SRI KRISHNA COLLEGE OF TECHNOLOGY
(An Autonomous Institution)
Approved by AICTE | Affiliated to Anna University Chennai|
Accredited by NBA - AICTE| Accredited by NAAC with 'A' Grade
KOVAIPUDUR, COIMBATORE 641042

JOB SEARCHING PORTAL

A PROJECT REPORT

Submitted by

THILAGAVATHI K **727822TUCS247**

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

JULY 2024

BONAFIDE CERTIFICATE

Certified that this project report **JOB SERCHING PORTAL** is the bonafide work **THILAGAVATHI K 727822TUCS247** who carried out the project work under my supervision.

SIGNATURE	*SIGNATURE*
Ms. A. GOMATHY	**Dr. M. UDHAYAMURTHI**
SUPERVISOR	**HEAD OF THE DEPARTMENT**
Assistant Professor,	Associate Professor,
Department of Computer Science and Engineering,	Department of Computer Science and Engineering,
Sri Krishna College of Technology,Coimbatore-641042	Sri Krishna College of Technology,Coimbatore-641042

Certified that the candidate was examined by me in the Project Work Viva Voce examination held on_____at Sri Krishna College of Technology, Coimbatore-641042.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First and foremost we thank the **Almighty** for being our light and for showering his gracious blessings throughout the course of this project.

We express our gratitude to our beloved Principal, **Dr. M.G. Sumithra,** for providing all facilities.

We are grateful to our beloved Head, Computing Sciences **Dr.T. Senthilnathan,** for her tireless and relentless support.

With the grateful heart, our sincere thanks to our Head of the Department **Dr. M. Udhayamurthi**, Department of Computer Science and Engineering for the motivation and all support to complete the project work.

We thank **Mr. A. Gomathy**, Assistant Professor, Department of Computer Science and Engineering, for his motivation and support.

We are thankful to all the **Teaching and Non-Teaching Staff** of Department of Computer Science and Engineering and to all those who have directly and indirectly extended their help to us in completing this project work successfully.

We extend our sincere thanks to our **family members** and our beloved **friends**, who had been strongly supporting us in all our endeavour.

ABSTRACT

The Job Searching Portal leverages advanced web technologies like React JS, Spring-Boot, REST API, and MySQL to create a comprehensive platform that revolutionizes the job search and recruitment process. It offers a robust, user-friendly interface enabling efficient job posting and application management based on user roles and preferences. Key features include role-based access control, advanced job search filters, application tracking, and real-time notifications. A single login form for both job seekers and employers ensures secure and appropriate access to different functionalities within the portal, redirecting users to the appropriate dashboard based on their role. Employers can post detailed job listings with descriptions, requirements, and deadlines, while job seekers can search for jobs using various filters like location, industry, job type, and salary range. Job seekers can apply for jobs, upload resumes, and track application statuses, while employers can review applications, shortlist candidates, and manage interviews. The comprehensive admin dashboard features cards for Total Users, Active Users, Total Orders, Revenue, Total Jobs Posted, Active Job Listings, Total Applications, and Hired Candidates, and includes tools for managing job listings, user accounts, and site settings. A responsive design ensures an optimal user experience on both mobile and desktop devices, utilizing modern frontend technologies to create a fluid and interactive interface. A frontend React-based chatbot assists users with common queries and guides them through the portal's features, enhancing user engagement and providing instant support. Role management is streamlined by managing users and admins within a single database table differentiated by roles, simplifying database design and enhancing maintainability.

TABLE OF CONTENT

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This project aims to offer a seamless and efficient solution for job seekers and employers to connect through an online platform. In this chapter, we will explore the problem statement, provide an overview, and outline the main objectives of the job searching portal.

1.1 PROBLEM STATEMENT

How can we develop a job searching portal that allows job seekers to efficiently find and apply for jobs, and employers to post and manage job listings, taking into account various filters, application statuses, and preferences, while ensuring an intuitive user interface and maximizing user engagement?

1.2 OVERVIEW

In the realm of job searching, job seekers and employers often face challenges such as finding suitable job matches, managing applications, and ensuring efficient communication. Traditional methods can be time-consuming and prone to errors, leading to inefficiencies and dissatisfaction among users. To address these issues, we propose the creation of a Job Searching Portal. This system will leverage modern web technologies to provide a robust, user-friendly platform that streamlines the job search and recruitment process. By incorporating advanced job search filters, application management, and real-time notifications, the system aims to enhance operational efficiency and user satisfaction.

1.3 OBJECTIVE

The primary objective of this project is to develop a Job Searching Portal that provides job seekers and employers with a user-friendly and efficient platform for connecting and managing job listings and applications. The system aims to streamline the job search and recruitment process, enhance user engagement, and improve communication by leveraging advanced job search filters, real-time notifications, and comprehensive application management.

CHAPTER 2
SYSTEM SPECIFICATION

In this chapter, we are going to see the software that we have used to build the website. This chapter gives you a small description about the software used in the project.

2.1 VS CODE

Visual Studio Code is a source code editor developed by Microsoft for Windows, Linux, and macOS. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is also customizable, so users can change the editor's theme, keyboard shortcuts, and preferences.

VS Code is an excellent code editor for React projects. It is lightweight, customizable, and has a wide range of features that make it ideal for React development. It has built-in support for JavaScript, JSX, and TypeScript, and enables developers to quickly move between files and view detailed type definitions. It also has a built-in terminal for running tasks. Additionally, VS Code has an extensive library of extensions that allow developers to quickly add features like code snippets, debugging tools, and linting supportto their projects.

2.2 LOCAL STORAGE

Local storage is a type of web storage for storing data on the client side of a web browser. It allows websites to store data on a user's computer, which can then be accessed by the website again when the user returns. Local storage is a more secure alternative to cookies because it allows websites to store data without having to send it back and forth with each request. Local storage is a key-value pair storage mechanism, meaning it stores data in the form of a key and corresponding value.

It is also used to cache data to improve the performance of a website. Local storage is supported by all modern web browsers, including Chrome,

Firefox, Safari, and Edge. It is accessible through the browser's JavaScript API. Local storage is a powerful tool for websites to store data on the client side. It is secure, efficient, and can be used to store data that does not need to be shared with other websites.

Local Storage is a great way to improve the performance of a website by caching data. Local storage in web browsers allows website data to be stored locally on the user's computer. It is a way of persistently storing data on the client side, which is not sent to the server with each request. This allows users to store data such as preferences, login information, and form data without needing to send it to a server. It is typically stored in a browser's cookie file, but it can also be stored in other locations such as HTML5 Local Storage and Indexed DB. The data stored in local storage is persistent and can be accessed by the website even if the user closes the browser or navigates to another page. It is a great way for websites to store user-specific data, as it is secure, reliable, and fast. It is also a great way for developers to store data that does not need to be sent to the server with each request.

One of the key benefits of using local storage is its reliability. Unlike server-side storage, which can be affected by network outages or other server issues, local storage is stored locally on the user's machine, and so is not affected by these issues. Another advantage of local storage is its speed. Because the data is stored locally, it is accessed quickly, as there is no need to send requests to a server. This makes it ideal for storing data that needs to be accessed quickly, such as user preferences or session data. Local storage is also secure, as the data is stored on the user's machine and not on a server. This means that the data is not accessible by anyone other than the user, making it a good choice for storing sensitive information.

