**SAFEHIRE MIGRATION**

**[Migratory Worker Verification System]**

*Main Project Report*

*Submitted by*

**GAYATHRI UNNIKRISHNAN**

**Reg. No.: AJC22MCA-2045**

*In Partial Fulfillment for the Award of the Degree of*

**MASTER OF COMPUTER APPLICATIONS**

**(MCA TWO YEAR)**

[Accredited by NBA]

**APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY**



**AMAL JYOTHI COLLEGE OF ENGINEERING**

**KANJIRAPPALLY**

[Affiliated to APJ Abdul Kalam Technological University, Kerala. Approved by AICTE, Accredited by NAAC . Koovappally, Kanjirappally, Kottayam, Kerala – 686518]

# 2023-2024

## DEPARTMENT OF COMPUTER APPLICATIONS

### AMAL JYOTHI COLLEGE OF ENGINEERING

**KANJIRAPPALLY**



**CERTIFICATE**

This is to certify that the Project report, “**SAFEHIRE MIGRATION”** is the bona fide work of **GAYATHRI UNNIKRISHNAN (Regno: AJC22MCA-2045)** in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications under APJ Abdul Kalam Technological University during the year 2023-24.

**Ms. Sona Maria Sebastian Ms. Meera Rose Mathew**

**Internal Guide Coordinator**

**Rev. Fr. Dr. Rubin Thottupurathu Jose**

**Head of the Department**

**DECLARATION**

I hereby declare that the project report **“SAFEHIRE MIGRATION”** is a bonafide work done at Amal Jyothi College of Engineering, towards the partial fulfilment of the requirements for the award of the Master of Computer Applications (MCA) from APJ Abdul Kalam Technological University, during the academic year 2023-2024.

**Date: GAYATHRI UNNIKRISHNAN**

**KANJIRAPPALLY Reg: AJC22MCA-2045**

# ACKNOWLEDGEMENT

First and foremost, I thank God almighty for his eternal love and protection throughout the project. I take this opportunity to express my gratitude to all who helped me in completing this project successfully. It has been said that gratitude is the memory of the heart. I wish to express my sincere gratitude to our Manager **Rev. Fr. Dr. Mathew Paikatt** and Principal **Dr. Lillykutty Jacob** for providing good faculty for guidance.

I owe a great depth of gratitude towards our Head of the Department **Rev.Fr.Dr. Rubin Thottupurathu Jose** for helping us. I extend my whole hearted thanks to the project coordinator **Ms. Meera Rose Mathew** for her valuable suggestions and for overwhelming concern and guidance from the beginning to the end of the project. I would also express sincere gratitude to my guide **Ms. Sona Maria Sebastian** for her inspiration and helping hand.

I thank our beloved teachers for their cooperation and suggestions that helped me throughout the project. I express my thanks to all my friends and classmates for their interest, dedication, and encouragement shown towards the project. I convey my hearty thanks to my family for the moral support, suggestions, and encouragement to make this venture a success.

GAYATHRI UNNIKRISHNAN

# ABSTRACT

The "SafeHire Migration" project aims to address the critical challenges faced by employers in efficiently managing migratory workers. Recognizing the importance of a secure hiring process, the platform incorporates robust verification measures to establish the credibility of workers. The system further facilitates streamlined booking management, centralizes document handling, and establishes a communication hub. This comprehensive solution empowers employers to engage seamlessly with verified workers, ultimately creating a safer and more productive working environment.

Modules:

Agent

* Agents create their profile
* Create profile and personal information of the workers
* Add another worker
* Track workers current status

Police

* See the profile of agents and workers
* Document Verification
* They can approve or reject
* Advanced digital Authentication of documents
* Incident reporting

Employer

* Only see the verified workers
* Contact workers by getting the contact details of agent
* Do payment for Booking worker
* Submit Job Opportunities
* Pay salary for workers

Admin

* View the profile of employers, agents, police
* Varify the users
* View all the worker details
* Add Work category
* Payment Management of employers
* Track workers current status

Technology Stack

* Frontend: HTML, CSS, JS, bootstrap for the user interface.
* Backend: Django, SQLite.
* Payment Integration: RazorPay

