

PLACEMENT GURU

A PROJECT REPORT

Submitted By

VIDHI AGGARWAL (2000290140131)

VAIBHAV GUPTA (2000290140128)

RAJ VARDHAN CHAUHAN (2000290140095)

ANKIT KUMAR SINGH (2000290140019)

**Submitted in partial fulfillment of the
Requirements for the Degree of**

MASTER OF COMPUTER APPLICATIONS

**Under the Supervision of
MR. NARESH CHANDRA
ASSISTANT PROFESSOR**



Submitted to

**DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206**

(13 JAN 2022)

CERTIFICATE

Certified that **Vidhi Aggarwal (200029014005816)**, **Vaibhav Gupta (200029014005813)**, **RajVardhan Chauhan (200029014005780)**, **Ankit Thakur (200029014005704)** have carried out the project work having “**PLACEMENT GURU**” for Master of Computer Applications from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Technical University, Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself / herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Date: 2022-01-12

VIDHI AGGARWAL (2000290140131)

VAIBHAV GUPTA (2000290140128)

RAJVARDHAN CHAUHAN (2000290140095)

ANKIT KUMAR SINGH (2000290140019)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: 2022-01-12

PROF. NARESH CHANDRA

ASSISTANT PROFESSOR

Department of Computer Applications

KIET Group of Institutions, Ghaziabad

Signature of Internal Examiner

Signature of External Examiner

Dr. Ajay Shrivastava
Head, Department of Computer Applications
KIET Group of Institutions, Ghaziabad

ABSTRACT

The Placement Guru to be developed benefits greatly the members. The system provides job catalogue and information to members and helps them decide on the jobs to apply for. The Admin can keep the jobs catalogue updated all the time so that the members get the updated information all the time.

The Placement Guru that is to be developed provides the members with job information, online applying for jobs, and many other facilities. The basic scope of the project is given as under: Maintain Job Seeker and Employer records Maintain and uploaded resumes Provide customised job postings Maintain Job Posting details and generate various reports.

ACKNOWLEDGEMENTS

Success in life is never attained single handedly. My deepest gratitude goes to my thesis supervisor, **PROF. NARESH CHANDRA** for his guidance, help and encouragement throughout my research work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to Dr. Ajay Kumar Shrivastava, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help at various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

VIDHI AGGARWAL

VAIBHAV GUPTA

RAJVARDHAN CHUHAN

ANKIT KUMAR SINGH

List of Chapters

Chapter 1 - Introduction

- 1.1 Project description
- 1.2 Project Scope
- 1.3 Overall Description
- 1.4 Hardware / Software used in Project
 - 1.4.1 Hardware Specification
 - 1.4.2 Software Specification

Chapter 2 Feasibility Study

- 2.1 Technical feasibility
- 2.2 Operational Feasibility
- 2.3 Behavioral Feasibility

Chapter 3 Database Design

- 3.1 Database Tables
 - 3.1.1 Users Database
 - 3.1.2 Jobs Database
- 3.2 Flow Chart
- 3.3 Use Case Diagram
- 3.4 Sequence Diagram
- 3.5 Activity Diagram
- 3.6 State Diagram
- 3.7 Component Diagram

Chapter 4 Implementation Details

- 4.1 Frontend
 - 4.1.1 Java Script
 - 4.1.2 React
- 4.2 Backend
 - 4.2.1 Express JS
 - 4.2.2 Node JS
- 4.3 Database
 - 4.3.1 MongoDB

Chapter 5 Testing

5.1 Unit Testing

5.2 Integration Testing

5.3 Software Verification And Validation

5.4 Black Box Testing

5.5 White Box Testing

5.6 System Testing

5.7 Test Cases

Chapter 6 Form Design

6.1 Registration Module

6.2 Login Module

6.3 Home Module

6.4 Filter Module

6.5 Profile Module

6.5.1 Personal Info Module

6.5.2 Skills & Education Module

6.1 Applied Jobs Module

6.7 Post Job Module

6.8 Posted Job Module

Chapter 7 Advantages

Chapter 8 Conclusion

Chapter 9 Bibliography

Chapter-1 INTRODUCTION

1.1 PROJECT DESCRIPTION

Visiting company web sites and applying for individual jobs are less motivated and a lot of hard work. Knowing about a company, knowing what kind of qualifications and requirements they want for a position is always so much time taking. I have always felt the need of friendly applications that gives me all these details in one place and saves a lot of my time. During our undergraduate and some graduate years, the only way we have looked up for jobs is through on campus placements, company websites or employee referrals or through a lot of networking with company personnel. But with the fast rate of technical advancement we have come across many online applications that makes finding a suitable job according to our qualifications much easier, knowing about different positions opened in our desired companies, the qualifications or requirements that the job position needs and search features to retrieve my desired information all bonded in one place. This motivated us to develop an online job search portal as we realised their value as a student and their importance too, as they save a lot of time and effort. Apart from this we were motivated to build this application to learn the usage of some cutting-edge technologies and gain some hands-on experience. We have used NodeJS, MongoDB as database, React, Express and have gained enough experience and exposure working on them. The purpose of designing the "Placement Guru" is to give the job seekers a platform for finding a right and a satisfactory job according to their qualification and skillset. It aims to provide all the placement related help to all job seekers as well as to the working professional to switch their jobs. This is basically a job portal where job seekers will get updates on off campus drives and various other job listings and can apply for jobs. Also, employer of a particular company can share job openings of their organisations and provide referral to applicants. The website will also provide some standard resume templates which will help students in resume shortlisting in their respective companies. It will also provide interview experiences, and all other placements related queries to the students through regular Blog postings.

1.2 PROJECT SCOPE

The Software Requirements Specification captures all the requirements in a single document. The Placement Guru that is to be developed provides the members with job information, online applying for jobs, and many other facilities. The basic scope of the project is given as under: Maintain Job Seeker and Employer records Maintain uploaded resumes Provide customised job postings Maintain Job Posting details and generate various reports.

1.3 OVERALL DESCRIPTION

The Placement Guru to be developed benefits greatly the members. The system provides job catalogue and information to members and helps them decide on the jobs to apply for. The Admin can keep the jobs catalogue updated all the time so that the members get the updated information all the time.

1.4 HARDWARE/SOFTWARE USED IN PROJECT

1.4.1 Hardware Specification

- HDD 20 GB Hard Disk space and above
- 4 GB RAM
- 2.8 GHz Processor and above

1.4.2 Software Specification

- Operating System Windows
- IDE:- Visual Studio Code
- Database:- MongoDB
- Backend:- Express JS, Node JS
- Frontend:- React

Chapter-2 FESIABILITY STUDY

A feasibility study is a high-level capsule version of the entire System analysis and Design Process. The study begins by classifying the problem definition. Feasibility is to determine if it's worth doing. Once an acceptance problem definition has been generated, the analyst develops a logical model of the system. A search for alternatives is analyzed carefully. There are 3 parts in feasibility study.

2.1 TECHNICAL FESIABILITY

This involves questions such as whether the technology needed for the system exists, how difficult it will be to build, and whether the firm has enough experience using that technology. The assessment is based on outline design of system requirements in terms of input, processes, output, fields, programs and procedures. This can be qualified in terms of volume of data, trends, frequency of updating in order to give an introduction to the technical system. The application is the fact that it has been developed on windows XP platform and a high configuration of 1GB RAM on Intel Pentium Dual core processor. This is technically feasible. The technical feasibility assessment is focused on gaining an understanding of the present technical resources of the organization and their applicability to the expected needs of the proposed system. It is an evaluation of the hardware and software and how it meets the need of the proposed system.

2.2 OPERATIONAL FESIABILITY

Operational feasibility is the measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development. The operational feasibility assessment focuses on the degree to which the proposed development projects fits in with the existing business environment and objectives with regard to development schedule, delivery date, corporate culture and existing business processes. To ensure success, desired operational outcomes must be imparted during design and development. These include such design-dependent parameters as reliability, maintainability, supportability, usability, producibility, disposability, sustainability, affordability and others. These parameters are required to be considered at the early stages of design if desired operational behaviours are to be realised. A system design and development requires appropriate and timely application of engineering and management efforts to meet the previously mentioned parameters.

A system may serve its intended purpose most effectively when its technical and operating characteristics are engineered into the design. Therefore, operational feasibility is a critical aspect of systems engineering that needs to be an integral part of the early design phases.

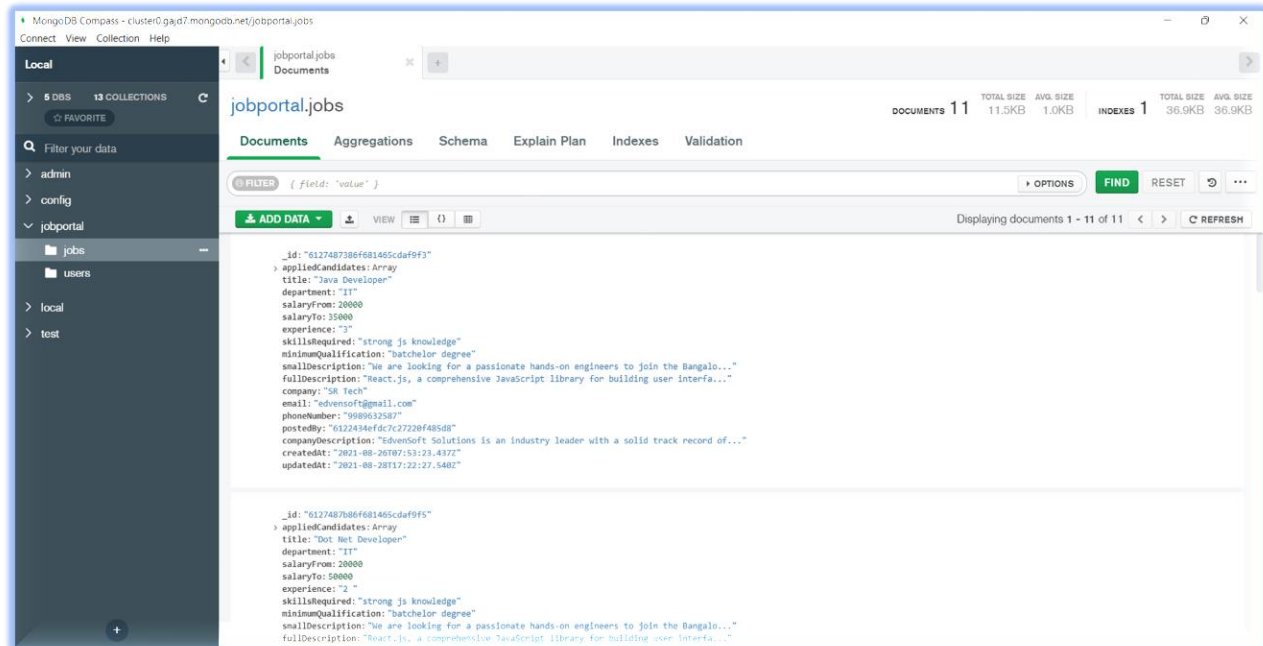
2.3 BEHAVIORAL FESIABILITY

Establishing the cost-effectiveness of the proposed system i.e. if the benefits do not outweigh the costs then it is not worth going ahead. In the fast paced world today there is a great need of online social networking facilities. Thus the benefits of this project in the current scenario make it economically feasible. The purpose of the economic feasibility assessment is to determine the positive economic benefits to the organization that the proposed system will provide. It includes quantification and identification of all the benefits expected. This assessment typically involves a cost/benefits analysis.

