



**Industry Project**

**Mobile Testing as a Service**

**By**

**Apoorva K R**  
**(1225907)**

**Under the guidance of**  
**Prof. Nachamai**

**April-2015**



# 1. INTRODUCTION

## 1.1 PROBLEM DESCRIPTION

Mobile Applications are a rapidly developing segment of the global Mobile Market. They consist of software that runs on a mobile device and performs certain tasks before the user of the Mobile Phone. They can be downloaded physically through USB / WIFI from a desktop or can be downloaded by a web server over internet.

The increase in the large-scale on-demand mobile test service requests, mobile test simulation and traffic loads and virtualization of mobile devices and environments have raised the importance of mobile app testing in the current market.

Mobile application testing is a process by which application software developed for hand held mobile devices is tested for its functionality, usability and consistency. Mobile application testing can be automated or manual type of testing. Mobile applications either come pre-installed or can be installed from mobile software distribution platforms. Mobile devices have witnessed a phenomenal growth in the past few years.

There are different platforms present in the industry such as android, apple, windows etc. The mobile testing will work around two major types of mobile apps namely native apps and mobile web app. Native apps live on the device and are accessed through icons on the device home screen. Native apps are installed through an application store (such as Google Play or Apple's App Store). They are developed specifically for one platform, and can take full advantage of all the device features — they can use the camera, the GPS, the accelerometer, the compass, the list of contacts, and so on.

Web apps are run by a browser and typically written in HTML5. Users first access them as they would access any web page, they navigate to a special URL and then have the option of “installing” them on their home screen. Testing these native and mobile web apps for their quality and performance is vital and the proposed system is aimed to develop an automated test framework which is compatible with IOS, Android and windows platform.

Service provided to the mobile application developers to verify their mobile application is Publishing ready. It also checks for the hybrid apps Cross-platform affinity.

There are three important features , namely, Access to native API, AppStore distribution, runs locally on the device , and it supports offline.

Though mobile testing showcases immense growth, there are certain challenges faced in mobile application testing:

- **Variety of Mobile Devices-** Mobile devices differ in screen sizes, input methods ([QWERTY](#), touch, normal) with different hardware capabilities.
- **Diversity in Mobile Platforms/OS-** There are different [Mobile Operating Systems](#) in the market. The major ones are [Android](#), [IOS](#), BREW, BREWMP, [Symbian](#), [Windows Phone](#), and [BlackBerry](#) (RIM). Each operating system has its own limitations. Testing a single application across multiple devices running on the same platform and every platform poses a unique challenge for testers.
- **Mobile network operators-** There are over 400 mobile network operators in the world, out of which some are [CDMA](#), some [GSM](#), whereas others use less common network standards like [FOMA](#), and [TD-SCDMA](#). Each network operator uses a different kind network infrastructure and this limits the flow of information.
- **Scripting-** The variety of devices makes executing the test script (Scripting) a key challenge. As devices differ in keystrokes, input methods, menu structure and display properties single script does not function on every device.
- Problems faced by the customers in the existing mobile testing system will provide a deeper understanding of the problem description.
- Variety of mobile devices in market and multiple manufacturers.
- Coping with the short lifecycle of the mobile application in market.
- Huge variety of hardware capabilities.
- Shorter duration of device life in the market.
- Variety of network modes like 2G/3G/4G/Wi-Fi/Wi-Max.

- Huge investment and high time frame to set up a test lab.
- Most of the tools are image based comparison tool.
- One test tool may not support all platforms versions.
- Tools may need jailbreak/rooting that invokes security threat.
- Testing on latest technologies like HTML5 etc.

## 1.2 EXISTING SYSTEM

Manual testing procedure is performed by a human sitting in front of a computer carefully going through application screens, trying various usage and input combinations, comparing the results to the expected behavior and recording their observations. Manual tests are repeated often during development cycles for source code changes and other situations like multiple operating environments and hardware configurations.

There are variety of problems that are faced by the customer in mobile testing. The main problem faced is the manual procedure of testing mobile applications.

There are many aspects involved in the existing system of mobile testing like different platforms android, ios, windows and blackberry. Different device specifications, application specific factors like the native apps, mobile apps, network specific factors like net connectivity, Wi-Fi, and geo specific problems are faced in the existing system. A framework which can handle all these aspects with ease and an automated environment is necessary in the market. Managing all the factors important for testing an android and validating the app and make it fit enough to be uploaded into the Google store, apple store etc

The proposed system aims at developing a testing framework which allow the customer to upload their apps and test various scenarios in a completely automated environment. The framework is compatible with all platforms and contains all types of testing which will test if the mobile application is in terms with all the conditions required to be used by the user with ease.

The existing system is involved with manual testing of an application. Manual testing is a popular practice but it can be time consuming and tedious. The proposed framework contains all the test cases in automated formatted which consumes less time and is cost-effective.

The main advantage of an automated framework are assumptions, concepts and tools that provide support for automated software testing is the low cost for maintenance. If there is change to any test case then only the test case file needs to be updated and the driver Script and start up script will remain the same. Tools like selendroid and UI automator is used to support the automated framework. Therefore the existing system is tedious to use and the manual style is time consuming. The customers have to depend on different vendors and testers in order to test their app under different platforms and devices. The framework proposed has completed automated test cases and is compatible with all platforms and the effective emulators and simulators will ease the work load and is cost-effective.

### 1.3 PROJECT SCOPE

Every software development group tests its products, yet delivered software always has defects. Test engineers strive to catch them before the product is released but they often reappear, even with the best manual testing processes. Automated software testing is the best way to increase the effectiveness, efficiency and coverage of your software testing.

An automated testing tool is able to playback pre-recorded and predefined actions, compare the results to the expected behavior and report the success or failure of these manual tests to a test engineer. Once automated tests are created they can easily be repeated and they can be extended to perform tasks impossible with manual testing. Because of this, automated software testing is an essential component of successful development projects.

The main aspect of this project which deals with Mobile Automation is :

- Find an element (e.g. Button)
- Interact with the element (e.g. Click a Button)

There are many factors which influence the need for mobile testing as a service. There are many diverse mobile devices and Hardware appliances & APIs, diverse mobile operation environments (platforms, connectivity, and configurations), diverse wireless connectivity and configurations factors which influence the need for mobile testing as a service.

Mobile TaaS offers a new business model for diverse mobile validation services using pay-as-u-test to achieve cost-saving and cost-reduction in mobile computing resources, networks, cloud computing and storage infrastructure.

MtaaS provides on demand testing services for mobile applications and SaaS to support software validation and quality engineering processes by leveraging a cloud based scalable mobile testing environment.

Before a developer publish the apps on Google Play and distribute them to users, the application needs to be tested and the app should satisfy all the standards and requirements of the Google play store, apple store accordingly.

The proposed system aims at developing a testing framework which allow the customer to upload their apps and test various scenarios in a completely automated environment.

The main aim of the framework is testing of a mobile app through defining various automated test case scenarios to improve quality and avoid bugs.

The main objectives of Mobile testing as a service are as follows

- Reduce mobile testing costs and complexity
- Easy to set up an integrated mobile test environment for projects
- Enable to support large-scale testing different types of mobile testing
- Compatibility with Android and ios.
- An ability to test in an automated environment with ease and define various test cases for native apps and web apps.

Automated software testing is a process in which software tools execute pre-scripted tests on a software application before it is released into production. The objective of automated testing is to simplify as much of the testing effort as possible with a minimum set of [scripts](#). If [unit testing](#) consumes a large percentage of a quality assurance ([QA](#)) team's resources, for example, then this process might be a good candidate for automation.

Automated testing tools are capable of executing tests, reporting outcomes and comparing results with earlier test runs. Tests carried out with these tools can be run repeatedly, at any time of day.



## 2. SYSTEM ANALYSIS

### 2.1 FUNCTIONAL SPECIFICATIONS

Interface of MTaaS is designed using HTML and written in JavaScript .

The Interface consists of a Homepage. The Homepage consists of three links namely Native Apps , Mobile Web App Testing and Contacts each leading to a new page when clicked.

#### **Native Apps Testing Page:**

When clicked on Native Apps Testing, Login page is displayed which asks for the user Login and Password. Once logged in, another page is displayed which consists of a 'Browse' button and a 'Select AVD' dropdown box.

We know that Native Apps can be accessed without a web connection. So in the framework, by clicking on the Browse button, the user can upload a particular app which needs to be tested browsing through his system. The user then needs to select a particular Virtual Device on which the application needs to be tested i.e. from a list of Virtual Devices available in the dropdown box.

Native App - Native apps are built for a specific platform with the platform SDK, tools and languages, typically provided by the platform vendor

- Best Suited to provide best user experience and for targeted devices
- Dependent on native platform and hence requires separate code bases for respective device platforms

#### **Advantages**

- Ability to leverage device-specific hardware and software
- A richer, more compelling user experience
- Ability to run offline

### Disadvantages

- Separate version required for different platforms, which requires more cost and time
- Updation of apps tedious compared to web apps

### Mobile Web App Testing Page:

When clicked on the Mobile Web App Testing link, a Login page is displayed which asks for the user Login and Password. Once logged in, another page is displayed which consists of a Text box and a 'Select AVD' dropdown box.

Mobile web refers to the content that is viewed through a Smartphone's web browser. If you have a mobile version of your website, any mobile device owner with an Internet browser installed (Internet Explorer, Mozilla Firefox) can view your mobile website.

Unlike Native app's Browse button, a particular URL needs to be entered into the Textbox of the mobile web app that needs to be tested.

The user then needs to select a particular Virtual Device on which the application needs to be tested i.e. from a list of Virtual Devices available in the dropdown box. Once the data is entered, the 'Submit' button needs to be clicked which stores the page in the server side in the form of an xml document.

Web App – Mobile Web apps are server-side apps, built with any server-side technology (PHP, Node.js, ASP.NET) that render HTML that has been styled so that it renders well on a device form factor. They are accessible over device native / third party browsers

- Applicable if existing web page functionality to be accessed from wide range of devices
- Maximum interoperability as there is little or no dependency on native device OS

### Advantages

- Compatibility – Mobile Websites are generally compatible across browsers
- Upgradability – Mobile Websites can be Updated Instantly and pushed to users
- Time and Cost - Mobile Websites are Easier to and Less Expensive to develop

### Disadvantages

Limited user experience as there will be a latency for every request to the server

- Accessing device features like Camera, Address book, Bluetooth etc is very limited
- Caching of data – only to certain extent

### **Emulator**

Android emulator comes as part of the android SDK commonly known as AVD – Android Virtual Device.

- It lets the user to prototype, develop, and test Android applications without using a physical device. Android Emulators
- The AVD's are OS version specific and provides the user the flexibility to customize OS version, resolution, skin, sd card size and various other hardware properties to be emulated.
- There are many command line utilities and tools which comes as part of the sdk which makes it easy to debug and interact with emulator Prerequisites for Android Emulator
- JRE – Java Runtime Environment
- Android SDK Installing an application
- If the application is available in Google Playstore it can be directly downloaded and installed on to the device.
- If the application is available in '.apk' format ,it can be installed using the command, 'adb install '.Adb is a command line utility which comes as part of the SDK.

### **Automated Framework:**

Once a particular apk file is uploaded (Native App) or an url is typed (Mobile Web App) into the testing framework and submitted. The files go to the server's side and the testing is done in a completely automated environment.

The proposed system aims at developing a testing framework which allows the customer to upload their apps and test various scenarios in a completely automated environment.

In this framework Native App as well as Mobile Web App test frameworks are embedded.

With a pay-as-you-test methodology, this framework makes the mobile app testing easier, faster and cost effective.

The reason why the mobile testing is rapidly expanding and marching towards the top of the market depends on the fact that it has shown remarkable growth in business and testing is a vital part.

Testing of a mobile app through defining various automated test case scenarios in order to avoid bugs and to improve quality is the main idea behind the framework.

### **2.1.1 NON-FUNCTIONAL SPECIFICATIONS**

Non-functional testing is an extensive part of software testing that obscures a lot of different aspects of software behavior.

Some of the most usual non-functional tests that are identified and conducted on a regular basis are performance, capacity, and failover tests.

The apps that are uploaded in the frame work exhibit non-functional requirements such as:

- Memory Management
- Reliability
- Usability
- Maintainability
- Portability
- Recovery

The app will be subjected to several non-functional testing to portray its behaviour.

MtaaS system is very reliable. Reliability Testing is about processing an application so that bugs are exposed and rectified before the system is used. The reason of this testing is

to establish system trustworthiness, and to conclude whether the application meets the client's reliability necessities. Usability testing, the framework tests the simplicity with which the user application can be used. It tests that is the software or the application created is easy to use or not. Usability testing has few components which are described as follows:

- **Learnability:** It is to check how simple for users to achieve fundamental tasks the foremost moment when they meet the interface?
- **Efficiency:** To test how quickly knowledgeable users can complete tasks.
- **Memorability:** After a phase of not using the system, when the users revisit to the interface do they memorize adequate to use it efficiently, or have to begin once more learning all.
- **Errors:** To checks how much bugs the users create, how rigorous are these bugs and how effortlessly can users rectify the bugs. Other than these factors performance, security and compatibility are also important non-functional requirements.
- **Performance testing:** This testing is organized, to establish how quickly several features of an application work under a specific load. It also exhibit the system's performance standard. Performance testing compares two systems to discover which one works better than other. It also measures what element of the system is responsible the system to carry out poorly.
- **Security testing:** Security testing ensures that the application or the system is secured or not. It checks that can somebody hack the software or login into the application by breaking the authentication. It concludes that a system protects information and maintains performance as proposed.
- **Compatibility testing:** Compatibility testing; generally tests the system or the product created with the computing background. It tests the application which is created is friendly with the operating system, hardware, database or other application or not. In this case it checks for compatibility with different platforms like android, apple, windows and blackberry accordingly.



