**Abstract**

The main function of Universal Online Exam Registration System is to provide a full exam registration service to students who are applying for their Semester Exams. This system provides a fleet of services to the students like Semester exams notifications, last dates for submission, paying exam fees online and raising requests to exam branch of the respective college to submit their application to the JNTUH as per student requests. This system provides each student a separate profile/account in which the students have to register first at once with valid credentials. For security purpose fingerprints are used for students and exam branch privacy. Students after registering in the website will pay their exam fees online and raise the request to exam branch to submit their details to JNTUH. Exam branch processes the raised student’s requests with in a period of time. After every successful registration a confirmation message is sent to student’s email/SMS which is provided while student registration.

**Introduction:**

Exam Registration System is an interface between the Student and the Exam Controller responsible for the Issue exam notification and student exam registration requests. It aims at improving the efficiency in the exam branch and reduces the complexities involved in it to the maximum possible extent.

Our Exam Registration System acts as an interface between the 'student' and the 'exam controller'. This system tries to make the interface as simple as possible and at the same time not risking the security of data stored in. This minimizes the time duration.

**Existing System:**

If the entire process of Exam Registration is done in a manual manner then it would takes student to apply more amount of time and they have to wait in queue.

**Disadvantages:**

* Students will come to know their status of application and the date in which they must subject themselves for manual document verification
* Time Consuming
* In Accurate Data
* No confirmation about registration

**Proposed System:**

Exam Registration system Is used in the effective dispatch of registration notification to all of the students. This system adopts a comprehensive approach to minimize the manual work and schedule resources, time in a cogent manner. The core of the system is to get the online registration form (with details such as name, transaction number etc.,) filled by the student whose testament is verified for its genuineness by the Exam Registration System with respect to the already existing information in the database. This forms the first and foremost step in the processing of exam application. After the first round of verification done by the system, the information is in turn forwarded to the Exam Controller. The application is then processed manually based on the report given by the system. The system also provides the student the list of exam dates.

**Advantages:**

* Secure Registration of information by the Students.
* Mail updates to the students by the controller.
* Controller can generate reports from the information and is the only authorized personnel to add the eligible application information to the database.

**STUDY OF THE SYSTEM**

To provide flexibility to the users, the interfaces have been developed that are accessible through a browser. The GUI’S at the top level have been categorized as

**Analysis**

Although the scale of this project is relatively small, to produce a professional solution is it imperative that the current problem is understood accurately. However, this task has been made doubly difficult by the lack of support from the company. Thankfully, the Application manager has been kind enough to spare me some of his own time to discuss the problem with me further. Therefore, this chapter is concerning with analyzing the current situation and expectations of the user for this system.

**Requirements**

The minimum requirements of the project are listed below:

* Examine the tools and methodologies required to gain an overview of the system requirements for the proposed database.
* Examine suitable database management systems that can be used to implement the proposed database.
* Evaluate appropriate website authoring and web graphic creation tools that can be used to develop web based forms for the proposed database
* Produce and apply suitable criteria for evaluating the solution

**FUNCTIONAL REQUIREMENTS:**

Functional requirement should include function performed by a specific screen outline work-flows performed by the system and other business or compliance requirement the system must meet.

Functional requirements specify which output file should be produced from the given file they describe the relationship between the input and output of the system, for each functional requirement a detailed description of all data inputs and their source and the range of valid inputs must be specified.

The functional specification describes what the system must do, how the system does it is described in the design specification.

If a user requirement specification was written, all requirements outlined in the user requirements specifications should be addressed in the functional requirements.

**NON-FUNCTIONAL REQUIREMENTS:**

Describe user-visible aspects of the system that are not directly related with the functional behavior of the system. Non-Functional requirements include quantitative constraints, such as response time (i.e. how fast the system reacts to user commands.) or accuracy (.e. how precise are the systems numerical answers.).

**UI Requirements**

1. **Administrative user interface**

The ‘administrative user interface’ concentrates on the consistent information that is practically, part of the organizational activities and which needs proper authentication for the data collection. These interfaces help the administrators with all the transactional states like Data insertion, Data deletion and Date updating along with the extensive data search capabilities.

1. **The operational or generic user interface**

The ‘operational or generic user interface’ helps the end users of the system in transactions through the existing data and required services. The operational user interface also helps the ordinary users in managing their own information in a customized manner as per the includedFlexibilities. This application consists following modules.

Software Requirements:

* Windows XP, Windows 7(ultimate, enterprise)
* Android SDK

Hardware Components:

* Processor – i3
* Hard Disk – 5 GB
* Memory – 1GB RAM

**Modules:**

1. Student Login
2. Exam branch(Admin)
3. Payment

**FEASIBILITY STUDY**

The next step in analysis is to verify the feasibility of the proposed system. “All projects are feasible given unlimited resources and infinite time“. But in reality both resources and time are scarce. Project should confirm to time bounce and should be optimal in there consumption of resources. These places a constant are approval of any project.

Feasibility has applied to Maintenance of Elementary School Data pertains to the following areas:

* Technical feasibility
* Operational feasibility
* Economical feasibility

**TECHNICAL FEASIBILITY:**

To determine whether the proposed system is technically feasible, we should take into consideration the technical issues involved behind the system.

Maintenance of Elementary School Data uses the web technologies, which is rampantly employed these days worldwide. The world without the web is incomprehensible today. That goes to proposed system is technically feasible.

**OPERATIONAL FEASIBILITY:**

To determine the operational feasibility of the system we should take into consideration the awareness level of the users. This system is operational feasible since the users are familiar with the technologies and hence there is no need to gear up the personnel to use system. Also the system is very friendly and to use.

**ECONOMIC FEASIBILITY**

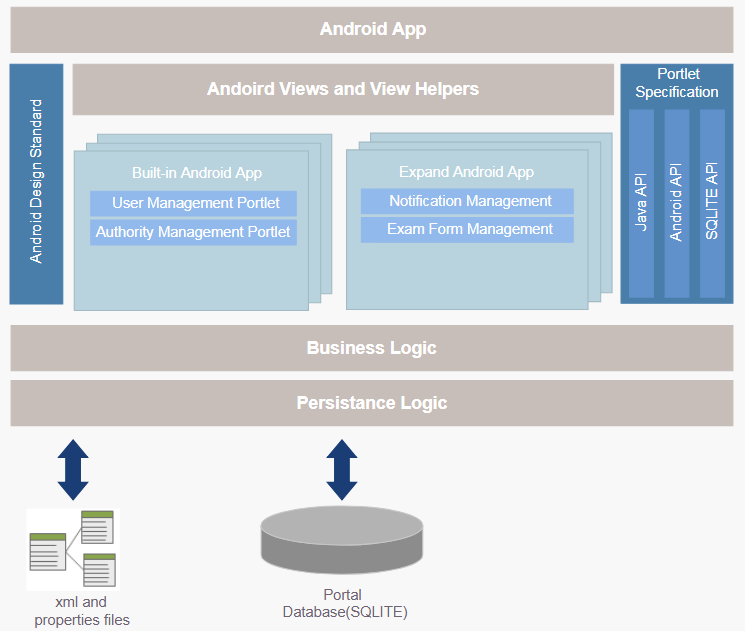
To decide whether a project is economically feasible, we have to consider various factors as:

