

SECURE AND SMART PARKING SYSTEM BASED ON ANDROID APPLICATION

**Submitted in partial fulfillment of the requirements for the award of
Bachelor of Science degree in Computer Science**

By

NISHANTHI. H (Reg. No. 39290073)

RITSIA. B (Reg. No. 39290087)



**DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF COMPUTING
SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade “A” by NAAC
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600 119**

MARCH - 2022



SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC
(Established under Section 3 of UGC Act, 1956)
JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI- 600119
www.sathyabama.ac.in



**DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF COMPUTING**

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of H. NISHANTHI (39290073) and B. RITSIA (39290087) who carried out the project entitled "SMART PARKING SYSTEM BASED ON ANDROID APPLICATION" under my supervision from _____ to _____.

Internal Guide

Dr. KANIPRIYA

Head of the Department

Dr. L. LAKSHMANAN, M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, **NISHANTHI.H(Reg. No. 39290073) & RITSIA. B (Reg. No. 39290087)** hereby declare that the Project Report entitled "**SMART PARKING SYSTEM BASED ON ANDROID APPLICATION**" done by me under the guidance of Dr. **KANIPRIYA** is submitted in partial fulfillment of the requirements for the award of Bachelor of Science degree in Computer Science.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. SASIKALA, M.E., Ph.D.**, Dean, School of Computing and **Dr. S. VIGNESHWARI, M.E., Ph.D.**, and **Dr. L. LAKSHMANAN, M.E., Ph.D.**, Head of the Department, Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. KANIPRIYA** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Computer Science and Engineering who were helpful in many ways For the Completion of the project.

Abstract

Nowadays the traffic problem was rapidly increasing urban areas. So, we can find the solution for this problem. In this paper we proposed the one solution for reducing this Problem. So, in this paper focus on smart android-based parking control application, and which will helps find the nearest parking area of our current location. In this application will Helps the pay process of parking. After the car is being parked, a timer will start to count the Amount of time the car is spent in the parking lot. When the car owner wants to release the parking slot, he/she will take away the car and the timer will be stopped.

TABLE OF CONTENTS

CHAPTER. NO	TITLE	PAGE. NO
	ABSTRACT	5
1	INTRODUCTION	8
	1.1 OVERVIEW OF PROJECT	8
2	LITERATURE SURVEY	9
3	SCOPE OF THE PRESENT INVESTIGATION	12
	3.1 SCOPE AND OBJECTIVE	12
	3.2 SYSTEM REQUIREMENTS	12
	3.2.1 HARDWARE REQUIREMENETS	12
	3.2.2 SOFTWARE REQUIREMENTS	12
4	METHODOLOGY	
	4.1 EXISTING SYSTEM	13
	4.2 PROPOSED SYSTEM	13
	4.3 MODULE DESCRIPTION	13
	4.4 ARCHITECTURE DIAGRAM	15
	4.5 BLOCK DIAGRAM	16
5	OVERVIEW OF ANDROID	22
6	CONCLUSION AND FUTURE ENHANCEMENT	49
	7.1 CONCLUSION	49
	7.2 FUTURE ENHANCEMENT	49
	REFERENCES	49
7	APPENDIX	
	A. SOURCE CODE	50

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO
4.5	DATA FLOW DIAGRAM	17
4.5	USE CASE DIAGRAM	19
4.5	CLASS DIAGRAM	20
4.5	SEQUENCE DIAGRAM	21
6.1	LIFE CYCLE OF ACTIVITY	27
6.1	ARCHITECTURE OF JSP	28

CHAPTER 1

1. INTRODUCTION

1.1 OVERVIEW OF PROJECT

With a rapid increase in the vehicle usage in the recent past, finding and reaching a free parking slot has become time consuming and tedious, especially in the metro cities. This creates a necessity to introduce an automated system that allows users to book their spot just by making a few clicks! And with the advent of newer and newer technologies this has been made possible; with the help of a mobile-based app that helps you in locating a free parking slot and then guiding you all the way to the empty slot found, also eliminating the conventional physical currency payment; rather making the payment through an easy-to-use mobile application.

This paper describes in detail how such a system can be implemented. It does not make use of any sophisticated devices or gadgets thereby ensuring it is economical to implement at a larger scale too! The prime aim of this research is to develop an application that allows the user to book a parking slot for him using his smart-phone and also make the payment online using internet banking. Besides this system would be very handy for large parking systems where it is difficult to locate the allotted slot; the user can see the shortest route to the allotted slot on the mobile app itself and hence reach there easily. The best part of the system lies in the fact that the entire system is connected to the cloud, so user can access the database, know the current status of the parking and book a slot for him accordingly from anywhere in the world. Also, the database keeps track of the frequent users and accordingly offers discounts and offers.

CHAPTER 2

2. LITERATURE SURVEY

1. D. J. Bonde “Automated car parking system commanded by android application” in Proc. IEEE

Conf.,03-05, Jan 2012

The aim of this project is to automate the car and car parking as well. A miniature model of an automated car

parking system that can regulate and manage number of cars that can be parked in given space at any given time based

on the availability of parking slot. Automated parking is a method of parking and existing cars using sensing device.

The entering and leaving to the lot is commanded by an android application [1].

2. Yanfeng Geng, Christos G. Cassandras, “A new “Smart Parking” system Infrastructure and Implementation”, Science Direct, Social and Science Behavioral sciences, 1278-1287 ,2012

Smart Parking adopts the basic structure of PGI systems. In addition, such a system includes Driver Request

Processing Centre (DRPC) and a Smart Parking Allocation Centre (SPAC). The Parking Resource Management

Centre(PRMC) collects and updates all real time parking information and disseminates it via internet. The DRPC

gathers driver parking requests and real time information(i.e., car location), keep track of driver allocation status, and

sends back the assignment result to driver. The Smart Parking Allocation center makes assignment decisions and

allocates and reserve parking spots for driver.[2]

3. M. A. R. Sarkar, A. A. Rokoni, M. O. Reza, M. F. Ismail, “Smart parking system with image processing facility”, I. J. Intelligent System and Application, 41-47, 2012.

Smart Parking Systems obtain information about available parking spaces, process it and then place the car at that position. A prototype of the parking assistance system based on the proposed architecture was constructed. The effective circular design is introduced having rack-pinion special mechanism which is used to lift and place the car in certain position[5].

4. M. M. Rashid, A. Musa, M. AtaurRahman, and N. Farahana, A. Farhana “Automatic Parking Management System and Parking Fee Collection Based on Number Plate Recognition” International Journal of Machine Learning and Computing, 93-98, 2012.

This paper discussed on automatic parking system and electronic parking fee collection based on vehicle number plate recognition. The aim of this research is to develop and implement an automatic parking system that will increase convenience and security of the public parking lot as well as collecting parking fee without hassles of using magnetic card. The auto parking system will able to have less interaction of humans and use no magnetic card and its devices. In additions to that, it has parking guidance system that can show and guide user towards a parking space. The system used image processing of recognizing number plates for operation of parking and billing system. Overall, the systems run with pre-programmed controller to make minimum human involvement in parking system and ensure access control in restricted places.

5. R. Yusnita, FarizaNorbaya, and Norazwinawati Basharuddin “ Intelligent Parking Space Detection System Based on Image Processing”, Internation Journal of Innovation, Management and Technology, 232-253, 2012.

This paper aims to present an intelligent system for parking space detection based on image processing technique that capture and process the brown rounded image drawn at parking lot and produce the information of the empty car parking spaces. It will be display at the display unit that consists of seven segments in real time. The seven segments display shows the number of current available parking lots in the parking area. This proposed system, has been developed in software and hardware platform.

CHAPTER 3

3.2 SCOPE AND OBJECTIVE:

SCOPE:

The proposed system can further be upgraded by including features like sensors and CCTV camera so that the latest status of the parking can be checked by the user through the same application, rather than providing statistical data (in the current system). Also, navigation facility can be added to guide the **user** at each step to make sure that the user reaches the destination slot in the minimum possible time

OBJECTIVE:

Smart parking system is designed successfully and in order to provide the android Application and secure system

3.2 SYSTEM ARCHITECTURE

3.2.1 HARDWARE REQUIREMENTS:

Hard disk: 250GB or above

Processor: i3 or above

Ram: 4GB or above

3.2.2 SOFTWARE REQUIREMENTS

Operating system: windows 7 or above (64bit)

Java: 1.8

Net beans: 8.0.2

MySQL: 5.0

Eclipse

CHAPTER

4.METHODOLOGY

4.1 Existing Systems:

Radio-frequency identification (RFID) Reader Technique.

GSM Based parking System.

Cloud based parking technology.

Intelligent parking technology.

4.2 Proposed System:

User authentication is one of the prime factors of our proposed system.

Android application

To make sure the user is authentic; we will use their mobile numbers to register.

Advantages:

Software based

Does not require hardware in front of door

Counter inside the application

User friendly application

Disadvantages:

Time consuming

Requires hardware and entire process was complex

Is not a user friendly.

TECHNOLOGIES:

ANDROID

4.3 MODULE DESCRIPTION

- Registration
- Admin
- Availability status
- Web service

REGISTRATION

Initially, the user has to register his details with the application for the first time. This is a onetime registration. The user has to enter details like username, gender, phone number and email-id. All this data will be stored on server. Booking for slots mandatorily has to be done four hours prior to arrival.

ADMIN

Whenever a new user registers with the application, the record will be stored in the server-side database. When the registered user selects the location, immediately server receives the client's request. After receiving the request for the desired location, server processes the related information and responds accordingly. Furthermore, the administrator has direct option to view user details and slot details stored on the server direct via the application.