## 2.2 BLOCK DIAGRAM

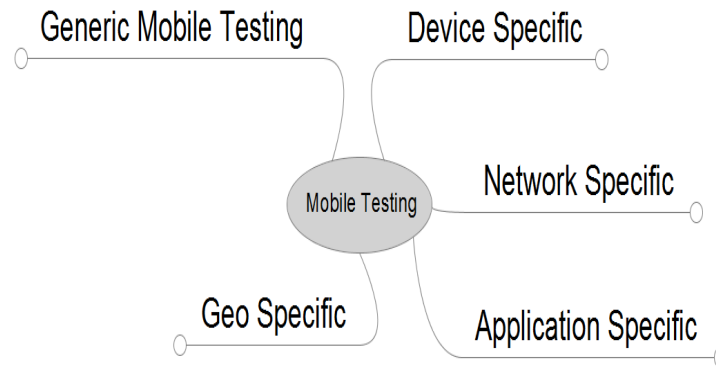


Fig 2.2.1 Block diagram of MTaaS

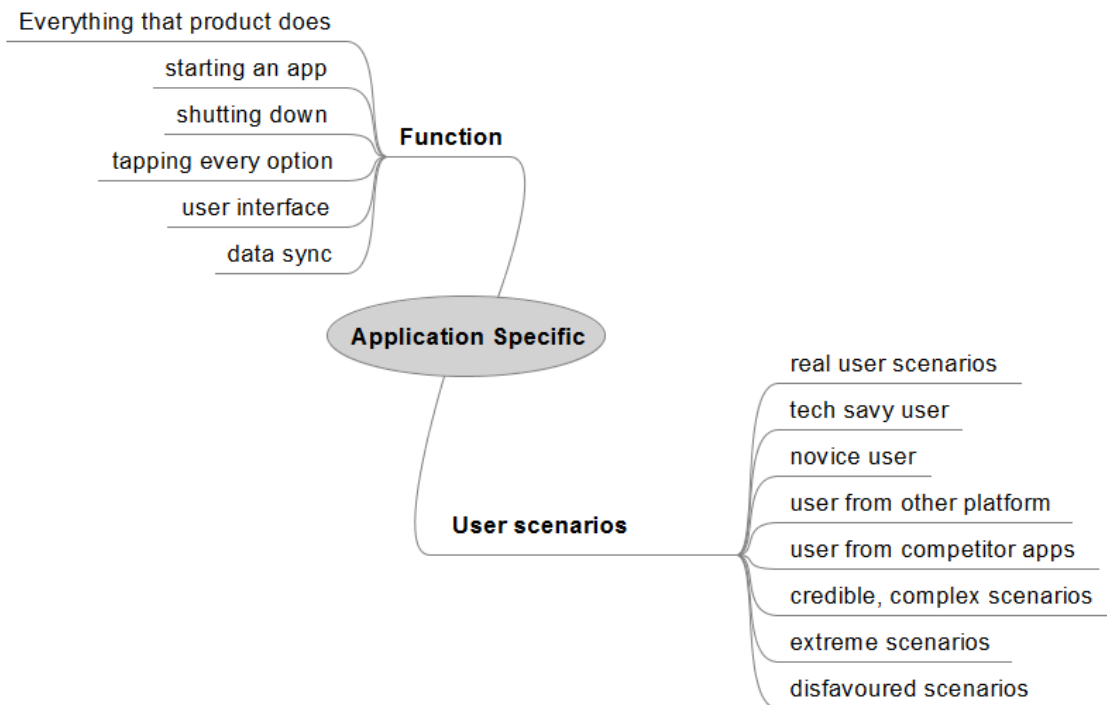


Fig 2.2.2 Application specific module diagram

## **2.3 SYSTEM REQUIREMENTS**

### **2.3.1 HARDWARE REQUIREMENTS**

- Hard Disk: 500GB
- RAM: 512MB

### **2.3.2 SOFTWARE REQUIREMENTS**

- Eclipse (Juno).
- Operating System: Windows 7
- Programming Language: Java.
- Testing Tool : Selendroid, UI Automator.

### **2.3.3 TOOL SURVEY**

Selenium originally is a tool to automate web browsers. The JSON Wire Protocol [2] describes user interactions with a browser. Each browser has a specific driver which is a native implementation for that specific browser. Selenium is a classic client/server architecture. The client side is decoupled from the driver (server). From a client perspective, the commands to interact with the browser are the same, regardless which browser is used.

Selendroid is a test automation framework which drives off the UI of Android native and hybrid applications (apps) and the mobile web. Tests are written using the Selenium 2 client API. Selendroid is a test automation framework which drives off the UI of Android native and hybrid applications and the mobile web. Tests are written using the Selenium 2 client AP. Selendroid can be used on emulators and real devices and can be integrated as a node into the Selenium Grid for scaling and parallel testing.

The use of Selendroid requires knowledge about how to use Selenium. For testing any iOS native, hybrid, or mobile web application using WebDriver. Selendroid supports



testing on hardware devices as well as using Android emulators. In the capabilities the properties are used to find the device for the test execution.

Selendroid is an open source automation framework which drives of UI of android native, hybrid and mobile web application. It supports both emulator and real device. It uses Json Wire Protocol to run webdriver test scripts on device. It can be integrated with selenium grid for parallel execution on multiple nodes. No need any modification in application and not need source code to automate application.

**Prerequisites:**

- JDK should be installed and java home path setup in the machine.
- Android SDK should be installed on the machine
- Download Selendroid
- Selenium jar file
- Eclipse.
- Create new Emulator or attached real devices with the machine.

**Features of selendroid**

- No modification of app under test required in order to automate it
- Testing the mobile web using built in Android driver web view app
- Same concept for automating native or hybrid apps
- UI elements can be found by different locator types
- Gestures are supported: Advanced User Interactions API
- Selendroid can interact with multiple Android devices (emulators or hardware devices) at the same time
- Existing emulators are started automatically
- Selendroid supports hot plugging of hardware devices
- Full integration as a node into Selenium Grid for scaling and parallel testing
- Multiple Android target API support (10 to 19)
- Built in Inspector to simplify test case development.

## UI AUTOMATOR

Android UI testing, the developer wants to test how the application interacts with a user on a real device. UI testing ensures that the application returns the correct UI output in response to a sequence of user actions, such as entering keyboard input or pressing toolbars, menus, dialogs, images, and other UI controls.

A GUI tool to scan and analyse the UI components of an Android application. The uiautomator tool provides a convenient visual interface to inspect the layout hierarchy and view the properties of the individual UI components that are displayed on the test device. Using this information, you can later create uiautomator tests with selector objects that target specific UI components to test. The uiautomator testing framework lets the tester to test the user interface (UI) efficiently by creating automated functional UI testcases that can be run against the app on one or more devices.

### Pre-requirements

- Install [Android SDK](#), and set android\_home environment to the correct path.
- Enable ADB setting on device and connect your android device using usb with your PC.

The [uiautomatorviewer](#), a GUI tool is used to scan and analyze the UI components of an Android application. In order to use uiautomatorviewer, the tester must first download and install the SDK and the Eclipse IDE according to the instructions at [Setting up the ADT Bundle](#). After the installation, the tool exists in the /tools/ folder and you can start it by typing: uiautomatorviewer from the command line.

With uiautomator, the tester can inspect the UI of an application in order to find the layout hierarchy and view the properties of the individual UI components of an application. This is very important, particularly when it is needed to construct automation testing because, by knowing the layout hierarchy of the application and the IDs of the individual widgets, the tester can use them in the uiautomator in order to create automation tests.

### 3. SYSTEM DESIGN

#### 3.1 SYSTEM ARCHITECTURE

#### 3.2 MODULE DESIGN

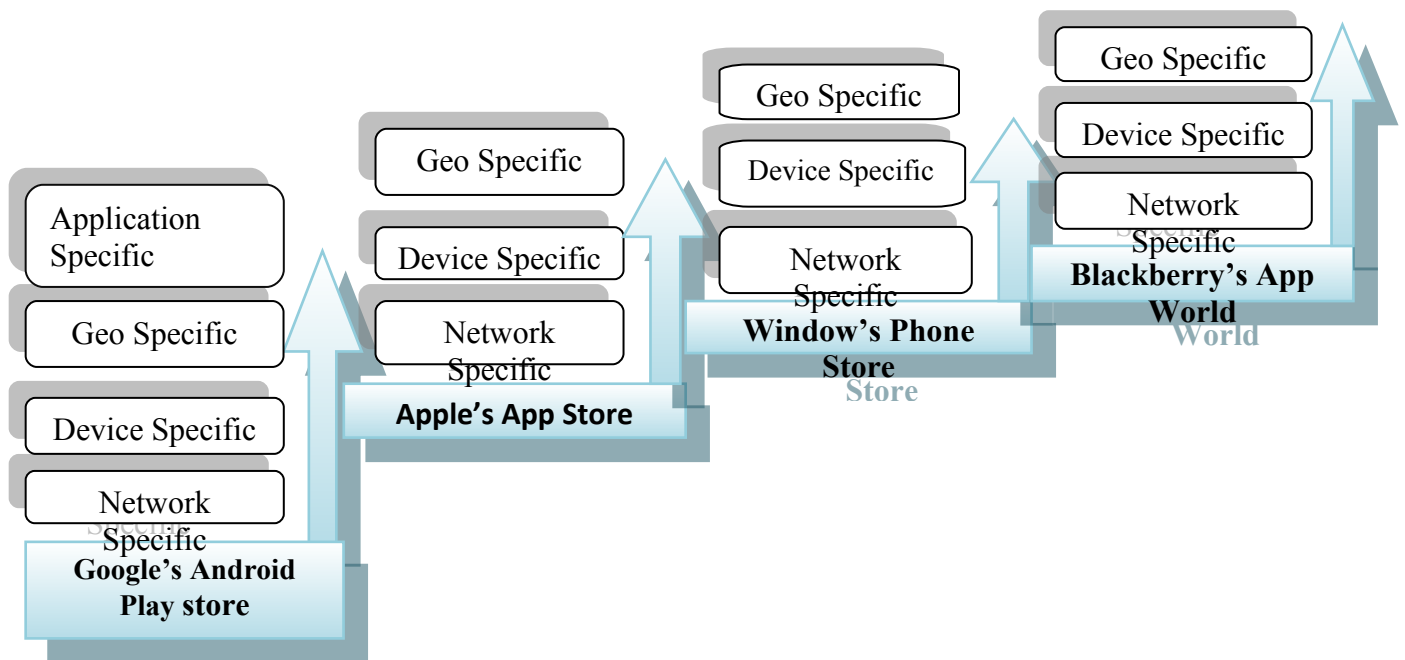


Fig 3.2.1 Modular Design of MtaaS

MTaaS is planned to be released in four stages or modules.

Phase 1: Google's Android Play Store

Phase 2: Apple's App Store.

Phase 3: Window's Phone Store.

Phase 4: Blackberry's App World.

### **Google's App Store**

Google Play, originally the Android Market, is a [digital distribution](#) platform operated by [Google](#). It serves as the official [app store](#) for the [Android](#) operating system, allowing users to browse and download applications developed with the [Android SDK](#) and published through Google. Google Play also serves as a [digital media](#) store, offering music, magazines, books, movies, and television programs. Users can also purchase hardware devices through the service, such as [Chrome books](#), [Google Nexus](#)-branded [mobile devices](#), [Chrome casts](#), and accessories.

Applications are available through Google Play either free of charge or at a cost. They can be downloaded directly to an Android or [Google TV](#) device through the Play Store [mobile app](#), or by [deploying](#) the application to a device from the Google Play website. Many applications can be targeted to specific users based on a particular hardware attribute of their device, such as a motion sensor (for motion-dependent games) or a front-facing camera (for online video calling).

## Apple App Store

The App Store is a [digital distribution](#) platform for [mobile apps](#) on [iOS](#), developed and maintained by [Apple Inc.](#) The service allows users to browse and download applications that are developed with Apple's [iOS SDK](#). The apps can be downloaded directly to an [iOS device](#), or onto a personal computer via [iTunes](#) (also developed and maintained by [Apple Inc.](#)).

Applications are targeted at iOS devices, including [iPhones](#) and [iPads](#), and may make use of specific attributes of those devices, such as motion sensors for game controls and cameras for online video calling. Apps may be downloaded for free or for a set cost, and they may include in-app monetization (costs levied through buyable features and/or advertising). Apple takes 30 percent of all revenue generated through apps, and 70 percent goes to the app's publisher.

## Window's Phone Store

Windows Phone Store (previously Windows Phone Marketplace) is a digital distribution platform developed by [Microsoft](#) for its [Windows Phone](#) platform that allows users to browse and download [applications](#) that have been developed by third parties.

Like much of the new Windows products, it features "[Metro UI](#)"; the [UI](#) is presented in a [panoramic](#) view where the user can browse categories and titles, see featured items, and get details with ratings, reviews, screen shots, and pricing information.

The Windows Phone Store (replacing Windows Marketplace for Mobile) was launched along with Windows Phone 7 in October 2010 in some countries. It was reported on October 4, 2010 that the Windows Phone [SDK](#) was downloaded over half a million times. At the end of February 2013, the Marketplace had more than 130,000 apps available. With the rollout of Mango (Windows Phone 7.5) the online web Marketplace was unveiled by Microsoft; it offers various features like silent, over the air installation of apps to the user's device.

## Blackberry's App World

BlackBerry World is an [application distribution](#) service and application by [BlackBerry Ltd](#) for a majority of [BlackBerry](#) devices. The service provides BlackBerry users with an environment to browse, download and update [third-party applications](#). The service went live on April 1, 2009. Of the three major app stores of different [Operating systems](#), it has the largest revenue per app compared to the Apple App Store and Google Play, respectively. On 21 January 2013, BlackBerry announced that it rebranded the BlackBerry App World to simpler BlackBerry World as part of the upcoming release of the [BlackBerry 10](#) operating system.

As of March 2013 - BlackBerry App World is available in 170 markets and supports 23 currencies and 33 languages. Over 6 million applications are downloaded daily with an aggregate of over 4 billion downloads to-date and accepts payment in all markets using a combination of PayPal, credit card, and carrier billing.

## SUB MODULES

### Application specific

Application specific module deals with two major components called native app and mobile web app.

Native App - Native apps are built for a specific platform with the platform SDK, tools and languages, typically provided by the platform vendor.

Web App – Mobile Web apps are server-side apps, built with any server-side technology (PHP, Node.js, ASP.NET) that render HTML that has been styled so that it renders well on a device form factor. They are accessible over device native / third party browsers

### Network specific

The network specific module deals with the strength, availability, reliability, switching between 3G and Wi-fi, 2G and hotspot related to different platforms like android, apple,

windows and blackberry. This tests how the app behaves when subjected to all these network related changes.

### **Device specific**

The device specific factors differs depending on the device that is used for testing an app. It is important to test and understand the behaviour of the app in different mobile devices. There are two main device specific factors to be considered, namely, platform and orientation.

Platform involves consistency, actions such as swipe , Zoom-in, Zoom-out, multi-touch , long press(touch and hold), double tap and rotate.

Orientation is of two types, Landscape and Portrait. This checks how the app behaves when the screen orientation is changed. The factors an app should satisfy are correctness in alignments, pop ups, notifications, font size, zoom and swipe.

### **Generic mobile testing**

Many generic specific modules deals with the general testing that the app should be subjected to, irrespective of the platform.

Each platform will have a set of guidelines or general requirement specifications to be followed before publishing the app into the app store.

General mobile testing has 4 main categories as mentioned below.

#### **Updates:**

- upgrade - os
- developer data seed
- developer license
- downgrade

**Notifications:**

- Enable / Disable
- Local notification
- Wi-Fi or Cellular - push notification
- Visual, Sound and Vibration
- Notification from other apps
- Screen locked
- Setting

**Guidelines**

- apple HIG
- apple app store review
- android design
- Google play review

**Communication interruptions**

- voice messages
- calls



- text messages

### **Interruption Testing**

- SMS, MMS, and Charger insertion.

