Software Design & Architecture Document

Project Title: Job Portal

Instructor: Muhammad Yaseen

Student Information			
Sr#	**Student Name**	**Sap ID**	**Email ID**
1	Uzair Hassan	48525	48525@students.riphah.edu.pk
2	Daniyal Wajid	48528	48528@students.riphah.edu.pk

Contents

1. Modeling and Analysis .....
1.1 Overview: .....
1.2 Abstract: .....
1.3 Functional Requirements: .....
2. Modeling and Analysis .....
2.1 Use Case Diagram .....
2.2 Class Diagram: .....
2.3 Sequence Diagrams: .....
2.4 Activity Diagrams: .....
2.5 Data Flow Diagrams .....
2.6 Communication Diagram .....
2.7 Object Diagram .....
2.8 Componenet Diagram .....
2.9 Deployement Diagram .....
3. Architecture Design: .....
3.1 Horizontal and Vertical Partitioning .....
3.2 Architecture Style: .....
4. Code Design .....
4.1 Class diagram to Code Mapping .....
4.2 Design Principles .....
4.3 Design Patterns .....
4.3.1 Creational Design Pattern .....
4.3.2 Structural Design Patterns .....
4.3.3 Behavioral Design Patterns .....
5. Protyping .....

1. Modeling and Analysis

1.1 Overview

Job seekers and companies can easily and effectively interact with **RIZQ**, a cutting-edge online job platform. The **Rizq Job Portal Project's** objective includes creating a thorough web platform with the goal of effectively matching job seekers and businesses.

It has features including the ability to register users and create profiles, a powerful job search engine, an employer portal for managing applications and job postings, a secure backend database, and communication tools for seamless user engagement.

1.2 Abstract

The **Rizq Job Portal Project** is a state-of-the-art online platform designed to facilitate seamless interaction between job seekers and companies. The platform's primary objective is to create an efficient and effective solution for matching job seekers with businesses.

Key features of the platform include:

- **User Registration and Profile Creation**: Users can quickly create accounts and build profiles.
- **Advanced Job Search Engine**: Enables job seekers to find relevant positions efficiently.
- **Employer Portal**: Provides employers with tools to manage applications and job postings.
- **Secure Backend Database**: Ensures the safety and privacy of job seeker profiles and resumes.
- **Communication Tools**: Streamlines interaction between companies and job seekers.

The project also includes:

- Comprehensive documentation for easy setup and usage.
- Support systems for both users and technical troubleshooting.

While direct recruitment services, offline initiatives, and integrations outside the platform's defined scope are not included, the initiative focuses on prioritizing:

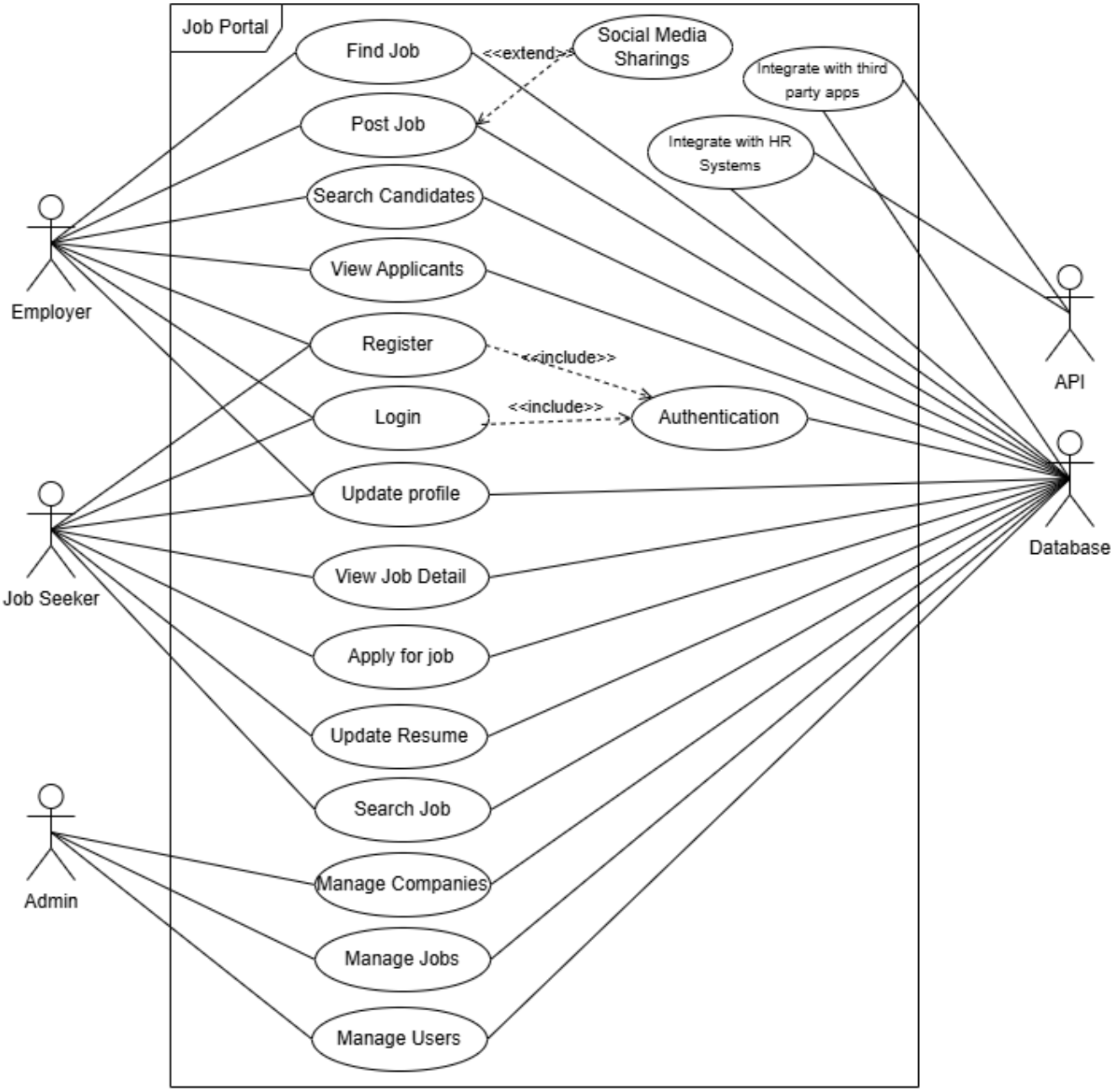
- **Data Security**: Ensuring user information is protected.
- **Privacy**: Adhering to strict regulatory compliance.
- **User-Friendly Design**: Providing an accessible and efficient experience for all users.

1.3 Functional Requirements

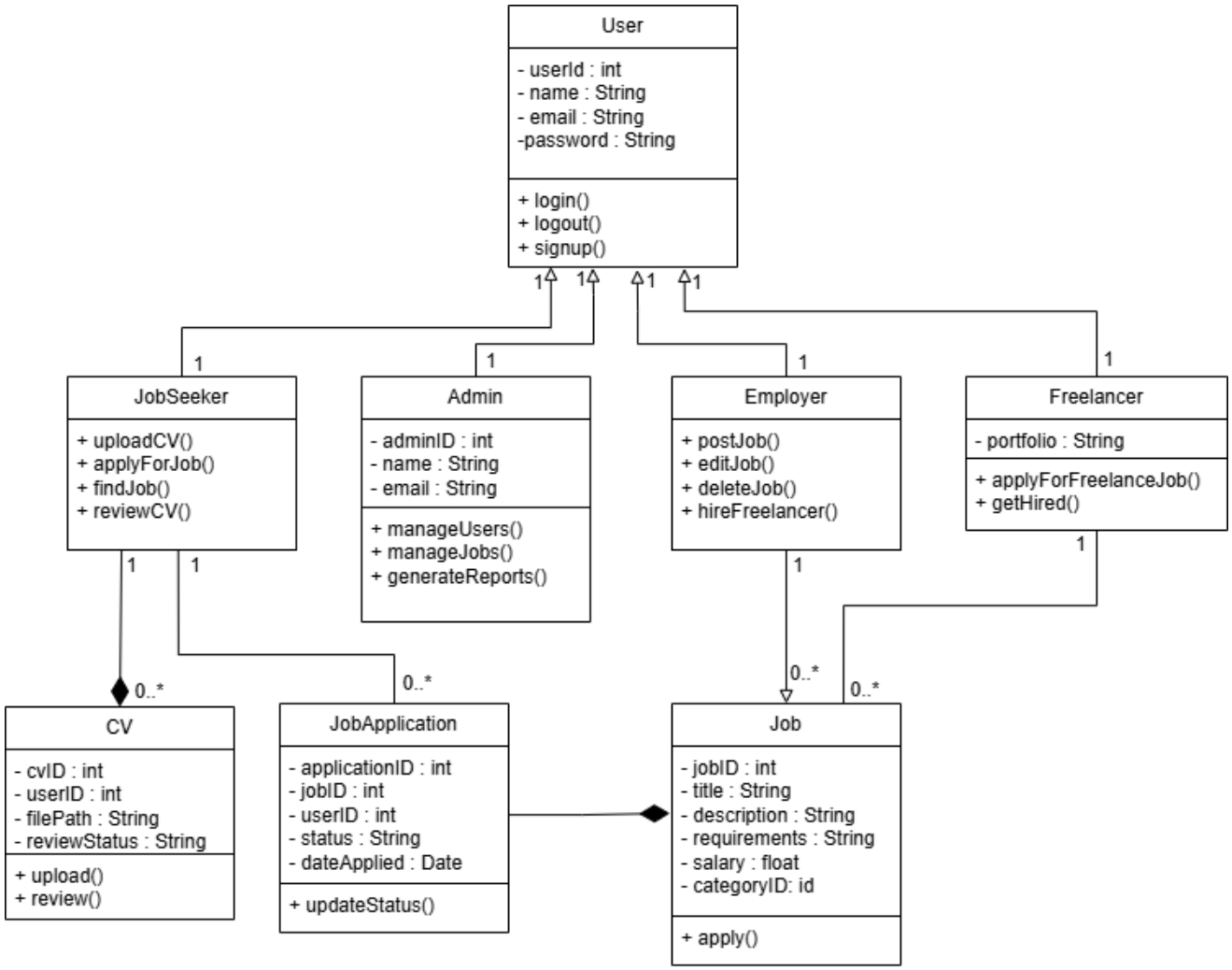
Functional Req. ID #	Function Name	Function Requirement Description
FR1	Login	Registration and authentication
FR2	Profile management	Registered users should have the ability to edit and update their profiles
FR3	Job Posting	Employers should be able to post jobs
FR4	Job Search	Users should be able to search for jobs based on different criteria's
FR5	Analytics	The platform provides analytics and reporting features for employers and administrators
FR6	Notifications	Users should receive notifications for new job postings, application status updates, messages, etc.
FR7	Communication	The platform should facilitate communication between job seekers and employers
FR8	Employee dashboard	Employers should have access to a dashboard
FR9	Job Alerts	Users should be able to set up job alerts based on their search criteria
FR10	Helpdesk	Section or helpdesk where users can find answers to frequently asked questions (FAQs)
FR11	API integration	Allowing third-party developers to build custom applications or integrate with existing HR systems.
FR12	Feedback	Users should be able to provide feedback and reviews