* + - * Cost benefit analysis
      * Long-term returns
      * Maintenance costs

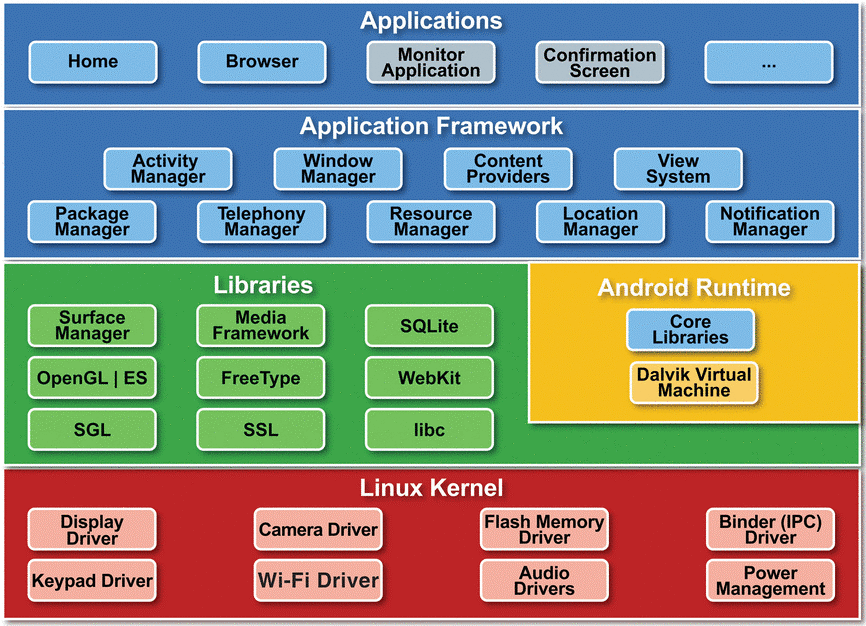
The proposed Maintenance of Elementary School Data is computer based. It requires average computing capabilities and access to internet, which are very basic requirements and can be afforded by any organization hence it doesn’t incur additional economic overheads, which renders the system economically feasible.

**ARCHITECTURE**

**System Architecture:**

****

**echnical Architecture:**



**SOFTWARE REQUIREMENT SPECIFICATION**

**What is SRS?**

Software Requirements Specification (SRS) is the starting point of the software developing activity. As system grew more complex it became evident that the goal of the entire system cannot be easily comprehended. Hence the need for the requirement phase arose. The software project is initiated by the client needs. The SRS is the means of translating the ideas of the minds of clients (the input) into a formal document (the output of the requirement phase).

The SRS phase consists of two basic activities:

**Problem/Requirement Analysis:**

The process is order and more nebulous of the two, deals with understand the problem, the goal and constraints.

**Requirement Specification:**

Here, the focus is on specifying what has been found giving analysis such as representation, Specification languages and tools, and checking the specifications are addressed during this activity.

The requirement phase terminates with the production of the validate SRS document. Producing the SRS document is the basic of this phase.

**Role of SRS:**

The purpose of the SRS is to reduce the communication gap between the clients and the developers. SRS is the medium though which the client and user needs are accurately specified.

It forms the basis of software development. A good SRS should satisfy all the parties involved in the system.

**Purpose:**

The purpose of this document is to describe all external requirements for the E-learning System. It also describes the interfaces for the system.

* Scope**:**

This document is the only one that describes the requirements of the system. It is meant for the use by the developers, and will also by the basis for validating the final deliver system. Any changes made to the requirements in the future will have to go through a formal change approval process. The developer is responsible for asking for clarifications, where necessary, and will not make any alternations without the permission of the client.

* Overview**:**

The SRS begins the translation process that converts the software Requirements into the language the developers will use. The SRS draws on the Use Cases from the user Requirement Document and analyses the situations from a number of perspectives to discover and eliminate inconsistencies, ambiguities and omissions before development progresses significantly under mistaken assumptions.

**Proposed System Architecture:**

The proposed system is built around conventional three-tier architecture. The three-tier architecture for web development allows programmers to separate various aspects of the solution design into modules and work on them separately. That is, a developer who is best at one part of development, say UI development need not worry about the implementation levels so much. It also allows for easy maintenance and future enhancements. The three-tiers of the solution include:

* The Layout**:**

This tier is at the uppermost layer and is closely bound to the user, i.e., the users of the system interact with it through this tier.

* The business-tier**:**

This tier is responsible for implementing all the business rules of the organization. It operates on the data provided by the users through the web-tier and the data stored in the underlying data-tier. So in a way this tier works on data from the web-tier and the data-tier in order to perform task for the users in agreement with the business rules of the organization.

* The data-tier**:**

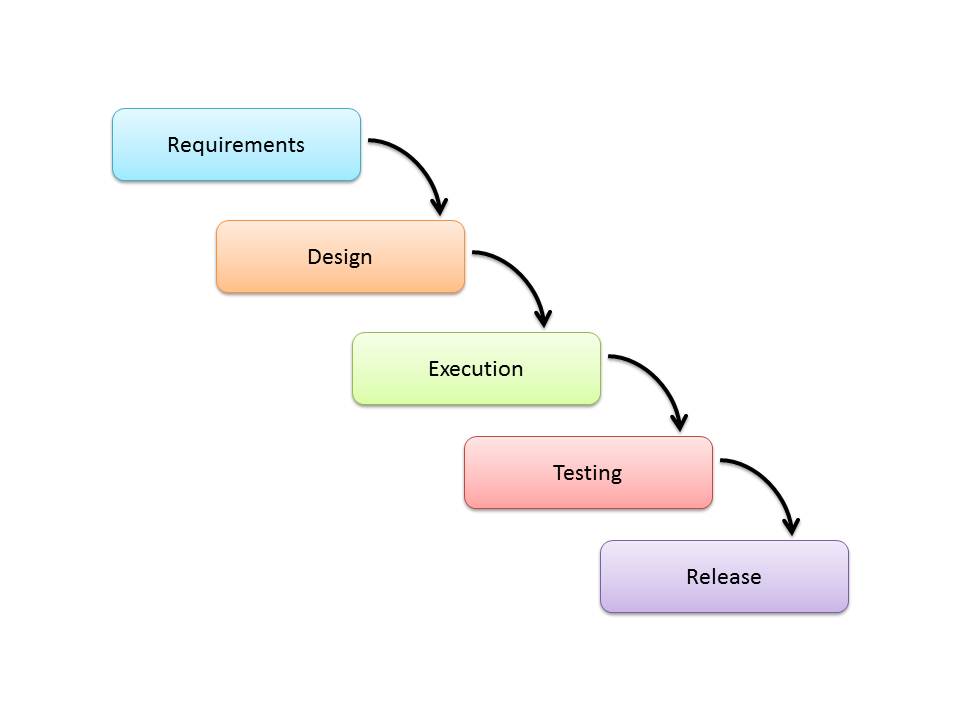
This tier contains the persist able data that is required by the business tier to operate on. Data plays a very important role in the functioning of any organization. Thus, persisting of such data is very important. The data tier performs the job of persisting the data.