### 3.3 DATABASE DESIGN

#### 3.3.1 DATA FLOW DIAGRAM

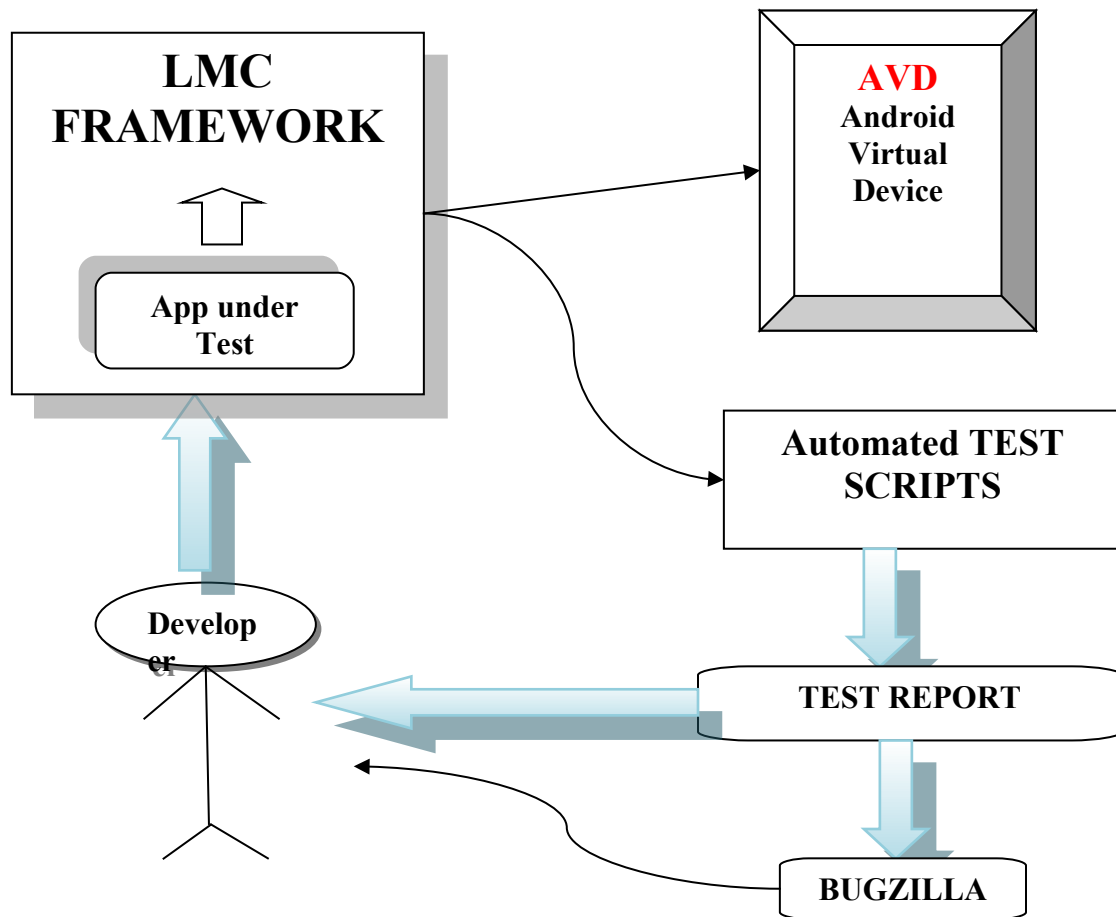


Fig 3.3.2.1 Data Flow Diagram

### 3.4 INTERFACE DESIGN

#### 3.4.1 USER INTERFACE SCREEN DESIGN

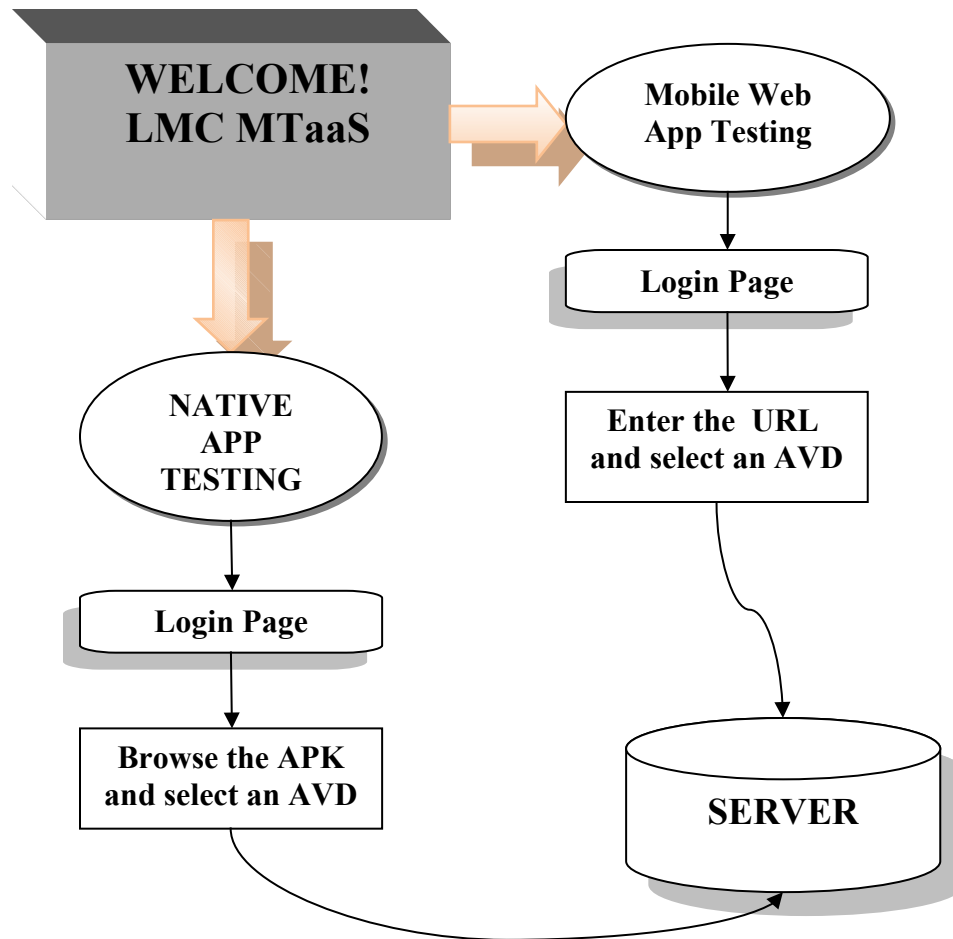


Fig 3.4.1 Interface Design



[illegible]

```

Password : <input type="password" name=MTaaSPassword required value="Pass-
word" onBlur="if(this.value=="")this.value='Password'" onFocus="if(this.value=='Pass-
word')this.value=" ">
<br> </br>
<br> </br>
<input type="submit" value="Login">
<footer class="clearfix">
<p>
<span class="info">?</span>
<a href="#">Forgot Password</a>
</p>
</footer>
</form>
</table>
</body>
</div>
</nav>
</html>

```

## Jsp

```

<html> <head> <meta charset="UTF-8">
<title>LMC Mobile Testing As A Service</title>
<link href="style.css" rel="stylesheet" type="text/css" /> </head>
<body> <table> <tr> <div id="wrapper"> <header><h1> LMC MTaaS - Mobile Web
App Testing</h1></header>

<nav> <a href="LMCcontactDetails">Contact Us</a></nav> </div> </tr>
<tr> <div>
<div id="wrapper">
<table >
<td> </td>

```

```

<td align="center">
  <form method=POST action=MTaaS.jsp>
<table><tr><td>
Enter the URL of the Mobile Web App to be tested:
<input type=text name=htmlAPP_URL size=16>
</td>
<br> </br>
<br> </br>
  <td> <div>
    <table>      <tr>
<div id="faq" class="super-container full-width" style="padding-top:40px;padding-
bottom:20px">
Select the Virtual Device on which the application to be tested
  <td><select name="htmlAVD" >
    <option value="DEVICEAPILevel14">
Name: device001 Device: Nexus One (Google) Target: Android 2.1 (API level 7)
Tag/ABI: default/armeabi Skin: 480x800</option>
    <option value="DEVICEAPILevel15"> Name: device003 Device: Nexus 5
(Google) Target: Android 4.4.2 (API level 19) Tag/ABI: default/armeabi-v7a Skin:
1080x1920</option> </select></td> </tr> </table>
</div> </td> </tr>
<tr> <td>
List of Physical Devices Available for Testing<table>
<tr>
<td> <div class="super-container full-width ">      <div class="container"
id="features">
      <div> <ul class="grid cs-style-1"> <li class="one-third column
defaultShadowStyle3 mywhite">
        <figure > <hText> <h1 class="iconSize alignC lightText3"><left>
 </left></h1>
<figcaption> <h3>Samsung Galaxy S5</h3> <p class="marginT10">

```

```

SM-G900H<br/> Android 4.4.2 <br/> 5.1" Screen<br/> 1080 x 1920 Resolution<br/> 2
GB Ram<br/>          xxhdpi </p> </figcaption> </figure> </li>
</td>
<td>
<li class="one-third column defaultShadowStyle3 mywhite">
<figure > <hText>    <h1 class="iconSize alignC lightText3">
<center> </center></h1>
<figcaption> <h3>Nokia X</h3> <p class="marginT10"> RM-980<br/> Android 4.1.2
<br/> 4.0" Screen<br/> 480 x 800 Resolution<br/> 512 MB Ram<br/> hdpi          </p>
</figcaption> </figure></li> </td>
<td>
<li class="one-third column defaultShadowStyle3 mywhite"> <figure >
    <hText>
<left>  </left>
    <h3>LG Nexus 5</h3> <p class="marginT10">
Nexus5<br/> Android 4.4.2 <br/>    5.0" Screen<br/>1080 x 1920 Resolution<br/>2 GB
Ram<br/>xxhdpi</p>
</figure></li>
</td>
</ul> </div> </div> <!-- End 960 Container --></div>    <!-- super container -->
</td></tr>
</div></tr>
</table>
</form>

</td>
<td> </td>
</tr>
</table>
</div>
</body></html>

```



```

<%@
page import = "java.lang.*" %><%@
page import = "java.io.*" %><%!
String appURL1 = null;
    String AVD1 = null;
    %>
    <% appURL1="<" + "MAINPROJECT" + "><" + "PROJECT"
+"><"+"APP_URL"+">" + request.getParameter("htmlAPP_URL")
+"</"+"APP_URL"+">"; %>
    <% AVD1="<DEVICE_NAME>" +request.getParameter("htmlAVD") +
"</DEVICE_NAME></PROJECT></MAINPROJECT>"; %>
<%
FileOutputStream fos = new
FileOutputStream("c:\\MTaaS\\EnvironmentDetails\\MobileWebApp.xml");
DataOutputStream dos = new DataOutputStream(fos);
dos.writeBytes(appURL1 + AVD1);
dos.flush();
    dos.close()
    %>
Css
* {
-moz-box-sizing: border-box;
box-sizing: border-box;
}
*:before,
*:after {
-moz-box-sizing: border-box;
box-sizing: border-box;
}
body {
background: #C0C0C0;

```

```
color: #999;
font: 400 16px/1.5em sans-serif;
margin: 2
}
h3 {
  margin: 0;
}
a {
  color: #999;
  text-decoration: none;
}
a:hover {
  color: #1dabb8;
}
fieldset {
  border: none;
  margin: 0
  display: block;
  margin-left: 2px;
  margin-right: 2px;
  padding-top: 0.35em;
  padding-bottom: 0.625em;
  padding-left: 0.75em;
  padding-right: 0.75em;
  border: 2px groove (internal value);
}
input {
  border: none;
  font-family: inherit;
  font-size: inherit;
  margin: 0;
```

```
-webkit-appearance: none;
}
input:focus {
  outline: none;
}
input[type="submit"] {
  cursor: pointer;
}
.clearfix {
  *zoom: 1;
}
.clearfix:before,
.clearfix:after {
  content: ' ';
  display: table;
}
.clearfix:after {
  clear: both;
}
}
.container {
  left: 50%;
  position: fixed;
  top: 50%;
  -webkit-transform: translate(-50%, -50%);
  -ms-transform: translate(-50%, -50%);
  transform: translate(-50%, -50%);
}
#login-form {
  width: 300px;
}
```

```
#login-form h3 {
  background-color: #282830;
  border-radius: 5px 5px 0 0;
  color: #fff;
  font-size: 14px;
  padding: 20px;
  text-align: center;
  text-transform: uppercase;
}

#login-form fieldset {
  background: #fff;
  border-radius: 0 0 5px 5px;
  padding: 20px;
  position: relative
}

#login-form fieldset:before
{
  background-color: #fff;
  content: "";
  height: 8px;
  left: 50%;
  margin: -4px 0 0 -4px;
  position: absolute;
  top: 0;
  -webkit-transform: rotate(45deg);
  -ms-transform: rotate(45deg);
  transform: rotate(45deg);
  width: 8px;
}

#login-form input {
  font-size: 14px
```

```
}  
#login-form input[type="email"],  
#login-form input[type="password"]  
{  
  border: 1px solid #dcdcdc  
  padding: 12px 10px;  
  width: 100%  
}  
#login-form input[type="email"] {  
  border-radius: 3px 3px 0 0;  
}  
#login-form input[type="password"] {  
  border-top: none;  
  border-radius: 0px 0px 3px 3px;  
}  
#login-form input[type="submit"] {  
  background: #1dabb8;  
  border-radius: 3px;  
  color: #fff;  
  float: right;  
  font-weight: bold;  
  margin-top: 20px;  
  padding: 12px 20px;  
}  
#login-form input[type="submit"]:hover  
{  
  background: #198d98  
}  
#login-form footer {  
  font-size: 12px;  
  margin-top: 16px
```

```
}  
.info {  
    background: #e5e5e5;  
    border-radius: 50%;  
    display: inline-block;  
    height: 20px;  
    line-height: 20px;  
    margin: 0 10px 0 0;  
    text-align: center;  
    width: 20px;  
}  
  
<html>  
<head><meta charset="UTF-8">  
<link rel="stylesheet" type="text/css" href="mystyle.css" />  
<div style="color:Black" font-family = "Arial, Verdana, sans-serif">  
<h1>LMC Mobile Testing As A Service</h1>  
</div>  
</head>  
<br>  
<br>  
<form method=POST action=MTaaS.jsp>  
<h3 >Enter the URL of the Mobile Web App to be tested:</h3>  
<input text-align="left" type=text name=htmlAPP_URL size=16>  
<br>  
<tr>  
<td>  
<style>  
body {background-color:lightgray;  
font-family: Arial, Verdana, sans-serif;}  
h3 {color:green;}
```

```

h3 {text-align:left}
</style>
</br>
</br>
<h3 ><span class="light"> Enter the AVD on which Mobile Web App to be
tested:</span></h3>
</br>
<td ><select name="htmlAVD" >
    <option value="1" > Name: device001 Device: Nexus One (Google) Target: Android
2.1 (API level 7) Tag/ABI: default/armeabi Skin: 480x800</option>
    <option value="2" > Name: device003 Device: Nexus 5 (Google) Target: Android
4.4.2 (API level 19) Tag/ABI: default/armeabi-v7a Skin: 1080x1920</option>
</td>
<br>
<br>
<input type=submit name=action value="Submit">
</form>
</html>

```

## JAVA

```

import java.util.List;
import java.util.concurrent.TimeUnit;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.Platform;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.interactions.Actions;