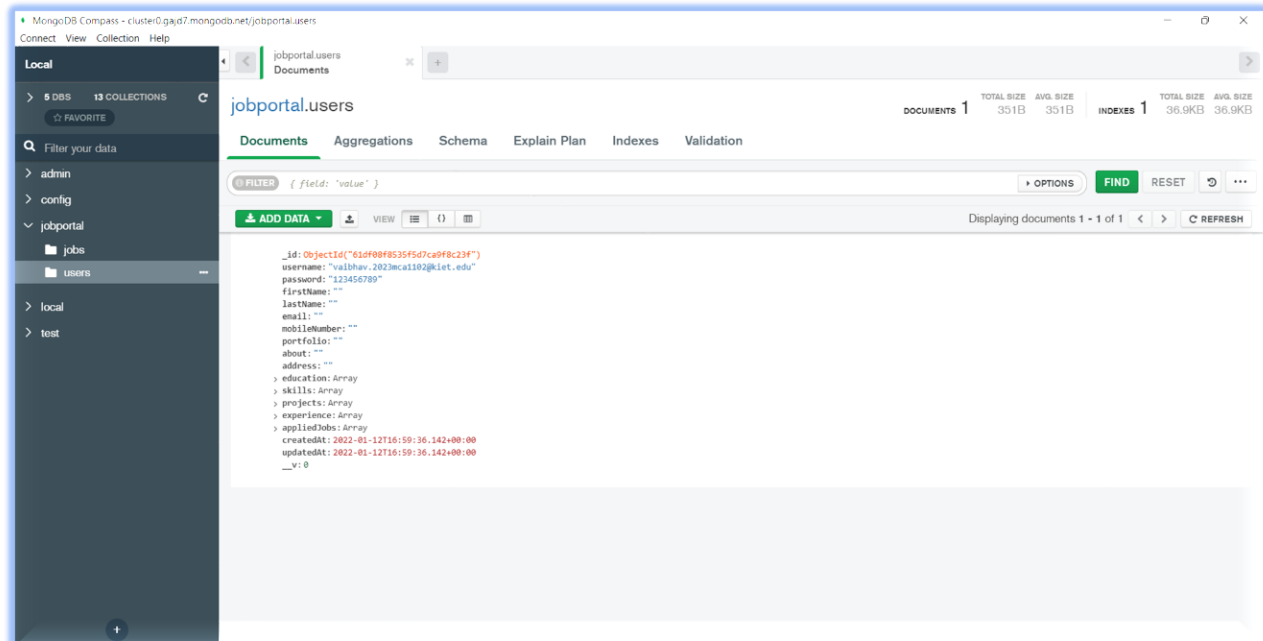
Chapter -3 DATABASE DESIGN

3.1 DATABASE TABLES

3.1.1 Users Database



3.1.2 Jobs Database



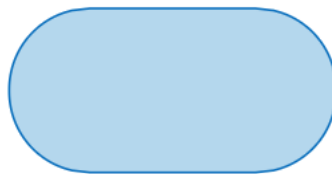
3.2 FLOW CHART

Flowchart is a graphical representation of an algorithm. Programmers often use it as a program-planning tool to solve a problem. It makes use of symbols which are connected among them to indicate the flow of information and processing.

The process of drawing a flowchart for an algorithm is known as “flowcharting”.

Basic Symbols used in Flowchart Designs

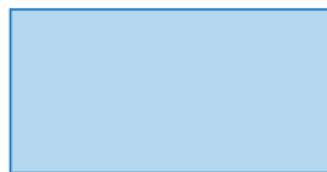
1. **Terminal:** The oval symbol indicates Start, Stop and Halt in a program’s logic flow. A pause/halt is generally used in a program logic under some error conditions. Terminal is the first and last symbols in the flowchart.



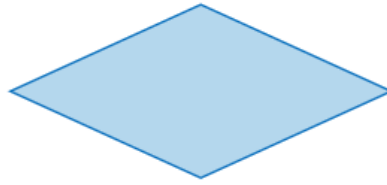
- **Input/Output:** A parallelogram denotes any function of input/output type. Program instructions that take input from input devices and display output on output devices are indicated with parallelogram in a flowchart.



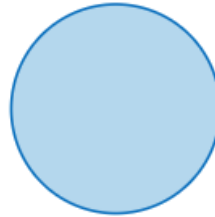
- **Processing:** A box represents arithmetic instructions. All arithmetic processes such as adding, subtracting, multiplication and division are indicated by action or process symbol.



- **Decision** Diamond symbol represents a decision point. Decision based operations such as yes/no question or true/false are indicated by diamond in flowchart.

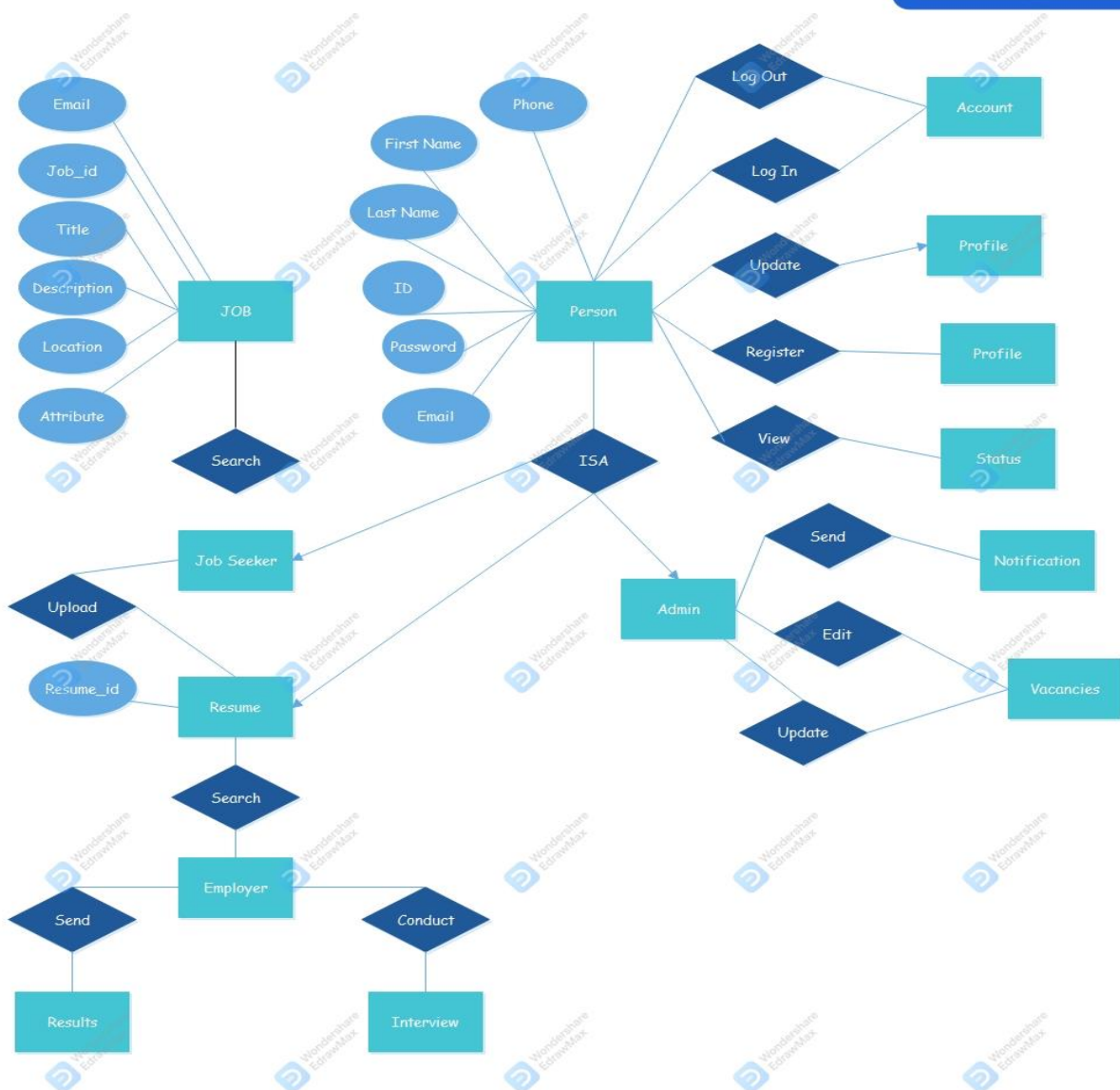


- **Connectors:** Whenever flowchart becomes complex or it spreads over more than one page, it is useful to use connectors to avoid any confusions. It is represented by a circle.



- **Flow lines:** Flow lines indicate the exact sequence in which instructions are executed. Arrows represent the direction of flow of control and relationship among different symbols of flowchart.





3.3 USE CASE DIAGRAM

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

- Scenarios in which your system or application interacts with people, organizations, or external systems
- Goals that your system or application helps those entities (known as actors) achieve

Use case diagram components

To answer the question, "What is a use case diagram?" you need to first understand its building blocks. Common components include:

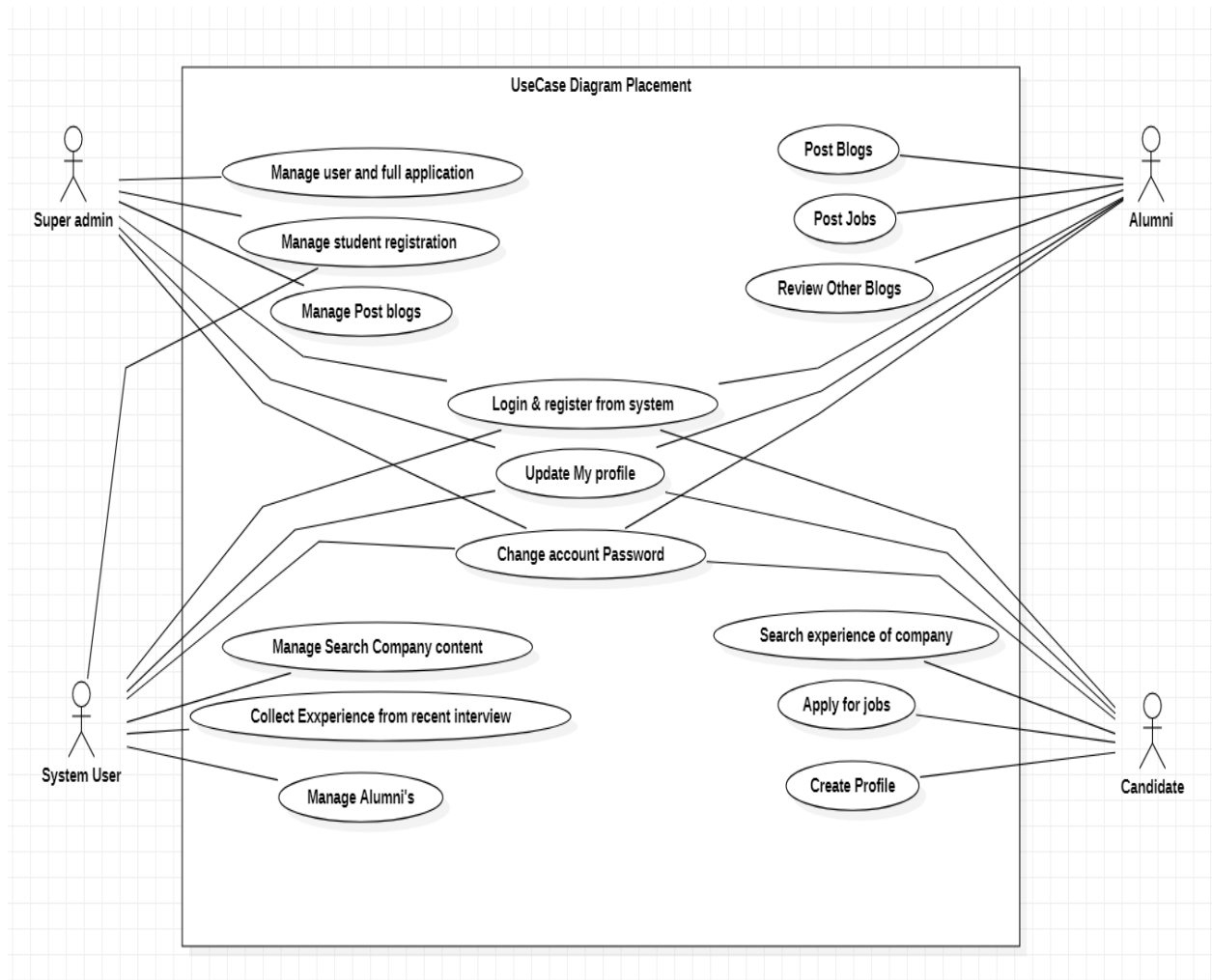
- **Actors:** The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data.
- **System:** A specific sequence of actions and interactions between actors and the system. A system may also be referred to as a scenario.
- **Goals:** The end result of most use cases. A successful diagram should describe the activities and variants used to reach the goal.

