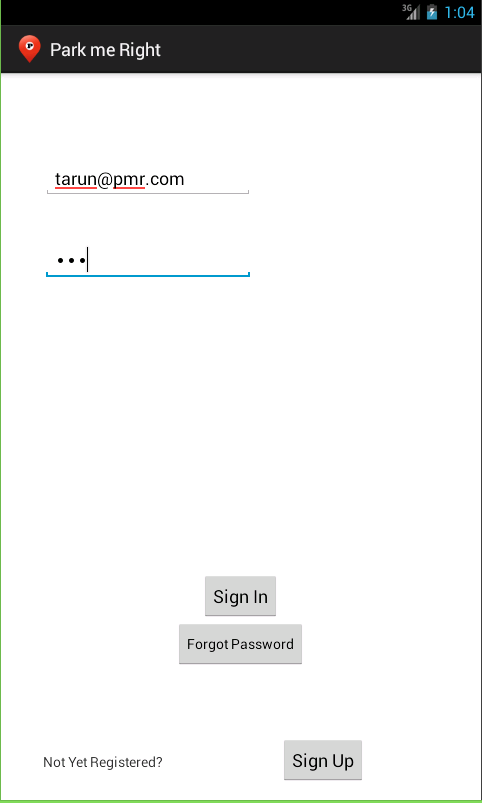
**Image 5: Booking a slot**



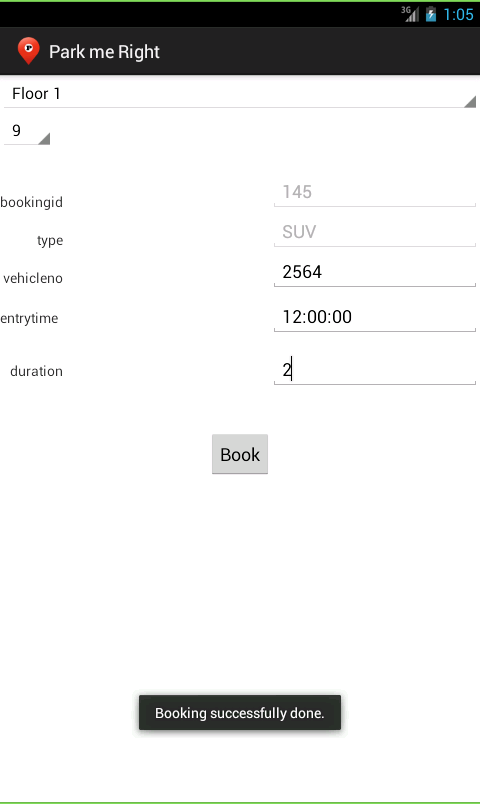
**Image 6: Profile Page**



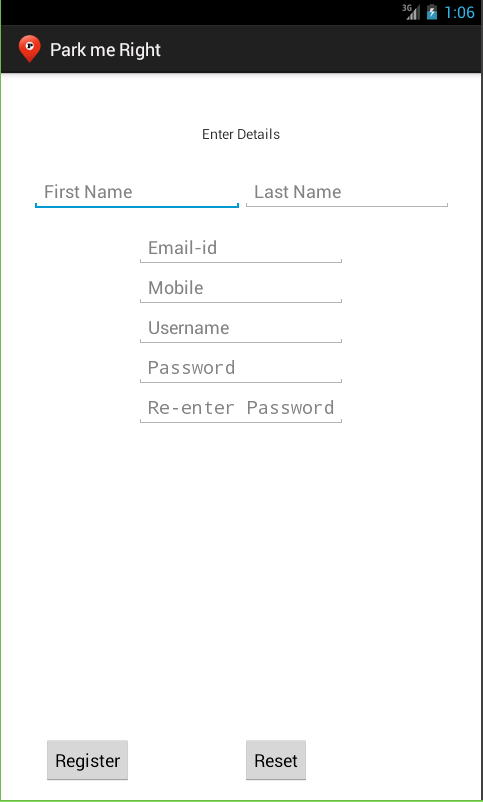
**Image 7: Forgot password facility**



**Image 8: Android App Login**

****

**Image 9: Advance Booking of Slot**

****

**Image 10: Registration Page**