2. Modeling and Analysis

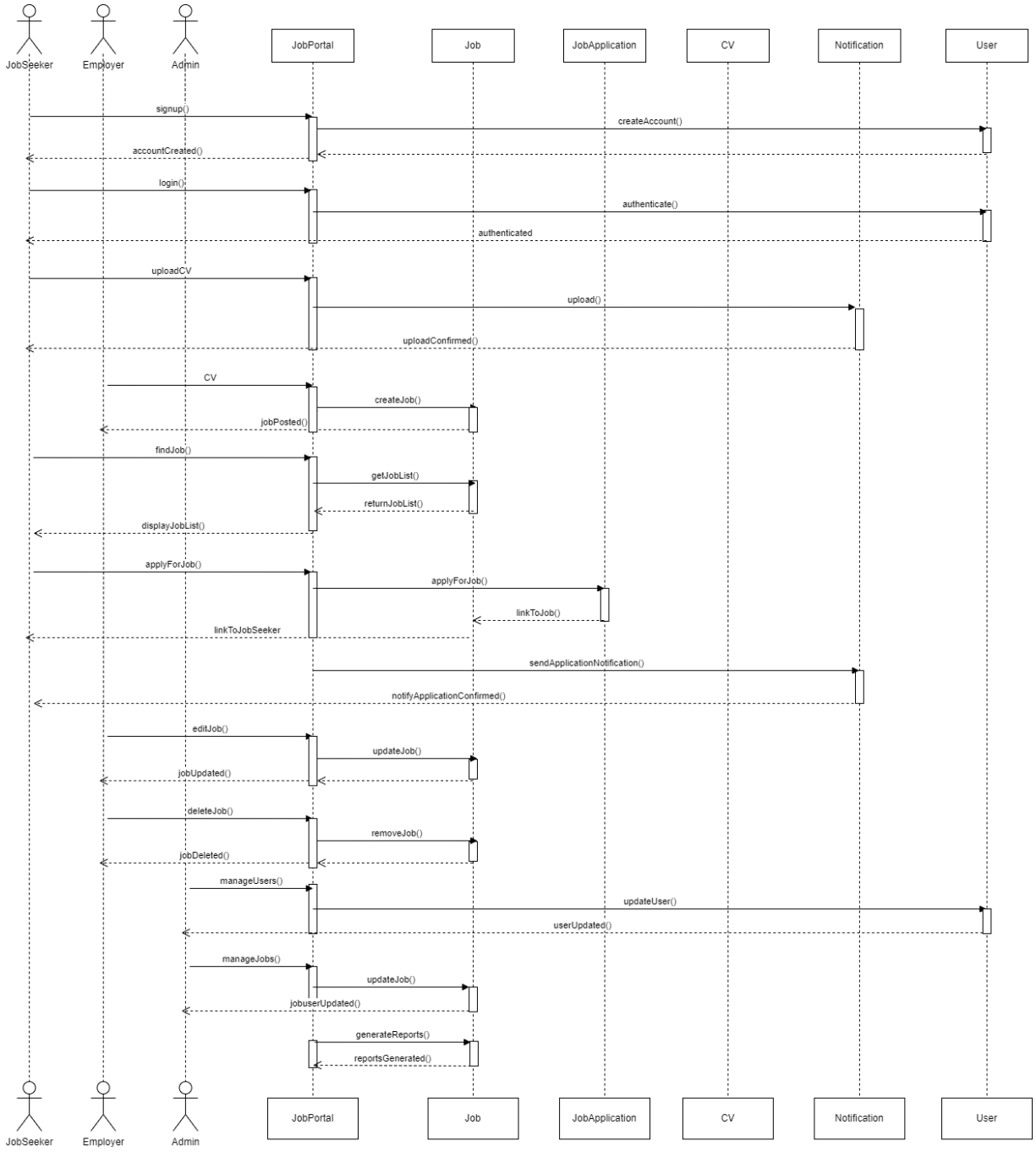
2.1 Use Case diagram



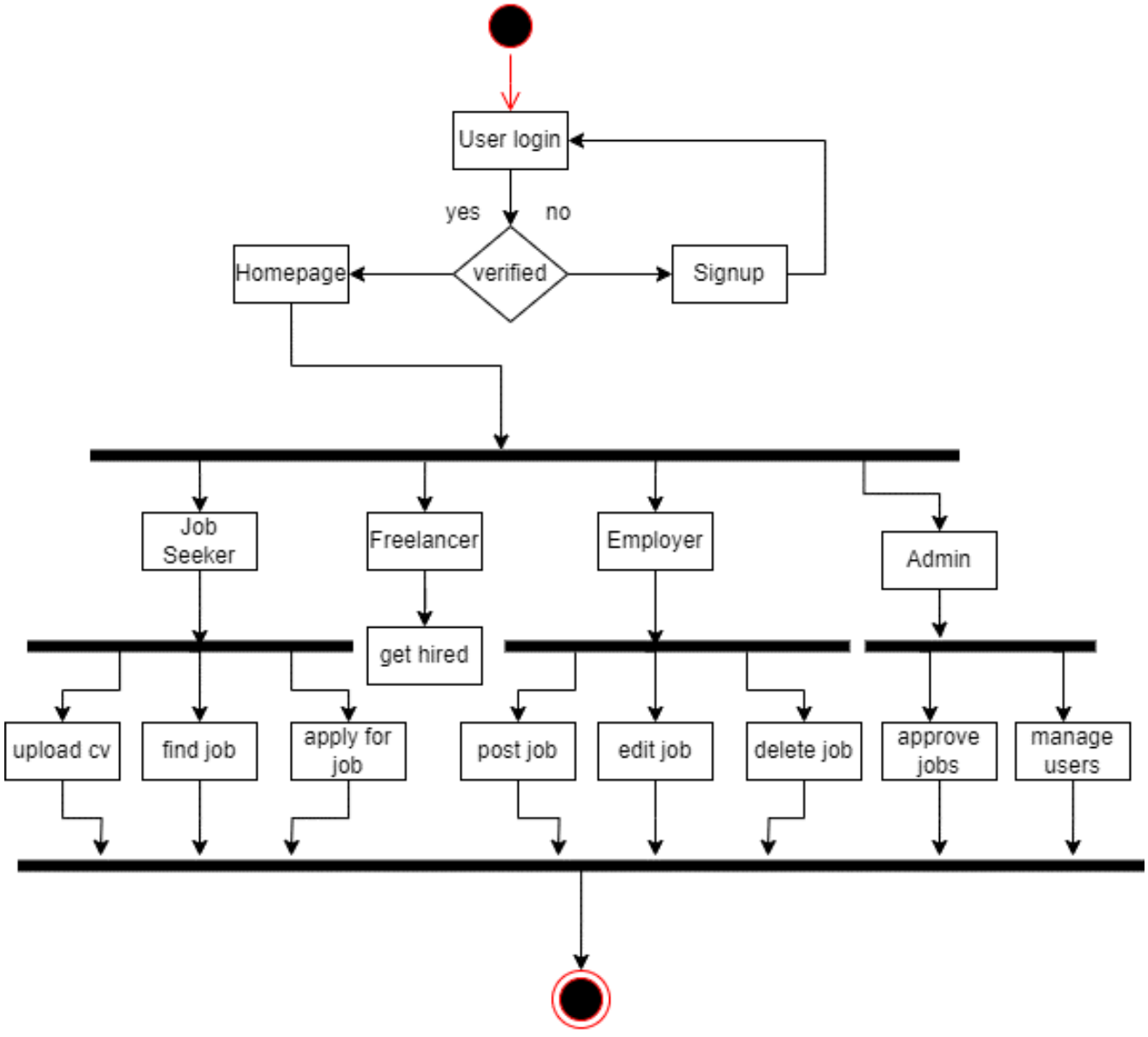
2.2 Class Diagram



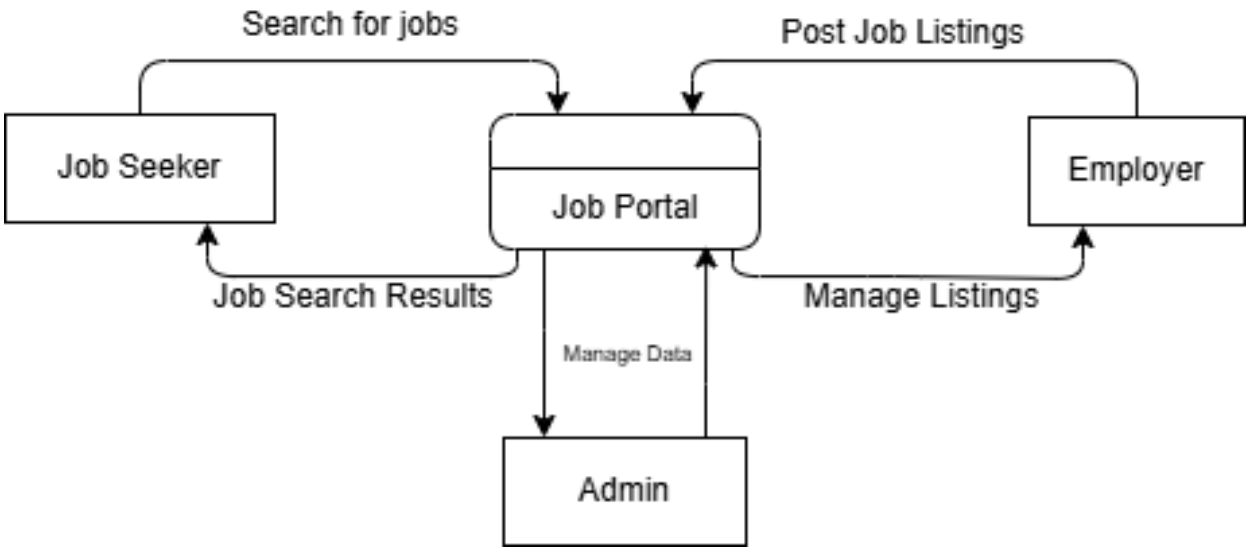
2.3 Sequence Diagram



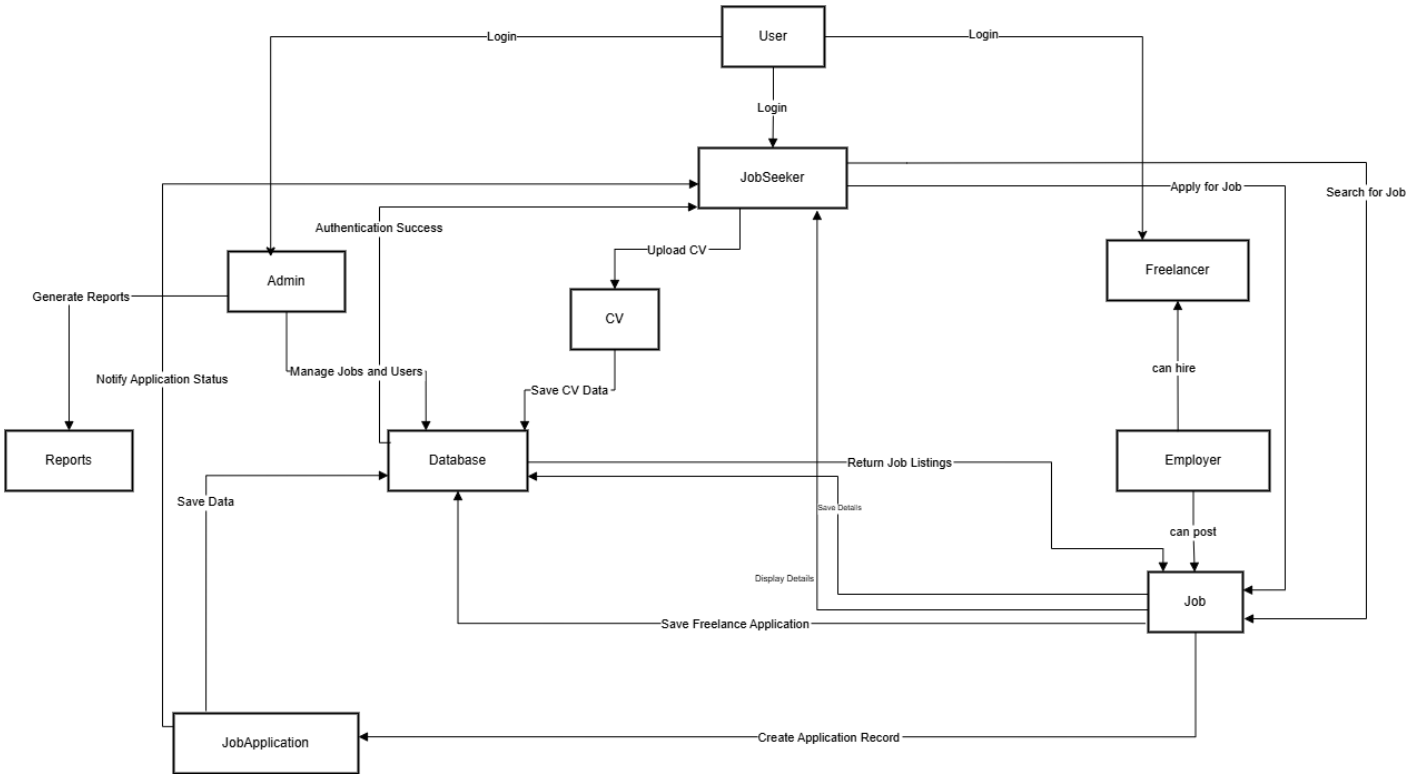
2.4 Activity Diagram



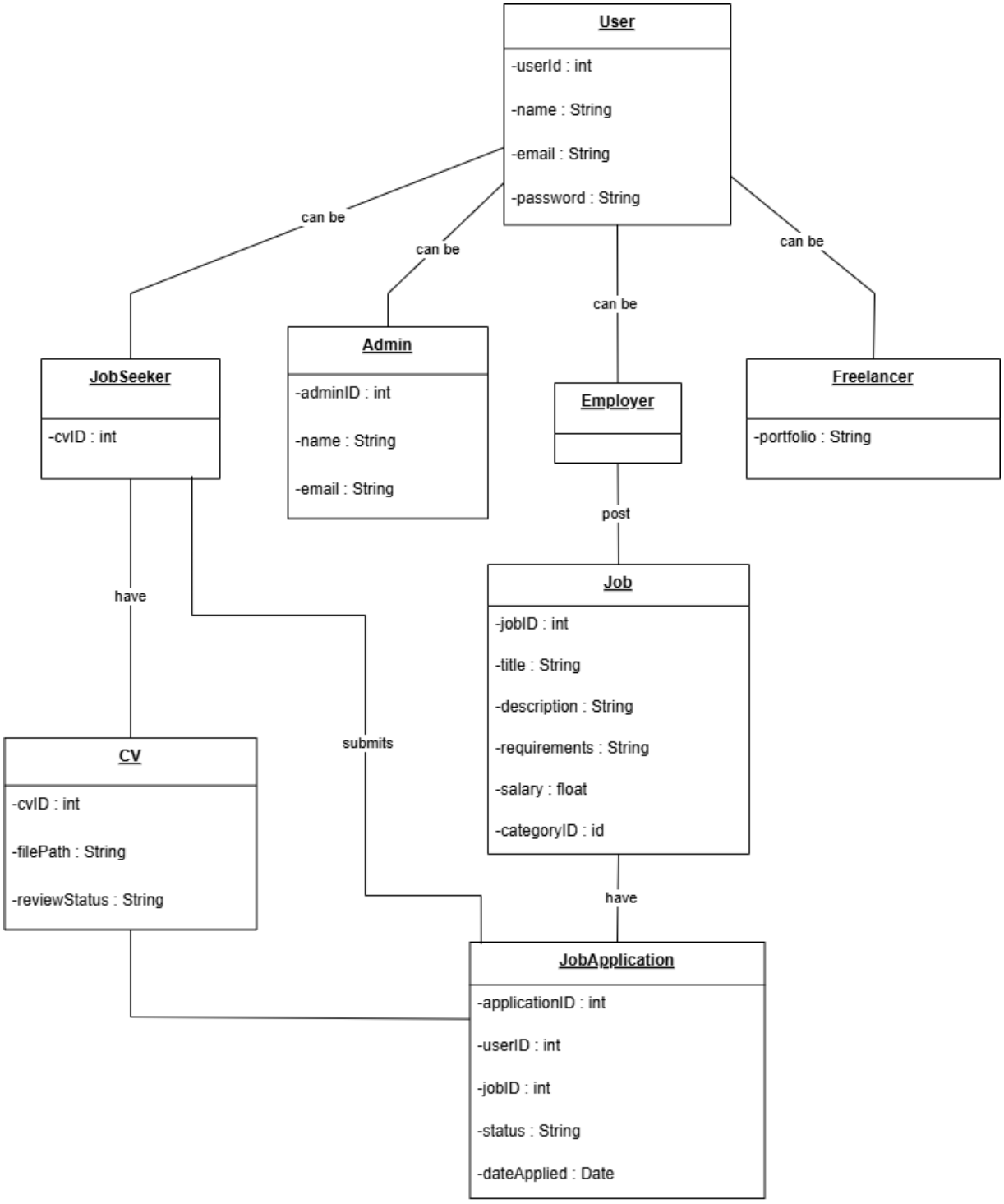
2.5 Data Flow Diagram



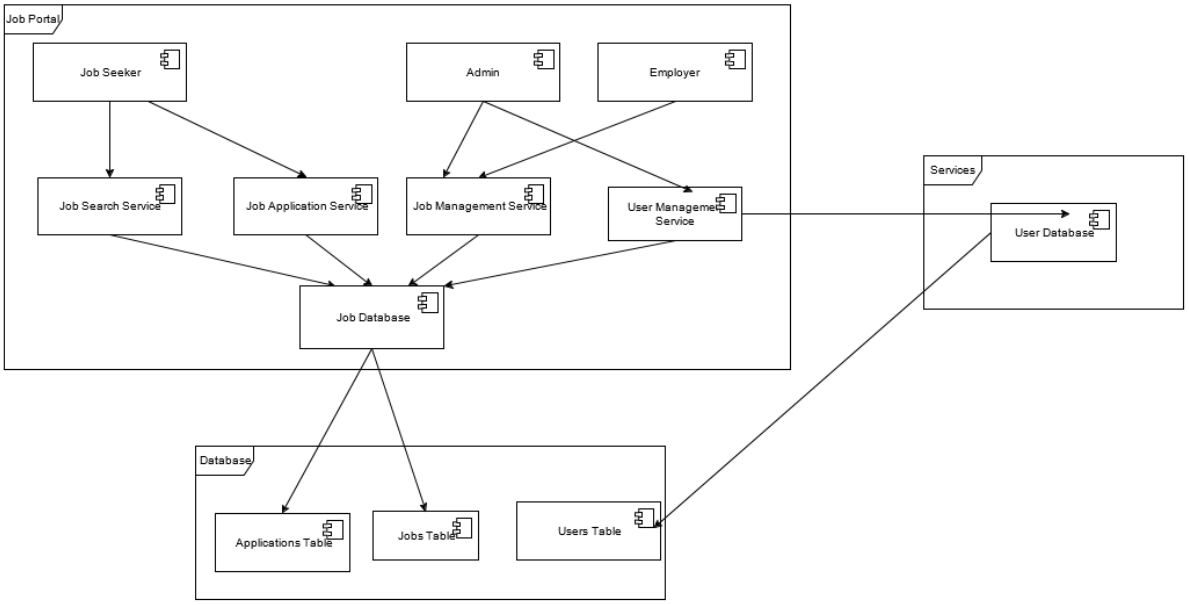
2.6 Communication Diagram



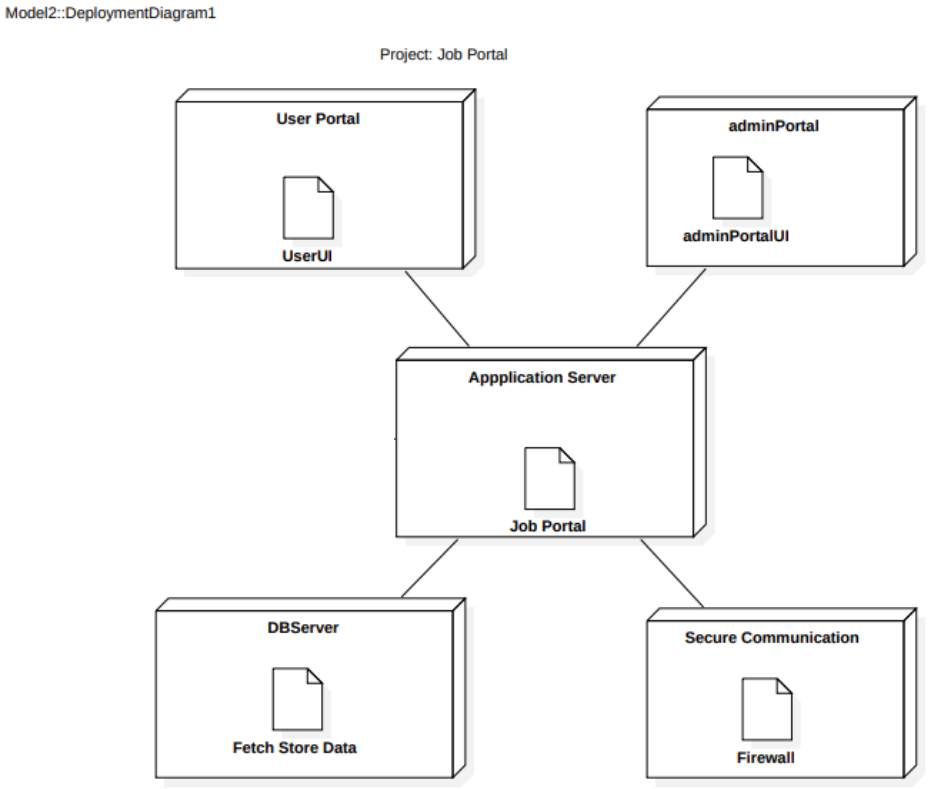
2.7 Object Diagram



2.8 Component Diagram



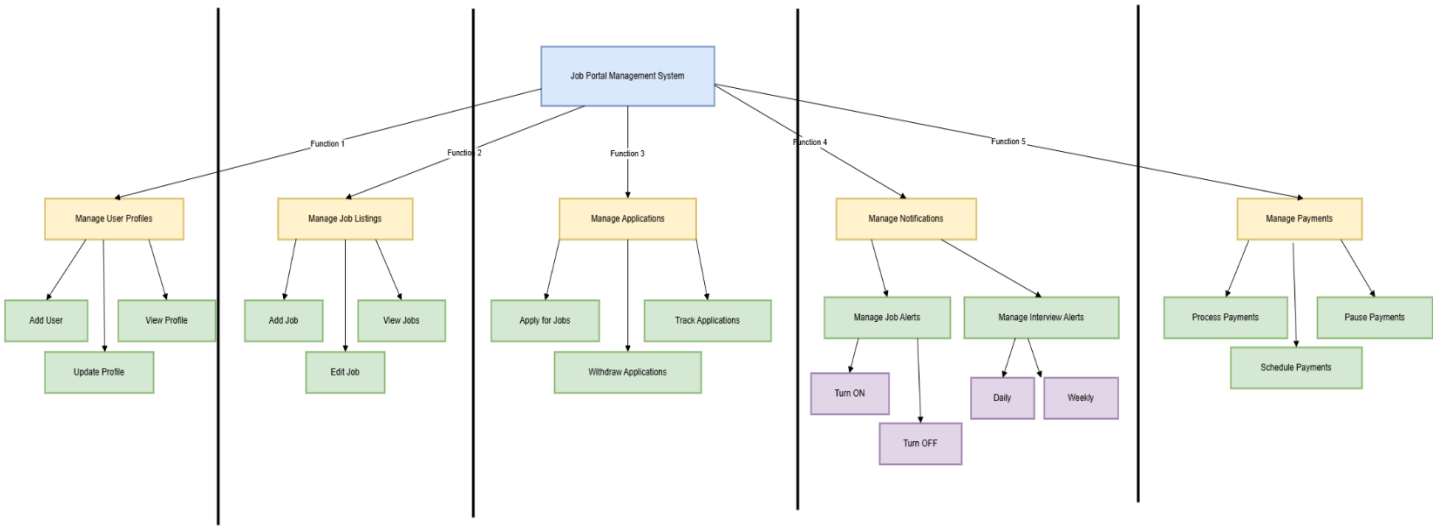
2.9 Deployment Diagram



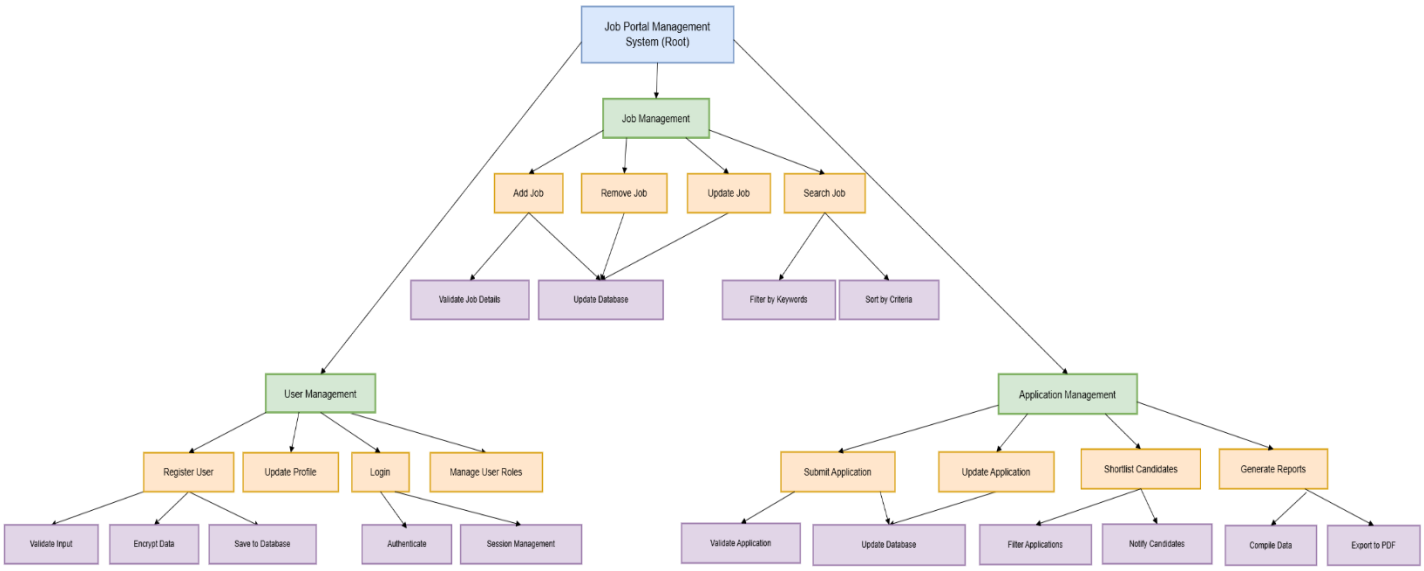
3. Architecture Design

3.1 Horizontal and Vertical Partitioning:

Horizontal Partitioning:

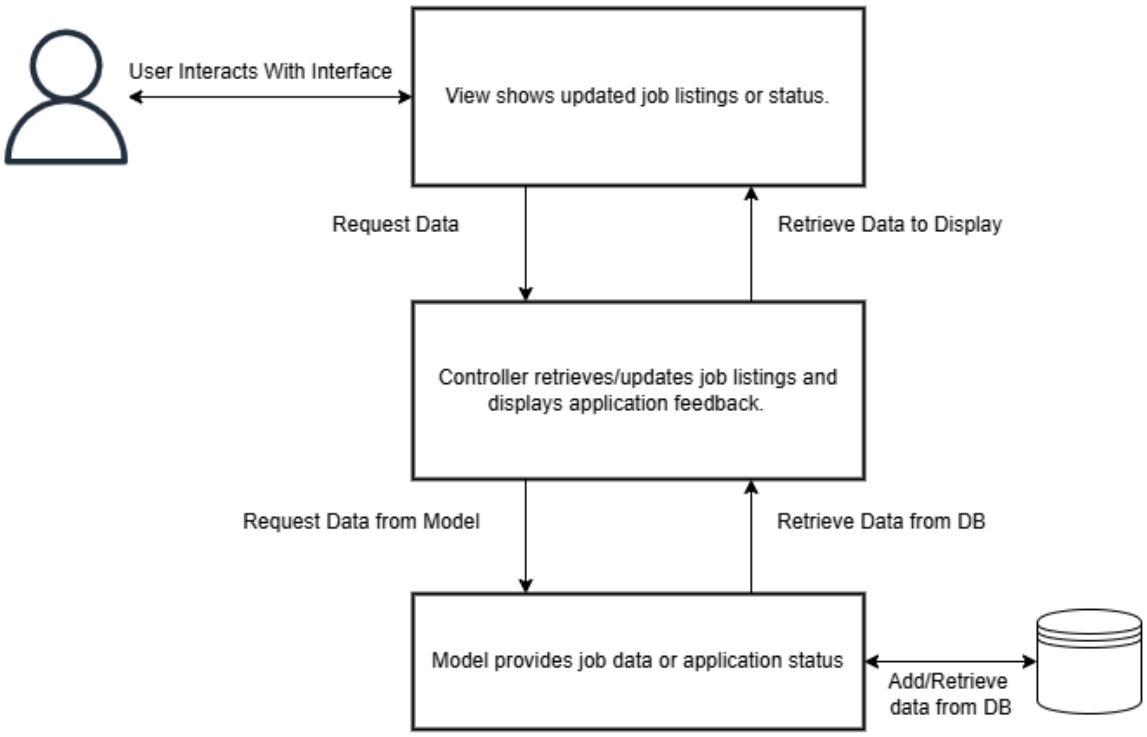


Vertical Partitioning:



3.2 Architecture Style:

MVC:



MVC architecture is used in a Job Portal to separate concerns: the **Model** handles data (job listings, user applications), the **View** displays the user interface, and the **Controller** processes user input and updates the View. This separation makes the portal easier to maintain, scale, and update while enhancing the user experience.

4. Code Design:

4.1 Class diagram to Code Mapping

```
import java.util.Date;
class User {
    protected int userId;
    protected String name;
    protected String email;
    protected String password;
    public void login() {
        System.out.println(name + " logged in.");
    }
    public void logout() {
        System.out.println(name + " logged out.");
    }
    public void signup() {
        System.out.println(name + " signed up.");
    }
}
class Job {
    private int jobID;
    private String title;
    private String description;
    private String requirements;
    private float salary;
    private int categoryID;
    public Job(int jobID, String title, String description, String requirements, float salary, int categoryID) {
        this.jobID = jobID;
        this.title = title;
        this.description = description;
        this.requirements = requirements;
        this.salary = salary;
        this.categoryID = categoryID;
    }
    public int getJobID() {
        return jobID;
    }
    public String getTitle() {
        return title;
    }
    public String getDescription() {
        return description;
    }
    public String getRequirements() {
        return requirements;
    }
    public float getSalary() {
        return salary;
    }
    public int getCategoryID() {
        return categoryID;
    }
    public void apply() {
        System.out.println("Job application for: " + title);
    }}
```

```
class JobSeeker extends User {
    private CV cv;
    public void uploadCV() {
        System.out.println("CV uploaded by " + name);
    }
    public void applyForJob(Job job) {
        JobApplication application = new JobApplication(this.userId, job.getJobID());
        System.out.println(name + " applied for job: " + job.getTitle());
    }
    public void findJob() {
        System.out.println(name + " is looking for a job.");
    }
    public void reviewCV() {
        System.out.println(name + " is reviewing their CV.");
    }}
class Admin extends User {
    private int adminID;
    public void manageUsers() {
        System.out.println("Admin managing users.");
    }
    public void manageJobs() {
        System.out.println("Admin managing jobs.");
    }
    public void generateReports() {
        System.out.println("Admin generating reports.");
    }}
class Employer extends User {
    public void postJob(Job job) {
        System.out.println("Employer posted job: " + job.getTitle());
    }
    public void editJob(Job job) {
        System.out.println("Employer edited job: " + job.getTitle());
    }
    public void deleteJob(Job job) {
        System.out.println("Employer deleted job: " + job.getTitle());
    }
    public void hireFreelancer(Freelancer freelancer) {
        System.out.println("Employer hired freelancer: " + freelancer.name);
    }}
class Freelancer extends User {
    private String portfolio;
    public void applyForFreelanceJob(Job job) {
        System.out.println(name + " applied for freelance job: " + job.getTitle());
    }
    public void getHired() {
        System.out.println(name + " got hired for a freelance job.");
    }}
class CV {
    private int cvID;
    private int userID;
    private String filePath;
    private String reviewStatus;
    public void upload() {
        System.out.println("CV uploaded at: " + filePath);
    }
    public void review() {
```

```
        System.out.println("CV is under review.");
    }
}
class JobApplication {
    private int applicationID;
    private int jobID;
    private int userID;
    private String status;
    private Date dateApplied;
    public JobApplication(int userID, int jobID) {
        this.userID = userID;
        this.jobID = jobID;
        this.status = "Applied";
        this.dateApplied = new Date();
        this.applicationID = (int) (Math.random() * 1000);
    }
    public void updateStatus(String newStatus) {
        this.status = newStatus;
        System.out.println("Application status updated to: " + status);
    }
}
public class Main {
    public static void main(String[] args) {
        JobSeeker jobSeeker = new JobSeeker();
        jobSeeker.name = "Daniyal Wajid";
        jobSeeker.signup();
        jobSeeker.uploadCV();

        Admin admin = new Admin();
        admin.name = "Admin User";
        admin.manageJobs();

        Employer employer = new Employer();
        employer.name = "Company Inc.";
        Job job = new Job(1, "Software Developer", "Develop applications", "Java, Spring", 60000, 101);
        employer.postJob(job);

        Freelancer freelancer = new Freelancer();
        freelancer.name = "Freelance Expert";
        freelancer.applyForFreelanceJob(job);
    }
}
```

Output:

```
Daniyal Wajid signed up.
CV uploaded by Daniyal Wajid
Admin managing jobs.
Employer posted job: Software Developer
Freelance Expert applied for freelance job: Software Developer
```

4.2 Design Principles:

4.2.1 Single Responsibility Principle:

Current Issues:

- User class: It handles multiple responsibilities, such as login, logout, and signup. These responsibilities could be separated into different classes.
- JobSeeker class: It handles both job-seeking actions (like applying for a job) and CV-related actions (like uploading a CV). These responsibilities can be split into separate classes.