Use case diagram symbols and notation

The notation for a use case diagram is pretty straightforward and doesn't involve as many types of symbols as other UML diagrams.

Use cases: Horizontally shaped ovals that represent the different uses that a user might have.

- **Actors:** Stick figures that represent the people actually employing the use cases.
- **Associations:** A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.
- **System boundary boxes:** A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system. For example, Psycho Killer is outside the scope of occupations in the chainsaw example found below.
- **Packages:** A UML shape that allows you to put different elements into groups. Just as with component diagrams, these groupings are represented as file folders.



3.4 SEQUENCE DIAGRAM

To understand what a sequence diagram is, it's important to know the role of the [Unified Modeling Language](#), better known as UML. UML is a modeling toolkit that guides the creation and notation of many types of diagrams, including behavior diagrams, interaction diagrams, and structure diagrams.

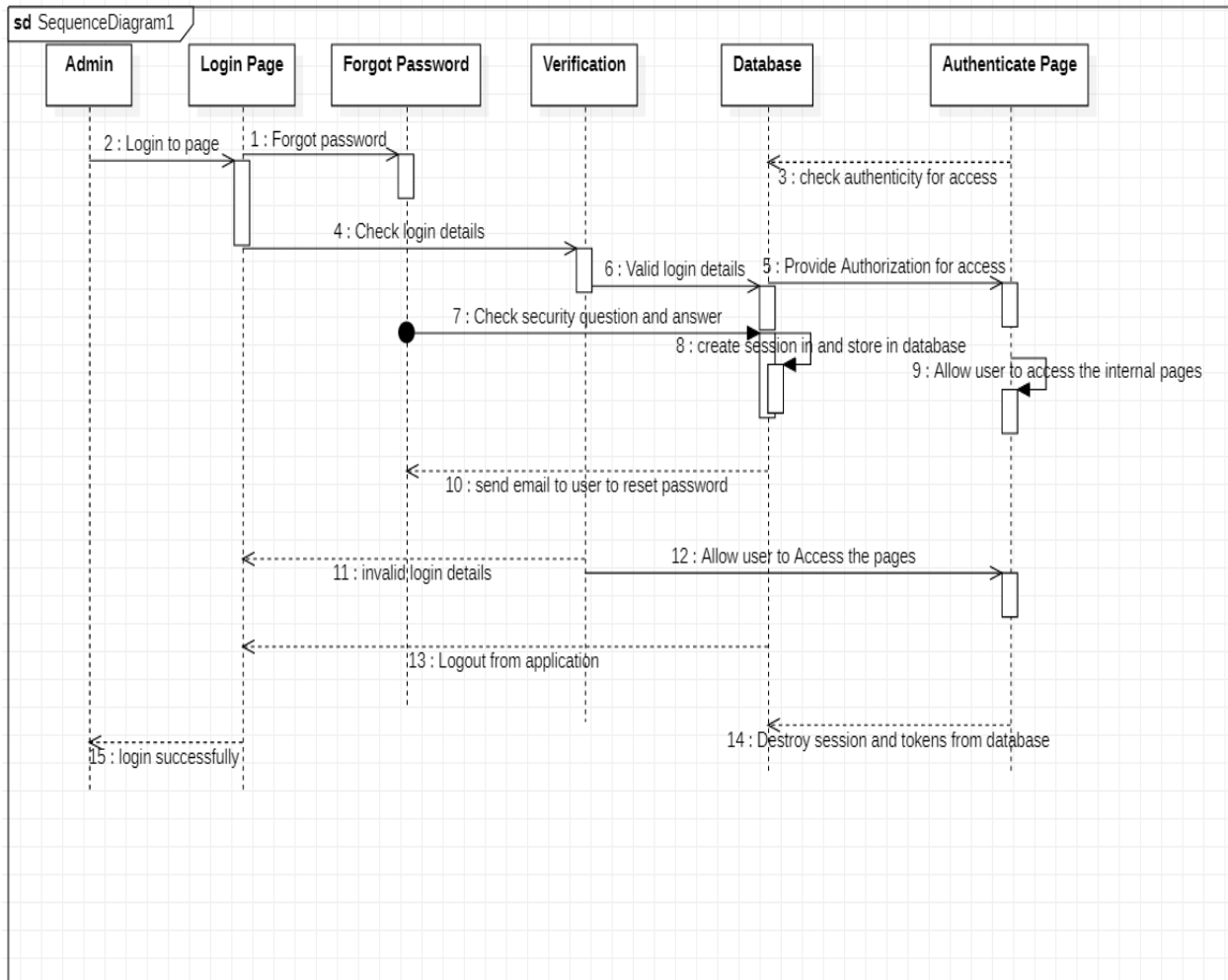
A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios.

Note that there are two types of sequence diagrams: UML diagrams and code-based diagrams. The latter is sourced from programming code and will not be covered in this guide.

Benefits of sequence diagrams

Sequence diagrams can be useful references for businesses and other organizations. Try drawing a sequence diagram to:

- Represent the details of a UML use case.
- Model the logic of a sophisticated procedure, function, or operation.
- See how objects and components interact with each other to complete a process.
- Plan and understand the detailed functionality of an existing or future scenario.



3.5 ACTIVITY DIAGRAM

We use **Activity Diagrams** to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram. An activity diagram focuses on condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram.

UML models basically three types of diagrams, namely, structure diagrams, interaction diagrams, and behavior diagrams. An activity diagram is a **behavioral diagram** i.e. it depicts the behavior of a system.

An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed. We can depict both sequential processing and concurrent processing of activities using an activity diagram. They are used in business and process modelling where their primary use is to depict the dynamic aspects of a system.

Activity Diagram Notations –

1. **Initial State** – The starting state before an activity takes place is depicted using the initial state.



Figure – notation for initial state or start state

A process can have only one initial state unless we are depicting nested activities. We use a black filled circle to depict the initial state of a system. For objects, this is the state when they are instantiated. The Initial State from the UML Activity Diagram marks the entry point and the initial Activity State.

For example – Here the initial state is the state of the system before the application is opened.

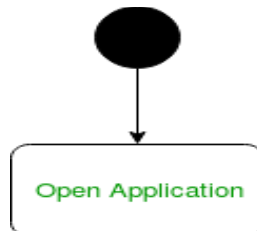


Figure – initial state symbol being used

2. **Action or Activity State** – An activity represents execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners. Basically any action or event that takes place is represented using an activity.



Figure – notation for an activity state

For example – Consider the previous example of opening an application opening the application is an activity state in the activity diagram.

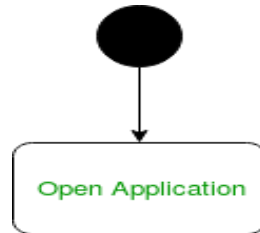


Figure – activity state symbol being used

3. **Action Flow or Control flows** – Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another.



Figure – notation for control Flow

An activity state can have multiple incoming and outgoing action flows. We use a line with an arrow head to depict a Control Flow. If there is a constraint to be adhered to while making the transition it is mentioned on the arrow.

Consider the example – Here both the states transit into one final state using action flow symbols i.e. arrows.

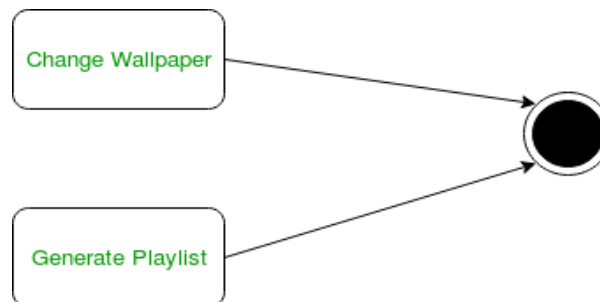


Figure – using action flows for transitions

4. **Decision node and Branching** – When we need to make a decision before deciding the flow of control, we use the decision node.

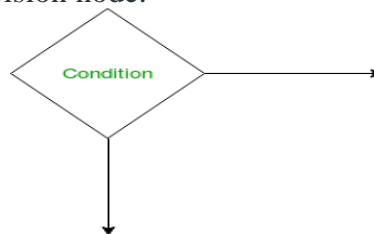


Figure – notation for decision node

The outgoing arrows from the decision node can be labelled with conditions or guard expressions. It always includes two or more output arrows.

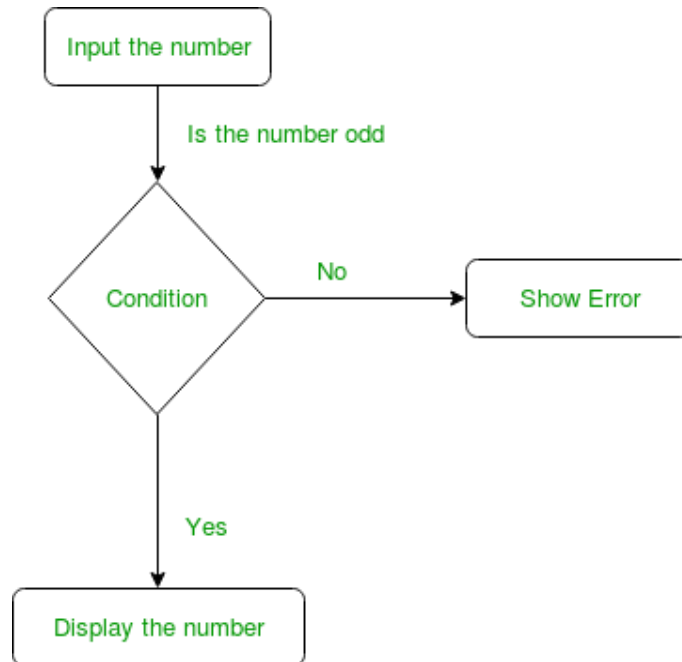


Figure – an activity diagram using decision node

5. **Guards** – A Guard refers to a statement written next to a decision node on an arrow sometimes within square brackets.

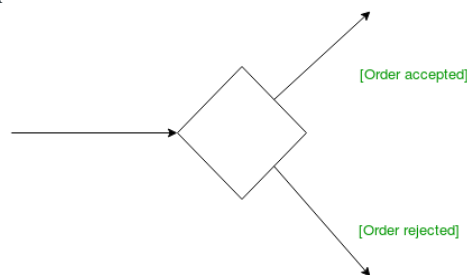


Figure – guards being used next to a decision node

The statement must be true for the control to shift along a particular direction. Guards help us know the constraints and conditions which determine the flow of a process.

