Table of Contents

E-Recruitment System Development Documentation

1. Problem Statement

Purpose

The purpose of this document is to outline the requirements, design, implementation, testing, and maintenance plan for an E-Recruitment System to replace the existing manual recruitment process in the organization. The system is intended to enhance the efficiency, accuracy, and effectiveness of the recruitment process.

Scope

The scope of this project includes the development of a web-based E-Recruitment system that will be used by the Human Resources (HR) department to manage job postings, handle applicant submissions, facilitate candidate screening, and generate recruitment reports. The system will serve HR personnel and job applicants.

---

2. Process Model

The process model that I will use to develop this software is Iterative Waterfall Model. The Iterative waterfall model is a sequential software development process that consists of the following phases:

1. **Feasibility Study:** The main focus of the feasibility study stage is to determine whether it would be financially and technically feasible to develop the software.

2. **Requirements Analysis and Specification:** In this phase, customer requirements are gathered and analyzed to determine what the software must do. Requirements are then documented in detail to serve as a foundation for the rest of the project.

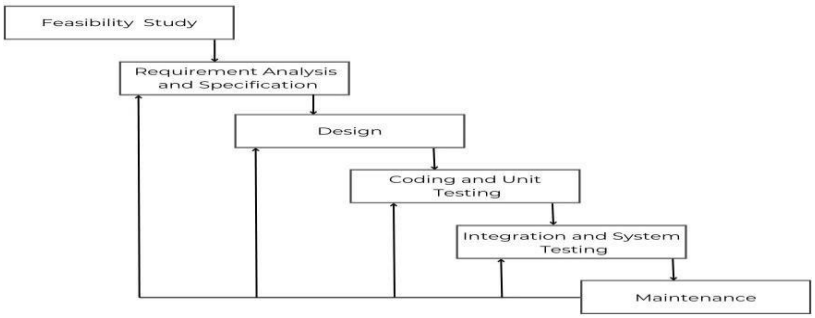
3. **Design:** This phase involves creating a design for the software based on the requirements gathered in the previous phase. The design should provide a comprehensive understanding of howthe software will meet the requirements.

4. **Coding and Unit Testing:** In this phase, the actual coding of the software takes place based on the design developed in the previous phase. The software is tested in individual units in this phaseto ensure it meets the requirements and functions as expected.

5. **Integration and System Testing:** In this phase the different modules are integrated incrementally and tested in a planned manner. After all the modules have been successfully integrated and tested, system testing is carried out on this fully working system.

6. **Maintenance:** In this phase, the project team provides ongoing support and maintenance for the solution.

Diagram.



3. **Feasibility Study:**

The software that I am going to develop is financially and technically feasible. In order to develop this software, I am going to use Python and MySQL. Python is developed under OSI-approved open-source license, making it freely usable and distributable. MySQL is an Open-Source Relational database management system. The computer available in the laboratory of our department contains Linux Operating System which is an Open-Source System software. The hardware configuration of the computers satisfies the minimum hardware requirement for installing and using Python and MySQL.

I. **Financial feasibility:** It is financially feasible because the hardware resources are available in the laboratory of our department where I am going to develop the software and the software resources required are freely available on the internet.

II. **Technical Feasibility:** It is technical feasible because the hardware and software resourcesavailable and I am familiar with resources to develop the software.

3.1 Scope of the Project:

The proposed system will affect or interface with the activities of graduate, employer and administrator. The system works and fulfills all the functionalities as per the proposed system. It will provide reduced response time against the queries made by different users. The administrator will have a clear view ofnumber of vacancies for a particular job, number of candidates applied and number of candidatesselected. All possible features such as verification, validation, security, user friendliness etc have beenconsidered.

The different types of modules present in this project are

Proposed System:

1. User Interface
   - Login
   - Register
   - Change password
   - Edit profile
2. Apply for jobs
   - Online exam
   - Deployment of question paper
   - Automatic uploading
   - Evaluation of answer sheets
3. Check status
   - employer
   - graduate
4. Verification
   - National Identity Card
   - Passport no
   - Graduation registration no
5. Administration
   - Create employer
   - Delete employer
6. Generate report
   - Generates the no of graduates registered
   - Generates the no of graduates selected

4. Project Management.

The main goal of software project management is to enable a group of developers to work effectively towards the successful completion of a project. A software project manager takes the overall responsibility of steering a project to success. We can broadly classify a project manager's varied responsibilities into the following two major categories:

Project planning, and Project monitoring and control

Project planning: Project planning involves estimating several characteristics of a project and then planning the project activities based on these estimates made.

Project monitoring and control: The focus of project monitoring and control activities is to ensure that the software development proceeds as per plan.

Once project planning is complete, project managers document their plans in a software project management plan (SPMP) document. The SPMP document contains different items out of which few items are discussed below:

Project estimates

a. Lines of Code (LOC): This metric measures the size of a project by counting the number of source instructions in the developed program. Determining the LOC count at the end of a project is very simple. However, accurate estimation of LOC count at the beginning of a project is a very difficult task. One canpossibly estimate the LOC count at the starting of a project, only by using some form of systematic guesswork. For my project I am assuming that the number of source instructions would be 300. Therefore,

LOC=300 KLOC=300/1000

b. Basic COCOMO Model(A Heuristic Estimation Technique):

Constructive Cost estimation Model (COCOMO) was proposed by Boehm 1981. Using COCOMO we can calculate Effort, Time and Cost required for developing a project.

Boehm postulated that any software development project can be classified into one of the following three categories based on the development complexity-organic, semidetached, and embedded. We know that a project is of organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar types of projects.

I am considering the project that I am going to develop to be of organic type because the project is a well-understood application program, the size of the development team is reasonably small as I am the only person who is working in this project and assuming that I am experienced in developing similar types of projects.

Now, I am can calculate the effort and time required to develop the product as follows:

Estimation of Development Effort

Effort = a1 $(KLOC)^{a2}$PM

=2.4$(KLOC)^{1.05}$ PM

$=2.4(300/1000)^{1.05}$ PM

=0.6779 PM

Estimation of Development Time

Tdev =b1 (Effort) ^{b1} months

=b1 (Effort) ^{b2} months

=2.5 (0.6779)0.38 months

=2.16 months

So, I can conclude that in order to develop my project approximately 2.16 months is required.

I am not going to calculate the cost of this project since I am developing this project only for learning purpose.

Schedule:

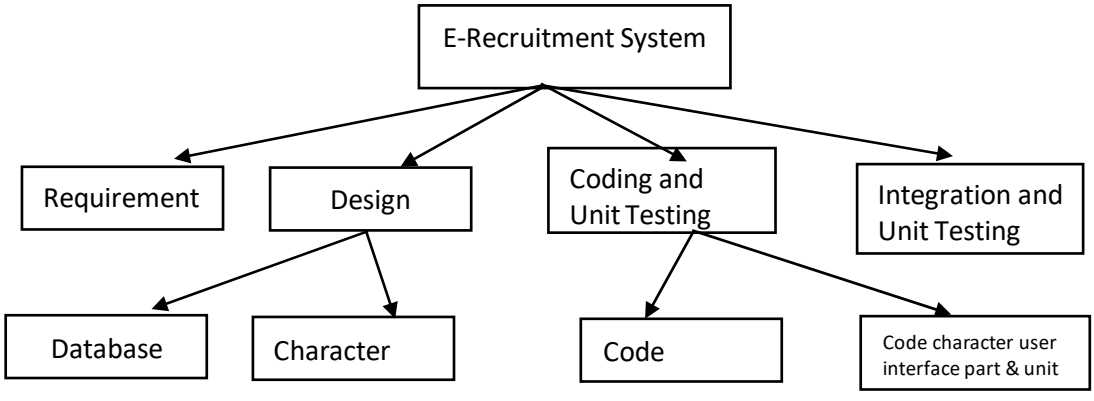
Scheduling the project tasks is an important project planning activity. The scheduling problem, in essence, consists of deciding which tasks would be taken up when and by whom. Once a schedule has been worked out and the project gets underway, the project manager monitors the timely completion of the tasks and takes any corrective action that may be necessary whenever there is a chance of schedule slippage.