CHAPTER 3

PROPOSED SYSTEM

This chapter gives a small description about the proposed idea behind the development of our website

3.1 Proposed System

This system offers a multitude of benefits from various perspectives. The online job searching platform empowers job seekers and employers to connect efficiently, taking into account job requirements, applicant qualifications, and preferences. The system simplifies the job search and recruitment process, enabling employers to create, modify, and manage job listings with ease, while job seekers can browse, apply for jobs, and track their application status online.

Once a job listing is created, it is directed to the designated personnel responsible for overseeing the recruitment process. The administrative team ensures that listings are accurate and up-to-date, with specialized staff members handling any necessary adjustments or application reviews within a specified timeframe. Confirmed listings are updated in the system and are accessible to all relevant users.

This system significantly reduces the administrative workload and streamlines operations in job searching and recruitment. Especially during peak hiring periods or when dealing with numerous applications, where manual processes may lead to delays or errors, the online job searching platform mitigates such challenges. Job seekers can directly apply for positions and track their applications online, bypassing potential bottlenecks such as limited office hours or manual processing delays.

Moreover, the system enhances operational efficiency, enabling employers to manage job listings promptly and effectively. By leveraging technology to automate data entry, application tracking, and notifications, the online job searching platform ensures swift and accurate processing, resulting in improved user satisfaction and efficiency in the job search and recruitment process.

3.1 ADVANTAGES

Efficiency: The job searching portal allows employers to create, modify, and manage job listings quickly and easily from anywhere with internet access. This eliminates the need for time-consuming manual processes, reducing administrative workload and saving time for both employers and job seekers.

Flexibility: Job seekers can browse job listings, apply for positions, and track their application status online at their convenience. This flexibility helps accommodate varying schedules and preferences, improving user satisfaction and allowing for a more accurate and efficient job search process.

Conflict Resolution: The system automatically detects and resolves application conflicts, ensuring that job listings are updated and managed without overlap or errors. This reduces the chances of duplicate applications and conflicting information, leading to smoother operations.

Accessibility: Employers and job seekers can access the job searching portal from any location with internet access. This accessibility ensures that job listings and applications can be managed and viewed remotely, facilitating better coordination and planning.

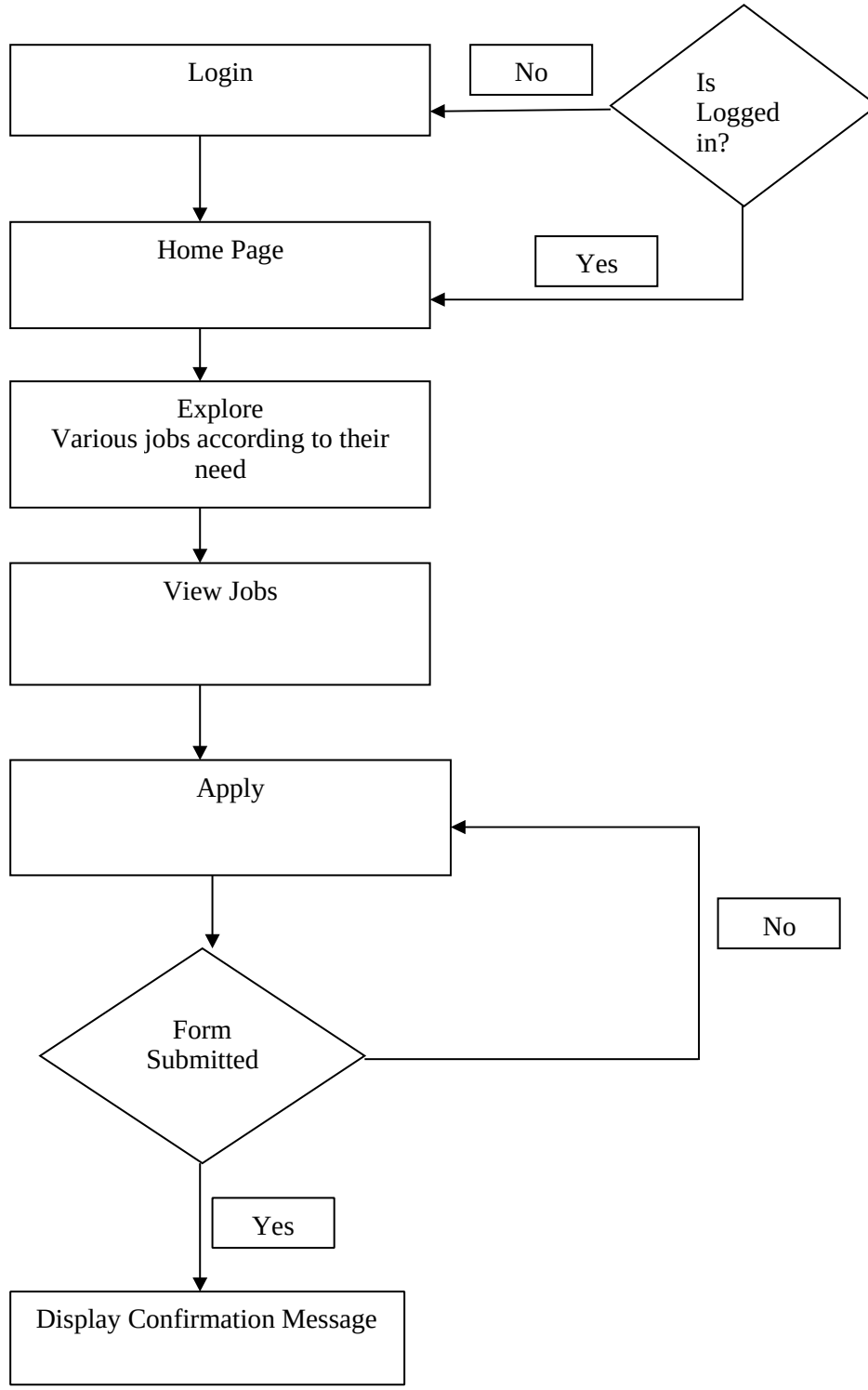
Transparency: The system provides clear and detailed information about job listings, application statuses, and recruitment processes. This transparency helps employers and job seekers stay informed about their activities and responsibilities, reducing confusion and enhancing communication.

Real-Time Notifications: Automated notifications keep job seekers updated on application statuses, new job postings, and important reminders. This feature ensures that everyone is informed in real-time, improving coordination and reducing the risk of missed opportunities.

