**IMPLEMENTATION**

The project implementation started with the communication stage. First of all, the team attended a meeting with the mentor and came upon with the idea. Once the project was finalized, the team worked upon the requirements gathering in which all the particulars were collected about the project such as the literatures, journals, the output which would be achieved at the end, user requirements etc.

After all this, the focus shifted on the module division. The team of five was distributed the work evenly and the first step was to design the User Interface.

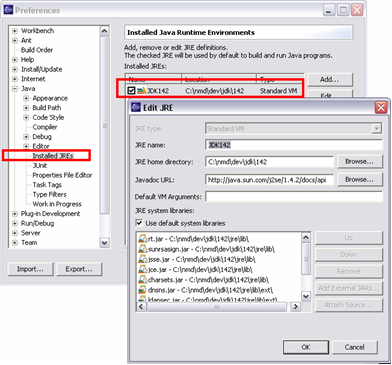
Once the UI was designed the database was created and the database connectivity was done for the web app.

The linking of pages was performed and android app was being developed side by side. On the completion of both the web app and android app were linked to each other using web services.

The project was complete after this step but the most important thing now to do was the testing process. The testing and debugging was done using the Rest Easy client tool which is specially used to check the working of web services created.

* 1. **STEPS FOR CREATING AND CONFIGURING WEB APPLICATION USING ECLIPSE**

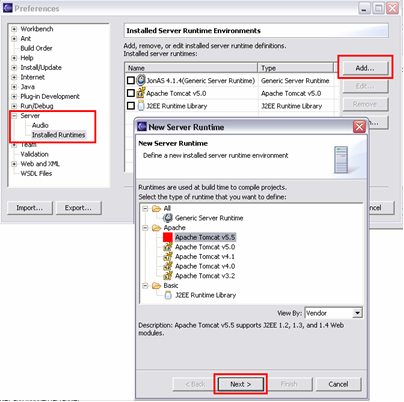
First of all, launch eclipse IDK. After you launch eclipse for the first time, but before we can build our first web application, we need to do a little eclipse housekeeping: If it is not already defined, you will need to set the Java preferences to point to your JDK installation (see figure). You should have obtained the latest JDK from Sun.



**Figure 6.1: Setting JDK runtime environment for the web application.**

### Choose a Server Runtime Environment

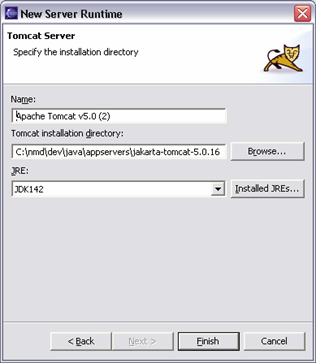
This step involves changing or creating the settings for Installed Server Runtimes. During web application development, we will need a server runtime environment to test, debug and run our project. We begin by telling WTP what our server runtime is, and where it is located. A “server runtime environment” is not a server. It provides the environment, libraries and infrastructure that a “server” needs. A server is an instance of the server runtime that can host our web applications and other server-side components. To define a server runtime, we need to visit the appropriate Preferences page:



**Figure 6.2: Setting Server runtime environments for the web application project**

Here we will find a list of server runtimes that have been defined previously. We can choose a server runtime and change its properties. To add a new one click “Add…”. A wizard will popup and display a list of server runtimes that are supported by WTP. Choose server runtime from the list. Jonas and JBoss are currently supported under the “Generic Server Runtime”. Basic “J2EE Runtime Library” is not a true server runtime, but can be used for coding support only. It does not provide a server instance that can be used to run web artifacts.

After we click “Next”, we will have to choose the JDK and the folder where we have installed our server runtime. Of course, based on your choice of the server runtime, we will be asked to provide different properties. Click finish.