In order to schedule the project activities, a software project manager needs to do the following:

- Identify all the major activities that need to be carried out to complete the project.
- Break down each activity into tasks.
- Determine the dependency among different tasks.
- Establish the estimates for the time durations necessary to complete the tasks.
- Represent the information in the form of an activity network.
- Determine task starting and ending dates from the information represented in the activity network.
- Determine the critical path. A critical path is a chain of tasks that deter- mines the duration of theproject.
- Allocate resources to tasks

a. Work Breakdown Structure:

Work breakdown structure (WBS) is used to recursively decompose a given set of activities into smaller activities. The Work breakdown structure corresponding to my project is as follows:

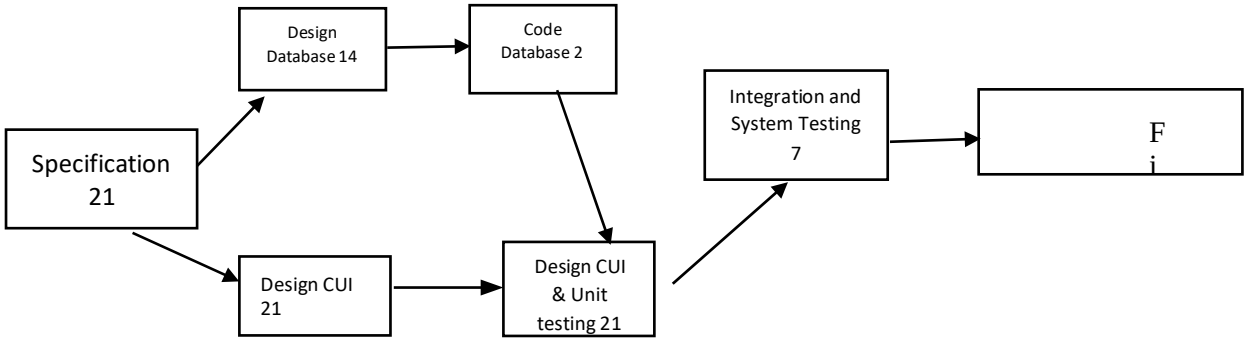


b. Activity Networks:

An activity network shows the different activities making up a project, their estimated durations, and their interdependencies. I have determined the tasks to be represented from the work breakdown structure given above and have determined the durations and dependencies for each task as shown in the following table:

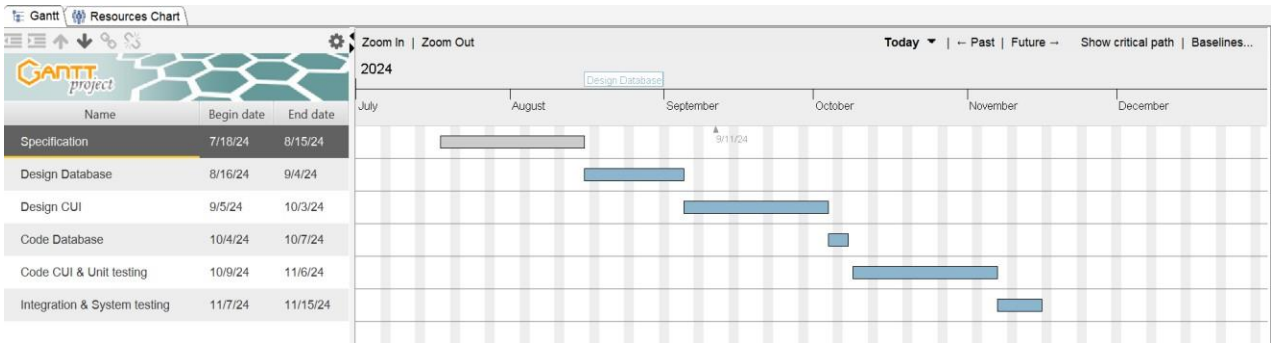
Task No.	Task	Duration (in days)	Dependent on Task
T1	Specification	21	-
T2	Design Database	14	T1
T3	Design CUI	21	T1
T4	Code Database	2	T2
T5	Code CUI & Unit Testing	21	T3 & T4
T6	Integration & System Testing	7	T4 & T5

The corresponding activity network representation has been shown in the following figure:



a. Gantt Chart:

A Gantt chart is a special type of bar chart where each bar represent an activity. The bars are drawn along a time line. The length of each bar is proportional to the duration of time planned for the corresponding activity. The Gant chart corresponding to my project is as follows:



Project resources:

(a) People: Only one person (i.e. myself) is involved in this project. I have performed and will performall possible tasks required to develop this project.

(b) Hardware and Software:

Hardware:

CPU: x86 64-bit CPU (Intel/AMD Architecture) Cores: Single (Dual/Quad Core is recommended) RAM: 4 GB (6 GB recommended)

Disk Space: Minimum 5 GB free

Software:

OS: Ubuntu Interpreter: PHP,Javascript and SQL

Programming Language: Python DBMS: MySQLOthers: LibreOffice Write

5. Requirement Analysis Specification

As we know, the two activities that are carried out in this phase are:

a) Requirement Gathering: Since I am developing the software in the laboratory for understanding the engineering way to develop a software, I have collected the information required to develop the software from our class teachers, senior, classmates through the process of discussion. Expect that I have collected the information from internet and by studying the document prepared by our seniors.

b) Requirement Analysis: After gathering the requirements, I have analyzed the requirement to obtain a clear understanding of the product and remove the ambiguities, incompleteness and inconsistencies

c) Requirement Specifications: After analyzing the gathered requirements, we can systematically organize the requirements in the form of Software 7 Requirements Specification (SRS) document. The SRS document contains several information out of which the most important parts are discussed below

Goals of Implementation:

a) In future the system should be able to include functionalities to perform login and logout in order to make the system secure

b) In future the system should be able to categories the functionalities depending on different class of users.

5.1.1 Functional Requirements

Job Posting Management

HR personnel can create, edit, and delete job postings. Job details include title, description, qualifications, etc.Job postings are displayed to job seekers.

Applicant Management

Applicants can register, log in, and apply for jobs.

Applicant information includes personal details, resume, cover letter, etc.Applications are stored in the database.

Candidate Screening

The system provides tools for initial screening, including online tests. Test responses are scored and ranked.

Interview Scheduling

HR can schedule interviews with candidates.

Interview details include date, time, and venue/online link. Candidates are notified of their interview schedules.

Reporting

The system generates recruitment reports, such as applicant numbers, time to hire, etc.

Non-Functional Requirements

Performance Requirements

System response time must be under 2 seconds per user action. The system must handle at least 5000 concurrent users.

Security Requirements

Data must be encrypted during transmission and storage. User authentication must prevent unauthorized access.

Usability Requirements

The UI should be user-friendly and intuitive.A help section must be available for users.

Compatibility Requirements

The system must be compatible with major web browsers (Chrome, Firefox, and Safari). The system must operate on Linux servers.

System Interfaces

Database Interface: Interaction with MySQL for data storage and retrieval.

User Interface: Web-based UI using HTML, CSS, and JavaScript.

System API: RESTful APIs for potential integration with other systems.

6.System Design

System Architecture

Presentation Layer: Developed using HTML, CSS, and JavaScript for the user interface.

Application Layer: Developed using PHP, responsible for business logic.