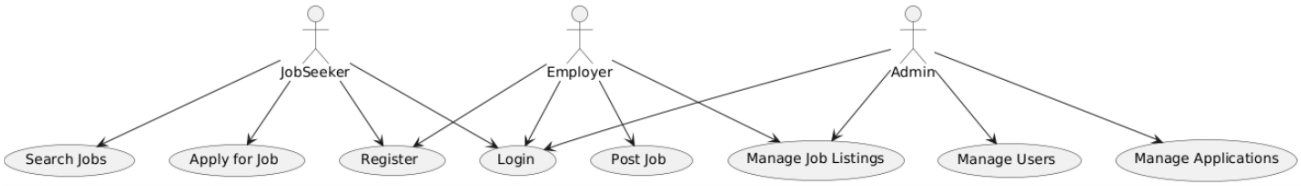
CHAPTER 4

METHODOLOGIES

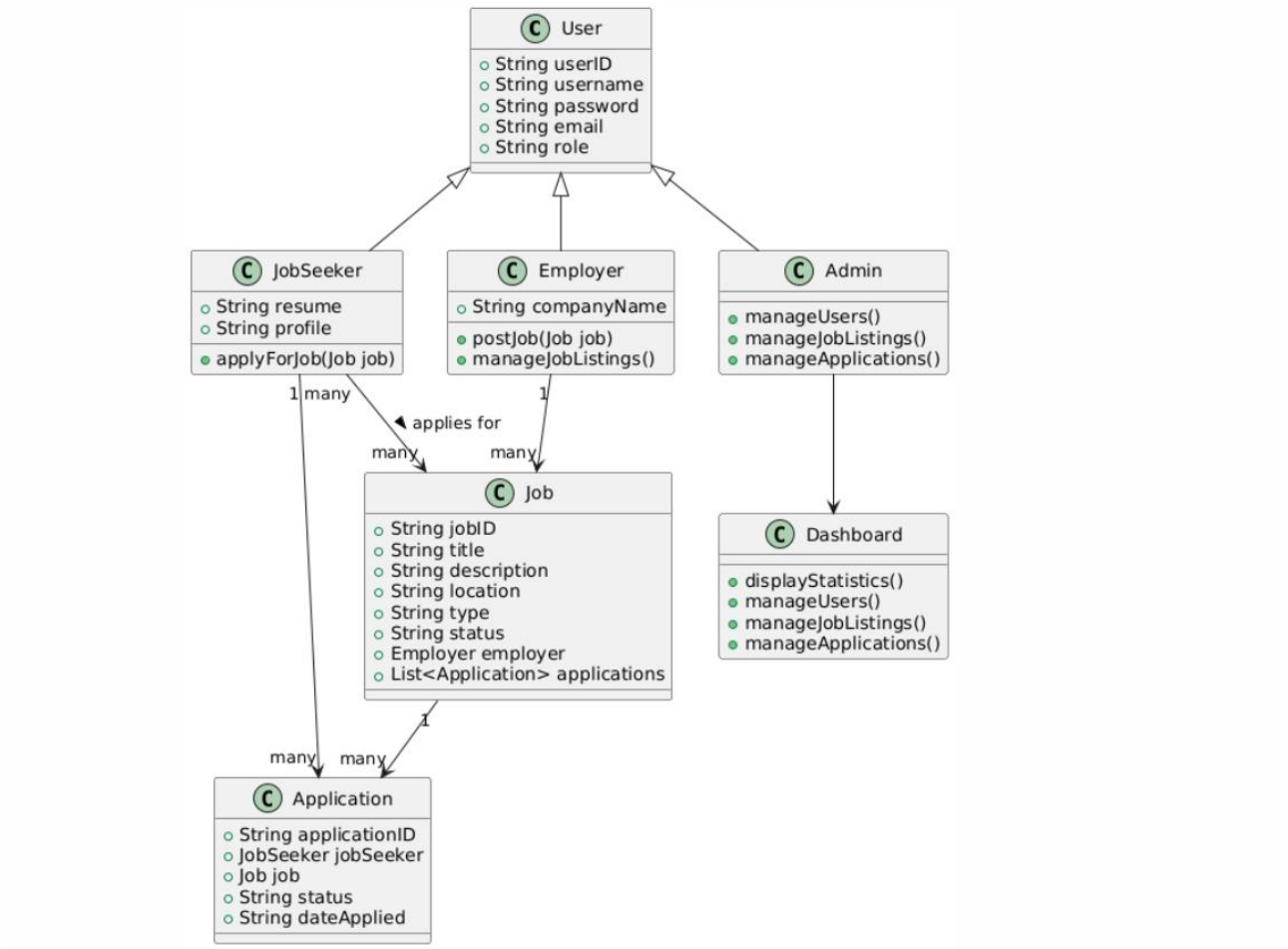
4.1 FLOW CHART



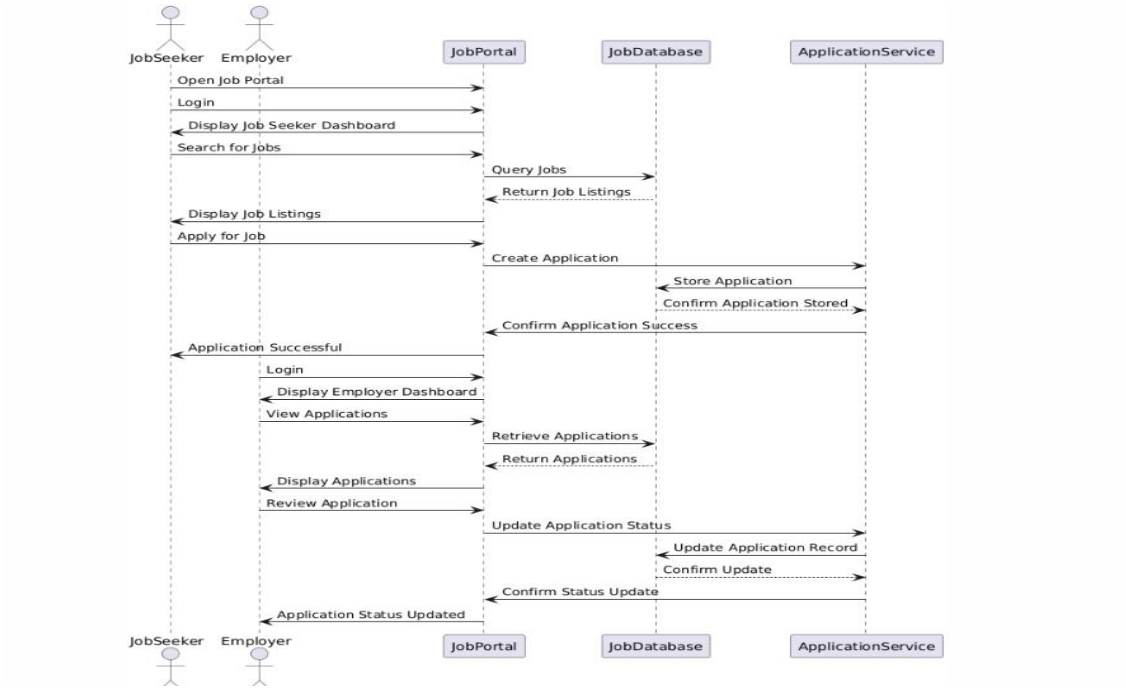
4.2 Use Case Diagram:



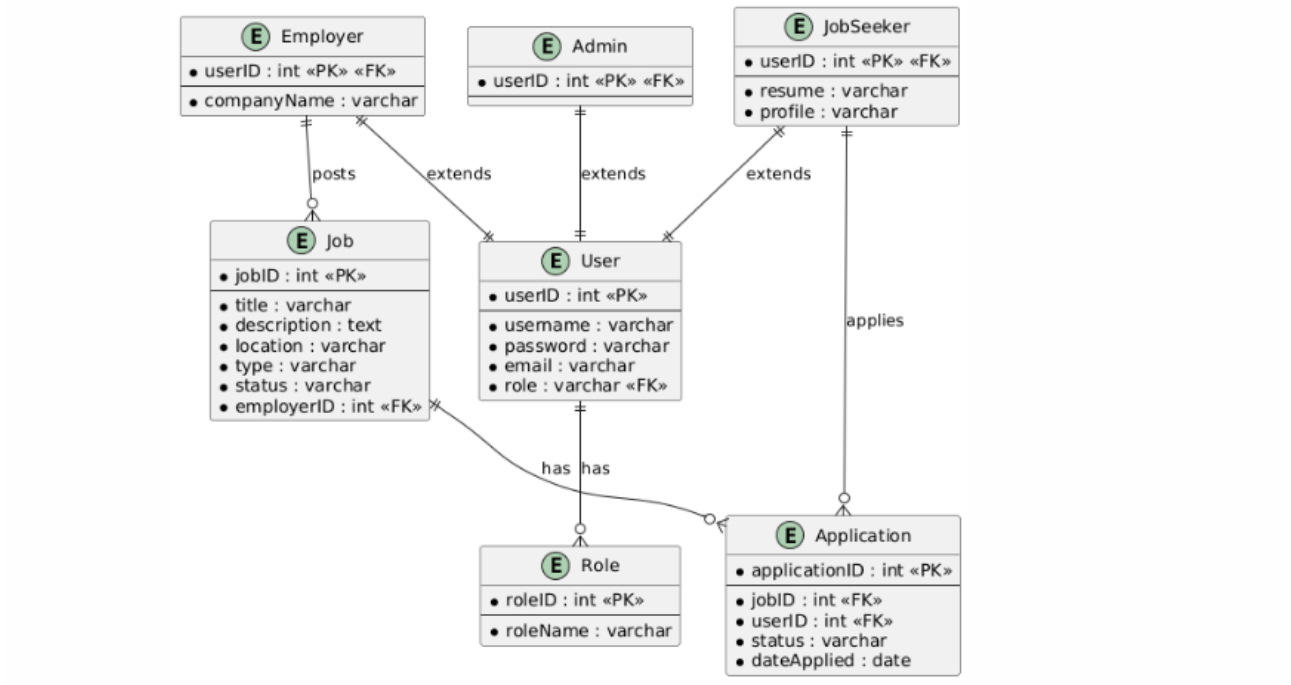
4.3 Class Diagram:



4.4 Sequence Diagram:



4.5 ER Diagram:

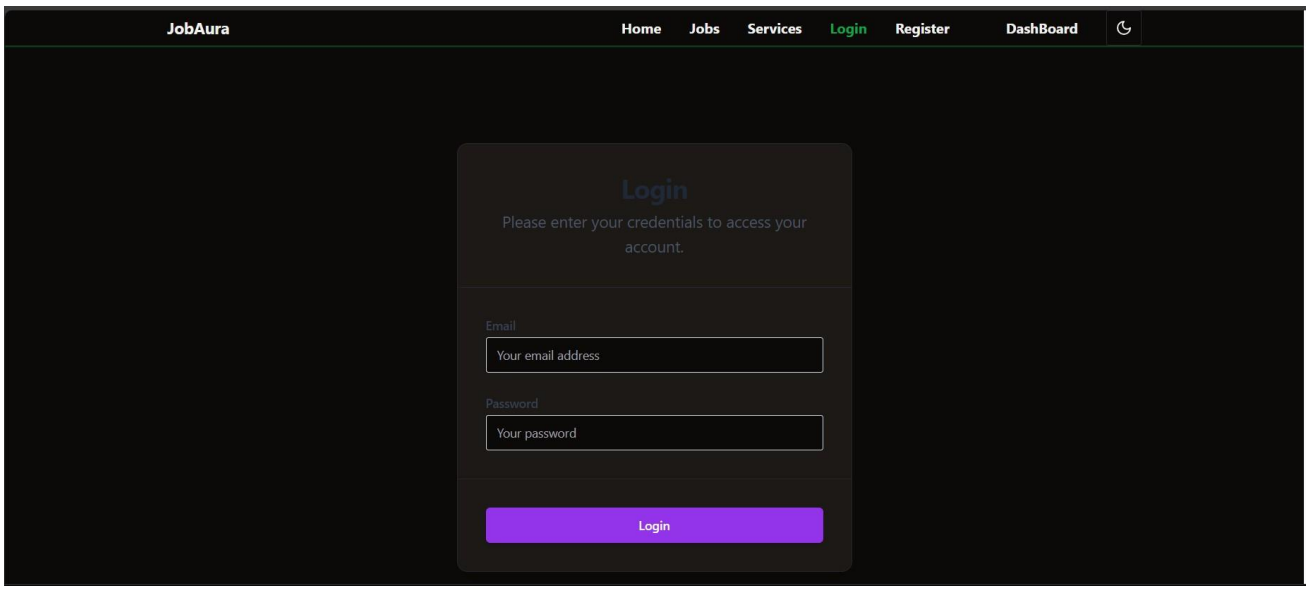


CHAPTER 5

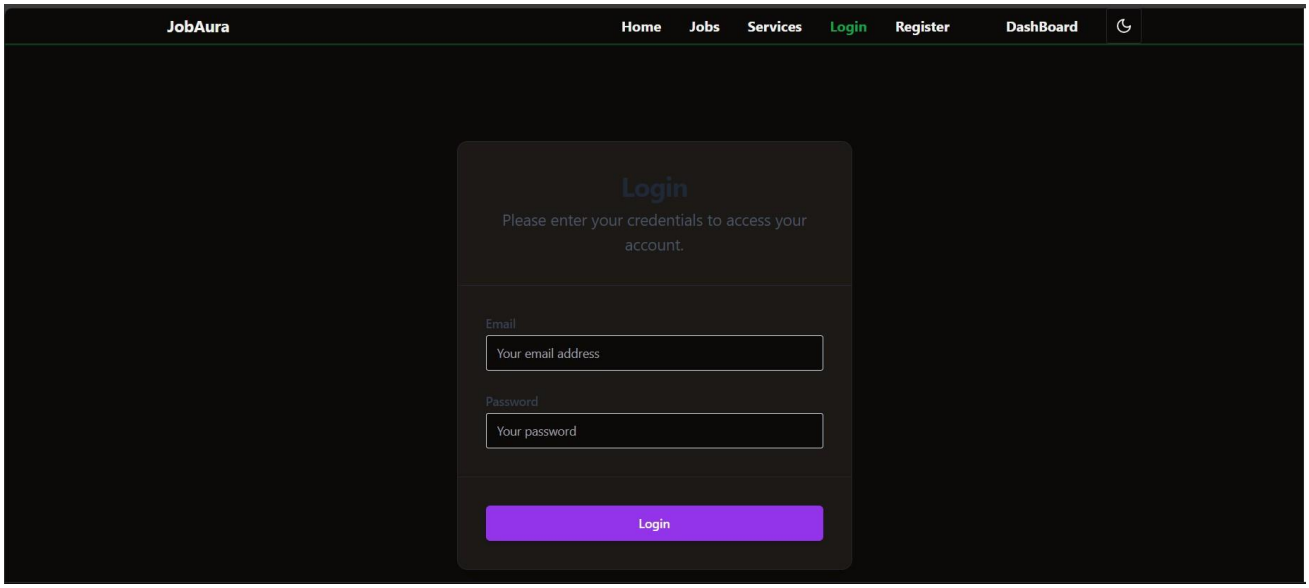
IMPLEMENTATION AND RESULT

This chapter gives a description about the output that we produced by developing the website of our idea.