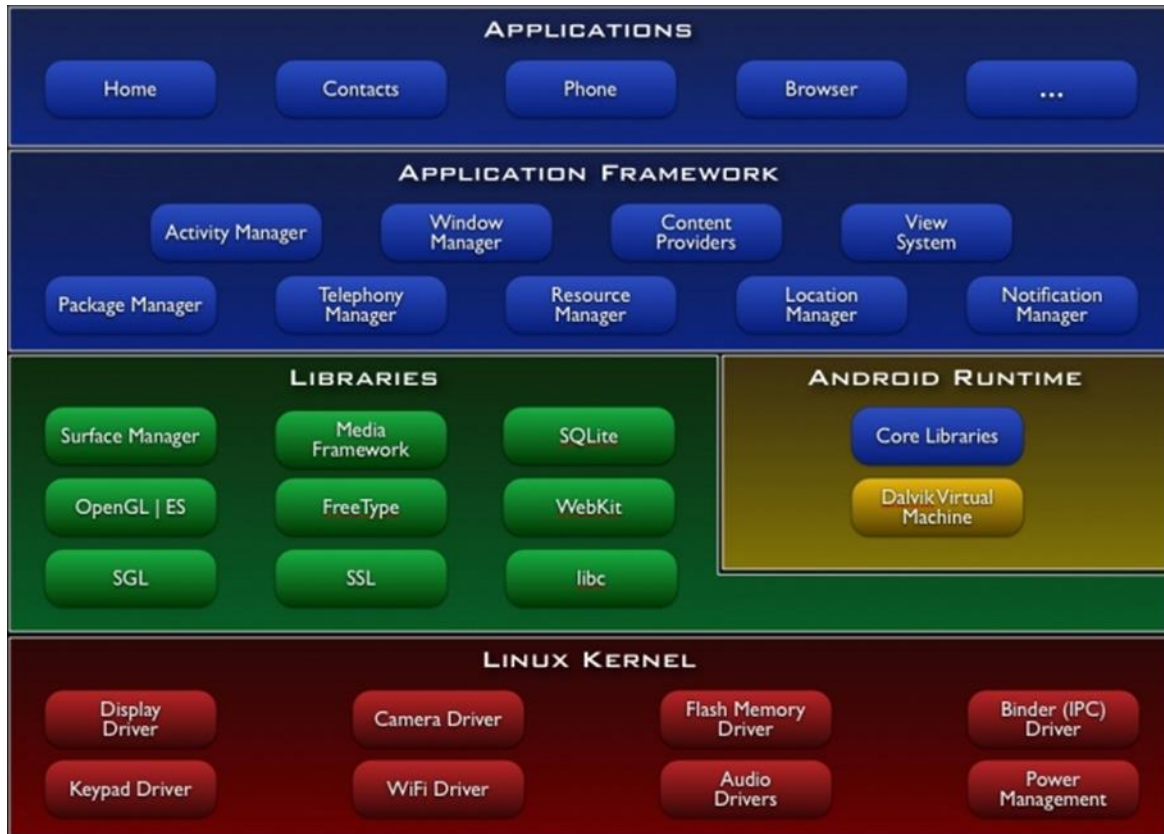
AVAILABILITY STATUS

Slots Based on the vehicle selected availability of the empty slots will be displayed along with the total slots reserved for that vehicle type. Color coding is used to indicate empty v/s reserved slots. Green indicates empty slots and Red indicates that currently there are no empty slots for reservation.

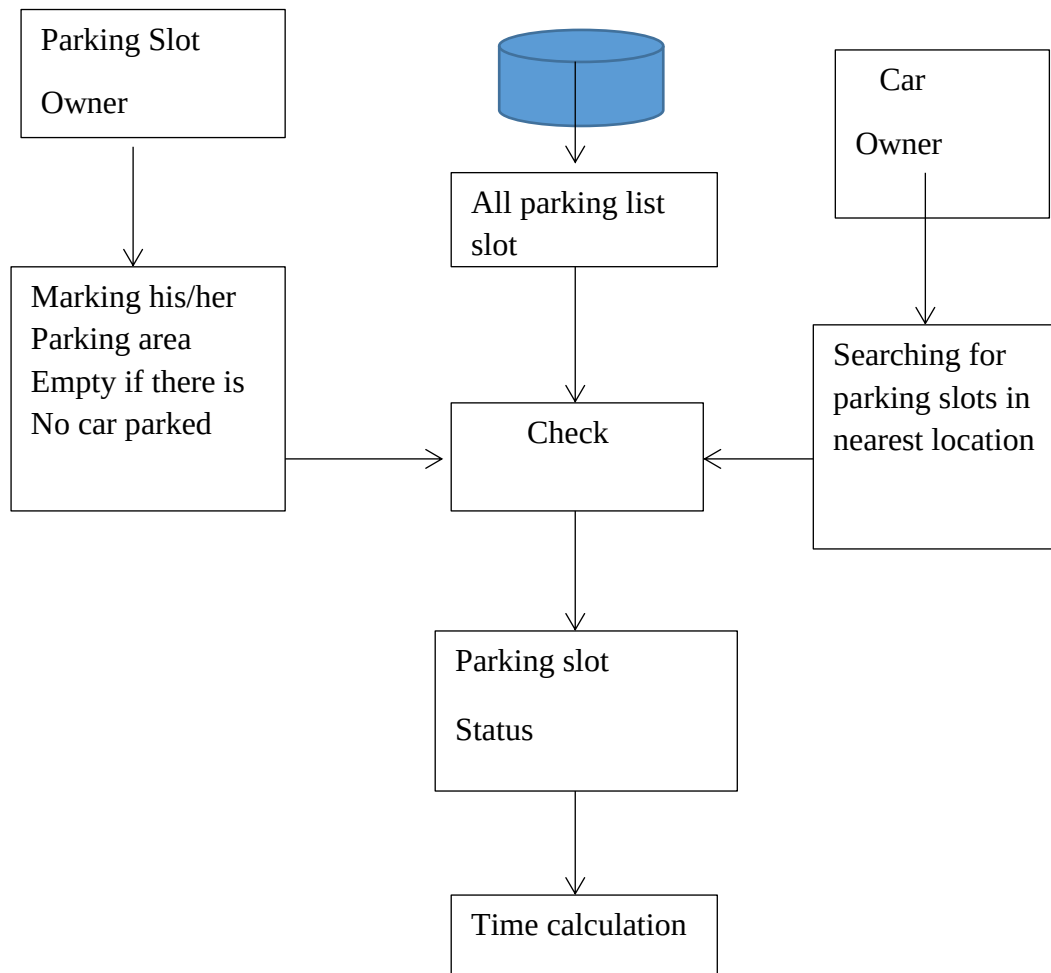
WEB SERVICE

Web service is like connecting android application and server. Server should run 24 hours and it has to give all the details to database which data's we are getting from users. Then using SOAP protocol, we can connect android application to server. If we are using SOAP protocol, it will collect all the details from android application and it will send to server.

4.4 ARCHITECTURE DIAGRAM



4.5 Block Diagram:

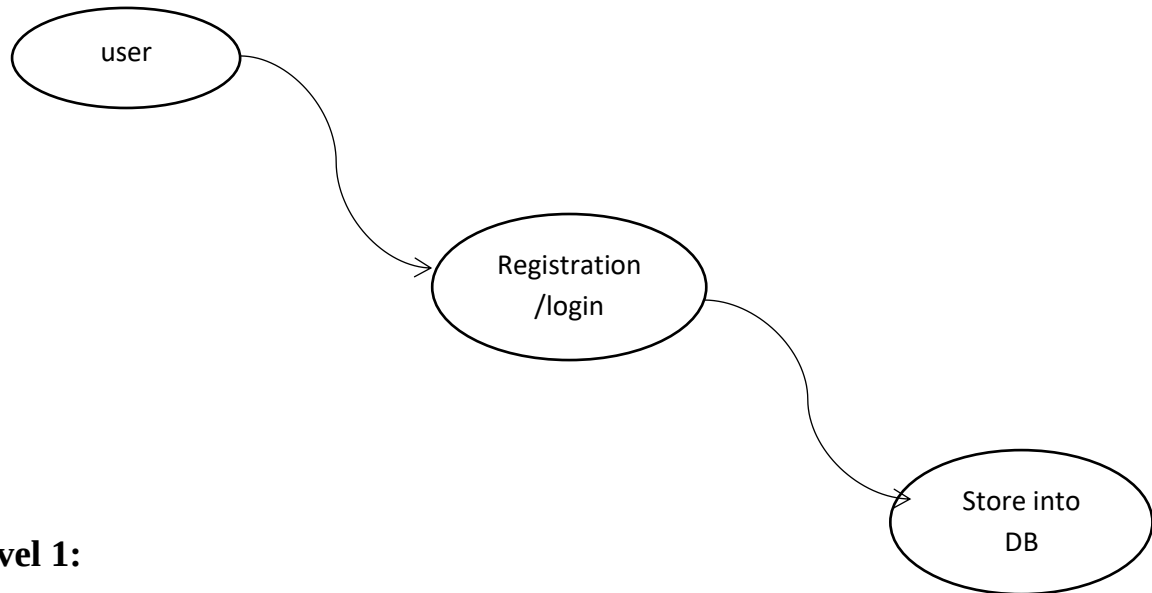


BLOCK DIAGRAM EXPLANATION:

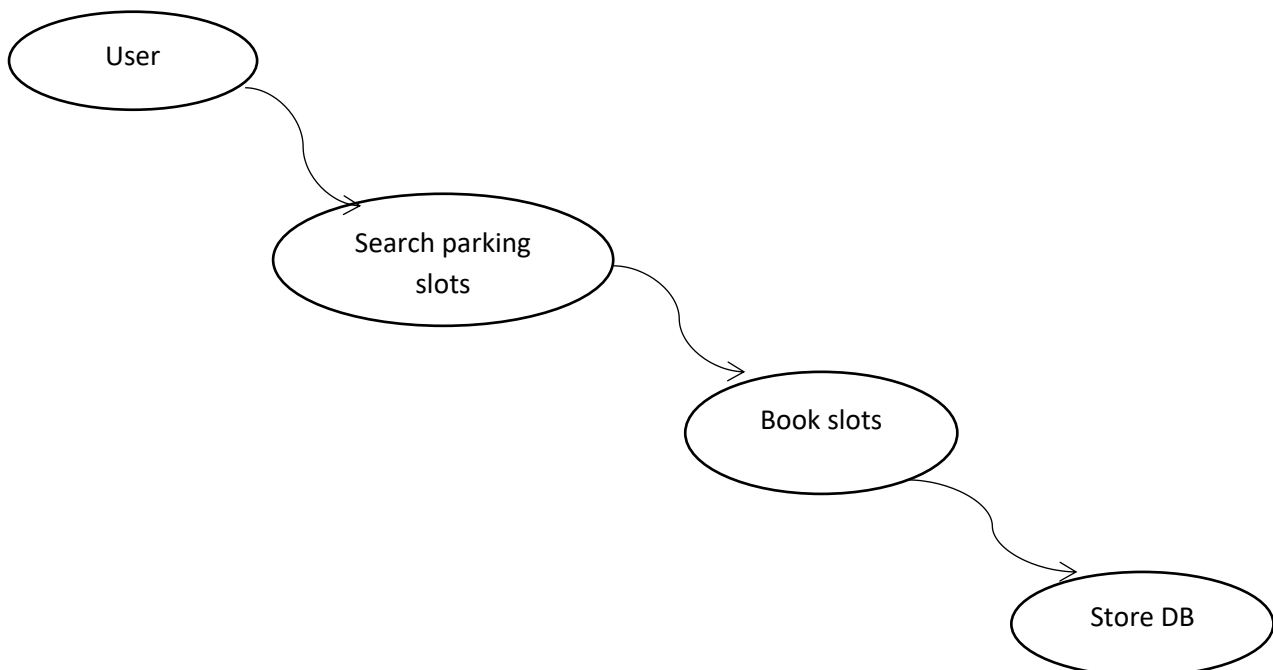
User authentication is one of the prime factors of our Proposed system. To make sure the user is authentic; we will use their mobile numbers to register. As, we all know in Bangladesh, all the mobile numbers are being verified by the National ID. So, once the owners install the APK _les of the system to their Android phones, they need to register him with their phone numbers. After giving phone number, a 6digit code will be given to a user for further process. The user needs to add the other information like his/her name Email Address, Car Number, Parking Slot Number. A user can register himself as a car owner, a parking slot owner or both.