6. **Fork** – Fork nodes are used to support concurrent activities.

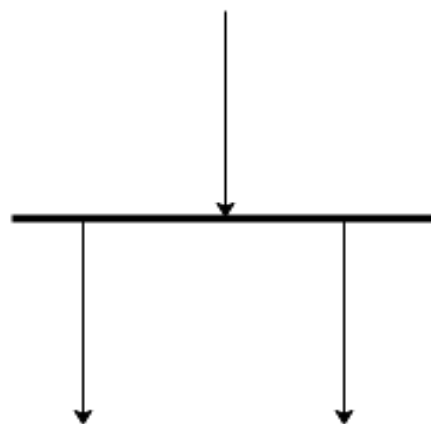


Figure – fork notation

When we use a fork node when both the activities get executed concurrently i.e. no decision is made before splitting the activity into two parts. Both parts need to be executed in case of a fork statement.

We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent activity state and outgoing arrows towards the newly created activities. For example: In the example below, the activity of making coffee can be split into two concurrent activities and hence we use the fork notation.

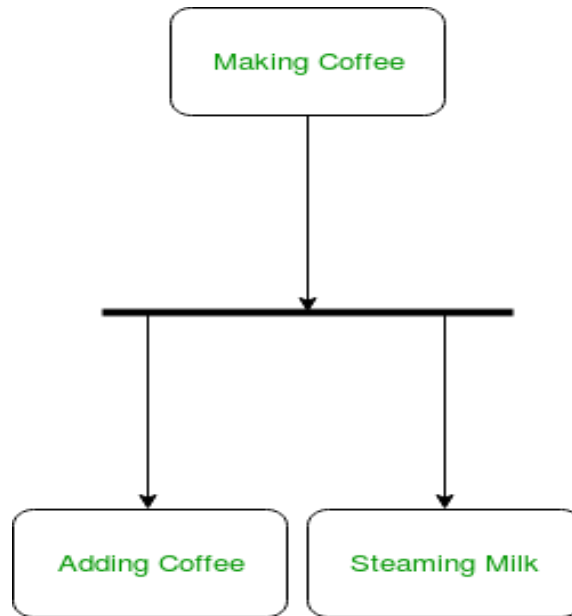


Figure – a diagram using fork

7. **Join** – Join nodes are used to support concurrent activities converging into one. For join notations we have two or more incoming edges and one outgoing edge.

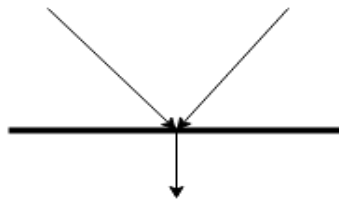


Figure – join notation

For example – When both activities i.e. steaming the milk and adding coffee get completed, we converge them into one final activity.

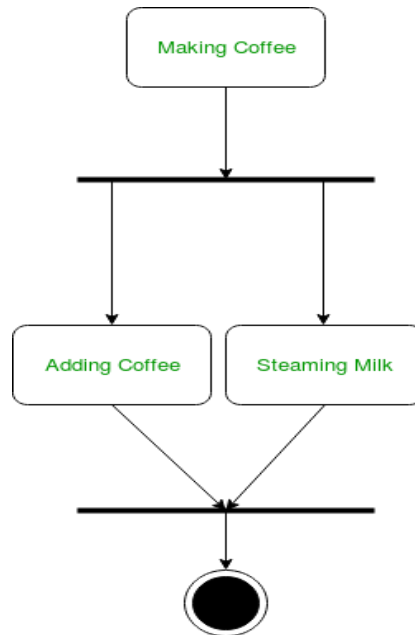


Figure – a diagram using join notation

8. **Merge or Merge Event** – Scenarios arise when activities which are not being executed concurrently have to be merged. We use the merge notation for such scenarios. We can merge two or more activities into one if the control proceeds onto the next activity irrespective of the path chosen.

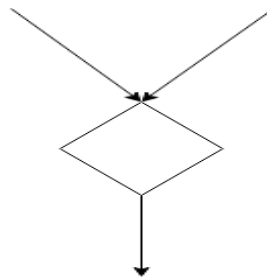


Figure – merge notation

For example – In the diagram below: we can't have both sides executing concurrently, but they finally merge into one. A number can't be both odd and even at the same time.

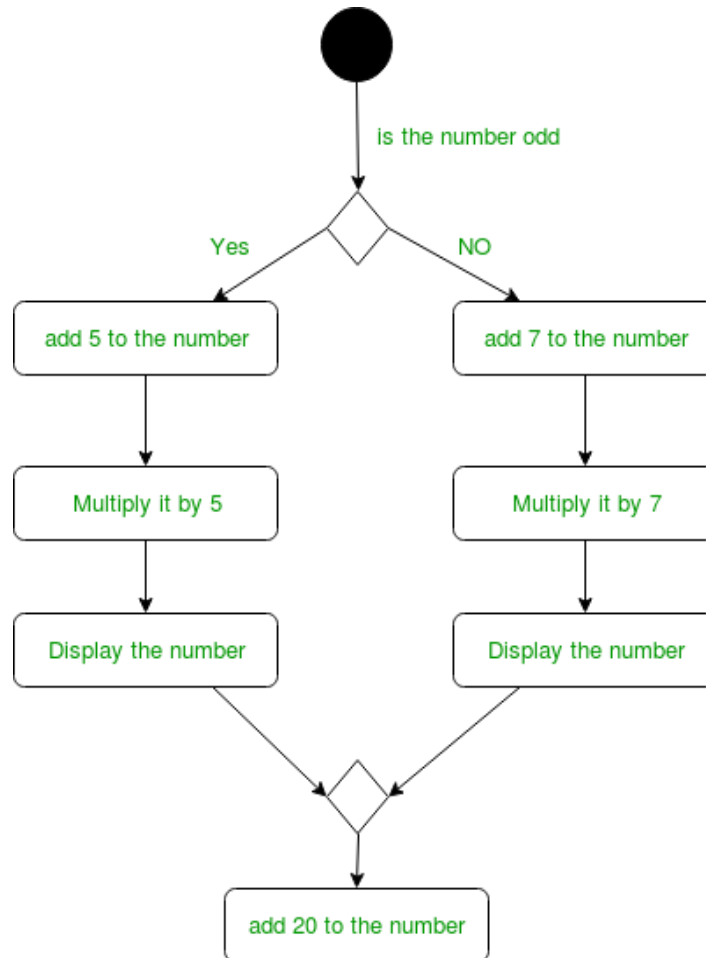


Figure – an activity diagram using merge notation

9. **Swimlanes** – We use swimlanes for grouping related activities in one column. Swimlanes group related activities into one column or one row. Swimlanes can be vertical and horizontal. Swimlanes are used to add modularity to the activity diagram. It is not mandatory to use swimlanes. They usually give more clarity to the activity diagram. It's similar to creating a function in a program. It's not mandatory to do so, but, it is a recommended practice.

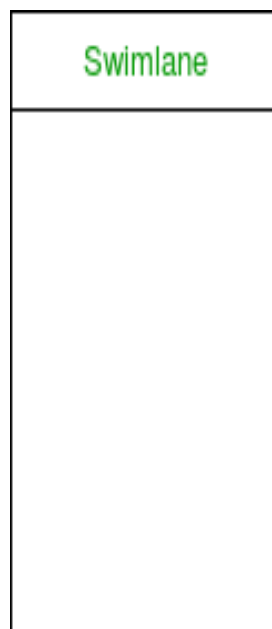


Figure – swimlanes notation

We use a rectangular column to represent a swimlane as shown in the figure above.

For example – Here different set of activities are executed based on if the number is odd or even. These activities are grouped into a swimlane.

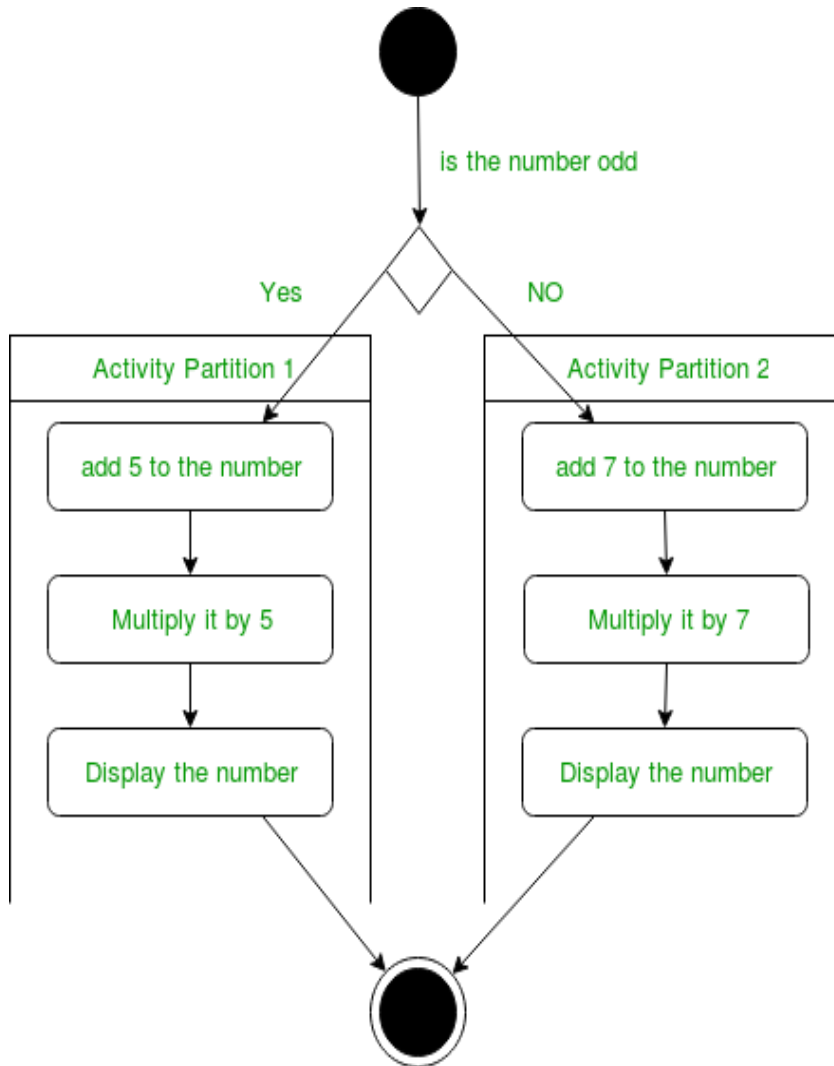


Figure – an activity diagram making use of swimlanes

10. Time Event –



Time Event

Figure – time event notation

We can have a scenario where an event takes some time to complete. We use an hourglass to represent a time event.

For example – Let us assume that the processing of an image takes a lot of time. Then it can be represented as shown below.

