Bridge to work (B2W)

An Inclusive Job and social media Platform for People with Special Needs

Graduation Project
by

Sara Medhat El-bayoumi Mohamed
Sohila Hamed Mohamed Mohamed
Mohamed Mahmoud Mahmoud Agwa
Mohamed Khaled Tharwat Shafiq
Ali Mohamed Tawfiq El-Sayed Teama
Mariem Waleed El-shahat Ashour
Eman Shaban Ali Khedr
Ahmed Moustafa Arafa Mohammed
Mazen Yasser El-sayed Abdullah
Mohamed Ali Mohamed Mahmoud

A project submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Information Technology

Supervised by

Prof.Dr. Amira Rezk
Eng. Yasmeen El-Sakar
2024/2025

Abstract

In today's world, the global pursuit of equality has brought attention to the barriers faced by individuals with disabilities many of whom possess exceptional skills yet are repeatedly denied employment opportunities due to biases and lack of inclusive systems. Our project addresses this issue by creating a digital platform that empowers people with physical disabilities to find meaningful and suitable job opportunities without the fear of rejection.

The platform is designed to emphasize the compatibility between the individual and the job, recognizing that disability does not diminish skill, efficiency, or potential. In fact, many individuals with disabilities offer unique strengths that are often overlooked in traditional hiring processes. Our application allows users to showcase their talents, qualifications, and experiences in a way that highlights their abilities rather than their limitations.

To ensure true accessibility, the platform incorporates a variety of user-friendly features such as screen reader compatibility, voice input, customizable display settings, and sign language integration. This makes the platform easy to navigate for users with different physical needs.

Beyond job matching, our goal is to challenge outdated hiring norms and encourage companies to recognize the value of a diverse workforce. By providing a supportive, inclusive environment, we aim to not only connect individuals with opportunities but also promote long-term change in how disability is viewed in the professional world.

By leveraging modern technologies and a user-centered design approach, our platform has the potential to scale beyond local boundaries and serve as a global solution for inclusive employment. It not only bridges the gap between employers and skilled individuals with disabilities but also raises awareness about the importance of accessible digital environments and equal opportunity hiring. With the right support and adoption, our project can become a catalyst for societal change, inspiring other platforms and organizations to adopt more inclusive practices and redefine the standards of professional recruitment.

Acknowledgement

First and foremost, I would like to express my sincere gratitude to my project supervisor**, Dr. Amira Rezk** and **Eng. Yasmeen El-Sakar** ,for their continuous support, guidance, and encouragement throughout the development of this project. Their expertise, valuable feedback, and constructive suggestions played a vital role in shaping this work.

I am also thankful to the members of the project committee for their time, insightful comments, and commitment to helping me improve both the technical and practical aspects of the project. Their observations helped me refine many details and address key challenges during the development process.

I would like to acknowledge the use of open-source tools and AI technologies that facilitated certain technical components of the project. Their availability greatly enhanced the development process.

Finally, I would like to thank my family and friends for their unwavering encouragement and patience during the course of this journey. Their moral support has been an essential source of strength, especially during challenging moments.

This project would not have been possible without the collective contributions and support of all the above-mentioned individuals and institutions.

Table of Content

List of Figures

List of Tables

Chapter 1:
Introduction

1.1 Introduction

Technological advancements have reached unprecedented levels, transforming various aspects of our lives. Recognizing the potential of technology to create positive change, we have developed an innovative application dedicated to empowering people with disabilities by integrating them into the labor market. This app is designed to enhance their quality of life by offering employment opportunities that align with their skills, qualifications, and specific needs, considering their individual disabilities.

Through this platform, users with disabilities can create detailed profiles showcasing their talents, experiences, and capabilities. The app simplifies the job search process, enabling them to find and apply for positions that match their qualifications with ease. Additionally, the platform facilitates direct communication between job seekers and employers, streamlining the recruitment process and fostering mutual understanding.

By addressing barriers such as mobility and accessibility, this application helps users overcome challenges they may face in traditional job-seeking methods. Ultimately, it increases their chances of securing meaningful employment, while promoting inclusivity, equality, and social integration in the workforce.

Furthermore, the app incorporates advanced features such as personalized job recommendations based on users' profiles, preferences, and unique needs. By leveraging machine learning algorithms, the platform continuously refines its job matching system, ensuring that users receive the most relevant and suitable opportunities. This level of personalization not only simplifies the job search but also helps individuals discover roles that they might not have considered otherwise, expanding their horizons and potential career paths.

To enhance inclusivity, the app also provides resources for employers, including guidelines on accommodating different types of disabilities in the workplace and tools for creating accessible job postings. Employers can access a database of skilled and diverse candidates, while receiving support in building an inclusive work environment. This promotes a culture of understanding and collaboration, where both employees and employers can thrive.

In addition to job matching, the app offers various support features, such as virtual interviews and mentorship programs, designed to empower users throughout their employment journey. These resources help build confidence, develop new skills, and provide guidance on navigating the workforce with a disability. By fostering connections with industry professionals, the platform not only assists in securing jobs but also cultivates long-term career development.

Ultimately, this app serves as a bridge between people with disabilities and employers, addressing the gaps in traditional employment systems. By removing barriers and promoting an inclusive hiring process, the platform contributes to a more diverse and equitable labor market, where individuals are recognized for their abilities rather than their limitations. This innovative approach to job seeking has the potential to reshape the future of work, driving positive social change and empowering people with disabilities to lead fulfilling professional lives.

Figure 1.1 people with special needs in work

1.2 Problem Definition

In the world's quest for equality and Equal opportunities, some people with exceptional skills and talents are unable to secure employment solely because of their disabilities.

People with special needs may encounter the following issues during their job search:

1) May face rejection due to companies doubting their ability to perform the job and underestimating their skills.
2) May encounter a toxic work environment created by both employers and employees.
3) May struggle to find a suitable workplace with comfortable equipment and facilities adapted for individuals with special needs.
4) Some companies, in order to comply with laws appointing 5% of their workforce to people with special needs, may ask them to stay at home and provide very low salaries, which undermines their abilities and income.

1.3 Project Objectives

1) Develop an integrated job search platform (mobile app and website) specifically designed for individuals with special needs, ensuring accessibility and ease of use across both interfaces.

2) Implement an intelligent job matching system that considers the unique skills, abilities, and accommodation requirements of job seekers with special needs, facilitating meaningful connections with inclusive employers.

3) Create a user-friendly profile system allowing job seekers to showcase their strengths and specific skills, while enabling employers to post jobs with clear information about workplace accommodations and inclusive practices.

4) Design and implement robust communication tools within the platform to facilitate seamless interaction between job seekers and potential employers, supporting various communication preferences and needs.

5) Incorporate educational resources and guides for both job seekers and employers, promoting best practices in inclusive hiring and workplace accommodations.

6) Ensure data security, privacy, and synchronization across the mobile app and website, while implementing a feedback system for continuous platform improvement based on user experiences.

In conclusion

This chapter introduces the idea of creating a comprehensive and inclusive job search platform, consisting of both a mobile app and website, specifically designed for individuals with special needs. The platform's primary goal is to bridge the gap between job seekers with diverse abilities and inclusive employers, utilizing intelligent job matching technology that considers unique skills and accommodation requirements. By providing accessible interfaces, facilitating clear communication, and offering educational resources, the platform seeks to empower both job seekers and employers in creating more inclusive workplaces. The objectives emphasize the importance of user-friendly profile systems, robust data security, and continuous improvement based on user feedback, ensuring that the platform remains effective and responsive to the needs of its community. Ultimately, these objectives work together to create a supportive ecosystem that promotes equal employment opportunities and workplace diversity.

1.4 Project Scope

This project encompasses the development, implementation, and maintenance of an accessible mobile app and website designed to connect individuals with special needs to suitable job opportunities.

1.5 Contributions of This Study

1-**Development of an Inclusive Platform**:

We present a digital solution that helps people with physical disabilities find suitable jobs by focusing on skills rather than limitations.

2-**Accessibility Integration**:

The platform includes key accessibility features such as screen reader support, voice commands, and sign language tools to ensure ease of use for all users.

3-**Promoting Equal Opportunities**:

This study highlights the challenges disabled individuals face in employment and offers a practical approach to making job markets more inclusive.

4-**Employer Engagement**:

The platform encourages companies to adopt inclusive hiring practices by making it easier to connect with qualified candidates with disabilities.

1.6 Document Organization

This document consists of six chapters in addition to one appendix. A brief description about the contents of each chapter is given in the following paragraphs:

Chapter 1, Introduction, introduces the project objectives, the motivation of the project, the approach used in this project, and the scope of the project.

Chapter 2, Literature Review, provides the reader with an overview of the previous related work, common technologies used, and the relation between our work and the relevant work.

Chapter 3, System Analysis, includes the analysis of existing system, system requirements, use requirements, system architecture, development methodology, the tools, and languages used in our system.

Chapter 4,System Design, provides the system design including class diagram, database design and interface design.

Chapter 5, System Implementation, shows the process of mapping design into implementation, sample application code, system testing, results of the investigation, and
goals achieved.

Chapter 6, Conclusion and Future Work summarizes the entire research, and addresses the suggested improvements for the system.

Chapter 2:
Literature review

2.1 Introduction

In this chapter, we provide a background about the tools and techniques needed to build our system. We also review the previous related work to our system, and the common technologies that are used. At the end of the chapter, we provide a comparison between the system we are going to build and the related systems.

2.2 Background

The system we are going to build, barium, will depend on a mobile application & web App to operate. The following subsections provide a brief background information about the equipment, tools, and techniques needed to build our system.

2.2.1 Flutter

Flutter is a UI toolkit for creating fast, beautiful, natively compiled mobile applications with one programming language and a single codebase. It is an open-source development framework developed by Google. Generally, Flutter is not a language; it is an SDK. Flutter apps use Dart programming language for creating an app. The first alpha version of Flutter was released in May 2017. Flutter is mainly optimized for 2D mobile apps that can run on both Android and iOS platforms. We can also use it to build full-featured apps, including camera, storage, geolocation, network, third-party SDKs, and more. Dart is a general-purpose, object-oriented programming language with C-style syntax. It is open-source and developed by Google in 2011. The purpose of Dart programming is to create a frontend user interfaces for the web and mobile apps. It is an important language for creating Flutter apps. The Dart language can be compiled both AOT (Ahead-of-Time) and JIT (Just-in-Time. [1] The popular advantages of the Flutter framework are as follows: • Cross-platform Development: This feature allows Flutter to write the code once, maintain, and can run on different platforms. It saves the time, effort, and money of the developers. lauding Flutter, .NET, programming languages, and libraries.

Faster Development: The performance of the Flutter application is fast. Flutter compiles the application by using the arm C/C++ library that makes it closer to machine code and gives the app a better native performance. • Good Community: Flutter has good community support where the

developers can ask the issues and get the result quickly. • Live and Hot Reloading: It makes the app development process extremely fast. This feature allows us to change or update the code are reflected as soon as the alterations are made. • Minimal code: Flutter app is developed by Dart programming language, which uses JIT and AOT compilation to improve the overall start-up time, functioning and accelerates the performance. JIT enhances the development system and refreshes the UI without putting extra effort into building a new one. • UI Focused: It has an excellent user interface because it uses a design-centric widget, high-development tools, advanced APIs, and many more features.

2.2.2 .NET

The DOT NET is a software framework. It is developed by Microsoft. It includes a large library and also provides language inter-operability across some programming languages. Language inter-operability refers the capability of two different languages to interact and operate on the same kind of data structures. The programs written for DOT NET execute in a software environment. The name of the software environment is Common Language Runtime (CLR). It is the virtual machine component. The compiled code is converted into machine code at first. Then it is executed by computer's CPU. The CLR provides additional services like exception handling, memory management, type safety, garbage collection, thread management etc. [2] The DOT NET Framework's Base Class Library offers user interface, database connectivity, data access, cryptography, web application development, numeric algorithms, network communications etc. Programmers produce software by combining their own source code with the DOT NET Framework and other libraries. The DOT NET Framework is projected to be used by most new applications created for the Windows platform. Microsoft also produces an integrated largely for DOT NET software called Visual Studio.

The popular advantages of the .NET framework are as follows:

• Interoperability

• Common Language Runtime engine (CLR)

• Language independence

• Base Class Library

• Simplified deployment

• Security

• Portability

2.2.3 React

React is a popular JavaScript library used for building user interfaces, particularly for single-page applications. Developed by Facebook, it allows developers to create reusable UI components, which makes code more organized and easier to maintain.

Key Features:

1. **Component-Based**: React encourages the development of components, which are self-contained units of code that manage their own state and rendering.
2. **Virtual DOM**: React uses a virtual representation of the DOM to optimize rendering. When changes occur, React calculates the minimal number of updates needed and efficiently updates the actual DOM.
3. **Declarative Syntax**: Developers describe how the UI should look based on the application state, making the code more predictable and easier to debug.
4. **State Management**: React allows components to manage their own state, which can be passed down to child components as props.
5. **Ecosystem**: React has a rich ecosystem with tools like React Router for navigation, Redux for state management, and many others that enhance its capabilities.

Overall, React streamlines the development process and helps create dynamic, high-performance web applications.

2.3 Review of Relevant Work

1-Helm-حلم

Helm is a popular package manager for Kubernetes that simplifies the management of applications and services on Kubernetes clusters.

Here are some of its key **Advantages**:

1. **Employment and Professional Support:**
   The platform offers job opportunities and vocational training for individuals with disabilities, helping them integrate into the job market.

2. **Educational Resources:**
   It provides a range of educational materials and training programs tailored to the needs of individuals with disabilities.

3. **Psychological and Social Support:**
   The platform offers counseling and psychological services to help users overcome challenges.

4. **Activities and Events:**
   It organizes social and cultural activities and events to promote interaction between individuals with disabilities and the community.

5. **Awareness and Education:**
   The platform works to raise awareness about disability issues and educate the community about the rights and needs of individuals with disabilities.

Disadvantages:

1. **Complexity**: Helm can add complexity to your Kubernetes deployments, especially for those unfamiliar with its concepts and structure.
2. **Learning Curve**: New users may face a steep learning curve due to the intricacies of Helm charts and templating.
3. **Dependency Issues**: Managing dependencies can become challenging, particularly if charts have conflicting requirements or versions.
4. **Version Compatibility**: Not all charts are maintained or compatible with the latest Helm versions, which can lead to issues during upgrades.
5. **Security Concerns**: Helm charts may include unverified or insecure code, potentially introducing vulnerabilities if not carefully audit.

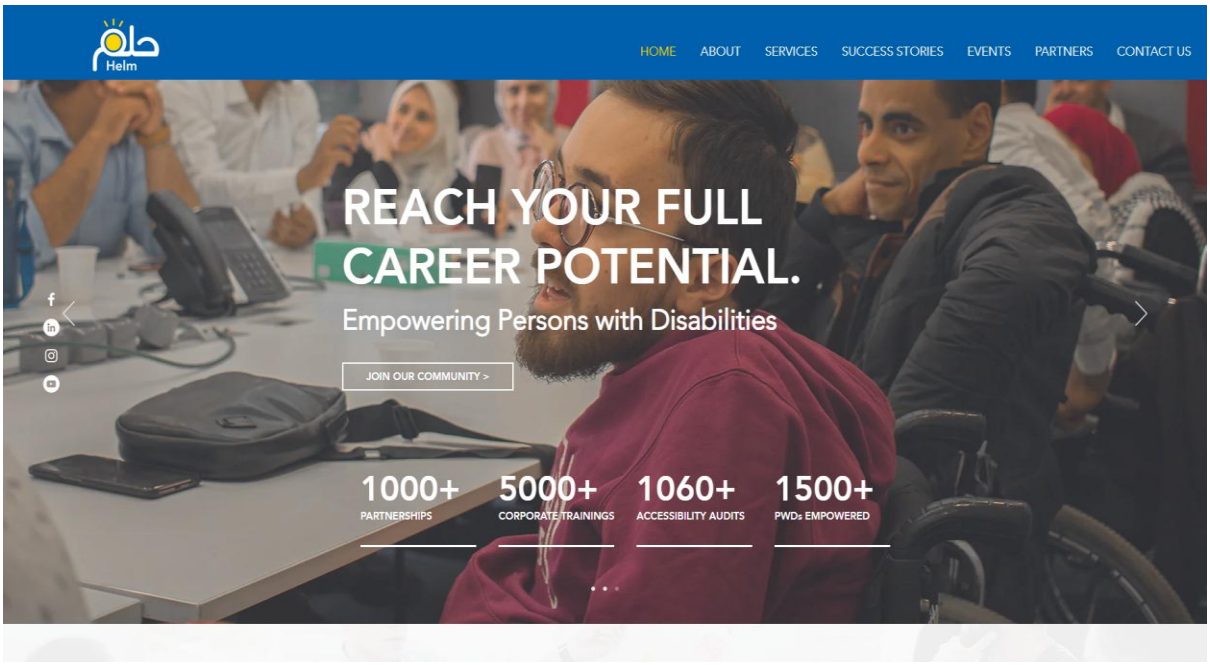


Figure2.1 - Helm platform

2-Ability Jobs:

is a platform focused on connecting individuals with disabilities to job opportunities. Its goal is to promote inclusivity in the workplace by providing resources and support for both job seekers and employers.

Advantages:

1. **Job Listings**: A comprehensive database of job opportunities from employers committed to hiring individuals with disabilities.
2. **Resume Resources**: Tools and tips to help users create effective resumes and prepare for interviews.
3. **Employer Connections**: Partnerships with companies dedicated to diversity and inclusion in their hiring practices.
4. **Community Support**: Forums and networking opportunities to connect job seekers with each other and with mentors.
5. **Educational Resources**: Information on rights, workplace accommodations, and skill development.

Disadvantages:

1. **Limited Reach:** The platform may not have as many job listings compared to larger job boards, limiting opportunities for users.
2. **Employer Participation:** Not all employers may be aware of or committed to hiring individuals with disabilities, which can affect the availability of suitable jobs.
3. **Niche Focus:** While the focus on disability is important, it might alienate some job seekers who do not identify as having a disability but support inclusivity.
4. **Awareness and Accessibility:** Potential users might not know about the platform, or it may lack accessibility features that some users need.
5. **Dependence on Employers:** The effectiveness of the platform relies heavily on employers' willingness to create inclusive job environments and accommodate diverse needs.
6. **Not supporting Arabic**.

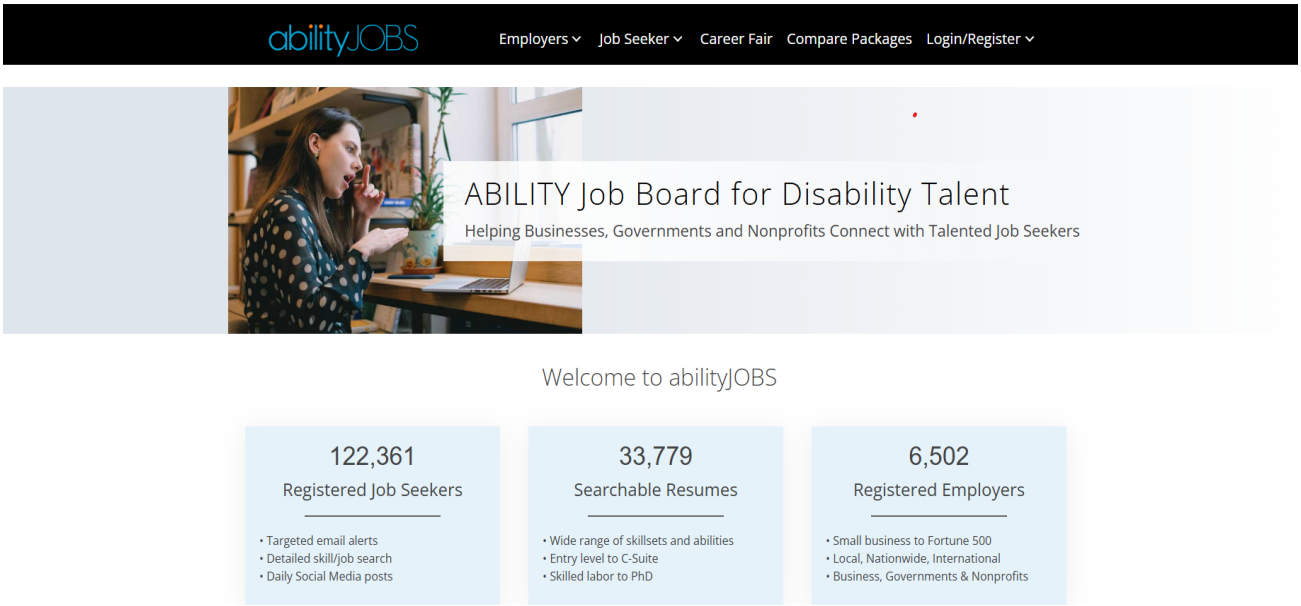


Figure2.2 - Ability Jobs platform

3-Disabled person

Advantages:

1. **Targeted Job Listings**: Disabled person focuses on job opportunities specifically for individuals with disabilities, making it easier to find suitable roles.
2. **Employer Commitment**: The platform partners with companies that are dedicated to diversity and inclusion, increasing the likelihood of supportive work environments.
3. **Resources for Job Seekers**: It provides tools, tips, and guidance for creating effective resumes and preparing for interviews, tailored for individuals with disabilities.
4. **Community Support**: Users can connect with peers and mentors, fostering a sense of community and support among job seekers.
5. **Educational Materials**: The platform offers information on rights, workplace accommodations, and skill development, helping users navigate the job market.

Disadvantages:

1. **Limited Reach**: The number of job listings may be less compared to larger job boards, which can restrict opportunities for users.
2. **Dependence on Employer Engagement**: The effectiveness of the platform relies on the willingness of employers to create inclusive workplaces and actively engage with the platform.
3. **Awareness Issues**: Some potential users may not be aware of the platform, limiting its reach and impact.
4. **Varied Job Quality**: The quality of job listings can vary, and not all employers may provide adequate accommodation or support.
5. **Niche Focus**: While focusing on disabilities is important, it might inadvertently alienate some users who do not identify as having a disability but support inclusivity.

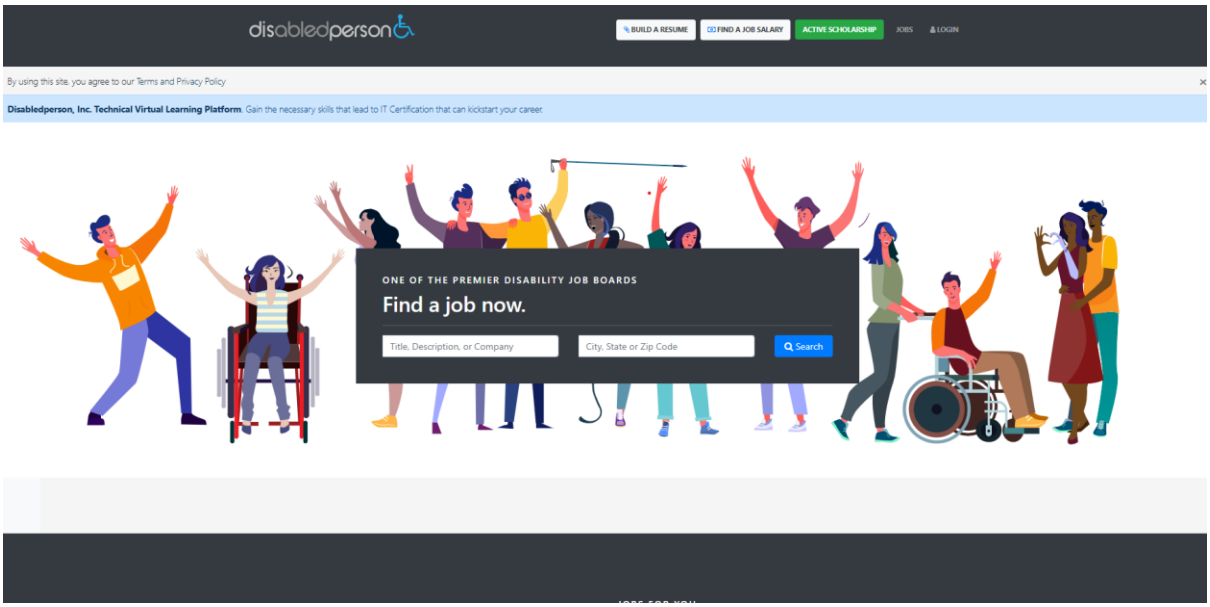


Figure2.3 - Disabled person platform

4-Disabilityin

Advantages:

1. **Global Reach and Network:** collaborates with a wide range of multinational companies and organizations, providing opportunities worldwide. Their large network offers job seekers access to top-tier inclusive employers.
2. **Comprehensive Programs**: Offers a variety of initiatives like the *NextGen Leaders* program for young professionals and mentorship opportunities. They also run the *Inclusion Works* program to help businesses improve their hiring and inclusion practices.
3. **Support for Employers:** Disability:IN educates businesses through benchmarking tools like the *Disability Equality Index (DEI)*, encouraging companies to become disability friendly. Helps employers improve accessibility, making workplaces more inclusive for people with disabilities.
4. **Tailored Resources for Job Seekers:** Provides resources for job seekers on accommodations, interview preparation, and workplace rights. Hosts events like virtual and in-person career fairs to connect individuals with hiring managers.
5. **Focus on Advocacy and Inclusion:** Advocates for disability inclusion in workplaces globally. Promotes awareness about the benefits of hiring people with disabilities, thereby reducing stigma.
6. **Diverse Industry Participation:** Their network spans industries, giving candidates a variety of fields to explore based on their interests and qualifications.

Disadvantages:

1. **Primarily Employer-Focused**: Much of Disability:IN's work centers on helping employers become inclusive. While this indirectly benefits job seekers, it may not always directly address individual job seeker needs.

2. **Limited Direct Job Listings:** Unlike platforms like Inclusively or Getting Hired, Disability:IN does not operate as a job board. Users may need to rely on employer partners to find job postings.
3. **Geographic Constraints:** Although global, the strongest focus tends to be on North America. Access to resources and programs may vary in other regions.
4. **Resource Intensity for Small Companies:** Small businesses might find it challenging to implement the comprehensive strategies promoted by Disability:IN, potentially reducing job opportunities for individuals in smaller firms.
5. **Membership Model:** Companies must pay to join the Disability:IN network, which might limit participation from smaller or resource-constrained organizations.
6. **Accessibility of Events and Resources:** Some events and programs may be virtual or require specific technology, which could pose challenges for individuals with limited access to digital tools.

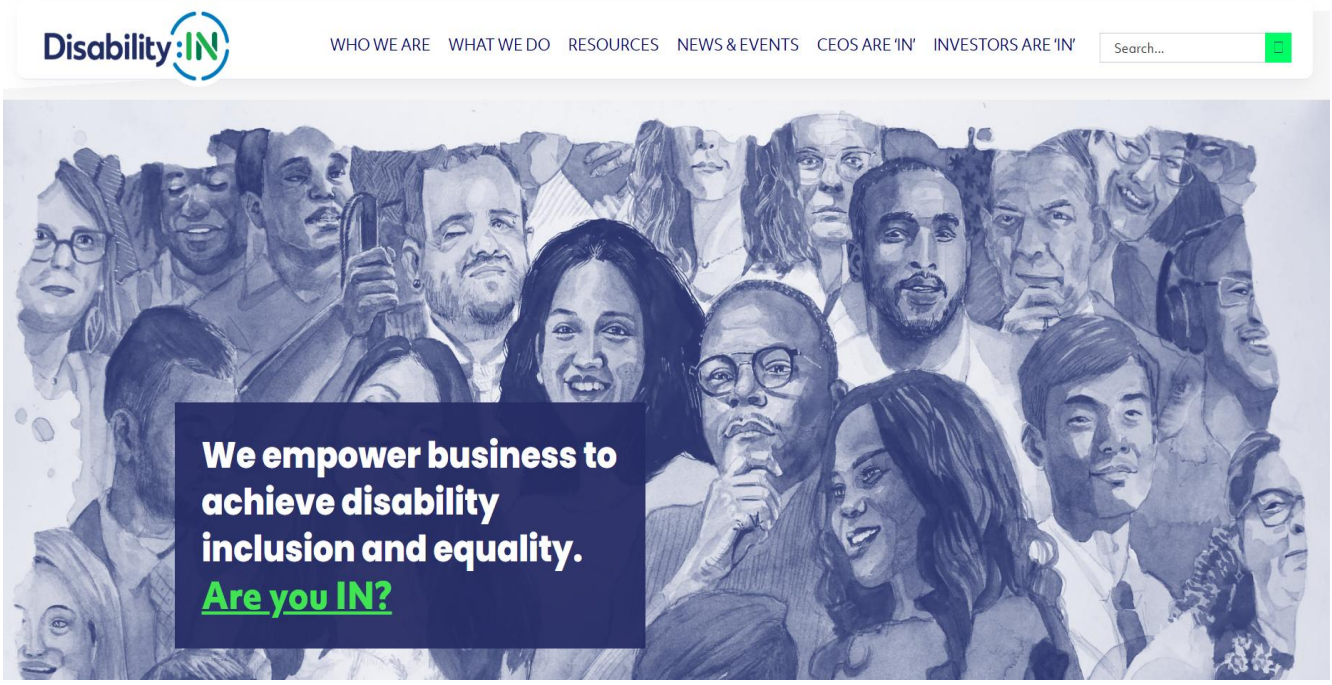


Figure 2.4 Disability:IN platform

2.4 relationship between the Relevant Work and our own work

Feature/platform	B2W	Disabled person	Ability Jobs	Helm-حلم	Disability/in
Social Media App.	✓	✗	✗	✗	✗
Multi-Language	English - Arabic	English	English	English - Arabic	English
Accessibility	✓	✗	✗	✓	✗
Application	✓	✗	✗	✗	✗
Target Job Listing	✓	✓	✓	✗	✓
Specialized Employer Resources	✓	✓	✓	✗	✗
User-Friendly Design	High	Moderate	Moderate	Moderate	Moderate
Job Opportunities	✓	✓	✓	✗	✓
Accessible Content	✓	✓	✗	✗	✓
Job seekers Profiles	✓	✗	✗	✗	✗
Company profiles	✓	✗	✗	✗	✗
Employer Rating	✓	✗	✗	✗	✗
Ai features	✓	✗	✗	✗	✗
Direct Messaging	✓	✗	✗	✗	✗

Table 2.1 Related Work

2.4 Summary

In this chapter, we reviewed several existing applications aimed at supporting Hiring People with special needs. The applications evaluated include Disabled person, Ability Jobs, Helm-حلم .

Each application has distinct features and drawbacks:

Helm-حلم: focuses on offering online courses and training for people with special needs, helping them develop skills and enhance their capabilities. However, it does not prioritize connecting them with employment opportunities or direct job placements.

Disabled person: A platform for people with special needs connects users to job opportunities, skill development resources, and community support. It emphasizes accessibility, allowing users to filter job listings based on their specific needs. The platform fosters networking and mentorship while ensuring privacy and security. Overall, it aims to empower individuals with disabilities and promote inclusivity in the workforce.

Ability Jobs: a platform dedicated to connecting job seekers with disabilities to inclusive employers. It provides a space for users to create profiles, search for job opportunities, and access resources for skill development. The platform emphasizes accessibility and encourages companies to showcase their commitment to diversity. Overall, Ability Jobs aims to empower individuals with disabilities in their career pursuits.

Disability:IN: is a platform for advocating disability inclusion in workplaces and connecting job seekers with inclusive employers. However, its employer-focused approach and reliance on company participation may not always meet the immediate needs of job seekers looking for job-specific resources or listings.

By comparing these applications, we identify gaps and areas for improvement that our social media platform for disability jobs aims to address, such as ensuring comprehensive care, enhancing user-friendliness, and reducing dependency on internet connectivity.

Chapter 3:

System Analysis

3.1 Introduction

System analysis is the process of studying a procedure or business to identify its goals and purposes and create systems and procedures that will achieve them in an efficient way. Another view sees system analysis as a problem-solving technique that divide a system into its component pieces for the purpose of the studying how well those component parts work and interact to accomplish their purpose. The field of system analysis relates closely to requirements analysis or to operations research. It is also an explicit formal inquiry carried out to help a decision maker identify a better course of action and make a better decision than they might have made.

The terms analysis and synthesis stem from Greek, meaning "to take apart" and "to put together," respectively. These terms are used in many scientific disciplines, from mathematics and logic to economics and psychology, to denote similar investigative procedures. Analysis is defined as "the procedure by which we break down an intellectual or substantial whole into parts," while synthesis means "the procedure by which we combine separate elements or components in order to form a coherent whole." System analysis researchers apply methodology to the systems involved, forming an overall picture.

System analysis is used in every field where something is developed. Analysis can also be a series of components that perform organic functions together, such as system engineering. System engineering is an interdisciplinary field of engineering that focuses on how complex engineering projects should be designed and managed.

This chapter provides a thorough analysis of the "B2W" system, outlining the requirements, use cases, process flows, system interfaces, and data model. This analysis serves as a foundation for the design and implementation of the system.

3.2 Analysis of Existing Systems

Analyzing existing systems for people with special needs involves several critical points to ensure effectiveness, accessibility, and inclusivity. Here are the most important points to consider:

1. User-Centered Design: Focus on the needs and preferences of individuals with special needs. Involve them in the design and evaluation process to ensure the system meets their requirements.

2. Accessibility: Evaluate the system's compliance with accessibility standards (e.g., WCAG). Ensure that it accommodates various disabilities, including visual, auditory, physical, and cognitive impairments.

3. Usability: Assess how easy the system is to use for individuals with special needs. This includes evaluating navigation, readability, and the overall user experience.

4. Functionality: Analyze whether the system provides the necessary features and tools that support the specific needs of users, such as communication aids, mobility assistance, or educational resources.

5. Integration with Other Services: Examine how well the system integrates with other services a support available to individuals with special needs, such as healthcare, education, and social services.

6. Training and Support: Evaluate the availability of training and support for users and caregivers. Ensure that resources are accessible and tailored to the needs of individuals with special needs.

7. Feedback Mechanisms: Implement and assess feedback mechanisms that allow users to report issues, suggest improvements, and share their experiences with the system.

8. Data Privacy and Security: Ensure that the system protects the privacy and security of users, especially when handling sensitive information related to their disabilities.

9. Cultural Competence: Consider the cultural and linguistic diversity of users. Ensure that the system is inclusive and respectful of different backgrounds and experiences.

10. Continuous Improvement: Establish a process for ongoing evaluation and improvement of the system based on user feedback and changing needs within the community.

By focusing on these points, you can effectively analyze existing system and identify areas for enhancement to better serve individuals with special needs.

3.3 System Requirements

System requirements are the needed configurations for the system to operate efficiently. The next three subsections will discuss the functional requirements, Nonfunctional requirements, and user requirements.

3.3.1 Functional Requirements

A Functional Requirement (FR) is a description of the service that. the software must offer. It describes a software system or its component. A function is nothing but inputs to the software. system, its behavior, and outputs. It can be a calculation, data. manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely. to perform. Functional Requirements in Software Engineering are also called Functional

Specification. In software engineering and systems engineering, a Functional Requirement can range from the high-level abstract statement of the sender's necessity to detailed mathematical functionality. requirement specifications. Functional software requirements help. you to capture the intended behavior of the system. In this subsection, we list the functions required in our system.

3.3.2 Non-Functional Requirements

Non-functional requirements or NFRs are a set of specifications. that describe the system's operation capabilities and constraints. and attempt to improve its functionality. These are basically the requirements that outline how well it will operate including things. like speed, security, reliability The system needs to operate efficiently and meet the requirements. Any failure of the components of the systems may lead to one or more of functions to stop or be misused. A non-functional requirement is a requirement that specifies criteria that. can be used to judge the operation of a system.

#	Functional Requirements	Requirement Description
1	Registration	Users should be able to create new account on the system and define their name, phone, email and password and authenticate them themselves.
2	Set up user profile	Enables user to define his information: name, age , phone, address , certifications , skills and his disability.
3	Accessibility menu customization	Enables user to choose the screen setting: font size, appearance of content text spacing, text alignment, and line height. and colour scheme.
4	System Services	Users can view system services like searching for a job, viewing messages and using AI tools.
5	Jobs search and filtering	Enables users to search for jobs based on his skills and disability.
6	Multilanguage Support	Enables user to choose English and Arabic Language.
7	CV Analysis	CV Reviewers services compare user's cv with the suitable job for his skills and telling him his missing skill he should have to be accepted and suggested some courses to learn.
8	Provide user with available equipment	The company provide available equipment for employees like comfortable chairs, lefts.
9	AI powered interview	Enables user to make interview with chatbot and get feedback.
10	Pre-job assessments	Provide an assessment to user to evaluate user's suitability for a specific job based on his skills.

Table 3.3.1 Functional Requirements

#	Non-functional requirements	Description
1	Accessibility	The system should be ease of use and accessible to all users.
2	Usability	The system should be ease of use and understandable to end users.
3	Performance	Optimize the system with speed performance within many users and data.
4	Security	Implement high safety and security to protect user from unauthorized access.
5	Scalability	The system should be designed to scale and accommodate growth in the future to increase the number of users.
6	Reliability	The ability of the system to operate function correctly and consistently without errors and down time.
7	Maintainability	The system should be eased to maintain including testing and modifying.
8	Accuracy	The system provides high accuracy in choosing the job related to the disability.

Table 3.3.2 Non-functional requirements

3.3.3 User Requirements

The user in the application is a person who has a disability that we want to provide suitable jobs for his disability based on his skills and the system go to fulfill his needs.

The following list will show all user requirements:

- The user can register and login to the system.
- The user can set his profile with his information, skills, certifications and his disability.
- The user can view the notifications.
- The user can choose the accessibility setting.
- The user can search for jobs related to his disability.
- The user can apply for a suitable job based on his skills.
- The user can choose the languages.
- Providing available equipment to user such as comfortable chairs, left.
- The user can have an assessment to evaluate his skill is suitable for a specific job.
- The company provide new skills to users to learn to enable him for the job.
- The users can compare his cv with cv analysis to get prospect of the acceptance.
- The user can request an AI interview with chatbot and get feedback.
- The user can provide feedback about the system services

3.4 System Architecture

After determining the requirements of the system, we will describe its major Components, their relationships (structures), and how they interact with each other.