Data Flow Diagram:

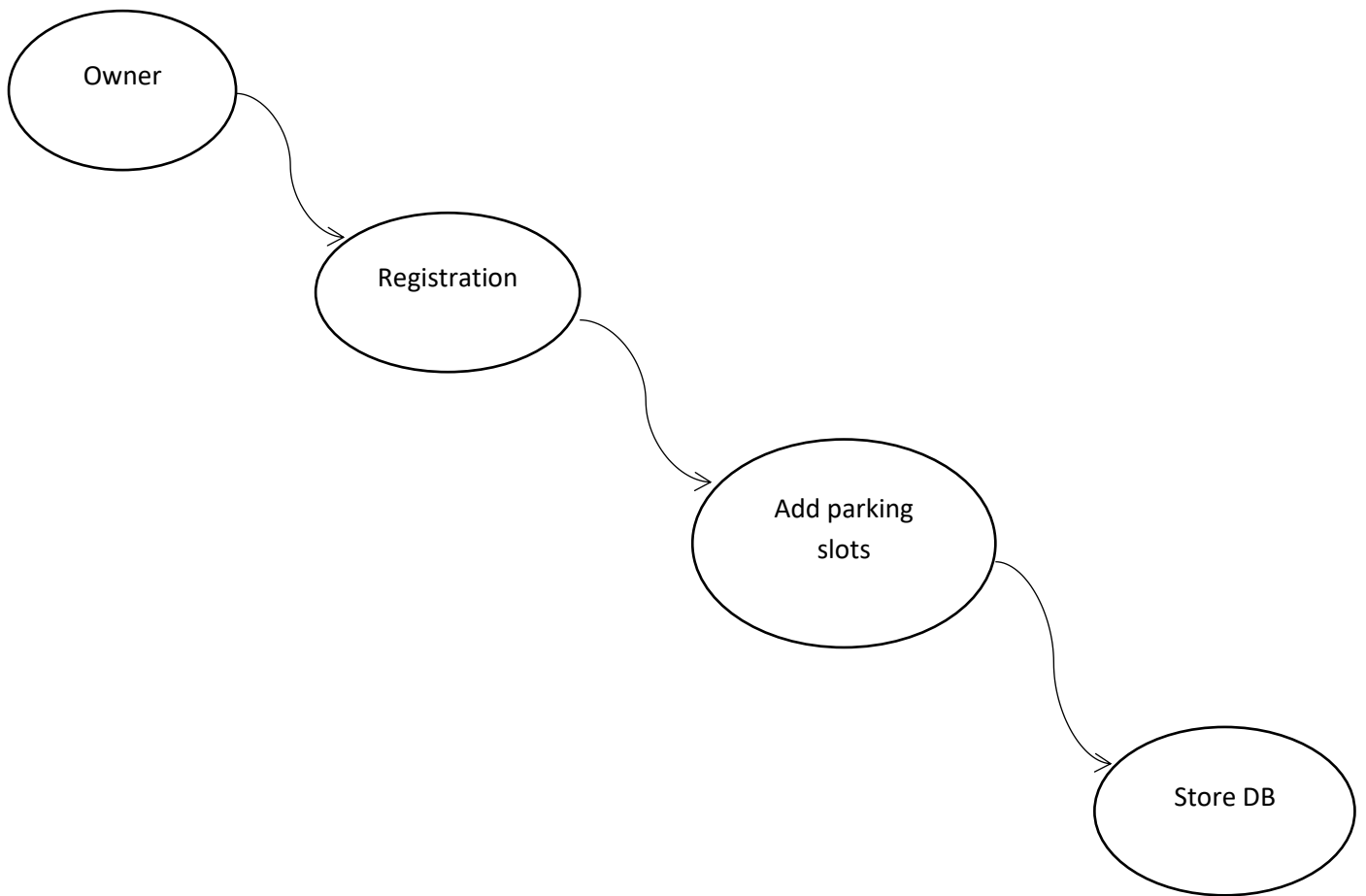
Level 0:



Level 1:

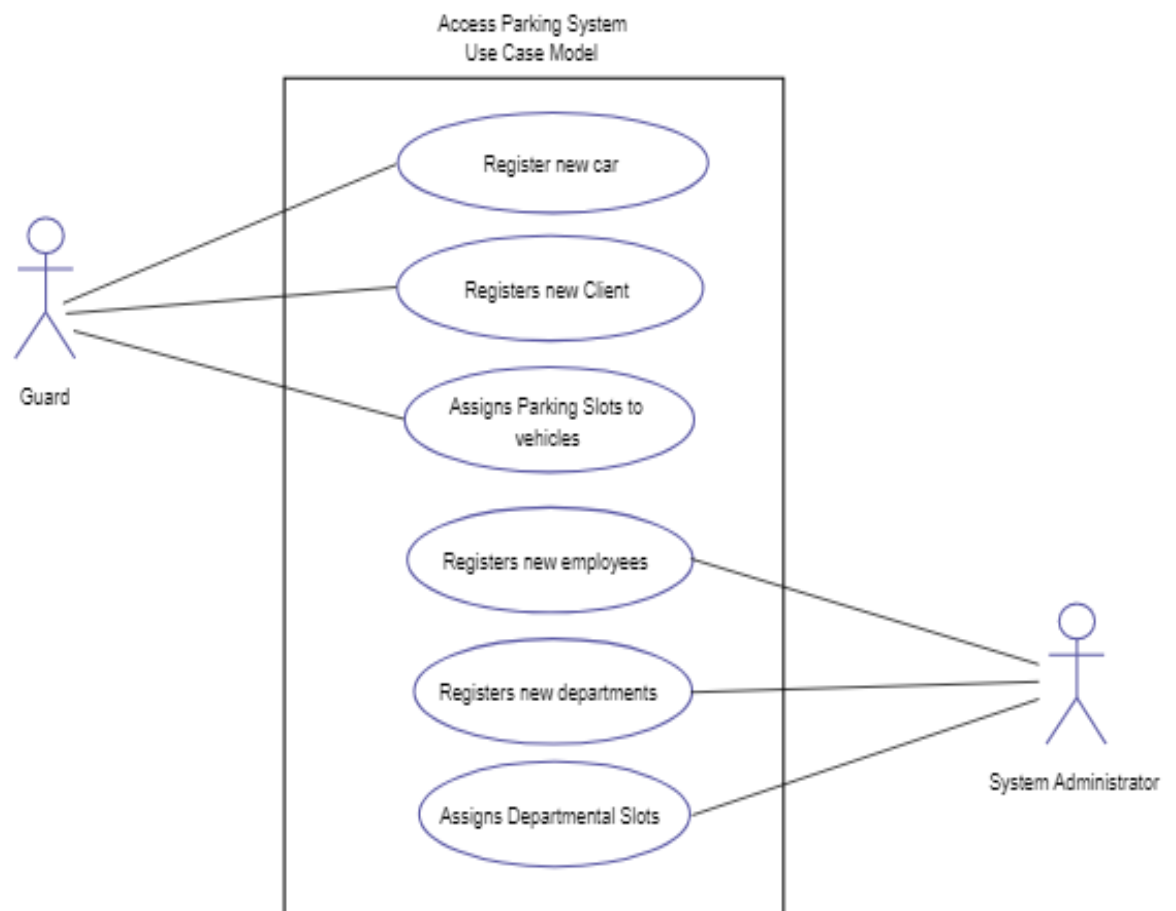


Level 2:

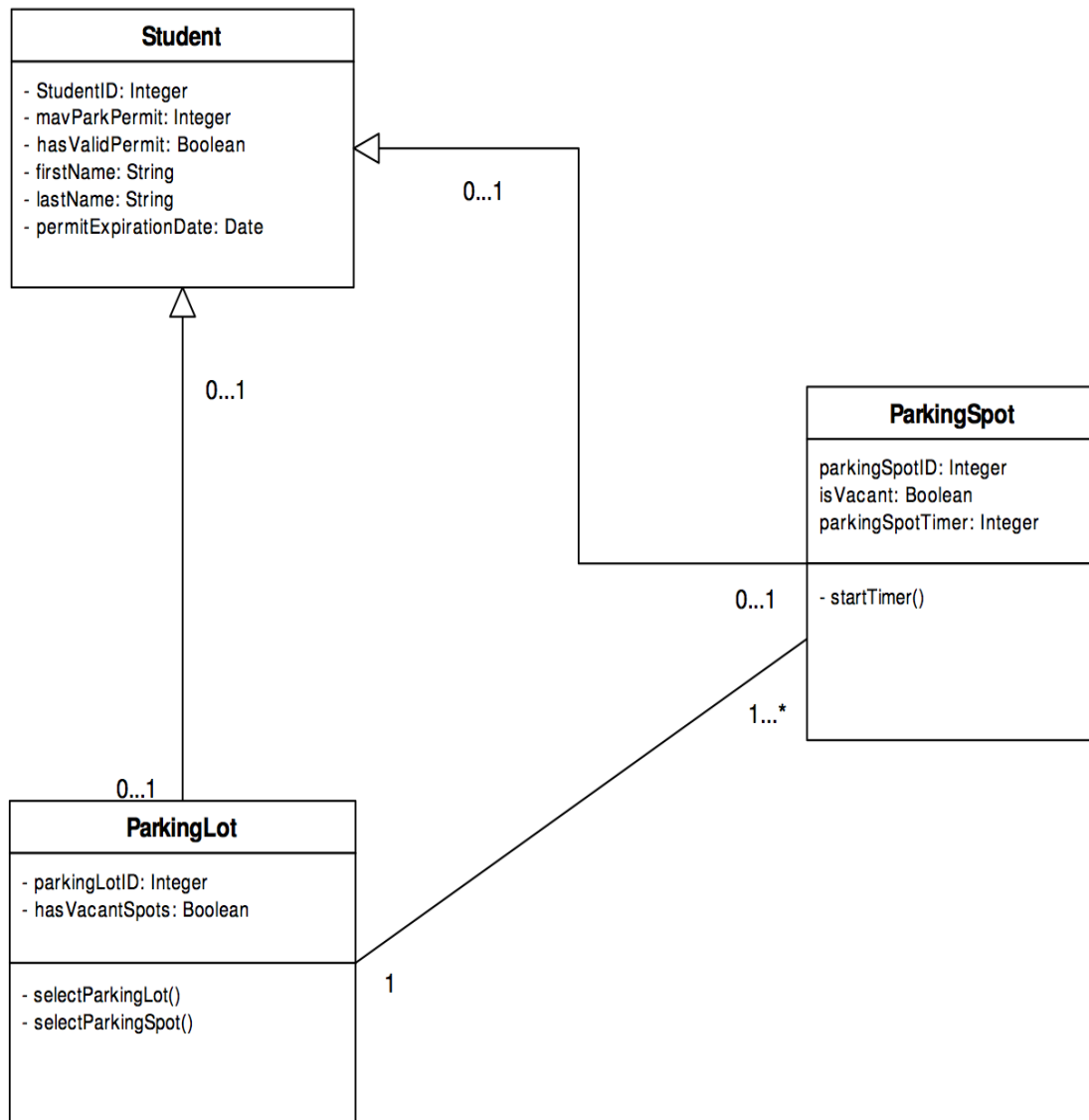


UML Diagrams:

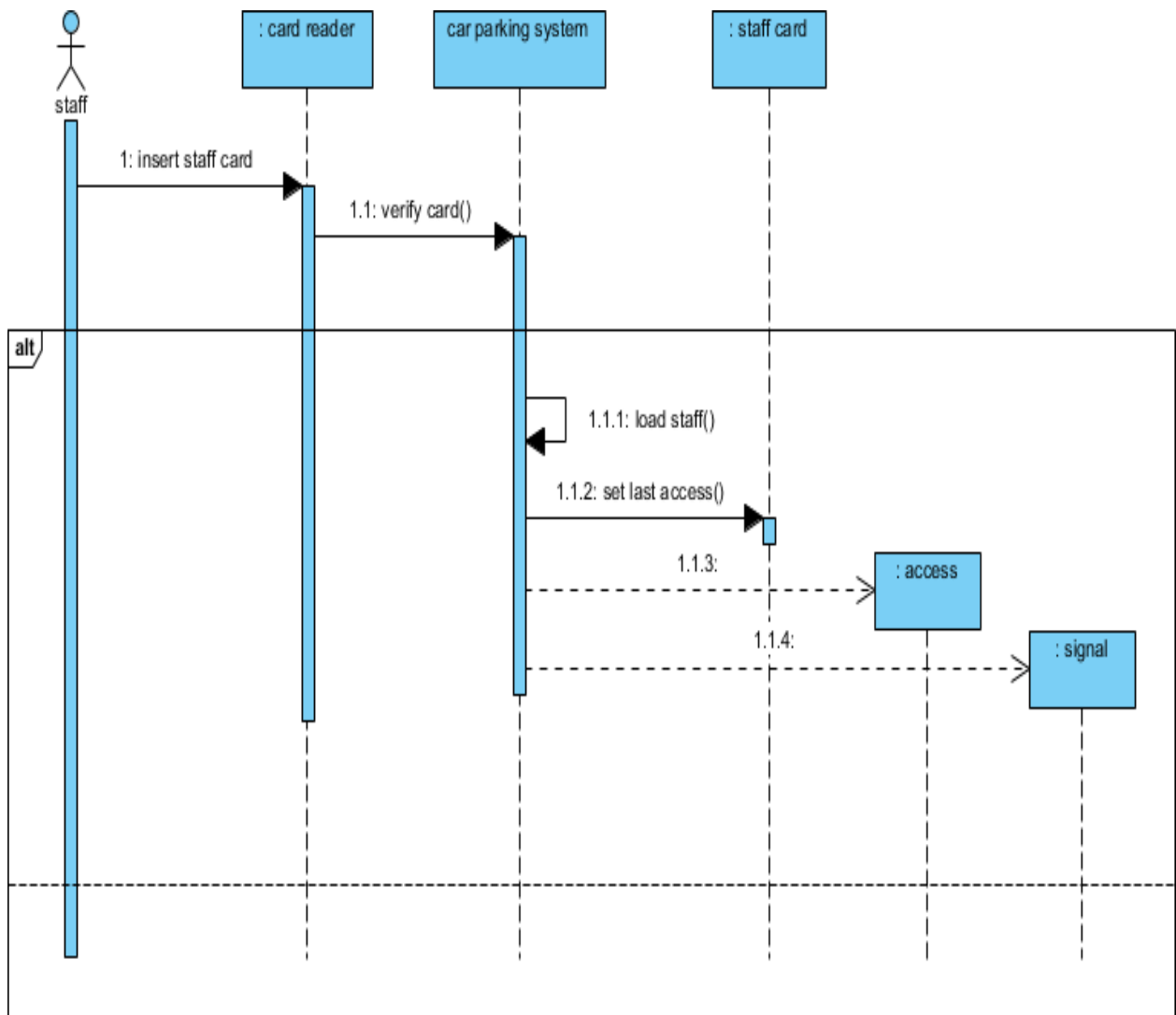
Use case Diagram:



Class Diagram:



Sequence Diagram:



6. Overview of Android

It is a free, open-source mobile platform. Linux-based, multi process, Multithreaded OS. Android is not a device or a product It's not even limited to phones You could build a DVR, a handheld GPS, an MP3 player, etc. Android is a software stack for mobile devices that includes an operating system, middleware and key applications.

The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.

- Makes mobile development easy.
- Full phone software stack including applications
- Designed as a platform for software development
- Android is open
- Android is free
- Community support
- **July 2005**
 - Google acquired Android Inc.
- **5 Nov 2007**
 - Open Handset Alliance formed-
 - Google, HTC, Intel, Motorola, QUALCOMM-Mobile
- Android is the OHA first product
- **12 Nov 2007**
 - OHA released a preview of the Android OHA
- **Oct-2008**
 - First Device – **T-Mobile G1** was released.
 - Latest – **Motorola Milestone** released in India.

Features

- **Application framework** enabling reuse and replacement of components
- **Dalvik virtual machine** optimized for mobile devices
- **Integrated browser** based on the open source Web Kit engine
- **Optimized graphics** powered by a custom 2D graphics library; 3Dgraphics based on the OpenGL ES 1.0 specification (hardware acceleration optional)
- **SQLite** for structured data storage
- **Media support** for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- **GSM Telephony** (hardware dependent)
- **Bluetooth, EDGE, 3G, and WIFI** (hardware dependent)
- **Camera, GPS, compass, and accelerometer** (hardware dependent)
- **Rich development environment** including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE

Linux Kernel

Android relies on Linux version 2.6 for core system services such as

- security
- memory management
- process management

- network stack
- driver model
- The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

Android Runtime

- Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language.
- Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently.
- The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint.
- The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

Libraries

Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework.

- System C Library
- Media Library

- Surface Manager
- LibWeb Core
- SGL
- 3D libraries
- Free Type
- SQLite

Application Framework

- Being an open development platform, Android offers developers the ability to build extremely rich and innovative applications.
- Developers have full access to the same framework APIs used by the core applications.
- **Views** – used to build applications (lists, grid, buttons, text boxes and even embeddable web browser)
- **Content providers** – enable applications to access data from other applications or share their own data.
- **Resource manager** – provides access to non-code resources such as localized strings, graphic and layout files.
- **Notification manager** – enables applications to display custom alerts in status bar
- **Activity manager** – manages lifecycle of applications and provides navigation back stack
- Android ships with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others.
- All applications are written using the Java programming language.

Anatomy of an Android Application

There are four building blocks for an Android application:

- **Activity** - a single screen
- **Broadcast Receiver**- to execute in reaction to an external event (Phone Ring)
- **Service** - code that is long-lived and runs without a UI (Media Player)
- **Content Provider** - an application's data to be shared with other applications

Android Building Blocks

These are the most important parts of the Android APIs:

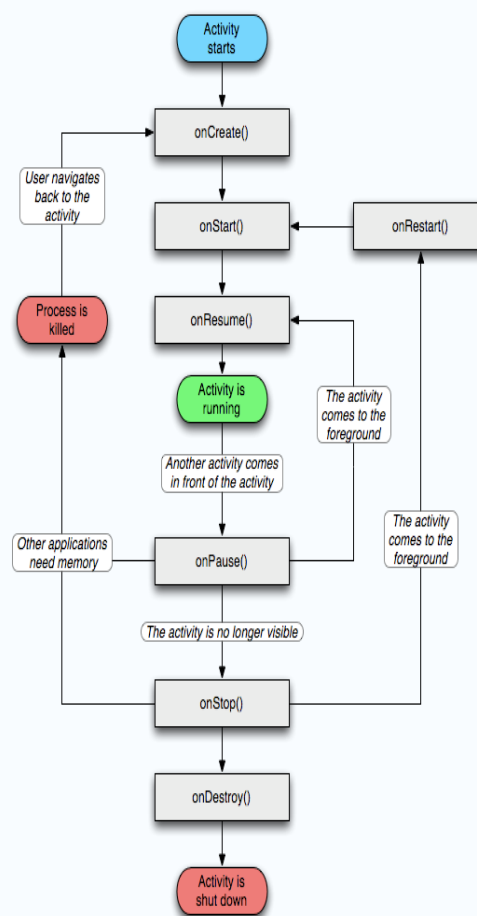
- **AndroidManifest.xml**
 - the control file-tells the system what to do with the top-level components
- **Activities**
 - an object that has a life cycle-is a chunk of code that does some work.
- **Views**
 - an object that knows how to draw itself to the screen
- **Intents**
 - a simple message object that represents an "intention" to do something
- **Notifications**
 - is a small icon that appears in the status bar (SMS messages)
 - for alerting the user
- **Services**
 - is a body of code that runs in the background

Development Tools

The Android SDK includes a variety of custom tools that help you develop mobile applications on the Android platform. Three of the most significant tools are:

- **Android Emulator** -A virtual mobile device that runs on our computer -use to design, debug, and test our applications in an actual Android run-time environment
- **Android Development Tools Plugin** -for the Eclipse IDE - adds powerful extensions to the Eclipse integrated environment
- **Dalvik Debug Monitor Service (DDMS)** -Integrated with Dalvik -this tool let us manage processes on an emulator and assists in debugging
- **Android Asset Packaging Tool (AAPT)** – Constructs the distributable Android package files (.apk)
- **Android Debug Bridge (ADB)** – provides link to a running emulator. Can copy files to emulator, install .apk files and run commands.