**Life Cycle Models and Methodologies:**

**Waterfall Model**

**Description:**

The Waterfall Model is a linear sequential flow. In which progress is seen as flowing steadily downwards (like a waterfall) through the phases of software implementation. This means that any phase in the development process begins only if the previous phase is complete. The waterfall approach does not define the process to go back to the previous phase to handle changes in requirement. The waterfall approach is the earliest approach and most widely known that was used for software development.



The usage

Projects which not focus on changing the requirements, for example, projects initiated from request for proposals (RFPs), the customer has a very clear documented requirements

**Advantages and Disadvantages:**

**Advantages:**

* Easy to explain to the users.
* Structures approach.
* Stages and activities are well defined.
* Helps to plan and schedule the project.
* Verification at each stage ensures early detection of errors/misunderstanding.
* Each phase has specific deliverables.

**Disadvantages:**

* Assumes that the requirements of a system can be frozen.
* Very difficult to go back to any stage after it finished.
* A little flexibility and adjusting scope is difficult and expensive.
* Costly and required more time, in addition to the detailed plan.

**ANDROID Technical ENVIRONMENT**

Android apps are written in the Java programming language. The Android SDK tools compile your code—along with any data and resource files—into an APK: an Android package, which is an archive file with an .apk suffix. One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app.

Once installed on a device, each Android app lives in its own security sandbox: The Android operating system is a multi-user Linux system in which each app is a different user.

By default, the system assigns each app a unique Linux user ID (the ID is used only by the system and is unknown to the app). The system sets permissions for all the files in an app so that only the user ID assigned to that app can access them. Each process has its own virtual machine (VM), so an app's code runs in isolation from other apps.

By default, every app runs in its own Linux process. Android starts the process when any of the app's components need to be executed, then shuts down the process when it's no longer needed or when the system must recover memory for other apps.

In this way, the Android system implements the principle of least privilege. That is, each app, by default, has access only to the components that it requires to do its work and no more. This creates a very secure environment in which an app cannot access parts of the system for which it is not given permission.

However, there are ways for an app to share data with other apps and for an app to access system services: It's possible to arrange for two apps to share the same Linux user ID, in which case they are able to access each other's files. To conserve system resources, apps with the same user ID can also arrange to run in the same Linux process and share the same VM (the apps must also be signed with the same certificate).

An app can request permission to access device data such as the user's contacts, SMS messages, the mountable storage (SD card), camera, Bluetooth, and more. All app permissions must be granted by the user at install time. That covers the basics regarding how an Android app exists within the system. The rest of this document introduces you to: The core framework components that define your app.

The manifest file in which you declare components and required device features for your app.

Resources that are separate from the app code and allow your app to gracefully optimize its behavior for a variety of device configurations.

**App Components**

App components are the essential building blocks of an Android app. Each component is a different point through which the system can enter your app. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role—each one is a unique building block that helps define your app's overall behavior.

There are four different types of app components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

Here are the four types of app components:

**Activities**

An activity represents a single screen with a user interface. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others. As such, a different app can start any one of these activities (if the email app allows it). For example, a camera app can start the activity in the email app that composes new mail, in order for the user to share a picture.

An activity is implemented as a subclass of Activity and you can learn more about it in the Activities developer guide.

**Services**

A service is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.

A service is implemented as a subclass of Service and you can learn more about it in the Services developer guide.

**Content providers**

A content provider manages a shared set of app data. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your app can access. Through the content provider, other apps can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information. As such, any app with the proper permissions can query part of the content provider (such as Contacts Contract. Data) to read and write information about a particular person.

Content providers are also useful for reading and writing data that is private to your app and not shared. For example, the Note Pad sample app uses a content provider to save notes.

A content provider is implemented as a subclass of Content Provider and must implement a standard set of APIs that enable other apps to perform transactions. For more information, see the Content Providers developer guide.

**Broadcast receivers**

A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.

A broadcast receiver is implemented as a subclass of Broadcast Receiver and each broadcast is delivered as an Intent object. For more information, see the Broadcast Receiver class.

A unique aspect of the Android system design is that any app can start another app’s component. For example, if you want the user to capture a photo with the device camera, there's probably another app that does that and your app can use it, instead of developing an activity to capture a photo yourself. You don't need to incorporate or even link to the code from the camera app. Instead, you can simply start the activity in the camera app that captures a photo. When complete, the photo is even returned to your app so you can use it. To the user, it seems as if the camera is actually a part of your app.

When the system starts a component, it starts the process for that app (if it's not already running) and instantiates the classes needed for the component. For example, if your app starts the activity in the camera app that captures a photo, that activity runs in the process that belongs to the camera app, not in your app's process. Therefore, unlike apps on most other systems, Android apps don't have a single entry point (there's no main() function, for example).

Because the system runs each app in a separate process with file permissions that restrict access to other apps, your app cannot directly activate a component from another app. The Android system, however, can. So, to activate a component in another app, you must deliver a message to the system that specifies your intent to start a particular component. The system then activates the component for you.

**Activating Components**

Three of the four component types—activities, services, and broadcast receivers—are activated by an asynchronous message called an intent. Intents bind individual components to each other at runtime (you can think of them as the messengers that request an action from other components), whether the component belongs to your app or another.

An intent is created with an Intent object, which defines a message to activate either a specific component or a specific type of component—an intent can be either explicit or implicit, respectively.

For activities and services, an intent defines the action to perform (for example, to "view" or "send" something) and may specify the URI of the data to act on (among other things that the component being started might need to know). For example, an intent might convey a request for an activity to show an image or to open a web page. In some cases, you can start an activity to receive a result, in which case, the activity also returns the result in an Intent (for example, you can issue an intent to let the user pick a personal contact and have it returned to you—the return intent includes a URI pointing to the chosen contact).

For broadcast receivers, the intent simply defines the announcement being broadcast (for example, a broadcast to indicate the device battery is low includes only a known action string that indicates "battery is low").