Refactored Code:

We'll break down responsibilities into smaller, more focused classes. Here's how we can refactor the code to apply SRP:

- Create a UserAuthentication class to handle login, logout, and signup.
- Create a CVManager class to handle CV uploads and reviews.
- Separate JobSeeker's responsibility for applying for jobs from CV management.

Fixed Code:

```
import java.util.Date;
class User {
    protected int userId;
    protected String name;
    protected String email;
    protected String password;
}
class UserAuthentication {
    public void login(User user) {
        System.out.println(user.name + " logged in.");
    }
    public void logout(User user) {
        System.out.println(user.name + " logged out.");
    }
    public void signup(User user) {
        System.out.println(user.name + " signed up.");
    }
}
class Job {
    private int jobID;
    private String title;
    private String description;
    private String requirements;
    private float salary;
    private int categoryID;
    public Job(int jobID, String title, String description, String requirements, float salary, int categoryID) {
        this.jobID = jobID;
        this.title = title;
        this.description = description;
        this.requirements = requirements;
        this.salary = salary;
        this.categoryID = categoryID;
    }
    public int getJobID() {
        return jobID;
    }
```

```
    public String getTitle() {
        return title;
    }
    public String getDescription() {
        return description;
    }
    public String getRequirements() {
        return requirements;
    }
    public float getSalary() {
        return salary;
    }
    public int getCategoryID() {
        return categoryID;
    }
    public void apply() {
        System.out.println("Job application for: " + title);
    }
}
class JobSeeker extends User {
    private CVManager cvManager;
    public JobSeeker() {
        this.cvManager = new CVManager();
    }
    public void applyForJob(Job job) {
        JobApplication application = new JobApplication(this.userId, job.getJobID());
        System.out.println(name + " applied for job: " + job.getTitle());
    }
    public void findJob() {
        System.out.println(name + " is looking for a job.");
    }
    public void uploadCV(CV cv) {
        cvManager.upload(cv);
    }
    public void reviewCV(CV cv) {
        cvManager.review(cv);
    }
}
class CV {
    private int cvID;
    private int userID;
    private String filePath;
    private String reviewStatus;
    public CV(int cvID, int userID, String filePath, String reviewStatus) {
        this.cvID = cvID;
        this.userID = userID;
        this.filePath = filePath;
        this.reviewStatus = reviewStatus;
    }
    public String getFilePath() {
        return filePath;
    }
}
```

```
class CVManager {
    public void upload(CV cv) {
        System.out.println("CV uploaded at: " + cv.getFilePath());
    }
    public void review(CV cv) {
        System.out.println("CV is under review.");
    }
}
class Admin extends User {
    private int adminID;
    public void manageUsers() {
        System.out.println("Admin managing users.");
    }
    public void manageJobs() {
        System.out.println("Admin managing jobs.");
    }
    public void generateReports() {
        System.out.println("Admin generating reports.");
    }
}
class Employer extends User {
    public void postJob(Job job) {
        System.out.println("Employer posted job: " + job.getTitle());
    }
    public void editJob(Job job) {
        System.out.println("Employer edited job: " + job.getTitle());
    }
    public void deleteJob(Job job) {
        System.out.println("Employer deleted job: " + job.getTitle());
    }
    public void hireFreelancer(Freelancer freelancer) {
        System.out.println("Employer hired freelancer: " + freelancer.name);
    }
}
class Freelancer extends User {
    private String portfolio;

    public void applyForFreelanceJob(Job job) {
        System.out.println(name + " applied for freelance job: " + job.getTitle());
    }
    public void getHired() {
        System.out.println(name + " got hired for a freelance job.");
    }
}
class JobApplication {
    private int applicationID;
    private int jobID;
    private int userID;
    private String status;
    private Date dateApplied;
    public JobApplication(int userID, int jobID) {
        this.userID = userID;
        this.jobID = jobID;
        this.status = "Applied";
        this.dateApplied = new Date();
```

```
        this.applicationID = (int) (Math.random() * 1000);
    }
    public void updateStatus(String newStatus) {
        this.status = newStatus;
        System.out.println("Application status updated to: " + status);
    }
}

public class Main1 {
    public static void main(String[] args) {
        User user = new User();
        user.name = "Daniyal Wajid";

        UserAuthentication auth = new UserAuthentication();
        auth.signup(user);

        JobSeeker jobSeeker = new JobSeeker();
        jobSeeker.name = "Daniyal Wajid";
        jobSeeker.uploadCV(new CV(1, jobSeeker.userId, "DB", "Pending"));

        Admin admin = new Admin();
        admin.name = "Admin User";
        admin.manageJobs();

        Employer employer = new Employer();
        employer.name = "Company Inc.";
        Job job = new Job(1, "Software Developer", "Develop applications", "Java, Spring", 60000, 101);
        employer.postJob(job);

        Freelancer freelancer = new Freelancer();
        freelancer.name = "Freelance Expert";
        freelancer.applyForFreelanceJob(job);
    }
}
```

Output:

```
Daniyal Wajid signed up.
CV uploaded at: DB
Admin managing jobs.
Employer posted job: Software Developer
Freelance Expert applied for freelance job: Software Developer
```

4.2.2 Open Closed Principle:

Current Issues:

- **Violation of OCP:** The classes (Employer, Admin, JobSeeker) are tightly coupled to specific implementations, making it hard to extend functionalities without modifying existing code.
- **Lack of Flexibility:** Adding new roles or functionalities requires changes to these classes, reducing adaptability.
- **Difficult to Extend:** Modifying job posting or CV management features for different user types (e.g., Admin, Freelancer) requires altering existing code, making it harder to maintain.