6.1 Lifecycle of activity

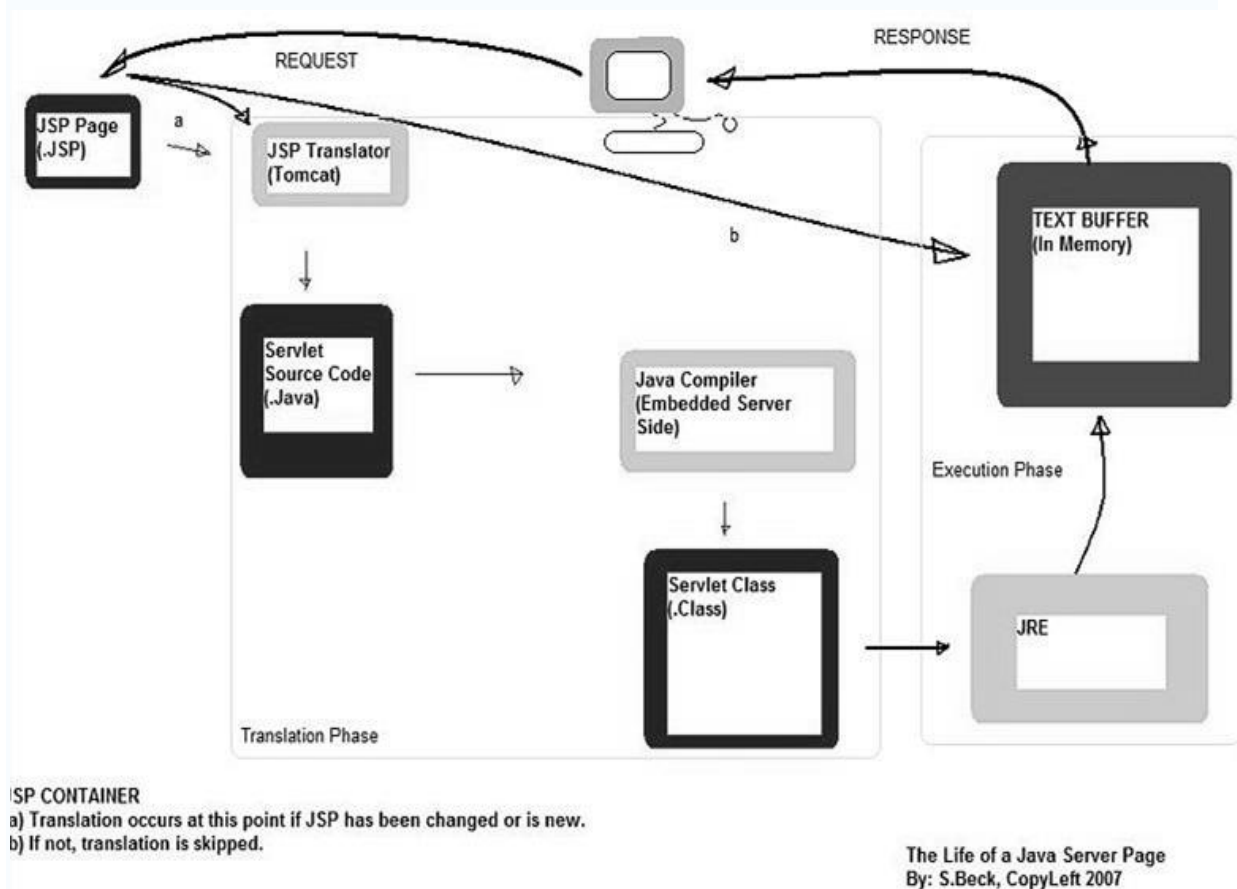


JSP:

Java Server Pages (JSP) is a Java technology that allows software developers to dynamically generate HTML, XML or other types of documents in response to a Web client request. The technology allows Java code and certain pre-defined actions to be embedded into static content.

JSPs are compiled into Java Servlets by a JSP compiler. A JSP compiler may generate a servlet in Java code that is then compiled by the Java compiler, or it may generate byte code for the servlet directly. JSPs can also be interpreted on-the-fly reducing the time taken to reload changes. Java Server Pages (JSP) technology provides a simplified, fast way to create dynamic web content. JSP technology enables rapid development of web-based applications that are server- and platform-independent.

Architecture OF JSP:



The Advantages of JSP:

- Active Server Pages (ASP). ASP is a similar technology from Microsoft. The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS-specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.
- Pure Servlets. JSP doesn't give you anything that you couldn't in principle do with a servlet. But it is more convenient to write (and to modify!) regular HTML than to have a zillion `println` statements that generate the HTML. Plus, by separating the look from the content you can put different people on different tasks: your Web page design experts can build the HTML, leaving places for your servlet programmers to insert the dynamic content.
- Server-Side Includes (SSI). SSI is a widely-supported technology for including externally-defined pieces into a static Web page. JSP is better because it lets you use servlets instead of a separate program to generate that dynamic part. Besides, SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
- JavaScript. JavaScript can generate HTML dynamically on the client. This is a useful capability, but only handles situations where the dynamic information is based on the client's environment. With the exception of cookies, HTTP and form submission data is not available to JavaScript. And, since it runs on the client, JavaScript can't access server-side resources like databases, catalogues, pricing information, and the like.
- Static HTML. Regular HTML, of course, cannot contain dynamic information. JSP is so easy and convenient that it is quite feasible to augment HTML pages that only benefit

marginally by the insertion of small amounts of dynamic data. Previously, the cost of using dynamic data would preclude its use in all but the most valuable instances.

SERVLETS

The Java Servlet API allows a software developer to add dynamic content to a Web server using the Java platform. The generated content is commonly HTML, but may be other data such as XML. Servlets are the Java counterpart to non-Java dynamic Web content technologies such as PHP, CGI and ASP.NET. Servlets can maintain state across many server transactions by using HTTP cookies, session variables or URL rewriting.

The Servlet API, contained in the Java package hierarchy javax.servlet, defines the expected interactions of a Web container and a servlet. A Web container is essentially the component of a Web server that interacts with the servlets. The Web container is responsible for managing the lifecycle of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights.

A Servlet is an object that receives a request and generates a response based on that request. The basic servlet package defines Java objects to represent servlet requests and responses, as well as objects to reflect the servlet's configuration parameters and execution environment. The package javax.servlet.http defines HTTP-specific subclasses of the generic servlet elements, including session management objects that track multiple requests and responses between the Web server and a client. Servlets may be packaged in a WAR file as a Web application.

Servlets can be generated automatically by Java Server Pages (JSP), or alternately by template engines such as Web Macro. Often servlets are used in conjunction with JSPs in a pattern called "Model 2", which is a flavor of the model-view-controller pattern.

Servlets are Java technology's answer to CGI programming. They are programs that run on a Web server and build Web pages. Building Web pages on the fly is useful (and commonly done) for a number of reasons:

- The Web page is based on data submitted by the user. For example, the results pages from search engines are generated this way, and programs that process orders for e-commerce sites do this as well.
- The data changes frequently. For example, a weather-report or news headlines page might build the page dynamically, perhaps returning a previously built page if it is still up to date.
- The Web page uses information from corporate databases or other such sources. For example, you would use this for making a Web page at an on-line store that lists current prices and number of items in stock.

The Servlet Run-time Environment:

A servlet is a Java class and therefore needs to be executed in a Java VM by a service we call a servlet engine.

The servlet engine loads the servlet class the first time the servlet is requested, or optionally already when the servlet engine is started. The servlet then stays loaded to handle multiple requests until it is explicitly unloaded or the servlet engine is shut down.

Some Web servers, such as Sun's Java Web Server (JWS), W3C's Jigsaw and Gefion Software's Lite Webserver (LWS) are implemented in Java and have a built-in servlet engine. Other Web servers, such as Netscape's Enterprise Server, Microsoft's Internet Information Server (IIS) and the Apache Group's Apache, require a servlet engine add-on module. The add-on intercepts all requests for servlets, executes them and returns the response through the Web server to the client. Examples of servlet engine add-ons are Gefion Software's WAICoolRunner, IBM's WebSphere, Live Software's JRun and New Atlanta's Servlet Exec.

All Servlet API classes and a simple servlet-enabled Web server are combined into the Java Servlet Development Kit (JSDK), available for download at Sun's official Servlet site .To get started with servlets I recommend that you download the JSDK and play around with the sample servlets.

Life Cycle OF Servlet

The Servlet lifecycle consists of the following steps:

1. The Servlet class is loaded by the container during start-up.
2. The container calls the `init()` method. This method initializes the servlet and must be called before the servlet can service any requests. In the entire life of a servlet, the `init()` method is called only once.
3. After initialization, the servlet can service client-requests. Each request is serviced in its own separate thread. The container calls the `service()` method of the servlet for every request. The `service()` method determines the kind of request being made and dispatches it to an appropriate method to handle the request. The developer of the servlet must provide an implementation for these methods. If a request for a method that is not implemented by the servlet is made, the method of the parent class is called, typically resulting in an error being returned to the requester.
4. Finally, the container calls the `destroy()` method which takes the servlet out of service. The `destroy()` method like `init()` is called only once in the lifecycle of a Servlet.

Testing

The various levels of testing are

1. White Box Testing
2. Black Box Testing
3. Unit Testing
4. Functional Testing
5. Performance Testing
6. Integration Testing

7. Objective
8. Integration Testing
9. Validation Testing
10. System Testing
11. Structure Testing
12. Output Testing
13. User Acceptance Testing

White Box Testing

White-box testing (also known as **clear box testing**, **glass box testing**, **transparent box testing**, and **structural testing**) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT).

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. It can test paths within a unit, paths between units during integration, and between subsystems during a system-level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

White-box test design techniques include:

- Control flow testing
- Data flow testing
- Branch testing
- Path testing
- Statement coverage
- Decision coverage

White-box testing is a method of testing the application at the level of the source code. The test cases are derived through the use of the design techniques mentioned above: control flow testing, data flow testing, branch testing, path testing, statement coverage and decision coverage as well as modified condition/decision coverage. White-box testing is the use of these techniques as guidelines to create an error free environment by examining any fragile code.

These White-box testing techniques are the building blocks of white-box testing, whose essence is the careful testing of the application at the source code level to prevent any hidden errors later on. These different techniques exercise every visible path of the source code to minimize errors and create an error-free environment. The whole point of white-box testing is the ability to know which line of the code is being executed and being able to identify what the correct output should be.

Levels

1. Unit testing. White-box testing is done during unit testing to ensure that the code is working as intended, before any integration happens with previously tested code. White-box testing during unit testing catches any defects early on and aids in any defects that happen later on after the code is integrated with the rest of the application and therefore prevents any type of errors later on.
2. Integration testing. White-box testing at this level are written to test the interactions of each interface with each other. The Unit level testing made sure that each code was tested and working accordingly in an isolated environment and integration examines the correctness of the behaviour in an open environment through the use of white-box testing for any interactions of interfaces that are known to the programmer.
3. Regression testing. White-box testing during regression testing is the use of recycled white-box test cases at the unit and integration testing levels.

White-box testing's basic procedures involve the understanding of the source code that you are testing at a deep level to be able to test them. The programmer must have a deep understanding of the application to know what kinds of test cases to create so that every visible path is exercised for testing. Once the source code is understood then the source code can be analysed for test cases to be created. These are the three basic steps that white-box testing takes in order to create test cases:

1. Input, involves different types of requirements, functional specifications, detailed designing of documents, proper source code, security specifications. This is the preparation stage of white-box testing to layout all of the basic information.
2. Processing Unit, involves performing risk analysis to guide whole testing process, proper test plan, execute test cases and communicate results. This is the phase of building test cases to make sure they thoroughly test the application the given results are recorded accordingly.
3. Output, prepare final report that encompasses all of the above preparations and results.

Black Box Testing

Black-box testing is a method of software testing that examines the functionality of an application (e.g., what the software does) without peering into its internal structures or workings (see white-box testing). This method of test can be applied to virtually every level of software testing: unit, integration, system and acceptance. It typically comprises most if not all higher-level testing, but can also dominate unit testing as well

Test procedures

Specific knowledge of the application's code/internal structure and programming knowledge in general is not required. The tester is aware of *what* the software is supposed to do but is not aware of *how* it does it. For instance, the tester is aware that a particular input returns a certain, invariable output but is not aware of *how* the software produces the output in the first place.