```

```
import org.openqa.selenium.interactions.touch.SingleTapAction;
import org.openqa.selenium.interactions.touch.TouchActions;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.Select;
import io.selendroid.SelendroidCapabilities;
import io.selendroid.SelendroidConfiguration;
import io.selendroid.SelendroidDriver;
import io.selendroid.SelendroidLauncher;
import java.io.File;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
public class TestUR4 {
    public static void main(String[] args) throws Exception
    {
        SelendroidLauncher selendroidServer = null;
        WebDriver driver = null;
        SelendroidConfiguration config = new SelendroidConfiguration();
        selendroidServer = new SelendroidLauncher(config);
        selendroidServer.launchSelendroid();
        SelendroidCapabilities sc = new SelendroidCapabilities();
        sc.setBrowserName("android");
        sc.setPlatform(Platform.ANDROID);
        sc.device("device1");
        driver = new SelendroidDriver(sc);
```



```
driver.get("http://m.ticketgoose.com");
WebElement b1 =
driver.findElement(By.xpath("//*[@id='homePagePopup']/div[1]"));
TouchActions flick1 = (TouchActions) new TouchActions(driver).click(b1);
flick1.click(b1);
driver.findElement(By.id("FromStationsNameTxt")).sendKeys("BANGALORE");
driver.findElement(By.id("ToStationsNameTxt")).sendKeys("MYSORE");
WebElement we = driver.findElement(By.id("day"));
Select sel1 = new Select(we );
System.out.println("TYPE IS -----");

System.out.println(we.getAttribute("type"));
sel1.selectByIndex(4);
WebElement we1 = driver.findElement(By.id("month"));
System.out.println("TYPE IS -----");
System.out.println(we1.getAttribute("type"));
Select sel2 = new Select( driver.findElement(By.id("month")));
TouchActions flick = new TouchActions(driver).singleTap(b1);
flick.perform();
TouchActions flick2 = (TouchActions) new TouchActions(driver).doubleTap(b1);
flick2.perform();
TakesScreenshot scrShot =((TakesScreenshot)driver);
SrcFile=scrShot.getScreenshotAs(OutputType.FILE);
File("..\\Reports\\ticketgoose.png")
FileUtils.copyFile(SrcFile, DestFile);
driver.switchTo().alert().dismiss();
}
}
```

## ReadXML

```
import java.io.File;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
public class ReadXML {
    public String readfile(String readData) throws ParserConfigurationException, SAXException, IOException
    {
        File xmlfile = new File("C:\\MTaaS\\EnvironmentDetails\\MobileWebApp.xml");

        DocumentBuilderFactory dbfactory = DocumentBuilderFactory.newInstance();

        DocumentBuilder dbuilder = dbfactory.newDocumentBuilder();
        Document doc = dbuilder.parse(xmlfile);
        NodeList nlist = doc.getChildNodes();
        Node node1 = nlist.item(0);
        Element e1 = (Element)node1;

        String url = e1.getElementsByTagName(readData).item(0).getTextContent();

        return url;
    }
}
```