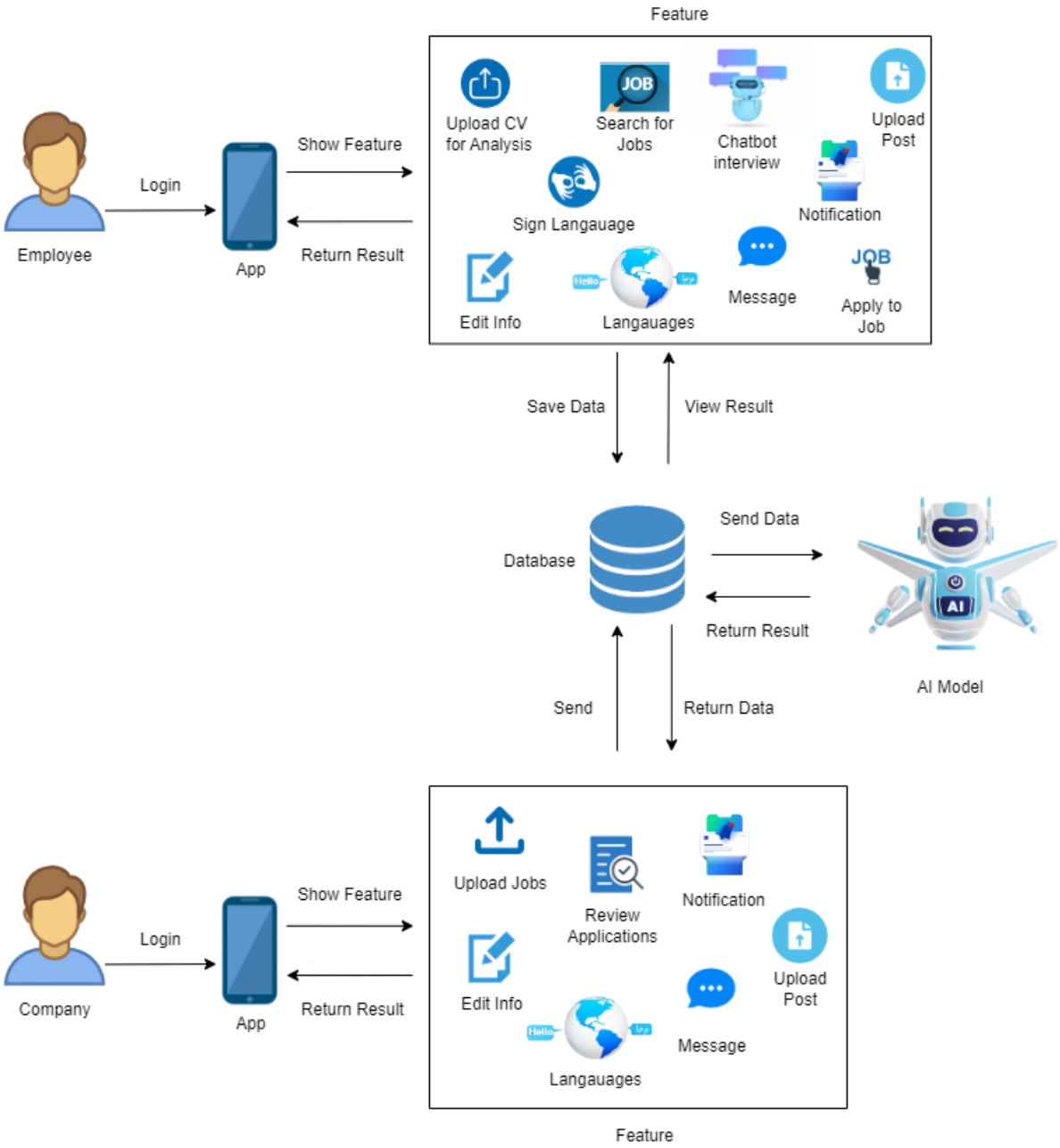


Figure 3.1 - System Architecture

3.5 Development Methodology

After we knew the basic structure of the system. We are going to view all of its functions, the relation between them, and the sequence of their executions in the following subsections.

3.5.1 Use Case Diagram

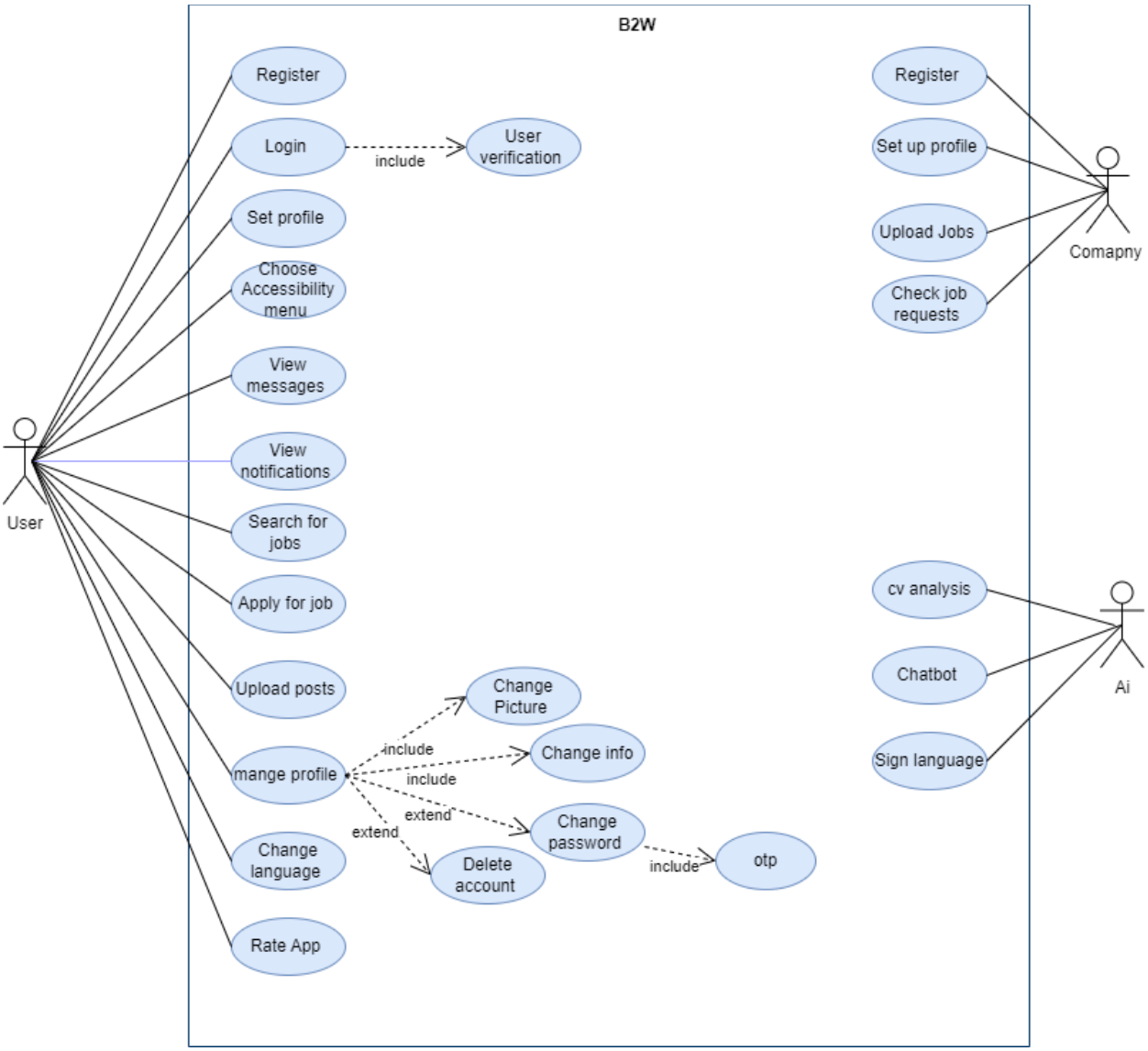


Figure 3.2 - Use Case Diagram.

The basics of use case diagram:

1. Actors:

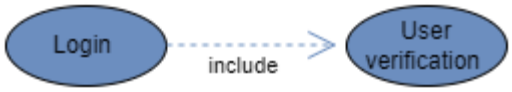
The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data.

2. Use case:

- System function (process - automated or manual)
- Named by verb + Noun (or Noun Phrase).
- Do something.
- Each Actor must be linked to a use case, while some use cases may not be linked to actors.
- Horizontally shaped ovals that represent the different uses that a user might have.

3. Communication Link

- The participation of an actor in a use case is shown by connecting an actor to a use case by a solid link.
- Actors may be connected to use cases by associations, indicating that the actor and the use case communicate with one another using messages.



4. Include

- relationship between use cases is shown by a dashed arrow with an open arrowhead from the including (base) use case to the included (common part) use case. The arrow is labeled with the keyword «include».

3.5.2 Use Case Description

Use Case: Bridge to work system based on AI.

Actors: User, AI, Company.

Goal: Build a system that provide work for people with disability.

Description:

B2W system based on AI, powered by AI techniques such as prediction and Recommendation.

When user registers his information, he sets up his profile with his name, phone, email, address, skills, certifications and his disability.

User searches for jobs that suitable for his disability and AI provide him recommendation jobs based on his disability.

User can get a test that show if he suitable for the job or he need to have more skills.

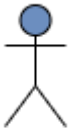
The system allows user to speak to chatbot to answer interview questions via text or voice and the system applying a sensitive analysis on the answered questions to provide feedback to user.

User can use cv Analysis to compare his cv with the suitable job for his skills and telling him his missing skill he should have to be accepted and suggested some courses to learn.

3.5.3 Sequence Diagrams

Having an idea about the operations available for each user, we are going to dig deeper and see how they are done. The sequence diagrams will show you how an operation is done and the inner details of it.

The basics of sequence diagram:



1. **Actor**

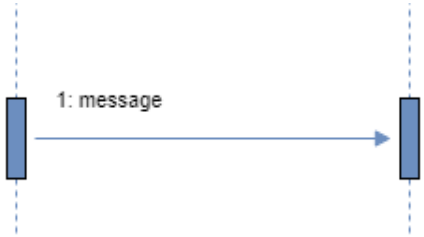
- a type of role played by an entity that interacts with the subject.
- (e.g., by exchanging signals and data)
- An actor does not necessarily represent a specific physical entity.
- but merely a particular role of some entity
- A person may play the role of several different actors and,
- conversely, a given actor may be played by multiple different persons.

2. **Lifeline**

- A lifeline represents an individual participant in the Interaction.

3. **Activations**

- A thin rectangle on a lifeline) represents the period during which an element is performing an operation.
- The top and the bottom of the rectangle are aligned with the initiation and the completion time respectively.



4. **Call Message**

- A message defines a particular communication between Lifelines of an Interaction.
- Call message is a kind of message that represents an invocation of operation of target lifeline

5. **Return Message**

- A message defines a particular communication between Lifelines of an interaction. Return Message is a kind of message that represents the pass of information back to the caller in a corresponded former message.

Figure 3.3 show the sequence for user setting up profile:

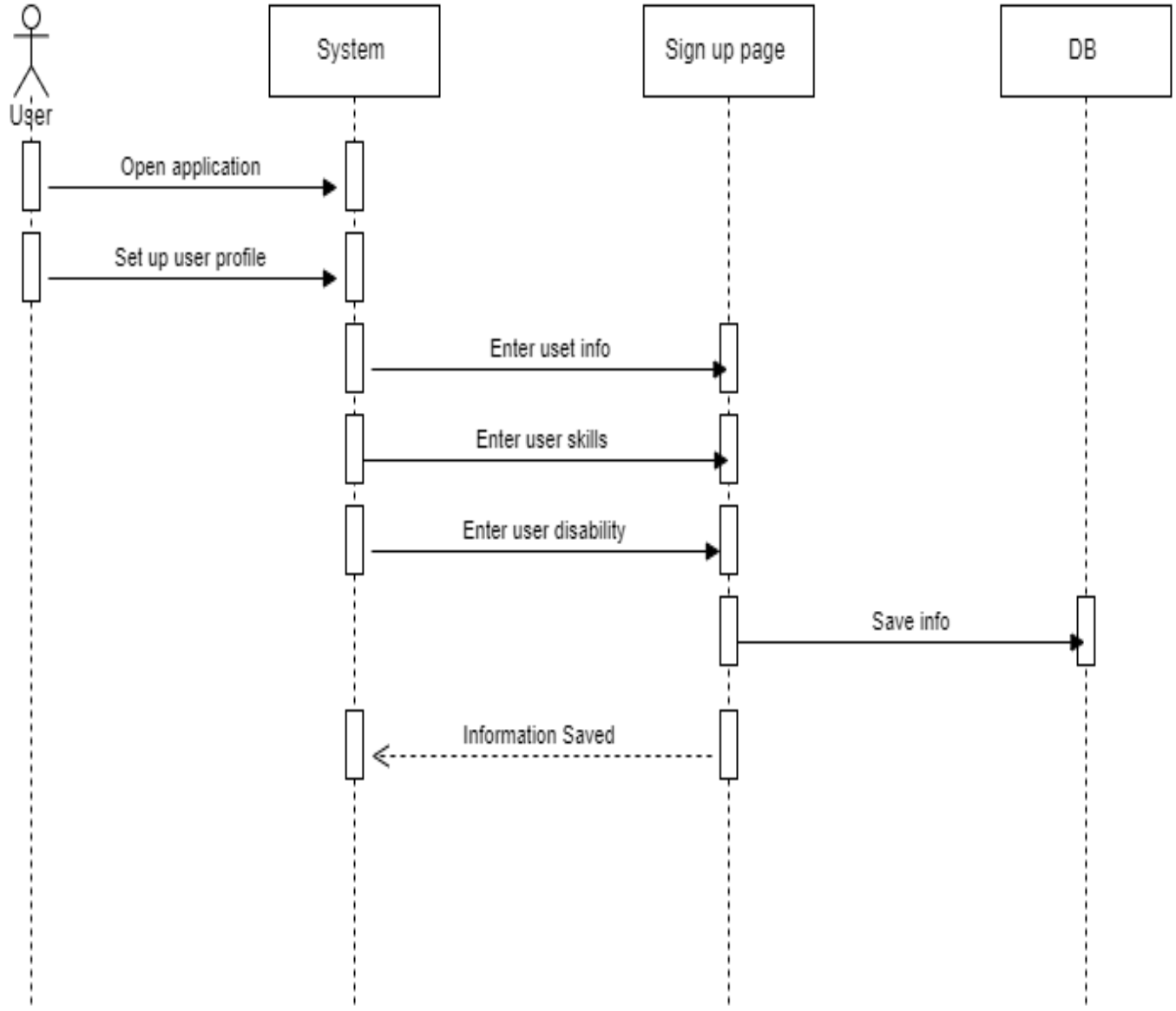


Figure 3.3 - user setting up profile sequence diagram.

Figure 3.4 show the sequence of company setting up profile:

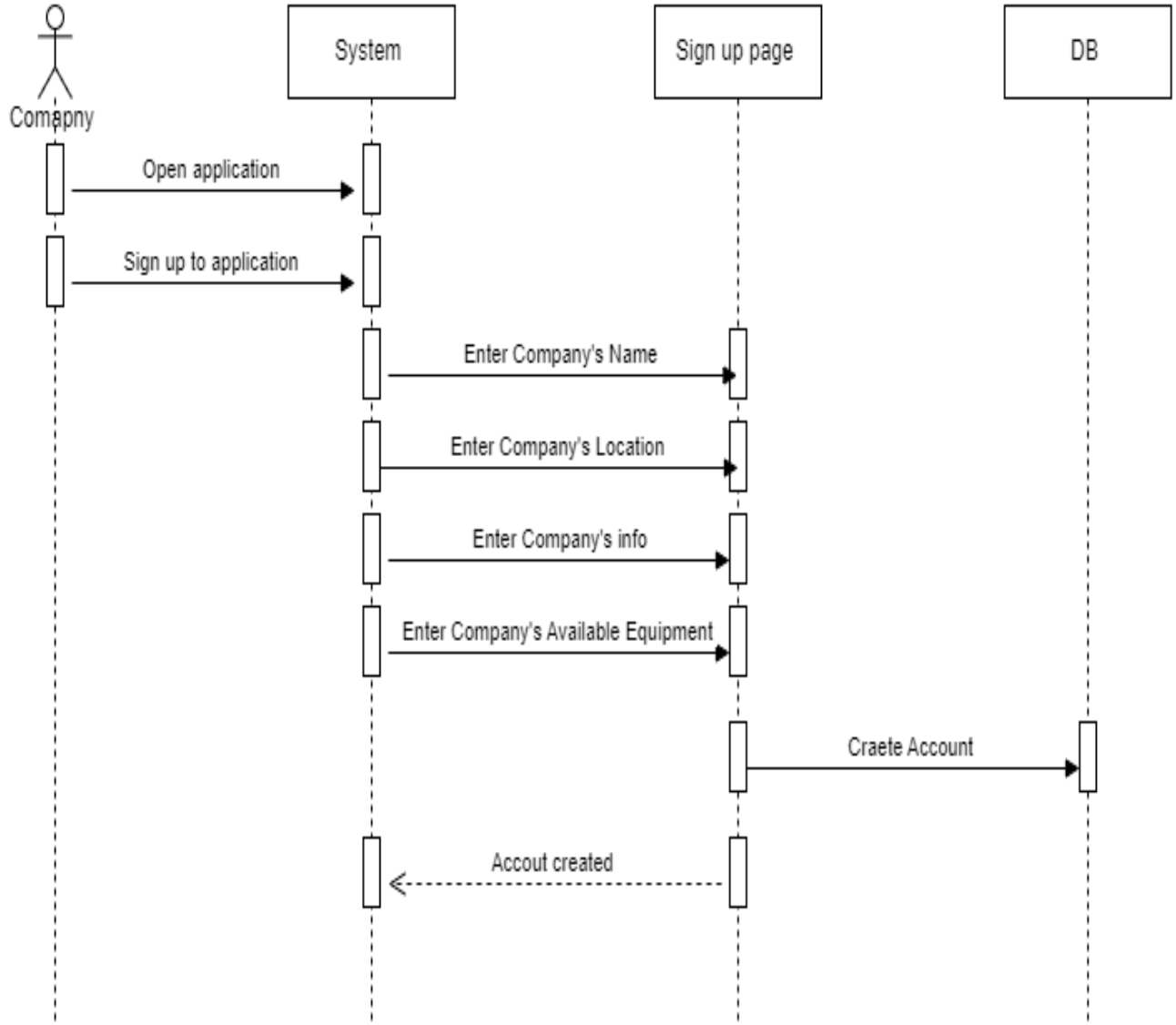


Figure 3.4 - company setting up profile sequence diagram.

Figure 3.5 show the sequence diagram of job recommendation:

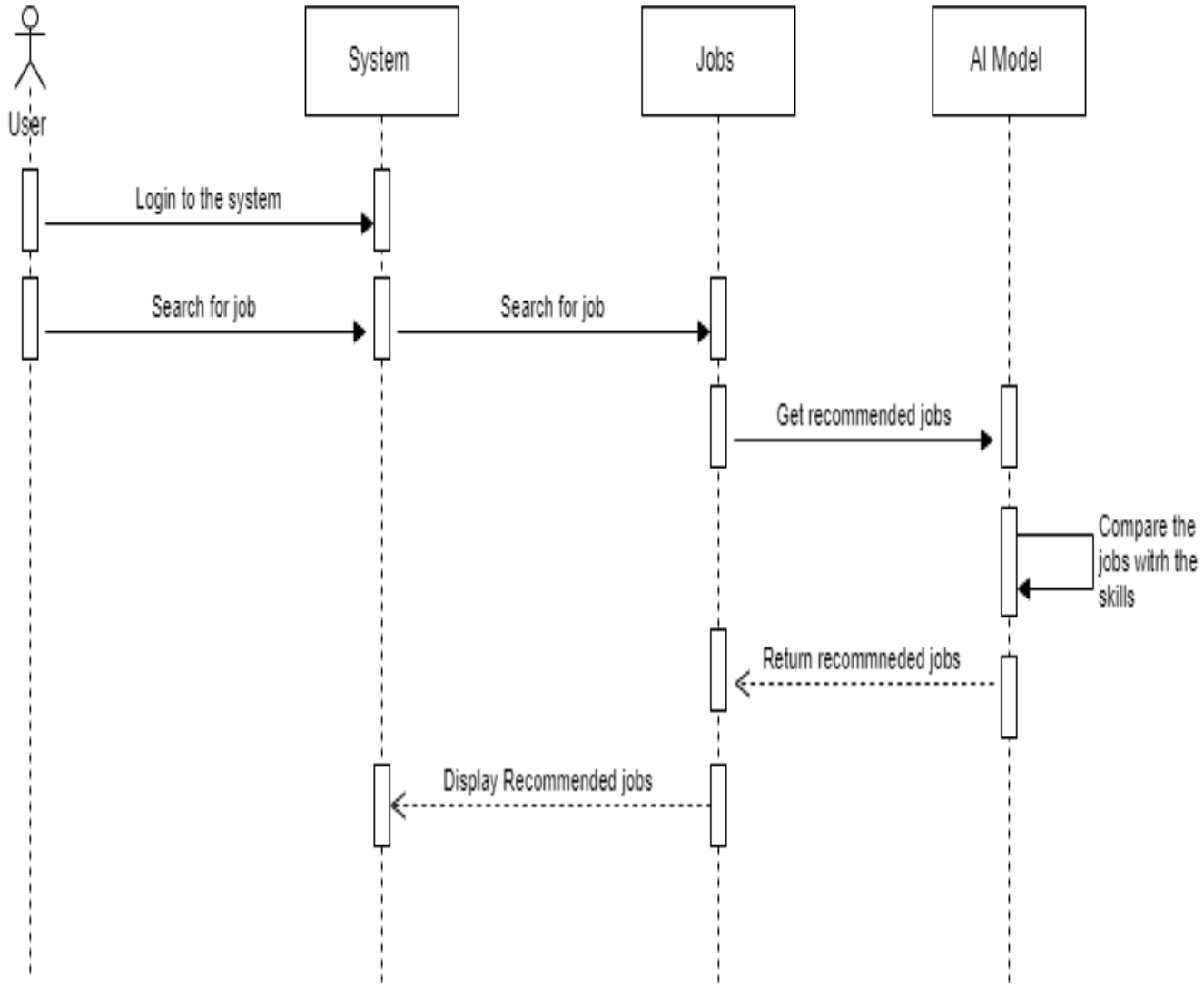


Figure 3.5 - Job recommendation sequence diagram

Figure 3.6 show the sequence diagram of applying to a job:

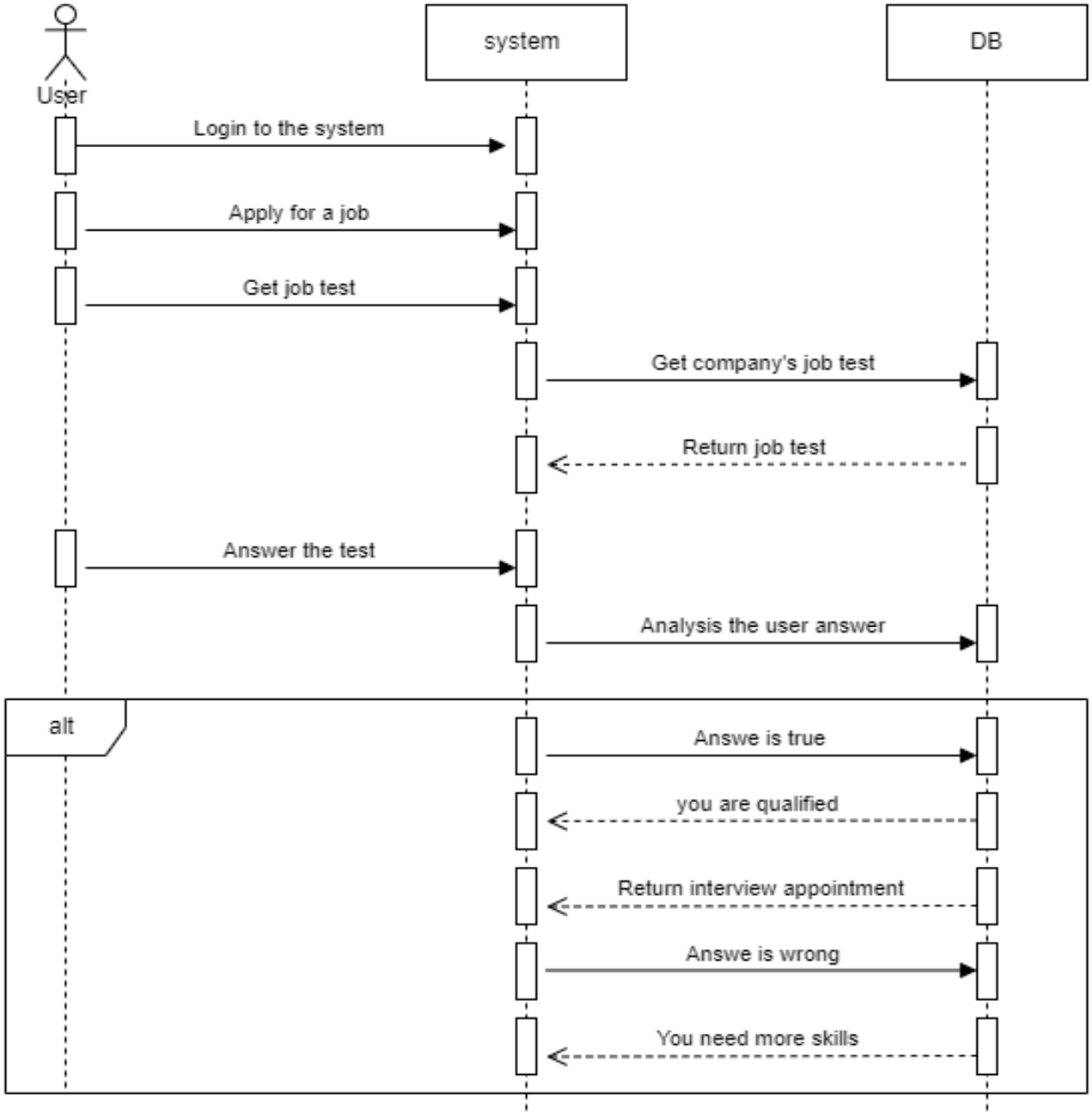


Figure 3.6 - applying for job sequence diagram

Figure 3.7 show the sequence diagram of cv Analysis that AI analysis user's cv and provide feedback to user:

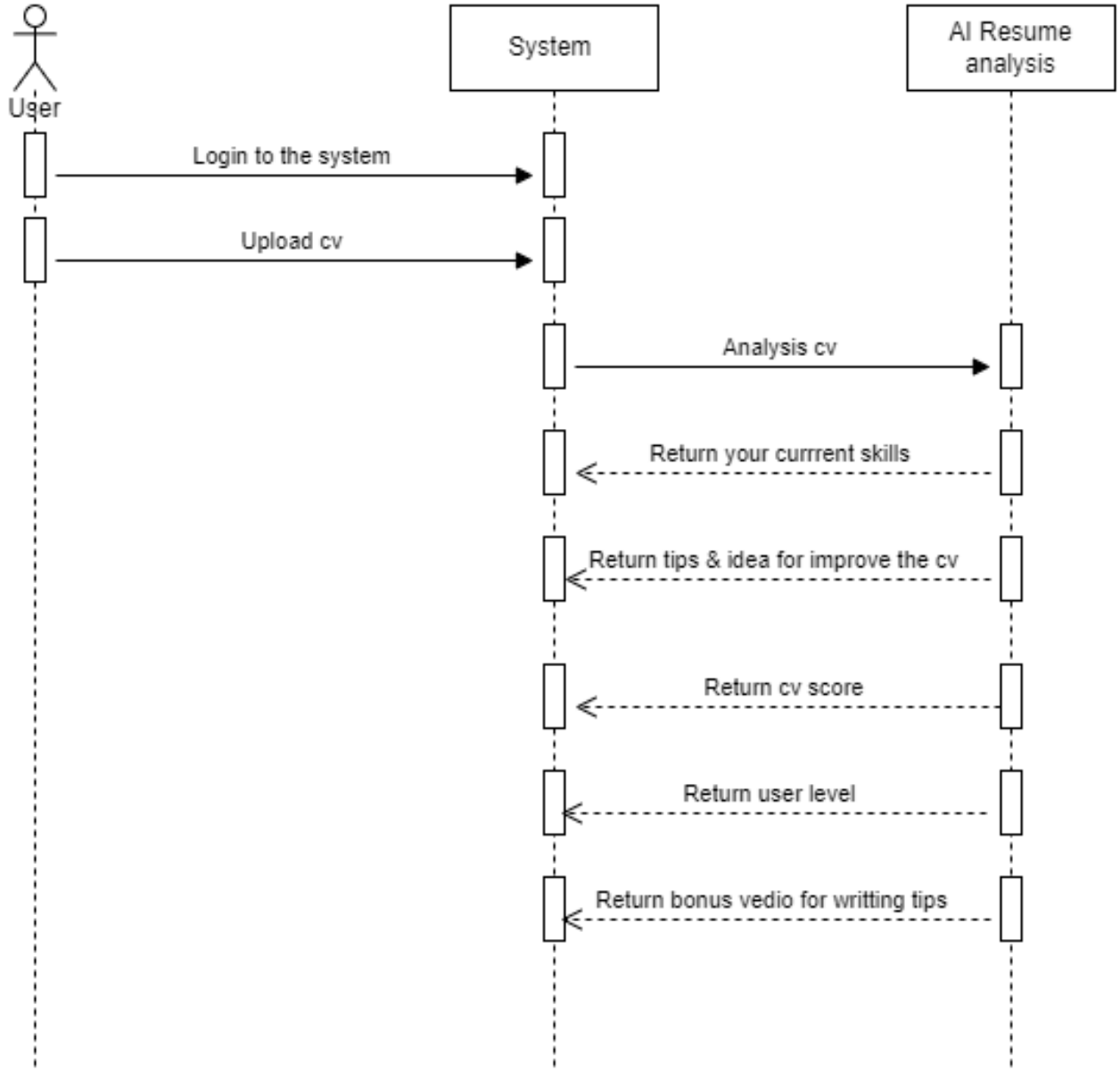


Figure 3.7 CV analysis sequence diagram

Figure 3.8 show the sequence diagram of chatbot that the user requests an interview and the chatbot provide user with the questions then the user answer the questions via text or voice.

Chatbot makes sensitive analysis to provide feedback to the user.

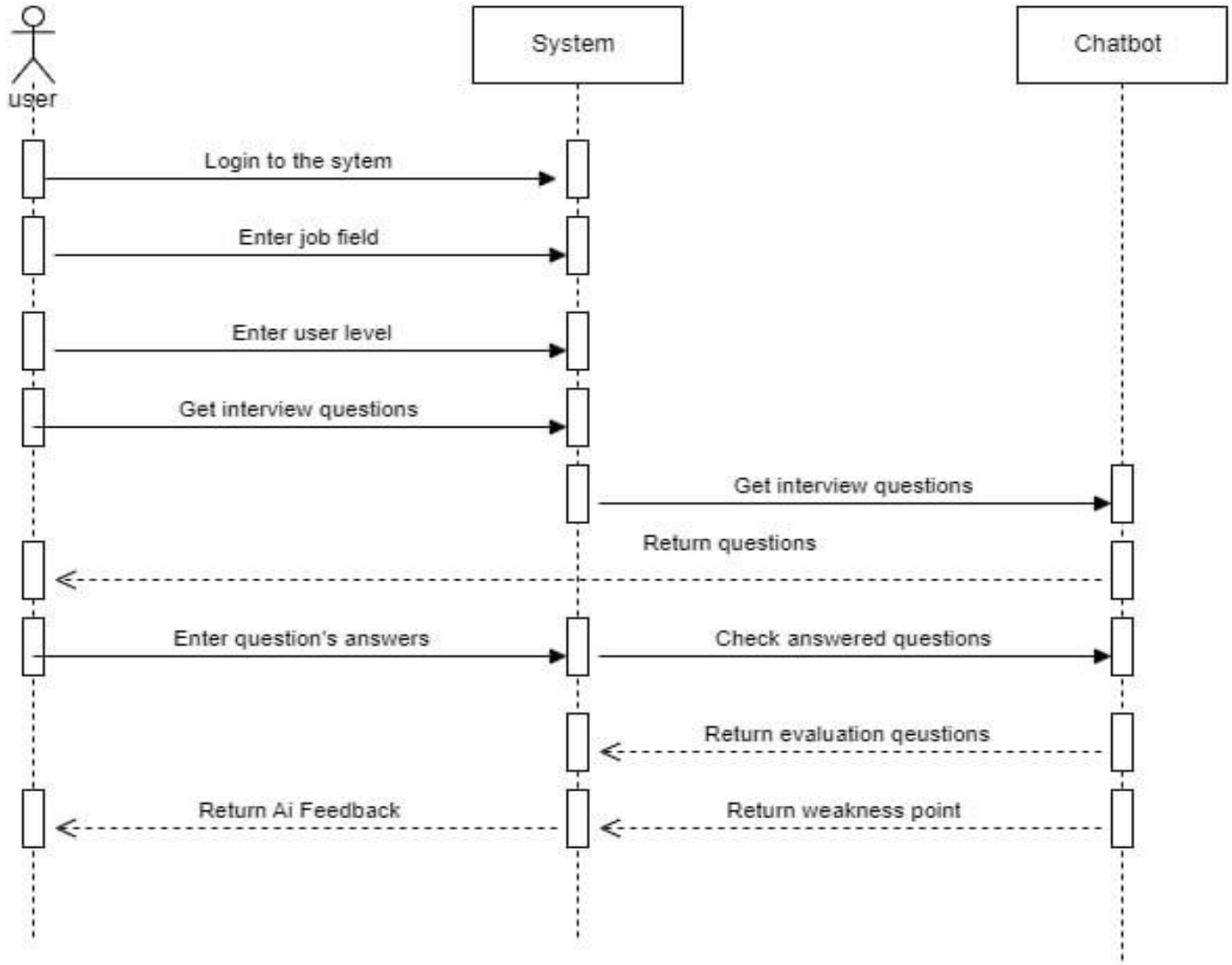


Figure 3.8 - Chatbot sequence diagram

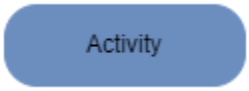
3.5.4 Activity Diagrams

Having an idea about the operations available for each user, we are going to dig deeper and see how they are done. The activity diagram will show the workflow of the system.

Activity diagram components:

1. **Initial State or Start Point**

- A small, filled circle followed by an arrow represents the initial action state or the start point for any activity diagram.

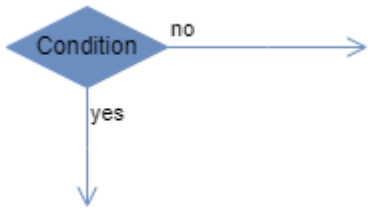


2. **Activity or Action State**

- An action state represents the non-interruptible action of objects.

3. **Action Flow**

- Action flows, also called edges and paths, illustrate the transitions from one action state to another. They are usually drawn with an arrowed line.



4. **Guards**

- In UML, guards are a statement written next to a decision diamond that must be true before moving next to the next activity. These are not essential, but are useful when a specific answer, such as "Yes, three labels are printed," is needed before moving forward.

Figure 3.9 show the activity diagram of login to the system.

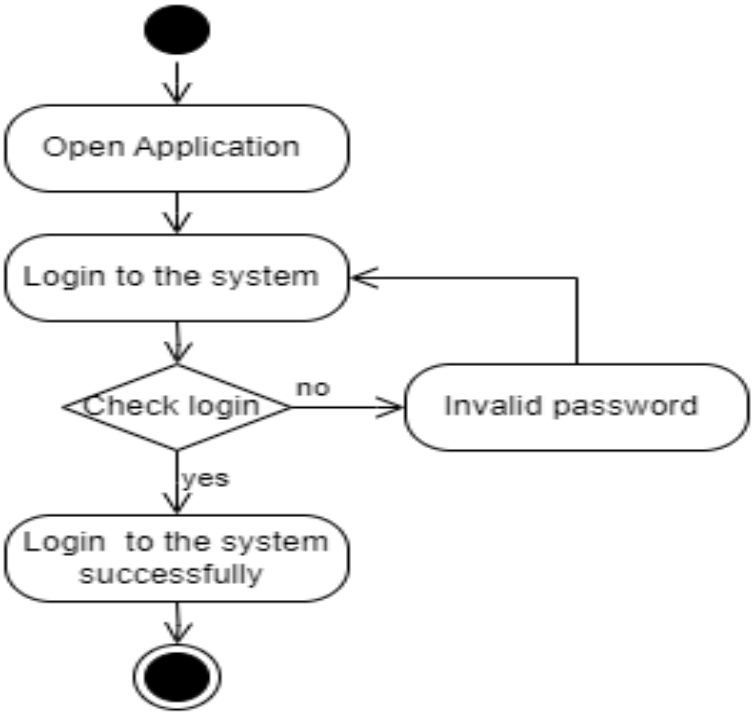


Figure 3.9 - login activity diagram.

Figure 3.10 show the activity diagram of company job requests.

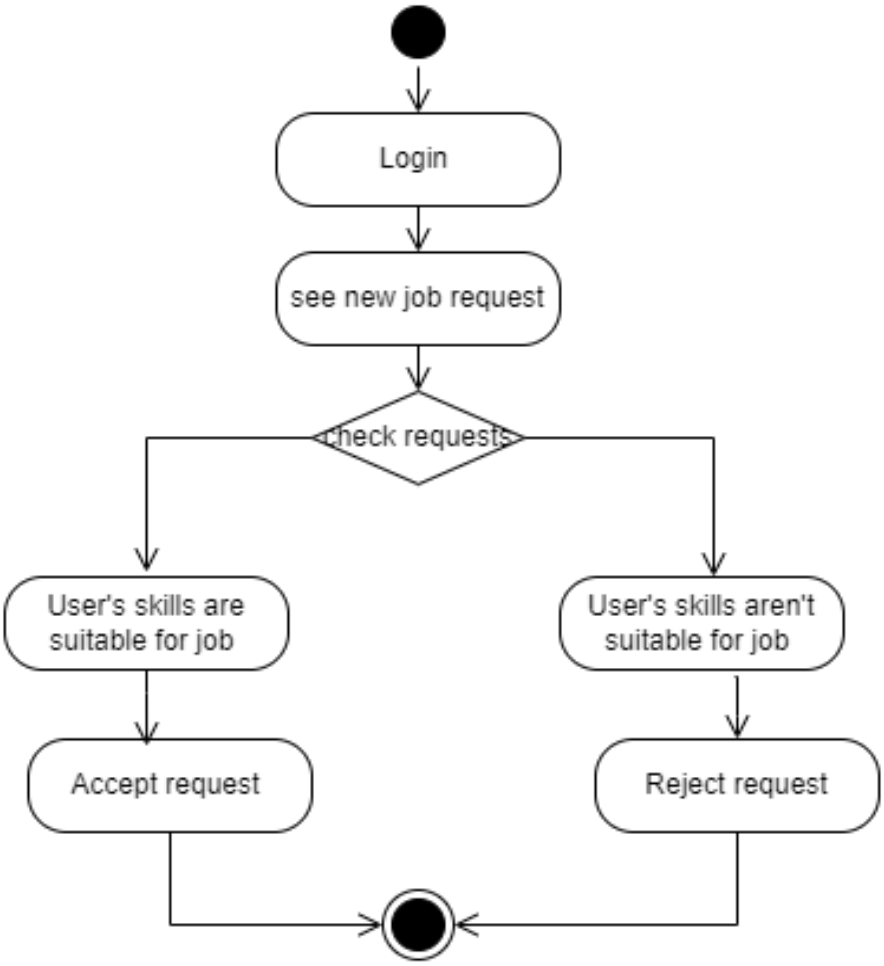


Figure 3.10 - Job Requests activity diagram

Figure 3.11 show the activity diagram of user services.

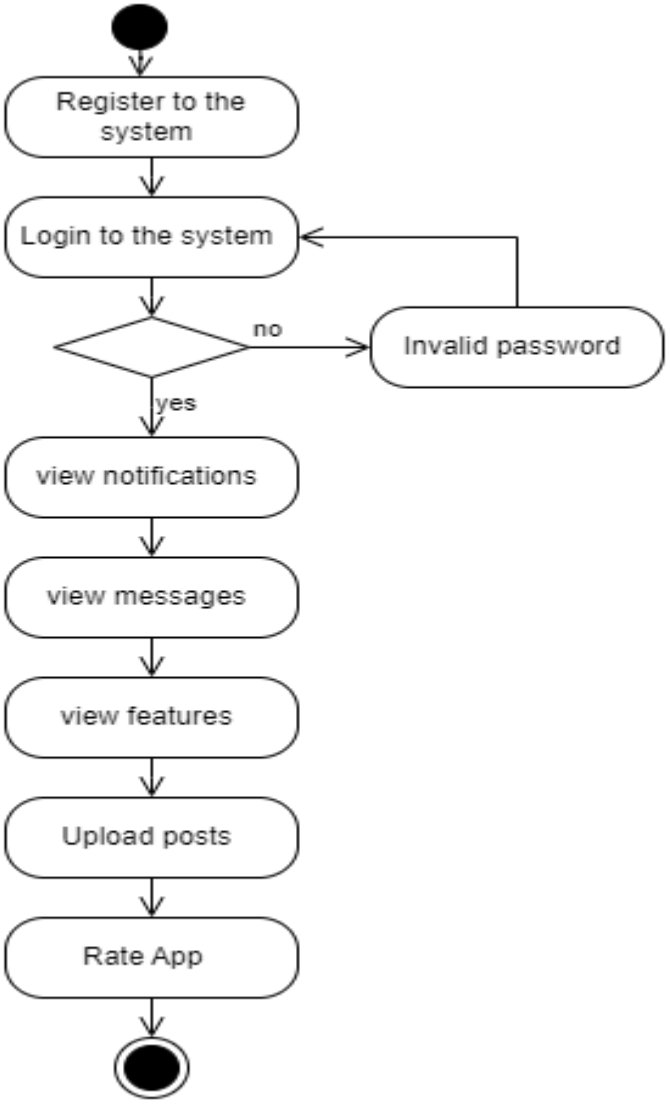
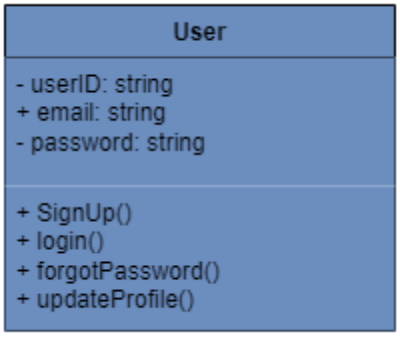


Figure 3.11 - User services activity diagram

3.5.5 Class Diagram

It describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects

The basics of class diagram:



- The class name is always shown in the first section, the **attributes** in the second, and **operations** in the third. **Attributes** are values that define a class. Classes can carry out processes known as **operations**.
- **Visibility** symbols are used to determine the accessibility of the information contained in classes.
- ***Note**: The "+" represents public operations, vice versa, "**-**" represents private operations. Plus, "#" is for the protected operations.*

Composition

- It is a form of aggregation where one class is dependent on another. One class is a part of the other.