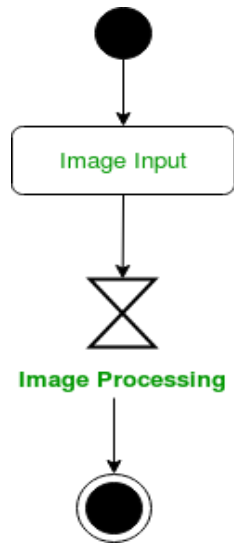
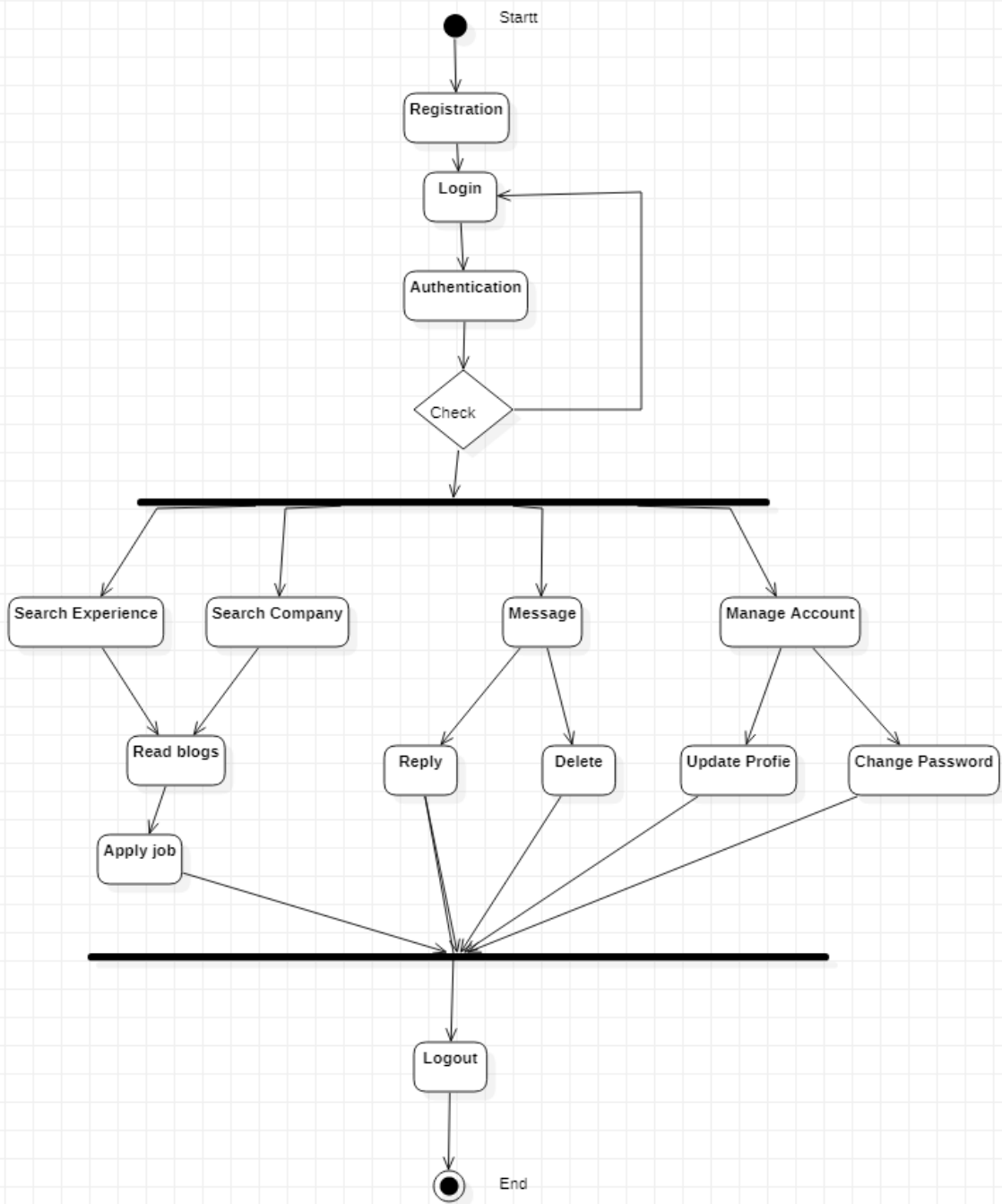


Figure – an activity diagram using time event

11. **Final State or End State** – The state which the system reaches when a particular process or activity ends is known as a Final State or End State. We use a filled circle within a circle notation to represent the final state in a state machine diagram. A system or a process can have multiple final states.



Figure – notation for final state

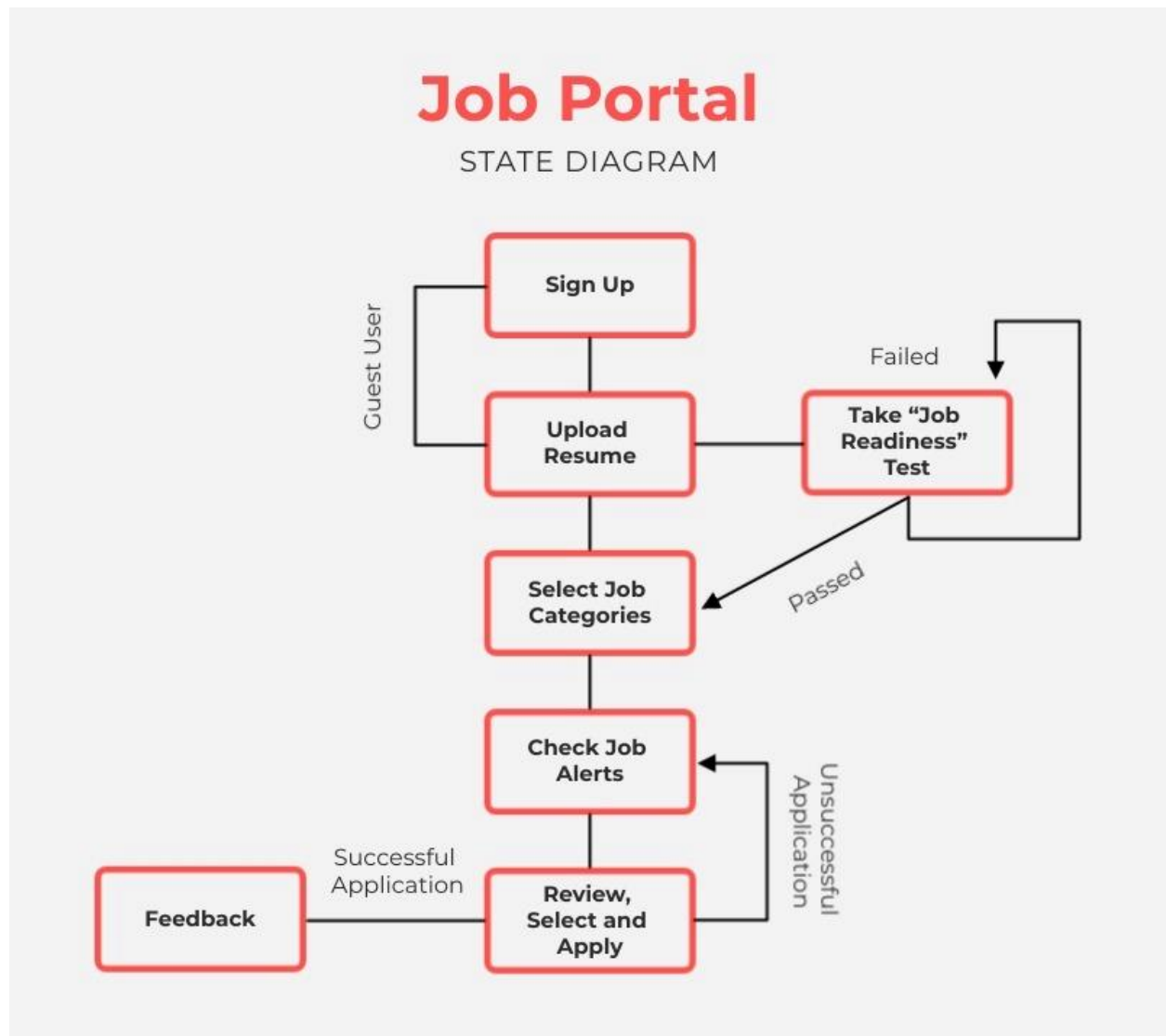


3.6 STATE DIAGRAM

A **state diagram** is used to represent the condition of the system or part of the system at finite instances of time. It's a **behavioral** diagram and it represents the behavior using finite state transitions. State diagrams are also referred to as **State machines** and **State-chart Diagrams**. These terms are often used interchangeably. So simply, a state diagram is used to model the dynamic behavior of a class in response to time and changing external stimuli. We can say that each and every class has a state but we don't model every class using State diagrams. We prefer to model the states with three or more states.

Uses of statechart diagram –

- We use it to state the events responsible for change in state (we do not show what processes cause those events).
- We use it to model the dynamic behavior of the system .
- To understand the reaction of objects/classes to internal or external stimuli.



3.7 COMPONENT DIAGRAM

The purpose of a component diagram is to show the relationship between different components in a system. For the purpose of UML 2.0, the term "component" refers to a module of classes that represent independent systems or subsystems with the ability to interface with the rest of the system.

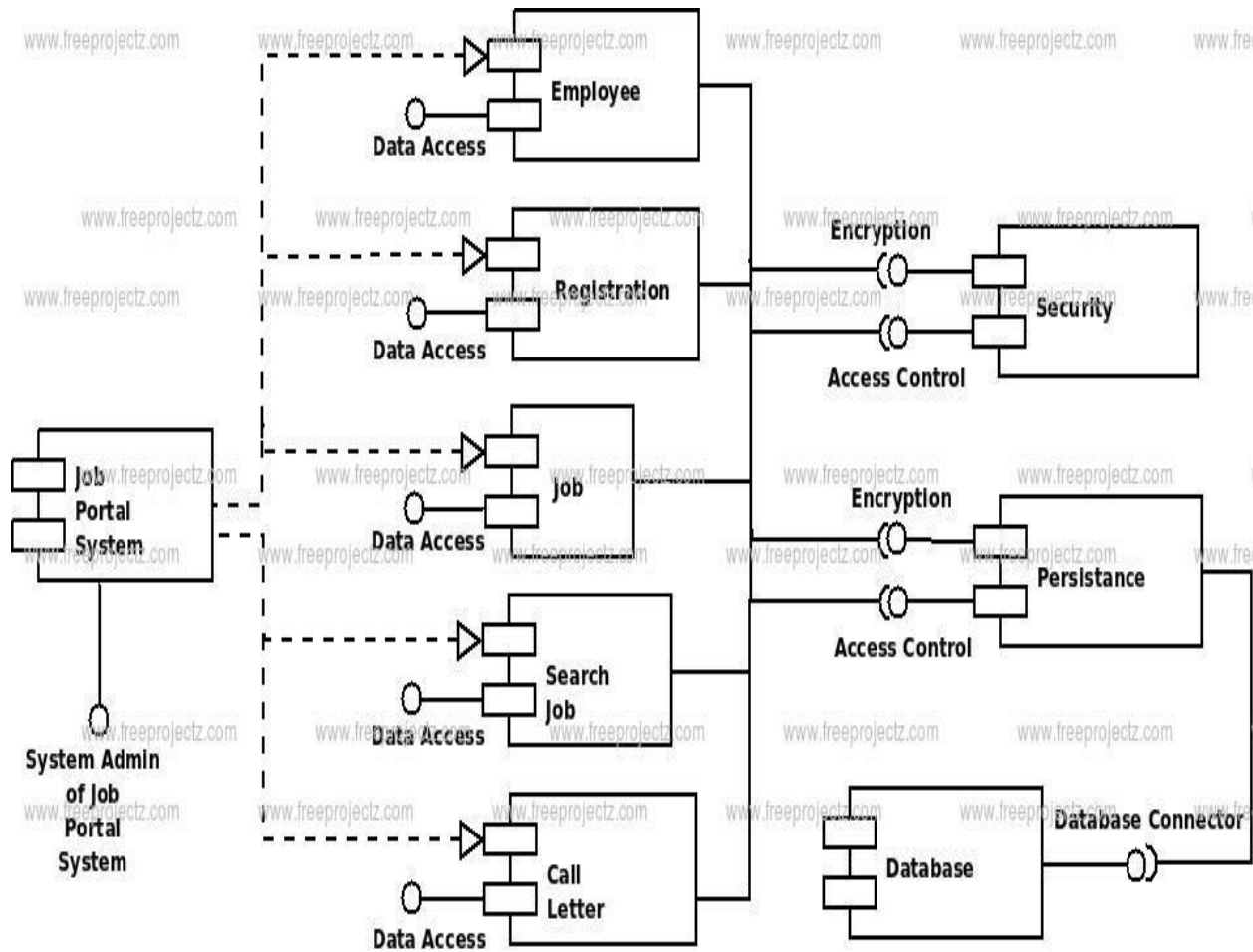
There exists a whole development approach that revolves around components: component-based development (CBD). In this approach, component diagrams allow the planner to identify the different components so the whole system does what it's supposed to do.