The other component type, content provider, is not activated by intents. Rather, it is activated when targeted by a request from a ContentResolver. The content resolver handles all direct transactions with the content provider so that the component that's performing transactions with the provider doesn't need to and instead calls methods on the Content Resolver object. This leaves a layer of abstraction between the content provider and the component requesting information (for security).

There are separate methods for activating each type of component:

You can start an activity (or give it something new to do) by passing an Intent to startActivity() or startActivityForResult() (when you want the activity to return a result).

You can start a service (or give new instructions to an ongoing service) by passing an Intent to startService(). Or you can bind to the service by passing an Intent to bindService().

You can initiate a broadcast by passing an Intent to methods like sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast().

You can perform a query to a content provider by calling query() on a ContentResolver.

For more information about using intents, see the Intents and Intent Filters document. More information about activating specific components is also provided in the following documents: Activities, Services, Broadcast Receiver and Content Providers.

**The Manifest File**

Before the Android system can start an app component, the system must know that the component exists by reading the app's AndroidManifest.xml file (the "manifest" file). Your app must declare all its components in this file, which must be at the root of the app project directory.

The manifest does a number of things in addition to declaring the app's components, such as: Identify any user permissions the app requires, such as Internet access or read-access to the user's contacts.

Declare the minimum API Level required by the app, based on which APIs the app uses. Declare hardware and software features used or required by the app, such as a camera, bluetooth services, or a multi touch screen. API libraries the app needs to be linked against (other than the Android framework APIs), such as the Google Maps library and more Declaring components

The primary task of the manifest is to inform the system about the app's components. For example, a manifest file can declare an activity as follows:

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

<manifest ... >

<application android:icon="@drawable/app\_icon.png" ... >

<activity android:name="com.example.project.ExampleActivity"

android:label="@string/example\_label" ... >

</activity>

...

</application>

</manifest>

In the <application> element, the android:icon attribute points to resources for an icon that identifies the app.

In the <activity> element, the android:name attribute specifies the fully qualified class name of the Activity subclass and the android:label attributes specifies a string to use as the user-visible label for the activity.

You must declare all app components this way:

<activity> elements for activities

<service> elements for services

<receiver> elements for broadcast receivers

<provider> elements for content providers

Activities, services, and content providers that you include in your source but do not declare in the manifest are not visible to the system and, consequently, can never run. However, broadcast receivers can be either declared in the manifest or created dynamically in code (as BroadcastReceiver objects) and registered with the system by calling registerReceiver().

For more about how to structure the manifest file for your app, see The AndroidManifest.xml File documentation.

**Declaring component capabilities**

As discussed above, in Activating Components, you can use an Intent to start activities, services, and broadcast receivers. You can do so by explicitly naming the target component (using the component class name) in the intent. However, the real power of intents lies in the concept of implicit intents. An implicit intent simply describes the type of action to perform (and, optionally, the data upon which you’d like to perform the action) and allows the system to find a component on the device that can perform the action and start it. If there are multiple components that can perform the action described by the intent, then the user selects which one to use.

The way the system identifies the components that can respond to an intent is by comparing the intent received to the intent filters provided in the manifest file of other apps on the device. When you declare an activity in your app's manifest, you can optionally include intent filters that declare the capabilities of the activity so it can respond to intents from other apps. You can declare an intent filter for your component by adding an <intent-filter> element as a child of the component's declaration element.

For example, if you've built an email app with an activity for composing a new email, you can declare an intent filter to respond to "send" intents (in order to send a new email) like this:

<manifest ... >  
    ...  
    <application ... >  
        <activity android:name="com.example.project.ComposeEmailActivity">  
            <intent-filter>  
                <action android:name="android.intent.action.SEND" />  
                <data android:type="\*/\*" />  
                <category android:name="android.intent.category.DEFAULT" />  
            </intent-filter>  
        </activity>  
    </application>  
</manifest>

Then, if another app creates an intent with the ACTION\_SEND action and pass it to startActivity(), the system may start your activity so the user can draft and send an email.

For more about creating intent filters, see the Intents and Intent Filters document.

**Declaring app requirements**

There are a variety of devices powered by Android and not all of them provide the same features and capabilities. In order to prevent your app from being installed on devices that lack features needed by your app, it's important that you clearly define a profile for the types of devices your app supports by declaring device and software requirements in your manifest file. Most of these declarations are informational only and the system does not read them, but external services such as Google Play do read them in order to provide filtering for users when they search for apps from their device.

For example, if your app requires a camera and uses APIs introduced in Android 2.1 (API Level 7), you should declare these as requirements in your manifest file like this:

<manifest ... >  
    <uses-feature android:name="android.hardware.camera.any"  
                  android:required="true" />  
    <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="19" />  
    ...  
</manifest>

Now, devices that do not have a camera and have an Android version lower than 2.1 cannot install your app from Google Play.

However, you can also declare that your app uses the camera, but does not require it. In that case, your app must set the required attribute to "false" and check at runtime whether the device has a camera and disable any camera features as appropriate.

More information about how you can manage your app's compatibility with different devices is provided in the Device Compatibility document.

**App Resources**

An Android app is composed of more than just code—it requires resources that are separate from the source code, such as images, audio files, and anything relating to the visual presentation of the app. For example, you should define animations, menus, styles, colors, and the layout of activity user interfaces with XML files. Using app resources makes it easy to update various characteristics of your app without modifying code and—by providing sets of alternative resources—enables you to optimize your app for a variety of device configurations (such as different languages and screen sizes).

For every resource that you include in your Android project, the SDK build tools define a unique integer ID, which you can use to reference the resource from your app code or from other resources defined in XML. For example, if your app contains an image file named logo.png (saved in the res/drawable/ directory), the SDK tools generate a resource ID named R.drawable.logo, which you can use to reference the image and insert it in your user interface.

One of the most important aspects of providing resources separate from your source code is the ability for you to provide alternative resources for different device configurations. For example, by defining UI strings in XML, you can translate the strings into other languages and save those strings in separate files. Then, based on a language qualifier that you append to the resource directory's name (such as res/values-fr/ for French string values) and the user's language setting, the Android system applies the appropriate language strings to your UI.

Android supports many different qualifiers for your alternative resources. The qualifier is a short string that you include in the name of your resource directories in order to define the device configuration for which those resources should be used. As another example, you should often create different layouts for your activities, depending on the device's screen orientation and size. For example, when the device screen is in portrait orientation (tall), you might want a layout with buttons to be vertical, but when the screen is in landscape orientation (wide), the buttons should be aligned horizontally. To change the layout depending on the orientation, you can define two different layouts and apply the appropriate qualifier to each layout's directory name. Then, the system automatically applies the appropriate layout depending on the current device orientation.

For more about the different kinds of resources you can include in your application and how to create alternative resources for different device configurations, read Providing Resources.