Test cases

Test cases are built around specifications and requirements, i.e., what the application is supposed to do. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters. Although the tests used are primarily *functional* in nature, *non-functional* tests may also be used. The test designer selects

both valid and invalid inputs and determines the correct output without any knowledge of the test object's internal structure.

Test design techniques

Typical black-box test design techniques include:

- Decision table testing
- All-pairs testing
- State transition tables
- Equivalence partitioning
- Boundary value analysis

Unit testing

In computer programming, **unit testing** is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In procedural programming, a unit could be an entire module, but is more commonly an individual function or procedure. In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are created by programmers or occasionally by white box testers during the development process.

Ideally, each test case is independent from the others. Substitutes such as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended. Its implementation can vary from being very manual (pencil and paper) to being formalized as part of build automation.

Testing will not catch every error in the program, since it cannot evaluate every execution path in any but the most trivial programs. The same is true for unit testing. Additionally, unit testing by definition only tests the functionality of the units themselves. Therefore, it will not catch integration errors or broader system-level errors (such as functions performed across multiple units, or non-functional test areas such as performance).

Unit testing should be done in conjunction with other software testing activities, as they can only show the presence or absence of particular errors; they cannot prove a complete absence of errors. In order to guarantee correct behaviour for every execution path and every possible input, and ensure the absence of errors, other techniques are required, namely the application of formal methods to proving that a software component has no unexpected behaviour.

Software testing is a combinatorial problem. For example, every Boolean decision statement requires at least two tests: one with an outcome of "true" and one with an outcome of "false". As a result, for every line of code written, programmers often need 3 to 5 lines of test code.

This obviously takes time and its investment may not be worth the effort. There are also many problems that cannot easily be tested at all – for example those that are nondeterministic or involve multiple threads. In addition, code for a unit test is likely to be at least as buggy as the code it is testing. Fred Brooks in *The Mythical Man-Month* quotes: *never take two chronometers to sea. Always take one or three.* Meaning, if two chronometers contradict, how do you know which one is correct?

Another challenge related to writing the unit tests is the difficulty of setting up realistic and useful tests. It is necessary to create relevant initial conditions so the part of the application being tested behaves like part of the complete system. If these initial conditions are not set correctly, the test will not be exercising the code in a realistic context, which diminishes the value and accuracy of unit test results.

To obtain the intended benefits from unit testing, rigorous discipline is needed throughout the software development process. It is essential to keep careful records not only of the tests that have been performed, but also of all changes that have been made to the source code of this or any other unit in the software. Use of a version control system is essential. If a later version of the unit fails a particular test that it had previously passed, the version-control software can provide a list of the source code changes (if any) that have been applied to the unit since that time.

It is also essential to implement a sustainable process for ensuring that test case failures are reviewed daily and addressed immediately if such a process is not implemented and ingrained into the team's workflow, the application will evolve out of sync with the unit test suite, increasing false positives and reducing the effectiveness of the test suite.

Unit testing embedded system software presents a unique challenge: Since the software is being developed on a different platform than the one it will eventually run on, you cannot readily run a test program in the actual deployment environment, as is possible with desktop programs.^[7]

Functional testing

Functional testing is a quality assurance (QA) process and a type of black box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered (not like in white-box testing). Functional Testing usually describes *what* the system does.

Functional testing differs from system testing in that functional testing "*verifies* a program by checking it against ... design document(s) or specification(s)", while system testing "*validate* a program by checking it against the published user or system requirements" (Kane, Falk, Nguyen 1999, p. 52).

Functional testing typically involves five steps. The identification of functions that the software is expected to perform

1. The creation of input data based on the function's specifications
2. The determination of output based on the function's specifications
3. The execution of the test case
4. The comparison of actual and expected outputs

Performance testing

In software engineering, **performance testing** is in general testing performed to determine how a system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

Performance testing is a subset of performance engineering, an emerging computer science practice which strives to build performance into the implementation, design and architecture of a system.

Testing types

Load testing

Load testing is the simplest form of performance testing. A load test is usually conducted to understand the behaviour of the system under a specific expected load. This load can be the expected concurrent number of users on the application performing a specific number of transactions within the set duration. This test will give out the response times of all the important business critical transactions. If the database, application server, etc. are also monitored, then this simple test can itself point towards bottlenecks in the application software.

Stress testing

Stress testing is normally used to understand the upper limits of capacity within the system. This kind of test is done to determine the system's robustness in terms of extreme load and helps application administrators to determine if the system will perform sufficiently if the current load goes well above the expected maximum.

Soak testing

Soak testing, also known as endurance testing, is usually done to determine if the system can sustain the continuous expected load. During soak tests, memory utilization is monitored to detect potential leaks. Also important, but often overlooked is performance degradation. That is, to ensure that the throughput and/or response times after some long period of sustained activity are as good as or better than at the beginning of the test. It essentially involves applying a significant load to a system for an extended, significant period of time. The goal is to discover how the system behaves under sustained use.

Spike testing

Spike testing is done by suddenly increasing the number of or load generated by, users by a very large amount and observing the behaviour of the system. The goal is to determine whether performance will suffer, the system will fail, or it will be able to handle dramatic changes in load.

Configuration testing

Rather than testing for performance from the perspective of load, tests are created to determine the effects of configuration changes to the system's components on the system's performance and behaviour. A common example would be experimenting with different methods of load-balancing.

Isolation testing

Isolation testing is not unique to performance testing but involves repeating a test execution that resulted in a system problem. Often used to isolate and confirm the fault domain.

Integration testing

Integration testing (sometimes called **integration and testing**, abbreviated **I&T**) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

Purpose

The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These "design items", i.e., assemblages (or groups of units), are exercised through their interfaces using black box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface.

Test cases are constructed to test whether all the components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e., unit testing. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.

Some different types of integration testing are big bang, top-down, and bottom-up. Other Integration Patterns are: Collaboration Integration, Backbone Integration, Layer Integration, Client/Server Integration, Distributed Services Integration and High-frequency Integration.

Big Bang

In this approach, all or most of the developed modules are coupled together to form a complete software system or major part of the system and then used for integration testing. The Big Bang method is very effective for saving time in the integration testing process. However, if the test cases and their results are not recorded properly, the entire integration process will be more complicated and may prevent the testing team from achieving the goal of integration testing.

A type of Big Bang Integration testing is called **Usage Model testing**. Usage Model Testing can be used in both software and hardware integration testing. The basis behind this type of integration testing is to run user-like workloads in integrated user-like environments. In doing the testing in this manner, the environment is proofed, while the individual components are proofed indirectly through their use.

Usage Model testing takes an optimistic approach to testing, because it expects to have few problems with the individual components. The strategy relies heavily on the component developers to do the isolated unit testing for their product. The goal of the strategy is to avoid redoing the testing done by the developers, and instead flesh-out problems caused by the interaction of the components in the environment.

For integration testing, Usage Model testing can be more efficient and provides better test coverage than traditional focused functional integration testing. To be more efficient and accurate, care must be used in defining the user-like workloads for creating realistic scenarios in exercising the environment. This gives confidence that the integrated environment will work as expected for the target customers.

Top-down and Bottom-up

Bottom-Up Testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher-level components. The process is repeated until the component at the top of the hierarchy is tested.

All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower-level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.

Top-Down Testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module.

Sandwich Testing is an approach to combine top-down testing with bottom-up testing. The main advantage of the Bottom-Up approach is that bugs are more easily found. With Top-Down, it is easier to find a missing branch link

Verification and validation

Verification and Validation are independent procedures that are used together for checking that a product, service, or system meets requirements and specifications and that it fully fills its intended purpose. These are critical components of a quality management system such as ISO 9000. The words "verification" and "validation" are sometimes preceded with "Independent" (or IV&V), indicating that the verification and validation is to be performed by a disinterested third party.

It is sometimes said that validation can be expressed by the query "Are you building the right thing?" and verification by "Are you building it right?" In practice, the usage of these terms varies. Sometimes they are even used interchangeably.

The PMBOK guide, an IEEE standard, defines them as follows in its 4th edition

- **"Validation.** The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. Contrast with *verification*."
- **"Verification.** The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process. Contrast with *validation*."
- Verification is intended to check that a product, service, or system (or portion thereof, or set thereof) meets a set of initial design specifications. In the development phase, verification procedures involve performing special tests to model or simulate a portion, or the entirety, of a product, service or system, then performing a review or analysis of the modelling results. In the post-development phase, verification procedures involve regularly repeating tests devised specifically to ensure that the product, service, or system continues to meet the initial design requirements, specifications, and regulations as time progresses. It is a process that is used to evaluate whether a product, service, or system complies with regulations, specifications, or conditions imposed at the start of a development phase. Verification can be in development, scale-up, or production. This is often an internal process.
- Validation is intended to check that development and verification procedures for a product, service, or system (or portion thereof, or set thereof) result in a product,

service, or system (or portion thereof, or set thereof) that meets initial requirements. For a new development flow or verification flow, validation procedures may involve modelling either flow and using simulations to predict faults or gaps that might lead to invalid or incomplete verification or development of a product, service, or system (or portion thereof, or set thereof). A set of validation requirements, specifications, and regulations may then be used as a basis for qualifying a development flow or verification flow for a product, service, or system (or portion thereof, or set thereof). Additional validation procedures also include those that are designed specifically to ensure that modifications made to an existing qualified development flow or verification flow will have the effect of producing a product, service, or system (or portion thereof, or set thereof) that meets the initial design requirements, specifications, and regulations; these validations help to keep the flow qualified. It is a process of establishing evidence that provides a high degree of assurance that a product, service, or system accomplishes its intended requirements. This often involves acceptance of fitness for purpose with end users and other product stakeholders. This is often an external process.

- It is sometimes said that validation can be expressed by the query "Are you building the right thing?" and verification by "Are you building it right?". "Building the right thing" refers back to the user's needs, while "building it right" checks that the specifications are correctly implemented by the system. In some contexts, it is required to have written requirements for both as well as formal procedures or protocols for determining compliance.
- It is entirely possible that a product passes when verified but fails when validated. This can happen when, say, a product is built as per the specifications but the specifications themselves fail to address the user's needs.

Activities

Verification of machinery and equipment usually consists of design qualification (DQ), installation qualification (IQ), operational qualification (OQ), and performance qualification (PQ). DQ is usually a vendor's job. However, DQ can also be performed by the user, by confirming through review and testing that the equipment meets the written acquisition

specification. If the relevant document or manuals of machinery/equipment are provided by vendors, the later 3Q needs to be thoroughly performed by the users who work in an industrial regulatory environment. Otherwise, the process of IQ, OQ and PQ is the task of validation. The typical example of such a case could be the loss or absence of vendor's documentation for legacy equipment or do-it-yourself (DIY) assemblies (e.g., cars, computers etc.) and, therefore, users should endeavour to acquire DQ document beforehand. Each template of DQ, IQ, OQ and PQ usually can be found on the internet respectively, whereas the DIY qualifications of machinery/equipment can be assisted either by the vendor's training course materials and tutorials, or by the published guidance books, such as *step-by-step* series if the acquisition of machinery/equipment is not bundled with on- site qualification services. This kind of the DIY approach is also applicable to the qualifications of software, computer operating systems and a manufacturing process. The most important and critical task as the last step of the activity is to generating and archiving machinery/equipment qualification reports for auditing purposes, if regulatory compliances are mandatory.