Refactored Code:

- **Introduce a JobPostingService interface** to decouple job posting from user roles.
- **Introduce a CVManagementService interface** to manage CV operations independently.
- **Use Dependency Injection** for flexibility in assigning services like job posting and CV management to different user roles (e.g., Admin, Employer, JobSeeker).

Fixed Code:

```
import java.util.Date;

interface JobPostingService {
   void postJob(Job job);
   void editJob(Job job);
   void deleteJob(Job job);
}

interface CVManagementService {
   void upload(CV cv);
   void review(CV cv);
}

abstract class User {
   protected int userId;
   protected String name;
   protected String email;
   protected String password;

}

class Job {
   private int jobID;
   private String title;
   private String description;
   private String requirements;
```

```
    private float salary;
    private int categoryID;

    public Job(int jobID, String title, String description, String requirements, float salary, int categoryID) {
        this.jobID = jobID;
        this.title = title;
        this.description = description;
        this.requirements = requirements;
        this.salary = salary;
        this.categoryID = categoryID;
    }

    public int getJobID() {
        return this.jobID;
    }

    public String getTitle() {
        return this.title;
    }

    public String getDescription() {
        return this.description;
    }

    public String getRequirements() {
        return this.requirements;
    }

    public float getSalary() {
        return this.salary;
    }

    public int getCategoryID() {
        return this.categoryID;
    }

    public void apply() {
        System.out.println("Job application for: " + title);
    }
}

class CV {
    private int cvID;
    private int userID;
    private String filePath;
    private String reviewStatus;

    public String getFilePath() {
        return this.filePath;
    }

    public void setFilePath(String filePath) {
        this.filePath = filePath;
    }

}
```

```
class JobApplication {
    private int applicationID;
    private int jobID;
    private int userID;
    private String status;
    private Date dateApplied;

    public JobApplication(int userID, int jobID) {
        this.userID = userID;
        this.jobID = jobID;
        this.status = "Applied";
        this.dateApplied = new Date();
        this.applicationID = (int) (Math.random() * 1000);
    }

    public void updateStatus(String newStatus) {
        this.status = newStatus;
        System.out.println("Application status updated to: " + status);
    }
}

class CVManager implements CVManagementService {
    public void upload(CV cv) {
        System.out.println("CV uploaded at: " + cv.getFilePath());
    }

    public void review(CV cv) {
        System.out.println("CV is under review.");
    }
}

class JobSeeker extends User {
    private JobPostingService jobPostingService;
    private CVManager cvManager;

    public JobSeeker(JobPostingService jobPostingService, CVManager cvManager) {
        this.jobPostingService = jobPostingService;
        this.cvManager = cvManager;
    }

    public void applyForJob(Job job) {
        JobApplication application = new JobApplication(this.userId, job.getJobID());
        System.out.println(name + " applied for job: " + job.getTitle());
    }

    public void findJob() {
        System.out.println(name + " is looking for a job.");
    }

    public void uploadCV(CV cv) {
        cvManager.upload(cv);
    }

    public void reviewCV(CV cv) {
        cvManager.review(cv);
```

```
    }
}

class Admin extends User implements JobPostingService, CVManagementService {
    public void postJob(Job job) {
        System.out.println("Admin posted job: " + job.getTitle());
    }

    public void editJob(Job job) {
        System.out.println("Admin edited job: " + job.getTitle());
    }

    public void deleteJob(Job job) {
        System.out.println("Admin deleted job: " + job.getTitle());
    }

    public void upload(CV cv) {
        System.out.println("Admin uploaded CV at: " + cv.getFilePath());
    }

    public void review(CV cv) {
        System.out.println("Admin is reviewing CV.");
    }

    public void manageJobs() {
        System.out.println("Admin is managing jobs.");
    }
}

class Employer extends User implements JobPostingService {
    public void postJob(Job job) {
        System.out.println("Employer posted job: " + job.getTitle());
    }

    public void editJob(Job job) {
        System.out.println("Employer edited job: " + job.getTitle());
    }

    public void deleteJob(Job job) {
        System.out.println("Employer deleted job: " + job.getTitle());
    }
}

class Freelancer extends User {
    private String portfolio;

    public void applyForFreelanceJob(Job job) {
        System.out.println(name + " applied for freelance job: " + job.getTitle());
    }

    public void getHired() {
        System.out.println(name + " got hired for a freelance job.");
    }
}

public class Main2 {
```

```
  public static void main(String[] args) {
    CVManager cvManager = new CVManager();
    JobPostingService jobPostingService = new Employer(); // Assuming Employer posts jobs

    JobSeeker jobSeeker = new JobSeeker(jobPostingService, cvManager);
    jobSeeker.name = "Daniyal Wajid";
    jobSeeker.uploadCV(new CV());

    Admin admin = new Admin();
    admin.name = "Admin User";
    admin.manageJobs();
    admin.postJob(new Job(1, "Software Developer", "Develop applications", "Java, Spring", 60000, 101));

    Employer employer = new Employer();
    employer.name = "Company Inc.";
    employer.postJob(new Job(2, "Project Manager", "Manage projects", "Leadership, Java", 70000, 102));

    Freelancer freelancer = new Freelancer();
    freelancer.name = "Freelance Expert";
    freelancer.applyForFreelanceJob(new Job(3, "Web Developer", "Create websites", "HTML, CSS, JS", 50000,
103));
  }
}
```

Output:

```
CV uploaded at: null
Admin is managing jobs.
Admin posted job: Software Developer
Employer posted job: Project Manager
Freelance Expert applied for freelance job: Web Developer
```

4.2.3 Liskov Substitution Principle:

Current Issues:

- **Violation of LSP:** Employer, Admin, and JobSeeker have specific job posting and CV management behaviors, making them difficult to substitute with subclasses without altering behavior.
- **Tight Coupling:** These classes are tightly coupled with JobPostingService and CVManagementService, making it hard to extend or modify functionalities.
- **Difficulty in Extending:** Modifying job posting or CV management requires changes to existing classes, reducing maintainability and flexibility.

Refactored Code:

- **JobPostingService Interface:** Decouples job posting from user roles.
- **CVManagementService Interface:** Manages CV operations independently.
- **Dependency Injection:** Adds flexibility by injecting services like job posting and CV management into user roles (e.g., Admin, Employer, JobSeeker).

Fixed Code:

```
interface JobPostingService {
   void postJob(Job job);
   void editJob(Job job);
   void deleteJob(Job job);}
interface CVManagementService {
   void upload(CV cv);
   void review(CV cv);}
abstract class User {
   protected int userId;
   protected String name;
   protected String email;
   protected String password;
   public abstract void performAction();}
class Job {
   private int jobID;
   private String title;
   private String description;
   private String requirements;
   private float salary;
   private int categoryID;
   public Job(int jobID, String title, String description, String requirements, float salary, int categoryID) {
      this.jobID = jobID;
      this.title = title;
      this.description = description;
      this.requirements = requirements;
      this.salary = salary;
      this.categoryID = categoryID;
   }
   public int getJobID() {
      return jobID;
   }
```

```
    public String getTitle() {
        return title;
    }
    public void apply() {
        System.out.println("Job application for: " + title);
    }}
class JobSeeker extends User {
    private CVManager cvManager;
    private JobPostingService jobPostingService;
    public JobSeeker(JobPostingService jobPostingService, CVManager cvManager) {
        this.jobPostingService = jobPostingService;
        this.cvManager = cvManager;
    }
    @Override
    public void performAction() {
        System.out.println(name + " is looking for a job.");
    }
    public void applyForJob(Job job) {
        System.out.println(name + " applied for job: " + job.getTitle());
    }
    public void uploadCV(CV cv) {
        cvManager.upload(cv);
    }
    public void reviewCV(CV cv) {
        cvManager.review(cv);
    }}
class CV {
    private int cvID;
    private int userID;
    private String filePath;
    private String reviewStatus;
    public String getFilePath() {
        return this.filePath;
    }}
class CVManager implements CVManagementService {
    public void upload(CV cv) {
        System.out.println("CV uploaded at: " + cv.getFilePath());
    }
    public void review(CV cv) {
        System.out.println("CV is under review.");
    }}
class Admin extends User implements JobPostingService {
    public Admin() {}
    @Override
    public void performAction() {
        System.out.println(name + " is managing users and jobs.");
    }
    public void postJob(Job job) {
        System.out.println("Admin posted job: " + job.getTitle());
    }
    public void editJob(Job job) {
        System.out.println("Admin edited job: " + job.getTitle());
    }
    public void deleteJob(Job job) {
        System.out.println("Admin deleted job: " + job.getTitle());
    }}
```

```
class Employer extends User implements JobPostingService {
   public Employer() {}
   @Override
   public void performAction() {
      System.out.println(name + " is managing jobs.");}
   public void postJob(Job job) {
      System.out.println("Employer posted job: " + job.getTitle());}
   public void editJob(Job job) {
      System.out.println("Employer edited job: " + job.getTitle());
   }
   public void deleteJob(Job job) {
      System.out.println("Employer deleted job: " + job.getTitle());
   }}
class Freelancer extends User {
   private String portfolio;
   @Override
   public void performAction() {
      System.out.println(name + " is looking for freelance work.");}
   public void applyForFreelanceJob(Job job) {
      System.out.println(name + " applied for freelance job: " + job.getTitle());
   }
   public void getHired() {
      System.out.println(name + " got hired for a freelance job.");
   }}
public class Main3 {
   public static void main(String[] args) {
      CVManager cvManager = new CVManager();
      JobPostingService jobPostingService = new Employer();
      JobSeeker jobSeeker = new JobSeeker(jobPostingService, cvManager);
      jobSeeker.name = "Daniyal Wajid";
      jobSeeker.applyForJob(new Job(1, "Software Developer", "Develop applications", "Java, Spring", 60000,
101));
      jobSeeker.uploadCV(new CV());
      Admin admin = new Admin();
      admin.name = "Admin User";
      admin.postJob(new Job(2, "Project Manager", "Manage projects", "Leadership, Java", 70000, 102));
      Employer employer = new Employer();
      employer.name = "Company Inc.";
      employer.postJob(new Job(3, "Web Developer", "Create websites", "HTML, CSS, JS", 50000, 103));
      Freelancer freelancer = new Freelancer();
      freelancer.name = "Freelance Expert";
      freelancer.applyForFreelanceJob(new Job(4, "Freelance Writer", "Write articles", "Writing, Research",
30000, 104));}}
```

Output:

```
Daniyal Wajid applied for job: Software Developer
CV uploaded at: null
Admin posted job: Project Manager
Employer posted job: Web Developer
Freelance Expert applied for freelance job: Freelance Writer
```

4.2.4 Interface Segregation Principle:

Current Issues:

- **User Class**: The User class is a base class for all roles, but not all roles share the same responsibilities, violating ISP.
- **Lack of Role-based Segregation**: JobSeeker, Admin, Employer, and Freelancer have different responsibilities, making the code hard to extend.
- **Tight Coupling**: JobSeeker and Employer depend on Job and CVManager, limiting flexibility.
- **Unnecessary Methods**: Methods like uploadCV and manageJobs are included in classes that don't require them, violating ISP.

Refactored Code (Applying ISP):

- Role-specific interfaces (JobSeekerActions, EmployerActions, AdminActions, FreelancerActions).
- Decoupled services (JobPostingService, CVManagementService) to ensure classes depend only on relevant interfaces.

Fixed Code:

```
interface JobSeekerActions {
    void applyForJob(Job job);
    void findJob();
    void uploadCV(CV cv);
    void reviewCV(CV cv);
}
interface EmployerActions {
    void postJob(Job job);
    void editJob(Job job);
    void deleteJob(Job job);
    void hireFreelancer(Freelancer freelancer);
}
interface AdminActions {
    void manageUsers();
    void manageJobs();
    void generateReports();
}
interface FreelancerActions {
    void applyForFreelanceJob(Job job);
    void getHired();
}
class User {
    protected int userId;
    protected String name;
    protected String email;
    protected String password;
}
class Job {
    private int jobID;
    private String title;
    private String description;
    private String requirements;
```

```
    private float salary;
    private int categoryID;
    public Job(int jobID, String title, String description, String requirements, float salary, int categoryID) {
        this.jobID = jobID;
        this.title = title;
        this.description = description;
        this.requirements = requirements;
        this.salary = salary;
        this.categoryID = categoryID;
    }
    public int getJobID() {
        return jobID;
    }
    public String getTitle() {
        return title;
    }
}
class CV {
    private int cvID;
    private int userID;
    private String filePath;
    private String reviewStatus;
    public String getFilePath() {
        return filePath;
    }
}
class CVManager {
    public void upload(CV cv) {
        System.out.println("CV uploaded at: " + cv.getFilePath());
    }

    public void review(CV cv) {
        System.out.println("CV is under review.");
    }
}
class JobSeeker extends User implements JobSeekerActions {
    private CVManager cvManager;
    public JobSeeker() {
        this.cvManager = new CVManager();
    }
    @Override
    public void applyForJob(Job job) {
        System.out.println(name + " applied for job: " + job.getTitle());
    }
    @Override
    public void findJob() {
        System.out.println(name + " is looking for a job.");
    }
    @Override
    public void uploadCV(CV cv) {
        cvManager.upload(cv);
    }
    @Override
    public void reviewCV(CV cv) {
        cvManager.review(cv);
    }
```

```
}

class Admin extends User implements AdminActions {
    public void manageUsers() {
        System.out.println(name + " managing users.");
    }
    public void manageJobs() {
        System.out.println(name + " managing jobs.");
    }
    public void generateReports() {
        System.out.println(name + " generating reports.");
    }
}

class Employer extends User implements EmployerActions {
    @Override
    public void postJob(Job job) {
        System.out.println(name + " posted job: " + job.getTitle());
    }

    @Override
    public void editJob(Job job) {
        System.out.println(name + " edited job: " + job.getTitle());
    }

    @Override
    public void deleteJob(Job job) {
        System.out.println(name + " deleted job: " + job.getTitle());
    }

    @Override
    public void hireFreelancer(Freelancer freelancer) {
        System.out.println(name + " hired freelancer: " + freelancer.name);
    }
}

class Freelancer extends User implements FreelancerActions {
    public void applyForFreelanceJob(Job job) {
        System.out.println(name + " applied for freelance job: " + job.getTitle());
    }

    public void getHired() {
        System.out.println(name + " got hired for a freelance job.");
    }
}

class UserAuthentication {
    public void login(User user) {
        System.out.println(user.name + " logged in.");
    }
    public void logout(User user) {
        System.out.println(user.name + " logged out.");
    }
    public void signup(User user) {
        System.out.println(user.name + " signed up.");
    }
```

```
}

public class Main4 {
    public static void main(String[] args) {
        UserAuthentication auth = new UserAuthentication();

        JobSeeker jobSeeker = new JobSeeker();
        jobSeeker.name = "Daniyal Wajid";
        auth.signup(jobSeeker);
        jobSeeker.applyForJob(new Job(1, "Software Developer", "Develop applications", "Java, Spring", 60000,
101));
        jobSeeker.uploadCV(new CV());

        Admin admin = new Admin();
        admin.name = "Admin User";
        admin.manageJobs();

        Employer employer = new Employer();
        employer.name = "Company Inc.";
        Job job = new Job(1, "Software Developer", "Develop applications", "Java, Spring", 60000, 101);
        employer.postJob(job);

        Freelancer freelancer = new Freelancer();
        freelancer.name = "Freelance Expert";
        freelancer.applyForFreelanceJob(job);
    }
}
```

Output:

```
Daniyal Wajid signed up.
Daniyal Wajid applied for job: Software Developer
CV uploaded at: null
Admin User managing jobs.
Company Inc. posted job: Software Developer
Freelance Expert applied for freelance job: Software Developer
```

4.2.5 Dependence Inversion Principle:

Issues with Current Code:

- **High-Level Modules Depend on Low-Level Modules**:JobSeeker, Employer, and Admin directly depend on CVManager and Job. Changes in CVManager or Job affect multiple classes.
- **Lack of Abstraction**:No interfaces between high-level and low-level modules. Direct instantiation and interaction with concrete classes like CVManager and Job.