More commonly, in an OO programming approach, the component diagram allows a senior developer to group classes together based on common purpose so that the developer and others can look at a software development project at a high level.

Benefits of component diagrams

Though component diagrams may seem complex at first glance, they are invaluable when it comes to building your system. Component diagrams can help your team:

- Imagine the system's physical structure.
- Pay attention to the system's components and how they relate.
- Emphasize the service behavior as it relates to the interface.



Component Diagram of Job Portal System

Chapter-4 IMPLEMENTATION DETAILS

4.1 FRONTEND

4.1.1 Java Script

JavaScript is a high-level, interpreted scripting language that conforms to the ECMAScript specification. JavaScript has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions. Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. JavaScript enables interactive web pages and is an essential part of web applications. The vast majority of websites use it, and major web browsers have a dedicated JavaScript engine to execute it. As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative (including object-oriented and prototype-based) programming styles. It has APIs for working with text, arrays, dates, regular expressions, and the DOM, but the language itself does not include any I/O, such as networking, storage, or graphics facilities. It relies upon the host environment in which it is embedded to provide these features.

Initially only implemented client-side in web browsers, JavaScript engines are now embedded in many other types of host software, including server-side in web servers and databases, and in non-web programs such as word processors and PDF software, and in runtime environments that make JavaScript available for writing mobile and desktop applications, including desktop widgets.

The terms Vanilla JavaScript and Vanilla JS refer to JavaScript not extended by any frameworks or additional libraries. Scripts written in Vanilla JS are plain JavaScript code. Google's Chrome extensions, Opera's extensions, Apple's Safari 5 extensions, Apple's Dashboard Widgets, Microsoft's Gadgets, Yahoo! Widgets, Google Desktop Gadgets, and Serence Klipfolio are implemented using JavaScript.

4.1.2 React JS

ReactJS is a **declarative, efficient**, and flexible **JavaScript library** for building reusable UI components. It is an open-source, component-based front end library which is responsible only for the view layer of the application. It was initially developed and maintained by Facebook and later used in its products like WhatsApp & Instagram.

Our ReactJS tutorial includes all the topics which help to learn ReactJS. These are ReactJS Introduction, ReactJS Features, ReactJS Installation, Pros and Cons of ReactJS, ReactJS JSX, ReactJS Components, ReactJS State, ReactJS Props, ReactJS Forms, ReactJS Events, ReactJS Animation and many more.

Why we use ReactJS?

The main objective of ReactJS is to develop User Interfaces (UI) that improves the speed of the apps. It uses virtual DOM (JavaScript object), which improves the performance of the app. The JavaScript virtual DOM is faster than the regular DOM. We can use ReactJS on the client and server-side as well as with other frameworks. It uses component and data patterns that improve readability and helps to maintain larger apps.

4.2 BACKEND

4.2.1 Express JS

Express is a fast, assertive, essential and moderate web framework of Node.js. You can assume express as a layer built on the top of the Node.js that helps manage a server and routes. It provides a robust set of features to develop web and mobile applications.

Express is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications. It facilitates the rapid development of Node based Web applications. Following are some of the core features of Express framework –

- Allows to set up middlewares to respond to HTTP Requests.
- Defines a routing table which is used to perform different actions based on HTTP Method and URL.
- Allows to dynamically render HTML Pages based on passing arguments to templates.

Let's see some of the core features of Express framework:

- It can be used to design single-page, multi-page and hybrid web applications.
- It allows to setup middlewares to respond to HTTP Requests.
- It defines a routing table which is used to perform different actions based on HTTP method and URL.
- It allows to dynamically render HTML Pages based on passing arguments to templates.

Why use Express

- Ultra fast I/O
- Asynchronous and single threaded
- MVC like structure
- Robust API makes routing easy

4.2.2 Node JS

As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications. In the following "hello world" example, many connections can be handled concurrently. Upon each connection, the callback is fired, but if there is no work to be done, Node.js will sleep.

This is in contrast to today's more common concurrency model, in which OS threads are employed. Thread-based networking is relatively inefficient and very difficult to use. Furthermore, users of Node.js are free from worries of dead-locking the process, since there are no locks. Almost no function in Node.js directly performs I/O, so the process never blocks except when the I/O is performed using synchronous methods of Node.js standard library. Because nothing blocks, scalable systems are very reasonable to develop in Node.js.

If some of this language is unfamiliar, there is a full article on [Blocking vs. Non-Blocking](#).

Node.js is similar in design to, and influenced by, systems like Ruby's [Event Machine](#) and Python's [Twisted](#). Node.js takes the event model a bit further. It presents an [event loop](#) as a runtime construct instead of as a library. In other systems, there is always a blocking call to start the event-loop. Typically, behavior is defined through callbacks at the beginning of a script, and at the end a server is started through a blocking call like `EventMachine::run()`. In Node.js, there is no such start-the-event-loop call. Node.js simply enters the event loop after executing the input script. Node.js exits the event loop when there are no more callbacks to perform. This behavior is like browser JavaScript — the event loop is hidden from the user.

HTTP is a first-class citizen in Node.js, designed with streaming and low latency in mind. This makes Node.js well suited for the foundation of a web library or framework.

Node.js being designed without threads doesn't mean you can't take advantage of multiple cores in your environment. Child processes can be spawned by using our [child_process.fork\(\)](#) API, and are designed to be easy to communicate with. Built upon that same interface is the [cluster](#) module, which allows you to share sockets between processes to enable load balancing over your cores.

4.3 Database

4.3.1 MongoDB

MongoDB is an open-source document-oriented database that is designed to store a large scale of data and also allows you to work with that data very efficiently. It is categorized under the NoSQL (Not only SQL) database because the storage and retrieval of data in the MongoDB are not in the form of tables.

The MongoDB database is developed and managed by MongoDB, Inc. under SSPL (Server Side Public License) and initially released in February 2009. It also provides official driver support for all the popular languages like C, C++, C#, and .Net, Go, Java, Node.js, Perl, PHP, Python, Ruby, Scala, Swift, Mongoid. So, that you can create an application using any of these languages. Nowadays there are so many companies that use MongoDB like Facebook, Nokia, eBay, Adobe, Google, etc. to store their large amount of data.

How it works ?

Now, we will see how actually thing happens behind the scene. As we know that MongoDB is a database server and the data is stored in these databases. Or in other words, MongoDB environment gives you a server that you can start and then create multiple databases on it using MongoDB.

Because of its NoSQL database, the data is stored in the collections and documents. Hence the database, collection, and documents are related to each other as shown below:

- The MongoDB database contains collections just like the MySQL database contains tables. You are allowed to create multiple databases and multiple collections.
- Now inside of the collection we have documents. These documents contain the data we want to store in the MongoDB database and a single collection can contain multiple documents and you are schema-less means it is not necessary that one document is similar to another.
- The documents are created using the fields. Fields are key-value pairs in the documents, it is just like columns in the relation database. The value of the fields can be of any BSON data types like double, string, boolean, etc.
- The data stored in the MongoDB is in the format of BSON documents. Here, BSON stands for Binary representation of JSON documents. Or in other words, in the backend, the MongoDB server converts the JSON data into a binary form that is known as BSON and this BSON is stored and queried more efficiently.
- In MongoDB documents, you are allowed to store nested data. This nesting of data allows you to create complex relations between data and store them in the same document which makes the working and fetching of data extremely efficient as compared to SQL. In SQL, you need to write complex joins to get the data from table 1 and table 2. The maximum size of the BSON document is 16MB.

NOTE: In MongoDB server, you are allowed to run multiple databases.

Chapter-5 TESTING

5.1 Unit Testing

In computer programming, unit testing is a software testing 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 whether 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 it 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 short code fragments created by programmers or occasionally by white box testers during the development process. It forms the basis for component testing. 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.

5.1.2 Benefits

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits.

1) Find problems early : Unit testing finds problems early in the development cycle. In test-driven development (TDD), which is frequently used in both extreme programming and scrum, unit tests are created before the code itself is written. When the tests pass, that code is considered complete. The same unit tests are run against that function frequently as the larger code base is developed either as the code is changed or via an automated process with the build. If the unit tests fail, it is considered to be a bug either in the changed code or the tests themselves. The unit tests then allow the location of the fault or failure to be easily traced. Since the unit tests alert the development team of the problem before handing the code off to testers or clients, it is still early in the development process.

2) Facilitates Change : Unit testing allows the programmer to refactor code or upgrade system libraries at a later date, and make sure the module still works correctly (e.g., in regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified. Unit tests detect changes which may break a design contract.

3) Simplifies Integration : Unit testing may reduce uncertainty in the units themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier.

4) Documentation : Unit testing provides a sort of living documentation of the system. Developers looking to learn what functionality is provided by a unit, and how to use it, can look at the unit tests to gain a basic understanding of the unit's interface (API). Unit test cases embody characteristics that are critical to the success of the unit. These characteristics can indicate appropriate/inappropriate use of a unit as well as negative behaviors that are to be trapped by the unit. A unit test case, in and of itself, documents these critical characteristics, although many software development environments do not rely solely upon code to document the product in development.

5.2 : 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.

5.2.1 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. Software integration testing is performed according to the software development life cycle (SDLC) after module and functional tests. The cross-dependencies for software integration testing are: schedule for integration testing, strategy and selection of the tools used for integration, define the cyclomatical complexity of the software and software architecture, reusability of modules and life-cycle and versioning management. Some different types of integration testing are big-bang, top-down, and bottom-up, mixed (sandwich) and risky-hardest. Other Integration Patterns[2] are: collaboration integration, backbone integration, layer integration, client-server integration, distributed services integration and high-frequency integration.

5.2.1.1 Big Bang

In the big-bang approach, 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. This 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" which 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.

5.2.1.2 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.

In computer programming, unit testing is a software testing 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 whether 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 it 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 short code fragments created by programmers or occasionally by white box testers during the development process. It forms the basis for component testing. 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.

5.1.2 Benefits

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits.

- 1) Find problems early :** Unit testing finds problems early in the development cycle. In test-driven development (TDD), which is frequently used in both extreme programming and scrum, unit tests are created before the code itself is written. When the tests pass, that code is considered complete. The same unit tests are run against that function frequently as the larger code base is developed either as the code is changed or via an automated process with the build. If the unit tests fail, it is considered to be a bug either in the changed code or the tests themselves. The unit tests then allow the location of the fault or failure to be easily traced. Since the unit tests alert the development team of the problem before handing the code off to testers or clients, it is still early in the development process.
- 2) Facilitates Change :** Unit testing allows the programmer to refactor code or upgrade system libraries at a later date, and make sure the module still works correctly (e.g., in regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified. Unit tests detect changes which may break a design contract.
- 3) Simplifies Integration :** Unit testing may reduce uncertainty in the units themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier.
- 4) Documentation :** Unit testing provides a sort of living documentation of the system. Developers looking to learn what functionality is provided by a unit, and how to use it, can look at the unit tests to gain a basic understanding of the unit's interface (API). Unit test cases embody characteristics that are critical to the success of the unit. These characteristics can indicate appropriate/inappropriate use of a unit as well as negative behaviors that are to be trapped by the unit. A unit test case, in and of itself, documents these critical characteristics, although many software development environments do not rely solely upon code to document the product in development.