Qualification of machinery/equipment is venue dependent, in particular items that are shock sensitive and require balancing or calibration, and re-qualification needs to be conducted once the objects are relocated. The full scales of some equipment qualifications are even time dependent as consumables are used up (i.e. filters) or springs stretch out, requiring recalibration, and hence re-certification is necessary when a specified due time lapse Re-qualification of machinery/equipment should also be conducted when replacement of parts, or coupling with another device, or installing a new application software and restructuring of the computer which affects especially the pre-settings, such as on BIOS, registry, disk drive partition table, dynamically-linked (shared) libraries, or an ini file etc., have been necessary. In such a situation, the specifications of the parts/devices/software and restructuring proposals should be appended to the qualification document whether the parts/devices/software are genuine or not.

Torres and Hyman have discussed the suitability of non-genuine parts for clinical use and provided guidelines for equipment users to select appropriate substitutes which are capable to avoid adverse effects. In the case when genuine parts/devices/software are demanded by some of regulatory requirements, then re-qualification does not need to be conducted on the non-genuine assemblies. Instead, the asset has to be recycled for non-regulatory purposes.

When machinery/equipment qualification is conducted by a standard endorsed third party such as by an ISO standard accredited company for a particular division, the process is called certification. Currently, the coverage of ISO/IEC 15408 certification by an ISO/IEC 27001 accredited organization is limited; the scheme requires a fair amount of efforts to get popularized.

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 passed integration testing and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called *assemblages*) or between any of the *assemblages* and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

System testing is performed on the entire system in the context of a Functional Requirement Specification(s) (FRS) and/or a System Requirement Specification (SRS). System testing tests not only the design, but also the behavior and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specification

Types of tests to include in system testing

The following examples are different types of testing that should be considered during System testing:

- Graphical user interface testing
- Usability testing

- Software performance testing
- Compatibility testing
- Exception handling
- Load testing
- Volume testing
- Stress testing
- Security testing
- Scalability testing
- Sanity testing
- Smoke testing
- Exploratory testing
- Ad hoc testing
- Regression testing
- Installation testing
- Maintenance testing Recovery testing and failover testing.
- Accessibility testing, including compliance with:
 - Americans with Disabilities Act of 1990
 - Section 508 Amendment to the Rehabilitation Act of 1973
 - Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C)

Although different testing organizations may prescribe different tests as part of System testing, this list serves as a general framework or foundation to begin with.

Structure Testing:

It is concerned with exercising the internal logic of a program and traversing particular execution paths.

Output Testing:

- Output of test cases compared with the expected results created during design of test cases.

- Asking the user about the format required by them tests the output generated or displayed by the system under consideration.
- Here, the output format is considered into two was, one is on screen and another one is printed format.
- The output on the screen is found to be correct as the format was designed in the system design phase according to user needs.
- The output comes out as the specified requirements as the user's hard copy.

User acceptance Testing:

- Final Stage, before handling over to the customer which is usually carried out by the customer where the test cases are executed with actual data.
- The system under consideration is tested for user acceptance and constantly keeping touch with the prospective system user at the time of developing and making changes whenever required.
- It involves planning and execution of various types of test in order to demonstrate that the implemented software system satisfies the requirements stated in the requirement document.

Two set of acceptance test to be run:

1. Those developed by quality assurance group.
2. Those developed by customer

7. CONCLUSION AND FUTURE ENHANCEMENT

7.1 Conclusion:

Thus, we proposed and implemented an app-based parking reservation system which facilitates the user to book a parking slot. It also searches for the nearest parking slot and includes features like QR code scanning in order to ensure consistency. At last, the online payment option eliminates the need for human intervention and making the system automated in real sense. Also, the proposed system is very economical and easy to implement as it does not involve any expensive hardware or devices.

7.2 FUTURE ENHANCEMENT:

The proposed system can further be upgraded by including features like sensors and CCTV camera so that the latest status of the parking can be checked by the user through the same application, rather than providing statistical data (in the current system). Also, navigation facility can be added to guide the user at each step to make sure that the user reaches the destination slot in the minimum possible time

References:

- [1] V. G. Sahu, V. Gulhane, and N. Shelokar, "A web based centralized vehicle parking system using gsm security," IJAIEM, vol. 2, no. 4, 2013.
- [2] Z. Ji, I. Ganchev, M. O'Droma, L. Zhao, and X. Zhang, "A cloud-based car parking middleware for iot-based smart cities: Design and implementation," Sensors, vol. 14, no. 12, pp. 22 372{22 393, 2014.

- [3] V. P. Bilodeau, "Intelligent parking technology adoption," Ph.D. dissertation, University of Southern Queensland, 2010.
- [4] Y. Zhao and E. G. Collins Jr, "Robust automatic parallel parking in tight spaces via fuzzy logic," Robotics and Autonomous Systems, vol. 51, no. 2-3, pp. 111-127, 2005.
- [5] B. R. T. Authority, "Number of registered motor vehicles in Bangladesh (year wise)," 2017, [On-line; accessed 15-September-2017]. [Online]. Available: [p://www.brta.gov.bd/newsite/en/whole-bangladesh-up-to-sep-2016/](http://www.brta.gov.bd/newsite/en/whole-bangladesh-up-to-sep-2016/)

APPENDIX

A. SOURCE CODE

NEW PARK:

```

/*
SQLyog - Free MySQL GUI v5.13
Host - 5.0.22-community-nt : Database - newpark
*****

Server version : 5.0.22-community-nt
*/

SET NAMES utf8;

SET SQL_MODE="";

create database if not exists `newpark`;

```

```

USE `newpark`;
SET                                     @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO';

/*Table structure for table `bankdetails` */

DROP TABLE IF EXISTS `bankdetails`;

CREATE TABLE `bankdetails` (
  `name` varchar(20) default NULL,
  `accountno` varchar(200) default NULL,
  `cvv` varchar(200) default NULL,
  `expirydate` varchar(200) default NULL,
  `password` varchar(200) default NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*Data for the table `bankdetails` */

insert into `bankdetails` (`name`,`accountno`,`cvv`,`expirydate`,`password`) values
('sri','1234567890','999','12/2024','asdf');

/*Table structure for table `slots` */

DROP TABLE IF EXISTS `slots`;

CREATE TABLE `slots` (
  `Place` varchar(200) default NULL,
  `slots` varchar(200) default NULL,
  `price` varchar(200) default NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*Data for the table `slots` */

```

```

insert into `slots` (`Place`,`slots`,`price`) values ('tnagar','5','20.00');
insert into `slots` (`Place`,`slots`,`price`) values ('gundiy','5','20.00');
insert into `slots` (`Place`,`slots`,`price`) values ('pilar','5','20.00');
insert into `slots` (`Place`,`slots`,`price`) values ('porur','5','20.00');

```

```

/*Table structure for table `user` */

```

```

DROP TABLE IF EXISTS `user`;

```

```

CREATE TABLE `user` (
  `username` varchar(30) default NULL,
  `password` varchar(30) default NULL,
  `mail` varchar(30) default NULL,
  `mobile` varchar(30) default NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

```

```

/*Data for the table `user` */

```

```

insert      into      `user`      (`username`,`password`,`mail`,`mobile`)      values
('sri','asdf',NULL,'9884113466');

```

```

SET SQL_MODE=@OLD_SQL_MODE;

```

```

package com.example.smartparking;

```

```

import java.io.IOException;

```

```

import org.ksoap2.SoapEnvelope;

```

```

import org.ksoap2.serialization.SoapObject;

import org.ksoap2.serialization.SoapPrimitive;

import org.ksoap2.serialization.SoapSerializationEnvelope;

import org.ksoap2.transport.HttpTransportSE;

import org.xmlpull.v1.XmlPullParserException;


public class Callservice {


    private static String NAMESPACE="http://New/";

    private static String
URL="http://192.168.1.5:8080/Parking_service/Parkingservice?WSDL";


    public static String loginService(String s1, String s2, String method) {
        // TODO Auto-generated method stub

        String restex=null;

        SoapObject soap=new SoapObject(NAMESPACE, method);

        soap.addProperty("user_name",s1);

        soap.addProperty("password",s2);

        SoapSerializationEnvelope envelope=new
SoapSerializationEnvelope(SoapEnvelope.VER11);

        envelope.setOutputSoapObject(soap);

        HttpTransportSE http=new HttpTransportSE(URL);

        try {

            http.call(NAMESPACE+method, envelope);

```

```

        SoapPrimitive
        primitive
    =(SoapPrimitive)envelope.getResponse();

    restex=primitive.toString();

    } catch (IOException e) {

        e.printStackTrace();

        return "error";

    } catch (XmlPullParserException e) {

        e.printStackTrace();

        return "error";

    }

    return restex;

}

```

```

public static String registerService(String user, String pass, String mail,
    String mobile, String method) {

    String restex=null;

    SoapObject soap=new SoapObject(NAMESPACE, method);

    soap.addProperty("username",user);

    soap.addProperty("password",pass);

    soap.addProperty("mail",mail);

    soap.addProperty("mobile",mobile);

    SoapSerializationEnvelope
    envelope=new
    SoapSerializationEnvelope(SoapEnvelope.VER11);

    envelope.setOutputSoapObject(soap);

    HttpTransportSE http=new HttpTransportSE(URL);

```

```

try {

    http.call(NAMESPACE+method, envelope);

    SoapPrimitive primitive =(SoapPrimitive) envelope.getResponse();

    restex=primitive.toString();


} catch (IOException e) {

    e.printStackTrace();

    return "error";


} catch (XmlPullParserException e) {

    e.printStackTrace();

    return "error";

}

return restex;

}

public static String getDeviceList1(String from, String method) {

    String restex=null;

    SoapObject soap=new SoapObject(NAMESPACE, method);

    soap.addProperty("from",from);

//    soap.addProperty("password",password);

    SoapSerializationEnvelope envelope=new
SoapSerializationEnvelope(SoapEnvelope.VER11);

    envelope.setOutputSoapObject(soap);

```

```

HttpTransportSE http=new HttpTransportSE(URL);
try {
    http.call(NAMESPACE+method, envelope);
    SoapPrimitive primitive =(SoapPrimitive) envelope.getResponse();
    restex=primitive.toString();

} catch (IOException e) {
    e.printStackTrace();
    return "error";

} catch (XmlPullParserException e) {
    e.printStackTrace();
    return "error";

}

return restex;

}