5.1 LOGIN

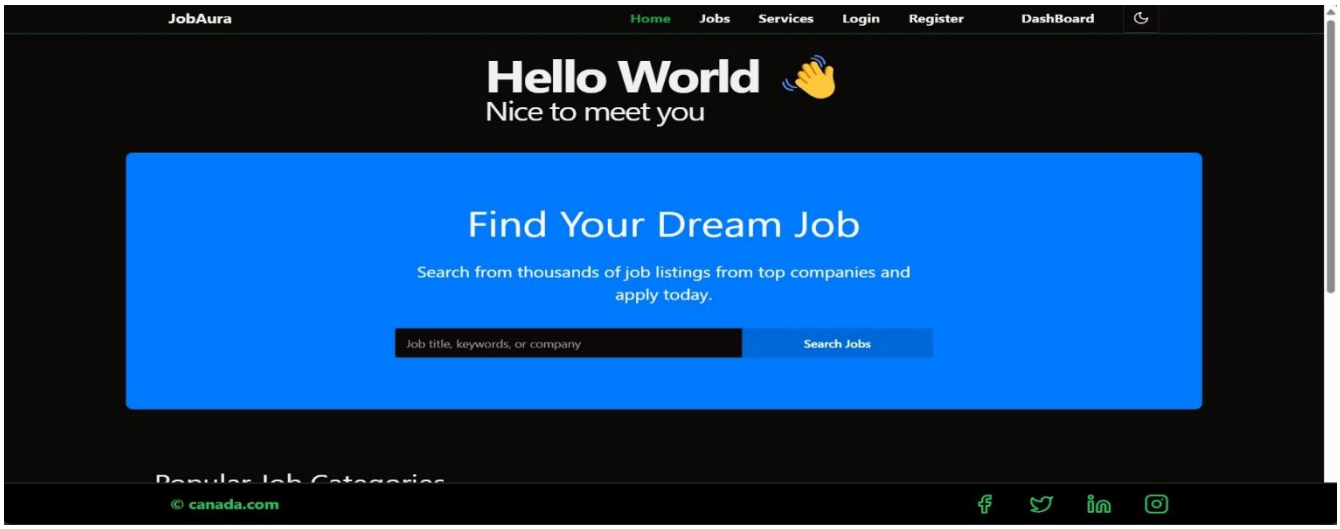


5.2 REGISTER

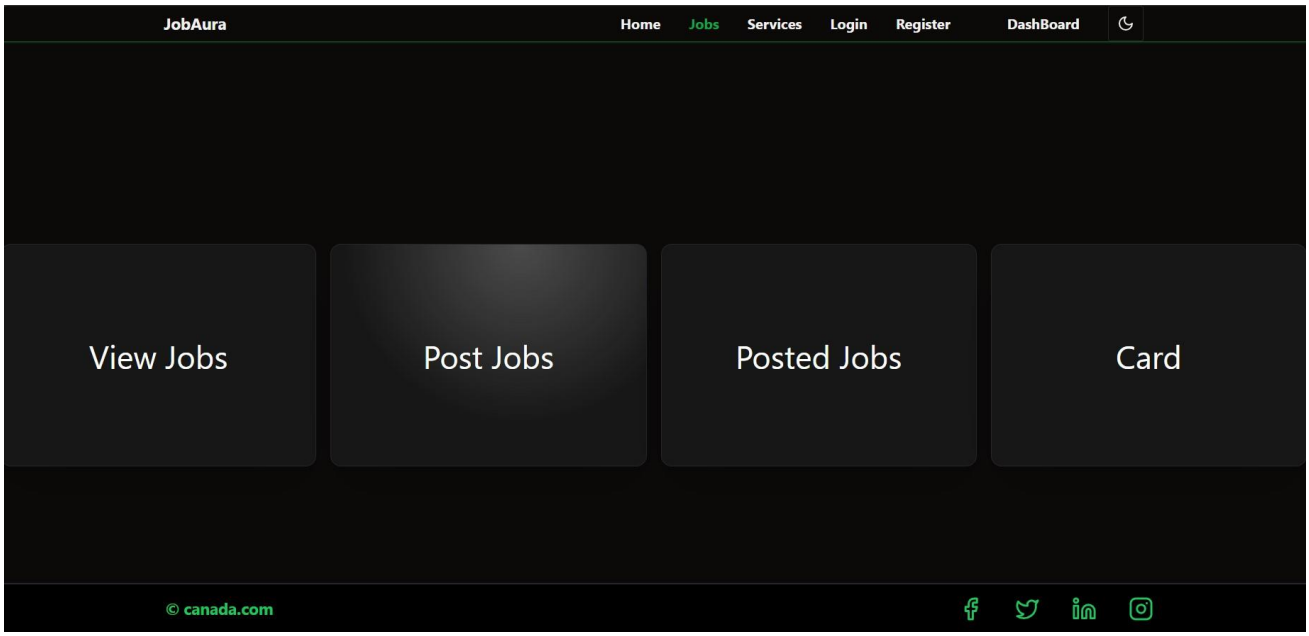


5.3 USER NAVIGATION

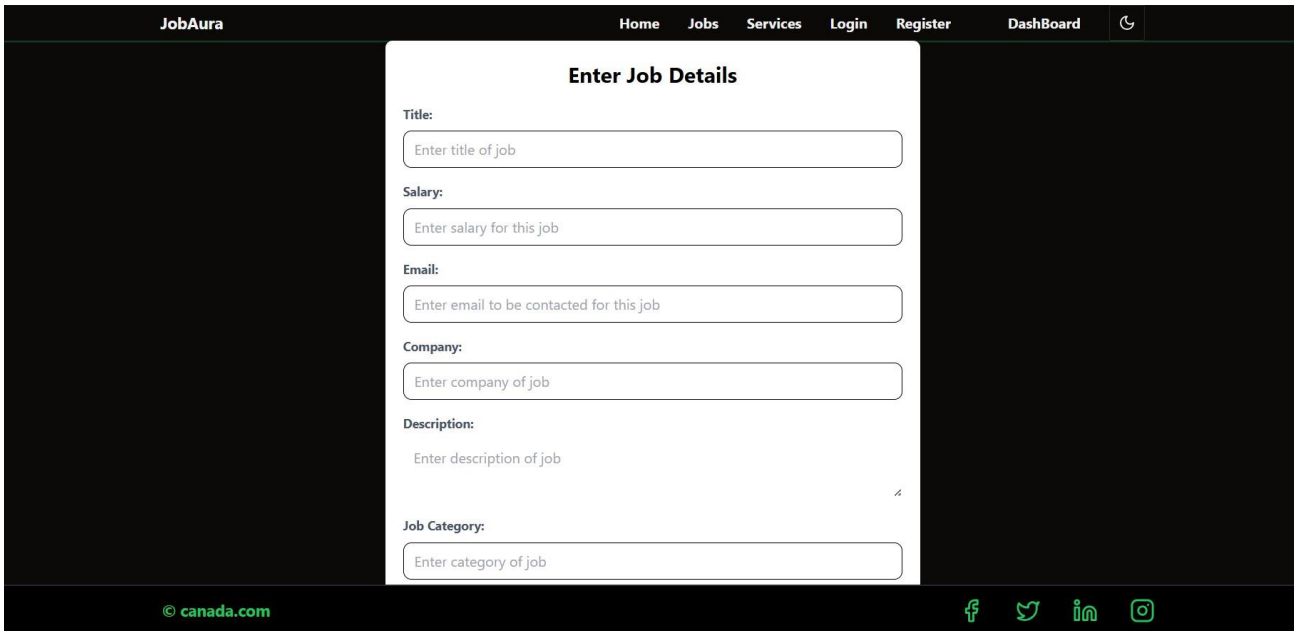
5.4 HOME PAGE



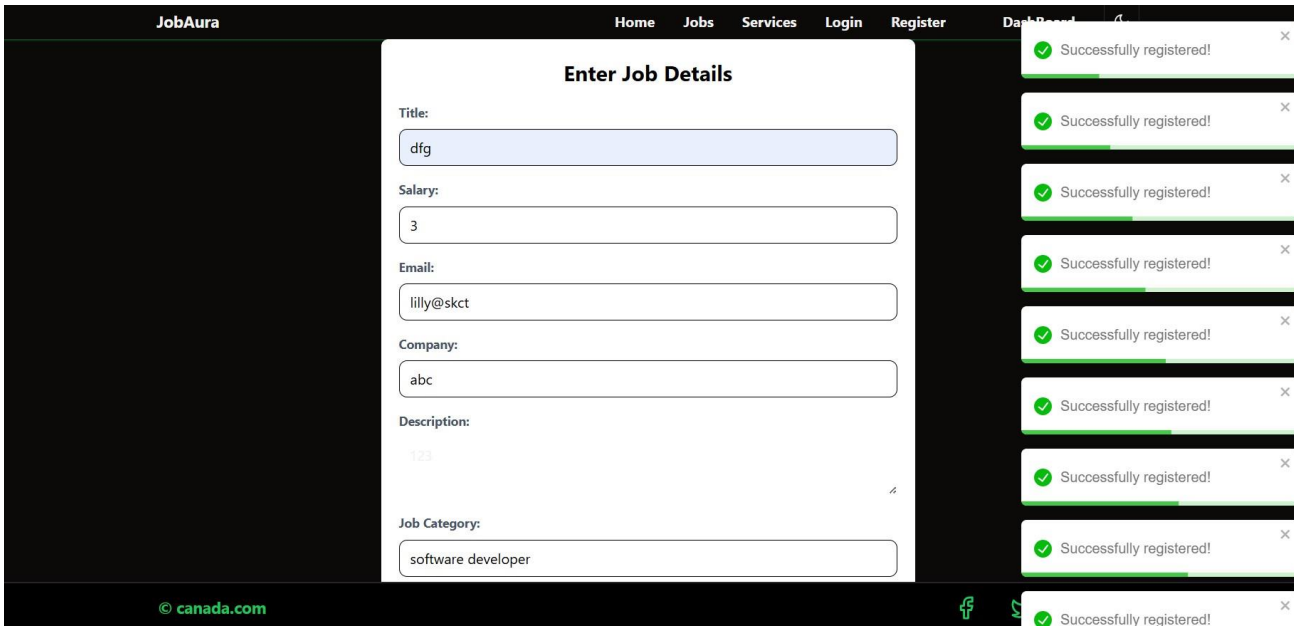
5.5. Jobs Page



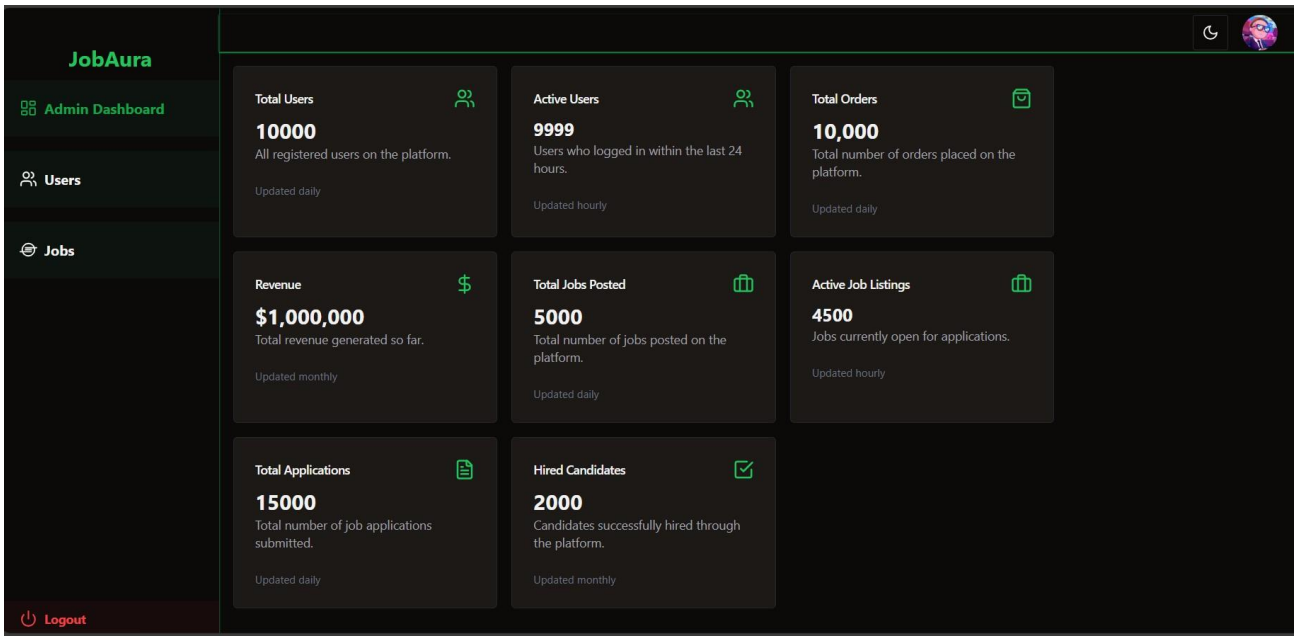
5.6 APPLY JOBS



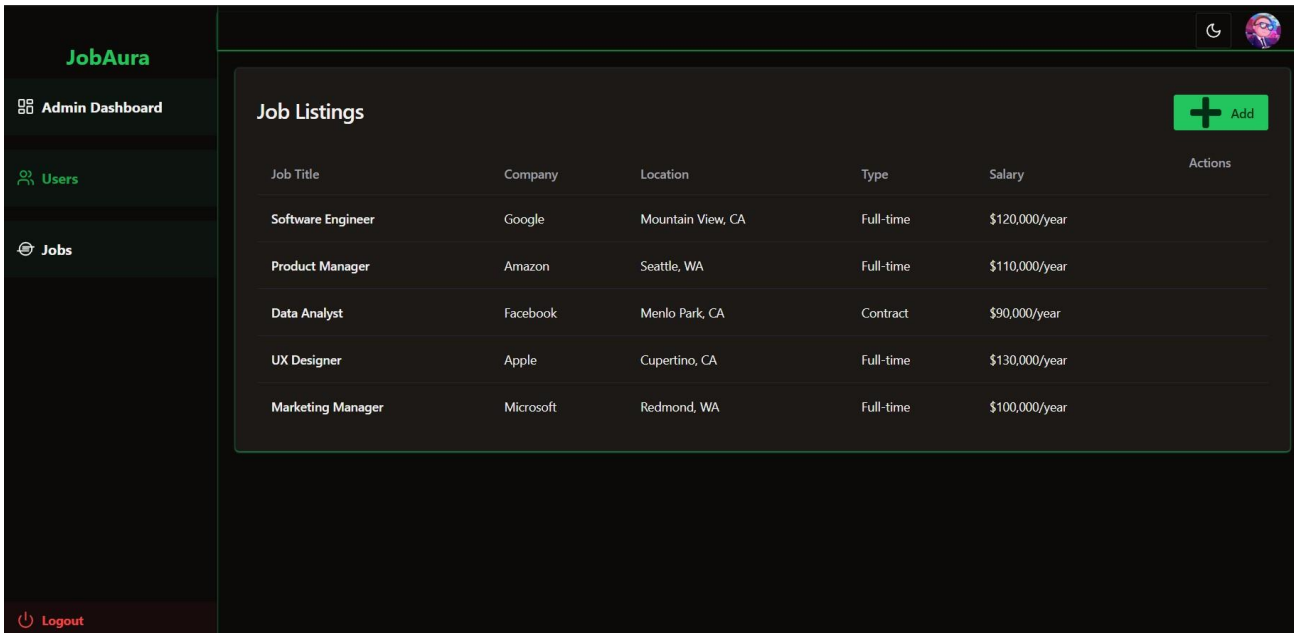
5.7 CONFIRMATION MESSAGE



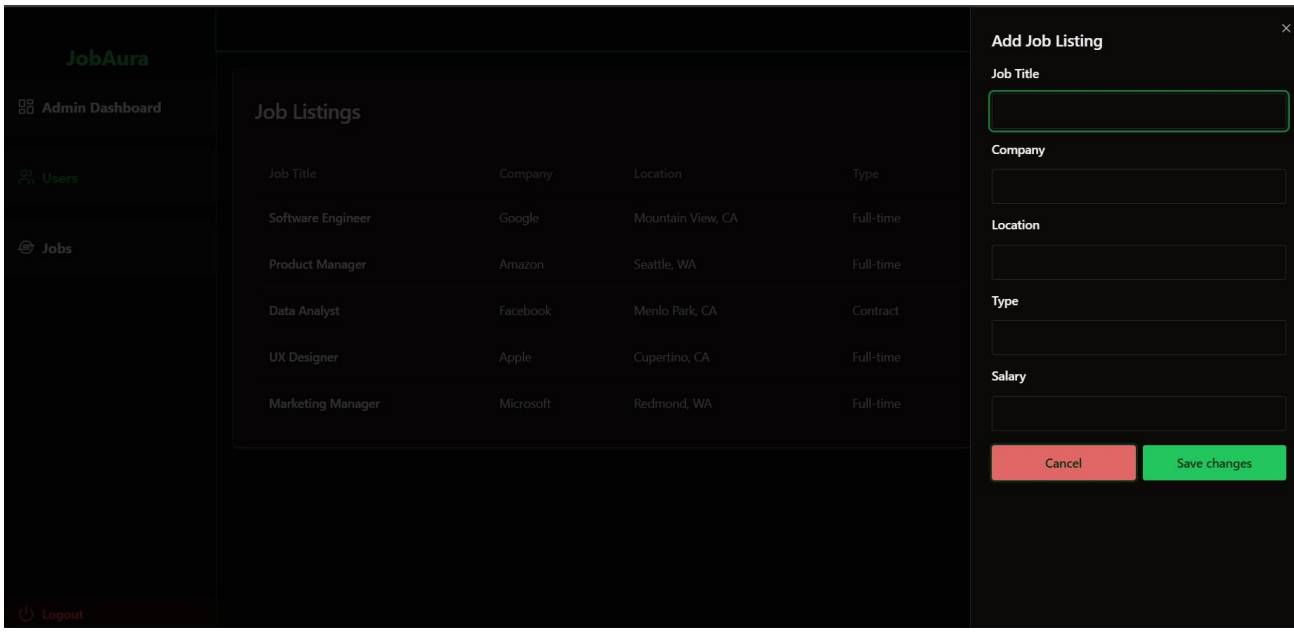
5.8 ADMIN DASHBOARD



5.9 USERS PAGE



5.10 ADD USERS



5.11 FOOTER

Login:

```
const Login = () => {
   const [email, setEmail] = useState('');
   const [password, setPassword] = useState('');
   const [errors, setErrors] = useState({});
   const navigate = useNavigate();

   const validate = () => {
      const newErrors = {};
      if (!email) {
         newErrors.email = 'Email is required';
      } else if (!/\S+@\S+\.\S+/.test(email)) {
         newErrors.email = 'Email address is invalid';
      }
      if (!password) {
         newErrors.password = 'Password is required';
      } else if (password.length < 6) {
         newErrors.password = 'Password must be at least 6 characters';
      }
      return newErrors;
   };

   const handleSubmit = (e) => {
      e.preventDefault();
      const newErrors = validate();
      if (Object.keys(newErrors).length === 0) {
         const existingUsers = [
            { email: 'iam@example.com', password: 'password123' },
         ];
         const adminUsers = [
            { email: 'admin@example.com', password: 'admin123' },
         ];
         const user = existingUsers.find(user => user.email === email && user.password ===
password);