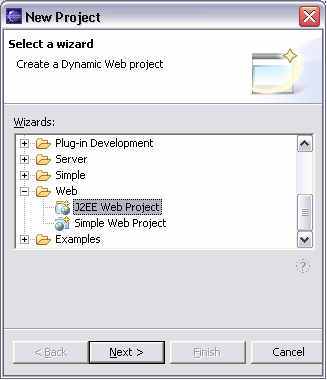


**Figure 6.3: Selecting Tomcat Server As the New Server Runtime**

We chose Apache Tomcat. You can also choose another server such as ObjectWeb Jonas. If you would like to use them as your runtime, select the “Generic Server”, and follow the wizard to define the properties for Jonas, JBoss or Weblogic. There are differences between for server runtimes; Apache Tomcat provides dynamic development support; i.e: the ability to run the web application from the eclipse project without the need to publish the artifacts to a server. Generic server does not support dynamic development, but it will automatically publish the artifacts (your web application) to the server when you run.

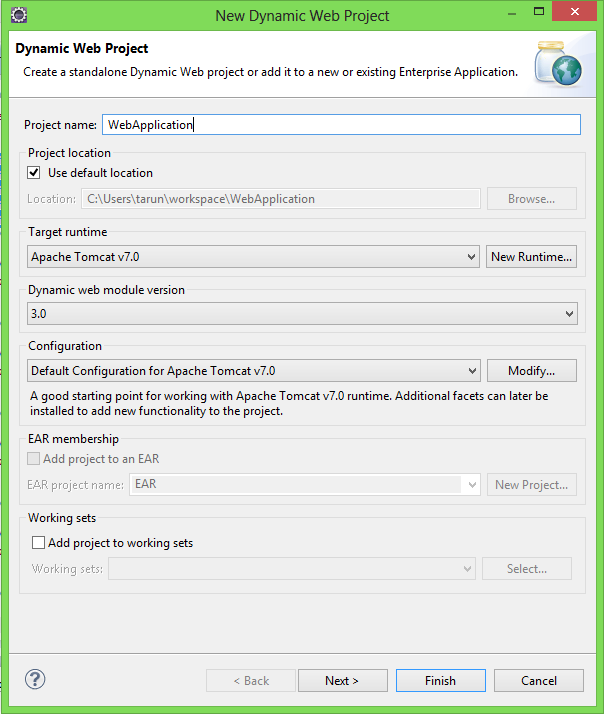
* + 1. **Create a J2EE web project with a web module**

Go to New->Project. Choose Web category from the list of available wizards to create a new eclipse resource. You will find two project types available under this category. Simple Web Project is a basic Eclipse resource project that can be associated with a server. We are interested in creating a J2EE Web Application, so we will choose the J2EE Web Project here:



**Figure 6.4: J2EE Project under configured server runtime environment**

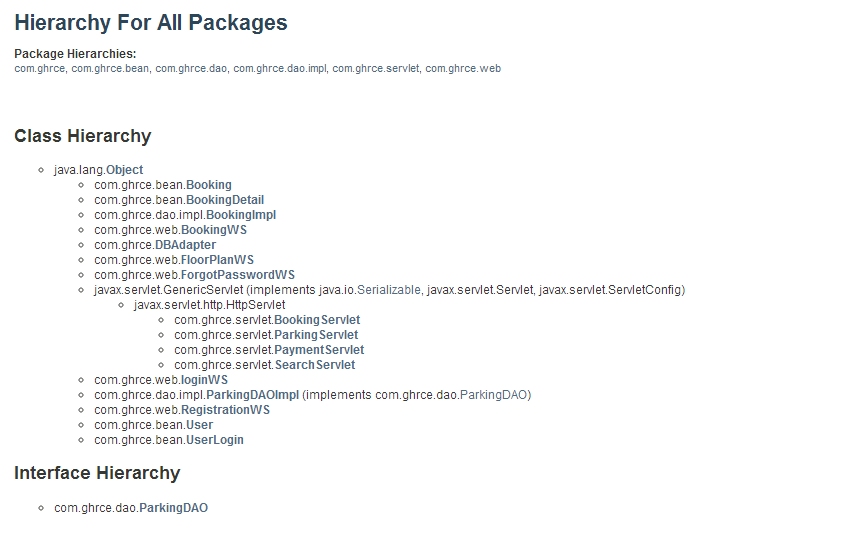
A J2EE Web Project is a type of Java project that has been initialized to host a server-side application. It needs to be associated with a Target Server.   
A target server provides the container which will be used to execute our web application. Specifically, it will provide a set of libraries (jars), that are added to the project classpath, which are needed to compile our custom classes. E.g.: to create a new Servlet, we need javax.servlet.\* packages. Server runtime provides them to the project. Make sure that “Add module to an EAR project” is not selected. We will create a standalone web application that can be deployed as a standard web module.  
Finally, we will change the context root to “parkmeright”. This is used when you access the web application with a URL, i.e. http://localhost:8080/parkmeright