Refactoring Explanation (Applying DIP):

- **High-Level Modules Depend on Abstractions**: JobSeeker, Admin, and Employer depend on ICVManager and IJobPostingService.
- **Low-Level Modules Implement Interfaces**: CVManager implements ICVManager and JobPostingService implements IJobPostingService.
- **Dependency Injection**: Dependencies (ICVManager, IJobPostingService) are injected into constructors.
- **Flexibility**: Easy to swap implementations (e.g., change CVManager storage method) without affecting high-level modules.

Fixed Code:

```
import java.util.Date;
class CV {
    private String filePath;

    public CV(String filePath) {
        this.filePath = filePath;
    }
    public String getFilePath() {
        return filePath;
    }
}
interface ICVManager {
    void upload(CV cv);
    void review(CV cv);
}
interface IJobPostingService {
    void postJob(Job job);
    void editJob(Job job);
    void deleteJob(Job job);
}
class User {
    protected int userId;
    protected String name;
    protected String email;
    protected String password;
}
class UserAuthentication {
    public void login(User user) {
```

```
        System.out.println(user.name + " logged in.");
    }
    public void logout(User user) {
        System.out.println(user.name + " logged out.");
    }
    public void signup(User user) {
        System.out.println(user.name + " signed up.");
    }
}
class Job {
    private int jobID;
    private String title;
    private String description;
    private String requirements;
    private float salary;
    private int categoryID;
    public Job(int jobID, String title, String description, String requirements, float salary, int categoryID) {
        this.jobID = jobID;
        this.title = title;
        this.description = description;
        this.requirements = requirements;
        this.salary = salary;
        this.categoryID = categoryID;
    }
    public String getTitle() {
        return title;
    }
    public int getJobID() {
        return jobID;
    }
}
class CVManager implements ICVManager {
    public void upload(CV cv) {
        System.out.println("CV uploaded at: " + cv.getFilePath());
    }
    public void review(CV cv) {
        System.out.println("CV is under review.");
    }
}
class JobPostingService implements IJobPostingService {
    public void postJob(Job job) {
        System.out.println("Job posted: " + job.getTitle());
    }
    public void editJob(Job job) {
        System.out.println("Job edited: " + job.getTitle());
    }
    public void deleteJob(Job job) {
        System.out.println("Job deleted: " + job.getTitle());
    }
}
class JobSeeker extends User {
    private ICVManager cvManager;
    public JobSeeker(ICVManager cvManager) {
        this.cvManager = cvManager;
    }
    public void applyForJob(Job job) {
```

```
        JobApplication application = new JobApplication(this.userId, job.getJobID());
        System.out.println(name + " applied for job: " + job.getTitle());
    }
    public void findJob() {
        System.out.println(name + " is looking for a job.");
    }
    public void uploadCV(CV cv) {
        cvManager.upload(cv);
    }
    public void reviewCV(CV cv) {
        cvManager.review(cv);
    }
}
class Admin extends User {
    private IJobPostingService jobPostingService;
    public Admin(IJobPostingService jobPostingService) {
        this.jobPostingService = jobPostingService;
    }
    public void manageUsers() {
        System.out.println("Admin managing users.");
    }
    public void manageJobs() {
        System.out.println("Admin managing jobs.");
    }
    public void generateReports() {
        System.out.println("Admin generating reports.");
    }
    public void postJob(Job job) {
        jobPostingService.postJob(job);
    }
}
class Employer extends User {
    private IJobPostingService jobPostingService;
    public Employer(IJobPostingService jobPostingService) {
        this.jobPostingService = jobPostingService;
    }
    public void postJob(Job job) {
        jobPostingService.postJob(job);
    }
    public void editJob(Job job) {
        jobPostingService.editJob(job);
    }
    public void deleteJob(Job job) {
        jobPostingService.deleteJob(job);
    }
    public void hireFreelancer(Freelancer freelancer) {
        System.out.println("Employer hired freelancer: " + freelancer.name);
    }
}
class Freelancer extends User {
    private String portfolio;
    public void applyForFreelanceJob(Job job) {
        System.out.println(name + " applied for freelance job: " + job.getTitle());
    }
    public void getHired() {
        System.out.println(name + " got hired for a freelance job.");
```

```
    }}
class JobApplication {
    private int applicationID;
    private int jobID;
    private int userID;
    private String status;
    private Date dateApplied;
    public JobApplication(int userID, int jobID) {
        this.userID = userID;
        this.jobID = jobID;
        this.status = "Applied";
        this.dateApplied = new Date();
        this.applicationID = (int) (Math.random() * 1000);
    }
    public void updateStatus(String newStatus) {
        this.status = newStatus;
        System.out.println("Application status updated to: " + status);
    }}
public class Main5 {
    public static void main(String[] args) {
        User user = new User();
        user.name = "Daniyal Wajid";
        UserAuthentication auth = new UserAuthentication();
        auth.signup(user);
        ICVManager cvManager = new CVManager();
        JobSeeker jobSeeker = new JobSeeker(cvManager);
        jobSeeker.name = "Daniyal Wajid";
        jobSeeker.uploadCV(new CV("path/to/cv"));
        IJobPostingService jobPostingService = new JobPostingService();
        Admin admin = new Admin(jobPostingService);
        admin.name = "Admin User";
        admin.manageJobs();
        admin.postJob(new Job(1, "Software Developer", "Develop applications", "Java, Spring", 60000, 101));

        Employer employer = new Employer(jobPostingService);
        employer.name = "Company Inc.";
        Job job = new Job(1, "Software Developer", "Develop applications", "Java, Spring", 60000, 101);
        employer.postJob(job);

        Freelancer freelancer = new Freelancer();
        freelancer.name = "Freelance Expert";
        freelancer.applyForFreelanceJob(job);
    }
}
```

Output:

```
Daniyal Wajid signed up.
CV uploaded at: path/to/cv
Admin managing jobs.
Job posted: Software Developer
Job posted: Software Developer
Freelance Expert applied for freelance job: Software Developer
```

4.3 Design Patterns:

4.3.1 Creational Design Pattern

Singleton Pattern

CVManager:

- The CVManager class is made a Singleton by ensuring that only one instance of it can exist.
- The getInstance() method provides access to the single instance of CVManager.
- The constructor of CVManager is private to prevent instantiation outside the class.

JobSeeker Class:

- In the JobSeeker class, we use the Singleton instance of CVManager to handle CV operations.
- The CVManager.getInstance() method is used to access the single instance of CVManager.

Main Method:

- The Main class demonstrates creating instances of JobSeeker, Employer, Admin, and Freelancer.
- The Singleton pattern is tested by checking if two calls to CVManager.getInstance() return the same instance.

Code:

```
import java.util.Date;
class User {
    protected int userId;
    protected String name;
    protected String email;
    protected String password;
}
class UserAuthentication {
    public void login(User user) {
        System.out.println(user.name + " logged in.");
    }
    public void logout(User user) {
        System.out.println(user.name + " logged out.");
    }
    public void signup(User user) {
        System.out.println(user.name + " signed up.");
    }}
class Job {
    private int jobID;
    private String title;
    private String description;
```

```
    private String requirements;
    private float salary;
    private int categoryID;
    public Job(int jobID, String title, String description, String requirements, float salary, int categoryID) {
        this.jobID = jobID;
        this.title = title;
        this.description = description;
        this.requirements = requirements;
        this.salary = salary;
        this.categoryID = categoryID;
    }
    public void apply() {
        System.out.println("Job application for: " + title);
    }
    public String getTitle() {
        return title;
    }
    public int getJobID() {
        return jobID;
    }}
class CV {
    private int cvID;
    private int userID;
    private String filePath;
    private String reviewStatus;
    public String getFilePath() {
        return filePath;
    }
    public void setFilePath(String filePath) {
        this.filePath = filePath;
    }}
class CVManager {
    private static CVManager instance;
    private CVManager() {}
    public static CVManager getInstance() {
        if (instance == null) {
            synchronized (CVManager.class) {
                if (instance == null) {
                    instance = new CVManager();
                }
            }
        }
        return instance;
    }
    public void upload(CV cv) {
        System.out.println("CV uploaded at: " + cv.getFilePath());
    }
    public void review(CV cv) {
        System.out.println("CV is under review.");
    }
}
class JobSeeker extends User {
    private CVManager cvManager;
    public JobSeeker() {
        this.cvManager = CVManager.getInstance();
    }
```

```
    public void applyForJob(Job job) {
        JobApplication application = new JobApplication(this.userId, job.getJobID());
        System.out.println(name + " applied for job: " + job.getTitle());
    }
    public void findJob() {
        System.out.println(name + " is looking for a job.");
    }
    public void uploadCV(CV cv) {
        cvManager.upload(cv);
    }
    public void reviewCV(CV cv) {
        cvManager.review(cv);
    }
}
class Admin extends User {
    private int adminID;
    public void manageUsers() {
        System.out.println("Admin managing users.");
    }
    public void manageJobs() {
        System.out.println("Admin managing jobs.");
    }
    public void generateReports() {
        System.out.println("Admin generating reports.");
    }
}
class Employer extends User {
    public void postJob(Job job) {
        System.out.println("Employer posted job: " + job.getTitle());
    }
    public void editJob(Job job) {
        System.out.println("Employer edited job: " + job.getTitle());
    }
    public void deleteJob(Job job) {
        System.out.println("Employer deleted job: " + job.getTitle());
    }
    public void hireFreelancer(Freelancer freelancer) {
        System.out.println("Employer hired freelancer: " + freelancer.name);
    }
}
class Freelancer extends User {
    private String portfolio;

    public void applyForFreelanceJob(Job job) {
        System.out.println(name + " applied for freelance job: " + job.getTitle());
    }
    public void getHired() {
        System.out.println(name + " got hired for a freelance job.");
    }}
class JobApplication {
    private int applicationID;
    private int jobID;
    private int userID;
    private String status;
    private Date dateApplied;
   public JobApplication(int userID, int jobID) {
```

```
        this.userID = userID;
        this.jobID = jobID;
        this.status = "Applied";
        this.dateApplied = new Date();
        this.applicationID = (int) (Math.random() * 1000);
    }
    public void updateStatus(String newStatus) {
        this.status = newStatus;
        System.out.println("Application status updated to: " + status);
    }}
public class Main6 {
    public static void main(String[] args) {
        User user = new User();
        user.name = "Daniyal Wajid";
        UserAuthentication auth = new UserAuthentication();
        auth.signup(user);

        JobSeeker jobSeeker = new JobSeeker();
        jobSeeker.name = "Daniyal Wajid";
        jobSeeker.uploadCV(new CV());

        Admin admin = new Admin();
        admin.name = "Admin User";
        admin.manageJobs();

        Employer employer = new Employer();
        employer.name = "Company Inc.";
        Job job = new Job(1, "Software Developer", "Develop applications", "Java, Spring", 60000, 101);
        employer.postJob(job);

        Freelancer freelancer = new Freelancer();
        freelancer.name = "Freelance Expert";
        freelancer.applyForFreelanceJob(job);

        CVManager cvManager1 = CVManager.getInstance();
        CVManager cvManager2 = CVManager.getInstance();

        System.out.println("CVManager instance 1: " + cvManager1);
        System.out.println("CVManager instance 2: " + cvManager2);
        System.out.println("Are both instances the same? " + (cvManager1 == cvManager2));
    }}
```

Output:

```
Daniyal Wajid signed up.
CV uploaded at: null
Admin managing jobs.
Employer posted job: Software Developer
Freelance Expert applied for freelance job: Software Developer
CVManager instance 1: CVManager@1ddc4ec2
CVManager instance 2: CVManager@1ddc4ec2
Are both instances the same? true
```

Factory Pattern:

Changes in the code:

- **UserFactory**: Centralizes object creation for different user types (JobSeeker, Admin, etc.).
- **Main Class**: Uses UserFactory to create users, making the code more maintainable and extensible.

Benefits:

- **Decoupling**: No need to directly instantiate specific classes in client code.
- **Maintainability**: Easy to add new user types by modifying the factory.
- **Extensibility**: New types can be added without affecting existing code.

Code:

```
import java.util.Date;
class User {
    protected int userId;
    protected String name;
    protected String email;
    protected String password;
}
class UserAuthentication {
    public void login(User user) {
        System.out.println(user.name + " logged in.");
    }
    public void logout(User user) {
        System.out.println(user.name + " logged out.");
    }
    public void signup(User user) {
        System.out.println(user.name + " signed up.");
    }
}
class Job {
    private int jobID;
    private String title;
    private String description;
    private String requirements;
    private float salary;
    private int categoryID;
    public Job(int jobID, String title, String description, String requirements, float salary, int categoryID) {
        this.jobID = jobID;
        this.title = title;
        this.description = description;
        this.requirements = requirements;
        this.salary = salary;
        this.categoryID = categoryID;
    }
    public void apply() {
```

```
        System.out.println("Job application for: " + title);
    }
    public String getTitle() {
        return title;
    }
    public int getJobID() {
        return jobID;
    }
}
class JobSeeker extends User {
    private CVManager cvManager;
    public JobSeeker() {
        this.cvManager = new CVManager(); // Initialize CVManager
    }
    public void applyForJob(Job job) {
        JobApplication application = new JobApplication(this.userId, job.getJobID());
        System.out.println(name + " applied for job: " + job.getTitle());
    }
    public void findJob() {
        System.out.println(name + " is looking for a job.");
    }
    public void uploadCV(CV cv) {
        cvManager.upload(cv);
    }
    public void reviewCV(CV cv) {
        cvManager.review(cv);
    }
}
class CV {
    private int cvID;
    private int userID;
    private String filePath;
    private String reviewStatus;
    public String getFilePath() {
        return filePath;
    }
    public void setFilePath(String filePath) {
        this.filePath = filePath;
    }
}
class CVManager {
    public void upload(CV cv) {
        System.out.println("CV uploaded at: " + cv.getFilePath());
    }
    public void review(CV cv) {
        System.out.println("CV is under review.");
    }
}
class Admin extends User {
    private int adminID;
    public void manageUsers() {
        System.out.println("Admin managing users.");
    }
    public void manageJobs() {
        System.out.println("Admin managing jobs.");
    }
```

```
    public void generateReports() {
        System.out.println("Admin generating reports.");
    }
}
class Employer extends User {
    public void postJob(Job job) {
        System.out.println("Employer posted job: " + job.getTitle());
    }
    public void editJob(Job job) {
        System.out.println("Employer edited job: " + job.getTitle());
    }
    public void deleteJob(Job job) {
        System.out.println("Employer deleted job: " + job.getTitle());
    }
    public void hireFreelancer(Freelancer freelancer) {
        System.out.println("Employer hired freelancer: " + freelancer.name);
    }
}
class Freelancer extends User {
    private String portfolio;
    public void applyForFreelanceJob(Job job) {
        System.out.println(name + " applied for freelance job: " + job.getTitle());
    }
    public void getHired() {
        System.out.println(name + " got hired for a freelance job.");
    }
}
class JobApplication {
    private int applicationID;
    private int jobID;
    private int userID;
    private String status;
    private Date dateApplied;
    public JobApplication(int userID, int jobID) {
        this.userID = userID;
        this.jobID = jobID;
        this.status = "Applied";
        this.dateApplied = new Date();
        this.applicationID = (int) (Math.random() * 1000);
    }
    public void updateStatus(String newStatus) {
        this.status = newStatus;
        System.out.println("Application status updated to: " + status);
    }
}
class UserFactory {
    public static User createUser(String userType) {
        switch (userType.toLowerCase()) {
            case "jobseeker":
                return new JobSeeker();
            case "admin":
                return new Admin();
            case "employer":
                return new Employer();
            case "freelancer":
                return new Freelancer();
```

```
        default:
            throw new IllegalArgumentException("Invalid user type.");
    }
  }
}

public class Main7 {
  public static void main(String[] args) {
    User user1 = UserFactory.createUser("jobseeker");
    user1.name = "Daniyal Wajid";
    UserAuthentication auth = new UserAuthentication();
    auth.signup(user1);

    JobSeeker jobSeeker = (JobSeeker) user1;
    jobSeeker.uploadCV(new CV());

    User user2 = UserFactory.createUser("admin");
    user2.name = "Admin User";
    Admin admin = (Admin) user2;
    admin.manageJobs();

    User user3 = UserFactory.createUser("employer");
    user3.name = "Company Inc.";
    Employer employer = (Employer) user3;
    Job job = new Job(1, "Software Developer", "Develop applications", "Java, Spring", 60000, 101);
    employer.postJob(job);

    User user4 = UserFactory.createUser("freelancer");
    user4.name = "Freelance Expert";
    Freelancer freelancer = (Freelancer) user4;
    freelancer.applyForFreelanceJob(job);
  }
}
```

Output:

```
Daniyal Wajid signed up.
CV uploaded at: null
Admin managing jobs.
Employer posted job: Software Developer
Freelance Expert applied for freelance job: Software Developer
```

Abstract Pattern:

- **Abstract Factory**: UserFactory defines a common interface for creating User objects.
- **Concrete Factories**: Each concrete factory (JobSeekerFactory, AdminFactory, etc.) is responsible for creating the corresponding user type.
- **Client Code**: The Main class creates users using the appropriate factory, without needing to know the specific class type.