## 4.2 SCREEN SHOTS

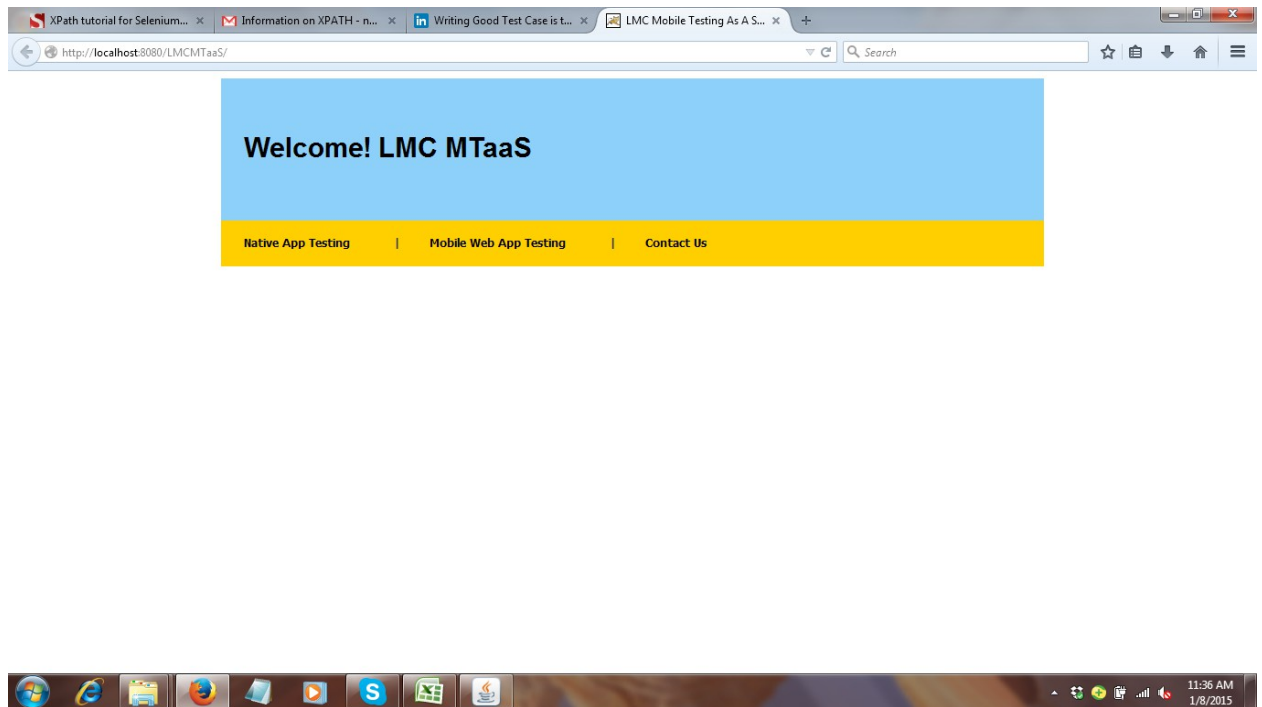


Fig 4.2.1 Welcome Page

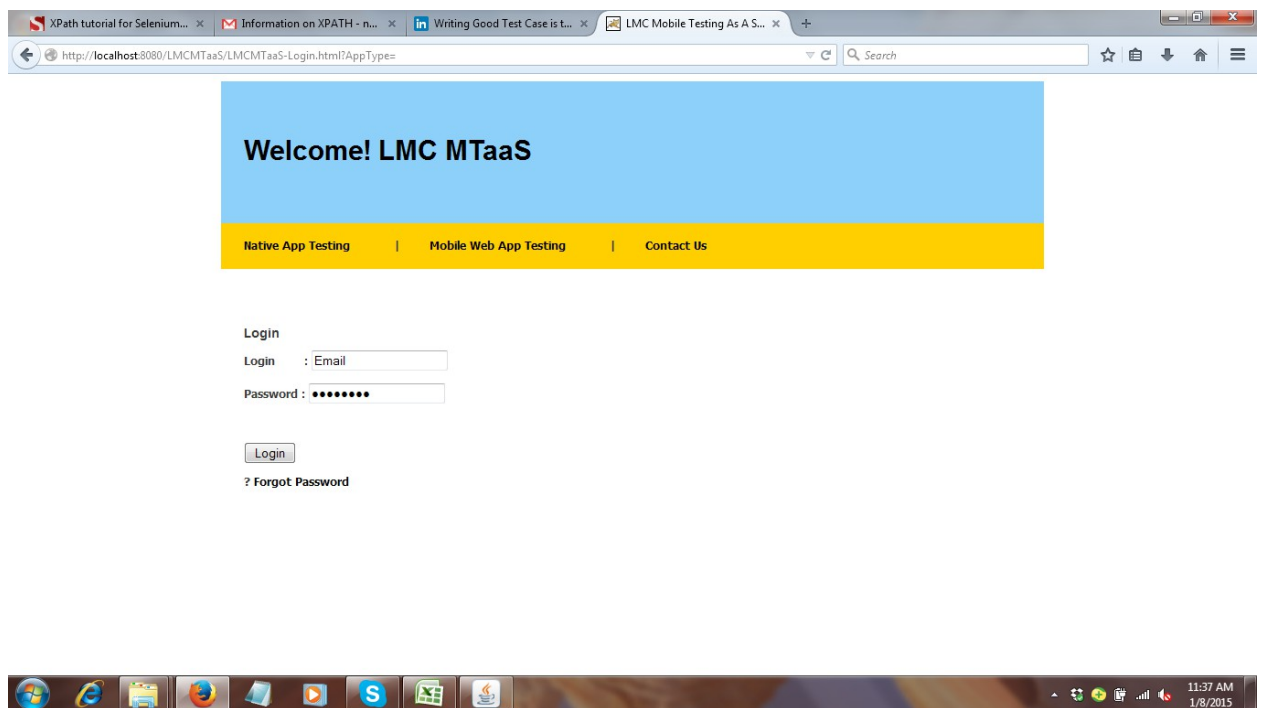


Fig 4.2.2 Login Page

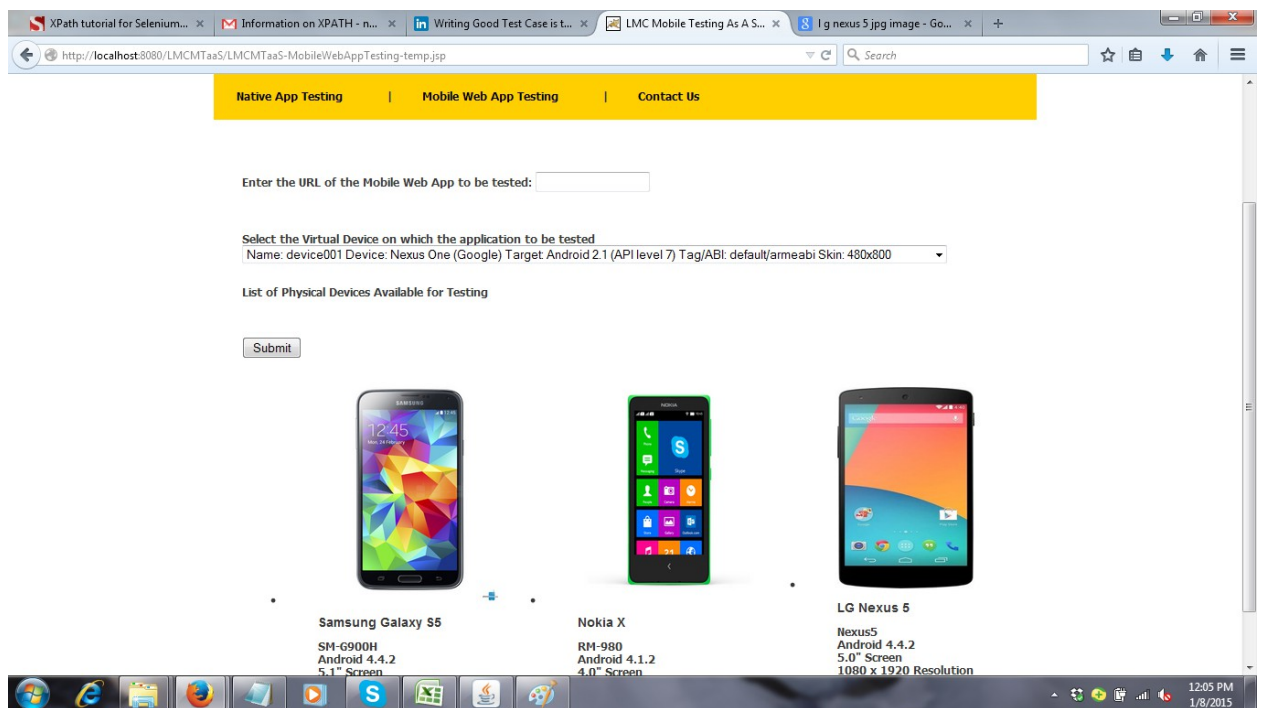


Fig 4.2.3 Mobile Web App Testing Page

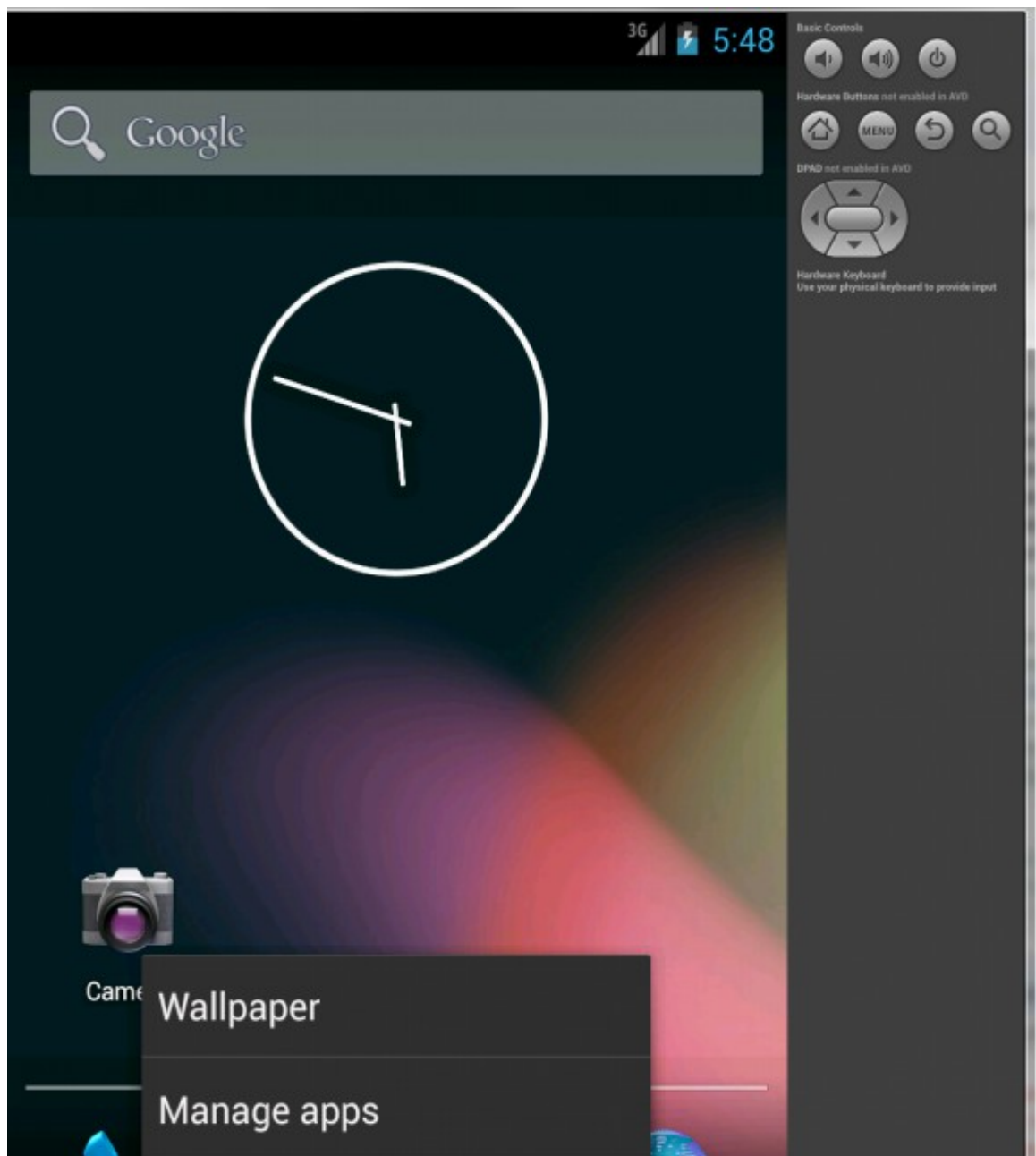


Fig 4.2.4 Automatic Launching of Emulator

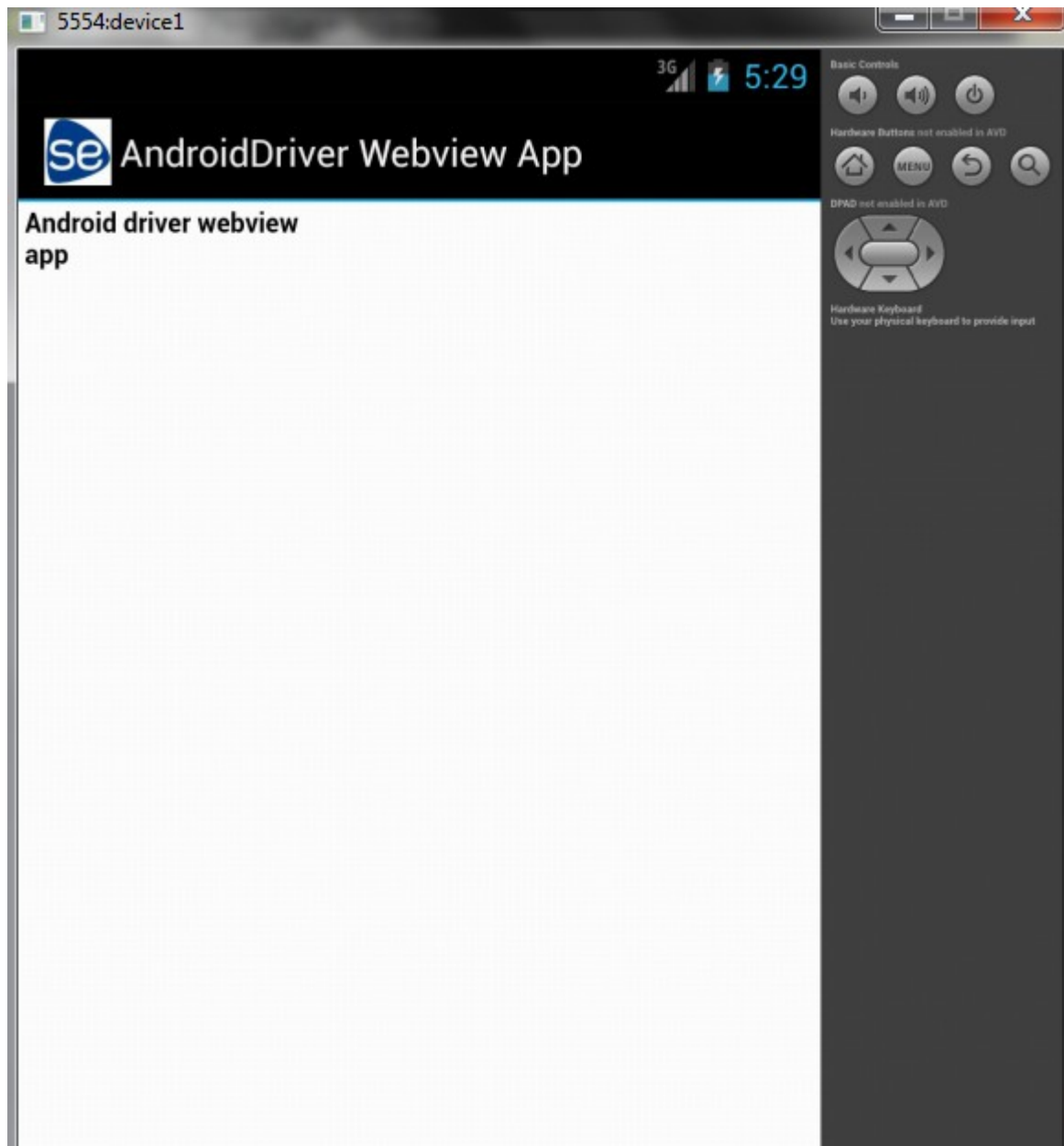


Fig 4.2.5 Launching of Selenium Web Driver

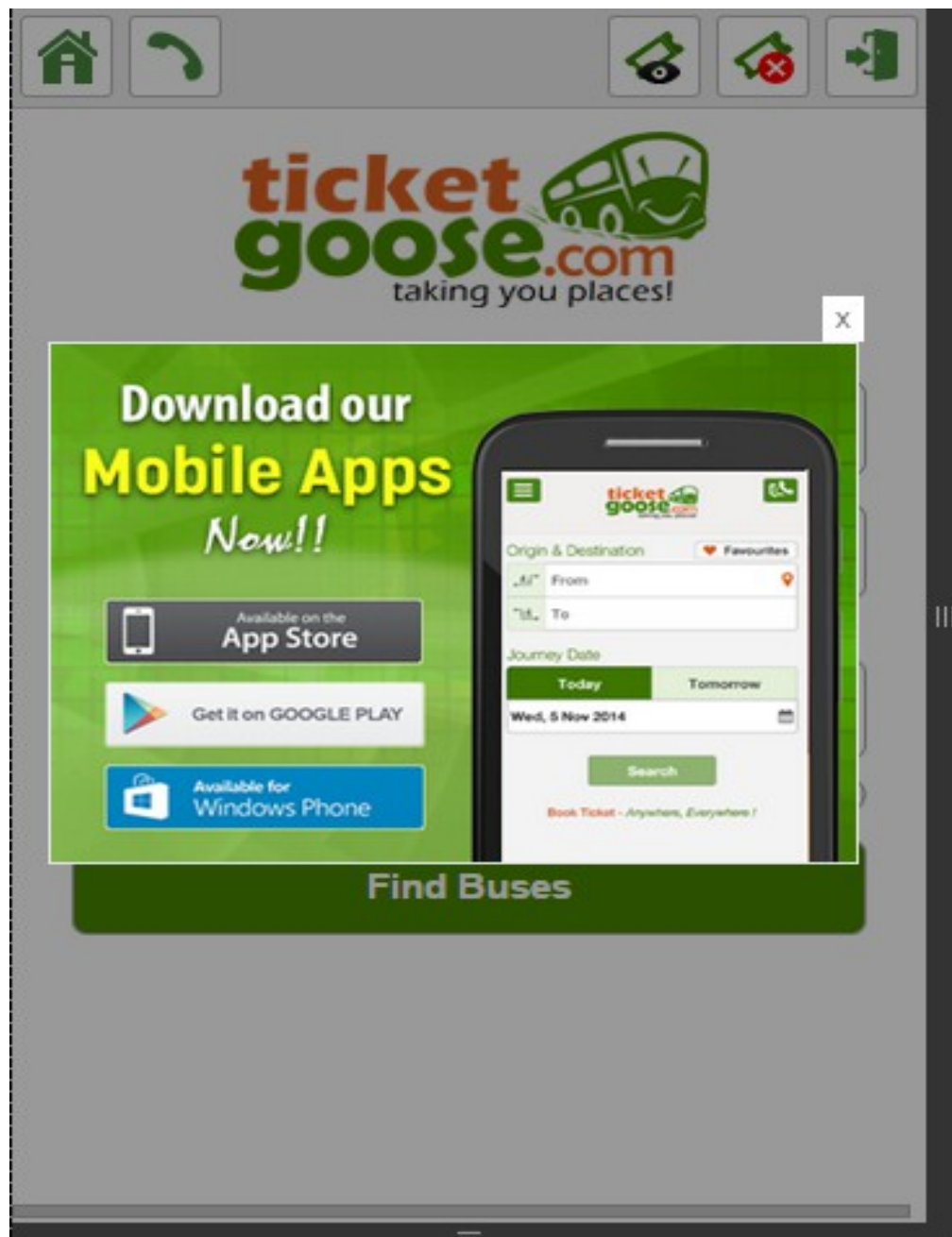


Fig 4.2.6 Ad Displayed



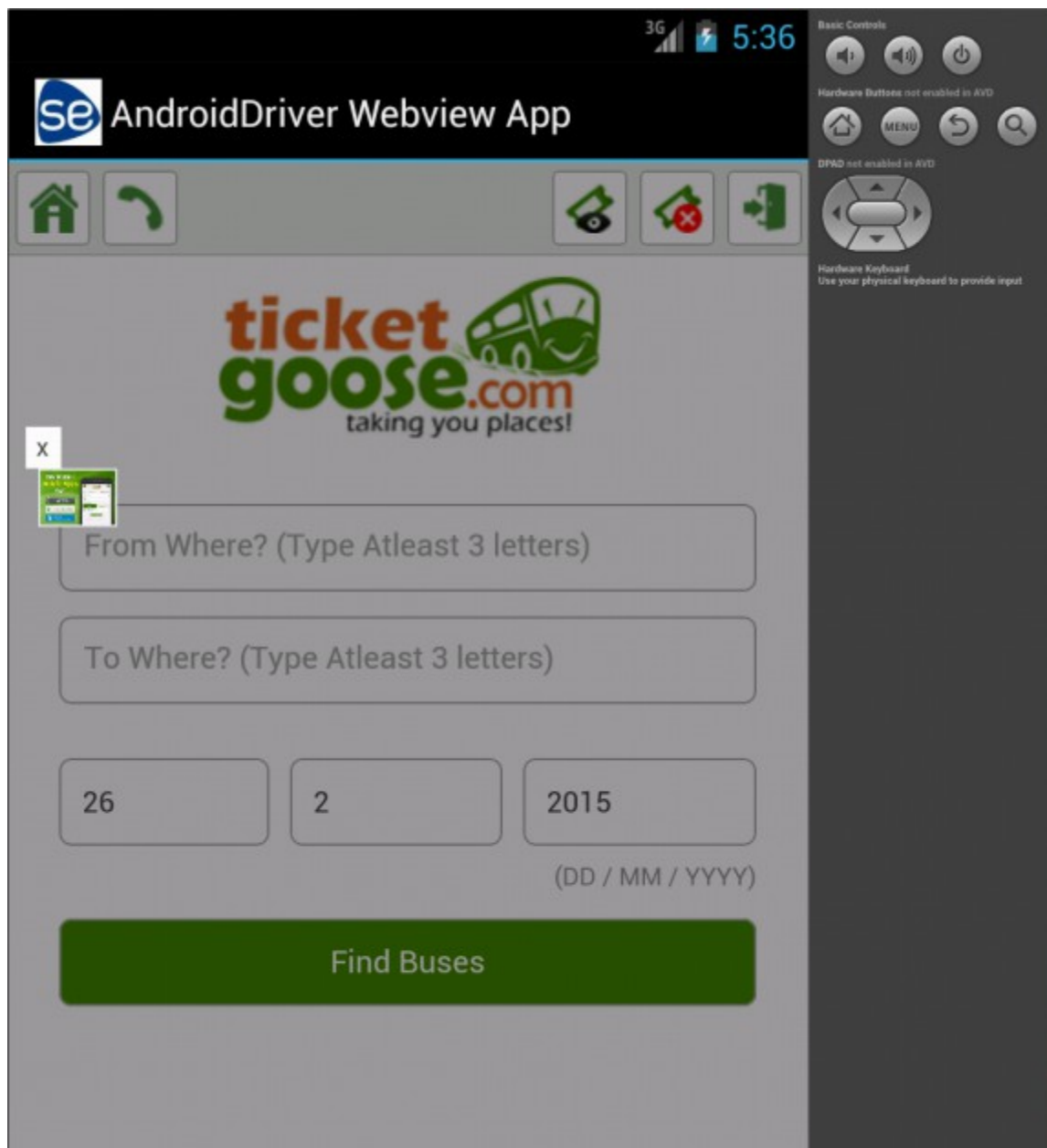


Fig 4.2.7 Ad Blocked Automatically



Fig 4.2.8 Automatic Click and dropdown menu displayed for From Station

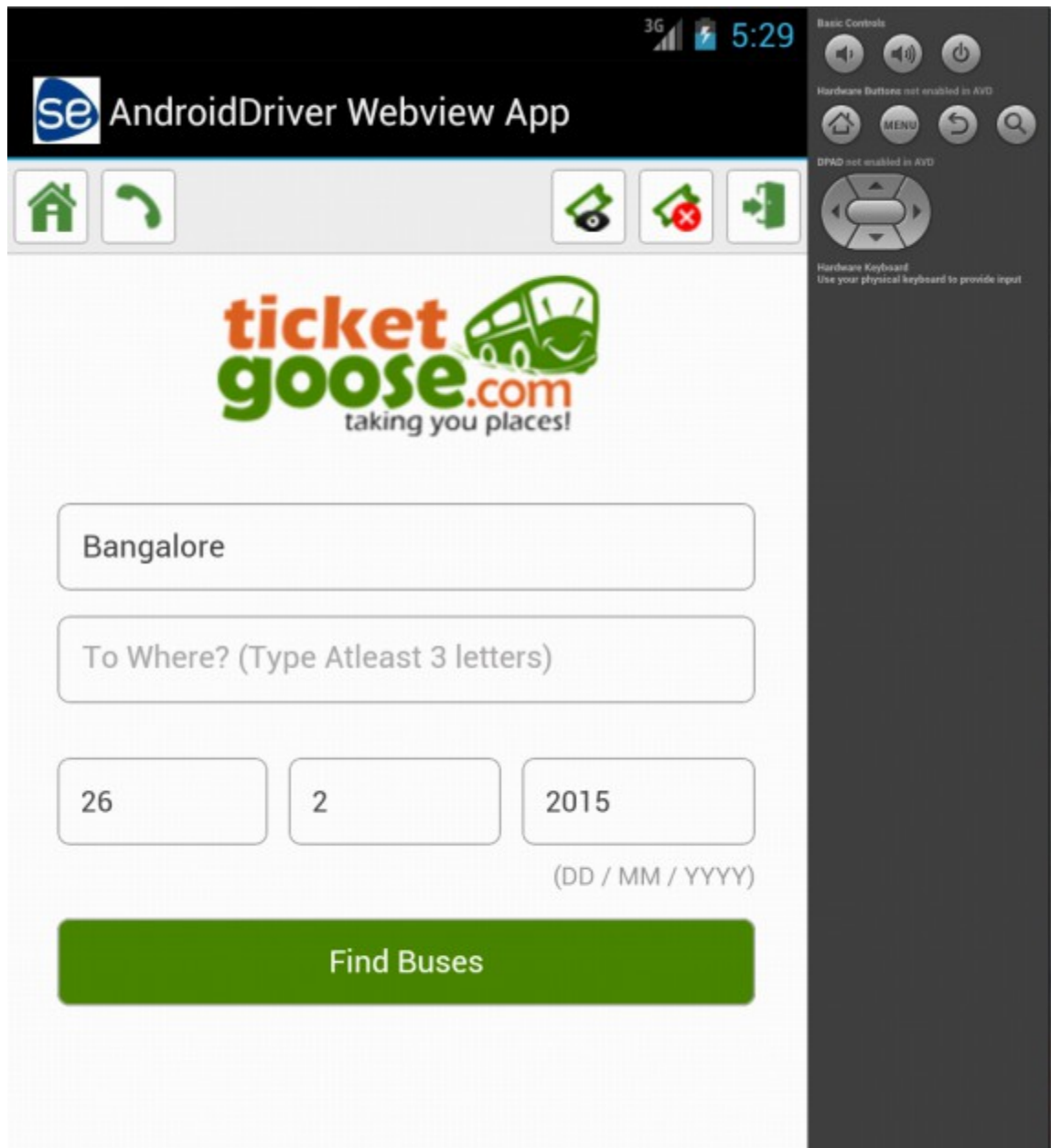


Fig 4.2.9 From Station selected from the dropdown and displayed



Fig 4.2.10 Automatic Click and dropdown menu displayed for To Station

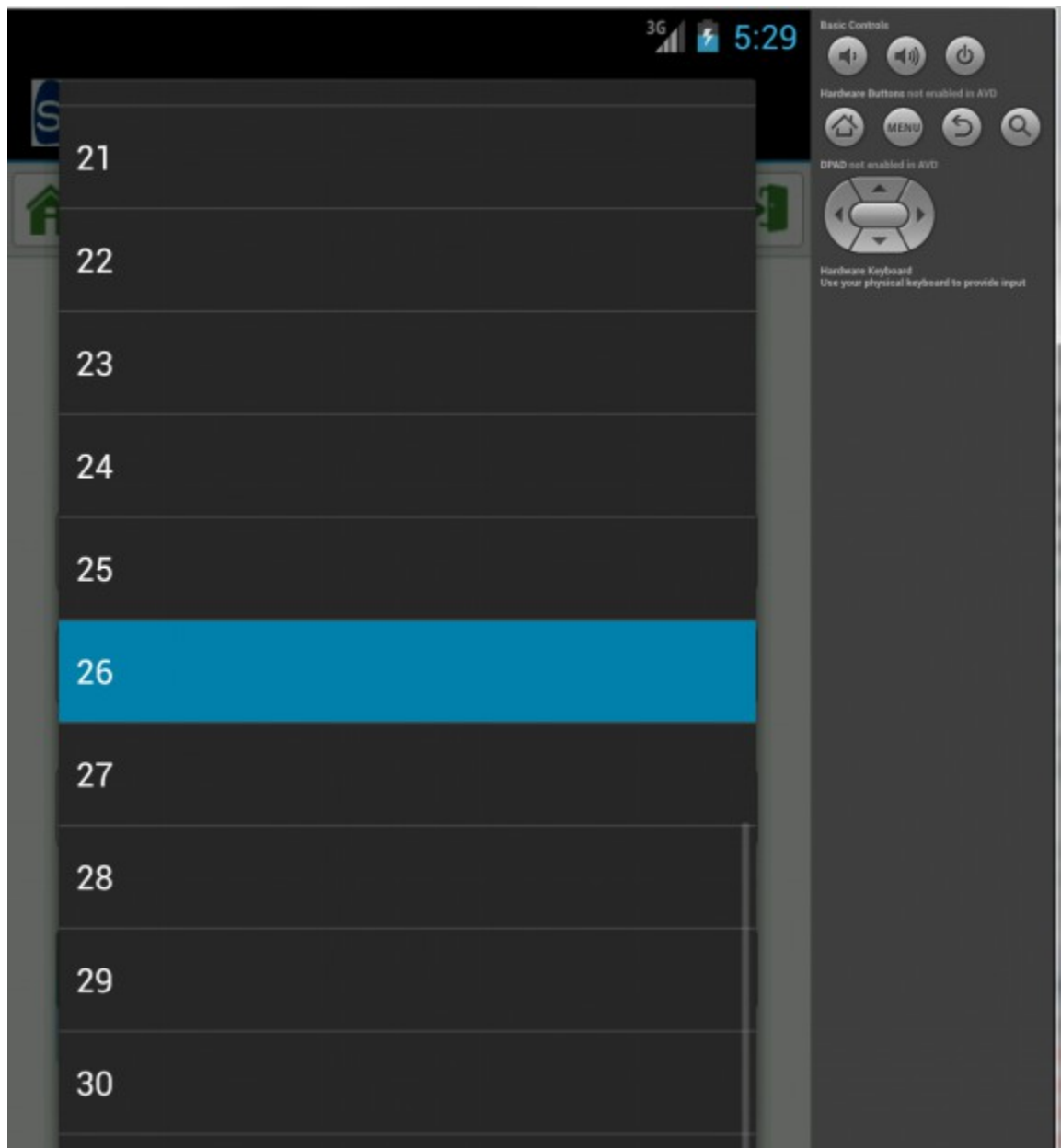


Fig 4.2.11 Automation selection of the Day from the dropdown list

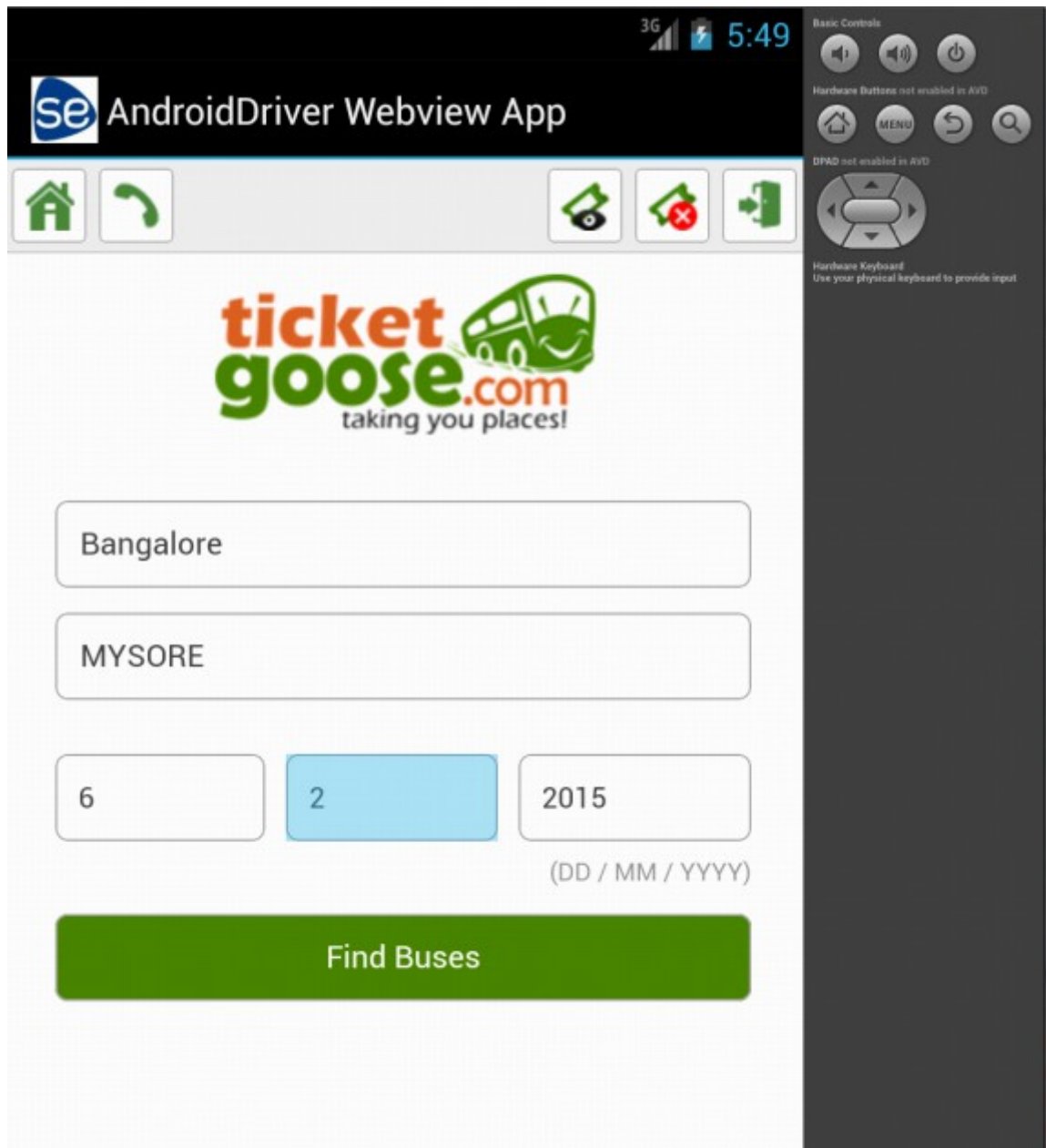


Fig 4.2.12 Automatic Click on the Month

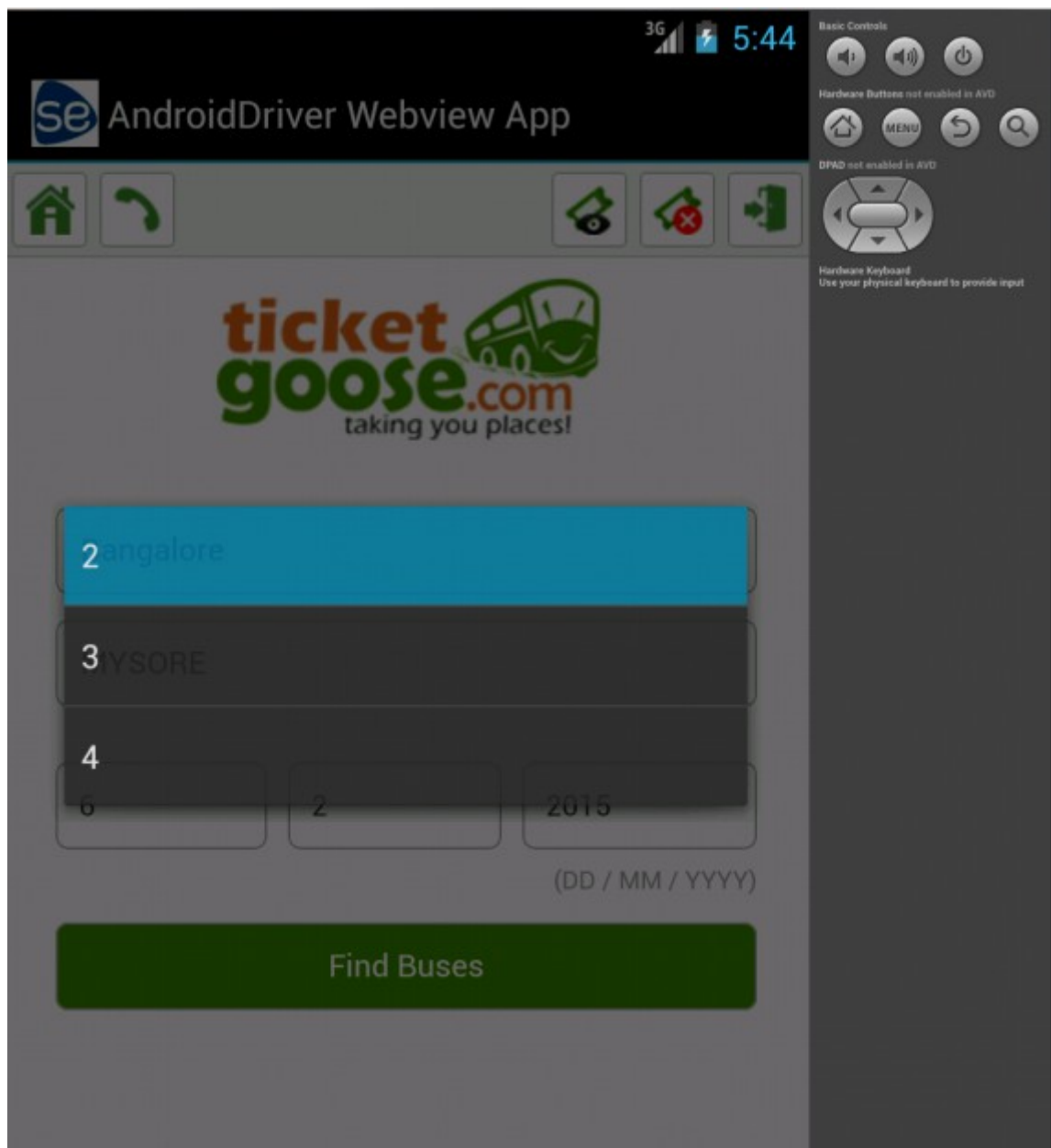


Fig 4.2.13 Automation selection of the Month from the dropdown list

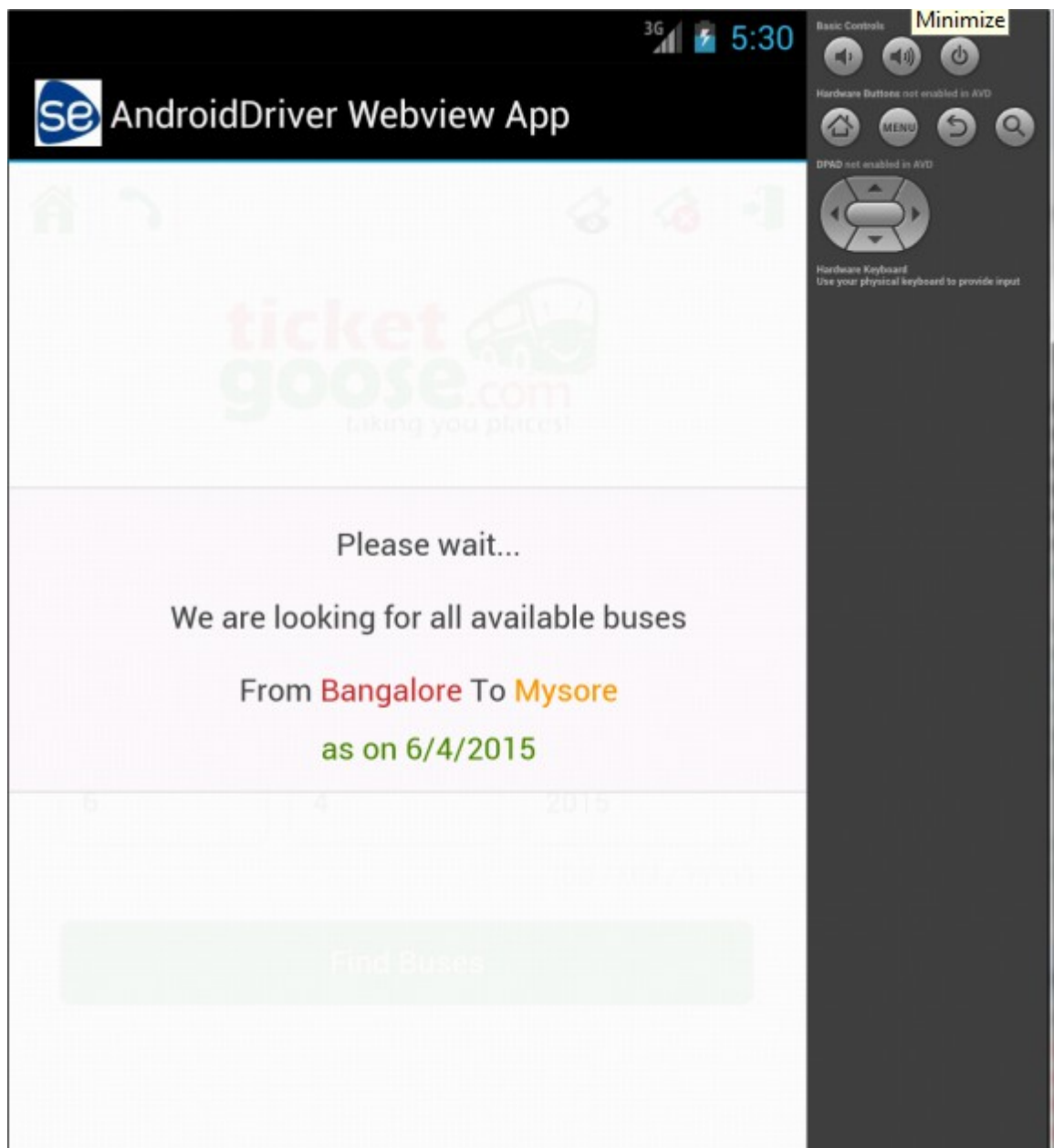


Fig 4.2.14 Result in process



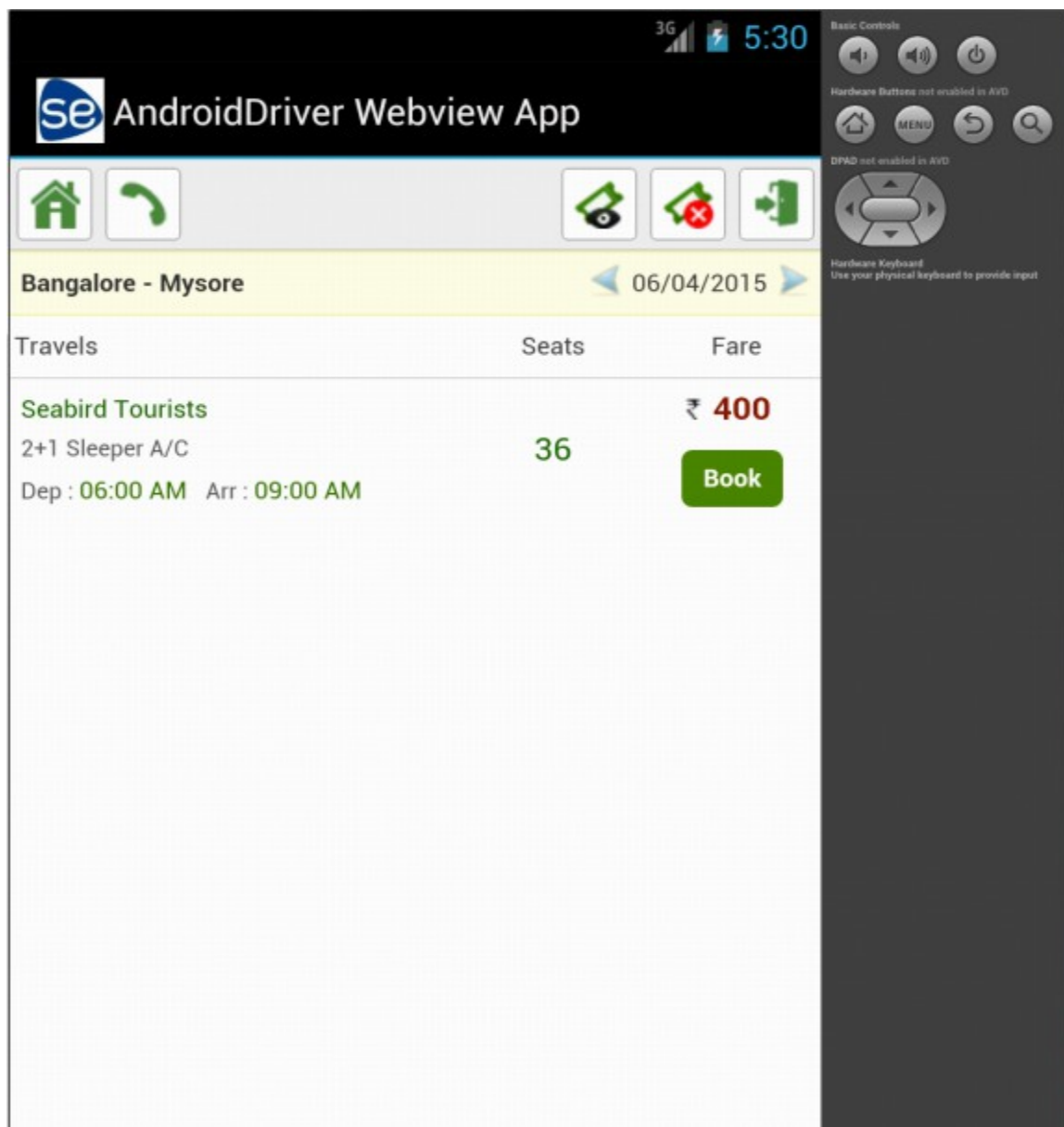


Fig 4.2.15 Result Displayed Automatically

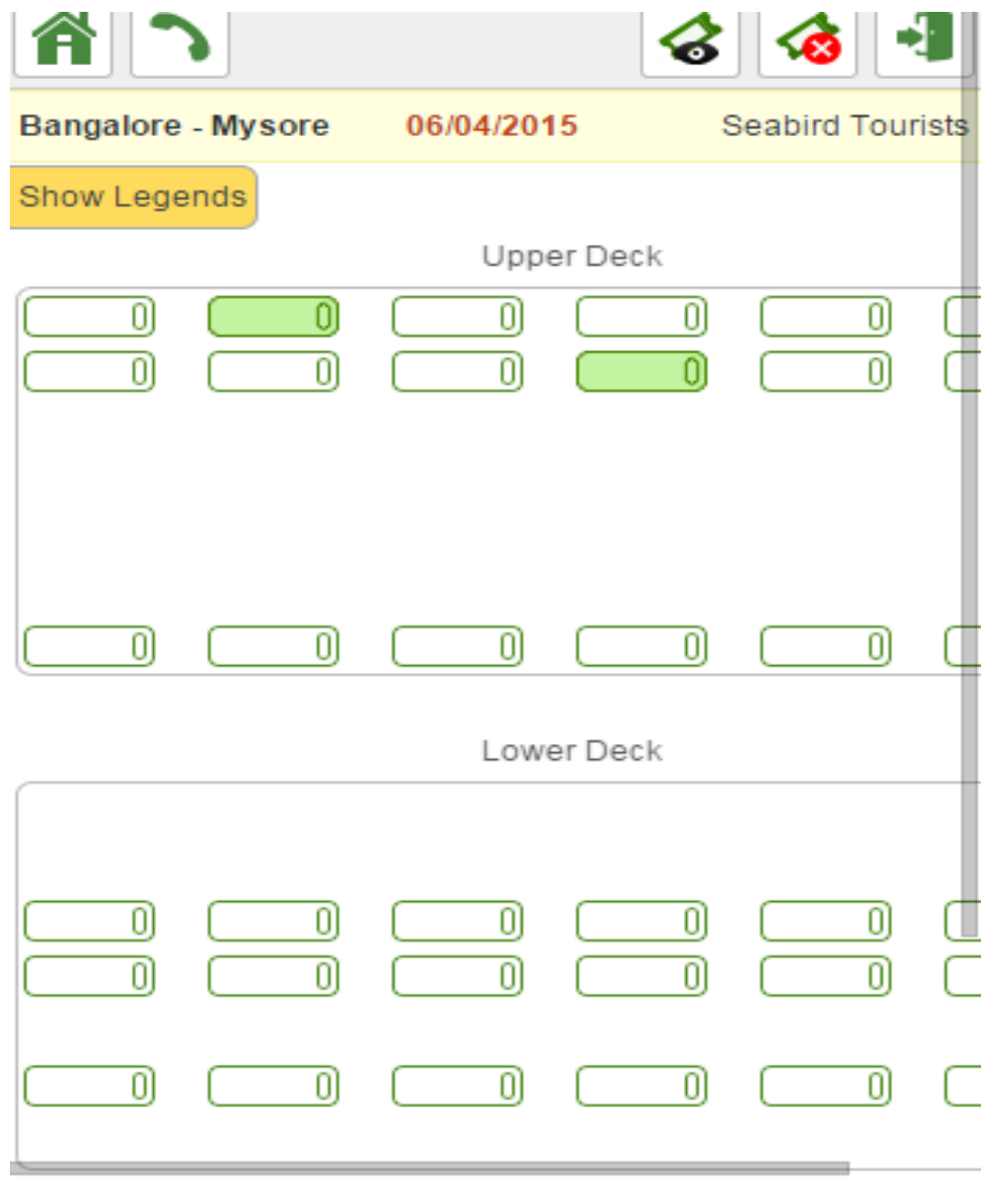


Fig 4.2.16 Automatic selection of Upper Deck

The image shows a mobile application interface with a header bar containing icons for home, back, and other functions. Below the header, there is a yellow bar with the text "Bangalore - Mysore", "06/04/2015", and "Seabird Tourists". A "Show Legends" button is located below the yellow bar. The main content area is divided into two sections: "Upper Deck" and "Lower Deck". Each section contains a grid of buttons, some of which are highlighted in green.

**Upper Deck**

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

**Lower Deck**

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Fig 4.2.17 Automatic selection of Lower Deck