         const admin = adminUsers.find(admin => admin.email === email && admin.password
=== password);

         if (user) {
            toast.success('Login successful!');
            setTimeout(() => {
               navigate('/');
            }, 2000);
         } else if (admin) {
            toast.success('Login successful!');
            setTimeout(() => {
               navigate('/admin/dashboard');
```

```
            }, 2000);
        } else {
            toast.error('Incorrect credentials. Please try again.');
        }
    } else {
        setErrors(newErrors);
    }
};

return (
    <div className='min-h-screen flex items-center justify-center'>
        <ToastContainer />
        <Card className="w-full max-w-md shadow-xl rounded-xl">
            <CardHeader className="p-8 text-center border-b">
                <CardTitle className="text-3xl font-bold text-gray-800">Login</CardTitle>
                <CardDescription className="mt-2 text-lg text-gray-600">
                    Please enter your credentials to access your account.
                </CardDescription>
            </CardHeader>
            <form onSubmit={handleSubmit}>
                <CardContent className="p-8">
                    <div className="space-y-6">
                        <div>
                            <Label htmlFor="email" className="block text-sm text-gray-
700">Email</Label>
                            <Input
                                id="email"
                                type="email"
                                value={email}
                                onChange={(e) => setEmail(e.target.value)}
                                placeholder="Your email address"