Benefits:

- **Encapsulation**: The Main class doesn't need to know about specific User classes. It just interacts with the abstract UserFactory.
- **Flexibility**: If new user types need to be added, you just need to create new concrete factories without changing the client code.

Code:

```
abstract class UserFactory {
    public abstract User createUser();
}
class JobSeekerFactory extends UserFactory {
    @Override
    public User createUser() {
        return new JobSeeker();
    }
}
class AdminFactory extends UserFactory {
    @Override
    public User createUser() {
        return new Admin();
    }
}
class EmployerFactory extends UserFactory {
    @Override
    public User createUser() {
        return new Employer();
    }
}
class FreelancerFactory extends UserFactory {
    @Override
    public User createUser() {
        return new Freelancer();
    }
}
abstract class User {
    protected int userId;
    protected String name;
    protected String email;
    protected String password;
    public abstract void displayRole();
}
class JobSeeker extends User {
    private CVManager cvManager;
```

```
    public JobSeeker() {
        this.cvManager = new CVManager();
    }
    public void applyForJob(Job job) {
        System.out.println(name + " applied for job: " + job.getTitle());
    }
    public void findJob() {
        System.out.println(name + " is looking for a job.");
    }
    public void uploadCV(CV cv) {
        cvManager.upload(cv);
    }
    public void reviewCV(CV cv) {
        cvManager.review(cv);
    }
    @Override
    public void displayRole() {
        System.out.println(name + " is a Job Seeker.");
    }
}
class Admin extends User {
    private int adminID;
    public void manageUsers() {
        System.out.println("Admin managing users.");
    }
    public void manageJobs() {
        System.out.println("Admin managing jobs.");
    }
    public void generateReports() {
        System.out.println("Admin generating reports.");
    }
    @Override
    public void displayRole() {
        System.out.println(name + " is an Admin.");
    }
}
class Employer extends User {
    public void postJob(Job job) {
        System.out.println("Employer posted job: " + job.getTitle());
    }
    public void editJob(Job job) {
        System.out.println("Employer edited job: " + job.getTitle());
    }
    public void deleteJob(Job job) {
        System.out.println("Employer deleted job: " + job.getTitle());
    }
    public void hireFreelancer(Freelancer freelancer) {
        System.out.println("Employer hired freelancer: " + freelancer.name);
    }
    @Override
    public void displayRole() {
        System.out.println(name + " is an Employer.");
    }
}
class Freelancer extends User {
    private String portfolio;
```

```
    public void applyForFreelanceJob(Job job) {
        System.out.println(name + " applied for freelance job: " + job.getTitle());
    }
    public void getHired() {
        System.out.println(name + " got hired for a freelance job.");
    }
    @Override
    public void displayRole() {
        System.out.println(name + " is a Freelancer.");
    }
}
class Job {
    private int jobID;
    private String title;
    private String description;
    private String requirements;
    private float salary;
    private int categoryID;
    public Job(int jobID, String title, String description, String requirements, float salary, int categoryID) {
        this.jobID = jobID;
        this.title = title;
        this.description = description;
        this.requirements = requirements;
        this.salary = salary;
        this.categoryID = categoryID;
    }
    public String getTitle() {
        return title;
    }
}
class CV {
    private int cvID;
    private int userID;
    private String filePath;
    private String reviewStatus;
}
class CVManager {
    public void upload(CV cv) {
        System.out.println("CV uploaded at: DB");
    }
    public void review(CV cv) {
        System.out.println("CV is under review.");
    }
}
public class Main8 {
    public static void main(String[] args) {
        UserFactory jobSeekerFactory = new JobSeekerFactory();
        UserFactory adminFactory = new AdminFactory();
        UserFactory employerFactory = new EmployerFactory();
        UserFactory freelancerFactory = new FreelancerFactory();

        User jobSeeker = jobSeekerFactory.createUser();
        jobSeeker.name = "Daniyal Wajid";

        User admin = adminFactory.createUser();
        admin.name = "Admin User";
```

```
        User employer = employerFactory.createUser();
        employer.name = "Company Inc.";

        User freelancer = freelancerFactory.createUser();
        freelancer.name = "Freelance Expert";

        jobSeeker.displayRole();
        admin.displayRole();
        employer.displayRole();
        freelancer.displayRole();

        JobSeeker jobSeekerUser = (JobSeeker) jobSeeker;
        jobSeekerUser.applyForJob(new Job(1, "Software Developer", "Develop applications", "Java, Spring",
60000, 101));

        Employer employerUser = (Employer) employer;
        employerUser.postJob(new Job(2, "Data Scientist", "Analyze data", "Python, ML", 80000, 102));

        Freelancer freelancerUser = (Freelancer) freelancer;
        freelancerUser.applyForFreelanceJob(new Job(3, "Web Developer", "Build websites", "HTML, CSS,
JavaScript", 50000, 103));
    }
}
```

Output:

```
Daniyal Wajid is a Job Seeker.
Admin User is an Admin.
Company Inc. is an Employer.
Freelance Expert is a Freelancer.
Daniyal Wajid applied for job: Software Developer
Employer posted job: Data Scientist
Freelance Expert applied for freelance job: Web Developer
```

4.3.2 Structural Design Pattttern

Apdapter Pattttern:

Assume that you want to introduce a new system for logging in and managing users that does not follow the same structure as the UserAuthentication class. If the new system has a different interface, you can use the Adapter Pattern to make the new system compatible with your existing code.

Steps to Implement Adapter Pattttern:

1. **Create an Interface for the Target System**: The interface your existiting system (in this case UserAuthentitication) expects.
2. **Create an Adapter Class**: This class will adapt the new system to the target interface.
3. **Implement the Adapter**: In the adapter class, you implement the methods required by the target interface and delegate calls to the new system.

Code:

```
import java.util.Date;


class User {
    protected int userId;
    protected String name;
    protected String email;
    protected String password;
}

class UserAuthentication {
    public void login(User user) {
        System.out.println(user.name + " logged in.");
    }

    public void logout(User user) {
        System.out.println(user.name + " logged out.");
    }

    public void signup(User user) {
        System.out.println(user.name + " signed up.");
    }
}

class NewAuthSystem {
    public void registerUser(String name, String email) {
        System.out.println(name + " is registered using the new system.");
    }

    public void loginUser(String name) {
        System.out.println(name + " logged in using the new system.");
```

```
    }
}

class AuthenticationAdapter extends UserAuthentication {
    private NewAuthSystem newAuthSystem;

    public AuthenticationAdapter(NewAuthSystem newAuthSystem) {
        this.newAuthSystem = newAuthSystem;
    }

    @Override
    public void signup(User user) {
        newAuthSystem.registerUser(user.name, user.email);
    }

    @Override
    public void login(User user) {
        newAuthSystem.loginUser(user.name);
    }

    @Override
    public void logout(User user) {
        System.out.println(user.name + " logged out using the new system.");
    }
}

public class Main9 {
    public static void main(String[] args) {
        User user = new User();
        user.name = "Daniyal Wajid";
        user.email = "daniyal@example.com";

        NewAuthSystem newAuthSystem = new NewAuthSystem();

        UserAuthentication authAdapter = new AuthenticationAdapter(newAuthSystem);

        authAdapter.signup(user);
        authAdapter.login(user);
        authAdapter.logout(user);
    }
}
```

Output:

```
Daniyal Wajid is registered using the new system.
Daniyal Wajid logged in using the new system.
Daniyal Wajid logged out using the new system.
```

Bridge Pattern:

Explanation:

1. **Abstraction (UserAuthentication)**: It holds a reference to an AuthenticationSystem object, which is a bridge to different authentication systems.
2. **Implementor (AuthenticationSystem)**: It is an interface that provides methods like signup, login, and logout. The implementor defines the contract for authentication.
3. **ConcreteImplementor (NewAuthSystem)**: This is a concrete class that implements AuthenticationSystem. It has methods like registerUser, loginUser, and logoutUser, which contain the specific logic for the new authentication system.
4. **Bridge**: The UserAuthentication class is now decoupled from the actual authentication logic. It can switch between different authentication systems (like NewAuthSystem) without changing the UserAuthentication class.

Code:

```
import java.util.Date;
class User {
    protected int userId;
    protected String name;
    protected String email;
    protected String password;
}
interface AuthenticationSystem {
    void signup(User user);
    void login(User user);
    void logout(User user);
}
class UserAuthentication {
    protected AuthenticationSystem authSystem;
    public UserAuthentication(AuthenticationSystem authSystem) {
        this.authSystem = authSystem;
    }
    public void login(User user) {
        authSystem.login(user);
    }
    public void logout(User user) {
        authSystem.logout(user);
    }
    public void signup(User user) {
        authSystem.signup(user);
    }
}
class NewAuthSystem implements AuthenticationSystem {
    public void registerUser(String name, String email) {
        System.out.println(name + " is registered using the new system.");
    }
    public void loginUser(String name) {
        System.out.println(name + " logged in using the new system.");
    }
    public void logoutUser(String name) {
        System.out.println(name + " logged out using the new system.");
```

```
    }
    @Override
    public void signup(User user) {
        registerUser(user.name, user.email);
    }
    @Override
    public void login(User user) {
        loginUser(user.name);
    }
    @Override
    public void logout(User user) {
        logoutUser(user.name);
    }
}
class Job {
    private int jobID;
    private String title;
    private String description;
    private String requirements;
    private float salary;
    private int categoryID;
    public Job(int jobID, String title, String description, String requirements, float salary, int categoryID) {
        this.jobID = jobID;
        this.title = title;
        this.description = description;
        this.requirements = requirements;
        this.salary = salary;
        this.categoryID = categoryID;
    }
    public String getTitle() {
        return title;
    }
    public int getJobID() {
        return jobID;
    }
    public void apply() {
        System.out.println("Job application for: " + title);
    }
}
class JobSeeker extends User {
    private CVManager cvManager;
    public JobSeeker() {
        this.cvManager = new CVManager();
    }
    public void applyForJob(Job job) {
        System.out.println(name + " applied for job: " + job.getTitle());
    }
    public void findJob() {
        System.out.println(name + " is looking for a job.");
    }
    public void uploadCV(CV cv) {
        cvManager.upload(cv);
    }
    public void reviewCV(CV cv) {
        cvManager.review(cv);
    }
```

```
}
class CV {
    private int cvID;
    private int userID;
    private String filePath;
    private String reviewStatus;
    public String getFilePath() {
        return filePath;
    }
}
class CVManager {
    public void upload(CV cv) {
        System.out.println("CV uploaded at: " + cv.getFilePath());
    }

    public void review(CV cv) {
        System.out.println("CV is under review.");
    }
}
class Admin extends User {
    public void manageUsers() {
        System.out.println("Admin managing users.");
    }
    public void manageJobs() {
        System.out.println("Admin managing jobs.");
    }
    public void generateReports() {
        System.out.println("Admin generating reports.");
    }
}
class Employer extends User {
    public void postJob(Job job) {
        System.out.println("Employer posted job: " + job.getTitle());
    }
}
class Freelancer extends User {
    private String portfolio;
    public void applyForFreelanceJob(Job job) {
        System.out.println(name + " applied for freelance job: " + job.getTitle());
    }
    public void getHired() {
        System.out.println(name + " got hired for a freelance job.");
    }
}
public class Main10 {
    public static void main(String[] args) {
        User user = new User();
        user.name = "Daniyal Wajid";
        user.email = "daniyal@example.com";

        NewAuthSystem newAuthSystem = new NewAuthSystem();
        UserAuthentication authAdapter = new UserAuthentication(newAuthSystem);

        authAdapter.signup(user);
        authAdapter.login(user);
        authAdapter.logout(user);
```

```
    JobSeeker jobSeeker = new JobSeeker();
    jobSeeker.name = "Daniyal Wajid";
    jobSeeker.uploadCV(new CV());

    Admin admin = new Admin();
    admin.name = "Admin User";
    admin.manageJobs();

    Employer employer = new Employer();
    employer.name = "Company Inc.";
    Job job = new Job(1, "Software Developer", "Develop applications", "Java, Spring", 60000, 101);
    employer.postJob(job);

    Freelancer freelancer = new Freelancer();
    freelancer.name = "Freelance Expert";
    freelancer.applyForFreelanceJob(job);
  }
}
```

Output:

```
Daniyal Wajid is registered using the new system.
Daniyal Wajid logged in using the new system.
Daniyal Wajid logged out using the new system.
CV uploaded at: null
Admin managing jobs.
Employer posted job: Software Developer
Freelance Expert applied for freelance job: Software Developer
```

4.3.3 Behavioral Design Pattern

Command Pattern:

Refactoring Explanation:

- **Command Interface**: We define a UserCommand interface with an execute method. This allows any action related to user authentication (login, signup, logout) to be treated as a command.
- **Concrete Command Classes**: We create specific command classes (LoginCommand, SignupCommand, LogoutCommand) that implement the UserCommand interface and provide their respective implementation for execute (i.e., calling login, signup, and logout on UserAuthentication).
- **Invoker**: The UserCommandInvoker is the class that takes a command and invokes it. The client (i.e., Main method) creates the command objects and sets them in the invoker, which then executes the command.

Benefits:

- **Decoupling**: The client code doesn't need to know the details of how login, signup, or logout is implemented. It just calls invoke() on the invoker.
- **Extensibility**: If we want to add more actions (like resetting a password), we can just create a new command class without modifying the existing code.

Code:

```
import java.util.Date;
class User {
    protected int userId;
    protected String name;
    protected String email;
    protected String password;
}
interface AuthenticationSystem {
    void signup(User user);
    void login(User user);
    void logout(User user);
}
class UserAuthentication {
    protected AuthenticationSystem authSystem;
    public UserAuthentication(AuthenticationSystem authSystem) {
        this.authSystem = authSystem;
    }
    public void login(User user) {
        authSystem.login(user);
    }
    public void logout(User user) {
        authSystem.logout(user);
    }
    public void signup(User user) {
```

```
        authSystem.signup(user);
    }
}
class NewAuthSystem implements AuthenticationSystem {
    public void registerUser(String name, String email) {
        System.out.println(name + " is registered using the new system.");
    }
    public void loginUser(String name) {
        System.out.println(name + " logged in using the new system.");
    }
    public void logoutUser(String name) {
        System.out.println(name + " logged out using the new system.");
    }
    @Override
    public void signup(User user) {
        registerUser(user.name, user.email);
    }
    @Override
    public void login(User user) {
        loginUser(user.name);
    }
    @Override
    public void logout(User user) {
        logoutUser(user.name);
    }
}
class Job {
    private int jobID;
    private String title;
    private String description;
    private String requirements;
    private float salary;
    private int categoryID;
    public Job(int jobID, String title, String description, String requirements, float salary, int categoryID) {
        this.jobID = jobID;
        this.title = title;
        this.description = description;
        this.requirements = requirements;
        this.salary = salary;
        this.categoryID = categoryID;
    }
    public String getTitle() {
        return title;
    }
    public int getJobID() {
        return jobID;
    }
    public void apply() {
        System.out.println("Job application for: " + title);
    }
}
class JobSeeker extends User {
    private CVManager cvManager;
    public JobSeeker() {
        this.cvManager = new CVManager();
    }
```

```
    public void applyForJob(Job job) {
        System.out.println(name + " applied for job: " + job.getTitle());
    }
    public void findJob() {
        System.out.println(name + " is looking for a job.");
    }
    public void uploadCV(CV cv) {
        cvManager.upload(cv);
    }
    public void reviewCV(CV cv) {
        cvManager.review(cv);
    }}
class CV {
    private int cvID;
    private int userID;
    private String filePath;
    private String reviewStatus;
    public String getFilePath() {
        return filePath;
    }}
class CVManager {
    public void upload(CV cv) {
        System.out.println("CV uploaded at: " + cv.getFilePath());
    }
    public void review(CV cv) {
        System.out.println("CV is under review.");
    }}
class Admin extends User {
    public void manageUsers() {
        System.out.println("Admin managing users.");
    }

    public void manageJobs() {
        System.out.println("Admin managing jobs.");
    }

    public void generateReports() {
        System.out.println("Admin generating reports.");
    }
}
class Employer extends User {
    public void postJob(Job job) {
        System.out.println("Employer posted job: " + job.getTitle());
    }
}
class Freelancer extends User {
    private String portfolio;

    public void applyForFreelanceJob(Job job) {
        System.out.println(name + " applied for freelance job: " + job.getTitle());
    }

    public void getHired() {
        System.out.println(name + " got hired for a freelance job.");
    }
}
```

```java
public class Main10 {
    public static void main(String[] args) {
        User user = new User();
        user.name = "Daniyal Wajid";
        user.email = "daniyal@example.com";

        NewAuthSystem newAuthSystem = new NewAuthSystem();
        UserAuthentication authAdapter = new UserAuthentication(newAuthSystem);

        authAdapter.signup(user);
        authAdapter.login(user);
        authAdapter.logout(user);

        JobSeeker jobSeeker = new JobSeeker();
        jobSeeker.name = "Daniyal Wajid";
        jobSeeker.uploadCV(new CV());

        Admin admin = new Admin();
        admin.name = "Admin User";
        admin.manageJobs();

        Employer employer = new Employer();
        employer.name = "Company Inc.";
        Job job = new Job(1, "Software Developer", "Develop applications", "Java, Spring", 60000, 101);
        employer.postJob(job);

        Freelancer freelancer = new Freelancer();
        freelancer.name = "Freelance Expert";
        freelancer.applyForFreelanceJob(job);
    }
}
```

Observer Pattern:

Refactoring Explanation:

- **Subject Interface**: The JobApplicationSubject interface defines methods to add, remove, and notify observers.
- **Concrete Subject**: The JobApplication class implements this interface and manages a list of observers. When the application status changes (via updateStatus), it notifies all observers.
- **Observer Interface**: The JobApplicationObserver interface defines the update method, which is called when the status changes.
- **Concrete Observers**: Classes like JobSeekerObserver and AdminObserver implement the JobApplicationObserver interface. These classes will react whenever the job application's status changes.