Multiplicity

- Multiplicity is used to determine how many times an attribute occurs.

Figure 3.12 B2W class diagram

3.6 Tools and Languages

The development of our software application can be divided into two main parts, the design part and the implementation part. The design component includes designing graphics and designing the user interface for mobile applications. The implementation part includes programming languages, IDEs, frameworks, and libraries. The following list shows the tools needed for software development and a brief description of how to use them:

1. Software Ideas Modeler: is a smart CASE tool and diagram software that supports UML, SysML, ERD, BPMN, ArchiMate, flowcharts, user stories, wireframes.

2. Adobe XD: is a vector design tool for web and mobile applications, developed and published by Adobe Inc.

3.Draw io: is a powerful and intuitive online diagramming tool that allows users to create a wide variety of diagrams, including flowcharts, network diagrams, organizational charts, UML diagrams, mind maps, and more.

4. Android Studio: it is an IDE to build mobile applications for Android OS.

5. Flutter: Flutter is an open-source framework developed and supported by Google. Frontend and full-stack developers use Flutter to build an application's user interface (UI) for multiple platforms with a single codebase.

6. Firebase: Firebase is a Backend-as-a-Service (BaaS) app development platform that provides hosted backend services such as a real-time database, cloud storage, authentication, crash reporting, machine learning, remote configuration, and hosting for your static files.

7. Flask: Flask is a micro web framework written in Python. It is classified as a micro framework because it does not require tools or libraries. It has no database abstraction

layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

8. Figma: Figma is a powerful design tool that helps you to create anything: websites, applications, logos, and much more.

3.7 Summary

In this chapter, we explored the foundational concepts and methodologies for system analysis and development that are applied in our project.

System Analysis: We defined system analysis as the process of studying procedures or businesses to identify their goals and create efficient systems. It involves breaking down a system into its components to understand their functionality and interactions. This approach is closely related to requirements analysis and operations research.

System Requirement: we defined Functional and non-functional requirement and their user requirement of the system.

System Architecture: We described the system major components and how it interacts with the system.

Use Case Diagrams: We highlighted the purpose of use case diagrams in capturing the dynamic aspects of a system. These diagrams provide a high-level view of interactions between users and the system, identifying different types of users and use cases. Key components of use case diagrams include actors, system boundaries, use cases, and relationships (include, extend, generalization, and association).

Sequence Diagrams: We examined sequence diagrams, which represent communication between objects in terms of a sequence of messages. The main components of sequence diagrams include actors, lifelines, activations, call messages, return messages, synchronous messages, and asynchronous messages.

Activity Diagram: We examined activity diagram, which represent the workflow of stepwise activities and actions with support of choice, iterations and concurrency.

Class Diagram: We examined class diagram which represent the structure of the system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

In the end of the chapter, we listed the needed tools for us to build the system.

Chapter 4:
System design

4.1 Introduction

This chapter focuses on the design process of the user interface for both the mobile application and the website of the employment platform for people with disabilities. The design aims to be accessible, user-friendly, and tailored to the needs of two different user types: job seekers and companies.

4.2 Design Goals and Principles

- **Accessibility**: High contrast colors, readable font sizes, and clear structure to support users with visual or cognitive disabilities.
- **Simplicity**: Minimal design to reduce confusion and cognitive load.
- **Consistency**: Unified experience across both mobile and web platforms.
- **Clarity**: Easy navigation and clear call-to-actions for all types of users.

4.3 User Types

- **Job Seeker (User)**: A person with disabilities who browses and applies for jobs.

- **Company (Recruiter)**: A company that posts job offers and manages applicants.

Each user type has its own set of screens and flows based on their tasks

4.4 Interface Design

4.4.1 Mobile App – Job Seeker

4.4.1.1 Splash Screen

This is the initial screen displayed when the app launches. It contains the app logo and name, giving a quick brand introduction. It creates a smooth transition to the onboarding, as shown in Figure 4.1.

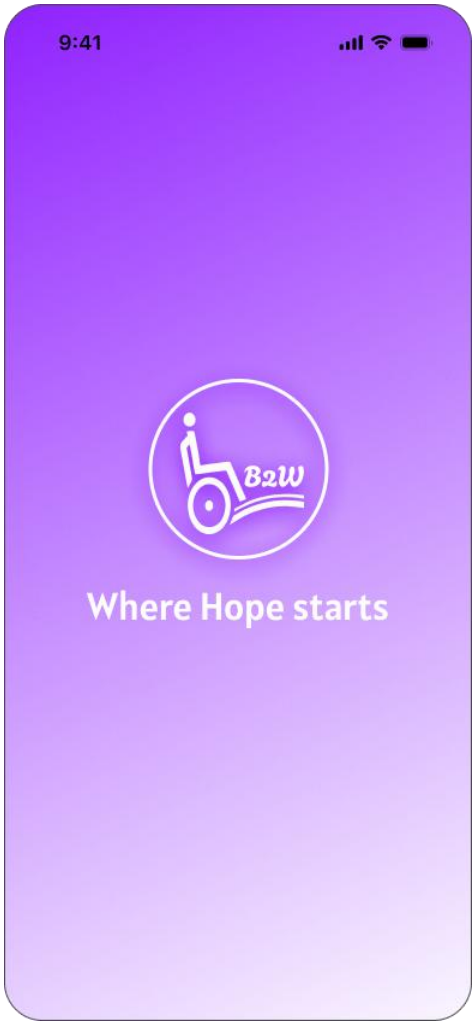


Figure 4.1 Splash Screen

4.4.1.2 Onboarding Screens

A series of welcome screens that introduce the app's purpose and key features, as shown in Figure 4.2.

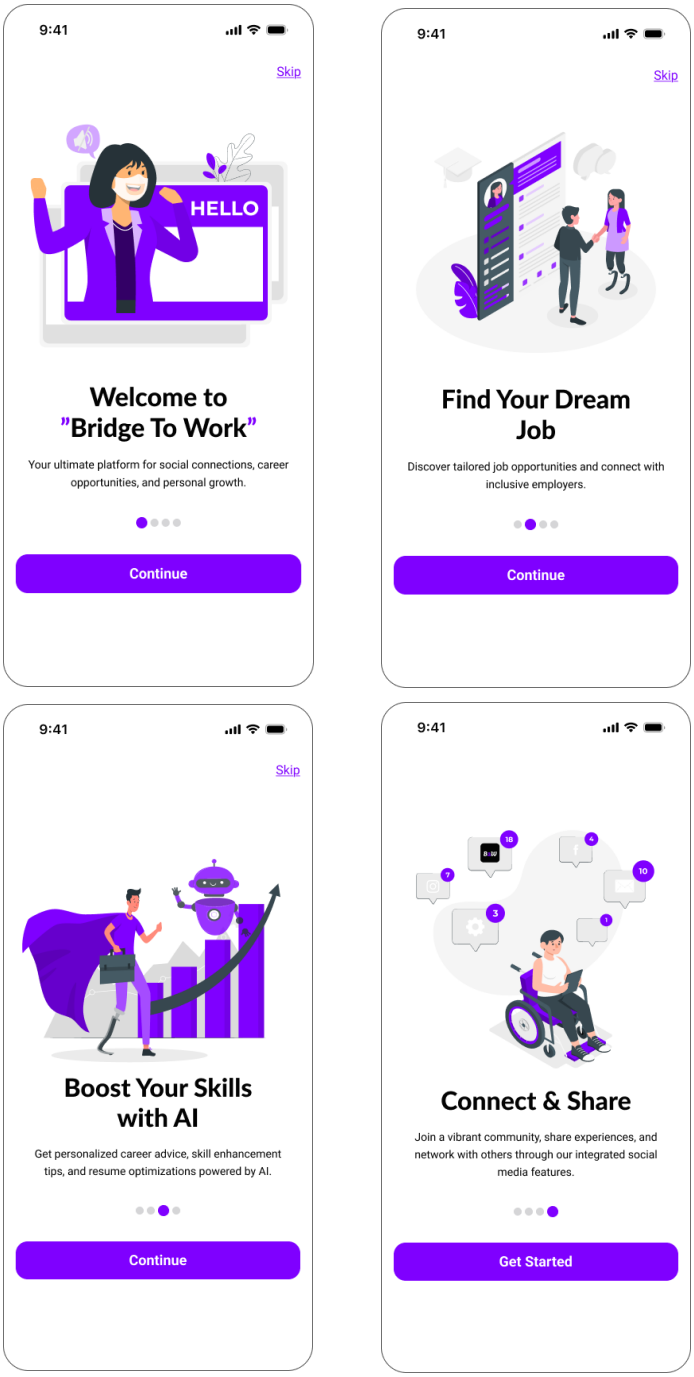


Figure 4.2 Onboarding Screens

4.4.1.3 Login Screen

A simple login form that allows existing users to access their accounts. It includes fields for email and password, as well as links to sign up or reset a forgotten password, as shown in Figure 4.3.

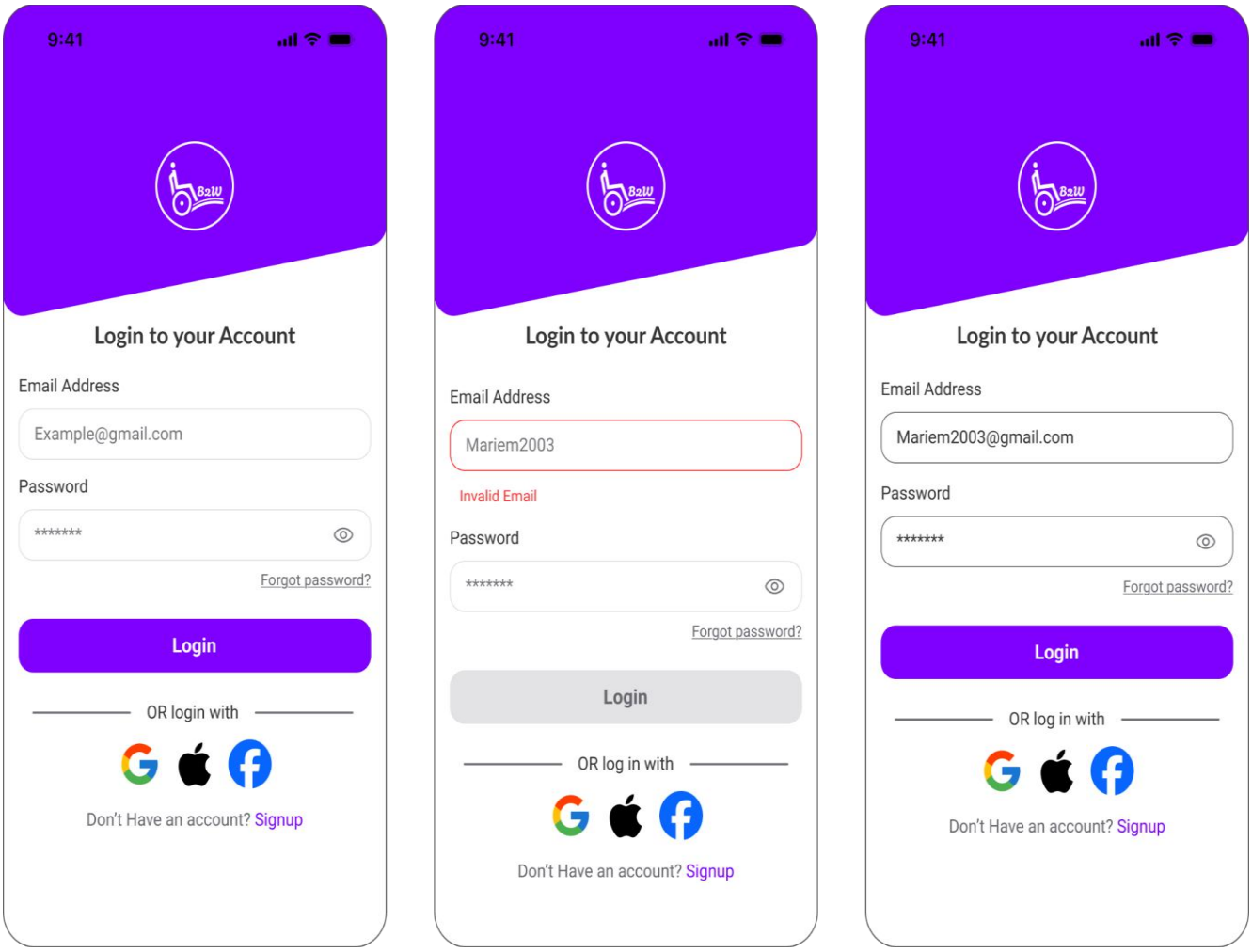


Figure 4.3 Login Screen

4.4.1.4 Forgot Password Screens

A screen that helps users recover access to their accounts. Users enter their registered email to receive a password reset link, as shown in Figure 4.4.

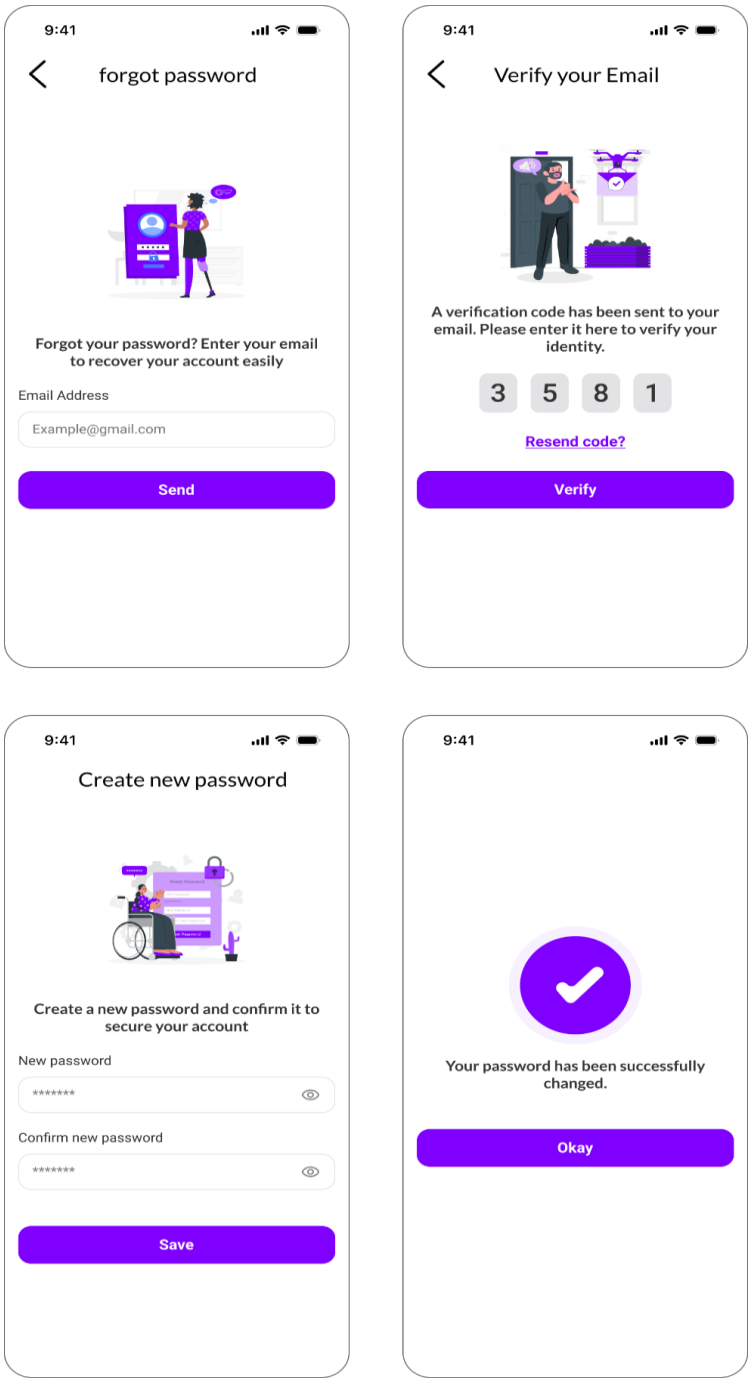


Figure 4.4 Forgot Password Screens

4.4.1.5 Sign Up Screens

A registration form for creating a new account. Users select their role (job seeker or company) and provide necessary information like email and password, as shown in Figure 4.5.

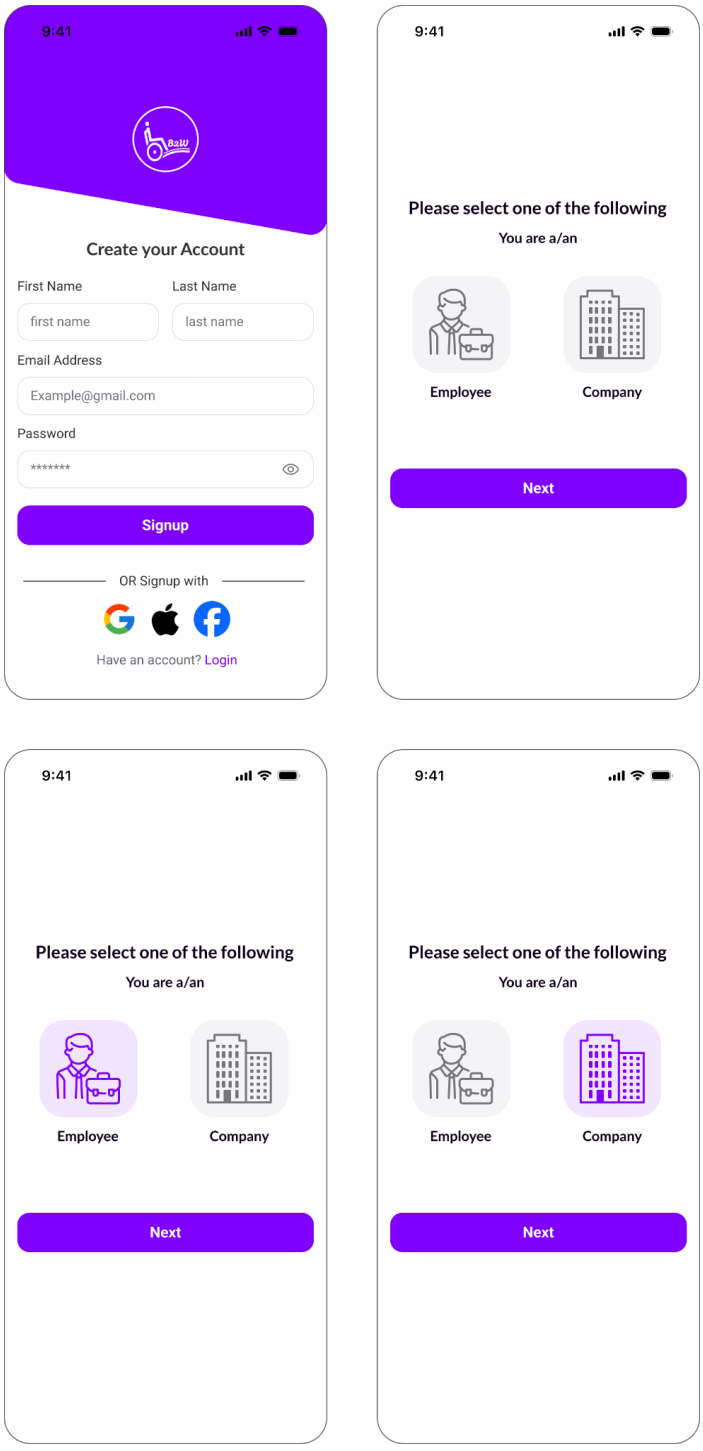


Figure 4.5 Sign Up Screens

4.4.1.6 Set Up User Profile Screens

After sign-up, this screen allows users to complete their profile based on their selected role, as shown in Figure 4.6.

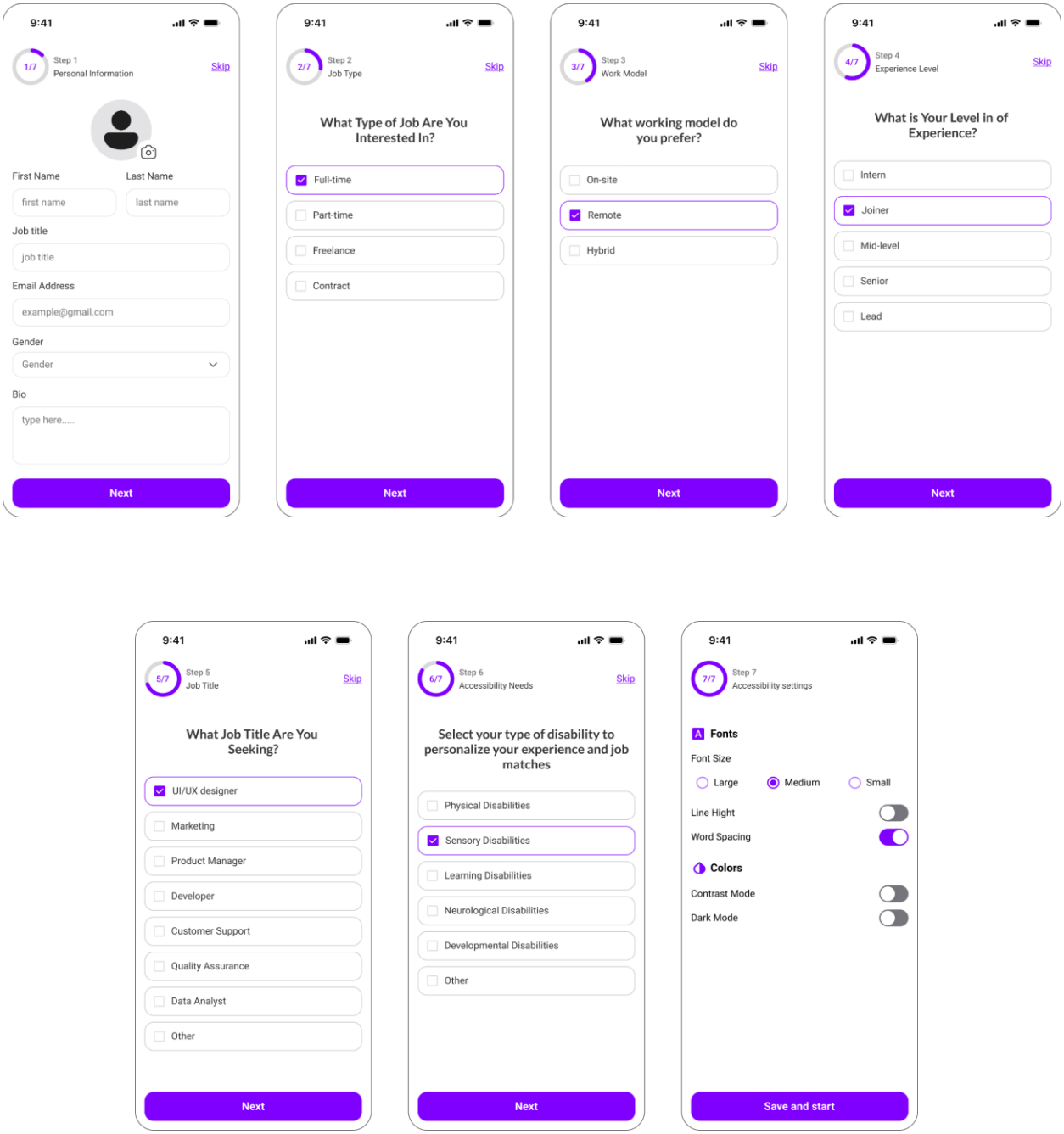


Figure 4.6 Set Up User Profile Screens

4.4.1.7 Home Screens

The Home screen serves as the central hub for job seekers within the mobile app. It provides quick access to all main features through a bottom navigation bar, which includes:

- Home Feed,
- Chat,
- Job Listings,
- Add New Post,
- User Profile,
- and Application Status (to track submitted job applications).
- AI Services

Inside the Home page, users can:

- Browse a community feed containing posts from other users (job seekers or companies), as shown in Figure 4.7,
- Interact with posts through likes and comments, promoting engagement, as shown in Figure 4.9,
- Search for people or companies using a global search feature, as shown in Figure 4.8,
- Access their notifications,
- and manage their Accessibility Settings through a dedicated screen (adjusting font size, color mode, etc.).

7.1 Home and Notifications Screens

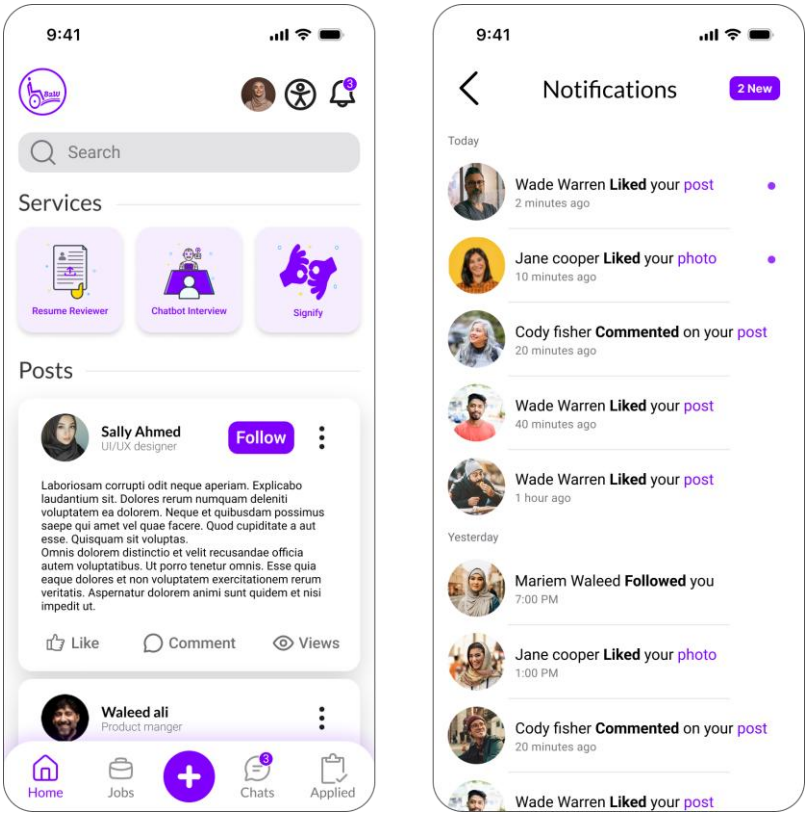


Figure 4.7 Home and Notifications Screens

7.2 Search Screens

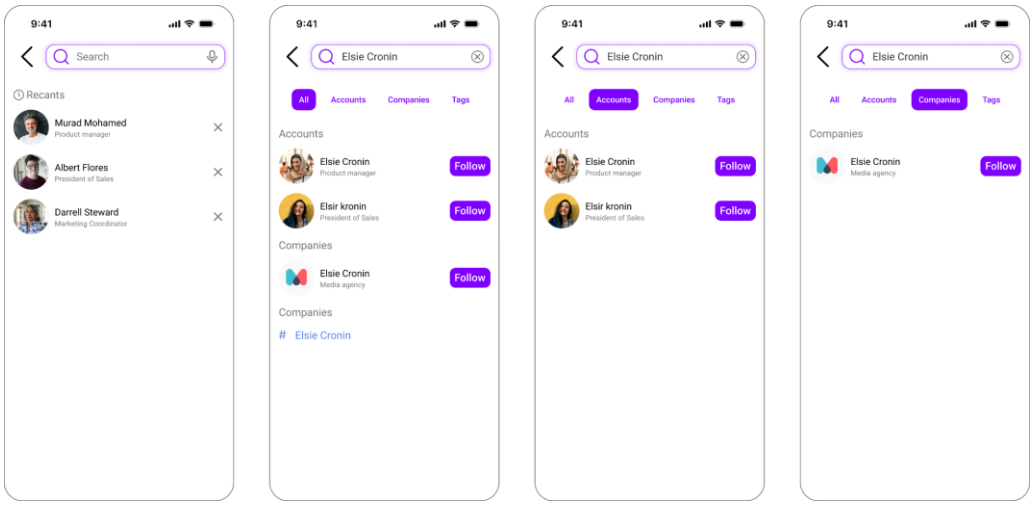


Figure 4.8 Search Screens

7.3 Add Post Screens

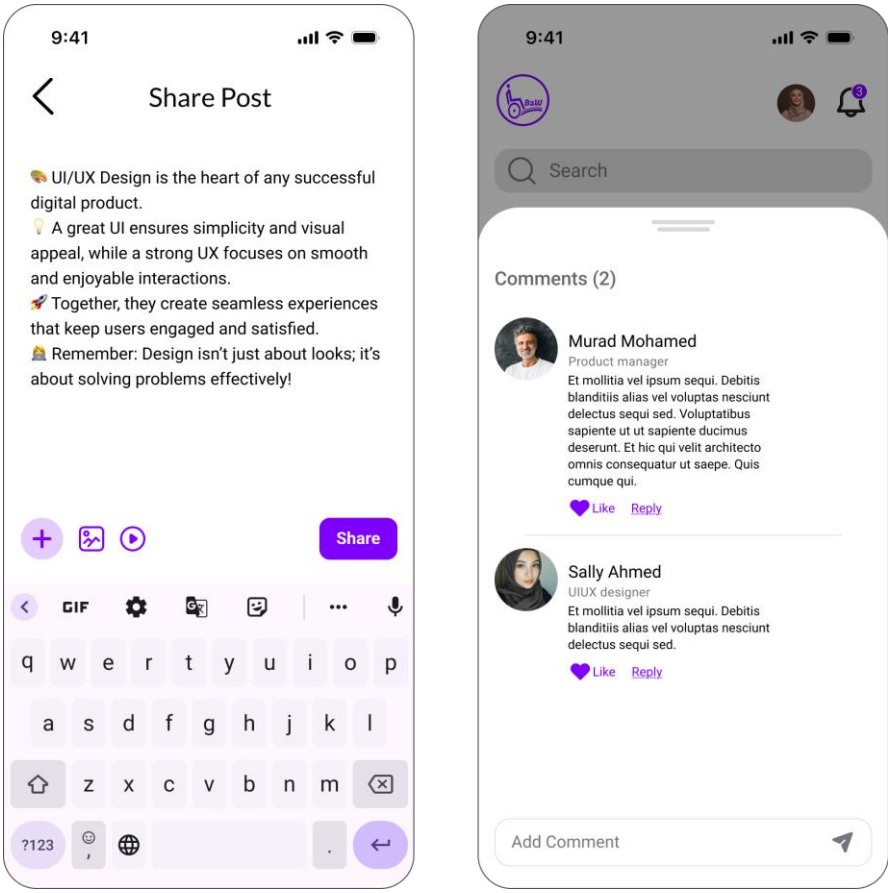


Figure 4.9 Add Post Screens

7.4 Ai Services Screens

- CV Analysis

This feature uses AI to analyze the user's uploaded CV or resume and automatically extract relevant information such as skills, experiences, and qualifications, as shown in Figure 4.10.

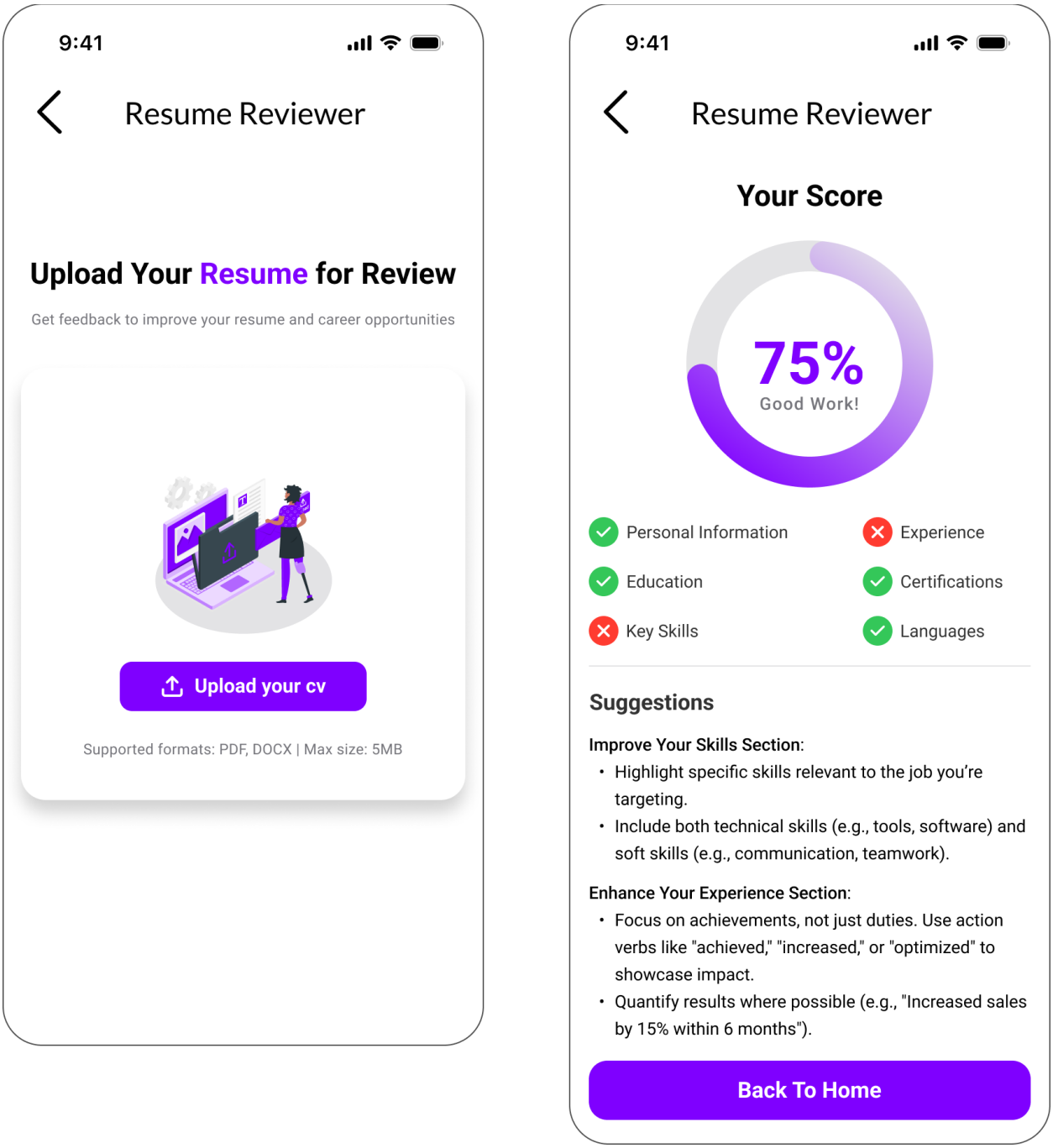


Figure 4.10 CV Analysis

- Sign Language Translator

To assist users with hearing impairments, the app includes an AI-powered sign language translator. It converts voice or text into sign language visuals (or vice versa), as shown in Figure 4.11.

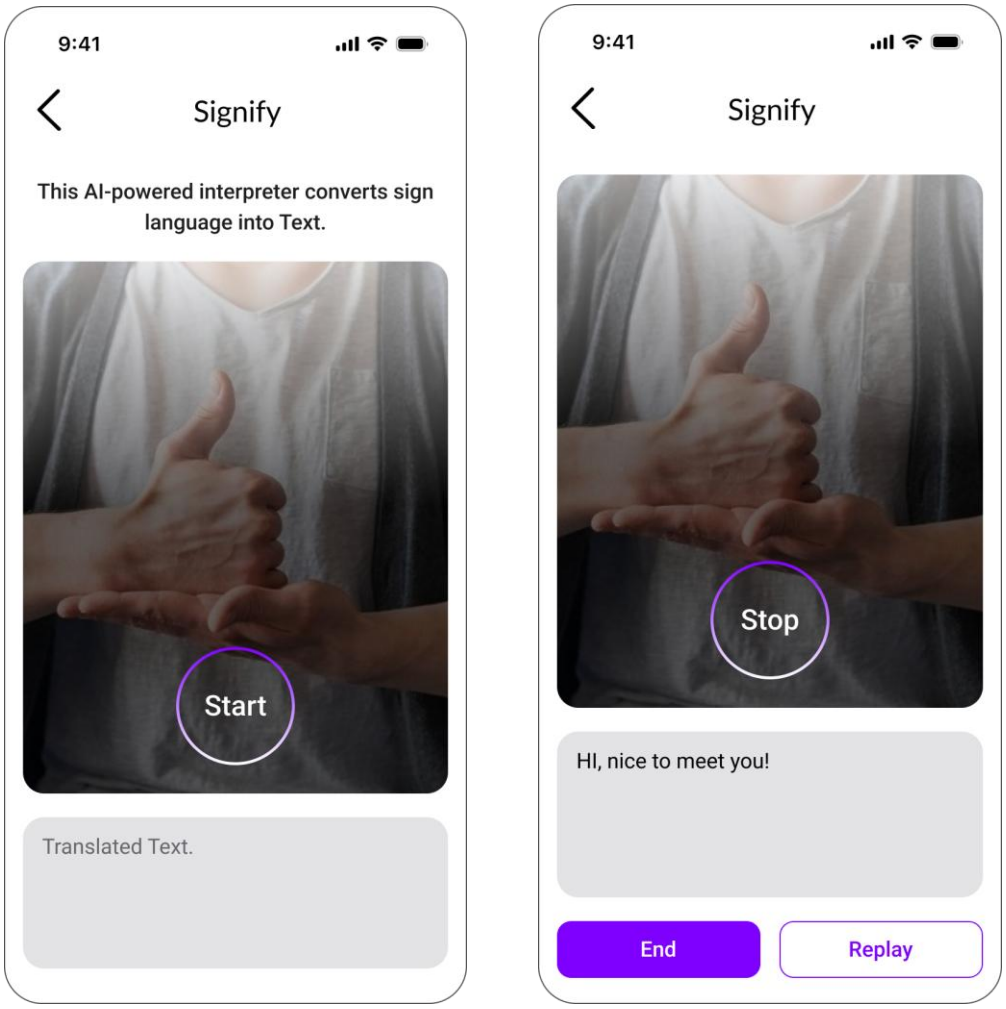


Figure 4.11 Sign Language Translator

- **Interview Chatbot**

The app features a smart chatbot that simulates real interview questions based on the user's job interest. It evaluates responses and gives tips on how to improve answers. This feature helps users prepare for interviews in a safe, private, and interactive way, as shown in Figure 4.12.