**About SQLite**

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format. SQLite database files are a recommended storage format by the US Library of Congress. Think of SQLite not as a replacement for Oracle but as a replacement for fopen()

SQLite is a compact library. With all features enabled, the library size can be less than 500KiB, depending on the target platform and compiler optimization settings. (64-bit code is larger. And some compiler optimizations such as aggressive function inlining and loop unrolling can cause the object code to be much larger.) There is a tradeoff between memory usage and speed. SQLite generally runs faster the more memory you give it. Nevertheless, performance is usually quite good even in low-memory environments. Depending on how it is used, SQLite can be faster than direct filesystem I/O.

SQLite is very carefully tested prior to every release and has a reputation for being very reliable. Most of the SQLite source code is devoted purely to testing and verification. An automated test suite runs millions and millions of test cases involving hundreds of millions of individual SQL statements and achieves 100% branch test coverage. SQLite responds gracefully to memory allocation failures and disk I/O errors. Transactions are ACID even if interrupted by system crashes or power failures. All of this is verified by the automated tests using special test harnesses which simulate system failures. Of course, even with all this testing, there are still bugs. But unlike some similar projects (especially commercial competitors) SQLite is open and honest about all bugs and provides bugs lists and minute-by-minute chronologies of code changes.

The SQLite code base is supported by an international team of developers who work on SQLite full-time. The developers continue to expand the capabilities of SQLite and enhance its reliability and performance while maintaining backwards compatibility with the published interface spec, SQL syntax, and database file format. The source code is absolutely free to anybody who wants it, but professional support is also available.

The SQLite project was started on 2000-05-09. The future is always hard to predict, but the intent of the developers is to support SQLite through the year 2050. Design decisions are made with that objective in mind.

We the developers hope that you find SQLite useful and we entreat you to use it well: to make good and beautiful products that are fast, reliable, and simple to use. Seek forgiveness for yourself as you forgive others. And just as you have received SQLite for free, so also freely give, paying the debt forward.

**SYSTEM DESIGN**

System design is transition from a user oriented document to programmers or data base personnel. The design is a solution, how to approach to the creation of a new system. This is composed of several steps. It provides the understanding and procedural details necessary for implementing the system recommended in the feasibility study. Designing goes through logical and physical stages of development, logical design reviews the present physical system, prepare input and output specification, details of implementation plan and prepare a logical design walkthrough.

The database tables are designed by analyzing functions involved in the system and format of the fields is also designed. The fields in the database tables should define their role in the system. The unnecessary fields should be avoided because it affects the storage areas of the system. Then in the input and output screen design, the design should be made user friendly. The menu should be precise and compact.

**SOFTWARE DESIGN**

In designing the software following principles are followed:

1. **Modularity and partitioning**: software is designed such that, each system should consists of hierarchy of modules and serve to partition into separate function.

2. **Coupling:** modules should have little dependence on other modules of a system.

3. **Cohesion:** modules should carry out in a single processing function.

4. **Shared use:** avoid duplication by allowing a single module be called by other that need the function it provide

**INPUT/OUTPUT DESIGN**

**Input design:** considering the requirements, procedures to collect the necessary input data in most efficiently designed. The input design has been done keeping in view that, the interaction of the user with the system being the most effective and simplified way.

Also the measures are taken for the following

* + - Controlling the amount of input
    - Avoid unauthorized access to the classroom.
    - Eliminating extra steps
    - Keeping the process simple
    - At this stage the input forms and screens are designed.

**Output design:** All the screens of the system are designed with a view to provide the user with easy operations in simpler and efficient way, minimum key strokes possible. Instructions and important information is emphasized on the screen. Almost every screen is provided with no error and important messages and option selection facilitates. Emphasis is given for speedy processing and speedy transaction between the screens. Each screen assigned to make it as much user friendly as possible by using interactive procedures. So to say user can operate the system without much help from the operating manual.

UML Concepts

The Unified Modelling Language (UML) is a standard language for writing software blue prints. The UML is a language for

* Visualizing
* Specifying
* Constructing
* Documenting the artefacts of a software intensive system.

The UML is a language which provides vocabulary and the rules for combining words in that vocabulary for the purpose of communication. A modelling language is a language whose vocabulary and the rules focus on the conceptual and physical representation of a system. Modelling yields an understanding of a system.

Building Blocks of the UML:

The vocabulary of the UML encompasses three kinds of building blocks:

* Things
* Relationships
* Diagrams

Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group interesting collections of things.

Things in the UML:

There are four kinds of things in the UML:

* Structural things
* Behavioral things
* Grouping things
* Annotational things

**Structural things** are the nouns of UML models. The structural things used in the project design are: First, a **class** is a description of a set of objects that share the same attributes, operations, relationships and semantics.

|  |
| --- |
| Window |
| origin  size |
| open()  close()  move()  display() |

##### Fig: Classes

Second, a **use case** is a description of set of sequence of actions that a system performs that yields an observable result of value to particular actor.



**Fig: Use Cases**

Third, a node is a physical element that exists at runtime and represents a computational resource, generally having at least some memory and often processing capability.

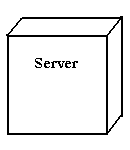


Fig: Nodes

**Behavioural things** are the dynamic parts of UML models. The behavioural thing used is:

**Interaction:**

An interaction is a behaviour that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose. An interaction involves a number of other elements, including messages, action sequences (the behaviour invoked by a message, and links (the connection between objects).



**Fig: Messages**

**5.1.3 Relationships in the UML:**

There are four kinds of relationships in the UML:

* Dependency
* Association
* Generalization
* Realization

A **dependency** is a semantic relationship between two things in which a change to one thing may affect the semantics of the other thing (the dependent thing).



**Fig: Dependencies**

An **association** is a structural relationship that describes a set links, a link being a connection among objects. Aggregation is a special kind of association, representing a structural relationship between a whole and its parts.



**Fig: Association**

A **generalization** is a specialization/ generalization relationship in which objects of the specialized element (the child) are substitutable for objects of the generalized element (the parent).



**Fig: Generalization**

A **realization** is a semantic relationship between classifiers, where in one classifier specifies a contract that another classifier guarantees to carry out.



Fig: Realization

## Sequence Diagrams:

UML sequence diagrams are used to represent the flow of messages, events and actions between the objects or components of a system. Time is represented in the vertical direction showing the sequence of interactions of the header elements, which are displayed horizontally at the top of the diagram.

Sequence Diagrams are used primarily to design, document and validate the architecture, interfaces and logic of the system by describing the sequence of actions that need to be performed to complete a task or scenario. UML sequence diagrams are useful design tools because they provide a dynamic view of the system behaviour which can be difficult to extract from static diagrams or specifications.