Benefits:

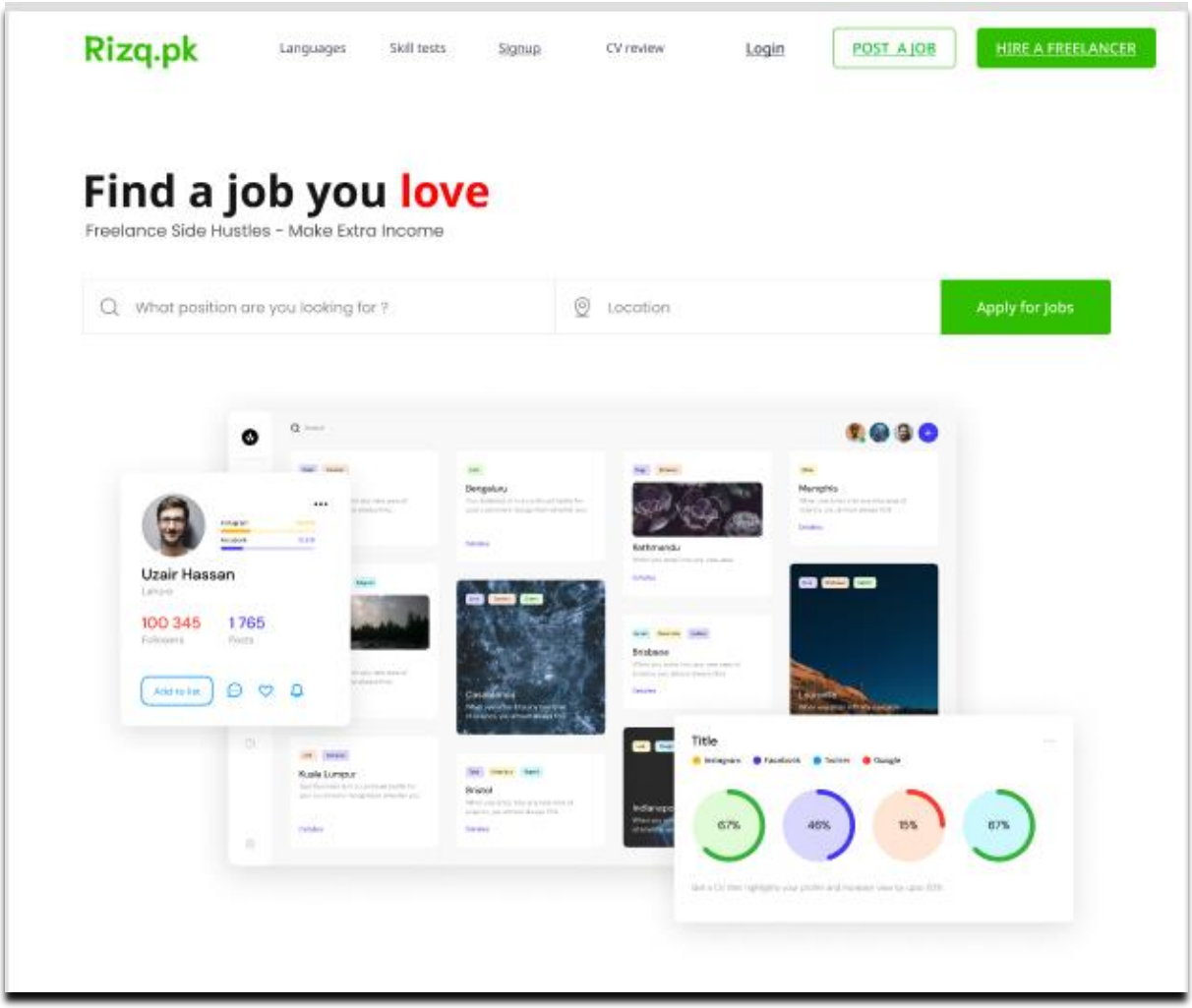
- **Decoupling**: The JobApplication doesn't need to know what the observers do. It just notifies them when the status changes. Observers like JobSeeker or Admin are free to implement their own logic for reacting to the change.
- **Flexibility**: New observers can be added easily without changing the JobApplication class. For example, if we need to notify a HRManagerObserver, we can simply add it to the observer list.
- **Dynamic Updates**: Multiple components in the system can be notified of a change without tightly coupling them.

Code:

```
interface JobApplicationSubject {
    void addObserver(JobApplicationObserver observer);
    void removeObserver(JobApplicationObserver observer);
    void notifyObservers();
}
interface JobApplicationObserver {
    void update(JobApplication jobApplication);
}
class JobSeekerObserver implements JobApplicationObserver {
    @Override
    public void update(JobApplication jobApplication) {
        System.out.println("Job Seeker notified: Application status updated to " + jobApplication.getStatus());
    }
}
class AdminObserver implements JobApplicationObserver {
    @Override
    public void update(JobApplication jobApplication) {
        System.out.println("Admin notified: Application status updated to " + jobApplication.getStatus());
    }
}
class JobApplication implements JobApplicationSubject {
    private int applicationID;
    private int jobID;
    private int userID;
    private String status;
```

```
    private Date dateApplied;
    private List<JobApplicationObserver> observers = new ArrayList<>();

    public JobApplication(int userID, int jobID) {
        this.userID = userID;
        this.jobID = jobID;
        this.status = "Applied";
        this.dateApplied = new Date();
        this.applicationID = (int) (Math.random() * 1000);
    }
    @Override
    public void addObserver(JobApplicationObserver observer) {
        observers.add(observer);
    }
    @Override
    public void removeObserver(JobApplicationObserver observer) {
        observers.remove(observer);
    }
    @Override
    public void notifyObservers() {
        for (JobApplicationObserver observer : observers) {
            observer.update(this);
        }
    }

    public void updateStatus(String newStatus) {
        this.status = newStatus;
        System.out.println("Application status updated to: " + status);
        notifyObservers();
    }

    public String getStatus() {
        return status;
    }
}
public class Main {
    public static void main(String[] args) {
        JobSeeker jobSeeker = new JobSeeker();
        Admin admin = new Admin();

        JobApplication application = new JobApplication(jobSeeker.userId, 101);

        application.addObserver(new JobSeekerObserver());
        application.addObserver(new AdminObserver());

        application.updateStatus("Under Review");
        application.updateStatus("Rejected");
    }
}
```

5. Prototype:


Rizq.pk
Languages
Skill tests
Signup
CV review
Login
POST A JOB
HIRE A FREELANCER
Find a job you love
Freelance Side Hustles - Make Extra Income
What position are you looking for ?
Location
Apply for Jobs
Uzair Hassan
100 345
1 765
Title
67%
46%
15%
87%

Rizq Top Employers Are!

ClickUp Dropbox elastic GitHub FreshBooks

Help Scout HubSpot intuit Google PAYCHEX

SAP segment servicenow shopify

How We Do?

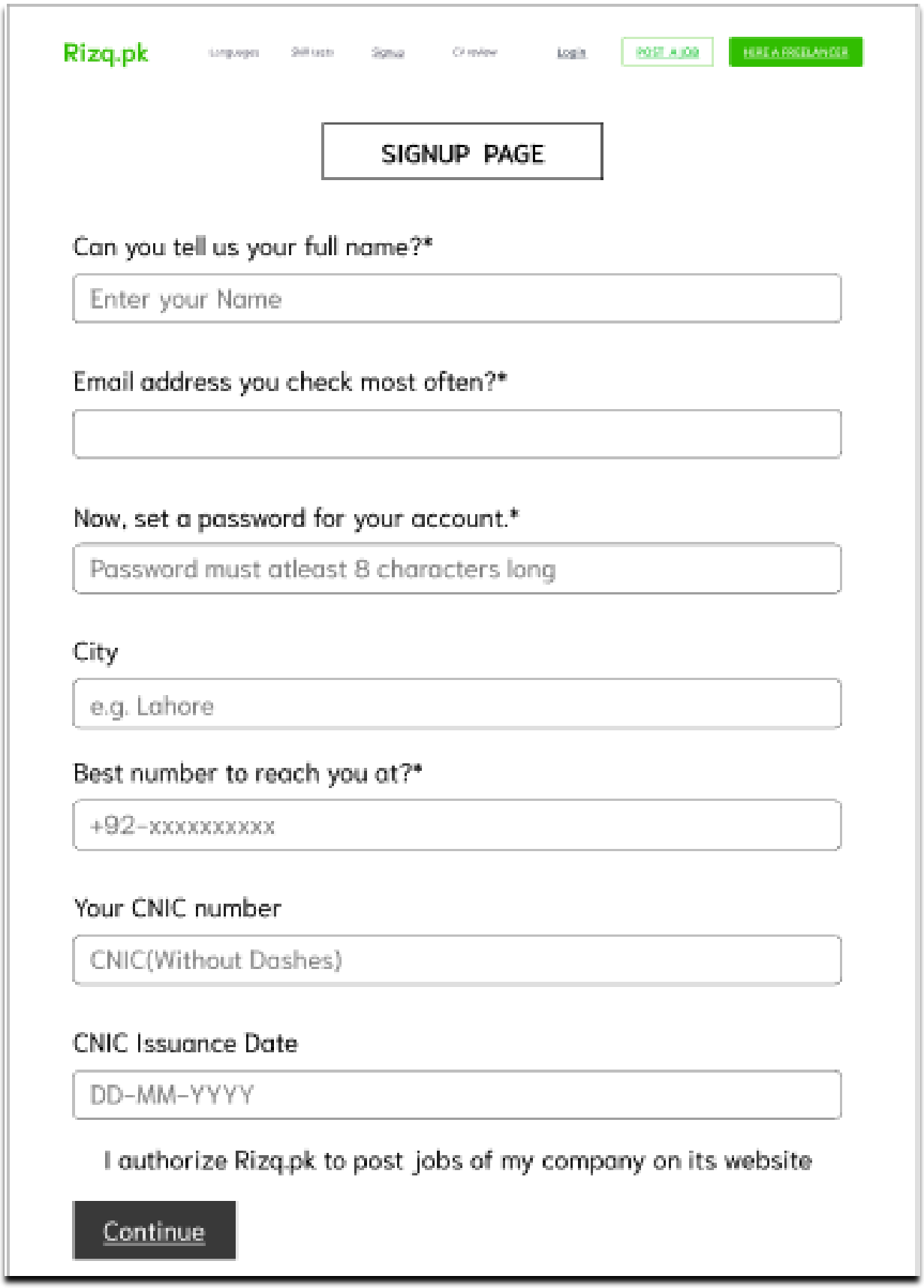
STEP 1: COMPLETE PROFILE

STEP 2: RECEIVE JOB OFFERS

STEP 3: ACCEPT DREAM JOB

Rizq.pk

Home	Course	Article	Contact Us
Product	Photoshop	New	johndoe@gmail.com
Course	Illustrator	Old	
About Us	Figma	Trend	
Log in	AfterEffects	Popular	


Rizq.pk
Login
POST A JOB
HIRE A PROGRAMMER
SIGNUP PAGE
Can you tell us your full name?*
Enter your Name
Email address you check most often?*
Now, set a password for your account.*
Password must atleast 8 characters long
City
e.g. Lahore
Best number to reach you at?*
+92-xxxxxxxxxx
Your CNIC number
CNIC(Without Dashes)
CNIC Issuance Date
DD-MM-YYYY
I authorize Rizq.pk to post jobs of my company on its website
Continue