```

```

public static String BankDetailsService(String s1, String s2, String s3,
    String s4, String s5, String method) {
    // TODO Auto-generated method stub
    String restex=null;
    SoapObject soap=new SoapObject(NAMESPACE, method);

```

```

        soap.addProperty("name",s1);
        soap.addProperty("accountno",s2);
        soap.addProperty("cvv",s3);
        soap.addProperty("expirydate",s4);
        soap.addProperty("password",s5);

        SoapSerializationEnvelope
        SoapSerializationEnvelope(SoapEnvelope.VER11);
        envelope=new
        envelope.setOutputSoapObject(soap);

        HttpTransportSE http=new HttpTransportSE(URL);
        try {
            http.call(NAMESPACE+method, envelope);

            SoapPrimitive primitive =(SoapPrimitive) envelope.getResponse();
            restex=primitive.toString();

        } catch (IOException e) {
            e.printStackTrace();
            return "error";

        } catch (XmlPullParserException e) {
            e.printStackTrace();
            return "error";

        }

        return restex;

    }

```

```

public static String bankloginService(String s1, String s2, String method) {
    // TODO Auto-generated method stub

    String restex=null;

    SoapObject soap=new SoapObject(NAMESPACE, method);
    soap.addProperty("user_name",s1);
    soap.addProperty("password",s2);

    SoapSerializationEnvelope envelope=new
SoapSerializationEnvelope(SoapEnvelope.VER11);

    envelope.setOutputSoapObject(soap);

    HttpTransportSE http=new HttpTransportSE(URL);

    try {

        http.call(NAMESPACE+method, envelope);

        SoapPrimitive primitive =(SoapPrimitive)envelope.getResponse();

        restex=primitive.toString();

    } catch (IOException e) {

        e.printStackTrace();

        return "error";

    } catch (XmlPullParserException e) {

        e.printStackTrace();

        return "error";

    }

    return restex;
}

```

```
}
```

```
/*public static String bookService(String qrvalue, String price, String cvv,  
    String method) {  
    // TODO Auto-generated method stub  
    String restex=null;  
    SoapObject soap=new SoapObject(NAMESPACE, method);  
    soap.addProperty("qrvalue",qrvalue);  
    soap.addProperty("price",price);  
    soap.addProperty("cvv",cvv);  
    SoapSerializationEnvelope envelope=new  
    SoapSerializationEnvelope(SoapEnvelope.VER11);  
    envelope.setOutputSoapObject(soap);  
    HttpTransportSE http=new HttpTransportSE(URL);  
    try {  
        http.call(NAMESPACE+method, envelope);  
        SoapPrimitive primitive =(SoapPrimitive) envelope.getResponse();  
        restex=primitive.toString();  
  
    } catch (IOException e) {  
        e.printStackTrace();  
        return "error";  
  
    } catch (XmlPullParserException e) {  
        e.printStackTrace();  
        return "error";  
    }  
}
```

```

    }

    return restex;

}*/

public static String bookService(String phno, String method) {
    // TODO Auto-generated method stub

    String restex=null;

    SoapObject soap=new SoapObject(NAMESPACE, method);

    soap.addProperty("phone",phno);

    SoapSerializationEnvelope envelope=new
    SoapSerializationEnvelope(SoapEnvelope.VER11);

    envelope.setOutputSoapObject(soap);

    HttpTransportSE http=new HttpTransportSE(URL);

    try {

        http.call(NAMESPACE+method, envelope);

        SoapPrimitive primitive =(SoapPrimitive) envelope.getResponse();

        restex=primitive.toString();

    } catch (IOException e) {

        e.printStackTrace();

        return "error";

    } catch (XmlPullParserException e) {

        e.printStackTrace();

        return "error";

```

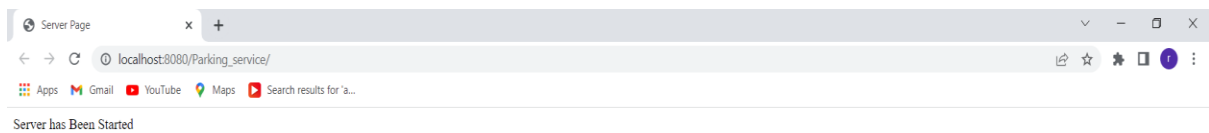
```
}
```

```
return restex;
```

```
}
```

```
}
```

OUTPUT:



APPLICATION SNAPSHOTS:

The image displays two screenshots of a mobile application interface. The top screenshot shows the 'Login Page' with a header bar containing a user icon and the word 'Login'. Below the header, the text 'Login Page' is centered. There are two input fields: the first contains the text 'ritsia' and the second contains three dots '...'. Below these fields are two buttons labeled 'signin' and 'signup'. The bottom screenshot shows the 'Register Page' with a header bar containing a user icon and the word 'Register'. Below the header, the text 'Register Page' is centered. There are four input fields: the first contains 'ritsia', the second contains three dots '...', the third contains 'ritsiababu@gmail.com', and the fourth contains '9150642650'. At the bottom of the page is a button labeled 'Submit'.

5554:POCO

Login

Login Page

ritsia

...

signin

signup

5554:POCO

Register

Register Page

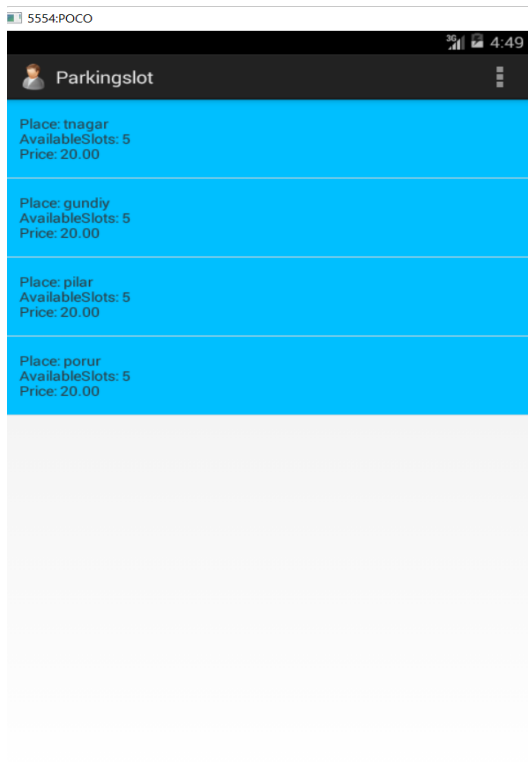
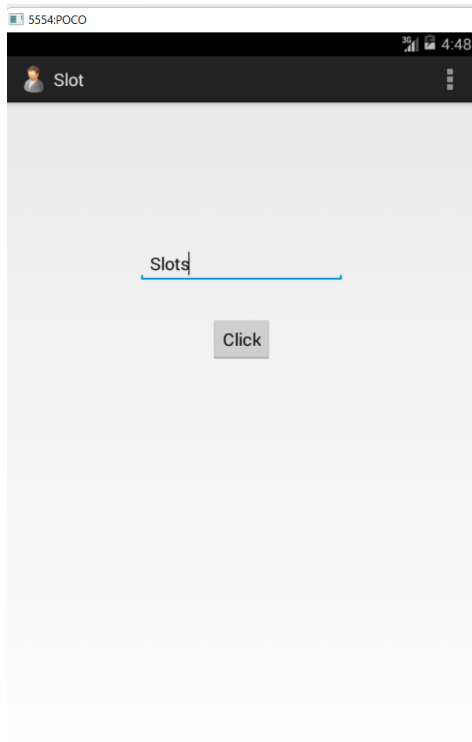
ritsia

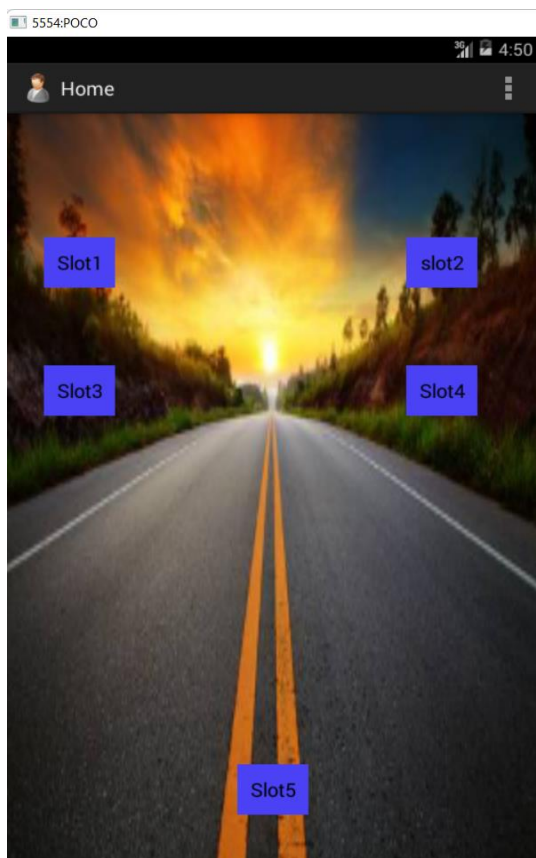
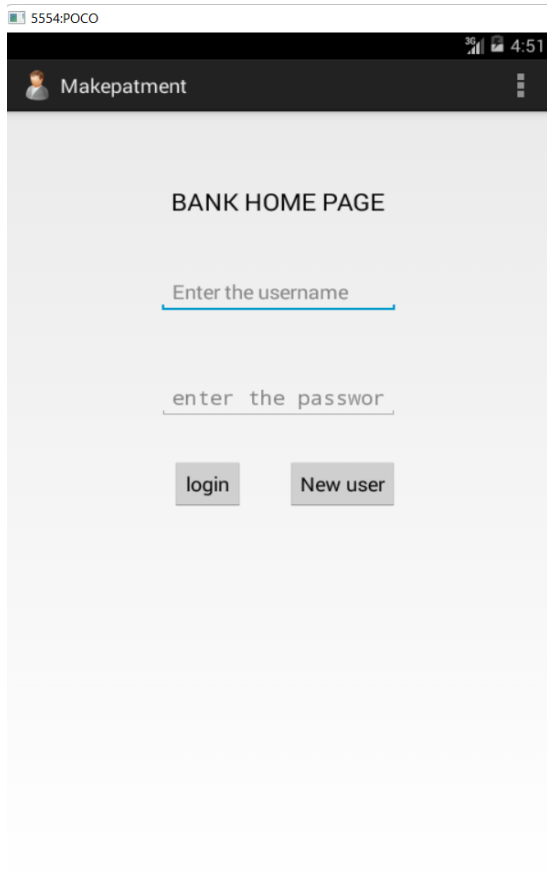
...

ritsiababu@gmail.com

9150642650

Submit





5554:POCO 4:53

Bregister

Enter Bank details

ritsia

1234567890

456

17/08/2022

...

SUBMIT

5554:POCO 4:53

Bregister

Enter Bank details

ritsia

1234567890

456

trus again

Account Created

Got it

SUBMIT