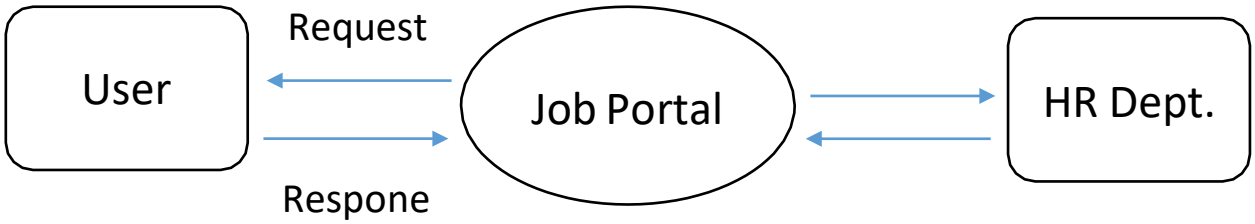
Database Layer: MySQL database for storing job postings, applicant data, test results, and reports.

a. Design CUI

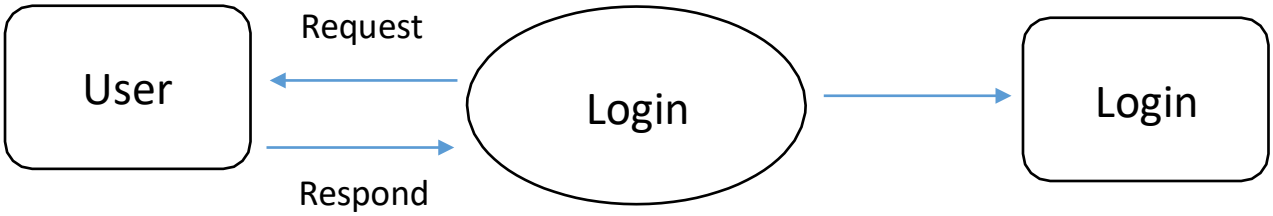
a. Data Flow Diagrams (DFDs)

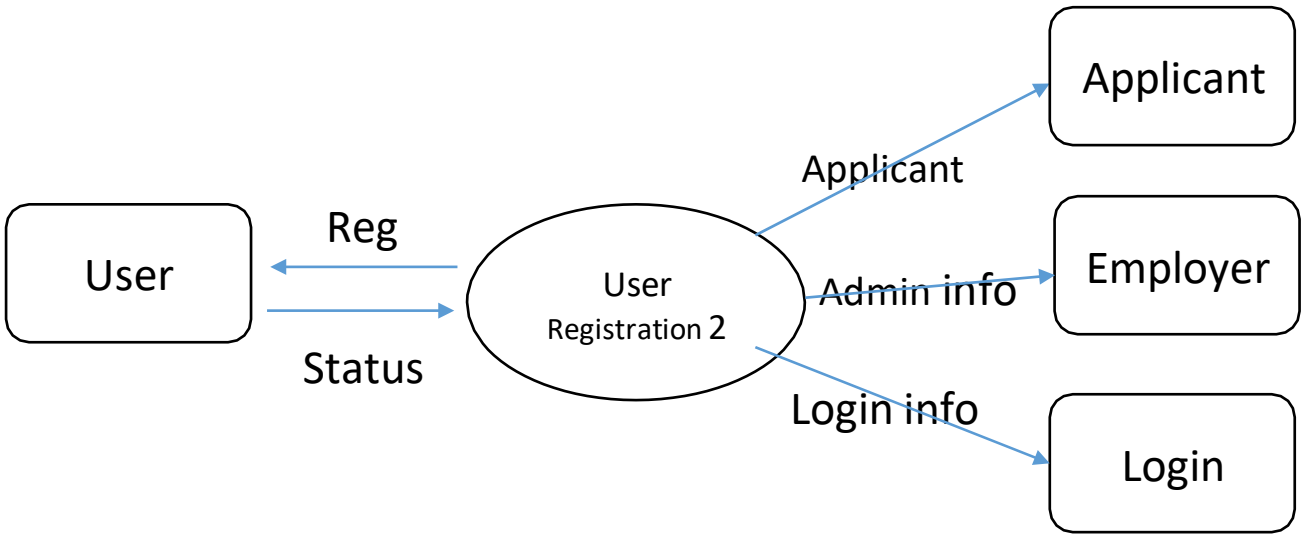
DFD Level 0 (Context Diagram)

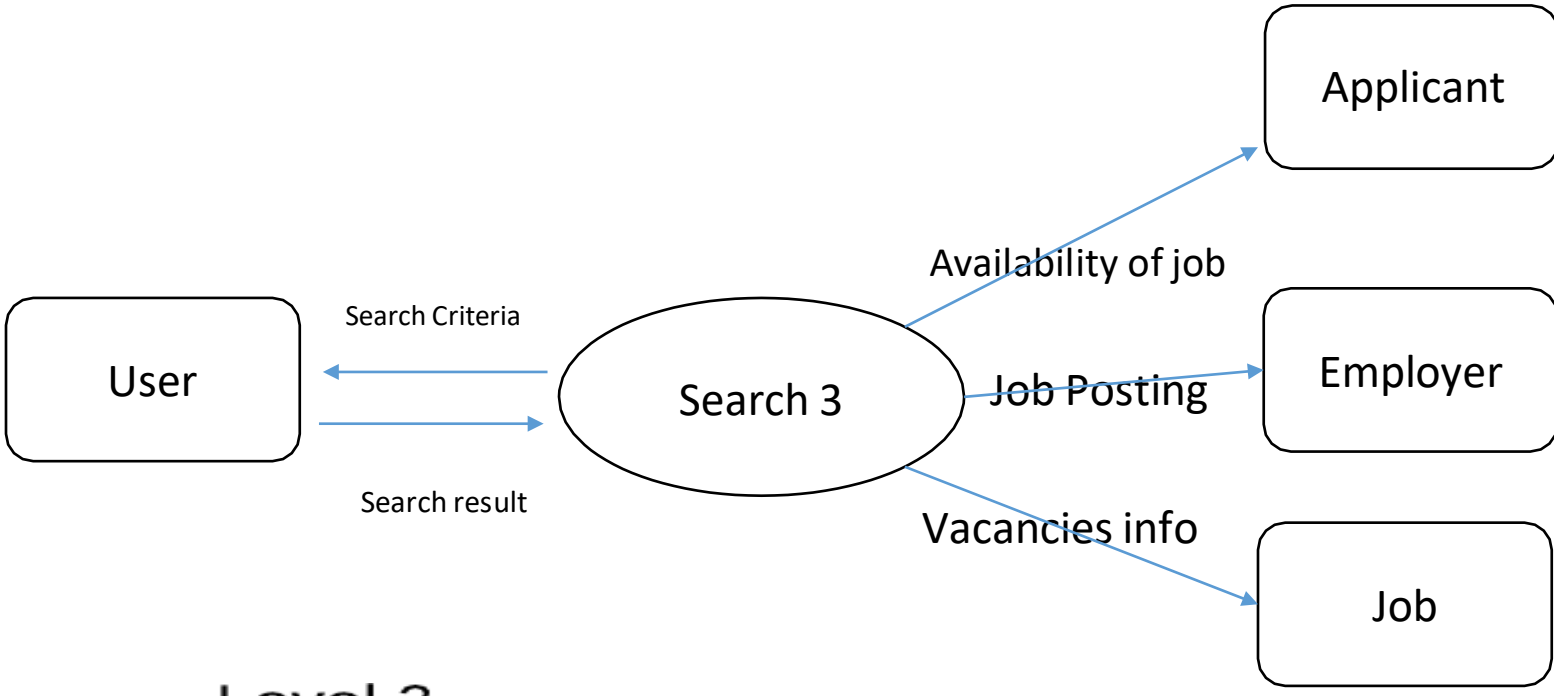
Shows interaction between users (HR and applicants) and the system.