**Figure 6.5: Creating new dynamic web application in new server runtime environment.**

When the wizard completes its tasks, you will find that it has create a new Java Project and has initialized it so that you can start with your development right away: It has created a java source folder. You will add your custom java packages and classes here. It has defined the classpath (JRE and Tomcat) so that you will have J2EE libraries in your project classpath. It has created a resource folder named “Web Content”, which mimics the standard WAR structure from the J2EE specification. It will contain the web resources that will be packaged with your J2EE web module. All artifacts inside this folder are accessible from the Web Application context.

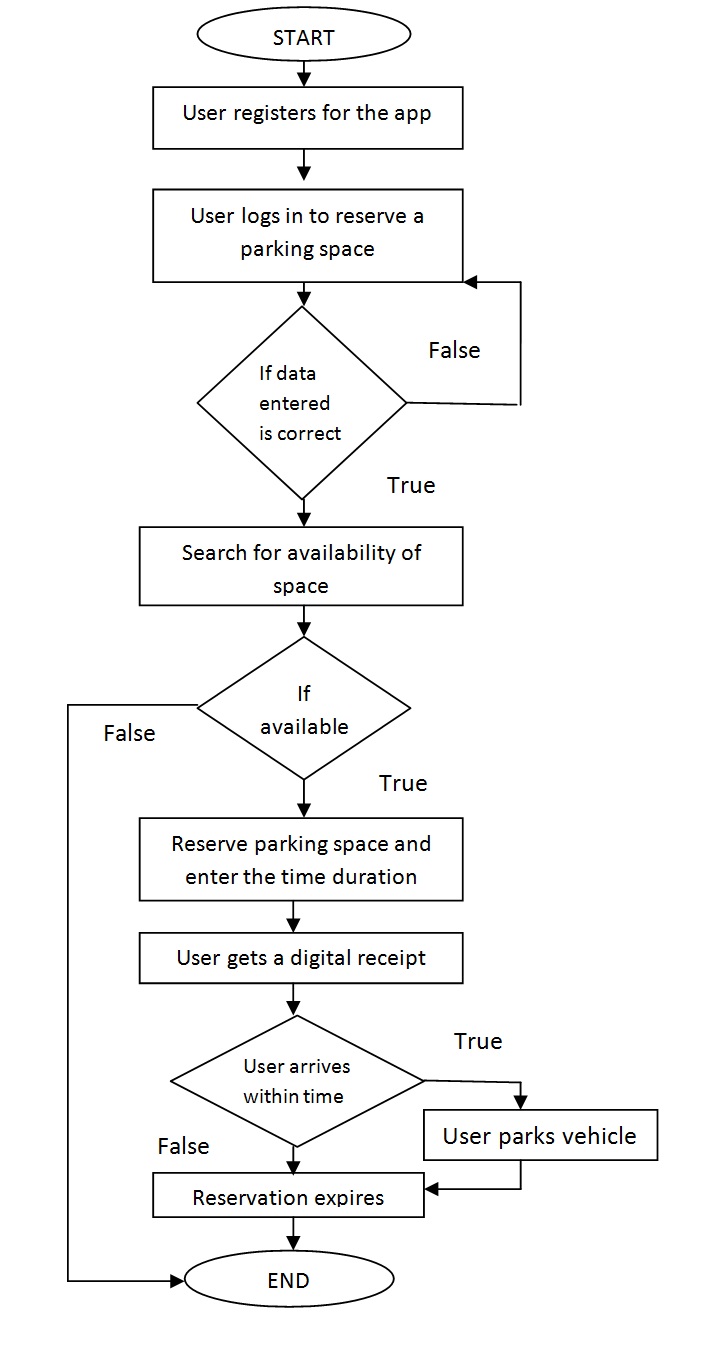
* 1. **Tree Hierarchy of Web App**



**Figure 6.6: Tree Hierarchy of Project**

* 1. **Flow Chart**

Given below is the flow chart designed by the project team which describes the flow of the activities in the project.



**Figure 6.7: Flow Chart**

1. As depicted by the given flow chart above, the user would first have to register for the app by providing the information asked for in the registration page.
2. Then once the account of the user is created, they can log in to reserve a parking slot in the car park.
3. If the data entered is correct, they would be able to access their account otherwise they would have to login again by providing correct credentials.
4. Once logged in, the user can see the availability of slots. If any slot is available, the user can reserve that place by providing some details like the entry time and duration after which the user would receive a confirmation message with booking id.
5. The user has to arrive on time and then park the vehicle and avail this facility or else the reservation gets expired.
   1. **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.

* 1. **Ajax**

**Asynchronous JavaScript and XML** is a group of interrelated web development techniques used on the client-side to create asynchronous web applications. With Ajax, web applications can send data to, and retrieve data from, a server asynchronously (in the background) without interfering with the display and behavior of the existing page. Data can be retrieved using the XML HttpRequest object. Despite the name, the use of XML is not required (JSON is often used instead), and the requests do not need to be asynchronous. JavaScript and the XMLHttpRequest object provide a method for exchanging data asynchronously between browser and server to avoid full page reloads.

**Steps of AJAX Operation**

1. A client event occurs