Rizq.pk

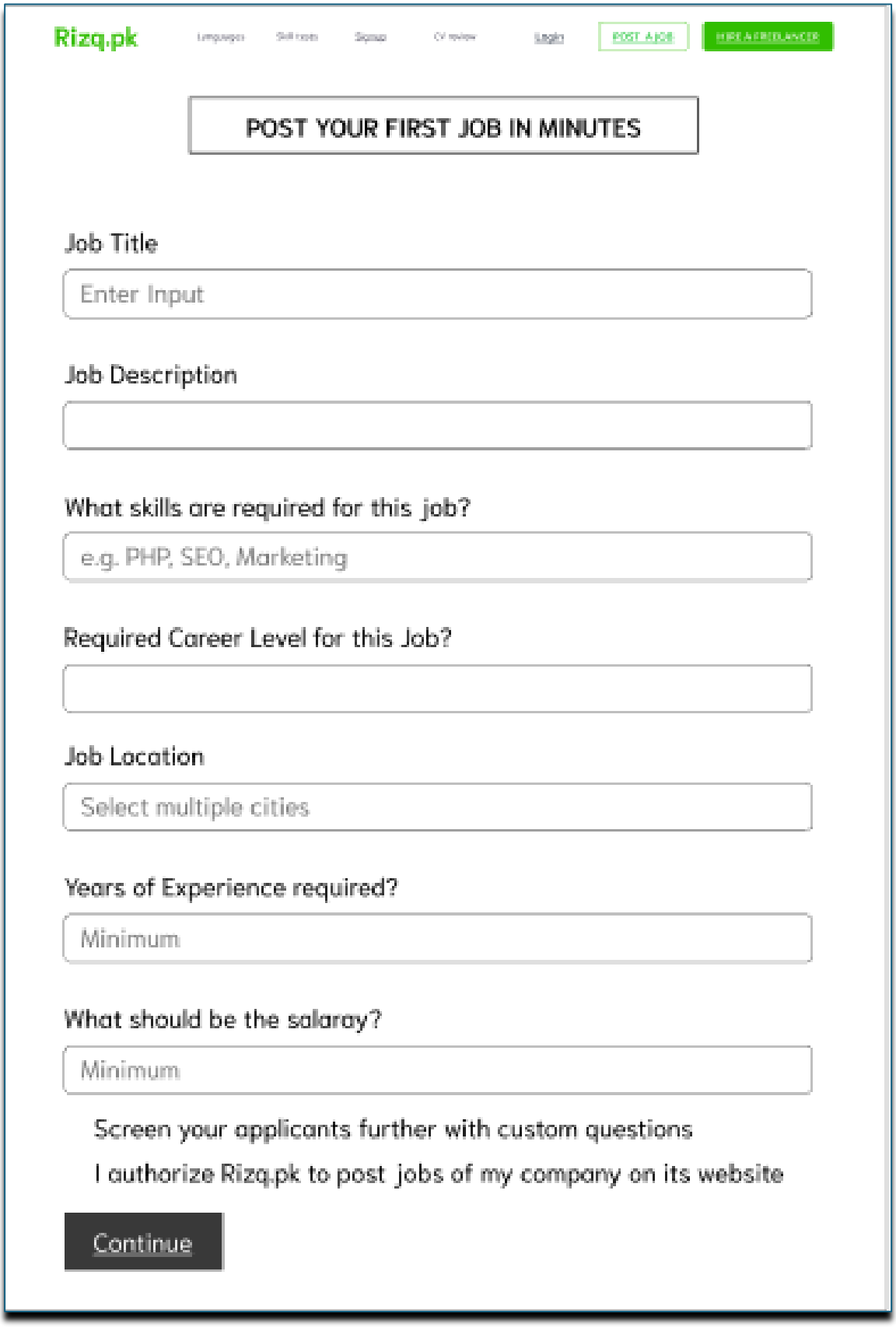
Email Address

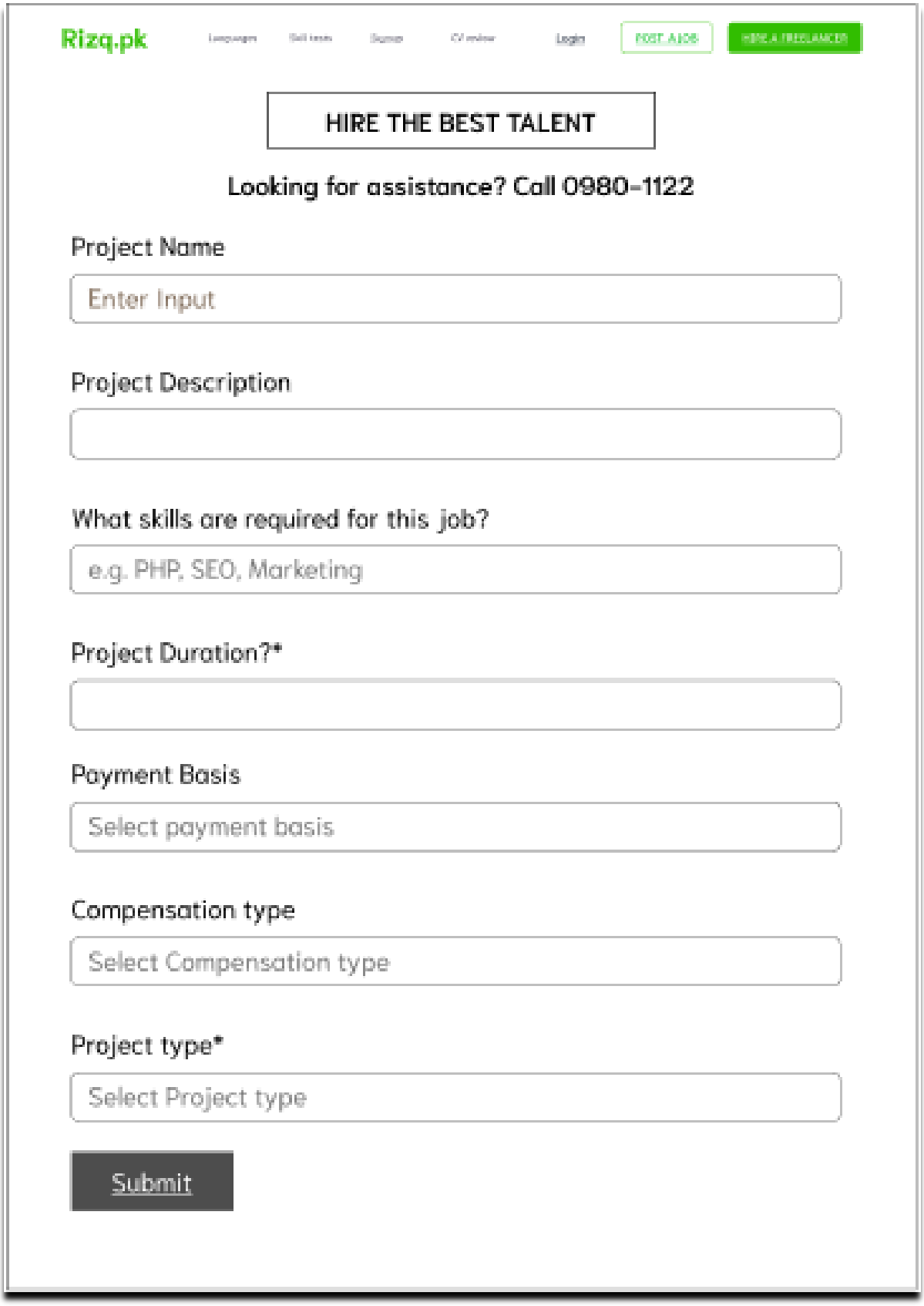
Password

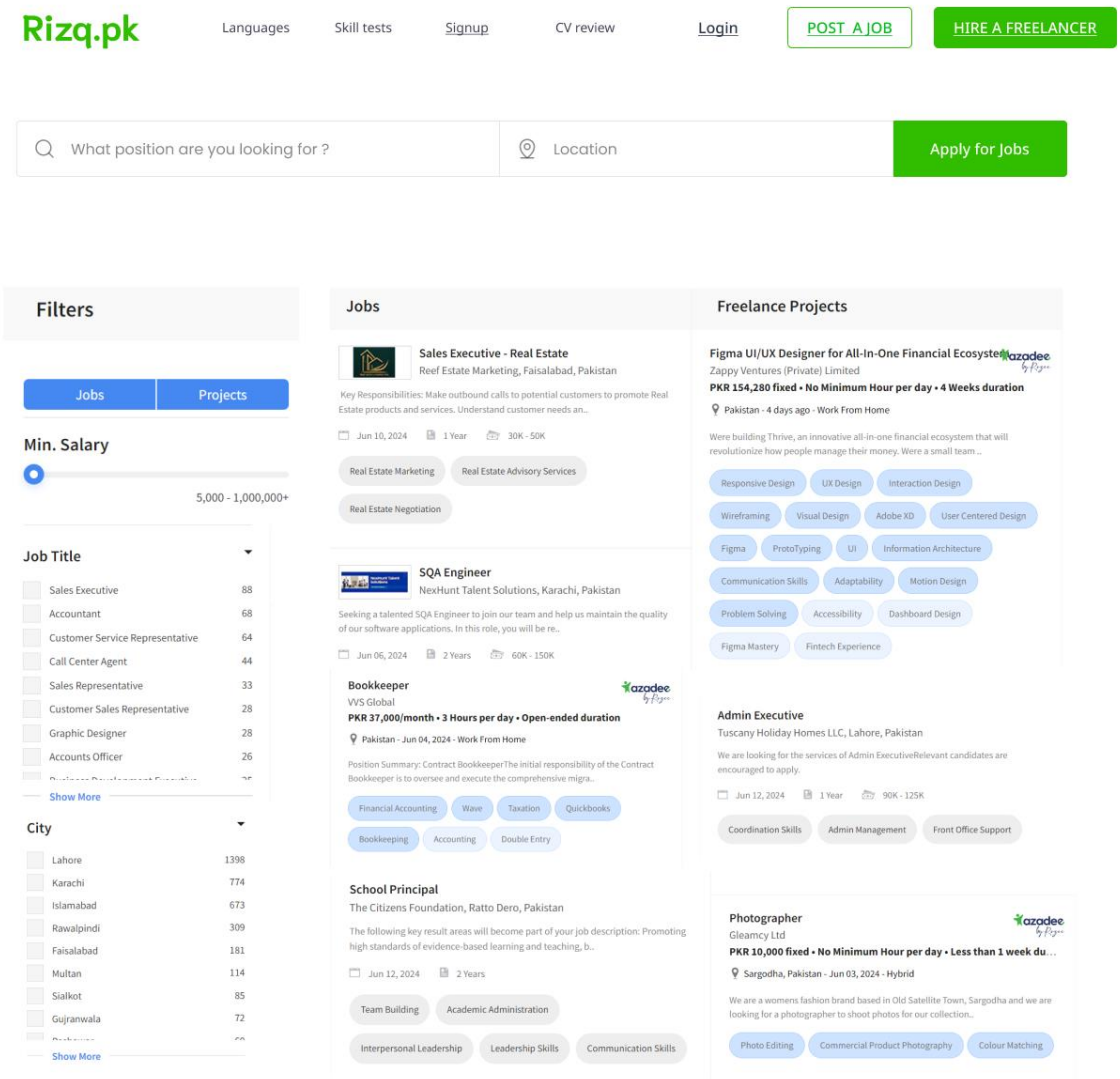
Login

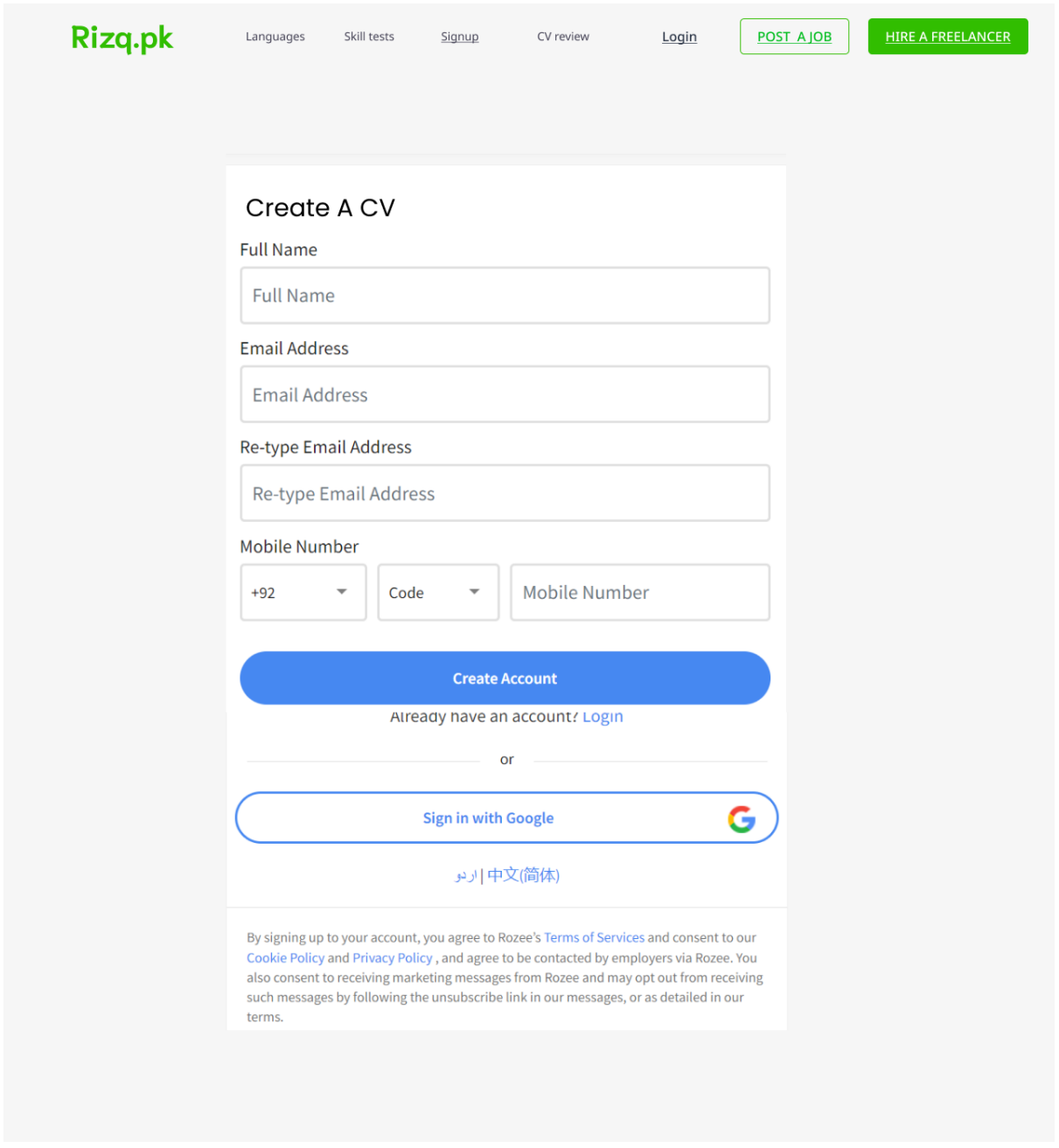
Don\'t have an account? Sign Up

Sign in with Google


Rizq.pk
Login
POST A JOB
HIRE A FREELANCER
POST YOUR FIRST JOB IN MINUTES
Job Title
Enter Input
Job Description
What skills are required for this job?
e.g. PHP, SEO, Marketing
Required Career Level for this Job?
Job Location
Select multiple cities
Years of Experience required?
Minimum
What should be the salaray?
Minimum
Screen your applicants further with custom questions
I authorize Rizq.pk to post jobs of my company on its website
Continue


Rizq.pk
Login
HIRE THE BEST TALENT
Looking for assistance? Call 0980-1122
Project Name
Enter Input
Project Description
What skills are required for this job?
e.g. PHP, SEO, Marketing
Project Duration?*
Payment Basis
Select payment basis
Compensation type
Select Compensation type
Project type*
Select Project type
Submit


Rizq.pk
Languages
Skill tests
Signup
CV review
Login
POST A JOB
HIRE A FREELANCER
What position are you looking for ?
Location
Apply for Jobs
Filters
Jobs
Projects
Min. Salary
5,000 - 1,000,000+
Job Title
Sales Executive 88
Accountant 68
Customer Service Representative 64
Call Center Agent 44
Sales Representative 33
Customer Sales Representative 28
Graphic Designer 28
Accounts Officer 26
Show More
City
Lahore 1398
Karachi 774
Islamabad 673
Rawalpindi 309
Faisalabad 181
Multan 114
Sialkot 85
Gujranwala 72
Show More
Jobs
Sales Executive - Real Estate
Reef Estate Marketing, Faisalabad, Pakistan
Key Responsibilities: Make outbound calls to potential customers to promote Real Estate products and services. Understand customer needs an..
Jun 10, 2024
1 Year
30K - 50K
Real Estate Marketing
Real Estate Advisory Services
Real Estate Negotiation
SQA Engineer
NexHunt Talent Solutions, Karachi, Pakistan
Seeking a talented SQA Engineer to join our team and help us maintain the quality of our software applications. In this role, you will be re..
Jun 06, 2024
2 Years
60K - 150K
Bookkeeper
VVS Global
PKR 37,000/month • 3 Hours per day • Open-ended duration
Pakistan - Jun 04, 2024 - Work From Home
Position Summary: Contract BookkeeperThe initial responsibility of the Contract Bookkeeper is to oversee and execute the comprehensive migra..
Financial Accounting
Wave
Taxation
Quickbooks
Bookkeeping
Accounting
Double Entry
School Principal
The Citizens Foundation, Ratto Dero, Pakistan
The following key result areas will become part of your job description: Promoting high standards of evidence-based learning and teaching, b..
Jun 12, 2024
2 Years
Team Building
Academic Administration
Interpersonal Leadership
Leadership Skills
Communication Skills
Freelance Projects
Figma UI/UX Designer for All-In-One Financial Ecosyste
Zappy Ventures (Private) Limited
PKR 154,280 fixed • No Minimum Hour per day • 4 Weeks duration
Pakistan - 4 days ago - Work From Home
Were building Thrive, an innovative all-in-one financial ecosystem that will revolutionize how people manage their money. Were a small team ..
Responsive Design
UX Design
Interaction Design
Wireframing
Visual Design
Adobe XD
User Centered Design
Figma
ProtoTyping
UI
Information Architecture
Communication Skills
Adaptability
Motion Design
Problem Solving
Accessibility
Dashboard Design
Figma Mastery
Fintech Experience
Admin Executive
Tuscany Holiday Homes LLC, Lahore, Pakistan
We are looking for the services of Admin ExecutiveRelevant candidates are encouraged to apply.
Jun 12, 2024
1 Year
90K - 125K
Coordination Skills
Admin Management
Front Office Support
Photographer
Gleamcy Ltd
PKR 10,000 fixed • No Minimum Hour per day • Less than 1 week du...
Sargodha, Pakistan - Jun 03, 2024 - Hybrid
We are a womens fashion brand based in Old Satellite Town, Sargodha and we are looking for a photographer to shoot photos for our collection..
Photo Editing
Commercial Product Photography
Colour Matching


Rizq.pk
Languages
Skill tests
Signup
CV review
Login
POST A JOB
HIRE A FREELANCER
Create A CV
Full Name
Full Name
Email Address
Email Address
Re-type Email Address
Re-type Email Address
Mobile Number
+92
Code
Mobile Number
Create Account
Already have an account? Login
or
Sign in with Google
اردو | 中文(简体)
By signing up to your account, you agree to Rozee's Terms of Services and consent to our Cookie Policy and Privacy Policy , and agree to be contacted by employers via Rozee. You also consent to receiving marketing messages from Rozee and may opt out from receiving such messages by following the unsubscribe link in our messages, or as detailed in our terms.