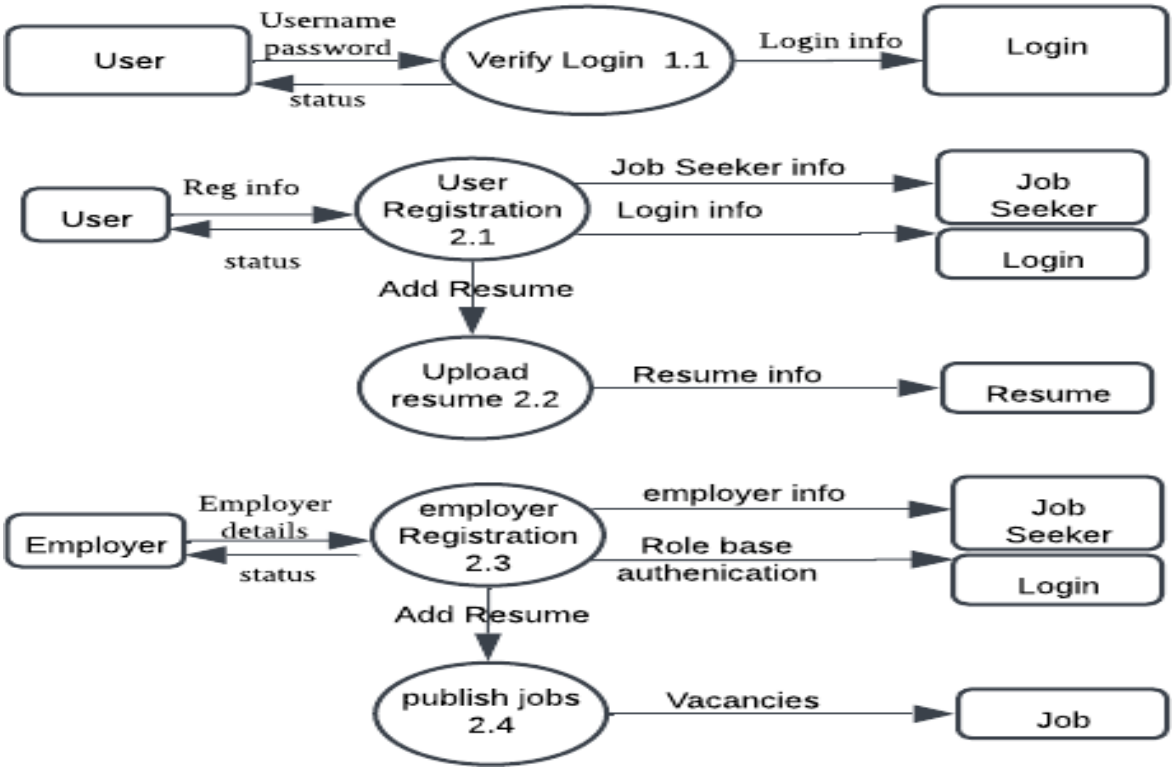
Level 1

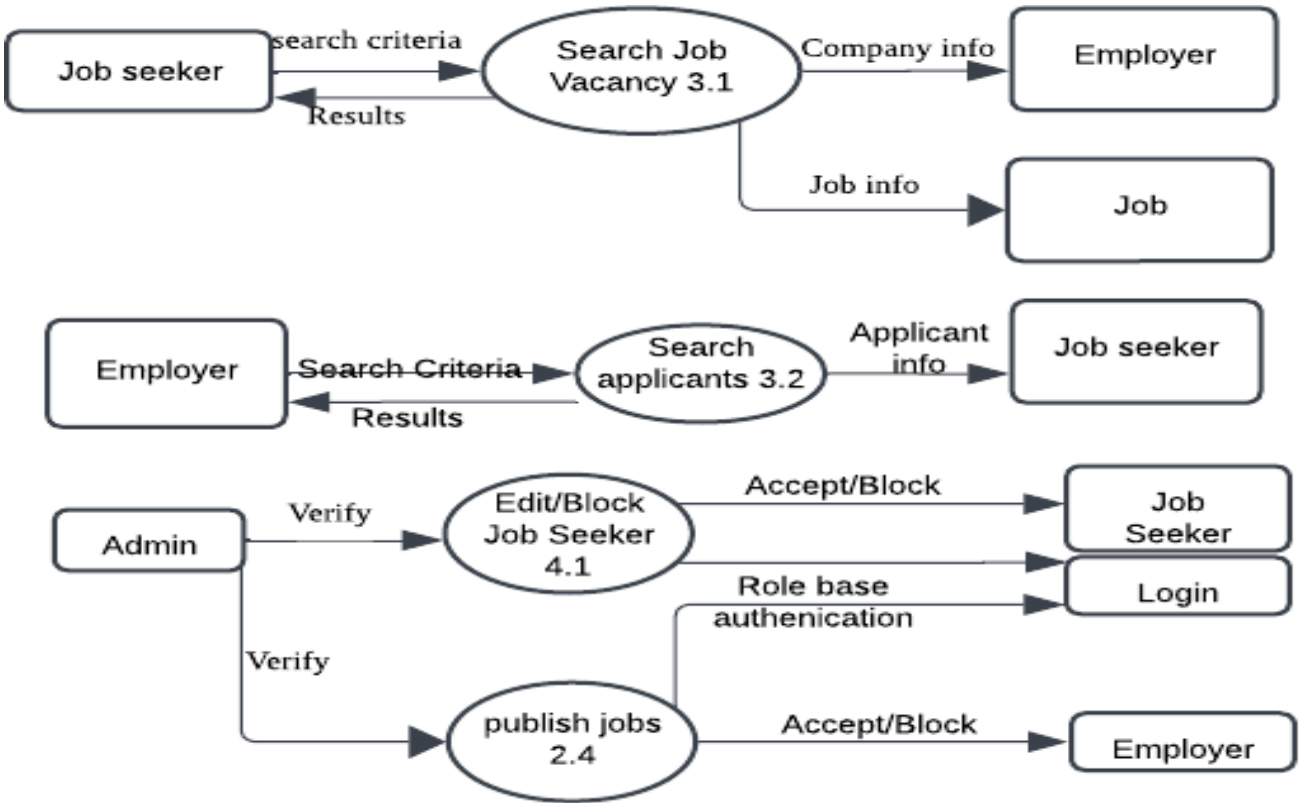





Applicant
Availability of job
User
Search Criteria
Search 3
Job Posting
Employer
Search result
Vacancies info
Job

Level 2


User
Username
password
status
Verify Login 1.1
Login info
Login
User
Reg info
status
User Registration 2.1
Job Seeker info
Login info
Job Seeker
Login
Add Resume
Upload resume 2.2
Resume info
Resume
Employer
Employer details
status
employer Registration 2.3
employer info
Role base authenication
Job Seeker
Login
Add Resume
publish jobs 2.4
Vacancies
Job

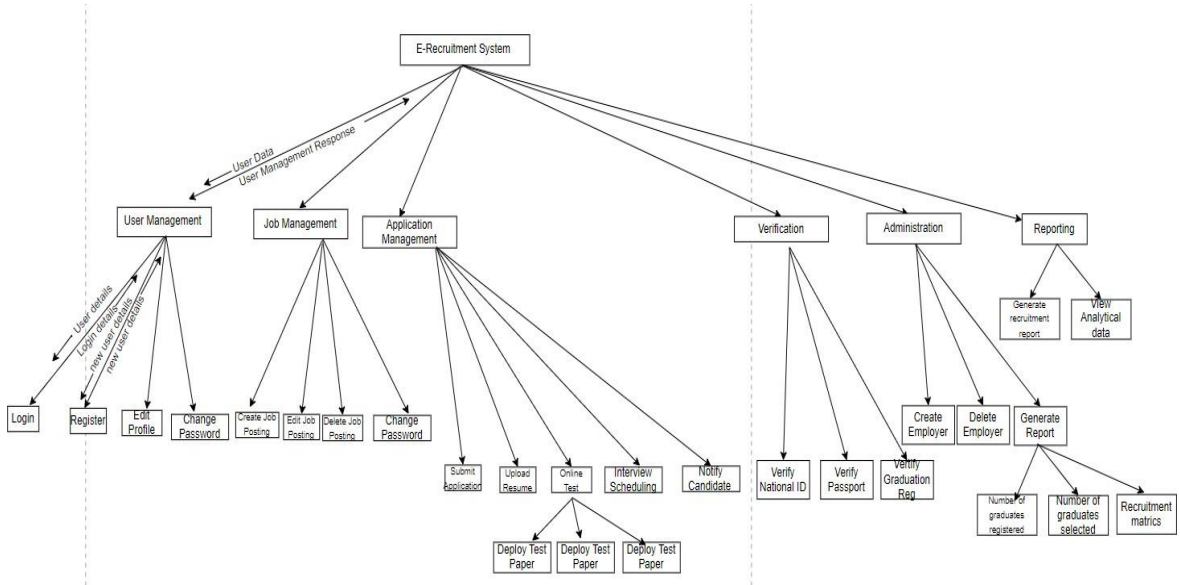


b) Data Dictionary:

Field Name	Data Type	Description	Constraints
UserID	Integer	Unique identifier for each user	Primary Key, Auto Increment
Username	Varchar(50)	User's login name	Unique, Not Null
Password	Varchar(255)	Encrypted user password	Not Null
Email	Varchar(100)	User's email address	Unique, Not Null
PhoneNumber	Varchar(15)	User's contact number	Not Null
Role	Varchar(20)	Role of the user (e.g., Admin, Employer, Applicant)	Not Null
JobID	Integer	Unique identifier for each job posting	Primary Key, Auto Increment
JobTitle	Varchar(100)	Title of the job posting	Not Null
JobDescription	Text	Detailed description of the job	Not Null

Qualifications	Text	Required qualifications for the job	Not Null
ApplicationID	Integer	Unique identifier for each job application	Primary Key, Auto Increment
ApplicantID	Integer	Identifier linking to the applicant	Foreign Key (UserID), Not Null
Resume	Blob	Uploaded resume of the applicant	Not Null
CoverLetter	Text	Cover letter provided by the applicant	Optional
TestScore	Float	Score of the applicant from the screening test	Default = 0
InterviewDate	DateTime	Scheduled interview date and time	Optional
ReportID	Integer	Unique identifier for each recruitment report	Primary Key, Auto Increment
ReportType	Varchar(50)	Type of the report (e.g., Application Summary, Selected Candidates)	Not Null
GeneratedDate	DateTime	Date and time the report was generated	Default = Current Timestamp

c.) Structure Chart



d. Algorithms:

Algorithm 1: Authenticate User

Purpose: Handles login verification for users.
Input: username, password
Output: success or failure

Steps**:**

Retrieve user record from the database where Username = username.

If no record is found:

Return "Failure: User not found".

Else:

Retrieve stored_hashed_password.

Hash the input password.

If hashed password matches stored_hashed_password:

Return "Success: User authenticated".

Else:

Return "Failure: Incorrect password".

Algorithm 2: Create Job Post

Purpose: Creates a new job posting in the system.
Input: job title, job description, qualifications, employer_id
Output: confirmation message
Steps**:**

Validate input fields (job title, job description, qualifications):

If any field is empty:

Return "Error: All fields are required"**.**

Insert the job details into the Job Postings table:

INSERT INTO Job Postings (job title, job description, qualifications, employer_id).

If insertion is successful:

Return **"Success: Job post created"**.

Else:

Return "Error: Job post creation failed"**.**

Algorithm 3: Submit Application

Purpose: Allows an applicant to submit a job application.
Input: applicant_id, job_id, resume, cover letter
Output: confirmation message

Steps**:**

Validate input fields (applicant_id, job_id, resume):

If any field is empty:

Return "Error: Missing required information"**.**

Check if applicant_id exists in Users table and job_id exists in Job Postings table:

If either does not exist:

Return "Error: Invalid applicant or job ID"**.**

Insert application data into Applications table:

INSERT INTO Applications (applicant_id, job_id, resume, cover letter).

If insertion is successful:

Return "Success: Application submitted"**.**

Else:

Return "Error: Application submission failed"**.**

Algorithm 4: Evaluate Test

Purpose: Automatically evaluates and ranks online test responses.
Input: applicant_id, job_id, test answers[]
Output: test_score, rank

Steps**:**

Retrieve correct answers for the job_id from the Test Bank table.

Initialize test_score = 0.

For each question in test_answers:

If test_answers[i] matches correct_answers[i]:

Increment test_score by the weight of the question.

Store the test_score in the TestResults table:

INSERT INTO TestResults (applicant_id, job_id, test_score).

Retrieve all test_scores for job_id and rank applicants:

SELECT applicant_id, RANK() OVER (ORDER BY test_score DESC) AS rank FROMTestResults.

Return test_score and rank for the applicant.

Algorithm 5: Schedule Interview

Purpose: Assigns an interview slot to a shortlisted candidate.
Input: applicant_id, job_id, interview_date, interview_time, location
Output: confirmation message

Steps**:**

Check if applicant_id is shortlisted for job_id:

SELECT status FROM Applications WHERE applicant_id = applicant_id AND job_id =job_id.

If status != "shortlisted":

Return "Error: Applicant is not shortlisted"**.**

Insert interview details into Interviews table:

INSERT INTO Interviews (applicant_id, job_id, interview_date, interview_time,location).

Notify the applicant:

SendEmail(applicant_email, "Interview Scheduled", details).

Return "Success: Interview scheduled and notification sent

Algorithm 6: Generate Recruitment Report

Purpose: Generates recruitment summary reports for HR.
Input: report_type, job_id
Output: report data

Steps**:**

If report_type == "Applications Summary":

SELECT job_id, COUNT(*) AS total_applications FROM Applications WHERE job_id = job_id.

Else if report_type == "Shortlisted Candidates":

SELECT applicant_id, name, email FROM Applications WHERE job_id = job_id ANDstatus = "shortlisted".

Else if report_type == "Time-to-Hire":

SELECT AVG(DATEDIFF(hire_date, application_date)) AS avg_time FROM HiresWHERE job_id = job_id.

Return the report data.

Design Database Part

a) Initial database schemas:

Database Design for E-Recruitment System
Below is the database schema design with tables, attributes, data types, and relationships:

1. Users Table

Purpose: Stores information about all users (applicants, employers, and admins).
Primary Key: `UserID`

Field Name	Data Type	Description	Constraints
`UserID`	INT	Unique identifier for each user	Primary Key, Auto Increment
`Username`	VARCHAR(50)	Username of the user	Unique, Not Null
`Password`	VARCHAR(255)	Encrypted password	Not Null
`Email`	VARCHAR(100)	User email	Unique, Not Null
`Field Name`	Data Type	Description	Constraints
`PhoneNumber`	VARCHAR(15)	Contact number	Not Null
`Role`	VARCHAR(20)	Role (Admin, Employer, Applicant)	Not Null

2. JobPostings Table

Purpose: Stores job posting information.
Primary Key: `JobID`
Foreign Key: `EmployerID` (references `Users.UserID`)

Field Name	Data Type	Description	Constraints
`JobID`	INT	Unique identifier for each job	Primary Key, Auto Increment

JobTitle	VARCHAR(100)	Title of the job	Not Null
JobDescription	TEXT	Detailed job description	Not Null
Qualifications	TEXT	Required qualifications for the job	Not Null
EmployerID	INT	Reference to the employer who posted	Foreign Key (Users.UserID)

3. Applications Table

Purpose: Stores job application details.
Primary Key: ApplicationID
Foreign Keys: ApplicantID (references Users.UserID), JobID (references JobPostings.JobID)

Field Name	Data Type	Description	Constraints
ApplicationID	INT	Unique identifier for each application	Primary Key, Auto Increment
ApplicantID	INT	Reference to the applicant	Foreign Key (Users.UserID)
JobID	INT	Reference to the job applied for	Foreign Key (JobPostings.JobID)
Resume	BLOB	Uploaded resume	Not Null
CoverLetter	TEXT	Cover letter	Optional
ApplicationDate	DATETIME	Timestamp of application submission	Default = Current Timestamp