**CONTENT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SL. NO** | | **TOPIC** | **PAGE NO** | |
| **1** | | **INTRODUCTION** | 1 | |
| **1.1** | | **PROJECT OVERVIEW** | 2 | |
| **1.2** | | **PROJECT SPECIFICATION** | 2 | |
| **2** | | **SYSTEM STUDY** | 4 | |
| **2.1** | | **INTRODUCTION** | 5 | |
| **2.2** | | **EXISTING SYSTEM** | 5 | |
| **2.3** | | **DRAWBACKS OF EXISTING SYSTEM** | 6 | |
| **2.4** | | **PROPOSED SYSTEM** | 6 | |
| **2.5** | | **ADVANTAGES OF PROPOSED SYSTEM** | 6 | |
| **3** | | **REQUIREMENT ANALYSIS** | 9 | |
| **3.1** | | **FEASIBILITY STUDY** | 10 | |
| **3.1.1** | | **ECONOMICAL FEASIBILITY** | 10 | |
| **3.1.2** | | **TECHNICAL FEASIBILITY** | 10 | |
| **3.1.3** | | **BEHAVIORAL FEASIBILITY** | 11 | |
| **3.1.4** | | **FEASIBILITY STUDY QUESTIONNAIRE** | 11 | |
| **3.2** | | **SYSTEM SPECIFICATION** | 14 | |
| **3.2.1** | | **HARDWARE SPECIFICATION** | 14 | |
| **3.2.2** | | **SOFTWARE SPECIFICATION** | 14 | |
| **3.3** | | **SOFTWARE DESCRIPTION** | 15 | |
| **3.3.1** | | **PYTHON-DJANGO** | 15 | |
| **3.3.2** | | **SQLITE** | 15 | |
| **4** | | **SYSTEM DESIGN** | 16 | |
| **4.1** | | **INTRODUCTION** | 17 | |
| **4.2** | | **UML DIAGRAM** | 17 | |
| **4.2.1** | | **USE CASE DIAGRAM** | 18 | |
| **4.2.2** | | **SEQUENCE DIAGRAM** | 20 | |
| **4.2.3** | | **STATE CHART DIAGRAM** | 22 | |
| **4.2.4** | | **ACTIVITY DIAGRAM** | 23 | |
| **4.2.5** | | **CLASS DIAGRAM** | 24 | |
| **4.2.6** | | **OBJECT DIAGRAM** | 25 | |
| **4.2.7** | | **COMPONENT DIAGRAM** | 26 | |
| **4.2.8** | | **DEPLOYMENT DIAGRAM** | 27 | |
| **4.2.9** | | **COLLABORATION DIAGRAM** | 29 | |
| **4.3** | | **USER INTERFACE DESIGN USING FIGMA** | 30 | |
| **4.4** | | **DATABASE DESIGN** | 32 | |
| **5** | | **SYSTEM TESTING** | 38 | |
| **5.1** | | **INTRODUCTION** | 39 | |
| **5.2** | | **TEST PLAN** | 39 | |
| **5.2.1** | **UNIT TESTING** | | 39 |
| **5.2.2** | **INTEGRATION TESTING** | | 40 |
| **5.2.3** | **VALIDATION TESTING** | | 40 |
| **5.2.4** | **USER ACCEPTANCE TESTING** | | 41 |
| **5.2.5** | **AUTOMATION TESTING** | | 41 |
| **5.2.6** | **SELENIUM TESTING** | | 41 |
| **6** | **IMPLEMENTATION** | | 52 |
| **6.1** | **INTRODUCTION** | | 53 |
| **6.2** | **IMPLEMENTATION PROCEDURE** | | 53 |
| **6.2.1** | **USER TRAINING** | | 54 |
| **6.2.2** | **TRAINING ON APPLICATION SOFTWARE** | | 54 |
| **6.2.3** | **SYSTEM MAINTENANCE** | | 54 |
| **7** | **CONCLUSION & FUTURE SCOPE** | | 55 |
| **7.1** | **CONCLUSION** | | 56 |
| **7.2** | **FUTURE SCOPE** | | 56 |
| **8** | **BIBLIOGRAPHY** | | 57 |
| **9** | **APPENDIX** | | 59 |
| **9.1** | **SAMPLE CODE** | | 60 |
| **9.2** | **SCREEN SHOTS** | | 90 |

## List of Abbreviation

IDE - Integrated Development Environment

HTML - Hyper Text Markup Language

CSS - Cascading Style Sheet

SQL - Structured Query Language

UML - Unified Modelling Language

JS - JavaScript

URL - Uniform Resource Locator

PY – Python

1NF - First Normal Form

2NF - Second Normal Form

3NF - Third Normal Form

PK - Primary Key

FK - Foreign Key

# CHAPTER 1

# INTRODUCTION

### PROJECT OVERVIEW

The SafeHire Migration System is designed to address the complex challenges associated with the recruitment and employment of migrant workers while ensuring compliance with labor laws and immigration regulations. The primary objectives of the project are:

To create a secure, transparent, and efficient digital platform for the recruitment and employment of migrant workers.

To streamline the verification of worker documentation to prevent fraud and ensure compliance.