5.2 : 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.

5.2.1 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. Software integration testing is performed according to the software development life cycle (SDLC) after module and functional tests. The cross-dependencies for software integration testing are: schedule for integration testing, strategy and selection of the tools used for integration, define the cyclomathical complexity of the software and software architecture, reusability of modules and life-cycle and versioning management. Some different types of integration testing are big-bang, top-down, and bottom-up, mixed (sandwich) and risky-hardest. Other Integration Patterns[2] are: collaboration integration, backbone integration, layer integration, client-server integration, distributed services integration and high-frequency integration.

5.2.1.1 Big Bang

In the big-bang approach, 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. This 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" which 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.

5.2.1.2 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.

5.3 : SOFTWARE VERIFICATION AND VALIDATION

5.3.1 Introduction

In software project management, software testing, and software engineering, verification and validation (V&V) is the process of checking that a software system meets specifications and that it fulfills its intended purpose. It may also be referred to as software quality control. It is normally the responsibility of software testers as part of the software development lifecycle. Validation checks that the product design satisfies or fits the intended use (high-level checking), i.e., the software meets the user requirements. This is done through dynamic testing and other forms of review. Verification and validation are not the same thing, although they are often confused. Boehm succinctly expressed the difference between

- Validation : Are we building the right product?
- Verification : Are we building the product right?

According to the Capability Maturity Model (CMMI-SW v1.1)

Software Verification: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Software Validation: The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.

In other words, software verification is ensuring that the product has been built according to the requirements and design specifications, while software validation ensures that the product meets the user's needs, and that the specifications were correct in the first place. Software verification ensures that "you built it right". Software validation ensures that "you built the right thing". Software validation confirms that the product, as provided, will fulfill its intended use.

From Testing Perspective

- Fault – wrong or missing function in the code.
- Failure – the manifestation of a fault during execution.
- Malfunction – according to its specification the system does not meet its specified functionality

Both verification and validation are related to the concepts of quality and of software quality assurance. By themselves, verification and validation do not guarantee software quality; planning, traceability, configuration management and other aspects of software engineering are required. Within the modeling and simulation (M&S) community, the definitions of verification, validation and accreditation are similar:

- M&S Verification is the process of determining that a • computer model, simulation, or federation of models and simulations implementations and their associated data accurately represent the developer's conceptual description and specifications.
- M&S Validation is the process of determining the degree to which a model, simulation, or federation of models and simulations, and their associated data are accurate representations of the real world from the perspective of the intended use(s).

5.3.2 Classification of Methods

In mission-critical software systems, where flawless performance is absolutely necessary, formal methods may be used to ensure the correct operation of a system. However, often for non-mission-critical software systems, formal methods prove to be very costly and an alternative method of software V&V must be sought out. In such cases, syntactic methods are often used.

5.3.3 Test Cases

A test case is a tool used in the process. Test cases may be prepared for software verification and software validation to determine if the product was built according to the requirements of the user. Other methods, such as reviews, may be used early in the life cycle to provide for software validation.

5.4 : Black-Box Testing

Black-box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or workings. This method of test can be applied virtually to 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.

5.4.1 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.

5.4.2 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, often with the help of an oracle or a previous result that is known to be good, without any knowledge of the test object's internal structure.

5.5 : 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). White-box testing can be applied at the unit, integration and system levels of the software testing process. Although traditional testers tended to think of white-box testing as being done at the unit level, it is used for integration and system testing more frequently today. 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 has the potential to miss unimplemented parts of the specification or missing requirements.

5.5.1 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.

5.5.2 Procedures

White-box testing's basic procedures involves the tester having a deep level of understanding of the source code being tested. 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 analyzed for test cases to be created. These are the three basic steps that white-box testing takes in order to create test cases:

- 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.
- Processing 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.
- Output involves preparing final report that encompasses all of the above preparations and results.

5.5.3 Advantages

White-box testing is one of the two biggest testing methodologies used today. It has several major advantages:

- Side effects of having the knowledge of the source code is beneficial to thorough testing.
- Optimization of code by revealing hidden errors and being able to remove these possible defects.
- Gives the programmer introspection because developers carefully describe any new implementation.
- Provides traceability of tests from the source, allowing future changes to the software to be easily captured in changes to the tests.
- White box testing give clear, engineering-based, rules for when to stop testing.

5.5.5 Disadvantages

Although white-box testing has great advantages, it is not perfect and contains some disadvantages:

- White-box testing brings complexity to testing because the tester must have knowledge of the program, including being a programmer. White-box testing requires a programmer with a high level of knowledge due to the complexity of the level of testing that needs to be done.
- On some occasions, it is not realistic to be able to test every single existing condition of the application and some conditions will be untested.
- The tests focus on the software as it exists, and missing functionality may not be discovered.

5.6 : 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.

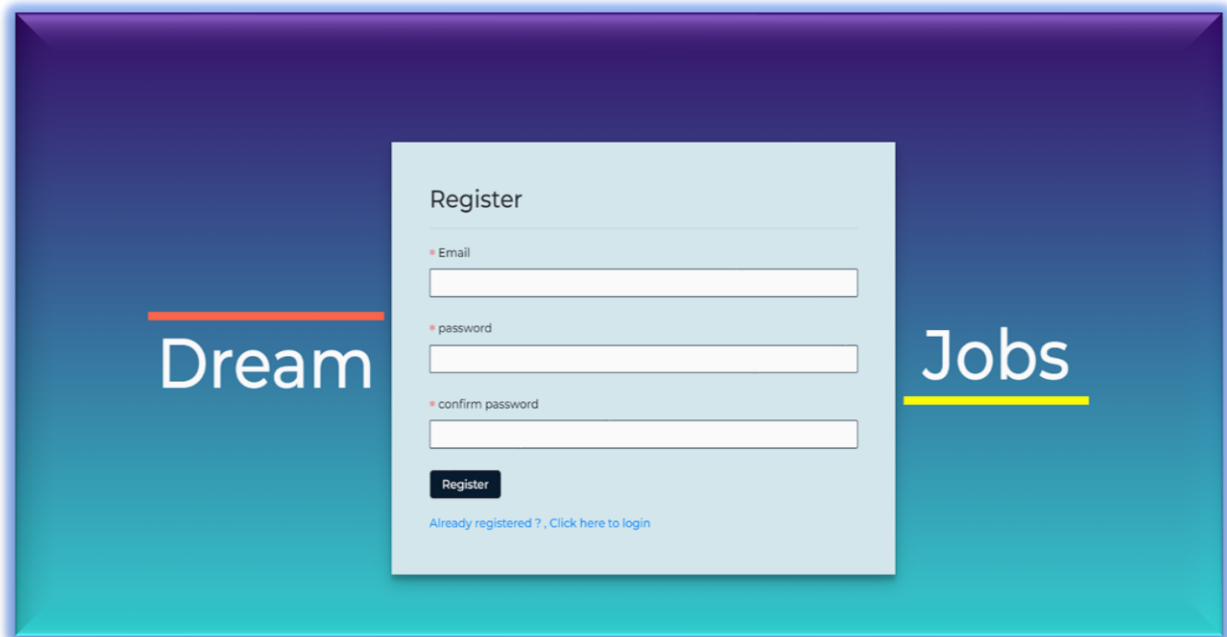
System testing is performed on the entire system in the context of a Functional Requirement Specification(s) (FRS) and/or a System Requirement Specification (SRS). System testing tests not only the design, but also the behavior and even the believed expectations of the customer. It is also intended to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

5.7 Test Cases

Project Name: Placement Guru			Project ID:GC9						
Project Manager: Vaibhav Gupta			QA Manager:Vidhi Aggarwal						
Execution Date: 11-01-2022									
Test Case Template									
TestCaselid	Component	Priority	Description/Test Summary	Pre-requisites	Test Steps	Expected Result	Actual Result	Status	Test Executed By
Enter Invalid mail id and any password and then hit login button	Login_Page	P0	Verify that user has input a valid mail Id Example: 123@123.123, should be considered as invalid	Valid Email ID	1. Enter your Email ID 2. Enter your password 3. Press on Login Button	The email ID that you have entered doesn't match any account. Sign up for an account	The email ID that you have entered doesn't match any account. Sign up for account	Pass	Team
Enter valid mail ID and incorrect password and then hit login button	Login_Page	P1	Verify that user has input a correct password that matches with an account in database	Correct Password	1. Enter your Email ID 2. Enter your password 3. Press on Login Button	The password you have entered is incorrect. forgotten password?	The password you have entered is entered incorrect. Forgotten password?	Pass	Team
Post job on job portal	post_job_Page	P2	Check all the details on job post enter valid range of CTC to search the job	job details	1. Enter valid company name 2. press on submit button	in job description company mail id salary should be valid	company mail id is valid in job description	Pass	Team
job search	job_search_Page	P3		salary>0	1. Enter salary		salary is valid	Pass	Team

CHAPTER 6: FORM DESIGN

6.1 Registration Module



A registration form titled "Register" is centered on a blue-to-teal gradient background. To the left of the form is the word "Dream" with an orange underline, and to the right is the word "Jobs" with a yellow underline. The form itself is light blue and contains three input fields labeled "Email", "password", and "confirm password", each with a red asterisk. Below the fields is a dark "Register" button and a link that says "Already registered ? , Click here to login".

Register

* Email

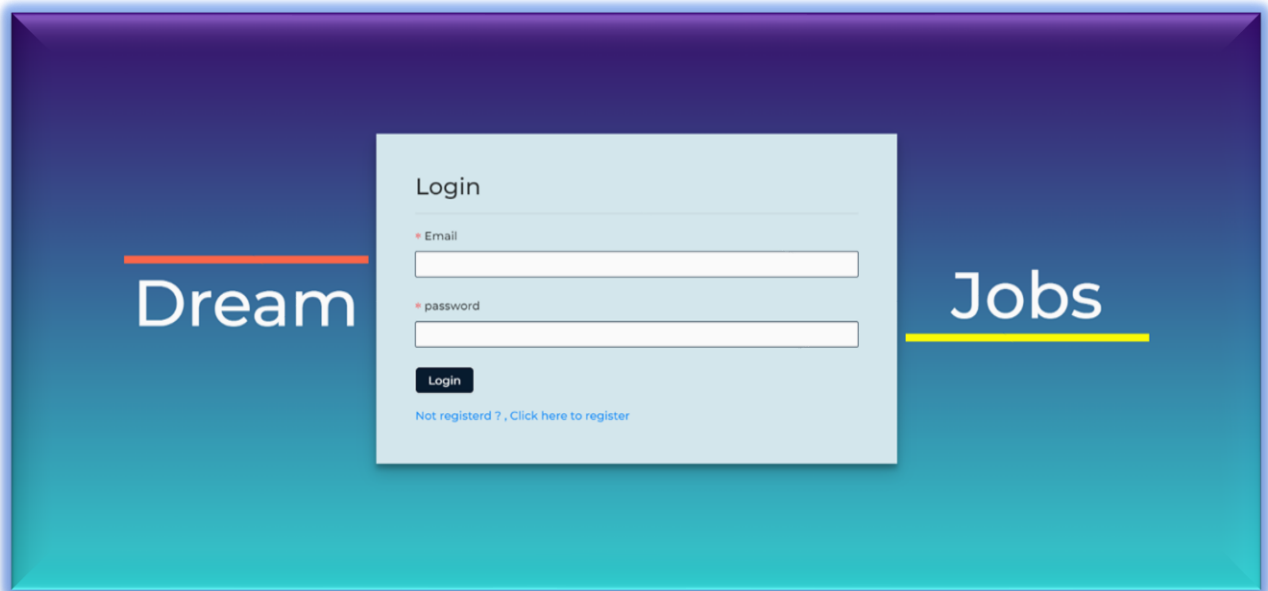
* password

* confirm password

Register

[Already registered ? , Click here to login](#)

6.2 Login module



A login form titled "Login" is centered on the same blue-to-teal gradient background. To the left of the form is the word "Dream" with an orange underline, and to the right is the word "Jobs" with a yellow underline. The form is light blue and contains two input fields labeled "Email" and "password", each with a red asterisk. Below the fields is a dark "Login" button and a link that says "Not registered ? , Click here to register".