---

4. TestResults Table

Purpose: Stores results of online tests.
Primary Key: TestResultID
Foreign Keys: ApplicantID (references Users.UserID), JobID (references JobPostings.JobID)

Field Name	Data Type	Description	Constraints
TestResultID	INT	Unique identifier for test result	Primary Key, Auto Increment
ApplicantID	INT	Reference to the applicant	Foreign Key (Users.UserID)
JobID	INT	Reference to the related job	Foreign Key (JobPostings.JobID)
TestScore	FLOAT	Score obtained in the test	Default = 0

5. Interviews Table

Purpose: Stores interview schedules.
Primary Key: InterviewID
Foreign Keys: ApplicantID (references Users.UserID), JobID (references JobPostings.JobID)

Field Name	Data Type	Description	Constraints
InterviewID	INT	Unique identifier for interview	Primary Key, Auto Increment
ApplicantID	INT	Reference to the applicant	Foreign Key (Users.UserID)
JobID	INT	Reference to the related job	Foreign Key (JobPostings.JobID)
InterviewDate	DATETIME	Scheduled interview date and time	Not Null
Location	VARCHAR(255)	Location of the interview	Optional

6. Reports Table

Purpose: Stores generated recruitment reports.

Primary Key: `ReportID`

Field Name	Data Type	Description	Constraints
`ReportID`	INT	Unique identifier for the report	Primary Key, Auto Increment
`ReportType`	VARCHAR(50)	Type of report (e.g., summary, metrics)	Not Null
`GeneratedDate`	DATETIME	Timestamp of report generation	Default = Current Timestamp

Relationships between Tables

Users (1) ↔ (N) Job Postings: An employer can post multiple jobs.
Users (1) ↔ (N) Applications: An applicant can submit multiple applications.
Job Postings (1) ↔ (N) Applications: A job can have multiple applicants.
Users (1) ↔ (N) Test Results: An applicant can have test results for different jobs.
Users (1) ↔ (N) Interviews: An applicant can have multiple interview schedules.

7. Coding and Unit Testing

a. Database Schema

Use the following SQL commands to create the required tables:

```
sql
Copy code
CREATE DATABASE e_recruitment;USE e_recruitment;
CREATE TABLE Users (
UserID INT AUTO_INCREMENT PRIMARY KEY,Username VARCHAR(50) UNIQUE NOT NULL, Password
VARCHAR(255) NOT NULL,
Email VARCHAR(100) UNIQUE NOT NULL,
Role ENUM('Admin', 'Employer', 'Applicant') NOT NULL
);

CREATE TABLE JobPostings (
JobID INT AUTO_INCREMENT PRIMARY KEY,JobTitle VARCHAR(100) NOT NULL,
JobDescription TEXT NOT NULL,Qualifications TEXT NOT NULL,EmployerID INT,
FOREIGN KEY (EmployerID) REFERENCES Users(UserID)
);

CREATE TABLE Applications (
ApplicationID INT AUTO_INCREMENT PRIMARY KEY,
ApplicantID INT,JobID INT,
Resume TEXT NOT NULL,
CoverLetter TEXT,
FOREIGN KEY (ApplicantID) REFERENCES Users(UserID),FOREIGN KEY (JobID) REFERENCES
JobPostings(JobID)
);
```

Frontend and Backend Implementation

Registration and Login (HTML + PHP)

a. Registration Form

```
html
Copy code
<!-- register.html -->
<!DOCTYPE html>
<html lang="en">
```

```
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Register</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
<h2>User Registration</h2>
<form action="register.php" method="POST">
<label for="username">Username:</label>
<input type="text" name="username" required><br>
<label for="password">Password:</label>
<input type="password" name="password" required><br>
<label for="email">Email:</label>
<input type="email" name="email" required><br>
<label for="role">Role:</label>
<select name="role" required>
<option value="Admin">Admin</option>
<option value="Employer">Employer</option>
<option value="Applicant">Applicant</option>
</select><br>
<button type="submit">Register</button>
</form>
</body>
</html>
```

b. Registration Backend

```
php
<!-- register.php -->
<?php
$host = 'localhost';
$user = 'root';
$password = '';
$dbname = 'e_recruitment';
$conn = new mysqli($host, $user, $password, $dbname);if ($_SERVER['REQUEST_METHOD'] == 'POST') {
$username = $_POST['username'];
$password = password_hash($_POST['password'], PASSWORD_BCRYPT);
$email = $_POST['email'];
$role = $_POST['role'];

$sql = "INSERT INTO Users (Username, Password, Email, Role) VALUES (?, ?,
?, ?)";
$stmt = $conn->prepare($sql);
$stmt->bind_param("ssss", $username, $password, $email, $role);
if ($stmt->execute()) {
echo "Registration successful!";header("Location: login.html");
} else {
echo "Error: " . $stmt->error;
}
$stmt->close();
}
$conn->close();
?>

 Login Form
<!-- login.html -->
<!DOCTYPE html>
<html lang="en">
```

```
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Login</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
<h2>User Login</h2>
<form action="login.php" method="POST">
<label for="username">Username:</label>
<input type="text" name="username" required><br>
<label for="password">Password:</label>
<input type="password" name="password" required><br>
<button type="submit">Login</button>
</form>
</body>
</html>
```

Login Backend

```
php
Copy code
<!-- login.php -->
<?php session_start();
$host = 'localhost';
$user = 'root';
$password = '';
$dbname = 'e_recruitment';
$conn = new mysqli($host, $user, $password, $dbname);if ($_SERVER['REQUEST_METHOD'] == 'POST') {
$username = $_POST['username'];
$password = $_POST['password'];

$sql = "SELECT * FROM Users WHERE Username = ?";
$stmt = $conn->prepare($sql);

$stmt->bind_param("s", $username);
$stmt->execute();
$result = $stmt->get_result();

if ($result->num_rows > 0) {
$user = $result->fetch_assoc();
if (password_verify($password, $user['Password'])) {
$_SESSION['UserID'] = $user['UserID'];
$_SESSION['Role'] = $user['Role'];
$_SESSION['Username'] = $user['Username'];echo "Login successful!"; header("Location: dashboard.php");
} else {
echo "Invalid password!";
}
} else {
echo "User not found!";
}
$stmt->close();
}
$conn->close();
?>
```

b. Job Posting

a. Post Job Form

```
html
Copy code
<!-- post_job.html -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Post a Job</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
<h2>Post a Job</h2>
<form action="post_job.php" method="POST">
<label for="title">Job Title:</label>
<input type="text" name="title" required><br>
<label for="description">Job Description:</label>
<textarea name="description" required></textarea><br>
<label for="qualifications">Qualifications:</label>
<textarea name="qualifications" required></textarea><br>
<button type="submit">Post Job</button>
</form>
</body>
</html>
```

b. Post Job Backend

```
<!-- post_job.php -->
<?php session_start();
if ($_SESSION['Role'] !== 'Employer') {die("Access denied!");
}

$host = 'localhost';
$user = 'root';
$password = '';
$dbname = 'e_recruitment';
$conn = new mysqli($host, $user, $password, $dbname);if ($_SERVER['REQUEST_METHOD'] == 'POST') {
$title = $_POST['title'];
$description = $_POST['description'];
$qualifications = $_POST['qualifications'];
$employer_id = $_SESSION['UserID'];

$sql = "INSERT INTO JobPostings (JobTitle, JobDescription,Qualifications, EmployerID) VALUES (?, ?, ?, ?)";
$stmt = $conn->prepare($sql);
$stmt->bind_param("sssi", $title, $description, $qualifications,
$employer_id);

if ($stmt->execute()) {
echo "Job posted successfully!";
} else {
echo "Error: " . $stmt->error;
}
$stmt->close();
}
$conn->close();
?>
```

c. Viewing Job Listings

```
<!-- jobs.html -->
```

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Job Listings</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
<h2>Available Jobs</h2>
<table border="1">
<thead>
<tr>
<th>Job ID</th>
<th>Job Title</th>
<th>Description</th>
<th>Qualifications</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<?php
$host = 'localhost';
$user = 'root';
$password = '';
$dbname = 'e_recruitment';