**Actor**

Represents an external person or entity that interacts with the system

**Sequence diagram actor element**

**Fig: Actor**

**Object**

Represents an object in the system or one of its components

**Sequence diagram object element**

**Fig: Object**

**Unit**

Represents a subsystem, component, unit, or other logical entity in the system (may or may not be implemented by objects)

**Sequence diagram unit element**

**Fig: Unit**

**Separator**

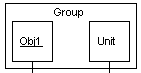
Represents an interface or boundary between subsystems, components or units (e.g., air interface, Internet, network)

**Sequence diagram separator element**

**Fig: Seperator**

**Group**

Groups related header elements into subsystems or components

****

**Fig: Group**

Sequence Diagram Body Elements

**Action**

Represents an action taken by an actor, object or unit

**Sequence diagram action element**

**Fig: Action**

**Asynchronous Message**

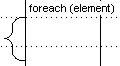
An asynchronous message between header elements

**Sequence diagram asynchronous message element**

**Fig: Asynchronous Message**

**Block**

A block representing a loop or conditional for a particular header element

****

**Fig: Block**

**Call Message**

A call (procedure) message between header elements

**Sequence diagram call message element**

**Fig: Call Message**

**Create Message**

A "create" message that creates a header element (represented by lifeline going from dashed to solid pattern)

**Sequence diagram create message element**

**Fig: Create Message**

**Diagram Link**

Represents a portion of a diagram being treated as a functional block. Similar to a procedure or function call that abstracts functionality or details not shown at this level. Can optionally be linked to another diagram for elaboration.

**Sequence diagram diagram link element**

**Fig: Link**

Else Block Represents an "else" block portion of a diagram block

**Sequence diagram else block element**

**Fig: Else**

**Message**

A simple message between header elements

**Sequence diagram message element**

**Fig: Message**

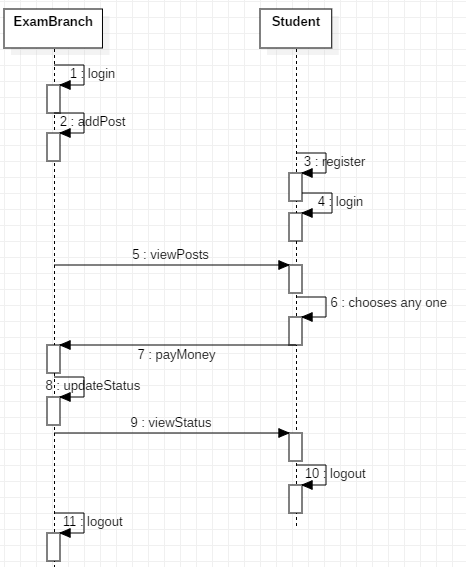
**Return Message**

A return message between header elements

**Sequence diagram return message element**

**Fig: Return**

**Sequence diagram**

****

## Use Case Diagram

A use case diagram is a graph of actors set of use cases enclosed by a system boundary, communication associations between the actors and users and generalization among use cases. The use case model defines the outside(actors) and inside(use case) of the system’s behavior.

use case diagram is quite simple in nature and depicts two types of elements: one representing the business roles and the other representing the business processes.

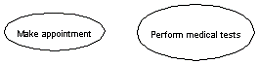


**Figure : actor**

To identify an actor, search in the problem statement for business terms that portray roles in the system. For example, in the statement "patients visit the doctor in the clinic for medical tests," "doctor" and "patients" are the business roles and can be easily identified as actors in the system.

**Use case:** A use case in a use case diagram is a visual representation of a distinct business functionality in a system. The key term here is "distinct business functionality." To choose a business process as a likely candidate for modelling as a use case, you need to ensure that the business process is discrete in nature.

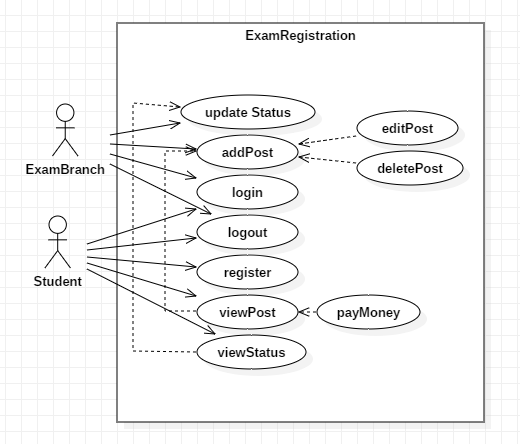
As the first step in identifying use cases, you should list the discrete business functions in your problem statement. Each of these business functions can be classified as a potential use case. Remember that identifying use cases is a discovery rather than a creation. As business functionality becomes clearer, the underlying use cases become more easily evident. A use case is shown as an ellipse in a use case diagram (see Figure ).



**Figure : use case**

Figure shows two uses cases: "Make appointment" and "Perform medical tests" in the use case diagram of a clinic system. As another example, consider that a business process such as "manage patient records" can in turn have sub-processes like "manage patient's personal information" and "manage patient's medical information." Discovering such implicit use cases is possible only with a thorough understanding of all the business processes of the system through discussions with potential users of the system and relevant domain knowledge.

Usecase:



## Activity Diagram

Activity diagrams represent the business and operational workflows of a system. An Activity diagram is a dynamic diagram that shows the activity and the event that causes the object to be in the particular state.

So, what is the importance of an Activity diagram, as opposed to a State diagram? A State diagram shows the different states an object is in during the lifecycle of its existence in the system, and the transitions in the states of the objects. These transitions depict the activities causing these transitions, shown by arrows.

An Activity diagram talks more about these transitions and activities causing the changes in the object states.

#### Defining an Activity diagram

Let us take a look at the building blocks of an Activity diagram.

#### Elements of an Activity diagram

An Activity diagram consists of the following behavioural elements:

**Initial Activity:** This shows the starting point or first activity of the flow. Denoted by a solid circle. This is similar to the notation used for Initial State.

http://www.developer.com/img/articles/2003/08/11/UML07T1.gif

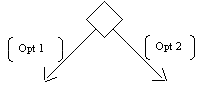
**Fig: Initial State**

**Activity:** Represented by a rectangle with rounded (almost oval) edges.

.http://www.developer.com/img/articles/2003/08/11/UML07T2.gif

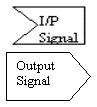
**Fig: Action State**

**Decisions:** Similar to flowcharts, a logic where a decision is to be made is depicted by a diamond, with the options written on either sides of the arrows emerging from the diamond, within box brackets.



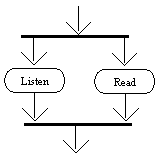
**Fig: Decision**

**Signal:** When an activity sends or receives a message, that activity is called a signal. Signals are of two types: Input signal (Message receiving activity) shown by a concave polygon and Output signal (Message sending activity) shown by a convex polygon.