                                className={`mt-1 block w-full p-3 border ${errors.email ? 'border-red-
500' : 'border-gray-300'} rounded-md shadow-sm focus:ring-2 focus:ring-purple-500`}
                            />
                            {errors.email && <p className="text-red-500 text-sm mt-
1">{errors.email}</p>}
                        </div>
                        <div>
                            <Label htmlFor="password" className="block text-sm font-medium text-
gray-700">Password</Label>
                            <Input
                                id="password"
                                type="password"
                                value={password}
                                onChange={(e) => setPassword(e.target.value)}
                                placeholder="Your password"
                                className={`mt-1 block w-full p-3 border ${errors.password ? 'border-
red-500' : 'border-gray-300'} rounded-md shadow-sm focus:ring-2 focus:ring-purple-500`}
```

```
                        />
                        {errors.password && <p className="text-red-500 text-sm mt-
1">{errors.password}</p>}
                    </div>
                </div>
            </CardContent>
            <CardFooter className="p-8 text-center border-t">
                <Button type="submit" className="w-full py-3 bg-purple-600 text-white font-
semibold rounded-lg shadow-md hover:bg-purple-700 focus:outline-none focus:ring-2
focus:ring-purple-500">
                    Login
                </Button>
            </CardFooter>
        </form>
    </Card>

    </div>
  );
};

export default Login;
```