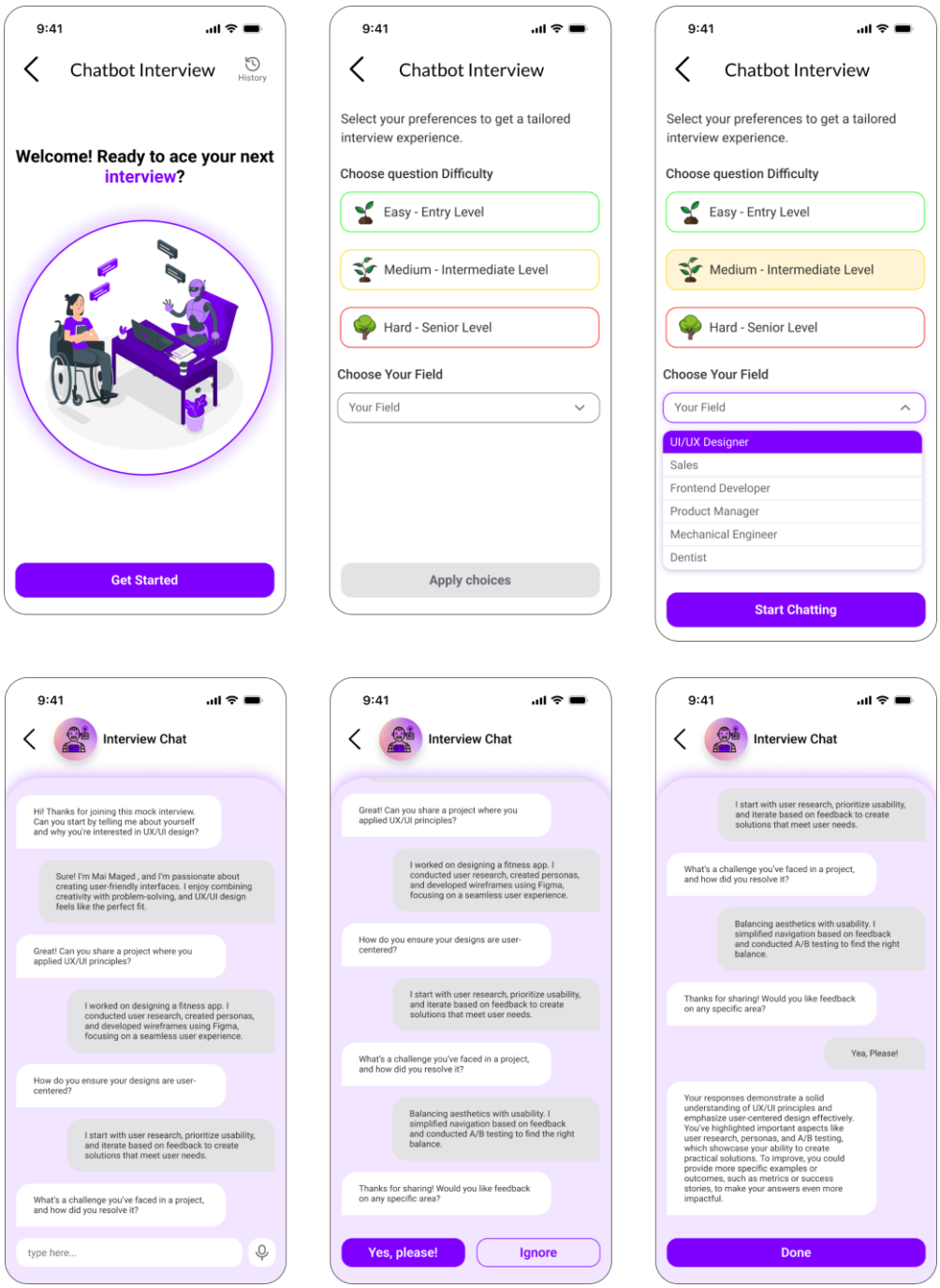


Figure 4.12 Interview Chatbot

4.4.1.8 Jobs Find Screens

The Job Find Pages is designed to help users explore, search for, and apply to job opportunities easily, as shown in Figure 4.13.

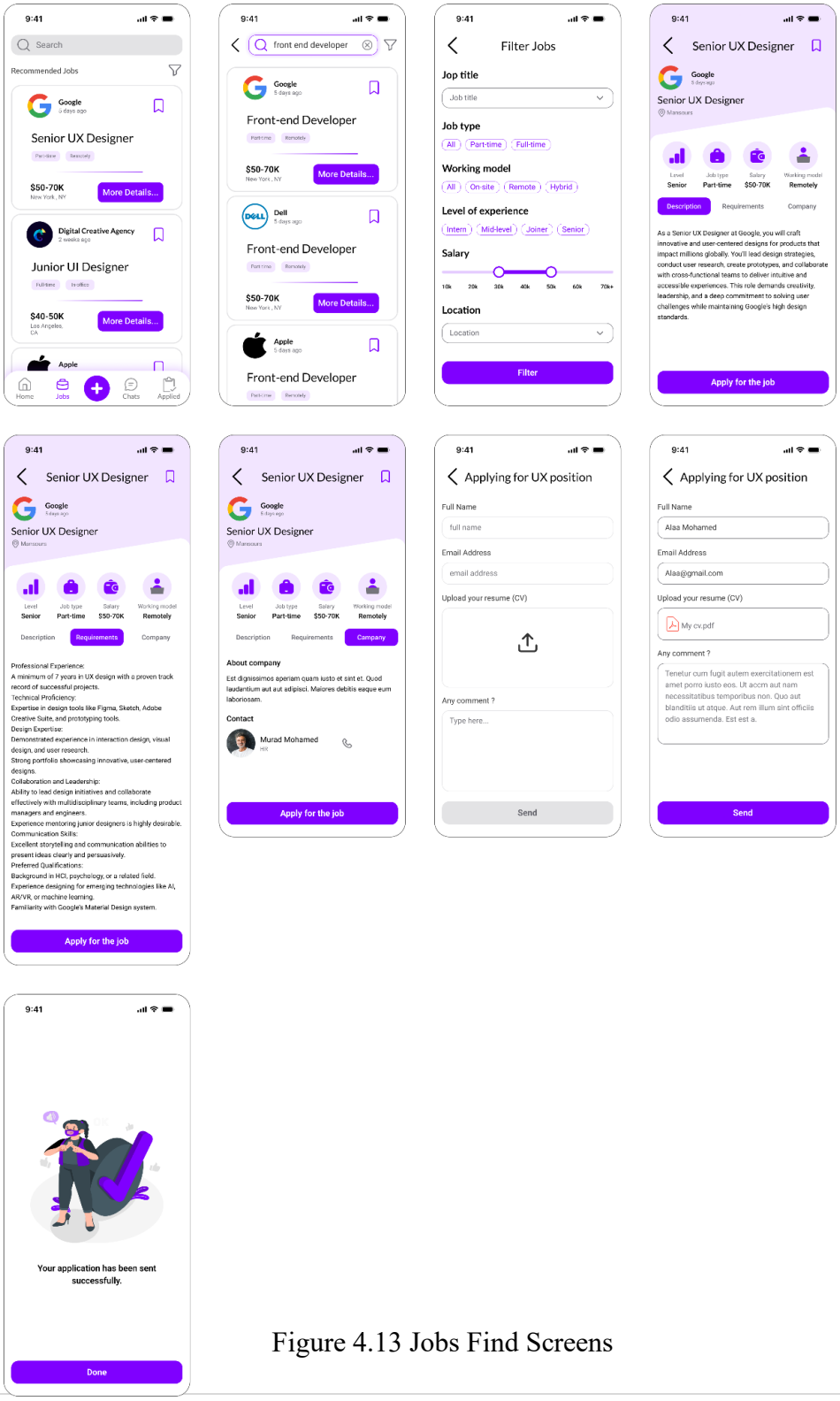


Figure 4.13 Jobs Find Screens

4.4.1.9 Applied jobs Status Screen

The Applied Jobs Status screen shows all the jobs the user has applied for, along with a clear status: Accepted, Rejected, or No Response Yet, as shown in Figure 4.14.

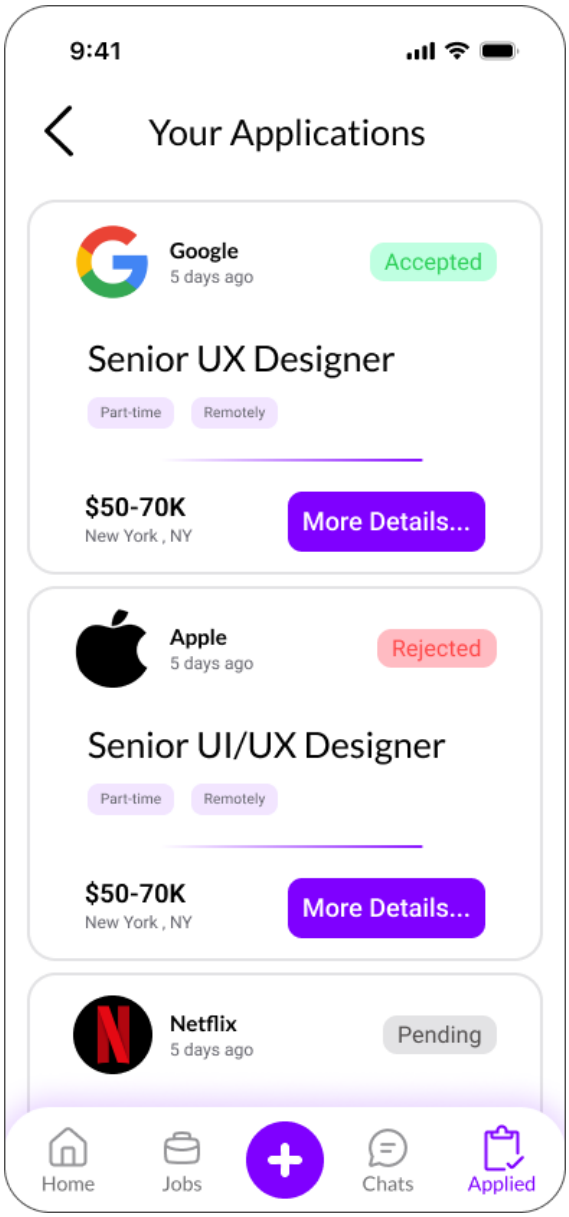


Figure 4.14 Applied jobs Status Screen

4.4.1.10 Chat Screens

The Chat Screens allow users to communicate directly with companies or other users through real-time messaging.

Each conversation shows the latest messages, and users can view full chats, search for new contacts, and stay connected easily, as shown in Figure 4.15.

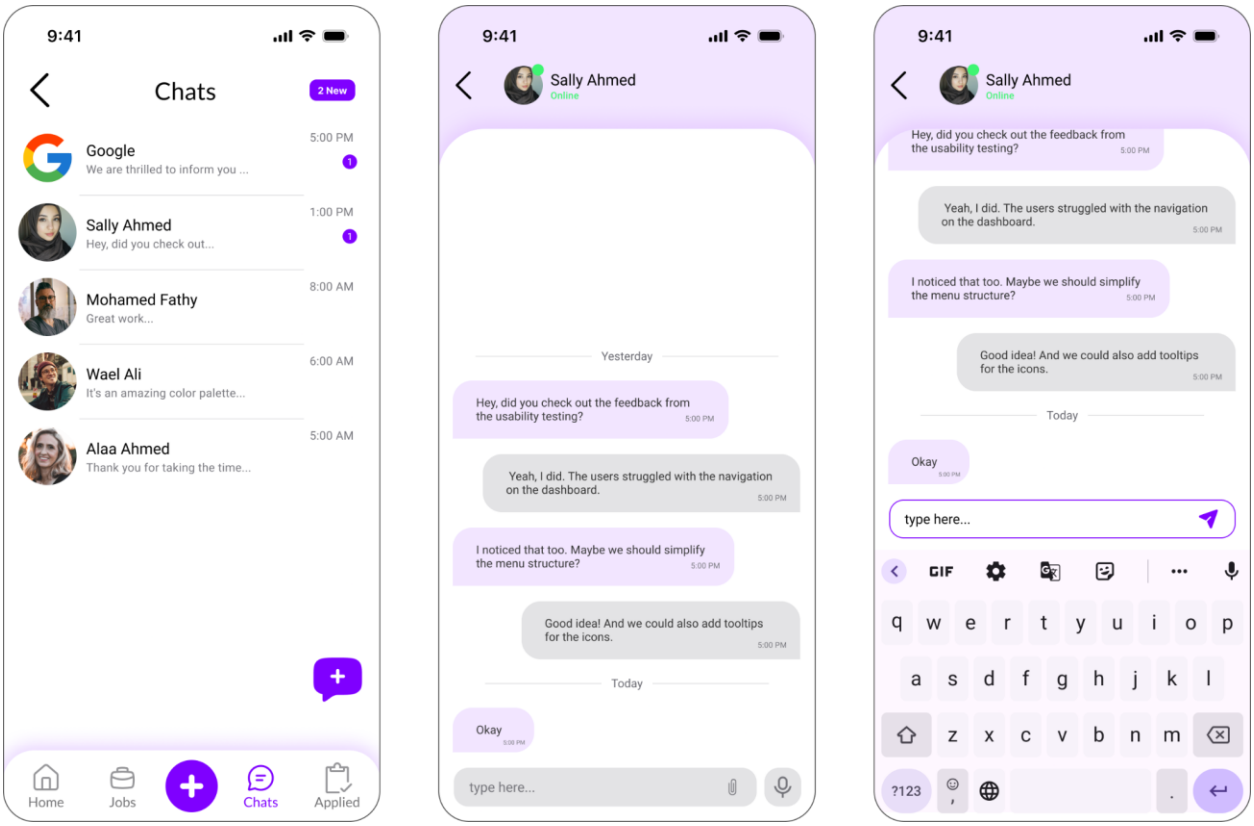


Figure 4.15 Chat Screens

4.4.1.11 User Account screens

The User Account screens display the user's shared posts, professional information (such as skills, experiences, and certifications), and a dedicated Achievements section showcasing projects and milestones, as shown in Figure 4.16. Users can also view their saved jobs and have full control to edit all profile details at any time, as shown in Figures 4.17 and 4.19.

11.1. User Account

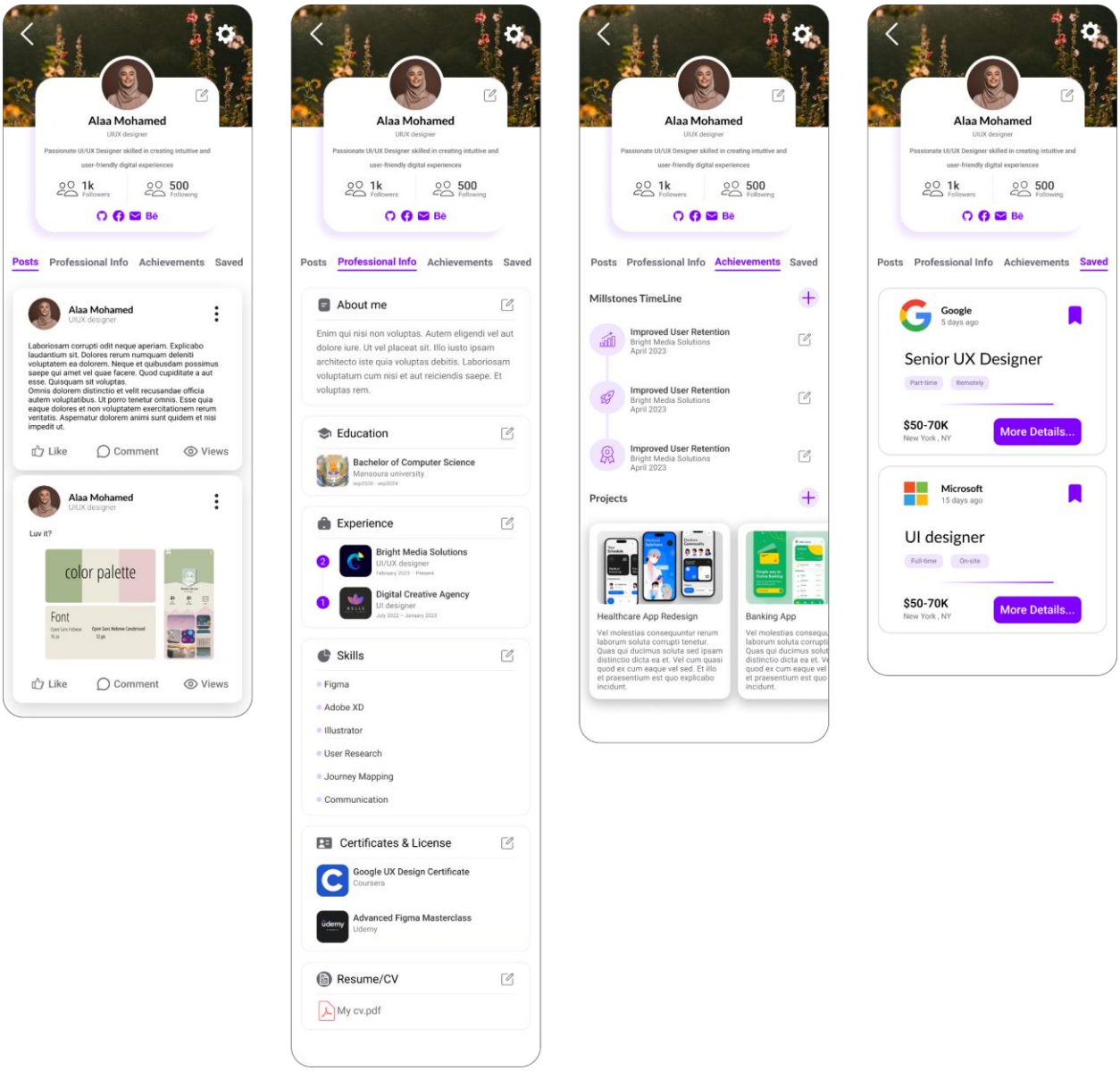


Figure 4.16 User Account

11.2. User Account (Edit Information)

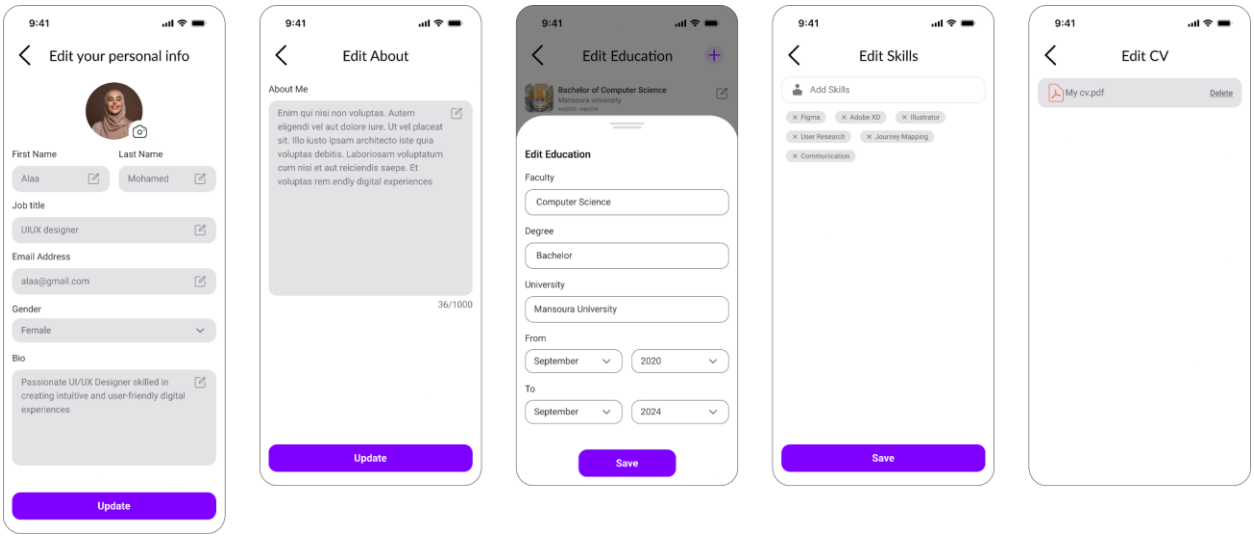


Figure 4.17 User Account (Edit Information)

11.3. User Account (Settings)

The Settings screen allows users to manage their account preferences, including privacy, notifications, language. They can also change their password, and log out from the app securely at any time, as shown in Figure 4.18.

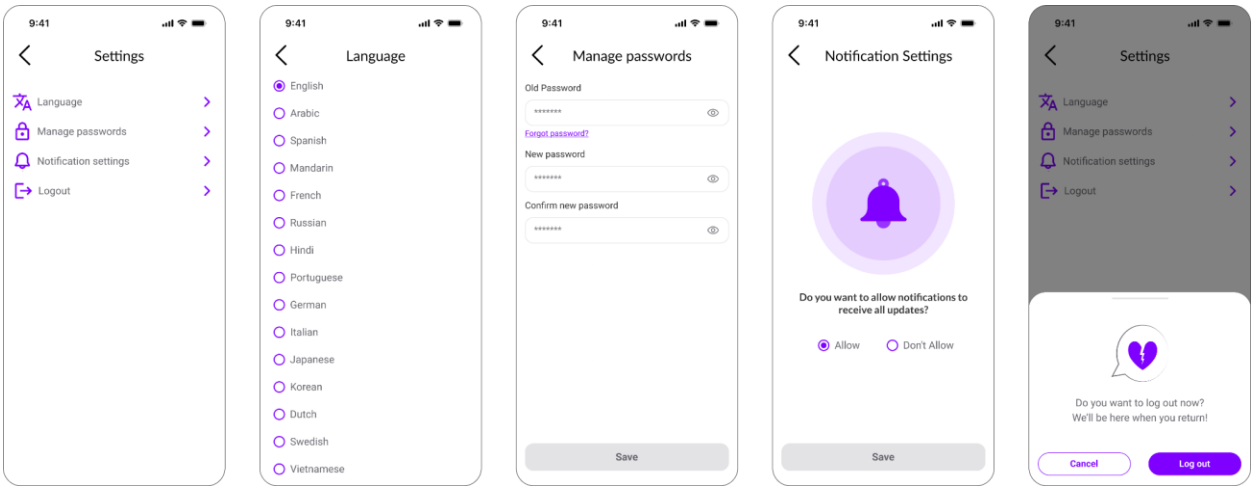


Figure 4.18 User Account (Settings)

11.4. User Account (Followers & Followings)

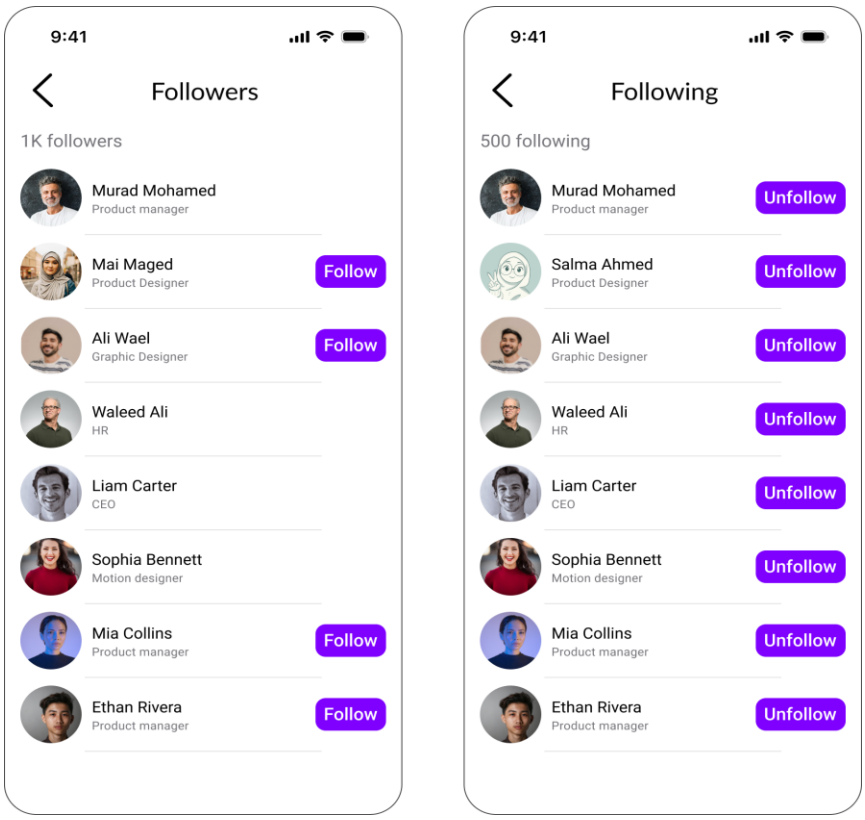


Figure 4.19 User Account (Followers & Followings)

4.4.1.12 Company Account View

The Company Profile screen displays general information about the company, along with available accessibility features it provides, such as wheelchair access or remote-friendly options. Users can also view the available job openings, employee list, and reviews submitted by other users about the company, as shown in Figure 4.20.

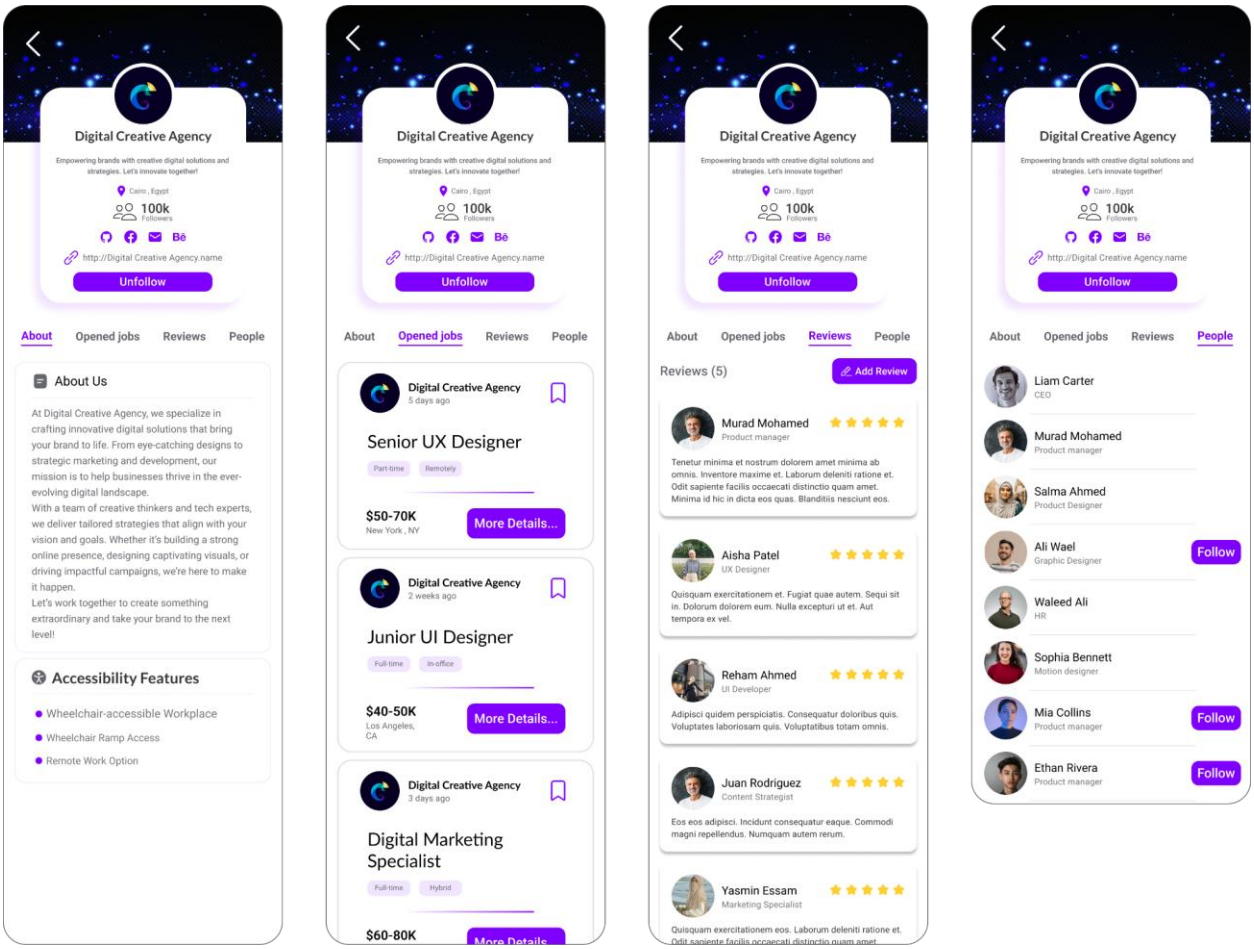


Figure 4.20 Company Account View

4.4.2 Mobile App – Company

4.4.2.1 Company Account setup

After signing up, the company sets up its profile information, including general details like name, location, and industry. It can also add its available accessibility tools and list the current employees working in the company, as shown in Figure 4.21.

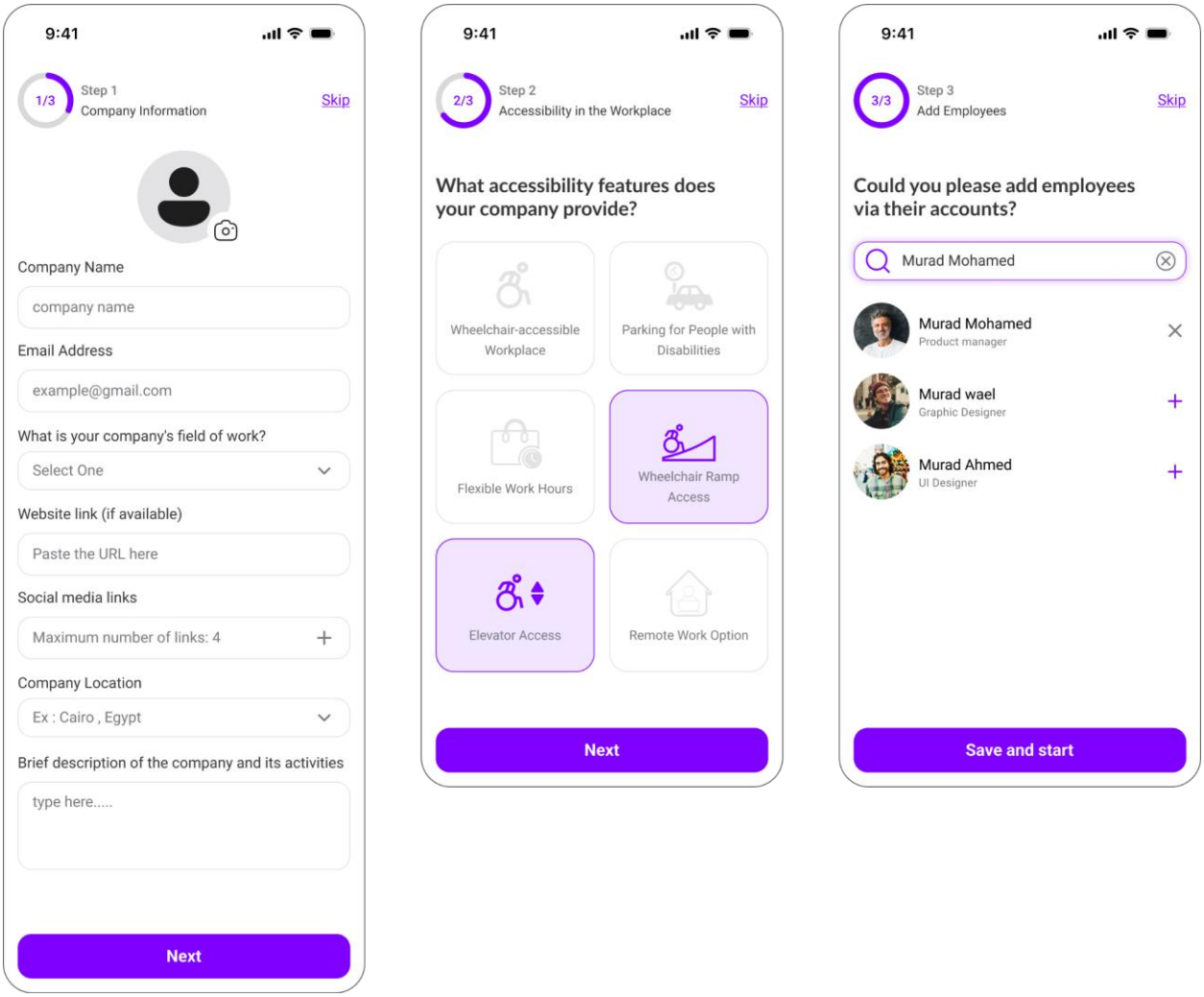


Figure 4.21 Company Account setup

4.4.2.2 Home Page

The Company Home Page displays all received job applications from users, allowing the company to review and manage candidates easily.

It also includes a notifications center and a search feature to quickly find applicants or filter through job-related updates, as shown in Figure 4.22.

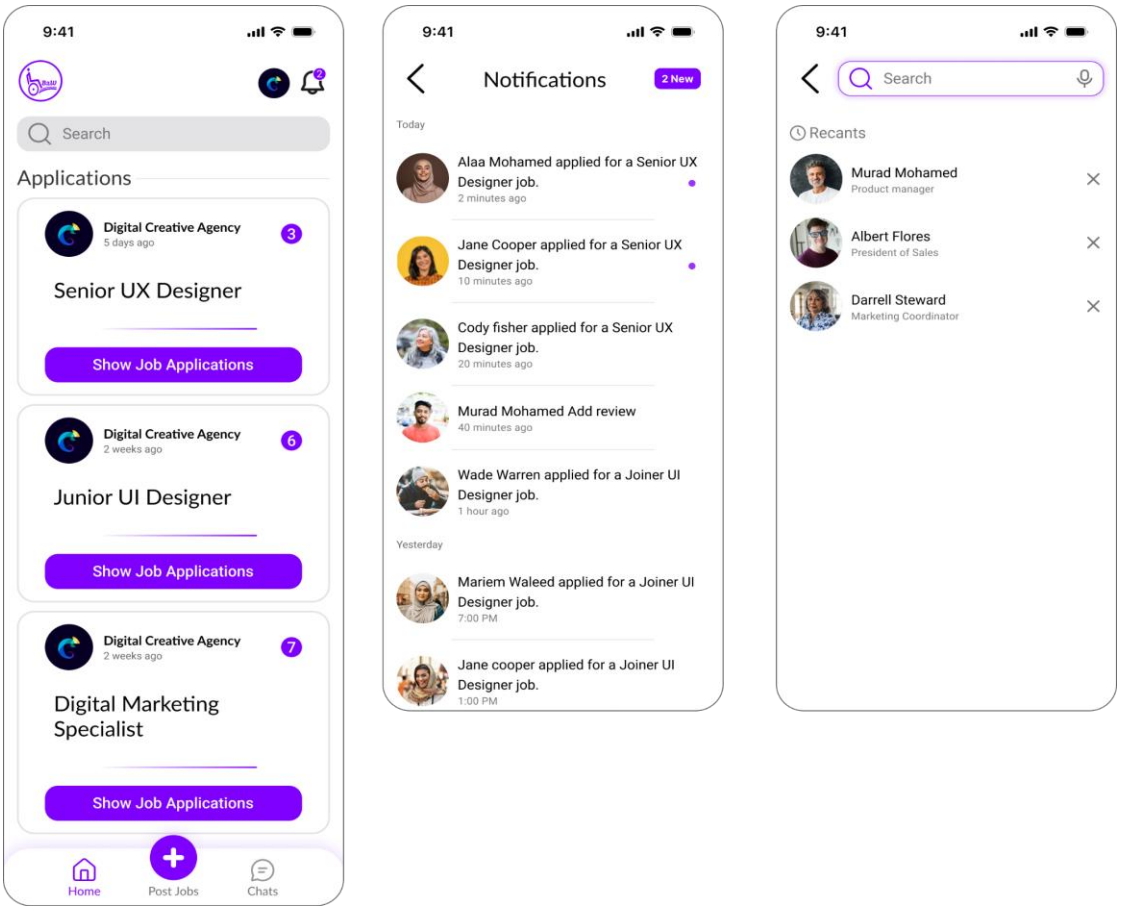


Figure 4.22 Home Page

When the company opens a job application, it can view the applicant's submitted form with personal and professional details, as shown in Figure 4.23.

The company can also visit the applicant's profile and respond with an acceptance or rejection message directly through chat.

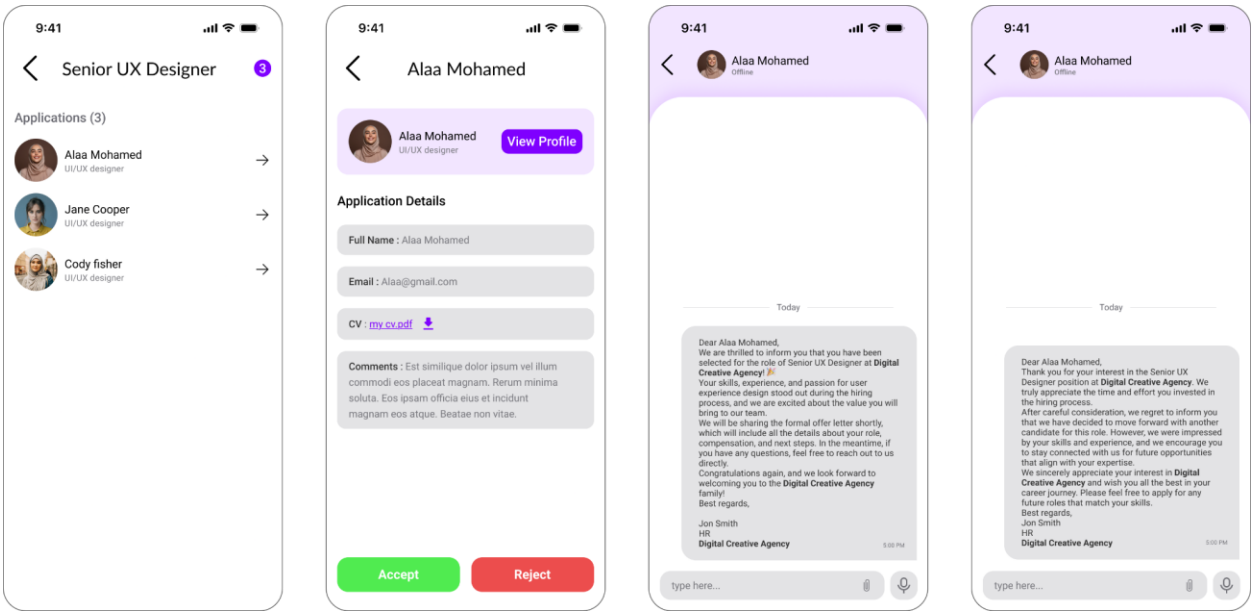


Figure 4.23 job application

4.4.2.3 Upload Job

The company can post new jobs by filling out a form with details like job title, description, location, and requirements. Once submitted, the job appears in its final user-facing format, and the company still has the option to edit or update it later if needed, as shown in Figure 4.24.