**Fig: Signal**

**Concurrent Activities:** Some activities occur simultaneously or in parallel. Such activities are called concurrent activities. For example, listening to the lecturer and looking at the blackboard is a parallel activity. This is represented by a horizontal split (thick dark line) and the two concurrent activities next to each other, and the horizontal line again to show the end of the parallel activity.

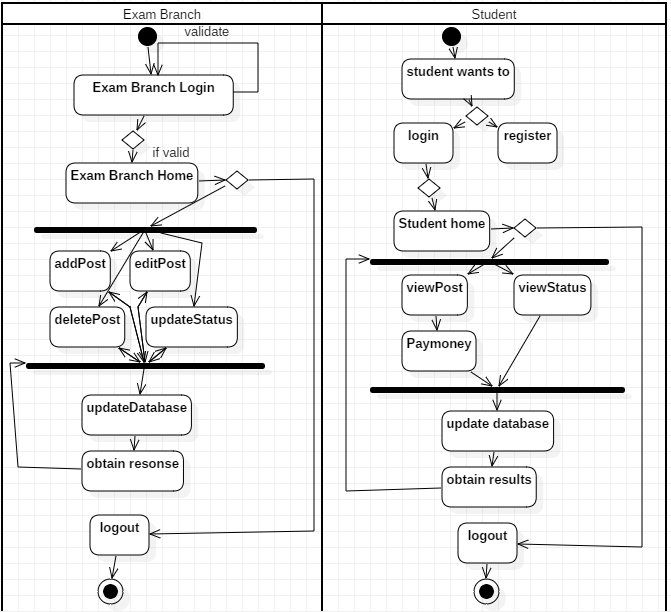


**Fig: Horizontal**

**Final Activity:** The end of the Activity diagram is shown by a bull's eye symbol, also called as a final activity.

http://www.developer.com/img/articles/2003/08/11/UML07T6.gif

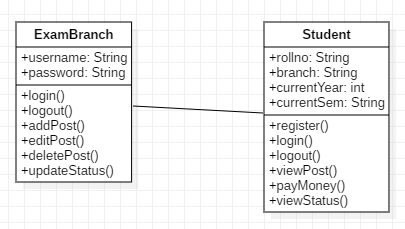
**Fig: Final State**



## CLASS DIAGRAM

An object is any person, place, thing, concept, event, screen, or report applicable to your system. Objects both know things (they have attributes) and they do things (they have methods).

A class is a representation of an object and, in many ways; it is simply a template from which objects are created. Classes form the main building blocks of an object-oriented application.



## Responsibilities

Classes are typically modeled as rectangles with three sections: the top section for the name of the class, the middle section for the attributes of the class, and the bottom section for the methods of the class. Attributes are the information stored about an object, while methods are the things an object or class do. For example, students have student numbers, names, addresses, and phone numbers. Those are all examples of the attributes of a student. Students also enroll in courses, drop courses, and request transcripts. Those are all examples of the things a student does, which get implemented (coded) as methods. You should think of methods as the object-oriented equivalent of functions and procedures.

**ER Diagram:**

Data models are tools used in analysis to describe the data requirements and assumptions in the system from a top-down perspective. They also set the stage for the design of databases later on in the SDLC.

There are three basic elements in ER models:

* Entities are the "things" about which we seek information.
* Attributes are the data we collect about the entities.
* Relationships provide the structure needed to draw information from multiple entities.

**Developing an ERD:**

Developing an ERD requires an understanding of the system and its components. Its components are:

* Rectangles**:** These are used forrepresenting entity sets.
* Ellipses**:** These are used for representing attributes.
* Diamonds**:** These are used for representing relationship sets.
* Lines**:** These are used for linking attributes to entity sets and entity sets to relationship sets.

**Start an ERD:**

* Define Entities**:**

These are usually nouns used in descriptions of the system, in the discussion of business rules, or in documentation.

* Define Relationships**:**

These are usually verbs used in descriptions of the system or in discussion of the business rules (entity \_ entity).

* Add attributes to the relations**:**

These are determined by the queries, and may also suggest new entities, e.g. grade; or they may suggest the need for keys or identifiers.

* Add cardinality to the relations**:**

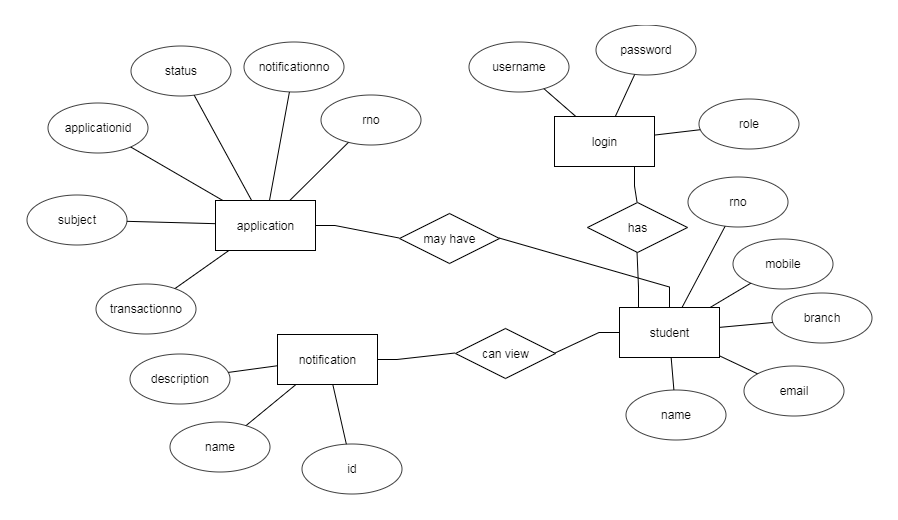
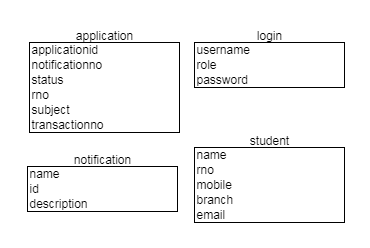
Many-to-Many must be resolved to two one-to-many with an additional entity. Automatically happens.

Sometimes involves introduction of a link entity (which will be all

Foreign key) this flexibility allows us to consider a variety of questions. Represent **t**hat information with symbols.

**ERD brings out issues:**

* Many-to-Many
* Ambiguities
* Entities and their relationships
* What data needs to be stored
* The Degree of a relationship

****

**SOFTWARE TESTING**

TESTING

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and code generation.

#### TESTING OBJECTIVES

* + To ensure that during operation the system will perform as per specification.
  + TO make sure that system meets the user requirements during operation
  + To make sure that during the operation, incorrect input, processing and output will be detected
  + To see that when correct inputs are fed to the system the outputs are correct
  + To verify that the controls incorporated in the same system as intended
  + Testing is a process of executing a program with the intent of finding an error
  + A good test case is one that has a high probability of finding an as yet undiscovered error

The software developed has been tested successfully using the following testing strategies and any errors that are encountered are corrected and again the part of the program or the procedure or function is put to testing until all the errors are removed. A successful test is one that uncovers an as yet undiscovered error.