Home:

```
const jobCategories = [
  "Software Development",
  "Marketing",
  "Finance",
  "Education",
  "Customer Service",
  "Engineering",
  "IT Support",
  "Project Management",
  "Data Science",
  "Hospitality",
  "Retail",
  "Construction",
];

const jobs = [
  { title: "Frontend Developer", category: "Software Development" },
  { title: "Marketing Specialist", category: "Marketing" },
  { title: "Accountant", category: "Finance" },
  { title: "Teacher", category: "Education" },
  { title: "Customer Support Representative", category: "Customer Service" },
```

```
  { title: "Mechanical Engineer", category: "Engineering" },
  { title: "IT Support Specialist", category: "IT Support" },
  { title: "Project Manager", category: "Project Management" },
  { title: "Data Scientist", category: "Data Science" },
  { title: "Hotel Manager", category: "Hospitality" },
  { title: "Store Manager", category: "Retail" },
  { title: "Construction Worker", category: "Construction" },
];

const Home = () => {
  const [selectedCategories, setSelectedCategories] = useState([]);
  const [filteredJobs, setFilteredJobs] = useState(jobs);

  useEffect(() => {
    if (selectedCategories.length === 0) {
      setFilteredJobs(jobs);
    } else {
      setFilteredJobs(
        jobs.filter((job) =>
          selectedCategories.includes(job.category)
        )
      );
    }
  }, [selectedCategories]);

  const handleCategoryChange = (category) => {
    setSelectedCategories((prevSelected) =>
      prevSelected.includes(category)
        ? prevSelected.filter((c) => c !== category)
        : [...prevSelected, category]
    );
  };
```

JOB APPLY

```
const JobForm = () => {
  const [formData, setFormData] = useState({
    title: '',
    salary: '',
    email: '',
    company: '',
    jobCategory: '',
    jobType: '',
  });

  const handleChange = (e) => {
```

```
  const { name, value } = e.target;
  setFormData({
    ...formData,
    [name]: value
  });
};

const handleSubmit = (e) => {
  e.preventDefault();
  const missingFields = Object.entries(formData).filter(([key, value]) => value.trim() === '');

  if (missingFields.length > 0) {
    toast.error('Please fill out all fields before submitting.');
    return;
  }

  console.log(formData);
  toast.success('Successfully registered! Waiting for confirmation.');
};
```

Config:

```
@Configuration
@RequiredArgsConstructor
public class ApplicationConfig {

    private final UserRepo userRepo;

    @Bean
    public UserDetailsService userDetailsService(){
        return  username -> userRepo.findByEmail(username)
                .orElseThrow(() -> new UsernameNotFoundException("User not Found"));
    }

    @Bean
    public AuthenticationProvider authenticationProvider(){

        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService());
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config)
throws Exception {
        return config.getAuthenticationManager();
```

```
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}


@Configuration
public class CorsConfig {

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**")
                        .allowedOrigins("localhost:5173")
                        .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
                        .allowedHeaders("*")
                        .allowCredentials(true);
            }
        };
    }
}



@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {
    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(
            @NonNull HttpServletRequest request,
            @NonNull HttpServletResponse response,
            @NonNull FilterChain filterChain) throws ServletException, IOException {
        final String authHeader =request.getHeader("Authorization");
        final String jwtToken;
        final String userEmail;

        if(authHeader == null || !authHeader.startsWith("Bearer "))
        {
            filterChain.doFilter(request,response);
            return;
```

```
        }

        jwtToken = authHeader.substring(7);
        userEmail = jwtService.extractUserName(jwtToken);
        if(userEmail != null && SecurityContextHolder.getContext().getAuthentication() == null){
            UserDetails userDetails = this.userDetailsService.loadUserByUsername(userEmail);
            if(jwtService.isTokenValid(jwtToken, userDetails)){
                UsernamePasswordAuthenticationToken authToken = new
UsernamePasswordAuthenticationToken(
                        userDetails,
                        null,
                        userDetails.getAuthorities()
                );
                authToken.setDetails(
                        new WebAuthenticationDetailsSource().buildDetails(request)
                );
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }
        filterChain.doFilter(request,response);


    }
}

@Service
public class JwtService {

    private static final String SECRET_KEY =
"EbeEsh7VhXpHMAkLz7Xb3TYm7a4KLMlYn0Kr1NJEhTIOeU9HJsv3t2bMa5OjoiaD";

    public String extractUserName(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver){
        final Claims claims  = extractAllClaims(token);
        return  claimsResolver.apply(claims);
    }

    public String generateToken(UserDetails userDetails){
        return generateToken(new HashMap<>(), userDetails);
    }

    public String generateToken(
            Map<String, Object> extraClaims,
            UserDetails userDetails
    ){
```

```
        return Jwts
                .builder()
                .claims(extraClaims).
                subject(userDetails.getUsername())
                .issuedAt(new Date(System.currentTimeMillis()))
                .expiration(new Date(System.currentTimeMillis() + 1000 * 60 * 24))
                .signWith(getSignInKey(), SignatureAlgorithm.HS256)
                .compact();

    }

    public Boolean isTokenValid(String token ,UserDetails userDetails)
    {
        final  String username = extractUserName(token);
        return  (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
    }

    private boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }

    private Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    private Claims extractAllClaims(String token)
    {
        return Jwts
                .parser()
                .setSigningKey(getSignInKey())
                .build()
                .parseClaimsJws(token)
                .getBody();
    }

    private Key getSignInKey() {
        byte[] keyByte = Decoders.BASE64.decode(SECRET_KEY);
        return Keys.hmacShaKeyFor(keyByte);
    }
}

@Configuration
public class LogoutConfiguration {

    @Bean
    public CustomLogoutHandler logoutHandler(TokenRepo tokenRepo, JwtService jwtService)
{
        return new CustomLogoutHandler(tokenRepo, jwtService);
```

```
    }

    @Bean
    public LogoutSuccessHandler logoutSuccessHandler() {
        return new CustomLogoutSuccessHandler();
    }
}
```

USER DETAILS

```
public class User implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;
    private String name;
    private String email;
    private String password;

    @Enumerated(EnumType.STRING)
    private Department department;

    @Enumerated(EnumType.STRING)
    private Role role;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return List.of(new SimpleGrantedAuthority(role.name()));
    }

    @Override
    public String getUsername() {
        return email;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
```

```
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    @JsonBackReference
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    private List<Token> tokens;


    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    @JsonIgnore
    private List<Schedule> schedules;


    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    @JsonIgnore
    private List<Attendance> attendances;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    @JsonIgnore
    private List<Notification> notifications;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    @JsonIgnore
    private List<Feedback> feedbacks;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    @JsonIgnore
    private List<LeaveEntity> leaves;

    @OneToMany(mappedBy = "manager", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    @JsonIgnore
    private List<Room> managedRooms;

    @OneToMany(mappedBy = "manager", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    @JsonIgnore
```

```
    private List<Resource> managedResources;

}



@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
@Entity
public class Schedule {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;

    private String title;
    private LocalDateTime date;
    private LocalDateTime time;
    private String assignedTo;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = “user_id”)
    @JsonIgnore
    private User user;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = “room_id”)
    @JsonIgnore
    private Room room;
}
```

CHAPTER 6

CONCLUSION

This chapter tells about the conclusion that anyone can drive from the project and the learning we learnt by taking over this project.

6.1 CONCLUSION

In conclusion, the proposed job searching portal is designed to streamline the job search process and enhance communication between different user roles, including job seekers, employers, and administrators. This system is scalable, accommodating the needs of various job markets, from small businesses to large corporations. It simplifies job posting and application management, improves transparency in job availability, and ensures the security of user data. With real-time updates and adaptability to varying job market demands, the portal empowers users to efficiently manage job postings and applications, reduce mismatches, and improve overall operational efficiency. By fostering seamless interactions and providing comprehensive job searching solutions, this system enhances user satisfaction and strengthens professional relationships.