2. An XMLHttpRequest object is created
3. The XMLHttpRequest object is configured
4. The XMLHttpRequest object makes an asynchronous request to the Web server.
5. Web server returns the result containing XML document.
6. The XMLHttpRequest object calls the callback () function and processes the result.
7. The HTML DOM is updated
   1. **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 API used to send and receive email via SMTP, POP3 and IMAP. JavaMail is built into the Java EE platform[12], but also provides an optional package for use in Java SE.

* 1. **Servlets [8]**

**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.

**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.

**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.

**Search Servlet:**

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

* 1. **Web Services**

RESTEasy[9] 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.

**BookingWS.java:**

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

**FloorPlanWS:**

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

**ForgotPasswordWS:**

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

**LoginWS:**

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

**RegistrationWS:**

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

* 1. **Android App [11]**

**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.

**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.

**Activities**

An activity provides a means of interaction to the user. It provides a window where the UI can be designed according to the window. Almost all activities interact with the users. This application has the following activities: Login Activity, Registration Activity, Forget Password Activity, Booking Activity.

**Login Activity:**

This class helps the user to login to the application through their unique username and password. After logging in the application the users will be directed to the main booking page from where they can book the suitable slot. The valid username and password will be checked from the database using the JSON webservice.

For creating the login activity first the layout for the application was created. The normal user’s comfort and understanding was kept in mind while designing the layout so that it is easier to use. Then the java class for creating the UI threads were made so that it could work in sync with other UI activities and also run smoothly. Then as to check whether the username and password is valid, also to feed its entry in the database a java class was created to post the username and password of the user on the remote server. This was implemented using JSON.