$conn = new mysqli($host, $user, $password, $dbname);

$sql = "SELECT * FROM JobPostings";
$result = $conn->query($sql);

while ($job = $result->fetch_assoc()) {echo "<tr>";
echo "<td>{$job['JobID']}</td>"; echo "<td>{$job['JobTitle']}</td>";
echo "<td>{$job['JobDescription']}</td>";echo "<td>{$job['Qualifications']}</td>";echo "<td><a
href='apply.php?job_id={$job['JobID']}'>Apply</a></td>"; echo "</tr>";
}

$conn->close();
?>
</tbody>
</table>
</body>
</html>
```

d. Apply for Job Page

```
<!-- apply.php -->
<?php session_start();
if ($_SESSION['Role'] !== 'Applicant') {
die("Access denied!");
}
$host = 'localhost';
$user = 'root';
$password = '';
$dbname = 'e_recruitment';
$conn = new mysqli($host, $user, $password, $dbname);if ($_SERVER['REQUEST_METHOD'] == 'POST') {
```

```
$applicant_id = $_SESSION['UserID'];
$job_id = $_POST['job_id'];
$resume = $_POST['resume'];
$cover_letter = $_POST['cover_letter'];

$sql = "INSERT INTO Applications (ApplicantID, JobID, Resume,CoverLetter) VALUES (?, ?, ?, ?)";
$stmt = $conn->prepare($sql);
$stmt->bind_param("iiss", $applicant_id, $job_id, $resume,
$cover_letter);

if ($stmt->execute()) {
echo "Application submitted successfully!";header("Location: jobs.html");
} else {
echo "Error: " . $stmt->error;
}
$stmt->close();
$conn->close(); exit();
}

$job_id = $_GET['job_id'];
?>

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Apply for Job</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
<h2>Apply for Job ID: <?php echo $job_id; ?></h2>
<form method="POST">
<input type="hidden" name="job_id" value="<?php echo $job_id; ?>">
<label for="resume">Resume:</label>
<textarea name="resume" required></textarea><br>
<label for="cover_letter">Cover Letter:</label>
<textarea name="cover_letter"></textarea><br>
<button type="submit">Submit Application</button>
</form>
</body>
</html>
```

Managing Applications

```
<!-- applications.php -->
<?php session_start();
if ($_SESSION['Role'] !== 'Admin') {die("Access denied!");
}

$host = 'localhost';
$user = 'root';
$password = '';
$dbname = 'e_recruitment';

$conn = new mysqli($host, $user, $password, $dbname);
```

```
$sql = "
SELECT a.ApplicationID, u.Username AS Applicant, j.JobTitle, a.Resume,a.CoverLetter
FROM Applications a
JOIN Users u ON a.ApplicantID = u.UserIDJOIN JobPostings j ON a.JobID = j.JobID
";
$result = $conn->query($sql);
?>

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Applications</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
<h2>All Applications</h2>
<table border="1">
<thead>
<tr>
<th>Application ID</th>
<th>Applicant</th>
<th>Job Title</th>
<th>Resume</th>
<th>Cover Letter</th>
</tr>
</thead>
<tbody>
<?php
while ($app = $result->fetch_assoc()) {
echo "<tr>";
echo "<td>{$app['ApplicationID']}</td>";echo "<td>{$app['Applicant']}</td>"; echo
"<td>{$app['JobTitle']}</td>"; echo "<td>{$app['Resume']}</td>";
echo "<td>{$app['CoverLetter']}</td>";echo "</tr>";
}
$conn->close();
?>
</tbody>
</table>
</body>
</html>
```

Logout Feature

To provide a way for users to log out of the system:

```
php
Copy code
<!-- logout.php -->
<?php session_start(); session_destroy();
header("Location: login.html");
?>
```

Client-Side Validation with JavaScript

Enhanced Registration Form with Validation

```
html
Copy code
```

```
<!-- register.html -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Register</title>
<link rel="stylesheet" href="styles.css">
<script>
function validateRegistrationForm() {const username =
document.forms["registrationForm"]["username"].value;const password =
document.forms["registrationForm"]["password"].value;
const email = document.forms["registrationForm"]["email"].value;const role =
document.forms["registrationForm"]["role"].value;

if (username === "" || password === "" || email === "" || role
=== "") {
alert("All fields are required.");
return false;
}

if (password.length < 6) {
alert("Password must be at least 6 characters long.");return false;
}

const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;if (!emailRegex.test(email)) {
alert("Please enter a valid email address.");return false;
}

return true;
}
</script>
</head>
<body>
<h2>User Registration</h2>
<form name="registrationForm" action="register.php" method="POST"onsubmit="return
validateRegistrationForm()">
<label for="username">Username:</label>
<input type="text" name="username" required><br>
<label for="password">Password:</label>
<input type="password" name="password" required><br>
<label for="email">Email:</label>
<input type="email" name="email" required><br>
<label for="role">Role:</label>
<select name="role" required>
<option value="Admin">Admin</option>
<option value="Employer">Employer</option>
<option value="Applicant">Applicant</option>
</select><br>
<button type="submit">Register</button>
</form>
</body>
</html>
```

Dashboard for User Roles

A dashboard is tailored to the user's role (Admin, Employer, Applicant) after login.

```
php
Copy code
<!-- dashboard.php -->
<?php session_start();
```

```
if (!isset($_SESSION['UserID'])) { header("Location: login.html");exit();
}
$role = $_SESSION['Role'];
$username = $_SESSION['Username'];
?>

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Dashboard</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
<h2>Welcome, <?php echo $username; ?> (<?php echo $role; ?>)</h2>
<nav>
<ul>
<?php if ($role === 'Admin'): ?>
<li><a href="applications.php">Manage Applications</a></li>
<?php elseif ($role === 'Employer'): ?>
<li><a href="post_job.html">Post a Job</a></li>
<li><a href="jobs.html">View Jobs</a></li>
<?php elseif ($role === 'Applicant'): ?>
<li><a href="jobs.html">View Jobs</a></li>
<li><a href="applications.php">My Applications</a></li>
<?php endif; ?>
<li><a href="logout.php">Logout</a></li>
</ul>
</nav>
</body>
</html>
```

Improved Styles with CSS
styles.css

```
css
Copy code
/* General Styles */body {
font-family: Arial, sans-serif;margin: 0;
padding: 0;
background-color: #f4f4f4;
}

h2 {
text-align: center;margin-top: 20px; color: #333;
}

/* Navigation Styles */nav ul {
list-style: none;
padding: 0; display: flex;
justify-content: center; background-color: #007BFF;margin: 0;
}

nav ul li {
margin: 0 15px;
}

nav ul li a {
```

```
text-decoration: none;color: white;
font-weight: bold; padding: 10px 15px;border-radius: 5px;
}

nav ul li a:hover { background-color: #0056b3;
}

/* Form Styles */form {
max-width: 400px; margin: 20px auto; padding: 20px; background: #fff; border-radius:
5px;
box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

form label {
display: block; margin-bottom: 5px;font-weight: bold;
}

form input, form select, form textarea, form button {width: 100%;
margin-bottom: 10px;padding: 10px;
border: 1px solid #ccc;border-radius: 5px;
}

form button {
background: #007BFF;color: white; border: none; cursor: pointer;
}

form button:hover { background: #0056b3;
}
/* Table Styles */table {
width: 90%; margin: 20px auto;
border-collapse: collapse;background: white;
}

table th, table td {padding: 10px;
border: 1px solid #ddd;text-align: left;
}

table th {
background: #007BFF;color: white;
}
```

View My Applications

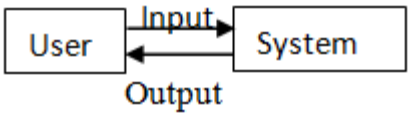
This page displays applications submitted by an applicant.