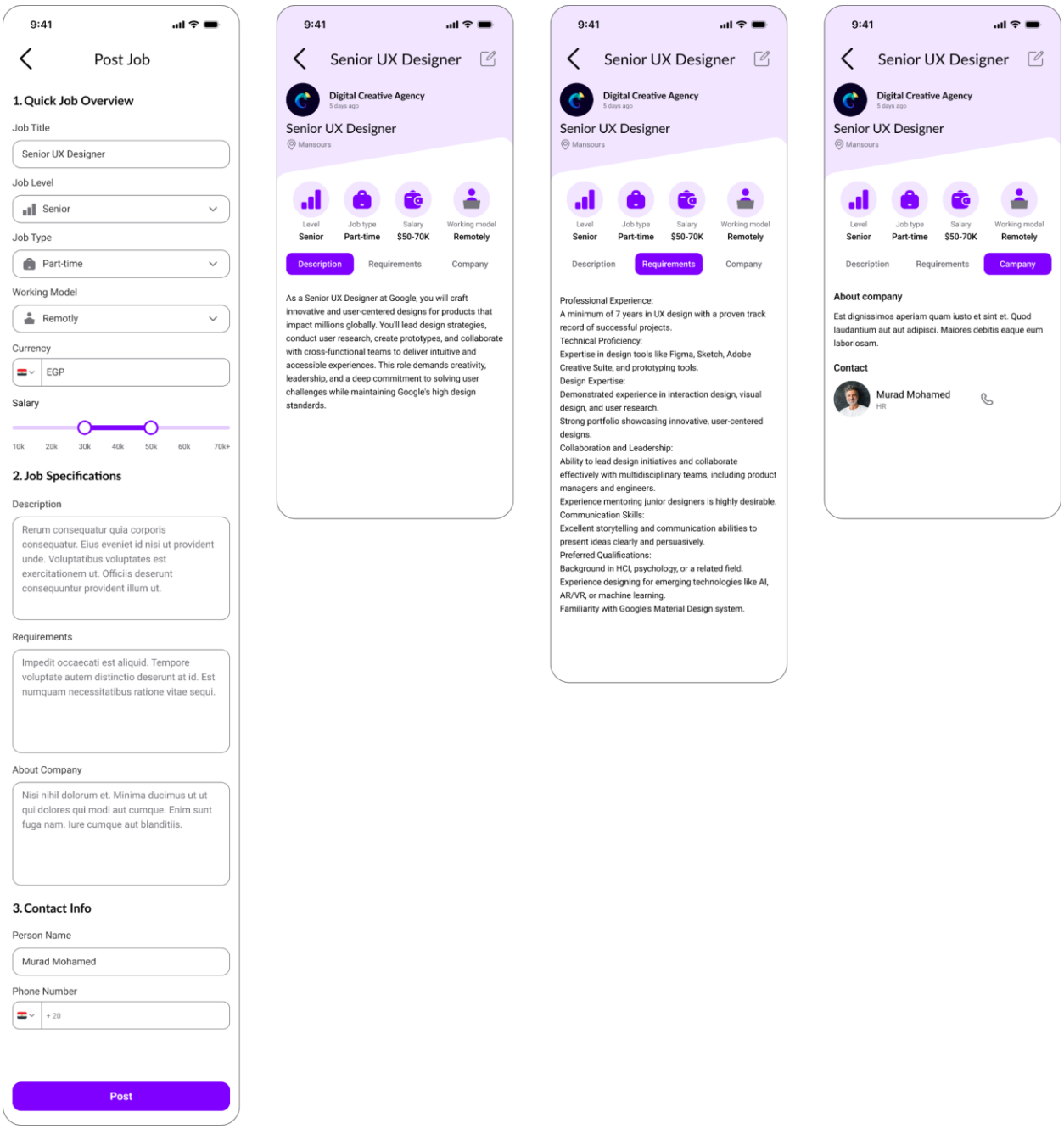


Figure 4.24 Upload Job

4.4.3 Website – Job Seeker

4.4.3.1 Splash Screen

The platform's logo and name appear when entering the website, as shown in Figure 4.25.

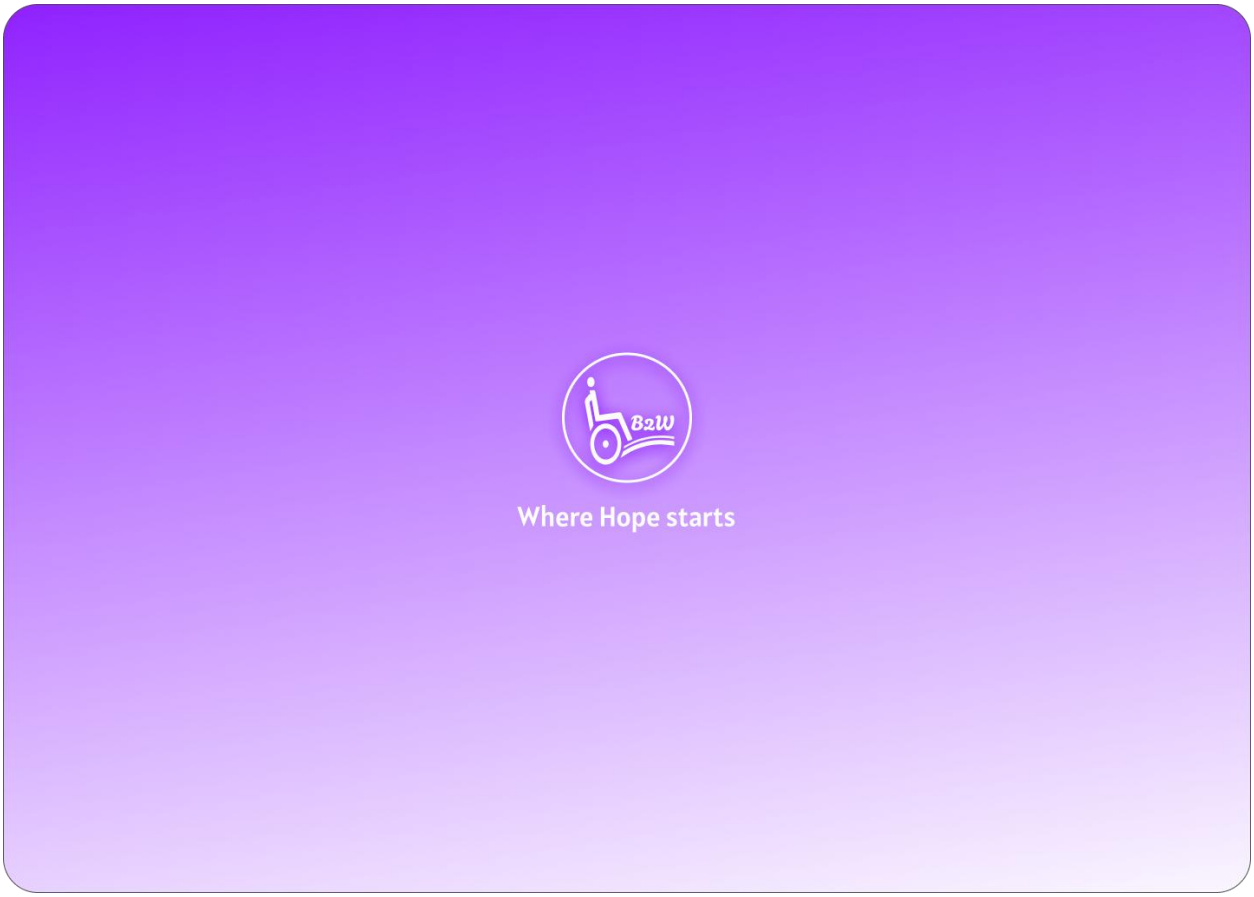


Figure 4.25 Splash Screen

4.4.3.2 Landing Page

The landing page features a clean design with the platform's mission statement and key services. It highlights accessibility and community support through simple visuals and concise text, as shown in Figure 4.26.

.

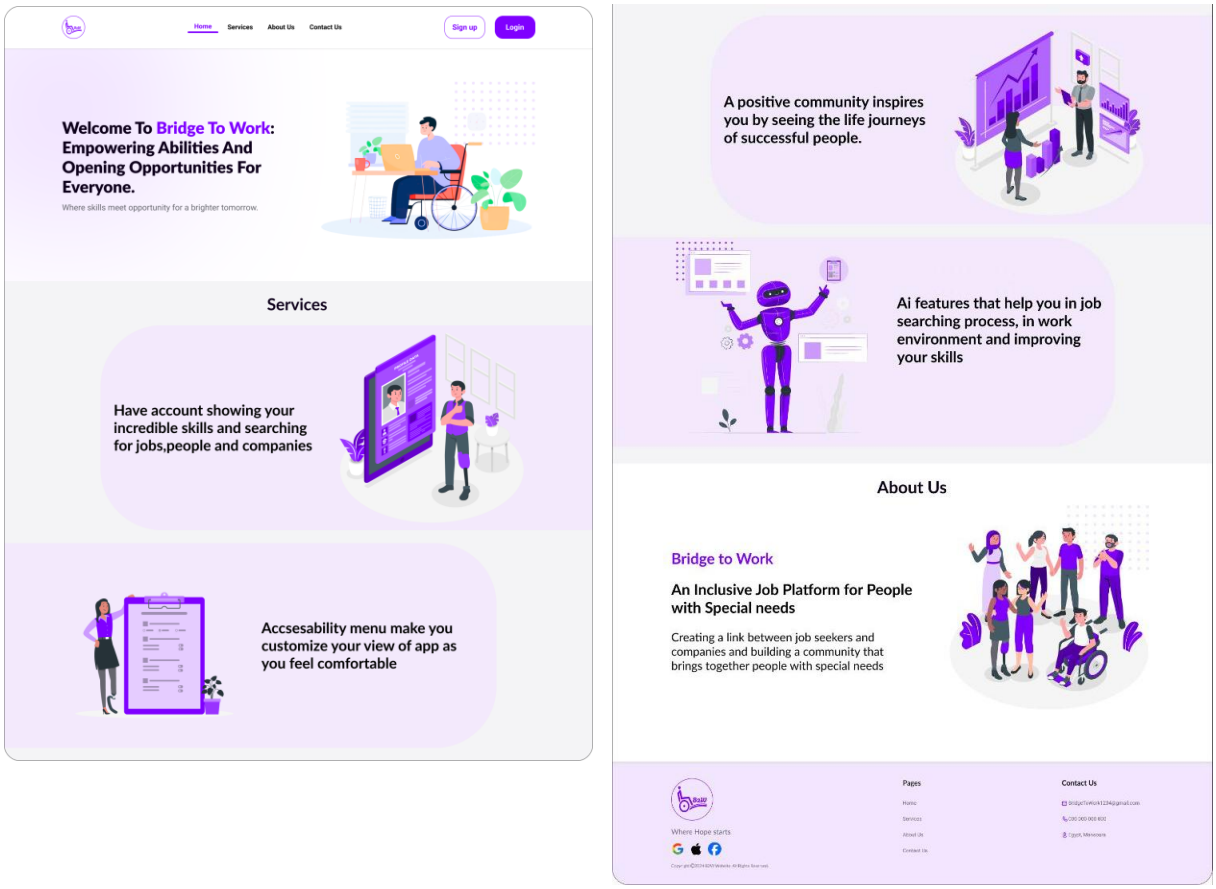


Figure 4.26 Landing Page

4.4.3.3 Home Screen

The home screen provides quick access to key features such as job listings, saved jobs, search functionality, chat, application status, and AI services. Users can browse and interact with community posts, search for opportunities, save preferred jobs, and manage accessibility settings, as shown in Figure 4.27.

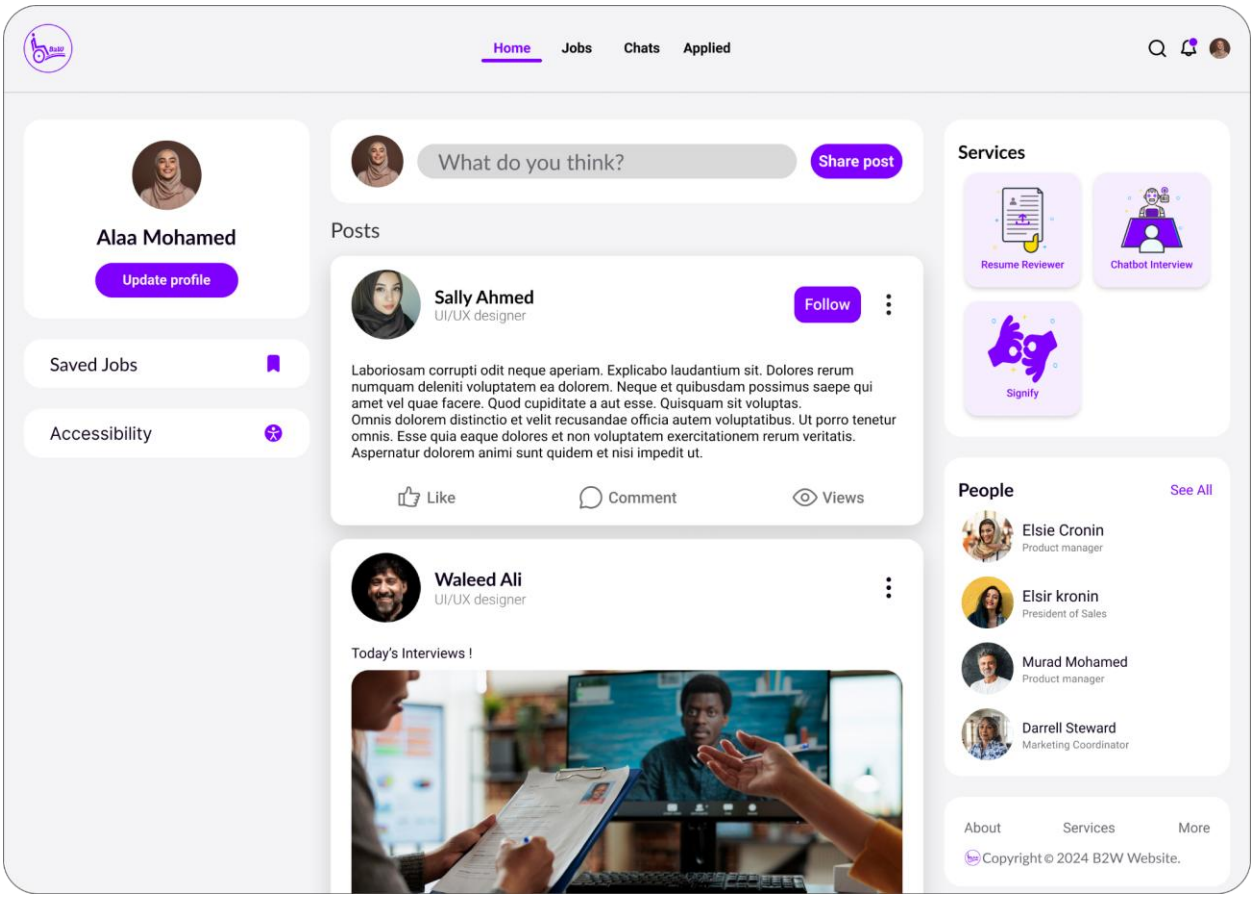


Figure 4.27 Home Screen

4.4.3.4 Jobs Find Screen

The jobs page displays recommended listings with key details (position type, salary range, and location), each featuring a “More Details” button. Users can save jobs for future reference and utilize filtering tools for efficient searching, as shown in Figure 4.28.

.

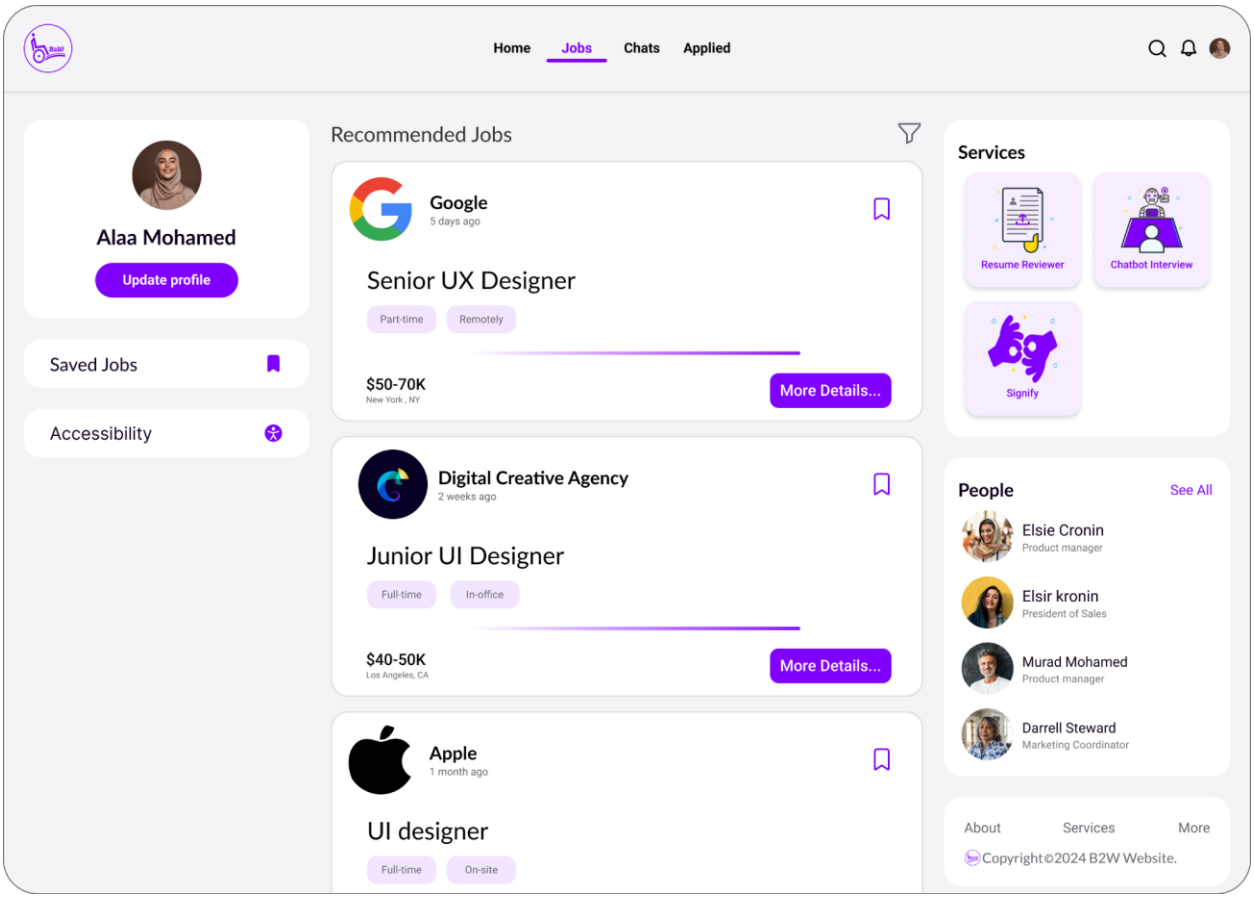


Figure 4.28 Jobs Find Screen

4.4.3.5 Chat Screen

The chat interface enables real time messaging with companies and other users, displaying conversation previews and connection statuses. Uses can view complete message histories, search contacts, and message conversations efficiently. A highlighted job acceptance message demonstrates successful hiring outcomes, as shown in Figure 4.29.

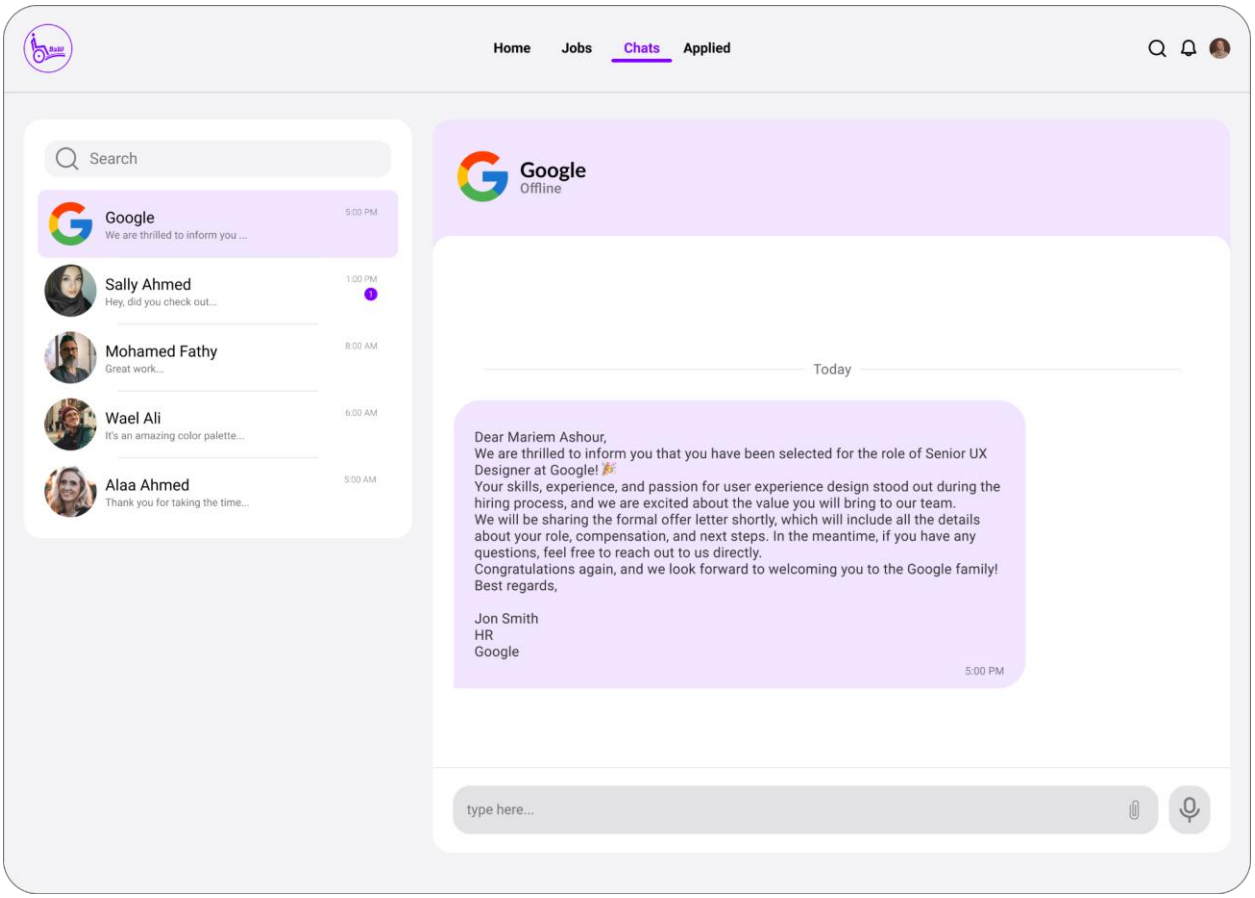


Figure 4.29 Chat Screen

4.4.3.6 User Account screen

The profile screen presents the user's professional identity and information, along with their shared posts, career achievements, and saved job listings. All profile sections are fully editable, giving users complete control over their information, as shown in Figure 4.30.

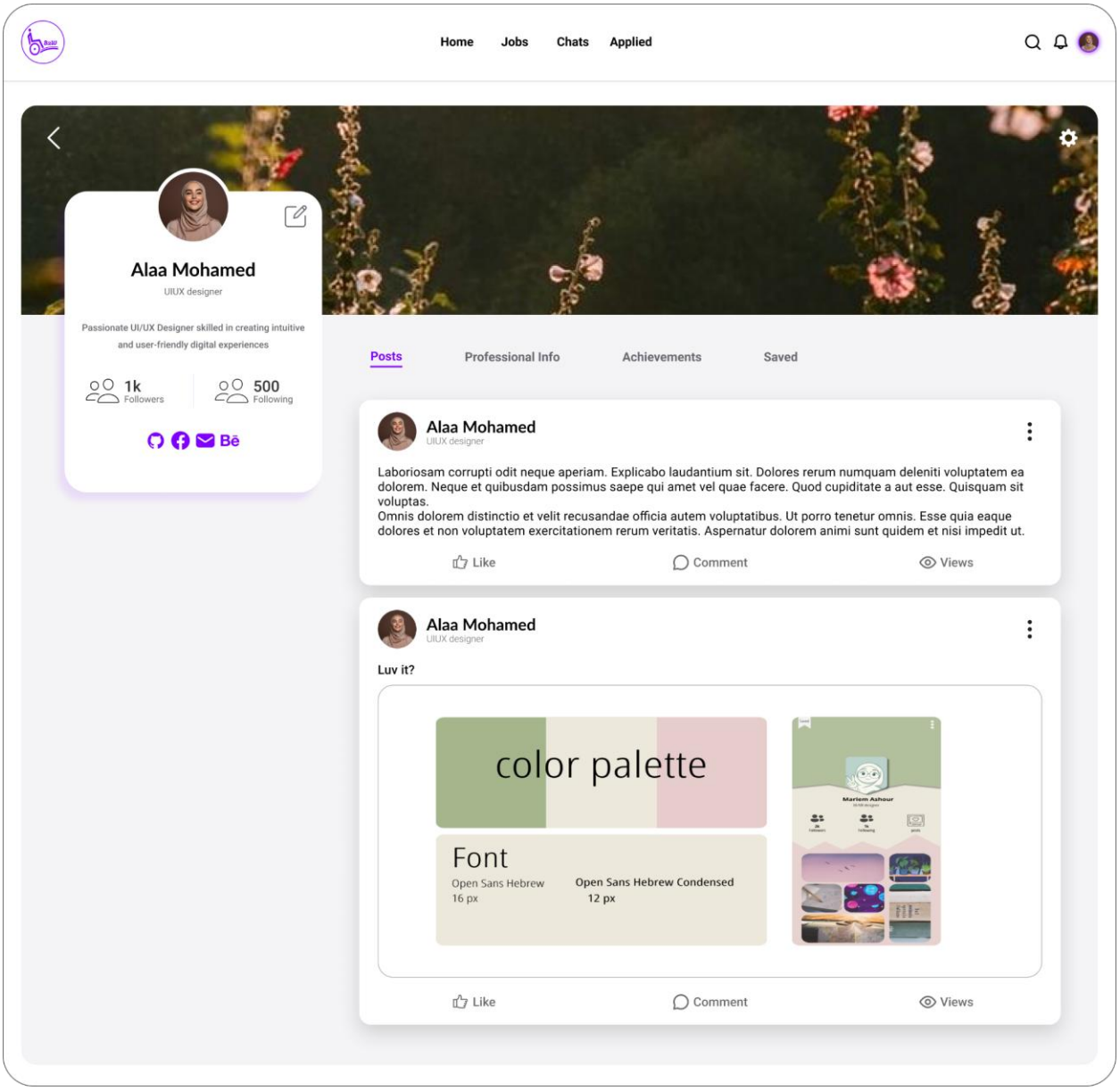


Figure 4.30 User Account screen

4.4.3.7 Company Account View

The company profile screen displays general information and accessibility features, including wheelchair access and remote-friendly options. Users can follow or unfollow the company, view available job openings, browse employee lists, and read reviews from other users, as shown in

Figure 4.31.

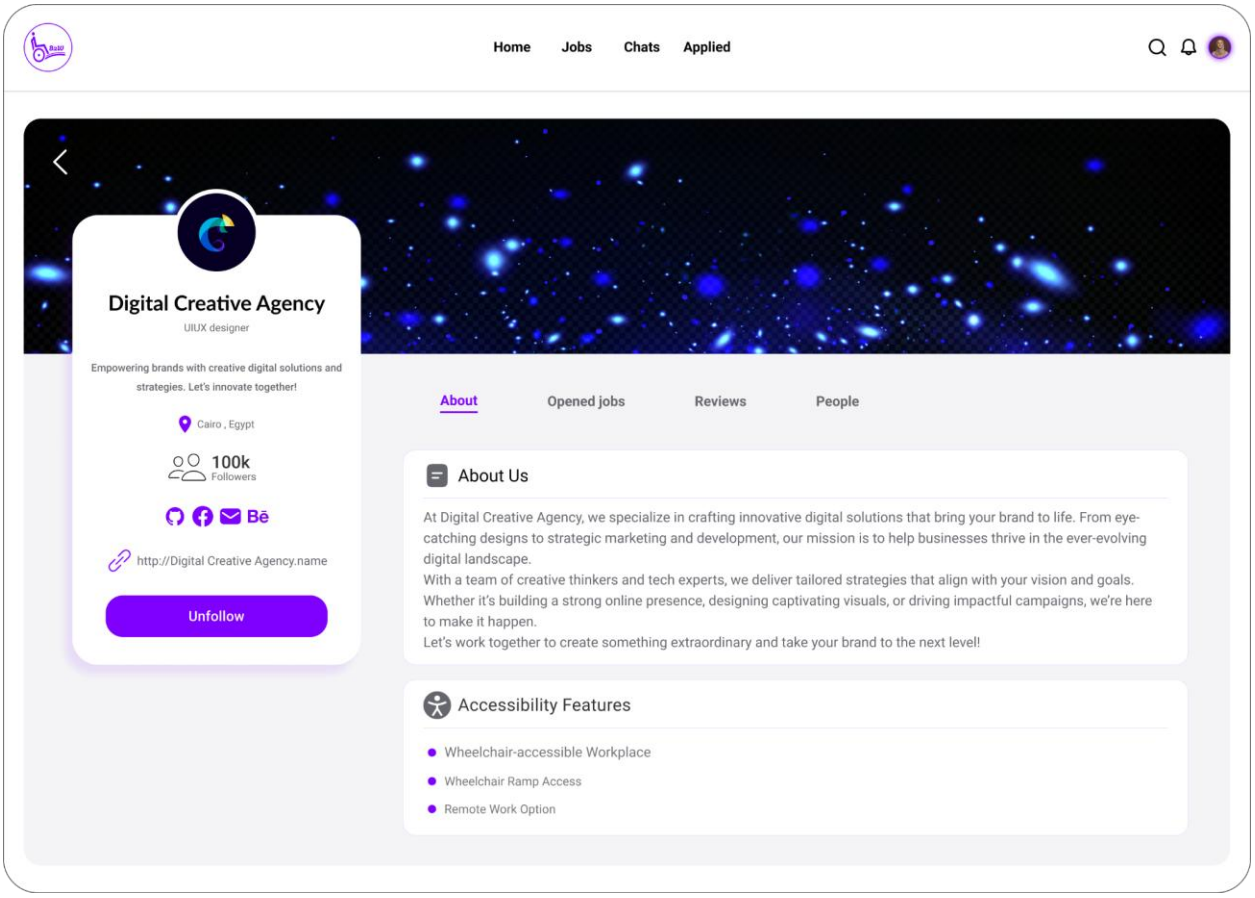


Figure 4.31 Company Account View

4.4.4 Website – Company

4.4.4.1 Home Page

The dashboard centralizes recruitment management, displaying applications with filtering options and status tracking. It includes notifications, search, profile editing, and accessibility settings, as shown in Figure 4.32.

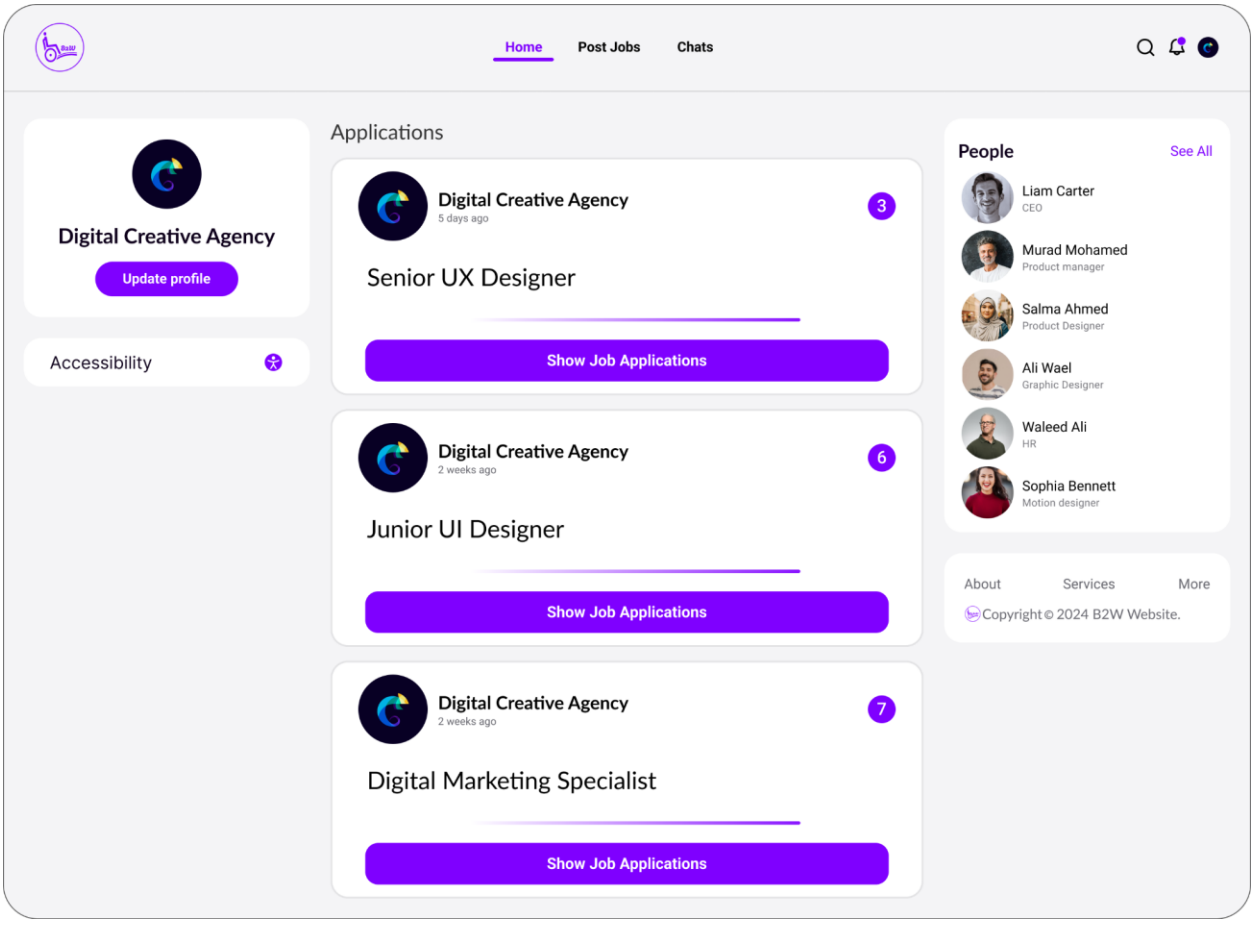


Figure 4.32 Home Page

4.4.4.2 Upload Job

Companies can create and edit job listings including title, description, and requirements through a dedicated form, with options for later updates, as shown in Figure 4.33.

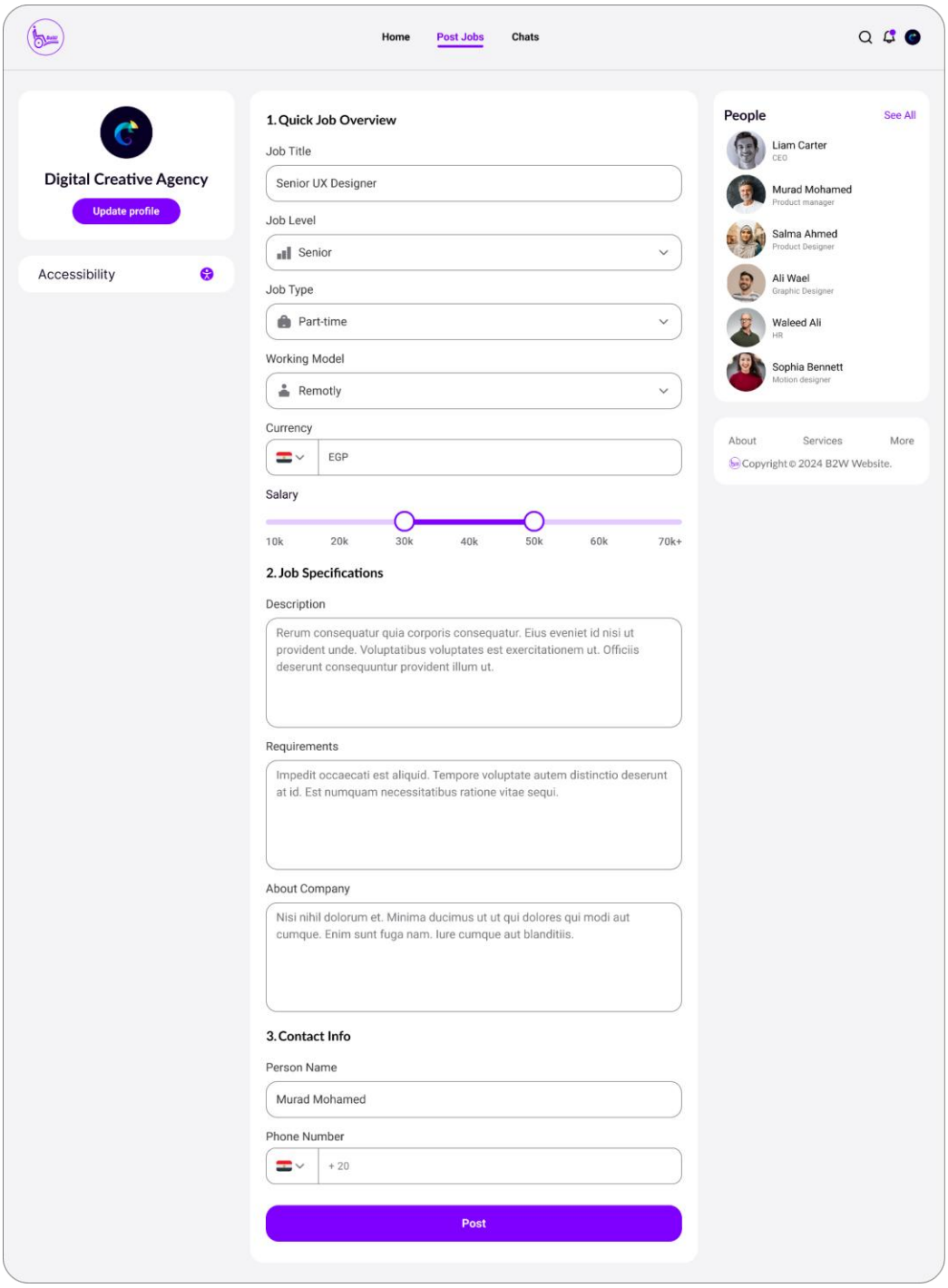


Figure 4.33 Upload Job

4.4.4.3 Chat Screen

Companies can view applicant profiles and send acceptance/rejection messages via chat, as shown in Figure 4.34.

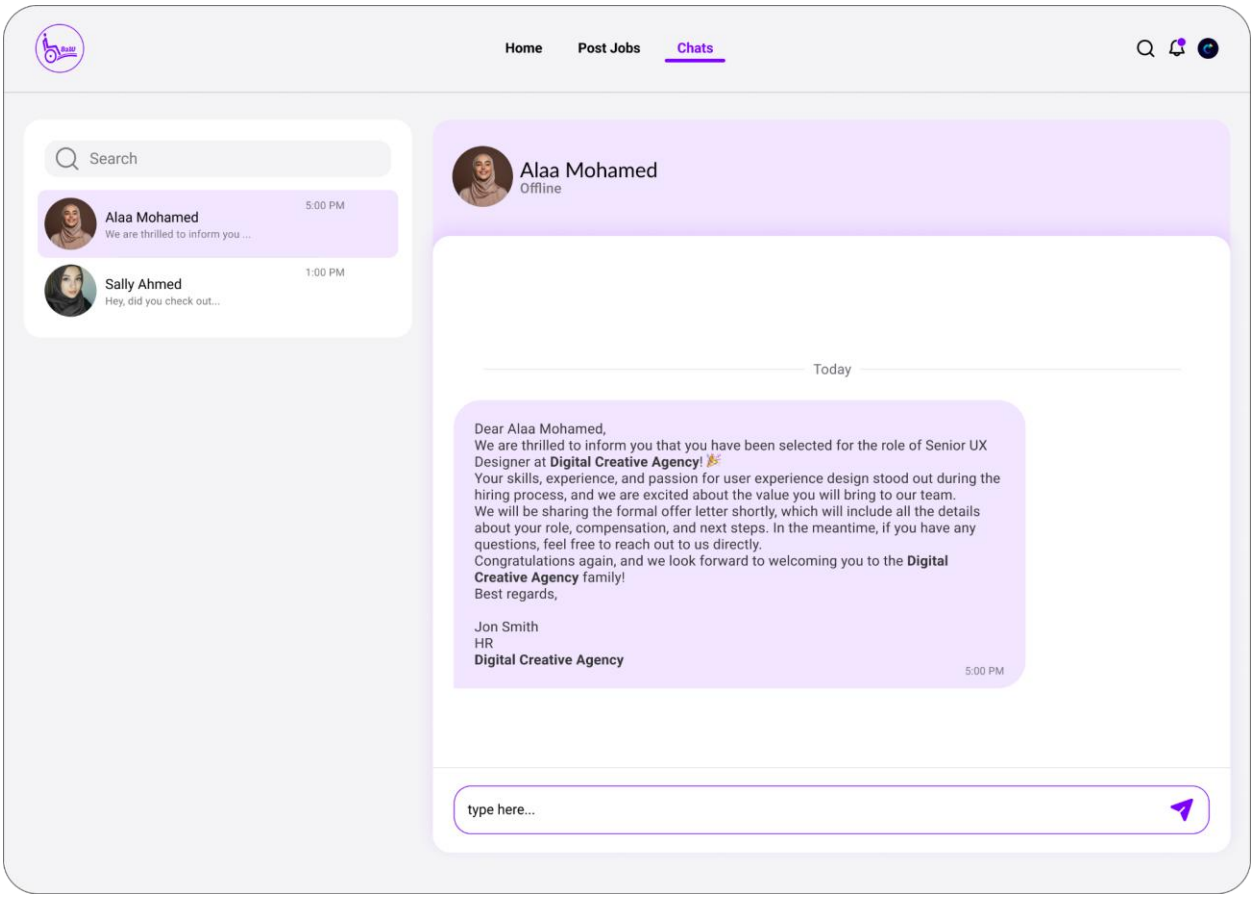


Figure 4.34 Chat Screen

4.4.4.4 Company Account screen

The screen displays company details and accessibility features. Users can follow/unfollow, view jobs, browse employees, and read reviews, as shown in Figure 4.35.