**Registration Activity:**

This class lets the user to register with the application. The user has to register by giving a valid password and an email address as username. Each registered user will have a unique username. All the user details are stored in a database located on a server. This enables the user to login from any device as the data is centrally located on a server. Certain constraints have been set on the password and the email is checked for a valid format. All the details that will be field by the user will be feeded in the database through JSON web services.

For creating the registration activity first the layout for the application was created. The normal user’s comfort and understanding was kept in mind while designing the layout so that it is easier to use. Also all the necessary fields regarding user’s information were added for making the application more efficient. Then the java class for creating the UI threads were made so that it could work in sync with other UI activities and also run smoothly. Then as to feed its entry in the database a java class was created to post all the information about the user on the remote server. This was implemented using JSON.

**Forget Password Activity:**

If the user forgets his password and clicks on the “Forget Password” button on the login page, he will be directed to this activity. Here the user has to enter his email address so that his password can be sent to his mail address by the application. In this secure way he can recover his password.

For creating this activity, first the layout was designed which contains only single field of email address of the user which user will enter to recover his password in case if he forgets it. Again the java class for creating the UI threads were made and also to direct the action to the server about sending the password to the provided mail id a class was made using JSON.

**Booking Activity:**

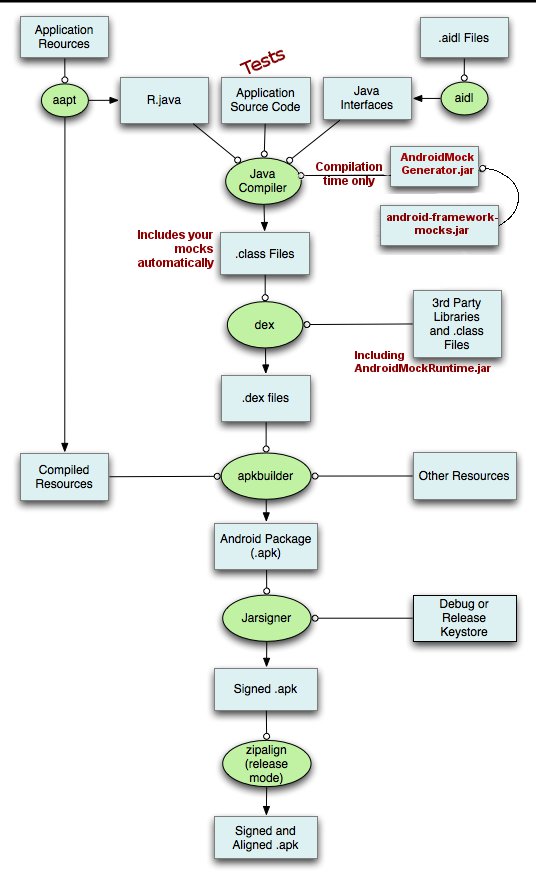
This class allows the users to select the suitable floor according to their car size and then choose the suitable slot from the available slots for their car. Here they merely need to enter their car number, entry time and the duration for which they have to park their vehical and then simply book it by clicking on the “Book” button.

For creating the booking activity first the layout for the application was created. The normal user’s comfort and understanding was kept in mind while designing the layout so that it is easier to use. Then the java class for creating the UI threads were made so that it could work in sync with other UI activities and also run smoothly. Spinners were used to make the UI simple and easy for showing the number of floors and available slot numbers. For the data feeded by the user and the data that has to be fetched from the database a class was created to connect the activity to the remote server using JSON.

**Intents**

The activities are activated through messages called intents. It basically has information that is required by the component. It helps in launching the activity or to do something with the existing activity. An intent passed to startActivity() is delivered only to an activity. The intent object holds the name of the component that handles the event, the action to be performed, data on which the operation is performed, the key value pairs to send some additional information. The intents that are used in this application are ACTION\_SEND and ACTION\_EDIT. The ParkMeRight Application makes use of intents to send the users details and current scenario of the parking lot.

**Executing Android Project**



**Figure 6.8: Android project execution**