The screenshot displays a mobile application interface. At the top, there are two rows of green rectangular buttons, each containing a white '0'. Below these is a section titled 'Lower Deck'. A dropdown menu is open, showing a list of boarding locations and times. The first item is 'Boarding At'. The list includes: 'Madiwala 05:30 am', 'Dairy Circle 05:40 am', 'Lalbagh 05:45 am' (which is highlighted in blue), 'Anand Rao Circle 06:00 am', 'R R Nagar 06:10 am', 'Kengeri Satellite Town(Slv Travels) 06:20 am', and 'Kengri PoliceStation 06:30 am'. Below the dropdown list is a green button labeled 'Continue'.

0 0 0 0 0 0

0 0 0 0 0 0

Lower Deck

Boarding At

Madiwala 05:30 am

Dairy Circle 05:40 am

**Lalbagh 05:45 am**

Anand Rao Circle 06:00 am

R R Nagar 06:10 am

Kengeri Satellite Town(Slv Travels) 06:20 am

Kengri PoliceStation 06:30 am

Lalbagh 05:45 am

**Continue**

Fig 4.2.18 Automatic selection of the Boarding Place and Time from the Dropdown list

**Bangalore - Mysore**    **06/04/2015**    Seabird Tourists

**Traveler Details**

neha.12katti@gmail.c    Mobile Number

☐ Enter all passengers name and age

Name    Age

<b>Male</b> <b>F</b>	Seat No : 6U	₹ 400
<b>Male</b> <b>F</b>	Seat No : 11U	₹ 400
<b>Male</b> <b>F</b>	Seat No : 3L	₹ 400
<b>Male</b> <b>F</b>	Seat No : 7L	₹ 400

**Payment Option**

☐ I accept the [Cancellation Policy](#) and [Terms & Conditions](#)

Total Fare : 1600.00  
**Grand Total : 1600.00**

**Continue**

Fig 4.2.19 Automatic entry of Email-id

**Bangalore - Mysore**    **06/04/2015**    Seabird Tourists

**Traveler Details**

neha.12katti@gmail.c    9747383733

☐ Enter all passengers name and age

Name    Age

<b>Male</b>	<b>F</b>	Seat No : 6U	₹ 400
<b>Male</b>	<b>F</b>	Seat No : 11U	₹ 400
<b>Male</b>	<b>F</b>	Seat No : 3L	₹ 400
<b>Male</b>	<b>F</b>	Seat No : 7L	₹ 400

**Payment Option**

☐ I accept the [Cancellation Policy](#) and [Terms & Conditions](#)

Total Fare : 1600.00

**Grand Total : 1600.00**

**Continue**

Fig 4.2.20 Automatic entry of Mobile number

**Bangalore - Mysore**      **06/04/2015**      Seabird Tourists

**Traveler Details**

neha.12katti@gmail.c      9747383733

☐ Enter all passengers name and age

APOORVA      Age

<b>Male</b>	<b>F</b>	Seat No : 6U	₹ 400
<b>Male</b>	<b>F</b>	Seat No : 11U	₹ 400
<b>Male</b>	<b>F</b>	Seat No : 3L	₹ 400
<b>Male</b>	<b>F</b>	Seat No : 7L	₹ 400

**Payment Option**

☐ I accept the [Cancellation Policy](#) and [Terms & Conditions](#)