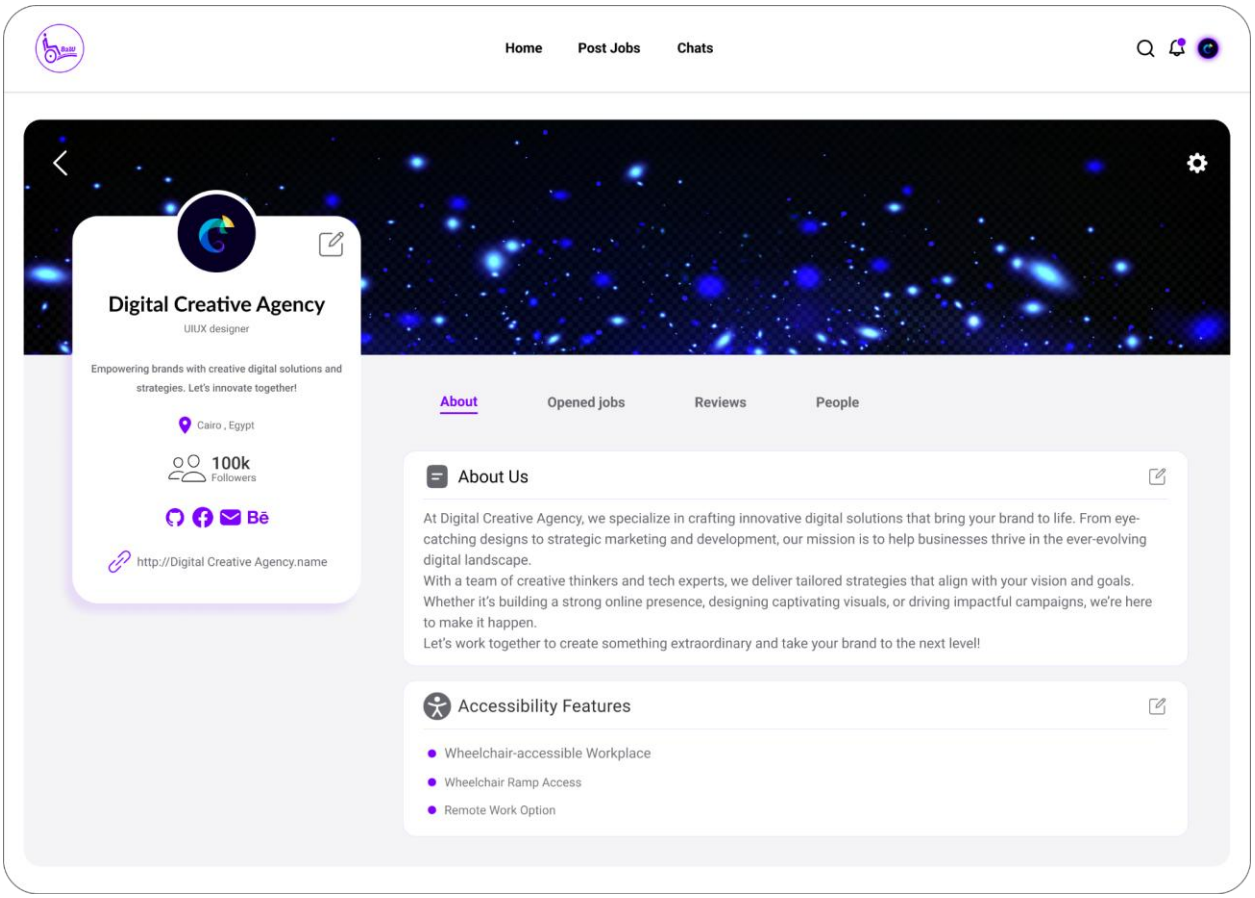


Figure 4.35 Company Account screen

4.5 Summary

This chapter outlined the platform's interface design, emphasizing accessibility and usability for both job seekers and companies. Key features include streamlined onboarding, intuitive job search, profile management, and messaging all designed with adjustable settings to accommodate diverse needs. The interface also integrates AI features to enhance user experience. In the next chapter, we will map our system design to implementation.

Chapter 5:
System Implementation

5.1 Introduction

In this chapter, we will start to map our design into implementation and turn the idea into a tangible product. From discussing the implementation process considering our objectives, to discuss the testing process and result respectively, to stand on the extent of its conformity with the goals that we set in advance and stand on the aspects of failure and success.

5.2 Sample Application Codes

Frontend Implementation 5.2.1

1-Sign-Up Component

The Createaccount component handles the **user registration process** in the application. It collects the user's personal information—**first name, last name, email, and password**—and sends it to the backend API via the registerUser function.

Upon successful registration:

- The returned user data and authentication token are stored in **localStorage**.
- The user is redirected to the **user type selection page**.

```jsx
import Banner from "../../components/signup/Banner"
import bannerImage from "../../assets/images/signup/signup.png"
import { Eye, EyeOff } from "lucide-react";
import Button from "../../components/global/Button";
import { FcGoogle } from "react-icons/fc";
import { FaApple, FaFacebook } from "react-icons/fa";

import { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import { useAuth } from "../../context/AuthContext";
import { registerUser } from "../../Api_Calls/Authservices";
function Createaccount() {
    const navigate = useNavigate()
    const [form, setForm] = useState({ firstName: "", lastName: "", email: "", userName: "", password: "" });
    const [error, setError] = useState(null)
    const { showPassword, setShowPassword, setAuthUser } = useAuth();
    const handleSubmit = async (e) => {
        e.preventDefault();
        console.log(form)
        const FORM = { ...form, userName: form.email }
        try {
            const data = await registerUser(FORM);
            const userData = {
                token: data.token,
                email: data.email,
                firstName: data.firstName,
                lastName: data.lastName,
                userId: data.userId,
                roles: data.roles,
                isAuthenticated: data.isAuthenticated,
            }
            setAuthUser(userData);
            localStorage.setItem("authUser", JSON.stringify(userData));
            navigate('/selectUser')
        } catch (err) {
            console.log(err?.response?.data?.errors ||err?.response?.data ||  "Login failed");
            setError(err.response.data.errors ||err.response.data ||  "Login failed")
        }

    }
    const handleChange = (e) => {
        setForm({ ...form, [e.target.name]: e.target.value })
    }
```

Figure 5.1.1 sign up component

```jsx
    return (
        <div className="grid grid-cols-2 h-screen bg-primry_purble">
            <Banner image={bannerImage} text="Where Hope starts" bgColor="primry_purble" logoColor="white" />
            <div className="bg-white p-8 rounded-lg shadow-md w-full flex flex-col justify-center px-20 rounded-bl-[6%] rounded-tl-[6%]">
                <h2 className="text-3xl font-bold text-center text-gray-900 mb-6 font-lato">
                    Create Your Account
                </h2>
                <form onSubmit={handleSubmit}>
                    {/* First & Last Name */}
                    <div className="flex gap-4">
                        <div className="flex flex-col w-1/2">
                            <label className="text-gray-600 font-roboto">First Name</label>
                            <input type="text" placeholder="First Name" name="firstName" value={form.firstName} onChange={handleChange}
                                className="mt-1 p-2 border placeholder:text-zinc-400 border-gray-300 rounded-2xl w-full focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-indigo-500"
                            />
                            <div className="text-xs mt-1 text-red-600">{error?.FirstName}</div>
                        </div>
                        <div className="flex flex-col w-1/2">
                            <label className="text-gray-600 font-roboto">Last Name</label>
                            <input type="text" placeholder="Last Name" name="lastName" value={form.lastName} onChange={handleChange}
                                className="mt-1 p-2 border border-gray-300 placeholder:text-zinc-400 rounded-2xl w-full focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-indigo-500"
                            />
                            <div className="text-xs mt-1 text-red-600">{error?.LastName}</div>
                        </div>
                    </div>

                    {/* Email */}
                    <div className="mt-4">
                        <label className="text-gray-600 font-roboto">Email Address</label>
                        <input type="email" placeholder="Example@gmail.com" name="email" value={form.email} onChange={handleChange}
                            className="mt-1 p-2 border placeholder:text-zinc-400 border-gray-300 rounded-2xl w-full focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-indigo-500"
                        />
                    </div>
                    <div className="text-xs mt-1 text-red-600">{error?.Email}</div>
                    {/* Password */}
                    <div className="mt-4 relative">
                        <label className="text-gray-600 font-roboto">Password</label>
                        <input type={showPassword ? "text" : "password"} placeholder="* * * * * * * * * *" name="password" value={form.password} onChange={handleChange}
                            className="mt-1 p-2 border border-gray-300 placeholder:text-zinc-400 rounded-2xl w-full focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-indigo-500"
                        />
                        <div className="text-xs mt-1 text-red-600">{error?.Password||error}</div>
                        <button type="button" onClick={() => setShowPassword(!showPassword)}
                            className="absolute right-3 top-[40px] font-roboto text-gray-500"
                        >
                            {showPassword ? <EyeOff size={18} /> : <Eye size={18} />}
                        </button>
                    </div>

                    {/* Submit Button */}
                    <Button btn_text="Sign Up" marg={7} />
                    <p className="font-roboto text-dark_gray font-medium">Already have an account? <span className="text-primry_purble font-bold">
                        <Link to="/Login">Login</Link>
                    </span>
                    </p>
                    <div className="flex items-center my-4 font-roboto">
                        <div className="flex-1 border-gray-300 border"></div>
                        <p className="mx-3">OR sign up with</p>
                        <div className="flex-1 border-gray-300 border-t"></div>
                    </div>
                    {/* social media icons */}
                    <div className="flex justify-center items-center gap-1">
                        <button className="p-2 rounded-full"><FcGoogle size={30} /></button>
                        <button className="p-2 rounded-full"><FaApple size={30} /></button>
                        <button className="p-2 rounded-full text-blue-600"><FaFacebook size={30} /></button>
                    </div>
                </form>
            </div>
        </div>
    )
}
export default Createaccount
```

Figure 5.1.2 sign up component

2-Login Component

The Login component is responsible for **authenticating existing users** in the application. It collects the user's **email and password**, sends them to the backend API through the loginUser function, and handles navigation based on the user's role after a successful login.

Upon successful authentication:

- The component stores the received **token** and **user details** in localStorage.
- It fetches additional user or company setup data using getUserData and getCompanyData.
- The user is redirected to either the User or Company dashboard depending on their role.

```jsx
import Banner from "../../components/signup/Banner"
import bannerImage from '../../assets/images/signup/login.png'
import { Eye, EyeOff } from "lucide-react";
import Button from "../../components/global/Button";
import { Link } from "react-router-dom";
import { FcGoogle } from "react-icons/fc";
import { FaApple, FaFacebook } from "react-icons/fa";
import { useNavigate } from "react-router-dom";
import { useState } from "react";
import { loginUser } from "../../Api_Calls/Authservices";
import { useAuth } from "../../context/AuthContext";
import { useContext } from "react";
import { SetupContext } from "../../context/SetupContext";
import { getCompanyData } from "../../Api_Calls/SetupServices";
import { getUserData } from "../../Api_Calls/SetupServices";

function Login() {
    const navigate = useNavigate()
    const [form, setForm] = useState({ email: "", password: "" });
    const [error, setError] = useState(null)
    const [emailError, setEmailError] = useState(null)
    const [PassError, setPassError] = useState(null)
    const { showPassword, setShowPassword, setAuthUser, authUser } = useAuth();
    const { setUserData, setComData } = useContext(SetupContext);
    const handleSubmit = async (e) => {
        e.preventDefault();
        if (form.email && form.password) {
            setEmailError("")
            setPassError("")
            try {
                const data = await loginUser(form);
                const userData = {
                    token: data.token,
                    email: data.email,
                    firstName: data.FirstName,
                    lastName: data.LastName,
                    userId: data.userId,
                    roles: data.roles,
                    isAuthenticated: data.isAuthenticated,
                }
                setAuthUser(userData);
                localStorage.setItem("authUser", JSON.stringify(userData));
                await getUserData(authUser.userId, setUserData);
                await getCompanyData(authUser.userId, setComData);
                authUser?.roles[0] === "User" ? navigate('/home') : navigate('/home-Company')
            } catch (err) {
                console.log(err.response.data || "Login failed");
                setError(err.response.data)
            }
        }
        else if (form.email === "" && form.password === "") {
            setEmailError('Email is required')
            setPassError('Password is required')
        }
        else if (form.password === "") {
            setPassError('Password is required')
            setEmailError("")
        }
        else {
            setEmailError('Email is required')
            setPassError("")
        }
    };
```

Figure 5.2.1 sign in component

```jsx
return (
        <div className="grid grid-cols-2 h-screen bg-primry_purble">
            <Banner image={bannerImage} text='Where Hope starts' bgColor="primry_purble" logoColor="white" />
            <div className="bg-white p-8 rounded-lg shadow-md w-full flex flex-col justify-center px-20 rounded-bl-[6%] rounded-tl-[6%]">
                <h2 className="text-4xl font-bold text-center text-gray-900 mb-6 font-lato">
                    Login
                </h2>
                <form action="" onSubmit={handleSubmit}>
                    {/* Email */}
                    <div className="mt-4">
                        <label className="text-gray-600 font-roboto">Email Address</label>
                        <input type="email" name="email" value={form.email} onChange={(e) => setForm({ ...form, email: e.target.value })} placeholder="Example@gmail.com"
                            className="mt-1 p-2 border border-gray-300 placeholder:text-zinc-400 rounded-2xl w-full focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-indigo-500"
                        />
                    </div>
                    {emailError && <div className="text-sm text-red-500 my-1">{emailError}</div>}
                    {/* Password */}
                    <div className="mt-4 relative">
                        <label className="text-gray-600 font-roboto">Password</label>
                        <input type={showPassword ? "text" : "password"} name="password" value={form.password} onChange={(e) => setForm({ ...form, password: e.target.value })} placeholder="* * * * * * * * *"
                            className="mt-1 p-2 border placeholder:text-zinc-400 border-gray-300 rounded-2xl w-full focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-indigo-500"
                        />
                        <button type="button" className="absolute right-3 top-[40px] font-roboto text-gray-500"
                            onClick={() => setShowPassword(!showPassword)}
                        >
                            {showPassword ? <EyeOff size={18} /> : <Eye size={18} />}
                        </button>
                    </div>
                    {PassError && <div className="text-sm text-red-500 my-1">{PassError}</div>}

                    <Link to="/forgotpassword">
                        <p className="text-md text-[#66666E] text-right font-roboto">Forgot password?</p>
                    </Link>
                    {error && <div className="text-sm text-red-500 mb-1">{error}</div>}
                    {/* Submit Button */}
                    <Button btn_text="Login" marg={7} />

                    <p className="text-dark_gray font-medium font-roboto">Don't have an account?
                        <span className="text-primry_purble font-bold">
                            <Link to="/signup"> Sign up</Link>
                        </span>
                    </p>
                    <div className="flex items-center my-4 font-roboto">
                        <div className="flex-1 border-gray-300 border"></div>
                        <p className="mx-3">OR sign up with</p>
                        <div className="flex-1 border-gray-300 border-t"></div>
                    </div>
                    {/* social media icons */}
                    <div className="flex justify-center items-center gap-1">
                        <button className="p-2 rounded-full"><FcGoogle size={30} /></button>
                        <button className="p-2 rounded-full"><FaApple size={30} /></button>
                        <button className="p-2 rounded-full text-blue-600"><FaFacebook size={30} /></button>
                    </div>
                </form>
            </div>
        </div>
    )
}

export default Login
```

Figure 5.2.2 sign in component

3-Authentication Context

The AuthContext provides a **centralized authentication system** for the entire application using React's Context API. It manages the authentication state, user details, and utilities like password visibility and user role selection.

```jsx
import { useState, useContext, createContext, useEffect } from "react";
import { useNavigate } from "react-router-dom";
const AuthContext = createContext()
export const AuthProvider = ({ children }) => {
    const [showPassword, setShowPassword] = useState(true);
    const [selectUser, setSelectUser] = useState(false)
    const navigate = useNavigate()
    const [authUser, setAuthUser] = useState(() => {
        const storedAuth = localStorage.getItem('authUser');
        return storedAuth
            ? JSON.parse(storedAuth)
            : {
                token: "", email: "", firstName: "", lastName: "", userId: "", roles: [], isAuthenticated: false,
            }
    })
    useEffect(() => {
        const authData = JSON.parse(localStorage.getItem('authUser'));
        if (authData) {
            setAuthUser(authData);
        }
    }, []);
    const logout = () => {
        localStorage.removeItem('authUser')
        setAuthUser({
            token: "", email: "", firstName: "", lastName: "", userId: "", roles: [], isAuthenticated: false,
        });
        navigate('/landing')
    }
    return (
        <AuthContext.Provider value={{ authUser, setAuthUser, logout, showPassword, setShowPassword, selectUser, setSelectUser }}>
            {children}
        </AuthContext.Provider>
    )
}
export const useAuth = () => {
    const context = useContext(AuthContext)
    if (context === undefined) {
        throw new Error("useAuth must be used within an AuthProvider")
    }
    return context
}
```

Figure 5.3 Authentication Context

4- Authentication API Layer

The authservice.js file is a centralized service module for **handling all authentication-related API requests** in the project using **Axios**.

It defines and exports reusable asynchronous functions to communicate with the backend **.NET API** for operations like login, registration, role management, and password recovery.

```
import axios from "axios";

const BASE_URL = "https://localhost:7287/api/Authentication";

export const api = axios.create({
    baseURL: BASE_URL,
    withCredentials: true, // Include cookies in requests
    headers: {
        'Content-Type': 'application/json',
    }
});

export const registerUser = async (formData) => {
    const res = await api.post(`/register`, formData);
    return res.data;
};

export const loginUser = async (formData) => {
    const res = await api.post(`/Login`, formData);
    console.log(res.data)
    return res.data;
};
export const AddRole = async (role) => {
    const res = await api.post(`/AddRole`, role);
    return res.data
}

export const sendResetToken = async (email) => {
    const res = await api.post(`/SendToken_ToMail`, { email });
    return res.data;
};

export const resetPassword = async (data) => {
    const res = await api.post(`/ResetPassword`, data);
    return res.data;
};
```

Figure 5.4 Authentication API Layer

5- Logout component

The Logout component allows the user to confirm logging out, and upon confirmation, it clears the authentication data from localStorage via the logout function in AuthContext.

```jsx
import { userprofileassets } from "../../assets/images/user Profile/userprofileAssets"
import { useAuth } from "../../context/AuthContext"
import Button from "../global/Button"
import { useNavigate } from "react-router-dom"
function Logout() {
    const { logout } = useAuth()
    const navigate = useNavigate()
    return (
        <div>
            <div className="w-1/2 m-auto mt-24">
                <img src={userprofileassets.logout} alt="" className="m-auto w-1/3" />
                <p className="font-lato font-medium text-center text-lg my-5">Do you want to log out now? We'll be here when you return!</p>
                <div className="flex gap-5 justify-center mt-5 w-3/4 m-auto">
                    <Button marg={5} btn_text={'Cancle'} bg={'bg-white text-primry_purble'} onHandleClick={() => navigate('/home')} />
                    <Button marg={5} btn_text={'Logout'} onHandleClick={logout} />

                </div>
            </div>
        </div>
    )
}

export default Logout
```

Figure 5.5 logout component

6- ProtectedRoutes component

The ProtectedRoutes component restricts access to private routes by checking if a valid token exists in authUser, redirecting unauthenticated users to the landing page.

```jsx
import { useAuth } from "../context/AuthContext"
import { Outlet, Navigate } from "react-router-dom"

function ProtectedRoutes() {
    const { authUser } = useAuth()
    if (!authUser.token) {
        return <Navigate to='/landing' />
    }
    return <Outlet />
}

export default ProtectedRoutes
```

Figure 5.6 ProtectedRoutes component

7-SelectUser Component

users choose their role as either "Employee" or "Company", it sends a role assignment request to the backend API, updates the authUser context and localStorage with the new role, and then navigates to either the company or employee setup page based on the selection.

Supporting multiple user roles, controlling access to different features, and ensuring that each user interacts with the application according to their responsibilities and permissions.

```jsx
import Button from "../../components/global/Button";
import Logo from "../../components/global/Logo";
import HeadingText from "../../components/signup/HeadingText";
import EmployeePhoto from "../../components/signup/EmployeePhoto";
import Company from "../../components/signup/Company";
import { useNavigate } from "react-router-dom";
import {useState } from "react";
import { useAuth } from "../../context/AuthContext";
import { AddRole } from "../../Api_Calls/Authservices";

function SelectUser() {
    const { selectUser, setSelectUser, authUser, setAuthUser } = useAuth();
    const [confirmed, setConfirmed] = useState(false);
    const navigate = useNavigate();

    // Handle "Next" Button Click
    const handleNext = async () => {
        if (selectUser) {
            setConfirmed(true);
            if (selectUser === "company")
                try {
                    const role = {
                        userId: `${authUser.userId}`, role: "Admin"
                    }
                    await AddRole(role)
                    const updatedUser = { ...authUser, roles: [role.role] };
                    setAuthUser(updatedUser);
                    localStorage.setItem("authUser", JSON.stringify(updatedUser));
                    navigate("/companysetup");
                }
                catch (err) {
                    console.log(err.response?.data)
                }
            else navigate("/setup");
        } else {
            alert("Please select an option before proceeding.");
        }
    };
    return (
        <div className="flex flex-col items-center">
            {/* Logo Position */}
            <div className="absolute top-[5%] left-[5%]">
                <Logo logoColor="#7F00FF" />
            </div>
            {/* Main Container */}
            <div className="flex flex-col justify-center items-center h-screen font-lato">
                <HeadingText mainText="Please select one of the following" subText="You are a/an " />
                <div className="flex justify-around w-full mt-6">
                    {/* Employee Card */}
                    <div className="flex items-center justify-center flex-col">
                        <div
                            onClick={() => !confirmed && setSelectUser("employee")}
                            className={`userCard p-4 border rounded-3xl shadow-md cursor-pointer hover:shadow-lg transition
                                ${selectUser === "employee" ? "bg-[#F2E5FF]" : "bg-[#F4F4F6]"}
                                ${confirmed && selectUser !== "employee" ? "opacity-50 pointer-events-none" : ""}`}
                        >
                            <EmployeePhoto isSelected={selectUser === "employee"} />
                        </div>
                        <h2 className="text-center mt-2 font-semibold">Employee</h2>
                    </div>
                    {/* Company Card */}
                    <div className="flex items-center justify-center flex-col">
                        <div
                            onClick={() => !confirmed && setSelectUser("company")}
                            className={`userCard p-4 border rounded-3xl shadow-md cursor-pointer hover:shadow-lg transition
                                ${selectUser === "company" ? " bg-[#F2E5FF]" : "bg-[#F4F4F6]"}
                                ${confirmed && selectUser !== "company" ? "opacity-50 pointer-events-none" : ""}`}
                        >
                            <Company isSelected={selectUser === "company"} />
                        </div>
                        <h2 className="text-center mt-2 font-semibold">Company</h2>
                    </div>
                </div>
                {/* "Next" Button */}
                <Button btn_text="Next" className="mt-6" onHandleClick={handleNext} marg={7} />
            </div>
        </div>
    );
}

export default SelectUser;
```

Figure 5.7 selectUser Component

8- Chatbot interview simulation component

The Chatbot component manages a multi-step conversational flow—starting with an introduction component , followed by level selection component, and ending with the chat interface between user and ai—wraps all logic within a global ChatProvider context for shared chatbot state management.

```jsx
import { useState } from 'react'
import Intro from '../../components/servicse/chatbot/Intro'
import Level_field from '../../components/servicse/chatbot/Level_field'
import Chat from '../../components/servicse/chatbot/Chat'
import ChatProvider from '../../context/ChatbotContext'

function Chatbot() {
    const [step, setstep] = useState(1)
    return (
        <ChatProvider>
            <div className=' col-span-9'>
            <div className='bg-white'>
                <div className='border-2 rounded-2xl '>
                    {step === 1 ?
                        <Intro setstep={setstep} />
                        : step === 2 ?
                            <Level_field setstep={setstep} />
                            : <Chat />
                    }
                </div>
            </div>
        </div>
        </ChatProvider>
    )
}

export default Chatbot
```

Figure 5.8.1 chatbot component

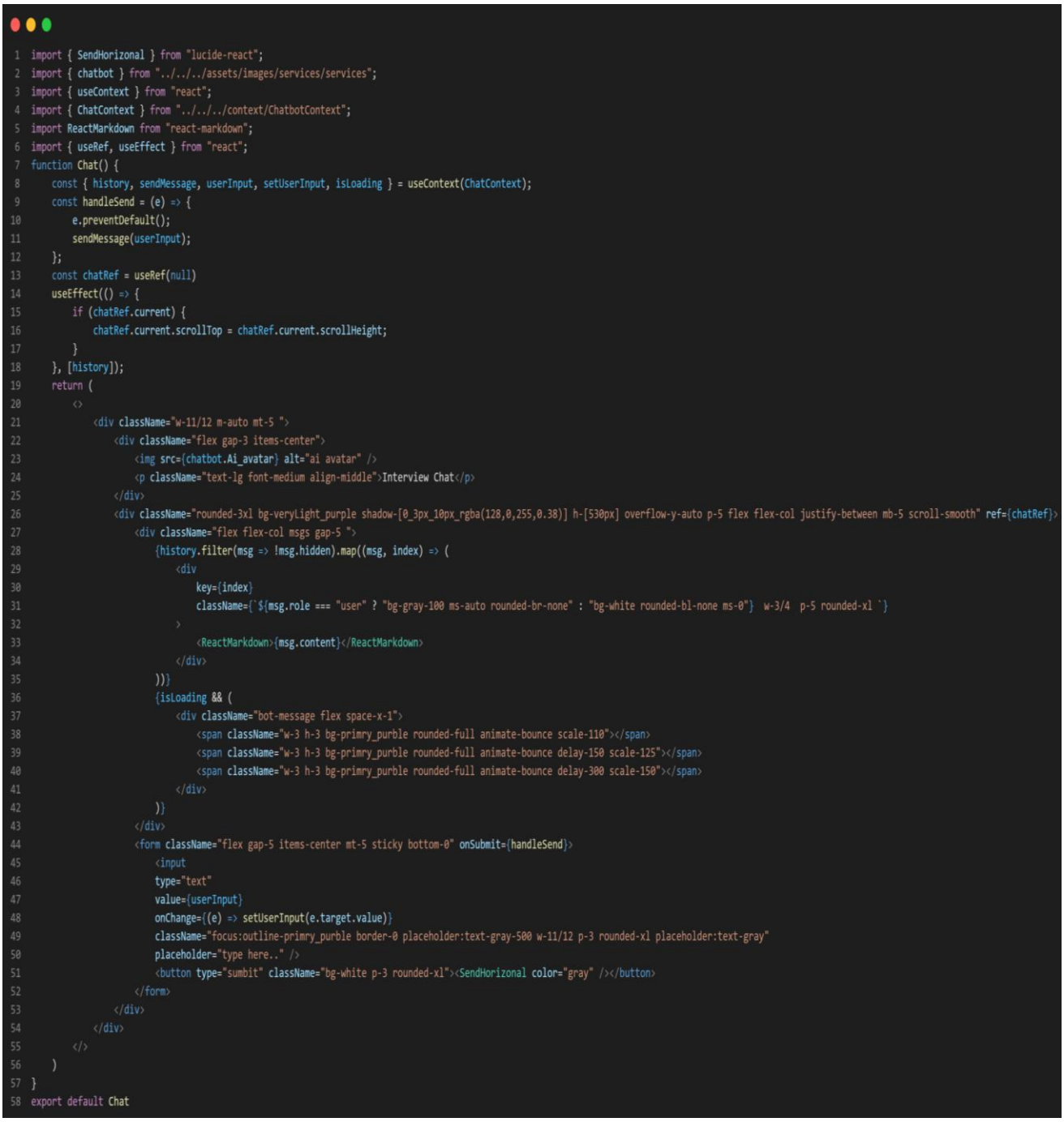
```jsx
import { chatbot } from '../../../assets/images/services/services'
import Button from '../../global/Button'
function Intro({setstep}) {

    const clickHandler=()=>{
        setstep(s=>s+1)
    }
    return (
        <div className='w-1/2 m-auto flex flex-col gap-2 text-center py-5'>
            <p className='font-lato text-2xl '>Interview Simulation</p>
            <p className='font-roboto text-2xl font-bold'>
                Welcome! Ready to ace your next interview?
            </p>
            <img src={chatbot.introAi} alt="intro-ai" width={430} className='self-center' />
                <Button btn_text='Get Started' onHandleClick={clickHandler} marg={0} />
        </div>
    )
}

export default Intro
```

figure 5.8.2 intro component

The Chat component displays a real-time chat interface where user and AI messages are rendered from shared context state (history), handles input submission via sendMessage, auto-scrolls to the latest message, and visually indicates AI response loading with animated typing dot.



```jsx
import { SendHorizonal } from "lucide-react";
import { chatbot } from "../../../assets/images/services/services";
import { useContext } from "react";
import { ChatContext } from "../../../context/ChatbotContext";
import ReactMarkdown from "react-markdown";
import { useRef, useEffect } from "react";
function Chat() {
    const { history, sendMessage, userInput, setUserInput, isLoading } = useContext(ChatContext);
    const handleSend = (e) => {
        e.preventDefault();
        sendMessage(userInput);
    };
    const chatRef = useRef(null)
    useEffect(() => {
        if (chatRef.current) {
            chatRef.current.scrollTop = chatRef.current.scrollHeight;
        }
    }, [history]);
    return (
        <>
            <div className="w-11/12 m-auto mt-5 ">
                <div className="flex gap-3 items-center">
                    <img src={chatbot.Ai_avatar} alt="ai avatar" />
                    <p className="text-lg font-medium align-middle">Interview Chat</p>
                </div>
                <div className="rounded-3xl bg-veryLight_purple shadow-[0_3px_10px_rgba(128,0,255,0.38)] h-[530px] overflow-y-auto p-5 flex flex-col justify-between mb-5 scroll-smooth" ref={chatRef}>
                    <div className="flex flex-col msgs gap-5 ">
                        {history.filter(msg => !msg.hidden).map((msg, index) => (
                            <div
                                key={index}
                                className={`${msg.role === "user" ? "bg-gray-100 ms-auto rounded-br-none" : "bg-white rounded-bl-none ms-0"}  w-3/4  p-5 rounded-xl `}
                            >
                                <ReactMarkdown>{msg.content}</ReactMarkdown>
                            </div>
                        ))}
                        {isLoading && (
                            <div className="bot-message flex space-x-1">
                                <span className="w-3 h-3 bg-primry_purble rounded-full animate-bounce scale-110"></span>
                                <span className="w-3 h-3 bg-primry_purble rounded-full animate-bounce delay-150 scale-125"></span>
                                <span className="w-3 h-3 bg-primry_purble rounded-full animate-bounce delay-300 scale-150"></span>
                            </div>
                        )}
                    </div>
                    <form className="flex gap-5 items-center mt-5 sticky bottom-0" onSubmit={handleSend}>
                        <input
                        type="text"
                        value={userInput}
                        onChange={(e) => setUserInput(e.target.value)}
                        className="focus:outline-primry_purble border-0 placeholder:text-gray-500 w-11/12 p-3 rounded-xl placeholder:text-gray"
                        placeholder="type here.." />
                        <button type="sumbit" className="bg-white p-3 rounded-xl"><SendHorizonal color="gray" /></button>
                    </form>
                </div>
            </div>
        </>
    )
}
export default Chat
```

Figure 5.8.3 chat component

9-Chatbot Context

The Chatbot Context manages all chatbot-related state and logic, including the user's selected interview **level** and **field**, chat history, userInput, and loading state, and provides core functions like startInterview, which dynamically generates a **customized AI prompt** based on the selected role and level to simulate a realistic interview experience, and sendMessage, which sends user replies and updates the conversation with AI-generated follow-ups using the Gemini model, and finally enabling the AI to review all stored answers and provide **final feedback with a score and suggestions** at the end of the session.

```jsx
import { createContext, useState } from "react";
import run from "../config/gemini";
export const ChatContext = createContext();
const ChatProvider = ({ children }) => {
    const [history, setHistory] = useState([]);
    const [level, setLevel] = useState("");
    const [field, setField] = useState("");
    const [interviewStarted, setInterviewStarted] = useState(false);
    const [userInput, setUserInput] = useState("");
    const [isLoading, setIsLoading] = useState(false);

    const startInterview = async () => {
        if (!level || !field) return;
        setInterviewStarted(true);
        const prompt = `You are an expert interviewer conducting a mock interview for a candidate applying for a ${field}
        position at the ${level} level mock interview with me give me 10 questions Ensure questions cover technical skills
        , problem-solving, and behavioral aspects, dont correct my answers or do anything just give me the next question and
        at the end evaluate my answeres give me overall score and enhancements for each answer if needed ,start the chat with
        'I'm B2W interviewer ,lets start our interview!' and give me first qustion in new line`;

        const userMessage = { role: "user", content: prompt ,hidden:true};
        setHistory([userMessage]);
        setIsLoading(true);
        try {
            const firstQuestion = await run(prompt, []);
            setHistory(prev => [...prev, { role: "model", content: firstQuestion }]);
        } catch (error) {
            console.error("Error fetching first question:", error);
        } finally {
            setIsLoading(false);
        }
    };
    const sendMessage = async () => {
        if (!userInput.trim()) return;
        const userMessage = { role: "user", content: userInput };
        setHistory(prev => [...prev, userMessage]);
        setUserInput("");
        setIsLoading(true)
        const botResponse = await run(userInput, history);
        setIsLoading(false)
        setHistory(prev => [...prev, { role: "model", content: botResponse }]);
    };
    return (
        <ChatContext.Provider value={{
            history,sendMessage,startInterview, interviewStarted,level,
            field,setLevel,setField,userInput,setUserInput,isLoading
        }}>
            {children}
        </ChatContext.Provider>
    );
};
export default ChatProvider;
```

Chatbot context 5.9

Back-end Implementation 5.2.2

The backend implementation of the project is built using C# and ASP.NET Core Web API. It follows RESTful architecture to handle HTTP requests and provide structured responses in JSON format. The project is organized into logical layers including Controllers, Models, and Services to ensure clean separation of concerns and scalable development.

The codebase is well-commented to maintain clarity and ease of understanding, which also helps simplify the integration process with the mobile application frontend.

Figure 5.10 Controllers' Structure

System Controllers:

When a new request is sent to the server, it is routed to a specific controller action using ASP.NET Core's routing mechanism.

This section shows the structure of the system's controllers, which are responsible for handling REST API endpoints exposed to the mobile application.

Each controller typically represents either a feature or a resource in the mobile app. The controller handles the request, communicates with the appropriate model or service to retrieve or manipulate data, formats the response as a JSON object, and returns it to the client.

1-Authentication Controller

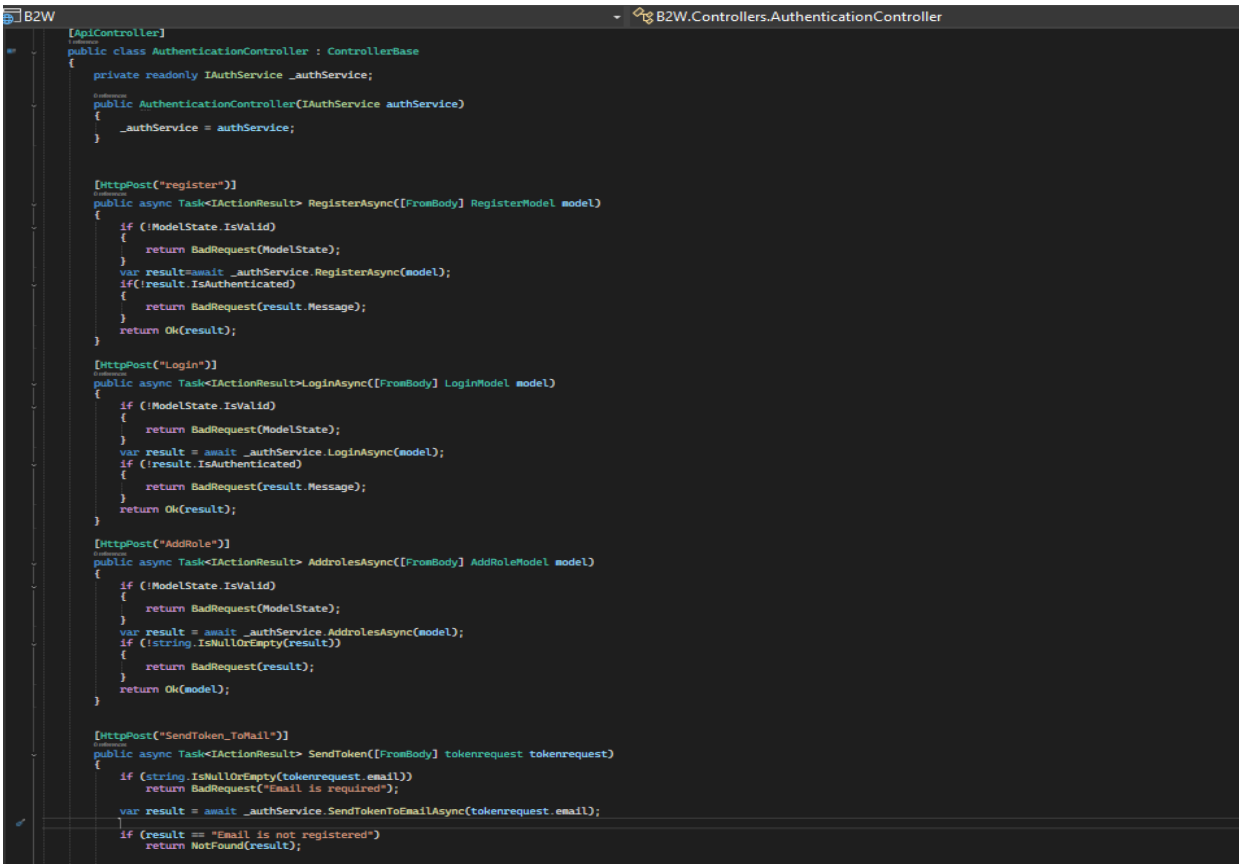
The Authentication Controller handles all user authentication and authorization operations in the system. It is responsible for registering users, logging them in, assigning roles, and handling password reset functionality.

When a user registers, they must choose whether they are a normal user or a company (admin). This role is stored and later used to determine permissions, such as whether the user can post jobs or not.

The controller includes the following main endpoints:

- Register: Registers a new user and assigns a role (user or company).
- Login: Authenticates a user and returns a token.
- Add Role: Allows assigning additional roles to existing users.
- SendToken_ToMail: Sends a reset token to the user's email.
- Reset Password: Allows the user to reset their password using the token.