Note that the result of the system testing will prove that the system is working correctly. It will give confidence to system designer, users of the system, prevent frustration during implementation process etc.,

## TEST CASE DESIGN:

## White box testing

White box testing is a testing case design method that uses the control structure of the procedure design to derive test cases. All independents path in a module are exercised at least once, all logical decisions are exercised at once, execute all loops at boundaries and within their operational bounds exercise internal data structure to ensure their validity. Here the customer is given three chances to enter a valid choice out of the given menu. After which the control exits the current menu.

## 

## Black Box Testing

Black Box Testing attempts to find errors in following areas or categories, incorrect or missing functions, interface error, errors in data structures, performance error and initialization and termination error. Here all the input data must match the data type to become a valid entry.

The following are the different tests at various levels:

**Unit Testing:**

Unit testing is essentially for the verification of the code produced during the coding phase and the goal is test the internal logic of the module/program. In the Generic code project, the unit testing is done during coding phase of data entry forms whether the functions are working properly or not. In this phase all the drivers are tested they are rightly connected or not.

**Integration Testing:**

All the tested modules are combined into sub systems, which are then tested. The goal is to see if the modules are properly integrated, and the emphasis being on the testing interfaces between the modules. In the generic code integration testing is done mainly on table creation module and insertion module.

## Validation Testing

This testing concentrates on confirming that the software is error-free in all respects. All the specified validations are verified and the software is subjected to hard-core testing. It also aims at determining the degree of deviation that exists in the software designed from the specification; they are listed out and are corrected.

## System Testing

This testing is a series of different tests whose primary is to fully exercise the computer-based system. This involves:

* Implementing the system in a simulated production environment and testing it.
* Introducing errors and testing for error handling.

**Test Cases:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tested | Test name | Inputs | Expected output | Actual output | status |
| 1 | Login test | Username, password | Need to validate on database | Its successfully validated on the database | success |
| 2 | Registration | Registration details | Need add the data in the table | Data added successfully | success |
| 3 | Add request | Request details | Need add the request details on database | Request details added | success |
| 5 | Updated request or request details | Status, id | Need to update the given status with given user id | Status successfully updated with given use rid | success |

**OUTPUT SCREENS**

Copy your screen shots here

**CONCLUSION**

Exam Registration System is an interface between the Student and the Exam Controller responsible for the Issue exam notification and student exam registration requests. It aims at improving the efficiency in the exam branch and reduces the complexities involved in it to the maximum possible extent.

Our Exam Registration System acts as an interface between the 'student' and the 'exam controller'. This system tries to make the interface as simple as possible and at the same time not risking the security of data stored in. This minimizes the time duration.

**Feature Enhancement:**

There is considerable scope to develop this project further. Integration with an accounting system to include revenues and expenses would be extremely beneficial to the Application manager. Furthermore, this system is only intended for use by the manager of one particular Application. It could be expanded to encompass all the Application operations.

**Project Review**

At the beginning of this project several minimum requirements were submitted. It seems rather obvious to state that the success of the project depends on the extent to which it met these minimum requirements.

The minimum requirements were specified These were:

* Examine the tools and methodologies required to gain an overview of the system requirements for the proposed database.
* Examine suitable database management systems that can be used to implement the proposed database.
* Evaluate appropriate website authoring and web graphic creation tools that can be used to develop web based forms for the proposed database
* Produce and apply suitable criteria for evaluating the solution

To meet these requirements research was conducting to find the various software development methodologies. After analyzing each technique, the chosen methodology was adapted to meet the needs of this project, and then followed as accurately as possible. Therefore, this project went through the appropriate development processes thus meeting this requirement

Considerable time was spent investigating the different database management systems currently available, financial and practical issues were taken into account and a suitable comprise was reached. However, this requirement was almost entirely governed by the systems.

The third requirement was to assess appropriate web authoring and web graphic creation tools. As the budget for this project was extremely limited this meant that the products were limited to freeware, shareware, or products that were already owned by the developer. The project looked at the different tools available to the developer and chose the one most appropriate for developing this type of system hence meeting this requirement.

Finally, to certify database and webpage integrity, evaluation techniques were appraised to establish the most appropriate procedure for this particular project. This evaluation criterion was then applied to the developed system.

The main aim of this project was to produce an appropriate solution which would meet the requirements of the user this project not only met all the functional requirements of the application, but exceeded them by implementing several of the non-essential requirements listed in section

**Bibliography**

**Textbooks Referred:**

1.Barry Burd, “Java Server Pages” , 1st Ed, IDG Books India(p) Ltd, pg 31-104, 237-

298, 305-355.

2.Richard Fairley, “Software Engineering Concepts”, 7th Ed., Tata McGraw Hill,

pg 282-303.

3. H.M.Deitel & P.J.Deitel, “Java How to program”, 6th Ed., PEARSON

Education, pg. 1346-1392.

4. Kevin Loney, George Koch, “ORACLE The Complete Reference”, Tata McGraw

Hill, pg. 41-69 & 165-190.

5.James Goodwill, “Developing Java Servlets”, SAMS Techmedia, pg.145-199, 253-

269 & 281-299.

6.Grady Booch, James Rambaugh, Ivar Jacobson, “The Unified Modelling

LanguageUser Guide”, 12th Ed, Pearson Education, pg. 219-229,233-239,205-215 &

243-255.

7.Raghu RamaKrishnan, Johannes Gehrke, “Database Management Systems”, 2nd Ed.

Mc Graw Hill, 24-45 & 119-150.

8.Roger S.Pressman,”Software Engineering-A practitioners Approach”, 5th Ed., Mc

Graw Hill, pg. 36-38,485-494.

9.Simon Roberts, Philip Heller, Michael Ernest, “Java 2 Certification Study”, BPB

Publications, pg. 509-536.

10.Korth F Henry, “Database System Concepts”, 4th Ed., Mc Graw Hill, pg. 27-62, 135-

168 & 225-238.

**Websites:**

Bibliography

<https://in.udacity.com/course/android-basics-nanodegree-by-google--nd803?utm_source=google&utm_medium=newacq&utm_campaign=PR-Search-D-Purchase-AndroidFoundation-Nanodegree-BMM&utm_term=%2Bandroid%20%2Bdevelopment&gclid=CjwKCAjwyrvaBRACEiwAcyuzRH0en5GcnaB5xvhuAga3qv0H047bXcf0z5-sogGbd5Y4Urlf5xyZpRoC3GEQAvD_BwE>

<https://developer.android.com/>

<https://www.edx.org/learn/android-development>

<https://www.youtube.com/watch?v=QAbQgLGKd3Y>