Total Fare : 1600.00

**Grand Total : 1600.00**

**Continue**

Fig 4.2.21 Automatic entry of the Name

**Bangalore - Mysore**    **06/04/2015**    Seabird Tourists

**Traveler Details**

neha.12katti@gmail.c    9747383733

☐ Enter all passengers name and age

APOORVA    20

<b>Male</b> <b>F</b>	Seat No : 6U	₹ 400
<b>Male</b> <b>F</b>	Seat No : 11U	₹ 400
<b>Male</b> <b>F</b>	Seat No : 3L	₹ 400
<b>Male</b> <b>F</b>	Seat No : 7L	₹ 400

**Payment Option**

☐ I accept the [Cancellation Policy](#) and [Terms & Conditions](#)

Total Fare : 1600.00

**Grand Total : 1600.00**

**Continue**

Fig 4.2.22 Automatic entry of the Age and selection of the Gender



Bangalore - Mysore 06/04/2015 Seabird Tourists

**Traveler Details**

neha.12katti@gmail.c 9747383733

☐ Enter all passengers name and age

APOORVA 20

<b>M</b> Female	Seat No : 6U	₹ 400
<b>M</b> Female	Seat No : 11U	₹ 400
<b>M</b> Female	Seat No : 3L	₹ 400
<b>M</b> Female	Seat No : 7L	₹ 400

**Payment Option**






☐ I accept the [Cancellation Policy](#) and [Terms & Conditions](#)

Total Fare : 1600.00

**Grand Total : 1600.00**

**Continue**

Fig 4.2.23 Automatic selection of 'continue' button without accepting the 'Terms and conditions' checkbox



Bangalore - Mysore06/04/2015Seabird Tourists

Accept terms and condition

**Traveler Details**

neha.12katti@gmail.c

9747383733

☐ Enter all passengers name and age

APOORVA

20

**M** Female

Seat No : 6U

₹ 400

**M** Female

Seat No : 11U

₹ 400

**M** Female

Seat No : 3L

₹ 400

**M** Female

Seat No : 7L

₹ 400

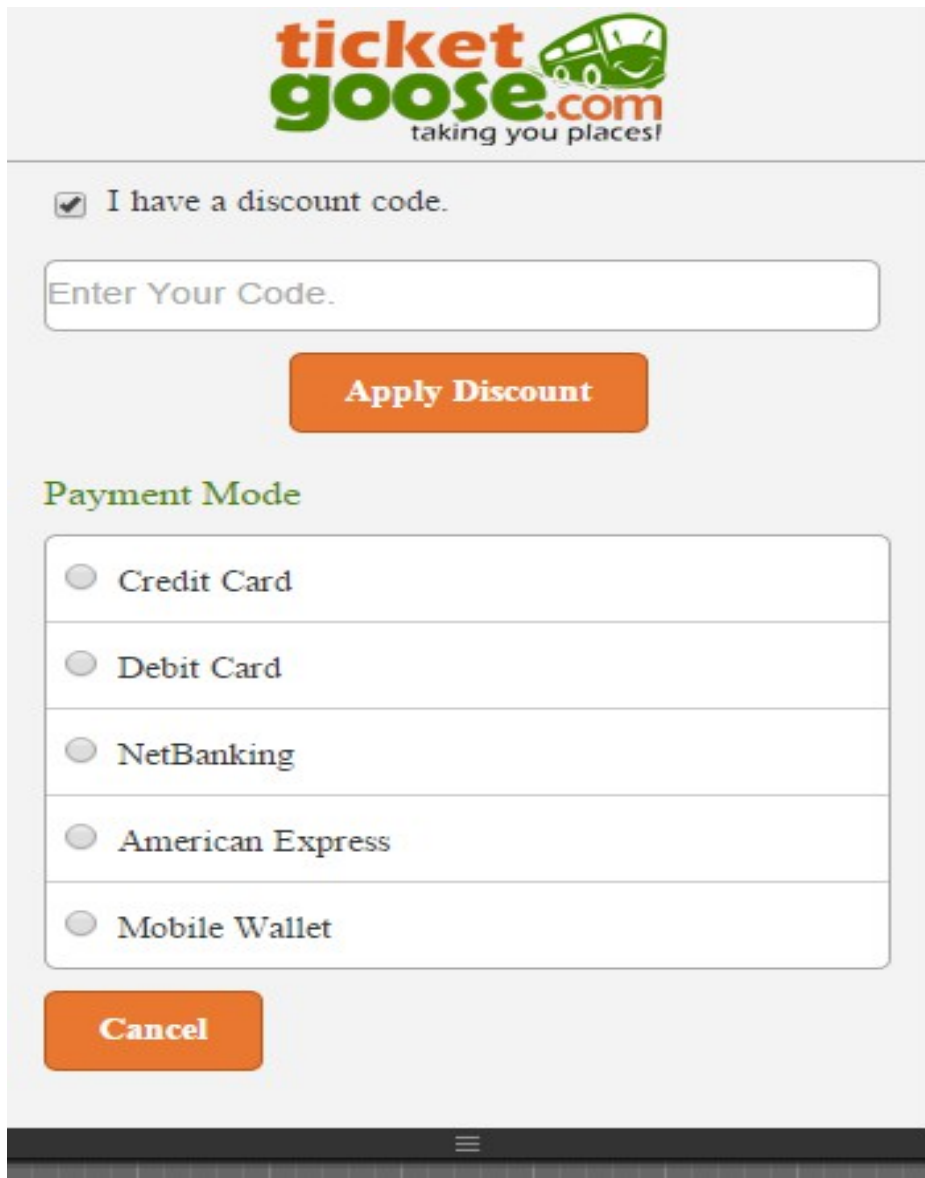
**Payment Option**

☐ I accept the [Cancellation Policy](#) and [Terms & Conditions](#)

Total Fare : 1600.00

**Grand Total : 1600.00**

Fig 4.2.24 Error message displayed



The screenshot displays the ticketgoose.com mobile application interface. At the top, the logo features the text "ticketgoose.com" in orange and green, with a green bus icon and the tagline "taking you places!" below it. Below the logo, there is a checkbox labeled "I have a discount code." which is checked. Underneath this is a text input field with the placeholder "Enter Your Code." and an orange button labeled "Apply Discount".