This controller ensures secure login and access management across the application.




```
[ApiController]
public class AuthenticationController : ControllerBase
{
    private readonly IAuthService _authService;

    public AuthenticationController(IAuthService authService)
    {
        _authService = authService;
    }


    [HttpPost("register")]
    public async Task<IActionResult> RegisterAsync([FromBody] RegisterModel model)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }
        var result=await _authService.RegisterAsync(model);
        if(!result.IsAuthenticated)
        {
            return BadRequest(result.Message);
        }
        return Ok(result);
    }

    [HttpPost("Login")]
    public async Task<IActionResult>LoginAsync([FromBody] LoginModel model)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }
        var result = await _authService.LoginAsync(model);
        if (!result.IsAuthenticated)
        {
            return BadRequest(result.Message);
        }
        return Ok(result);
    }

    [HttpPost("AddRole")]
    public async Task<IActionResult> AddrolesAsync([FromBody] AddRoleModel model)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }
        var result = await _authService.AddrolesAsync(model);
        if (!string.IsNullOrEmpty(result))
        {
            return BadRequest(result);
        }
        return Ok(model);
    }


    [HttpPost("SendToken_ToMail")]
    public async Task<IActionResult> SendToken([FromBody] tokenrequest tokenrequest)
    {
        if (string.IsNullOrEmpty(tokenrequest.email))
            return BadRequest("Email is required");

        var result = await _authService.SendTokenToEmailAsync(tokenrequest.email);
        }
        if (result == "Email is not registered")
            return NotFound(result);
```

Figure 5.11 Authentication Controller

2-User Profile Controller:

The UserProfileController is responsible for managing user profile information within the system. It handles requests related to viewing and updating user data such as name, email, phone number, bio, and other personal details.

The controller ensures that only authorized users can access or modify their profile by applying authentication and authorization mechanisms. It communicates with the underlying services or models to retrieve the user's current data or to update it in the database.

The controller returns responses in a structured JSON format, making it easy for the mobile application to consume and reflect any changes instantly. Validation is applied on incoming requests to ensure data integrity and consistency.

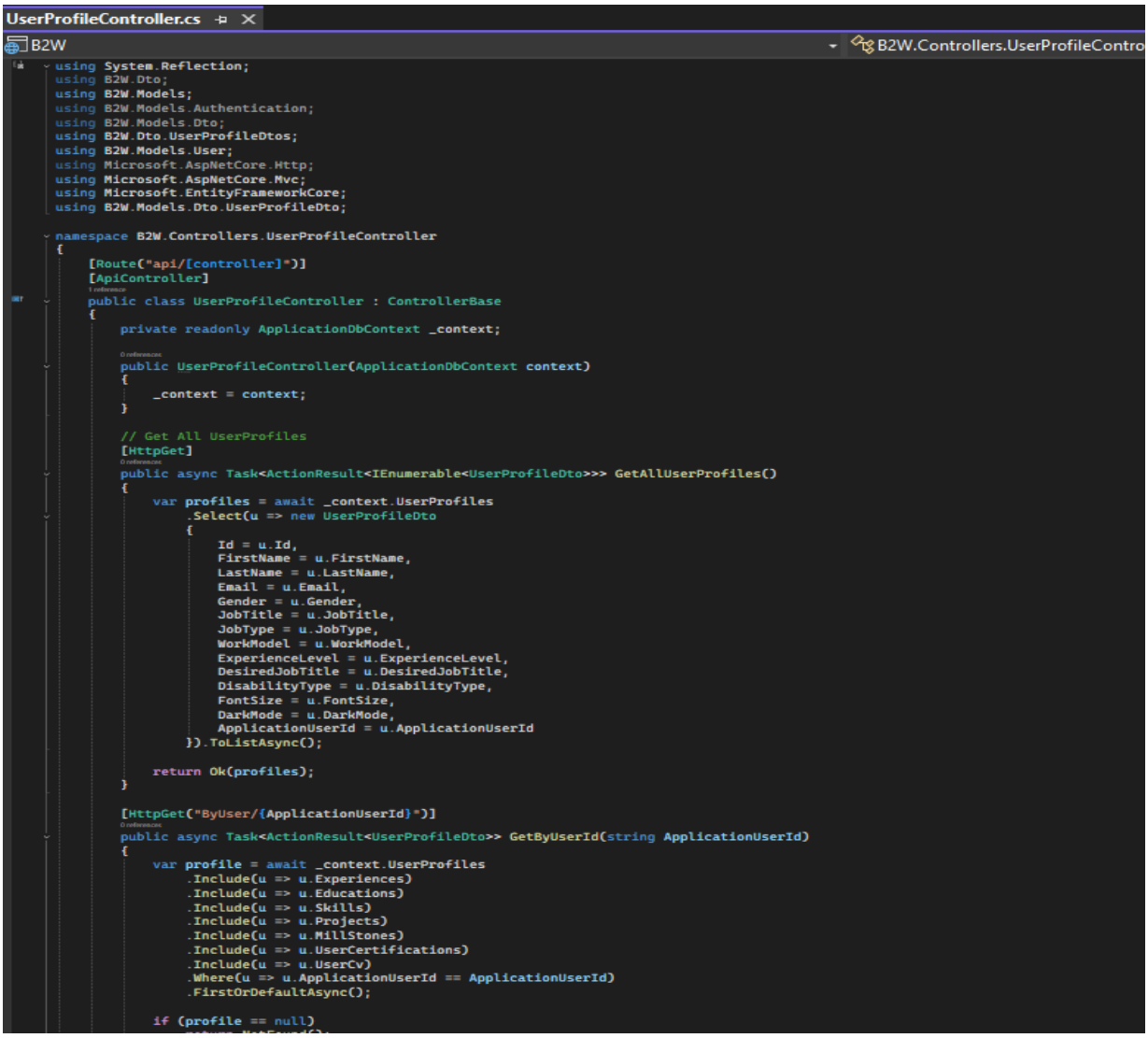


Figure 5.12 User Profile Controller

Create Method:

The Create method is responsible for receiving and processing incoming HTTP POST requests to create a new resource. It takes input data from the client (usually in JSON format), validates it to ensure all required fields are correctly provided, and then stores it in the database.

If the operation is successful, the method returns a success response along with the newly created object. Otherwise, it returns validation errors or an appropriate failure message. This method ensures data integrity and provides feedback to the frontend for a smooth user experience.

```
[HttpPost]
public async Task<IActionResult> Create([FromBody] UserProfileCreateDto dto)
{
    var profile = new UserProfile
    {

        FirstName = dto.FirstName,
        LastName = dto.LastName,
        Email = dto.Email,
        Gender = dto.Gender,
        JobTitle = dto.JobTitle,
        JobType = dto.JobType,
        WorkModel = dto.WorkModel,
        ExperienceLevel = dto.ExperienceLevel,
        DesiredJobTitle = dto.DesiredJobTitle,
        DisabilityType = dto.DisabilityType,
        FontSize = dto.FontSize,
        DarkMode = dto.DarkMode,
        ApplicationUserId = dto.ApplicationUserId
    };

    _context.UserProfiles.Add(profile);
    await _context.SaveChangesAsync();

    // ربط UserProfileId المستخدم في جدول
    var user = await _context.Users.FindAsync(dto.ApplicationUserId);
    if (user != null)
    {
        user.UserProfileId = profile.Id;
        _context.Users.Update(user);
        await _context.SaveChangesAsync();
    }
    var responseDto = new UserProfileCreateDto
    {
        id=profile.Id,
        FirstName = profile.FirstName,
        LastName = profile.LastName,
        Email = profile.Email,
        Gender = profile.Gender,
        JobTitle = profile.JobTitle,
        JobType = profile.JobType,
        WorkModel = profile.WorkModel,
        ExperienceLevel = profile.ExperienceLevel,
        DesiredJobTitle = profile.DesiredJobTitle,
        DisabilityType = profile.DisabilityType,
        FontSize = profile.FontSize,
        DarkMode = profile.DarkMode,
        ApplicationUserId = profile.ApplicationUserId
    };

    return CreatedAtAction(nameof(GetById), new { id = profile.Id }, responseDto);
}
```

Figure 5.13 Create method

Update (PATCH) Method:

The PATCH method is used to partially update user profile data. It receives a request containing only the fields that need to be changed, validates them, and then updates the relevant records in the database. This approach improves efficiency and avoids unnecessary overwriting of unchanged data.

Delete Method:

This method enables the removal of user-related data, either partially (such as a specific entry) or entirely. It guarantees secure deletion to prevent obsolete or incorrect data from remaining in the system. This functionality is important to maintain the relevance, privacy, and hygiene of the application's database.

```
    [ProducesResponseType(typeof(UserProfileCreateDto), StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status404NotFound)]

    [HttpPatch("{id}")]
    public async Task<IActionResult> Patch(int id, [FromBody] Dictionary<string, object> updates)
    {
        var profile = await _context.UserProfiles.FindAsync(id);
        if (profile == null) return NotFound();

        foreach (var update in updates)
        {
            var prop = typeof(UserProfile).GetProperty(update.Key, BindingFlags.Public | BindingFlags.Instance | BindingFlags.IgnoreCase);
            if (prop != null && prop.CanWrite)
            {
                prop.SetValue(profile, Convert.ChangeType(update.Value, prop.PropertyType));
            }
        }

        _context.Entry(profile).State = EntityState.Modified;
        await _context.SaveChangesAsync();

        // Manual mapping ←↓ UserProfileCreateDto
        var dto = new UserProfileCreateDto
        {
            id = profile.Id,
            FirstName = profile.FirstName,
            LastName = profile.LastName,
            Email = profile.Email,
            Gender = profile.Gender,
            JobTitle = profile.JobTitle,
            JobType = profile.JobType,
            WorkModel = profile.WorkModel,
            ExperienceLevel = profile.ExperienceLevel,
            DesiredJobTitle = profile.DesiredJobTitle,
            DisabilityType = profile.DisabilityType,
            FontSize = profile.FontSize,
            DarkMode = profile.DarkMode,
            ApplicationUserId = profile.ApplicationUserId
        };

        return Ok(dto);
    }

    // Delete
    [HttpDelete("{id}")]
    public async Task<IActionResult> Delete(int id)
    {
        var profile = await _context.UserProfiles.FindAsync(id);
        if (profile == null) return NotFound();

        _context.UserProfiles.Remove(profile);
        await _context.SaveChangesAsync();

        return NoContent();
    }
}
```

Figure 5.14 Update and delete method

3-Company Profile Controller:

The CompanyProfileController is responsible for managing the company profile data within the system. It is used by company-type users to create, view, and update their company information. This controller acts as a bridge between the frontend and the database through a set of methods that interact with the CompanyProfile model.

The main purpose of this controller is to allow companies to provide structured and accurate data, which can later be retrieved or used for recommendations and other interactions inside the application.

```
namespace B2W.Controllers.CompanyProfileController
{
    [Route("api/[controller]")]
    [ApiController]
    public class CompanyProfileController : ControllerBase
    {
        private readonly ApplicationDbContext _context;

        public CompanyProfileController(ApplicationDbContext context)
        {
            _context = context;
        }

        // Get All CompanyProfiles
        [HttpGet]
        public async Task<ActionResult<IEnumerable<CompanyProfilesDto>>> GetAllCompanyProfiles()
        {
            var companies = await _context.CompanyProfiles
                .Select(c => new CompanyProfilesDto
                {
                    CompanyProfileId = c.CompanyProfileId,
                    CompanyName = c.CompanyName,
                    Email = c.Email,
                    FieldOfWork = c.FieldOfWork,
                    WebsiteUrl = c.WebsiteUrl,
                    SocialMediaLinks = c.SocialMediaLinks,
                    Location = c.Location,
                    Description = c.Description,
                    CreatedAt = c.CreatedAt,
                    UpdatedAt = c.UpdatedAt,
                    ApplicationUserId = c.ApplicationUserId
                }).ToListAsync();

            return Ok(companies);
        }

        // Get By ApplicationUserId
        [HttpGet("ByUser/{ApplicationUserId}")]
        public async Task<ActionResult<CompanyProfilesDto>> GetByUserId(string ApplicationUserId)
        {
            var company = await _context.CompanyProfiles
                .Include(c => c.AccessibilityFeatures)
                .Include(c => c.Reviews)
                .Include(c => c.Employees)
                .FirstOrDefaultAsync(c => c.ApplicationUserId == ApplicationUserId);

            if (company == null) return NotFound();

            var dto = new CompanyProfilesDto
            {
                CompanyProfileId = company.CompanyProfileId,
                CompanyName = company.CompanyName,
                Email = company.Email,
                FieldOfWork = company.FieldOfWork,
                WebsiteUrl = company.WebsiteUrl,
                SocialMediaLinks = company.SocialMediaLinks,
                Location = company.Location,
```

Figure 5.15 Company Profile Controller

GetAllCompanyProfiles:

Retrieves a list of all registered company profiles in the system. Only selected fields are returned for each profile to ensure a lightweight response and fast processing. The method uses projection to convert each record into a simplified CompanyProfilesDto.

GetByUser:

Retrieves a specific company profile based on the ApplicationUserId. This method is typically used to fetch the profile of a currently logged-in company or to access company data by other parts of the system. It also loads related data such as reviews and employees for richer context.

4-Job Controller

The JobController is responsible for handling all job-related actions in the system, including posting new jobs, retrieving available jobs, and managing job details.

One of the key features of this controller is **role-based access control**, which ensures that only users with the **Company role (Admin)** are allowed to post or manage jobs. This access restriction is implemented to maintain the quality and reliability of job listings and prevent unauthorized users from creating invalid job data.

The controller provides several endpoints such as:

- **Create Job** – restricted to authenticated company users.
- **Get All Jobs** – accessible to all users to browse available job opportunities.
- **Get Job by ID** – fetches job details based on a specific ID.

This structure ensures that only verified companies can create job offers while allowing job seekers to view and apply to them freely.

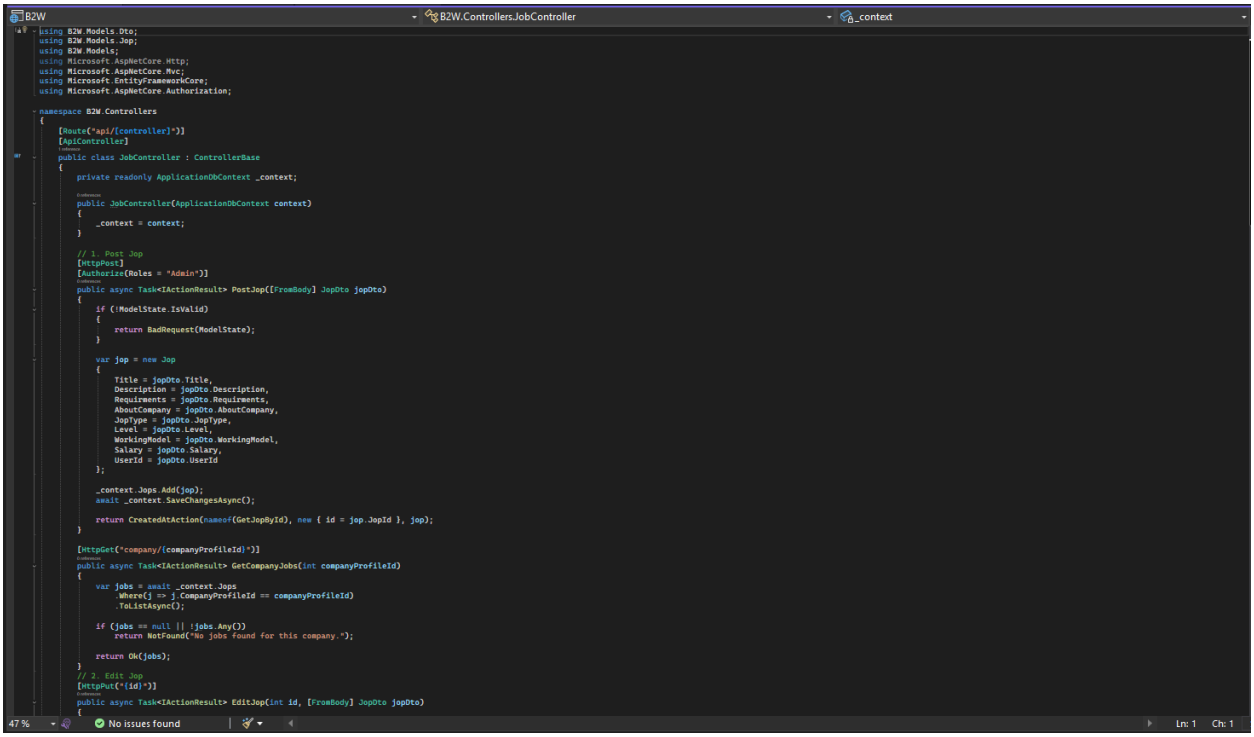


Figure 5.16 Job controller

System Models:

In this system, Models are core components representing the structure of the application's data and handling its business logic. Each model defines a set of properties that correspond to the database schema and are used to transfer and validate data between different layers of the API.

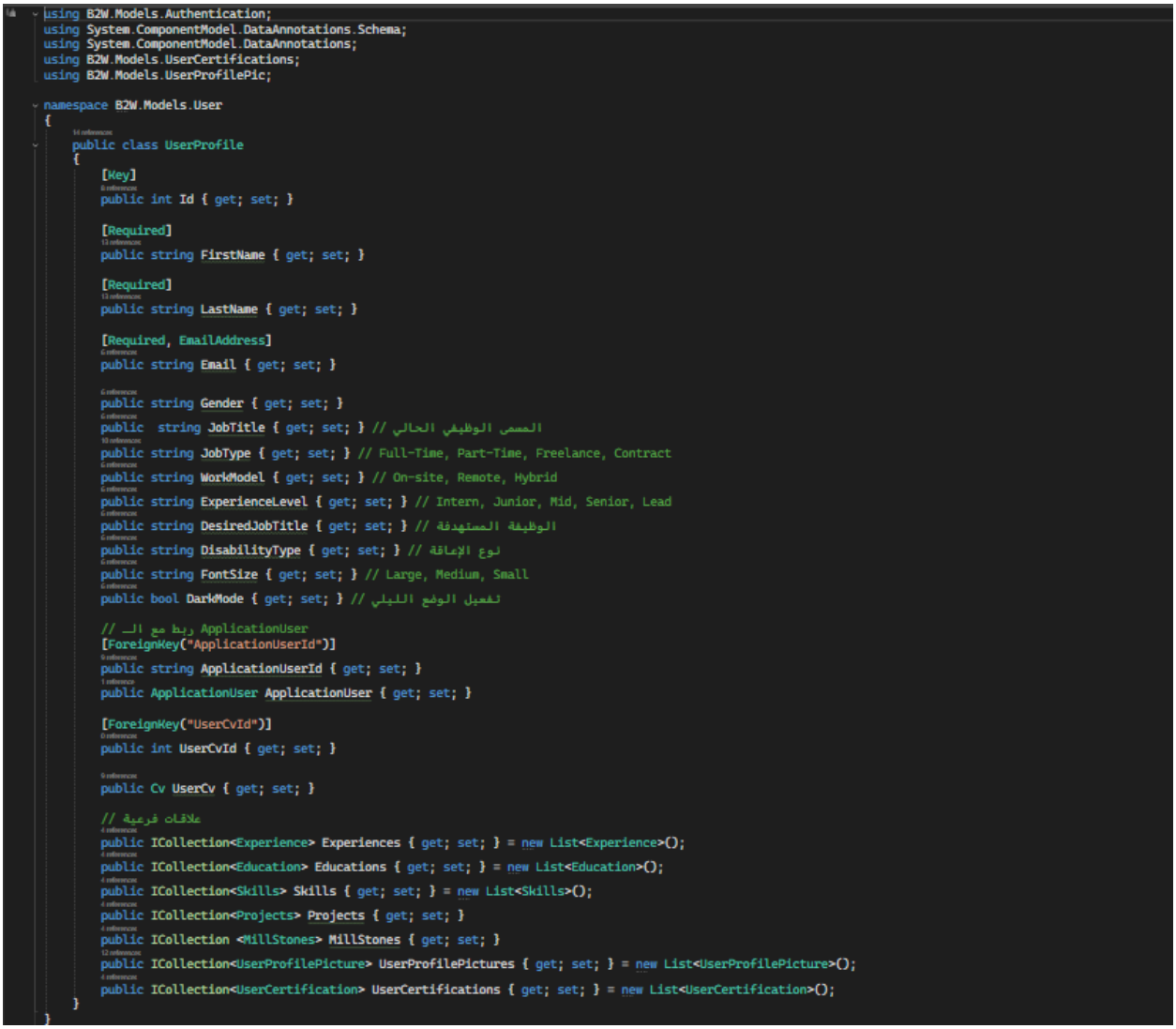
Models are designed to align with the database structure described in the “System Design Chapter” (ERD), and each model may include:

- Data annotations and validation rules to ensure data integrity during API operations.
- Public properties that represent the database fields.
- Navigation properties (if applicable) to define relationships between entities such as one-to-many or many-to-many.

The models serve as Data Transfer Objects (DTOs) or bind directly with the services and controllers to perform operations like creation, retrieval, and updates. These models are integral in maintaining consistency and structure across the API.

1-User Profile Model

This model stores extended user profile information such as personal data, job preferences, accessibility settings, and links to related records like CVs, skills, and certifications. It connects directly to the registered user account and supports building a full user portfolio inside the system.



```
using B2W.Models.Authentication;
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;
using B2W.Models.UserCertifications;
using B2W.Models.UserProfilePic;

namespace B2W.Models.User
{
    public class UserProfile
    {
        [Key]
        public int Id { get; set; }

        [Required]
        public string FirstName { get; set; }

        [Required]
        public string LastName { get; set; }

        [Required, EmailAddress]
        public string Email { get; set; }

        public string Gender { get; set; }
        public  string JobTitle { get; set; } // المسمى الوظيفي الحالي
        public string JobType { get; set; } // Full-Time, Part-Time, Freelance, Contract
        public string WorkModel { get; set; } // On-site, Remote, Hybrid
        public string ExperienceLevel { get; set; } // Intern, Junior, Mid, Senior, Lead
        public string DesiredJobTitle { get; set; } // الوظيفة المستهدفة
        public string DisabilityType { get; set; } // نوع الإعاقة
        public string FontSize { get; set; } // Large, Medium, Small
        public bool DarkMode { get; set; } // تفعيل الوضع الليلي

        // ربط مع الـ ApplicationUser
        [ForeignKey("ApplicationUserId")]
        public string ApplicationUserId { get; set; }
        public ApplicationUser ApplicationUser { get; set; }

        [ForeignKey("UserCvId")]
        public int UserCvId { get; set; }

        public Cv UserCv { get; set; }

        // علاقات فرعية
        public ICollection<Experience> Experiences { get; set; } = new List<Experience>();
        public ICollection<Education> Educations { get; set; } = new List<Education>();
        public ICollection<Skills> Skills { get; set; } = new List<Skills>();
        public ICollection<Projects> Projects { get; set; }
        public ICollection <MillStones> MillStones { get; set; }
        public ICollection<UserProfilePicture> UserProfilePictures { get; set; } = new List<UserProfilePicture>();
        public ICollection<UserCertification> UserCertifications { get; set; } = new List<UserCertification>();
    }
}
```

Figure 5.17 User Profile Model

2-Company Profile Model

This model holds the company's profile data, including name, contact info, website, description, and linked user account. It also defines relationships to posted jobs, company reviews, employees, and accessibility features. It's essential for managing and displaying company-related data in the system.

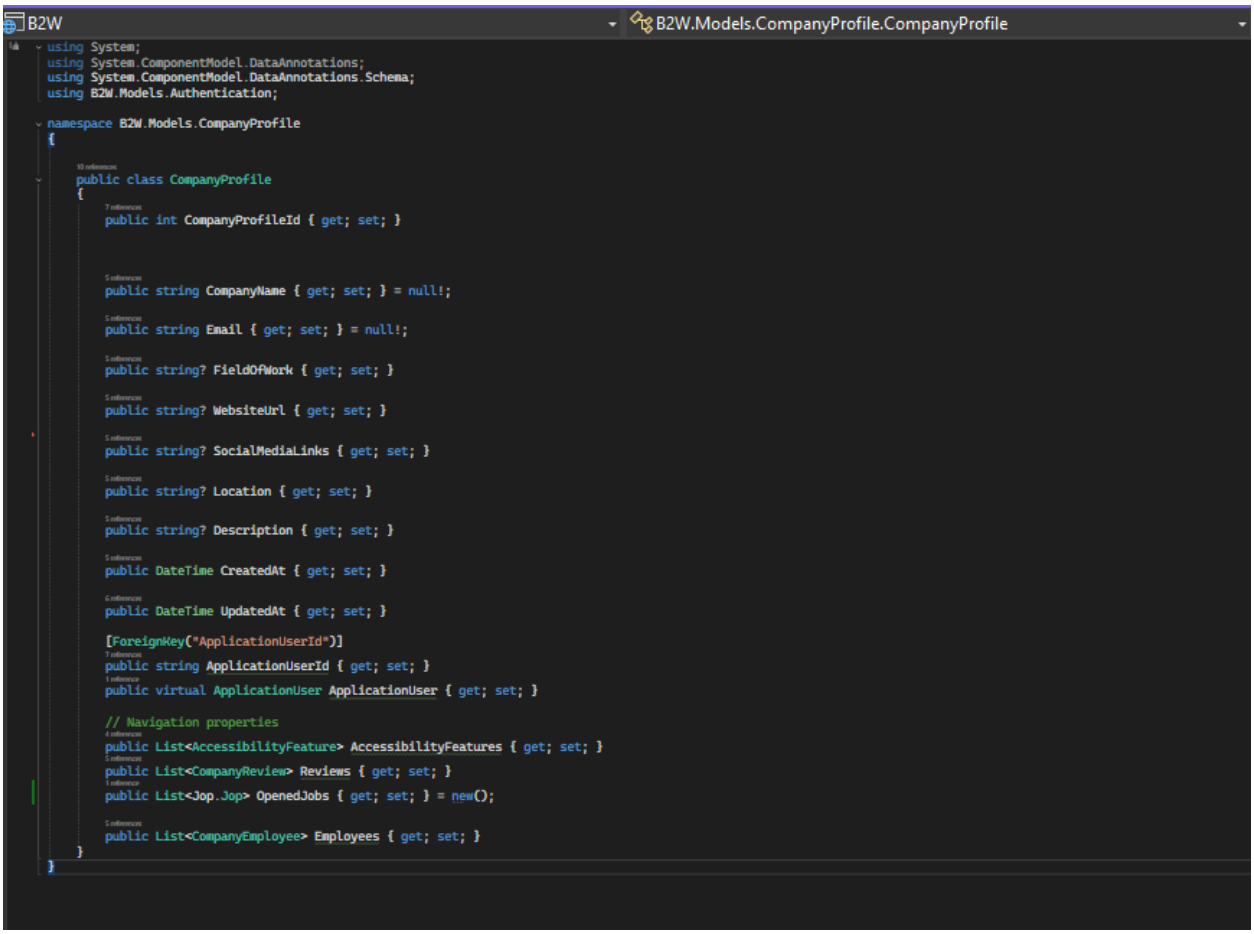


Figure 5.18 Company Profile Model

3-Job Model

This model represents a job listing. It contains details such as the title, description, requirements, job type, experience level, work model, and salary. It is linked to the company offering the job and the user who posted it. It also maintains a list of applicants who applied to the job.

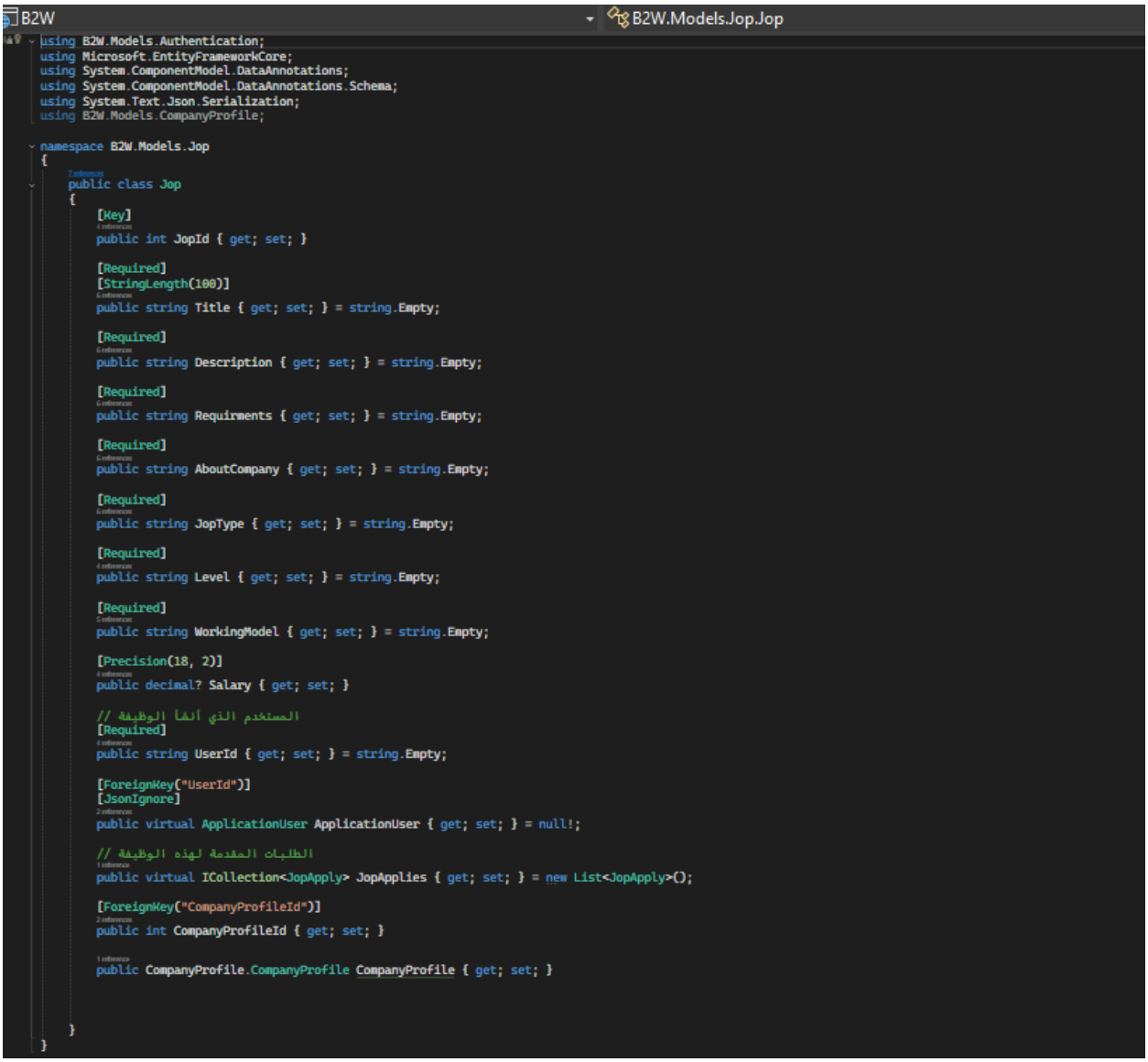


Figure 5.19 Job Model

Flutter Implementation 5.2.3

1-Login Screen

- **UI**: Background image, logo, email/password fields with validation, "Log in" button, "Forgot password?" link, and social login options (Google, Apple, Facebook).
- **Functionality**: Form validation (GlobalKey<FormState>), navigation to NavBarScreen or SignUpScreen, and responsive design with spacing widgets.
- **Flow**: Validates inputs before allowing login, with social sign-in placeholders.

```dart
import ...

class LoginScreen extends StatelessWidget {
  LoginScreen({super.key});
  final formKey = GlobalKey<FormState>();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Form(
        key: formKey,
        child: SingleChildScrollView(
          child: SafeArea(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.start,
              children: [
                Stack(
                  children: [

                    Container(
                      width: 395.w,
                      height: 194.h,

                      decoration: BoxDecoration(
                        borderRadius: BorderRadius.all(Radius.circular(20.r)),

                      ),  // BoxDecoration
                      child: Image.asset('Rectangle 1'.assetPNG,
                        width: 395.w,
                        height: 194.h,
                        fit: BoxFit.fill,


                      ),  // Image.asset
                    ),  // Container
                    Positioned(
                      top: 141,
                        left: 169 ,
                        child: Image.asset('logo1'.assetPNG,
                        width: 55.w,
                        height: 48.h,))  // Image.asset, Positioned

                  ],
```

Figure 5.20.1 Login screens

```
      validator: (value)
      {
        if (value!.isEmpty)
        {
          return 'password must not be empty';

        }
        return null;

      },
    ),  // AppTextField
    Align(
      alignment: Alignment.centerRight,

      child: AppText(title: 'Forgot password?',
        color:AppColors.grey ,
      fontSize: 14,
      decoration: TextDecoration.underline,
        onTap: ()=>RouteUtils.pushAndRemoveAll(context,ForgotPasswordScreen()),
      ),  // AppText
    ),  // Align
    SizedBox(
      height: 24.h,
    ),  // SizedBox
    AppButton(title: 'Log in',
        onTap: (){
      formKey.currentState!.validate();
      RouteUtils.push(context, NavBarScreen());

        },
    ),  // AppButton
  SizedBox(
    height: 8.h,
  ),  // SizedBox
   Row(
      children: [
        Expanded(
          child: Divider(
            height: 2,
            color: AppColors.grey,
          ),  // Divider
        ),  // Expanded
        SizedBox(
          width: 8.w,
```

```
          SizedBox(
            height: 8.h,
          ),  // SizedBox
          Padding(
            padding: const EdgeInsets.all(20.0),
            child: Column(
              children: [
                AppText(
                  title: 'Login to your Account',
                  fontWeight: FontWeight.w700,
                  fontFamily: 'Lato',
                  fontSize: 21,
                  textAlign: TextAlign.center,
                  color: AppColors.inputText,
                ),  // AppText
                SizedBox(
                  height: 8.h,
                ),  // SizedBox
                AppTextField(
                  hint: 'Example@gmail.com',
                  label: 'Email Address ' ,
                  keyboardType: TextInputType.emailAddress,

                  validator: (value)
                  {
                    if (value!.isEmpty)
                    {
                      return 'email must not be empty';

                    }
                    return null;

                  },

                  ),  // AppTextField
                SizedBox(
                  height: 8.h,
                ),  // SizedBox
                AppTextField
                  (
                  label: 'Password',
                  secure: true,
                  hint: '*******', keyboardType: TextInputType.visiblePassword,
```

Figure 5.20.2 Login screens

```
          ),  // SizedBox
          AppText(title: 'OR log in with',
          color: AppColors.inputText,
          fontSize: 16,
          fontWeight: FontWeight.w400,
          ),  // AppText
          SizedBox(
            width: 8.w,
          ),  // SizedBox
          Expanded(
            child: Divider(
              height: 2,
              color: AppColors.grey,
            ),  // Divider
          ),  // Expanded
        ],
      ),  // Row
      SizedBox(
        height: 8.h,
      ),  // SizedBox

      Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          InkWell(

            child: Image.asset('google'.assetPNG,
            width: 44.w,
            height: 44.w,),  // Image.asset
            onTap: (){},
          ),  // InkWell
          SizedBox(
            width: 4.w,
          ),  // SizedBox
          InkWell(
            child: Image.asset('apple'.assetPNG,
              width: 44.w,
              height: 44.w,),  // Image.asset
            onTap: (){},
          ),  // InkWell
          SizedBox(
            width: 4.w,
          ),  // SizedBox
          InkWell(
            child: Image.asset('facebook'.assetPNG,
              width: 44.w,
              height: 44.w,),  // Image.asset
            onTap: (){},
          ),  // InkWell
        ],
      ),  // Row
      SizedBox(
        height: 8.h,
      ),  // SizedBox
      Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          AppText(title: 'Don't Have an account?',
          color: AppColors.darkGrey,
          fontSize: 16,
          fontWeight: FontWeight.w400,),  // AppText
          InkWell(
            child: AppText(title: 'Sign up',
              color: AppColors.primary,
              fontSize: 16,
              fontWeight: FontWeight.w400,
            ),  // AppText
            onTap: (){
              if (formKey.currentState!.validate())
              {


              }
              RouteUtils.pushAndRemoveAll(context, SignUpScreen());
```

Figure 5.20.3 Login screens

2-Sign-Up Screen

- **UI**: Background image (Rectangle 2) with logo, form fields (first name, last name, email, password), "Sign up" button, and social sign-up options (Google, Apple, Facebook).
- **Functionality**: Form validation (GlobalKey<FormState>), navigation to SelectScreen after validation, and a "Log in" link for existing users.
- **Flow**: Validates inputs before submission, with dividers ("OR Sign Up with") separating social login options.

```dart
import ...

class SignUpScreen extends StatelessWidget {
  SignUpScreen ({super.key});

  final formKey = GlobalKey<FormState>();
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SingleChildScrollView(
        child: SafeArea(
          child: Column(
            children: [

              Stack(

                children: [
                  Container(
                    width:395.w,
                    height: 194.h,

                    decoration: BoxDecoration(
                      borderRadius: BorderRadius.all(Radius.circular(20.r)),

                    ),  // BoxDecoration
                    child: Image.asset('Rectangle 2'.assetPNG,
                      width: 395.w,
                      height: 194.h,
                      fit: BoxFit.fill,

                    ),  // Image.asset
                  ),  // Container

                  Positioned(
                      top: 141,
                      left: 169 ,
                      child: Image.asset('logo1'.assetPNG,
                        width: 55.w,
                        height: 48.h,))  // Image.asset, Positioned

                ],

              ),  // Stack
              SizedBox(
```

Figure 5.21.1 Signup screens

```
            height: 8.h,
          ),   // SizedBox
          Padding(
            padding: const EdgeInsets.all(20.0),
            child: Form(

              key: formKey,
              child: Column(

                children: [
                  AppText(
                    title: 'Create your Account',
                    fontWeight: FontWeight.w700,
                    fontFamily: 'Lato',
                    fontSize: 21,
                    textAlign: TextAlign.center,
                    color: AppColors.inputText,
                  ),   // AppText
                  SizedBox(
                    height: 8.h,
                  ),   // SizedBox
                  Row(
                    mainAxisAlignment: MainAxisAlignment.start,
                    children: [
                      Expanded(
                        child: AppTextField(
                          hint: 'first name',
                          label: 'First Name',
                          keyboardType: TextInputType.text,
                          width: 170.w,

                          validator: (value)
                          {
                            if (value!.isEmpty)
                            {
                              return 'Name must not be empty';

                            }
                            return null;

                          },

                        ),   // AppTextField
                      ),   // Expanded
                      SizedBox(
                        width: 8.w,
                      ),   // SizedBox
                      Expanded(
                        child: AppTextField(
                          hint: 'last name',
                          label: 'Last Name',
                          keyboardType: TextInputType.text,
                          width: 170.w,
                          validator: (value)
                          {
                            if (value!.isEmpty)
                            {
                              return ' Name must not be empty';

                            }
                            return null;

                          },

                        ),   // AppTextField
                      ),   // Expanded
                    ],
                  ),   // Row
                  SizedBox(
                    height: 8.h,
                  ),   // SizedBox
                  AppTextField
                    (hint: 'Example@gmail.com',
                    label: 'Email Address ' ,
                    keyboardType: TextInputType.emailAddress,
                    validator: (value)
                    {
                      if (value!.isEmpty)
                      {
                        return 'email must not be empty';

                      }
                      return null;

                    },
                  ),   // AppTextField
                  SizedBox(
                    height: 8.h,
```