```
php
Copy code
<!-- my_applications.php -->
<?php session_start();
if ($_SESSION['Role'] !== 'Applicant') {die("Access denied!");
}

$host = 'localhost';
$user = 'root';
$password = '';
$dbname = 'e_recruitment';

$conn = new mysqli($host, $user, $password, $dbname);

$applicant_id = $_SESSION['UserID'];
$sql = "
```

```
SELECT a.ApplicationID, j.JobTitle, a.Resume, a.CoverLetter, a.StatusFROM
Applications a
JOIN JobPostings j ON a.JobID = j.JobIDWHERE a.ApplicantID = ?
";
$stmt = $conn->prepare($sql);
$stmt->bind_param("i", $applicant_id);
$stmt->execute();
$result = $stmt->get_result();
?>
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>My Applications</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
<h2>My Applications</h2>
<table>
<thead>
<tr>
<th>Application ID</th>
<th>Job Title</th>
<th>Resume</th>
<th>Cover Letter</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<?php
while ($app = $result->fetch_assoc()) {echo "<tr>";
echo "<td>{$app['ApplicationID']}</td>";echo   "<td>{$app['JobTitle']}</td>"; echo
"<td>{$app['Resume']}</td>";
echo "<td>{$app['CoverLetter']}</td>";echo   "<td>{$app['Status']}</td>"; echo "</tr>";
}
$stmt->close();
$conn->close();
?>
</tbody>
</table>
</body>
</html>
```

8. Testing

Testing a program involves executing the program with a set of test inputs and observing if the program behaves as expected. If the program fails to behave as expected. If the program fails to behave as expected, then the input data and the conditions under which it fails are noted for later debugging and error correction. A highly simplified view of program testingis schematically shown in the following figure.



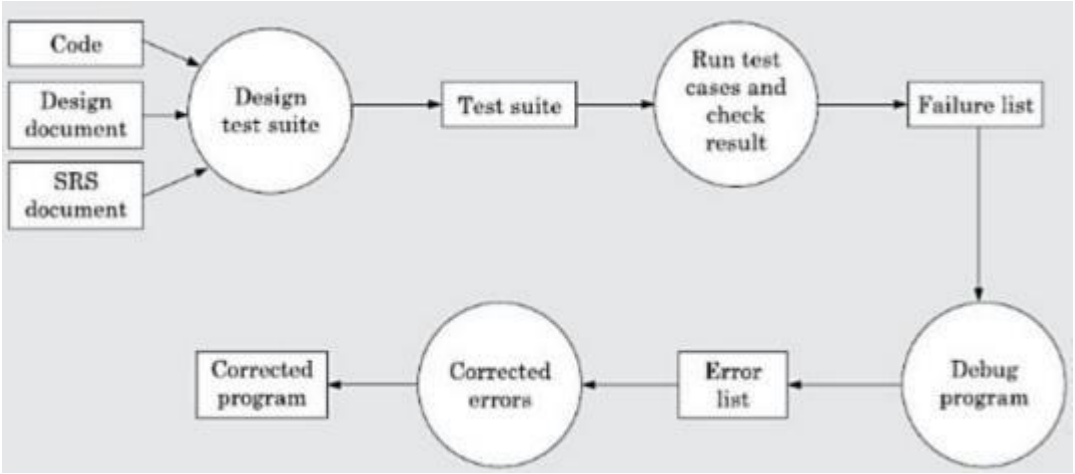
a. Testing Activities

- **Test suite design:** The set of test cases using which a program is to be tested is designed possibly using several test case design techniques.

- Running test cases and checking the results to detect failures: Each test case is run and the results are compared with the expected results. A mismatch between the actual result and expected results indicates a failure. The test cases for which the system fails are noted down for later debugging.

- **Locate error:** In this activity, the failure symptoms are analysed to locate the errors. For each failure observed during the previous activity, the statements that are in error are identified.

- **Error correction:** After the error is located during debugging, the code is appropriately changed to correct the error. The testing activities have been shown schematically in figure given below:



A software product is normally tested in three levels or stages:

- Unit testing
- Integration testing
- System testing

Unit Testing

During unit testing, the individual functions (or units) of a program are tested.

Test Suite Design

A Test Suite is a collection of test cases designed to validate the system's functionality,performance, and usability. Below is the structured design of a Test Suite for the E- Recruitment System.

Due to shortage of time I have designed test suite for only one function/unit toillustrate how to perform unit testing.

Running test cases and checking the results to detect failure I have run each test case corresponding to the table given above and the results are compared with the expected results and the failures are noted down as shown in the following table:

Test Cases

User Management

Objective: Verify registration, login, and role-specific access.

Database Table Before Registration

UserID	Username	Password (Hashed)	Email	Role
1	jane_admin	*hashed_password_1*	jane@admin.com	Admin

Database Table After Registration

UserID	Username	Password (Hashed)	Email	Role
1	jane_admin	*hashed_password_1*	jane@admin.com	Admin
2	john_doe	*hashed_password_2*	john@example.com	Applicant

Locate Error and Perform Error Correction

I have analyzed the failure symptoms to locate the errors and changed the code appropriately to correctthe errors. After performing the changes I have run the test cases again to verify whether the errors are removed or not. The final result is shown in the following table:

Test ID	Test Scenario	Expected Result	Actual Result
TC-001	Login with correct username/password.	Redirect to dashboard.	✓
TC-002	Login with incorrect password.	Display error: **"Invalid credentials."**	✓
TC-003	Login with non-existent username.	Display error: **"User not found."**	✓
TC-004	Empty username or password.	Display error: **"Cannot be empty."**	✓

ii) Integration and System Testing:

After testing all the units individually, the units are slowly integrated and tested after each step ofintegration (integration testing). Finally, the fully integrated system is tested (system testing).

Integration Testing: There are several approaches to perform Integration Testing. One of them is Big-bang approach. In this approach, all the modules making up a system are integrated in a single step. In simple words, all the unit tested modules of the system are simply linked together and tested. However,this technique can meaningfully be used only for very small systems. Since the system I have developedis very small I can use Big-bang approach. But since my system is not fully complete it is not possible to perform Integration testing at present.

iii) System Testing

There are essentially three main kinds of system testing depending on who carries out testing:

- **Alpha Testing:** Alpha testing refers to the system testing carried out by the test team within thedeveloping organization.

- **Beta Testing:** Beta testing is the system testing performed by a select group of friendly customers.

- **Acceptance Testing:** Acceptance testing is the system testing performed by the customer todetermine whether to accept the delivery of the system. Since my system is not fully complete it is not possible to perform system testing at present. If it wascomplete then System Testing could have been performed by me or by my friends.

9. Maintenance

Software maintenance denotes any changes made to a software product after it has been deliveredto the customer. Maintenance is inevitable for almost any kind of product. However, most productsneed maintenance due to the wear and tear caused by use. On the other hand, software products do not need maintenance on this count, but need maintenance to correct errors, enhance features,port to new platforms, etc. Maintenance is not required for the software that I have developed because this software has been developed only for understanding the engineering way to develop asoftware and it is not for any customer.

10. Conclusion :

In order to develop this software I followed and understood the engineering way to develop a software. While developing the software I faced a lot of challenges and practically implemented thetheoretical concepts that I studied in the subject Software Engineering. It will definitely help me in near future if I want

to work in any software development company

11. References

[1]R.S. Pressman, "Software Engineering: A Practitioner‘s Approach", 7th Edition, McGrawHill, 2009

[2]P. Jalote, "An Integrated Approach to Software Engineering", 2nd Edition, Narosa

[3]R. Mall, "Fundamentals of Software Engineering", 2nd Edition, Prentice-Hall of India, 2004

[4]R. Elmasri, S.B. Navathe, "Fundamentals of Database Systems", 6th Edition, Pearson Education,2010