Below the discount section, the heading "Payment Mode" is displayed in green. This is followed by a list of five payment options, each with a radio button: "Credit Card", "Debit Card", "NetBanking", "American Express", and "Mobile Wallet". At the bottom of this list is an orange button labeled "Cancel". The interface is framed by a dark grey header and footer, with a hamburger menu icon visible in the footer.

Fig 4.2.25 Navigates to the next page on clicking 'continue'

## 5. TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

### 5.1 TYPES OF TESTING METHODOLOGIES

The following are the Testing methodologies:

- Unit Testing.
- Integration Testing.
- System Testing.
- Validation Testing.
- Verification Testing.

#### 5.1.1 UNIT TESTING

The procedure level testing is made first. By giving improper inputs, the errors occurred are noted and eliminated. Then the web form level testing is made. For example, storing the data into the table in the correct manner is checked. The dates are entered in wrong manner and checked. Wrong email-id and web site URL (Universal Resource Locator) is given and checked.

#### 5.1.2 INTEGRATION TESTING

Testing is done for each module. After testing all the modules, the modules are integrated and testing of the final system is done with the test data, specially designed to show that

the system will operate successfully in all its aspects conditions. Thus the system testing is a confirmation that all is correct and an opportunity to show the user that the system works.

The following are the types of Integration Testing:

- **Top Down Integration**

This method is an incremental approach to the construction of program structure. Modules are integrated by moving down through the control hierarchy, beginning with the main program module. The module subordinates to the main program module are incorporated into the structure in either a depth first or breadth first manner.

In this method, the software is tested from main module and individual stubs are replaced when the test proceeds downwards.

- **Bottom-up Integration**

This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to a given level is always available and the need for stubs is eliminated.

The bottom up approaches tests each module individually and then each module is module is integrated with a main module and tested for functionality.

### 5.1.3 SYSTEM TESTING

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

As a rule, system testing takes, as its input, all of the "integrated" software components that have successfully passed integration testing and also the software system itself integrated with any applicable hardware system. System testing is performed on the entire system in the context of a System Requirement Specification (SRS). System testing

tests not only the design, but also the behavior and even the believed expectations of the customer.

#### **5.1.4 VALIDATION TESTING**

The final step involves validation testing, which determines whether the software function as the user expected. The end-user rather than the system developer conduct this test most software developers as a process called “Alpha and Beta testing” to uncover that only the end user seems able to find.

The compilation of the entire project is based on the full satisfaction of the end users. In the project, validation testing is made in various forms. In registration form Email id, phone number and also mandatory fields for the user is verified.

#### **5.1.5 VERIFICATION TESTING**

Verification is a fundamental concept in software design. This is the bridge between customer requirements and an implementation that satisfies those requirements. This is verifiable if it can be demonstrated that the testing will result in an implementation that satisfies the customer requirements.

In-adequate testing or non-testing leads to errors that may appear few months later. This will create two problems:

- Time delay between the cause and appearance of the problem.
- The effect of the system errors on files and records within the system.

## 5.2 TEST CASES

**Test title:** Relaunch the App

NO	TEST CASES	TEST STEPS	EXPECTED RESULT
1	From each app screen, press the device's Home key, then re-launch the app from the All Apps screen.	Launch the application	The home page gets displayed.
		Click on the Home button	Device's Home button gets clicked
		Click on the Browser again	The application in the browser should be re launched
2	From each app screen, switch to another running app and then return to the app under test using the Recents app switcher.	Launch the application	The home page gets displayed.
		Click on the Home button	Device's Home button gets clicked
		Launch another running Application	Another application gets launched.
		Long Press on the Home button	The Home button Is long pressed
		List of all the recently opened apps should be displayed	All the apps recently opened and under process gets displayed
		Click on the browser under test	The browser under test gets clicked
		Should Navigate to the last viewed page on the browser	The last viewed page on the browser gets displayed.
3	From each app screen (and dialogs), press the Back button.	Scenario 1	
		Launch the application	The home page gets displayed.
		Press on the device's back button	Home screen gets displayed
		Scenario 2	
		Launch the application	The home page gets displayed.

		Browse through different pages in the app	The next page gets displayed
		Press on the device's back button	Previous page of the app gets displayed
4	From each app screen, rotate the device between landscape and portrait orientation at least three times.	Launch the application in the Browser	The application in the Browser gets launched
		Press Ctrl+Fn+F12	The Screen rotates (Portrait/Landscape)
		Press Ctrl+Fn+F12 again	The Orientation changes
		Repeat the same procedure thrice	The Orientation changes accordingly.
5	Switch to another app to send the test app into the background. Go to Settings and check whether the test app has any services running while in the background.	Launch the Application under test	The Application under test gets launched.
		Press on the device's Home button	The app under test goes to the background
		Switch to another app	Another app gets launched.
		Go to menu	All the options in the Menu will be displayed
		Click on Settings	All the options under Settings will be displayed
		Click on Apps	All the Apps (Downloaded, Running, On SD Card and all) are displayed.
		Click on Running	All the apps running in the background will be displayed along with their services.
		Click on the app under test	The app under test and its services are displayed.
6	Press the power button to put the device to sleep, then press the power button again to awaken the screen.	Press the Power button	The device goes to sleep
		Again press the power button	The device awakens and is in the unlocked state



7	Set the device to lock when the power button is pressed. Press the power button to put the device to sleep, then press the power button again to awaken the screen, then unlock the device.	Press the Power button	The device goes to sleep
		Again press the power button	The device awakens and is in the unlocked state
		Unlock the device	The device gets unlocked and ready to use
8	Observe in the notifications drawer ,all types of notifications that the app can display. Expand notifications where applicable, and tap all actions offered.	Drag the Notifications drawer from the top of the screen.	All the notifications from the app (e.g., TOI , Whatsapp etc) are displayed in the drawer.
		Tap on the notifications where the action needs to be started	That particular notification expands.
9	Examine the permissions requested by the app by going to Settings > App Info.	Click on Settings	All the options under Settings gets displayed.
		Click on Apps	All the apps installed in the device gets displayed.
		Click on the particular app whose permissions have to be examined.	The permissions are displayed

### Test Title : Standard design

1	App does not redefine	Open an Application	The Application is opened.
---	-----------------------	---------------------	----------------------------

	the expected function of a system icon (such as the Back button).	Browse through different pages in the app	The next page gets displayed
		Press on the device's back button	The previous page is displayed
		Press on the device's Home button	The device's Home page gets displayed
		Press on the back button defined inside the app	Previous page of the app gets displayed same as when the device's Back button is pressed
		Press on the Home button defined inside the app	Home page gets displayed same as when the device's Home button is pressed
2	App does not replace a system icon with a completely different icon if it triggers the standard UI behavior.	Open an Application	The Application is opened.
		Browse through different pages in the app	The next page gets displayed
		Press on the device's back button	The previous page is displayed
		Press on the device's Home button	The device's Home page gets displayed
		Press on the back button defined inside the app which has the same icon as the device's back button	Previous page of the app gets displayed same as when the device's Back button is pressed
3	If the app provides a customized version of a standard system icon, the icon strongly resembles the system icon and	Press on the Home button defined inside the app which has the same icon as the device's Home button	Home page gets displayed same as when the device's Home button is pressed
		Open an Application	The Application is opened.
		Browse through different pages in the app	The next page gets displayed
		Press on the device's back button	The previous page is displayed

	triggers the standard system behavior.	Press on the device's Home button	The device's Home page gets displayed
		Press on the back button defined inside the app which has a customized version of the standard system icon	Previous page gets displayed same as when the device's back button is pressed(The customized version of the icon does not change the standard system behavior )
		Press on the Home button defined inside the app which has a customized version of the standard system icon	Home page gets displayed same as when the device's Home button is pressed(The customized version of the icon does not change the standard system behaviour ).
		If the Back button defined inside the app has a different icon from that of the device's Back button icon	No changes / Previous page of the app is not displayed
		If the Home button defined inside the app has a different icon from that of the device's Home button icon	No changes / Home page of the app is not displayed
4	App does not redefine or misuse Android UI patterns, such that icons or behaviors could be misleading or confusing to users.	If Home button's behaviour is used in the Back button's icon inside the app	That particular Back button shouldn't function as per the behaviour assigned to it
		If Back button's behaviour is used in the Home button's icon inside the app	That particular Home button shouldn't function as per the behaviour assigned to it

## **5.3 TEST REPORTS**

## 6.CONCLUSION

### 6.1 DESIGN AND IMPLEMENTATION ISSUES

The journey from strategy to implementation for mobile testing has many issues and hurdles. Mobile application testing is both a critical and a complex component of mobile application development. It is crucial to have a clearly defined and well-developed mobile testing strategy and framework. The main components of a mobile application testing strategy include usability; performance; security; and functional and non functional testing across multiple platforms, devices and browsers.

A complete mobile testing strategy must also account for testing across differing network connection speeds and geographical locations, as well as address the use of Wi-Fi, 3G or 4G connections. Testing must confront such issues as screen resolution and brightness, CPU, memory and OS optimization. The mobile testing strategy must be geared to the architecture of the applications under test whether they are Web, mobile Web, native applications or hybrids. Finally, an organization must consider the test approach, primarily the [use of emulators](#) versus actual devices, or even [real user monitoring](#).

Emulation remains a strong candidate, at least for initial testing, with popular and largely inexpensive emulators widely available for both iPhone and Android devices. However, most emulators remain incomplete in important ways. Experienced testers looking for accurate and repeatable results might not fully trust an emulator.

The journey to implement the strategy involves finding vendors to support a defined test strategy which is very critical. Crowdsourcing, may be an option for certain types of applications. While crowdsourcing can provide a lot of data quickly, it may also create incorrect initial impressions of the application's quality and utility.

Implementation issue is mainly focussed on automation. [Automation](#) has to play a key role in any enterprise testing strategy. Whether for tracking test case scripts and results, collecting and analyzing real user data, or analyzing performance on the client device and load on the server, the job is just too large for a manual approach.

Many application builders are still content with incomplete testing or even no testing, in part because mobile testing is still in its infancy and can be difficult to do correctly. The options and tradeoffs might seem to present insurmountable complexity. Inexperienced project managers might be tempted to simply let customers and users provide primary feedback on features and bugs.

## 6.2 ADVANTAGES AND LIMITATIONS

### ADVANTAGES

- **Automated Software Testing Saves Time and Money**

Software tests have to be repeated often during development cycles to ensure quality. Every time source code is modified software tests should be repeated. For each release of the software it may be tested on all supported operating systems and hardware configurations. Manually repeating these tests is costly and time consuming. Once created, automated tests can be run over and over again at no additional cost and they are much faster than manual tests. Automated software testing can reduce the time to run repetitive tests from days to hours. A time savings that translates directly into cost savings.

- **Testing Improves Accuracy**

Even the most conscientious tester will make mistakes during monotonous manual testing. Automated tests perform the same steps precisely every time they are executed and never forget to record detailed results.

- **Increase Test Coverage**

Automated software testing can increase the depth and scope of tests to help improve software quality. Lengthy tests that are often avoided during manual testing can be run unattended. They can even be run on multiple computers with different configurations. Automated software testing can look inside an application and see memory contents, data tables, file contents, and internal program states to determine if the product is behaving as expected. Automated software tests can easily execute thousands of different complex test cases during every test run providing coverage that is impossible with manual tests. Testers freed from repetitive manual tests have more time to create new automated software tests and deal with complex features.

- **Automation Does What Manual Testing Cannot**

Even the largest software departments cannot perform a controlled web application test with thousands of users. Automated testing can simulate tens, hundreds or thousands of virtual users interacting with network or web software and applications.

- **Automated QA Testing Helps Developers and Testers**

Shared automated tests can be used by developers to catch problems quickly before sending to QA. Tests can run automatically whenever source code changes are checked in and notify the team or the developer if they fail. Features like these save developers time and increase their confidence.

## **Limitations**

- Proficiency is required to write the mobile automation test scripts.
- Debugging the mobile app specific test script is major issue. If any error is present in the test script, sometimes it may lead to deadly consequences.
- Test maintenance is costly in case of playback methods. Even though a minor change occurs in the GUI of any mobile platform, the test script has to be rerecorded or replaced by a new test script.

- Maintenance of test data files is difficult, if the test script tests more screens.
- Since the main function of the mobile testing framework is automation, testing is usually done at unit testing level. Automation testing requires some code to be written which when executed, automatically tests certain unit of code
- However, in case of system and integration testing, automation testing tools usually don't have the ability to integrate test cases and output the result of complete system testing and they are limited to unit testing only.
- Last but not the least, robust and reliable automated testing tools are not available freely in the market. Most of the free tools available in the market at the moment are not efficient enough to provide the latest mobile automated testing features.
- Another limitation of mobile automation testing is that automated test cases are not run on single machine of a developer or tester. To obtain correct results test case needs to be executed on some central repository where it can access all the parts of application developed by different developers.
- The Real live interactions cannot be performed with emulator alone .It is not possible to test the applications on a live network connectivity.
- Emulator just mimics the mobile device from various platforms and hence testing on the emulator cannot guarantee the stability of the application.
- Some of the interruption test scenarios may also not work properly as like in real handset to predict the actual behavior of the application.



### 6.3 FUTURE ENHANCEMENTS

- The current project scenario deals with testing of apps on an android emulator. One of the most important enhancement is to expand the framework to work on other platforms like apple and windows.
- The testing strategies can be expanded to be compatible with screen resolution of a tablet mobile screen according to the latest market trend.
- The testing strategies can have gestures specific module added along with various other characteristics of a mobile application.

## REFERENCES