Figure 5.21.2 Signup screens

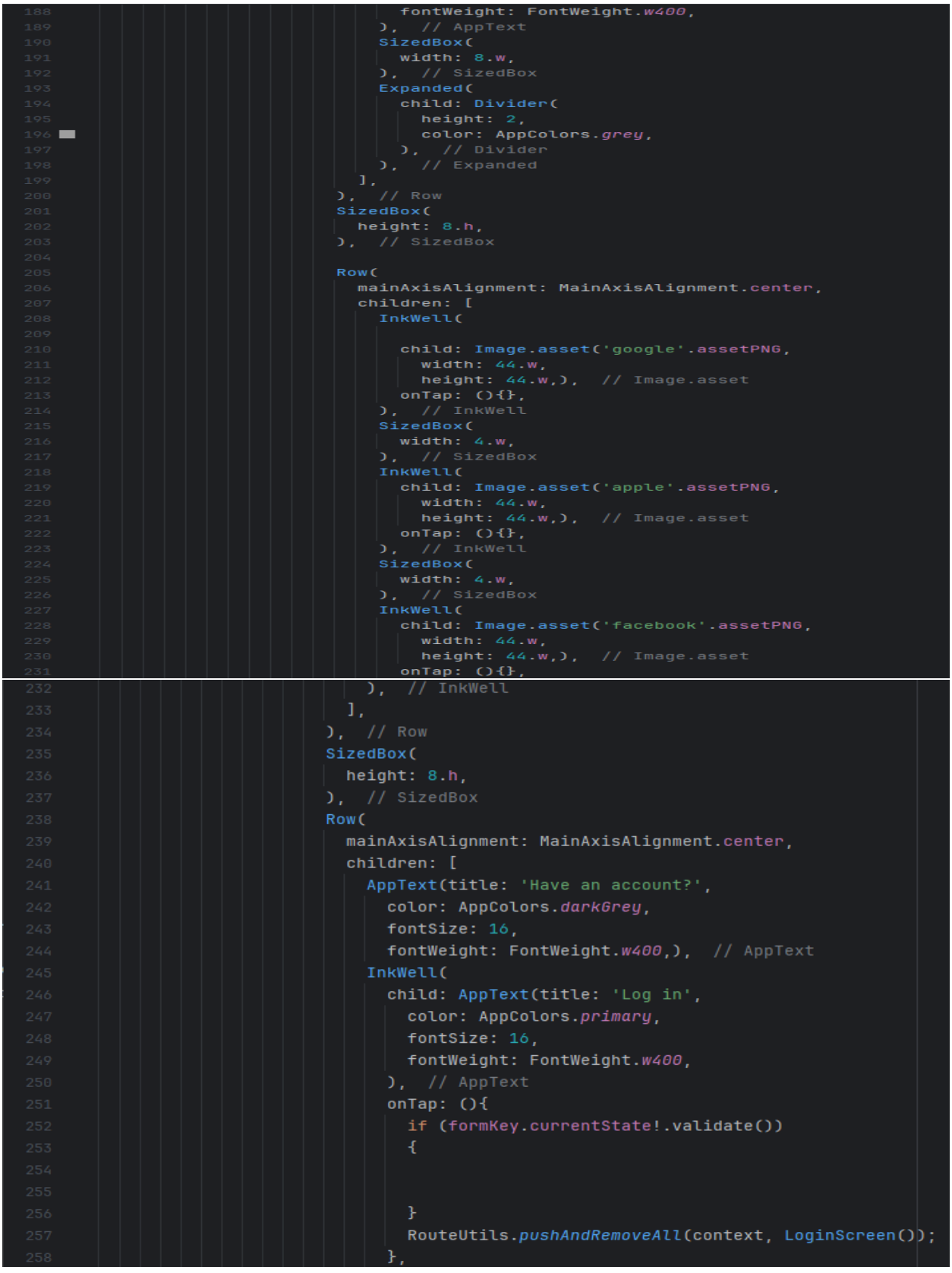
```
AppTextField
  (
  label: 'Password',
  secure: true,
  hint: '*******',
  keyboardType: TextInputType.visiblePassword,
  validator: (value)
  {
    if (value!.isEmpty)
    {
      return 'password must not be empty';

    }
    return null;

  },

),  // AppTextField
SizedBox(
  height: 8.h ,
),  // SizedBox
AppButton(title: 'Sign up',
  onTap: (){
    formKey.currentState!.validate();
    RouteUtils.push(context, SelectScreen());
  },
),  // AppButton
SizedBox(
  height: 8.h,
),  // SizedBox
Row(
  children: [
    Expanded(
      child: Divider(
        height: 2,
        color: AppColors.grey,
      ),  // Divider
    ),  // Expanded
    SizedBox(
      width: 8.w,
    ),  // SizedBox
    AppText(title: 'OR Sign Up with',
      color: AppColors.inputText,
      fontSize: 16,
```

Figure 5.21.3 Signup screens



```
                fontWeight: FontWeight.w400,
              ),  // AppText
              SizedBox(
                width: 8.w,
              ),  // SizedBox
              Expanded(
                child: Divider(
                  height: 2,
                  color: AppColors.grey,
                ),  // Divider
              ),  // Expanded
            ],
          ),  // Row
          SizedBox(
            height: 8.h,
          ),  // SizedBox

          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              InkWell(

                child: Image.asset('google'.assetPNG,
                  width: 44.w,
                  height: 44.w,),  // Image.asset
                onTap: (){},
              ),  // InkWell
              SizedBox(
                width: 4.w,
              ),  // SizedBox
              InkWell(
                child: Image.asset('apple'.assetPNG,
                  width: 44.w,
                  height: 44.w,),  // Image.asset
                onTap: (){},
              ),  // InkWell
              SizedBox(
                width: 4.w,
              ),  // SizedBox
              InkWell(
                child: Image.asset('facebook'.assetPNG,
                  width: 44.w,
                  height: 44.w,),  // Image.asset
                onTap: (){},
              ),  // InkWell
            ],
          ),  // Row
          SizedBox(
            height: 8.h,
          ),  // SizedBox
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              AppText(title: 'Have an account?',
                color: AppColors.darkGrey,
                fontSize: 16,
                fontWeight: FontWeight.w400,),  // AppText
              InkWell(
                child: AppText(title: 'Log in',
                  color: AppColors.primary,
                  fontSize: 16,
                  fontWeight: FontWeight.w400,
                ),  // AppText
                onTap: (){
                  if (formKey.currentState!.validate())
                  {


                  }
                  RouteUtils.pushAndRemoveAll(context, LoginScreen());
                },
```

Figure 5.21.4 Signup screens

3-Email Verification Screen

- **UI**: App bar with back button, OTP input field (4 digits), "Resend Code?" link, and "Verify" button.
- **Function**: Validates OTP code, navigates to ResetPasswordScreen on success. Handles code resend requests.
- **Flow**: User enters code → verification → password reset. Clean design with purple accents.

```dart
import ...

class VerifyEmailScreen extends StatefulWidget {
  @override
  State<VerifyEmailScreen> createState() => _VerifyEmailScreenState();
}

class _VerifyEmailScreenState extends State<VerifyEmailScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: AppColors.white,
      appBar: AppBar(
        title: AppText(
          title: 'Verify Your Code',
          fontWeight: FontWeight.w400,
          fontFamily: 'Lato',
          fontSize: 25.sp,
          textAlign: TextAlign.center,
          color: AppColors.black,
        ),  // AppText
        centerTitle: true,
        leading: IconButton(
          icon: Icon(
            Icons.arrow_back_ios,
            color: Colors.black,
            size: 30.sp,
          ),  // Icon
          onPressed: () => Navigator.pop(context),
        ),  // IconButton
        backgroundColor: Colors.white,
        elevation: 0,
      ),  // AppBar
      body: Padding(
        padding: const EdgeInsets.symmetric(horizontal: 24.0),
        child: SingleChildScrollView(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
              Image.asset(
                'Drone Delivery-cuate 1'.assetPNG,

              ),  // Image.asset
```

Figure 5.22.1 Email Verification Screen

```
SizedBox(height: 20.h),
AppText(
  title:
      'A verification code has been sent to your email. Please enter it here to verify your identity.',
  fontWeight: FontWeight.w700,
  fontFamily: 'Lato',
  fontSize: 18.sp,
  textAlign: TextAlign.center,
  color: AppColors.inputText,
),  // AppText
SizedBox(height: 10.h),
OtpTextField(
  numberOfFields: 4,
  fieldWidth: 43,
  borderRadius: BorderRadius.circular(10),
  borderWidth: 2,
  cursorColor: Colors.purple,
  focusedBorderColor: Colors.purple,
  showFieldAsBox: true,
  onCodeChanged: (String code) {},
  onSubmit: (String verificationCode) {},
),  // OtpTextField
SizedBox(height: 10.h),
TextButton(
  onPressed: () {

  },
  child:  AppText(
       decoration: TextDecoration.underline,
      title: 'Resend Code?',
      fontWeight: FontWeight.w700,
      fontFamily: 'Roboto',
      fontSize: 18.sp,
      textAlign: TextAlign.center,
      color: AppColors.primary,
    ) ),  // AppText, TextButton

SizedBox(height: 30.h),
AppButton(
  title: 'Verify',
  onTap: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => ResetPasswordScreen()),
```

Figure 5.22.2 Email Verification Screen

4-Add New Project For User Screens

- **Core UI**: App bar with back button, project title/description fields, image upload area, and "Add" button.
- **Key Features**:
    - Image picker (gallery) with preview
    - Expandable description field
    - Form validation for required fields
- **Flow**: User adds project details → attaches image → submits. Clean white/grey layout with proper spacing.

```dart
import ...

class AddNewProject extends StatefulWidget {
  const AddNewProject({super.key});

  @override
  State<AddNewProject> createState() => _AddNewProjectState();
}

class _AddNewProjectState extends State<AddNewProject> {
  XFile? _selectedImage;

  Future<void> _pickImage() async {
    final ImagePicker picker = ImagePicker();
    final XFile? image = await picker.pickImage(source: ImageSource.gallery);

    if (image != null) {
      setState(() {
        _selectedImage = image;
      });
    }
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        leading: IconButton(
          icon: Padding(
            padding: const EdgeInsets.all(20.0),
            child: Icon(Icons.arrow_back_ios, color: AppColors.black, size: 40),
          ),  // Padding
          onPressed: () => Navigator.of(context).pop(),
        ),  // IconButton
        title: Center(
          child: Text(
            "Add New Project",
            style: TextStyle(
              color: AppColors.black,
              fontSize: 25,
              fontWeight: FontWeight.w400,
              fontFamily: 'Lato',
            ),  // TextStyle
          ),  // Text
        ),  // Center
```

Figure 5.23.1 Add New Project

```
    elevation: 0,
    toolbarHeight: 100,
  ),  // AppBar
  body: SingleChildScrollView(
    child: Padding(
      padding: const EdgeInsets.all(20.0),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.start,
        children: [
          Row(
            mainAxisAlignment: MainAxisAlignment.start,
            children: [
              AppText(title: 'Project Title',color: AppColors.inputText,
                fontSize: 16,
                fontWeight: FontWeight.w400,),  // AppText
            ],
          ),  // Row
          SizedBox(
            height: 8.h,
          ),  // SizedBox
          AppTextField
            (hint: 'EX :  bookind app',

            keyboardType: TextInputType.text,
            validator: (value)
            {
              if (value!.isEmpty)
              {
                return 'Organization name must not be empty';

              }
              return null;

            },
          ),  // AppTextField
          SizedBox(
            height: 16,
          ),  // SizedBox
          Row(
            mainAxisAlignment: MainAxisAlignment.start,
            children: [
              AppText(title: 'Description',color: AppColors.inputText,
                fontSize: 16,
                fontWeight: FontWeight.w400,),  // AppText
```

```
      SizedBox(height: 8.h),
      Container(
        height: 170.h,
        decoration: BoxDecoration(
            color: AppColors.white,
            borderRadius: BorderRadius.circular(10),
            border: Border.all(
              color: AppColors.grey,
            )  // Border.all
        ),  // BoxDecoration
        child: TextField(

          maxLines: null,
          expands: true,
          decoration: InputDecoration(
            hintText: 'type here ..',
            helperStyle: TextStyle(color: AppColors.grey),
            border: InputBorder.none,
            contentPadding: EdgeInsets.all(12),
          ),  // InputDecoration
        ),  // TextField
      ),  // Container
      SizedBox(
        height: 16,
      ),  // SizedBox
      Row(
        mainAxisAlignment: MainAxisAlignment.start,
        children: [
          AppText(title: 'Media',color: AppColors.inputText,
            fontSize: 16,
            fontWeight: FontWeight.w400,),  // AppText
        ],
      ),  // Row
      GestureDetector(
        onTap: _pickImage,
        child: Container(
          height: 200.h,
          width: double.infinity,
          decoration: BoxDecoration(
            color: AppColors.white,
            borderRadius: BorderRadius.circular(15),
            border: Border.all(color: AppColors.grey),
          ),  // BoxDecoration
```

Figure 5.23.2 Add New Project

```
                child: _selectedImage == null
                    ? Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    Icon(Icons.image, size: 40, color: AppColors.darkGrey),
                    SizedBox(height: 8),
                    Text(
                      'Upload your image (JPEG, PNG only) by clicking here',
                      style: TextStyle(color: AppColors.grey),
                      textAlign: TextAlign.center,
                    ),  // Text
                  ],
                )  // Column
                    : Image.file(
                  File(_selectedImage!.path),
                  fit: BoxFit.cover,
                ),  // Image.file
              ),  // Container
            ),  // GestureDetector
            SizedBox(
              height: 16,
            ),  // SizedBox
            InkWell(
              onTap: (){},
              child: Container(
                width: double.infinity,
                height: 42,
                decoration: BoxDecoration(
                  color: AppColors.lightGrey,
                  borderRadius: BorderRadius.circular(15.r),
                  border: Border.all(
                    color: AppColors.lightGrey,
                  ),  // Border.all

                ),  // BoxDecoration
                child: Center(
                  child: Text(
                    'Add',
                    style: TextStyle(
                      color: AppColors.inputText,
                      fontSize: 18,
                      fontWeight: FontWeight.w700,
```

Figure 5.23.3 Add New Project

5-Company Profile Screen

- **UI**: Profile header (banner, logo, bio), 4-tab navigation (Posts/Jobs/Reviews/People), and team/review listings.
- **Features**:
    - **Team management**: Add/remove people with mock avatars (pravatar.cc).
    - **Reviews**: Displays employee feedback with local images.
    - **Tabs**: Purple-themed tab bar for navigation.
- **Actions**: Back button, settings access, and expandable sections.

```dart
import ...

class CompanyProfile extends StatefulWidget {

  const CompanyProfile({super.key});

  @override
  State<CompanyProfile> createState() => _CompanyProfileState();
}

class _CompanyProfileState extends State<CompanyProfile> {
  List<Map<String, String>> people = [
    {
      "name": "Liam Carter",
      "role": "CEO",
      "image": "https://i.pravatar.cc/150?img=1"
    },
    {
      "name": "Murad Mohamed",
      "role": "Product manager",
      "image": "https://i.pravatar.cc/150?img=3"
    },
    {
      "name": "Salma Ahmed",
      "role": "Product Designer",
      "image": "https://i.pravatar.cc/150?img=5"
    },
    {
      "name": "Ali Wael",
      "role": "Graphic Designer",
      "image": "https://i.pravatar.cc/150?img=7"
    },
    {
      "name": "Waleed Ali",
      "role": "HR",
      "image": "https://i.pravatar.cc/150?img=9"
    },
    {
      "name": "Sophia Bennett",
      "role": "Motion designer",
      "image": "https://i.pravatar.cc/150?img=10"
    },
    {
```

Figure 5.24.1 Company Profile

```
      "name": "Mia Collins",
      "role": "Product manager",
      "image": "https://i.pravatar.cc/150?img=15"
    },
    {
      "name": "Ethan Rivera",
      "role": "Product manager",
      "image": "https://i.pravatar.cc/150?img=20"
    },
  ];
  List<Map<String, String>> reviews = [
    {
      "name": "Murad Mohamed",
      "title": "Product Manager",
      "job": "Managing product development and strategy",
      "image": "assets/images/Ellipse 189(4).png"
    },
    {
      "name": "Aisha Patel",
      "title": "UX Designer",
      "job": "Creating user-friendly designs and experiences",
      "image": "assets/images/Ellipse 189(2).png"
    },
    {
      "name": "Reham Ahmed",
      "title": "UI Developer",
      "job": "Developing interactive UI components",
      "image": "assets/images/Ellipse 189(3).png"
    },
    {
      "name": "Juan Rodriguez",
      "title": "Content Strategist",
      "job": "Planning and optimizing digital content",
      "image": "assets/images/Ellipse 189(1).png"
    },
    {
      "name": "Yasmin Essam",
      "title": "Marketing Specialist",
      "job": "Developing marketing campaigns and strategies",
      "image": "assets/images/Ellipse 189.png"
    },
  ];
```

Figure 5.24.2 Company Profile

```dart
  void _removePerson(int index) {
    setState(() {
      people.removeAt(index);
    });
  }

  void _addPerson() {
    setState(() {
      people.add({
        "name": "New Person",
        "role": "New Role",
        "image": "https://i.pravatar.cc/150?img=${people.length + 1}"
      });
    });
  }

  @override
  Widget build(BuildContext context) {
    return DefaultTabController(
        length: 4, // عدد التبويبات
        child: Scaffold(
          backgroundColor: Colors.white,
            body: Column(children: [
          customStackImage(
              backgroundImage:
                  'assets/images/WhatsApp Image 2025-03-04 at 5.47.31 AM(1).jpeg',
              profileImage: 'assets/images/Ellipse 13.png',
              name: 'Alaa Mohamed',
              jobTitle: 'Digital Creative Agency',
              bio:
                  'Empowering brands with creative digital solutions and strategies. Let's innovate together!',
              followers: '100',
              posts: '',
              extraWidgets: [
                Center(
                  child: Row(
                    children: [
                      Icon(Icons.link),
                      spaceWidth(3),
                      Text("http://Digital Creative Agency.name")
                    ],
                  ),  // Row
                )  // Center
              ],
              showPosts: false, onBackPressed: () { Navigator.pop(context); }, onPressedSettings: ()
          { Navigator.push(context, MaterialPageRoute(builder: (context) => SettingsPage(),)); }),
          SizedBox(height: 260),
          TabBar(
            indicatorColor: Colors.purple,
            labelColor: Colors.purple,
            unselectedLabelColor: Colors.grey,
            labelStyle: TextStyle(fontWeight: FontWeight.bold),
            tabs: [
              Tab(text: "Posts"),
              Tab(text: "Open jobs"),
              Tab(text: "Reviews"),
              Tab(text: "people"),
            ],
          ),  // TabBar
          Expanded(
            child: TabBarView(
              children: [
                aboutTab(),
                openJobTab(),
                reviewsTab(),
                peopleTab(onPressed: () {Navigator.push(context, MaterialPageRoute(builder: (context) => AddPeopleScreen(),)) ; })
              ],
            ),  // TabBarView
          ),  // Expanded
        ])));  // Column, Scaffold, DefaultTabController
  }
```

Figure 5.24.3 Company Profile

6-Chatbot interview simulation component

This chatbot feature simulates a real job interview using artificial intelligence. It allows the user to practice interviews by chatting with a smart assistant powered by Google Gemini. The user first selects their desired **field** (such as front-end or back-end development) and **difficulty level** (easy, medium, or hard). Based on this input, the chatbot begins asking relevant interview questions. The user responds by typing their answers, and the AI provides helpful feedback and evaluations in real time. This process helps users prepare for actual interviews by improving their technical knowledge, confidence, and communication skills. The chatbot offers a friendly and interactive experience, especially useful for people with disabilities who may want to practice interviews in a comfortable, self-paced environment

```dart
import 'package:b2w/core/extentions/string.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:google_generative_ai/google_generative_ai.dart';
import '../../../core/utils/colors.dart';
const String apiKey = 'AIzaSyD-GqAOJ0TuoaukqZ3frSvjUvdthm56tRs';
class ChatbotItself extends StatefulWidget {
  const ChatbotItself({super.key});
  @override
  State<ChatbotItself> createState() => _ChatbotItselfState(
  );
}
class _ChatbotItselfState extends State<ChatbotItself> {
 late final GenerativeModel model;
 late final ChatSession chat ;
 final ScrollController scrollController = ScrollController();
 final TextEditingController textController = TextEditingController();
 final List< ChatMessages > messages = [];
 @override
  void initState() {
    super.initState();
    model = GenerativeModel(model: 'gemini-1.5-flash', apiKey: apiKey);
    chat = model.startChat();
  }
  void scrollDown(){
   WidgetsBinding.instance.addPostFrameCallback((_)=> scrollController
       .animateTo(scrollController.position.maxScrollExtent, duration: Duration(milliseconds: 750), curve: Curves.easeInOutCirc));
}
```

Figure 5.25.1 Chatbot interview

```
Future <void> sendChatMessage(String message) async{
setState(() {
  messages.add(ChatMessages(text: message, isUser: true));
});
try{
  final response = await chat.sendMessage(Content.text(message));
  final text = response.text;
  setState(() {
    messages.add(ChatMessages(text: text!, isUser : false));
    scrollDown();
  });
}catch(e){
  setState(() {
    messages.add(ChatMessages(text: 'Error occured ', isUser: false));
  });
}finally {
  textController.clear();
}
```

```
class ChatMessages {
  final String text;
  final bool isUser;
  ChatMessages({required this.text, required this.isUser});

}
class ChatBubble extends StatelessWidget {
  final ChatMessages messages;
  const ChatBubble({super.key, required this.messages});
  @override
  Widget build(BuildContext context) {
    return Container(
      margin: EdgeInsets.symmetric(vertical: 10, horizontal: 14),
        alignment: messages.isUser ? Alignment.centerRight:Alignment.centerLeft,
        child: Container(
          width: 284,
          decoration: BoxDecoration(
              color: messages.isUser ?AppColors.white: AppColors.lightGrey,
              borderRadius: BorderRadius.only(
                topLeft: Radius.circular(20),
                topRight: Radius.circular(20),
                bottomRight:messages.isUser? Radius.circular(0):Radius.circular(20),
                bottomLeft: messages.isUser? Radius.circular(20):Radius.circular(0),
              )  // BorderRadius.only
          ),  // BoxDecoration
          constraints: BoxConstraints(
            maxWidth: MediaQuery.of(context).size.width /1.5,
          ),  // BoxConstraints
```

Figure 5.25.2 Chatbot interview

Ai and machine learning implementation 5.2.4

1. CV Analysis:

is the use of machine learning, natural language processing (NLP), and data mining techniques to automatically read, understand, classify, and score job applicants' resumes in order to streamline recruitment and match candidates to job requirements.

Resume Parsing (Information Extraction):

The code uses the pyresparser library, which relies on NLP techniques to extract structured data from unstructured resume text:

- Name, email, phone number
- Number of pages
- Detected skills

This is a classic application of AI in the form of text analysis.

2. Smart Recommendations:

Once skills are extracted, the system uses a rule-based AI approach to match the user to a job domain:

- Data Science
- Web Development
- Android/iOS/UX
- It then recommends:

  Additional skills to learn

  Relevant courses and certifications

This is basic AI logic based on pattern recognition.

3. Resume Scoring:

The code checks whether your resume includes essential components like:

- Objective
- Declaration
- Projects
- Achievements
- Hobbies

Based on that, it gives a resume score, simulating a basic AI assistant to evaluate the quality of your resume.

1) Importing Libraries

First, we imported the libraries we were going to use

Streamlit: Makes it easy to create and share custom web applications for machine learning and data science.

Pandas: used for working with datasets, it has functions for analyzing, cleaning, exploring, and manipulating data.

Base64, Random: Converts data to/from Base64 and generates random numbers.

Time, datetime: used to handle time and dates.

Io: to handle the stream of files and data.

pyresparser: a library for parsing CVs. It often relies on spacy and pdfminer.

pdfminer3: a library for extracting text from PDF files.

streamlit_tags: a Streamlit plugin for providing interactive components such as tag boxes.

PIL: a library for image processing. Image is used to open and modify images.

pymysql: for interacting with MySQL databases using Python.

pafy: a library for downloading information or videos from YouTube.

plotly. express (px): for easily creating interactive charts.

nltk: a library for Natural Language Processing.

```python
import streamlit as st
import pandas as pd
import base64,random
import time,datetime
#libraries to parse the resume pdf files
from pyresparser import ResumeParser
from pdfminer3.layout import LAParams, LTTextBox
from pdfminer3.pdfpage import PDFPage
from pdfminer3.pdfinterp import PDFResourceManager
from pdfminer3.pdfinterp import PDFPageInterpreter
from pdfminer3.converter import TextConverter
import io,random
from streamlit_tags import st_tags
from PIL import Image
import pymysql
from Courses import ds_course,web_course,android_course,ios_course,uiux_course,resume_videos,interview_videos
import pafy #for uploading youtube videos
import plotly.express as px #to create visualisations at the admin session
import nltk
```

Figure 5.26 importing libraries

```
# Create table
DB_table_name = 'user_data'
table_sql = "CREATE TABLE IF NOT EXISTS " + DB_table_name + """
                (ID INT NOT NULL AUTO_INCREMENT,
                 Name varchar(500) NOT NULL,
                 Email_ID VARCHAR(500) NOT NULL,
                 resume_score VARCHAR(8) NOT NULL,
                 Timestamp VARCHAR(50) NOT NULL,
                 Page_no VARCHAR(5) NOT NULL,
                 Predicted_Field BLOB NOT NULL,
                 User_level BLOB NOT NULL,
                 Actual_skills BLOB NOT NULL,
                 Recommended_skills BLOB NOT NULL,
                 Recommended_courses BLOB NOT NULL,
                 PRIMARY KEY (ID));
                """
```

Figure 5.27 Storing results

```
def pdf_reader(file):
    resource_manager = PDFResourceManager()
    fake_file_handle = io.StringIO()
    converter = TextConverter(resource_manager, fake_file_handle, laparams=LAParams())
    page_interpreter = PDFPageInterpreter(resource_manager, converter)
    with open(file, 'rb') as fh:
        for page in PDFPage.get_pages(fh,
                                      caching=True,
                                      check_extractable=True):
            page_interpreter.process_page(page)
            print(page)
        text = fake_file_handle.getvalue()

    # close open handles
    converter.close()
    fake_file_handle.close()
    return text

def show_pdf(file_path):
    with open(file_path, "rb") as f:
        base64_pdf = base64.b64encode(f.read()).decode('utf-8')
    # pdf_display = f'<embed src="data:application/pdf;base64,{base64_pdf}" width="700" height="1000" type="application/pdf">'
    pdf_display = F'<iframe src="data:application/pdf;base64,{base64_pdf}" width="700" height="1000" type="application/pdf"></iframe>'
    st.markdown(pdf_display, unsafe_allow_html=True)
```

Figure 5.28 : Read and Show PDF

2. Sign Language Translator:

is used to automatically translate gestures, facial expressions, and body movements used in sign language into spoken or written language, and vice versa. It helps bridge communication between deaf or hard-of-hearing individuals and those who do not understand sign language.

Dataset (Sign Language MNIST):

The original MNIST image dataset of handwritten digits is a popular benchmark for image-based machine learning methods but researchers have renewed efforts to update it and develop drop-in replacements that are more challenging for computer vision and original for real-world applications. As noted in one recent replacement called the Fashion-MNIST dataset, the Zalando researchers quoted the startling claim that "Most pairs of MNIST digits (784 total pixels per sample) can be distinguished pretty well by just one pixel". To stimulate the community to develop more drop-in replacements, the Sign Language MNIST is presented here and follows the same CSV format with labels and pixel values in single rows. The American Sign Language letter database of hand gestures represent a multi-class problem with 24 classes of letters (excluding J and Z which require motion).

The dataset format is patterned to match closely with the classic MNIST. Each training and test case represents a label (0-25) as a one-to-one map for each alphabetic letter A-Z (and no cases for 9=J or 25=Z because of gesture motions). The training data (27,455 cases) and test data (7172 cases) are approximately half the size of the standard MNIST but otherwise similar with a header row of label, pixel1,pixel2….pixel784 which represent a single 28x28 pixel image with grayscale values between 0-255. The original hand gesture image data represented multiple users repeating the gesture against different backgrounds. The Sign Language MNIST data came from greatly extending the small number (1704) of the color images included as not cropped around the hand region of interest. To create new data, an image pipeline was used based on ImageMagick and included cropping to hands-only, gray-scaling, resizing, and then creating at least 50+ variations to enlarge the quantity. The modification and expansion strategy was filters ('Mitchell', 'Robidoux', 'Catrom', 'Spline', 'Hermite'), along with 5% random pixelation, +/- 15% brightness/contrast, and finally 3 degrees rotation. Because of the tiny size of the images, these ways.

Type	CNN	LSTM	Mobilenetv2
Accuracy	98%	81%	95%
Loss	0.03	0.89	0.2

Table 5.1 accuracy and loss

According to this table, we will use CNN because it has the highest accuracy and lowest LOSS.

```
plt.figure(figsize=(12, 5))

# Plot loss
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plot accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

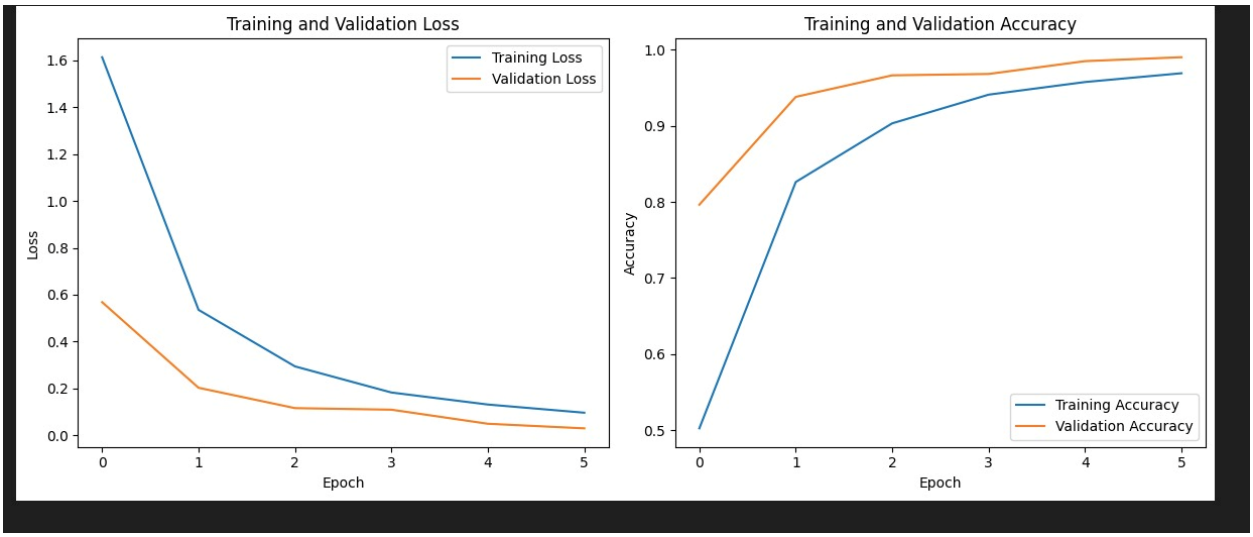


Figure 5.29 Accuracy and Loss

```
import matplotlib.pyplot as plt
fig,axe=plt.subplots(1,2)
fig.suptitle('Preview of dataset')
axe[0].imshow(train_images[0].reshape(28,28),cmap='gray')
axe[0].set_title('label: 3  letter: C')
axe[1].imshow(train_images[1].reshape(28,28),cmap='gray')
axe[1].set_title('label: 6  letter: F')
```

```
Text(0.5, 1.0, 'label: 6  letter: F')
```

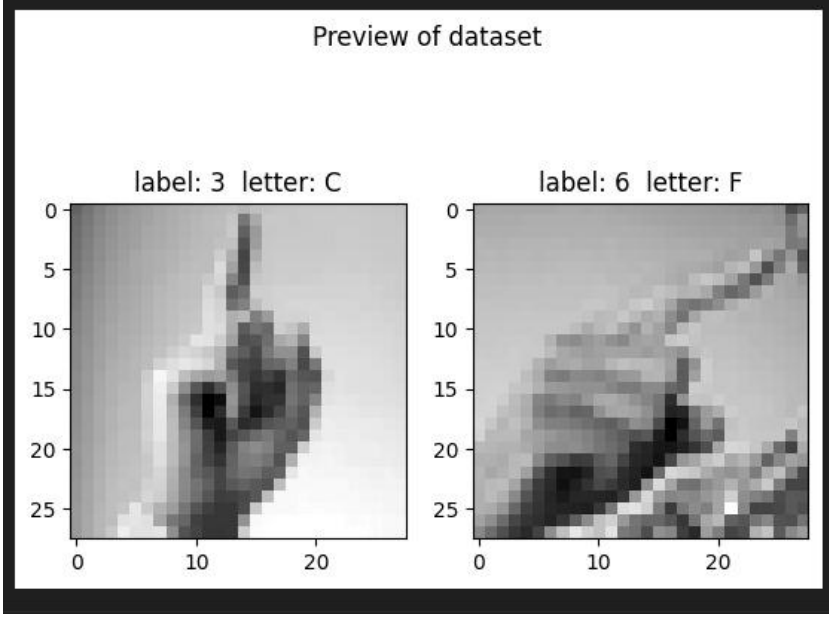


Figure 5.30 View Letters (Results)

5.3 System Testing

Name	Description	Test Result
Register (create account)	Create new accounts for employee or companies each one with their information	success
login	Check authenticity for each user	success
Upload jobs	Company upload jobs	success
Edit profile data	Edit company or employee profile information	success
Apply to jobs	Each employee can applies for company's positions	success
Upload posts	Each employee can upload posts	success
Cv analysis	Analysis cv and give enhancement and recommendations	success
Chatbot interview	Simulation of interview with AI	success
Sign language translator	Translate sign language to text	success

Table 5.2 : system testing

5.4 Goals Achieved

Throughout the implementation of our system, we were able to transform our initial design and conceptual framework into a fully functional application that effectively meets the objectives we originally set. The integration between the frontend, backend, and AI-based components has been completed successfully, and each module functions as expected across various user scenarios.

From the user perspective, we have successfully implemented an intuitive and responsive interface that allows both employees and companies to interact with the platform efficiently. Essential features such as account registration, login, role assignment, and protected routing were implemented with proper state management using React Context API. Furthermore, features like dynamic dashboards, post uploading, and user role-based navigation were built in a scalable and maintainable structure.

In terms of backend achievements, the system architecture was structured using ASP.NET Core Web API, following RESTful principles. All controllers were correctly mapped to handle authentication, user and company profiles, and job management. The system supports CRUD operations, secure token-based authentication, and enforces role-based access to ensure only authorized users can access or perform specific actions.

In addition to the core system functionality, we successfully integrated two AI-powered features: CV Analysis and Sign Language Translator. The CV analysis module reads and interprets resumes using natural language processing, extracts useful information, and provides customized suggestions to the user. It even evaluates the resume with a score, simulating a real recruitment assistant. On the other hand, the Sign Language Translator component provides an inclusive solution by converting hand gestures into text using a trained CNN model with high accuracy, enhancing accessibility for users with hearing impairments.

Another key achievement is the chatbot interview simulation. This feature provides users with a realistic mock interview experience powered by AI, tailored according to their chosen field and level. It records user responses, evaluates them, and gives comprehensive feedback, making it a powerful tool for skill development and interview preparation.

System testing confirmed the success of all major features. Each function—from registration and login to job application and AI services—returned successful results during test scenarios. The overall integration between frontend, backend, and AI modules proved to be stable and reliable, validating the coherence of our implementation with the original design and functional requirements.

Ultimately, the goals outlined at the beginning of the project have been fully achieved. The system not only delivers the expected functionalities but also demonstrates extensibility, performance, and user-centric design. This milestone marks a successful step toward deploying a real-world, production-level application that serves both users and companies effectively.

Chapter 6
Conclusion & Future Work

6.1 Conclusion

The *Bridge to Work (B2W)* application is designed to support people with disabilities in finding jobs that match their skills, abilities, and special needs. Many people with disabilities face difficulties when trying to find work, and this app helps to reduce these challenges. The application includes helpful features such as an AI-powered chatbot that allows users to practice answering interview questions, making them more confident when they attend real interviews. It also offers tools to help users create strong CVs and resumes that show their experiences and talents. In addition, the app has a system that highlights (signifies) the user's special skills, making it easier for employers to see what the user can offer. One of the most important features is that the app suggests jobs that are suitable for the user's type of disability, ensuring that the work environment and tasks match their abilities. B2W also includes social media features, where users can share posts and add their achievements, making their profiles more attractive to employers. By using *Bridge to Work (B2W)*, people with disabilities have a better chance to find the right job, improve their skills, and become more independent. This project aims to create a more inclusive society where everyone has an equal opportunity to work and succeed.

6.2 Future Work

In the future, we have several ideas to improve and develop *Bridge to Work (B2W)* to serve people with disabilities even better:

- **Notifications System:**
  We plan to add a smart notifications system that will remind users about important events. For example, users will receive alerts about new job opportunities, upcoming interviews, updates on their job applications, or any changes in the job market related to their skills. These notifications will help users stay updated and not miss any important chances.

- **Chat Feature:**
  We want to add a chat feature inside the application that allows users to communicate directly with each other. This will help users build friendships, share experiences, and

give advice to one another. It can also help users discuss job opportunities, interview experiences, and provide moral support, creating a strong and helpful community.

- **Follow and Connect Feature:**
  We plan to allow users to follow each other, just like on social media platforms. This will help users build professional relationships and stay connected with people who have similar skills, interests, or experiences. It will also create a supportive network where users can learn from each other and grow together.

- **Community Building:**
  We want *B2W* to become not only a job-finding tool but also a place where people with disabilities can feel connected and supported. By adding these new features, we hope to build a friendly, helpful, and active community where everyone helps each other to grow and succeed.

- **Multi-Language Support:**
  To make the application available for more people worldwide, we plan to add multi-language support. This will allow users to use the app in their preferred language, making it easier to understand, especially for people with different native languages or reading difficulties.

- **Dark Mode:**
  For better accessibility and user comfort especially in low-light environments we intend to implement a fully functional dark mode. This feature will reduce eye strain and improve the overall user experience, particularly for users who are sensitive to bright screens.

References

- World Health Organization. (2011). *World Report on Disability.* World Health Organization. https://www.who.int/publications/i/item/9789241564182
- Lindsay, S., Cagliostro, E., Albarico, M., Mortaji, N., & Karon, L. (2018). *A Systematic Review of the Benefits of Employment for People with Disabilities.* Journal of Occupational Rehabilitation, 28(1), 1-13. https://doi.org/10.1007/s10926-017-9730-5
- Schur, L., Kruse, D., & Blanck, P. (2013). *People with Disabilities: Sidelined or Mainstreamed?* Cambridge University Press.
- Kim, H., Park, H. (2020). *AI-powered job interview systems: Opportunities and ethical challenges.* AI & Society, 35, 441–449. https://doi.org/10.1007/s00146-020-00999-7
- GeeksforGeeks. *Non-Functional Requirements in Software Engineering.* https://www.geeksforgeeks.org/non-functional-requirements-in-software-engineering/
- Jackson, J. (2023). *How AI is transforming the job market for people with disabilities.* Forbes. https://www.forbes.com/sites/forbestechcouncil/2023/05/18/how-ai-is-transforming-the-job-market-for-people-with-disabilities/
- WHO. (2023). *Assistive technology and digital inclusion for people with disabilities.* https://www.who.int/news-room/fact-sheets/detail/assistive-technology
- Indeed.com. (2023). *Indeed Accessibility Features for Job Seekers with Disabilities.* https://www.indeed.com/accessibility
- Job Accommodation Network (JAN). (2023). *Workplace accommodations for individuals with disabilities.* U.S. Department of Labor. https://askjan.org/
- WHO. (2023). *Assistive technology and digital inclusion for people with disabilities.* https://www.who.int/news-room/fact-sheets/detail/assistive-technology
- LinkedIn. *LinkedIn Accessibility Features.* https://www.linkedin.com/help/linkedin/answer/70990/accessibility-features-on-linkedin
- JobAccess. (2022). *Helping people with disability find employment.* Australian Government. https://www.jobaccess.gov.au/