Login

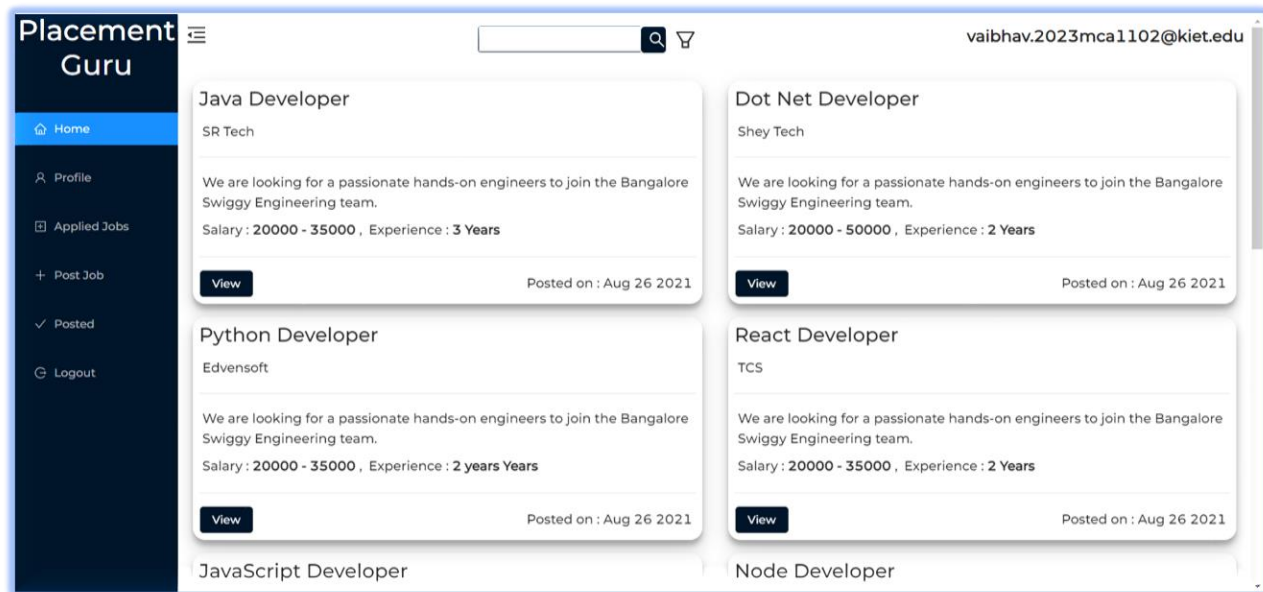
* Email

* password

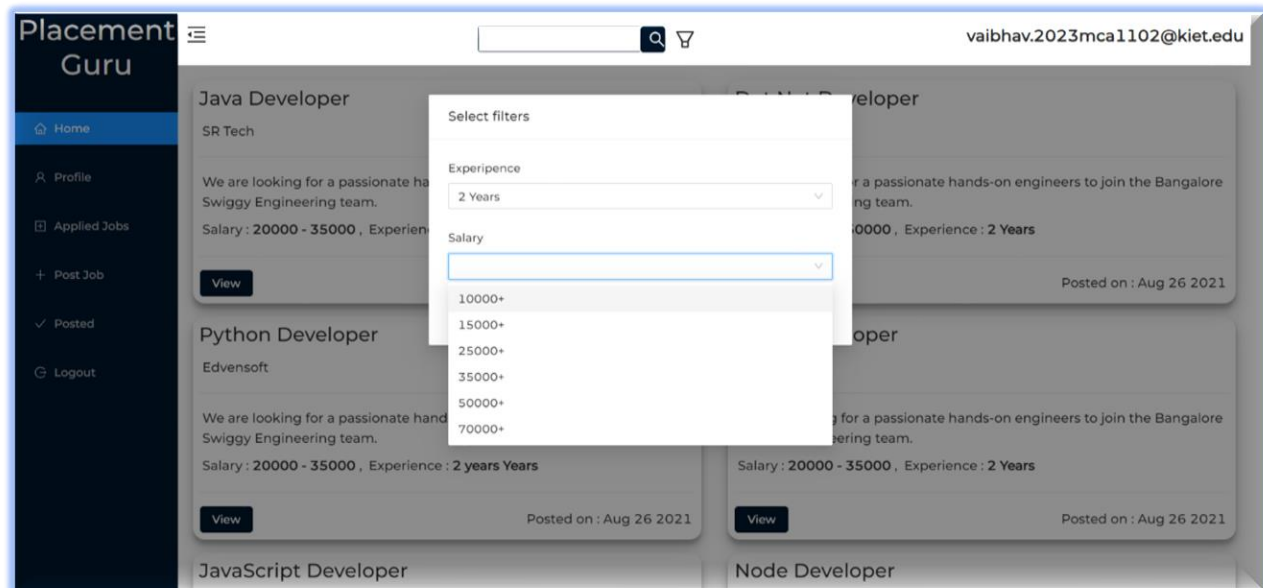
Login

[Not registered ? , Click here to register](#)

6.3 Home Module



6.4 Filter Jobs Module



6.5 Profile Module

6.5.1 Personal Info Module

The screenshot shows the 'Personal Info' module in the Placement Guru application. The interface includes a dark blue sidebar on the left with navigation links: Home, Profile (selected), Applied Jobs, Post Job, Posted, and Logout. The main content area has a header with the application name 'Placement Guru', a search bar, and the user's email 'vaibhav.2023mca1102@kiet.edu'. Below the header, there are two tabs: 'Personal Info' (active) and 'Skills and Education'. The 'Personal Info' tab contains several form fields: 'First name', 'Last name', 'Email', 'Mobile Number', and 'Portfolio'. There are also two larger text areas for 'About' and 'Address'. A 'Next' button is located at the bottom of the form.

Placement Guru

vaibhav.2023mca1102@kiet.edu

Personal Info Skills and Education

* First name * Last name * Email

* Mobile Number * Portfolio

* About

* Address

Next

6.5.2 Skills And Education Module

The screenshot shows the 'Skills and Education' module in the Placement Guru application. The interface is similar to the previous one, with the same sidebar and header. The 'Skills and Education' tab is now active. It contains four form fields: 'Education', 'Skill', 'Project', and 'Experience'. Each field has an 'Add more' button next to it.

Placement Guru

vaibhav.2023mca1102@kiet.edu

Personal Info Skills and Education

* Education

* Skill

* Project

* Experience

Add more

Add more

Add more

Add more

6.6 Applied Jobs Module

Placement
Guru

Home

Profile

Applied Jobs

Post Job

Posted

Logout

vaibhav.2023mca1102@kiet.edu

AppliedJobs

Job Title	Company	Applied Date
<div>No Data</div>		

Posttest 1000.appliedjobs

6.7 Post Job Module

Placement
Guru

Home

Profile

Applied Jobs

Post Job

Posted

Logout

vaibhav.2023mca1102@kiet.edu

Job Info

Company Info

* Title

* Department

* Experience

* Salary From

* Salary To

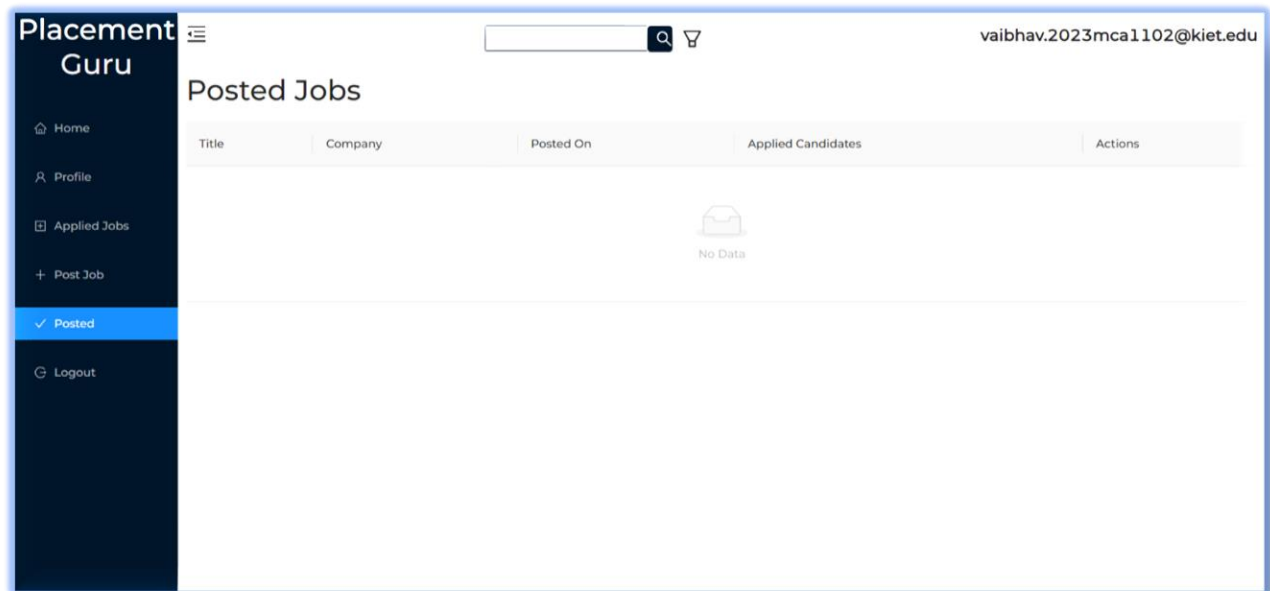
* Skills

* Minimum Qualification

* Small description

* Full description

6.8 Posted Job Module



CHAPTER 7 : ADVANTAGES

- You Can Reach a Bigger Audience.
- You Can Make Your Job Ad More Dynamic.
- User friendly.
- Flexible and durable.

CHAPTER 8 : CONCLUSION

CONCLUSION At last it can be concluded that the Job Portal System was a real learning experience. The principles of software production were well implemented throughout the system. The whole project undergoes with full of enthusiasm and with full of joyous moments. The project has been made as per as the given specification. The system has been made as user friendly. That is why Management holds an important place in the software production. Working on the Project was really a learning experience and we have come a long way in building our concepts of Software engineering. The overall purpose of this system is to computerized the whole process and thus prevent the intervening errors. We also tried to follow the holistic design principle so that the interface of the system is simple. During the course of this assignment we have gone through many obstacles which made us to research and though increased our knowledge. After applying all the data modelling, object modelling and process modelling techniques now we are very well clear with all these concepts and fundamentals which will be going to help us in the future.

CHAPTER 9 : BIBLIOGRAPHY

- <https://www.tutorialspoint.com/index.htm>
- <https://www.javatpoint.com>
- <https://www.w3schools.com>
- <https://html.com>