To facilitate ethical and lawful labor migration practices that protect the rights and welfare of migrant workers.

To enhance communication and accountability among agents, employers, workers, and government authorities involved in the migration process

### PROJECT SPECIFICATION

The proposed "SafeHire Migration System" is designed to address several critical aspects and challenges related to the recruitment and employment of migrant workers. Its scope is comprehensive and aims to provide solutions to various stakeholders involved in the migration and employment process. It also helps to avoid crimes done by migrant workers.

### The system consists of 4 actors. They are:

Agent

* Agents create their profile
* Create profile and personal information of the workers
* Add another worker
* Track workers current status

Police

* See the profile of agents and workers
* Document Verification
* They can approve or reject
* Advanced digital Authentication of documents
* Incident reporting

Employer

* Only see the verified workers
* Contact workers by getting the contact details of agent
* Do payment for Booking worker
* Submit Job Opportunities
* Pay salary for workers

Admin

* View the profile of employers, agents, police
* Varify the users
* View all the worker details
* Add Work category
* Payment Management of employers
* Track workers current status

Technology Stack

* **Frontend:** HTML, CSS, JS, bootstrap for the user interface.
* **Backend:** Django, SQLite.
* **Payment Integration:** RazorPay

# CHAPTER 2

# SYSTEM STUDY

### INTRODUCTION

A critical step in system development is system analysis, which involves gathering and analyzing data to identify problems and provide solutions. During this phase, effective communication between system users and developers is essential. Any system development process should always begin with a system analysis. The system analyst performs the role of an investigator by carefully examining the current system's performance. Identification of the system's inputs and outputs as well as a connection between its processes and organizational results are all included.

A range of techniques, including surveys and interviews, are used to acquire information. Understanding the system's operation, locating problem areas, and suggesting fixes to the issues the business is having are the objectives. The designer assumes the role of issue solver, and the suggested fixes are rigorously compared with the current system. The user is given the chance to accept or reject the advice after the best choice has been chosen. The procedure continues until the user is pleased after the idea has been evaluated in light of their comments.

The process of acquiring and analyzing data for future system studies is called a preliminary study. Conducting a thorough preliminary study is essential to ensure the success of a system development project.

### EXISTING SYSTEM

The existing manual paper-based system for managing migratory workers involves a cumbersome and time-consuming process. Employers, agents, and police personnel rely on physical documents and face-to-face interactions for profile creation, verification, and worker bookings. This manual approach is prone to errors, delays, and inefficiencies, leading to challenges such as lost or misplaced documents, difficulty in tracking worker profiles, and increased administrative overhead. The lack of a centralized system makes it challenging to verify worker credentials efficiently. Additionally, communication between stakeholders is hindered, and the absence of an integrated payment system further complicates the booking process. Overall, the manual system lacks scalability, transparency, and the security measures necessary for effective migratory worker management in today's dynamic and fast-paced environment.

**2.2.1 NATURAL SYSTEM STUDIED**

The "SafeHire Migration" project is an innovative solution revolutionizing the management of migratory workers. Through ecosystem analysis, it identifies key components and explores their interdependencies, prioritizing environmental impact and sustainable practices. The project addresses climate and weather patterns, ensuring worker safety and committing to biodiversity conservation. Geographical considerations involve mapping migration routes and cultural sensitivity, accommodating diverse practices. Legal compliance and adherence to regulatory frameworks demonstrate ethical environmental practices. The project engages in positive ecotourism, empowering local communities, and integrates monitoring, reporting, and educational initiatives for a holistic impact on the natural environment. In summary, "SafeHire Migration" blends efficiency in worker management with a strong commitment to environmental sustainability and cultural inclusivity.

**2.2.2 DESIGNED SYSTEM STUDIED**

There is no existing system which mirror my system .By incorporating robust verification measures, real-time tracking, streamlined booking management, and a centralized document handling system, the platform enhances the hiring experience for employers. The focus on transparency, reliability, and safety in the hiring process, coupled with cultural sensitivity and adherence to legal and environmental regulations, positions the project as a holistic solution. The integration of monitoring, reporting, and educational initiatives further contributes to creating a safer, more accountable, and environmentally conscious working environment for both employers and migratory workers.

### DRAWBACKS OF EXISTING SYSTEM

* Inefficiency and Time-Consuming Processes
* Error-Prone and Unreliable Data Handling
* Lack of Centralized Information
* Limited Communication and Payment Integration
* Lacks scalability and adaptability
* Pose security risks with physical document handling
* Lack comprehensive user verification

### PROPOSED SYSTEM

The proposed "SafeHire Migration" system is a cutting-edge solution designed to address the complex challenges faced by employers in managing migratory workers efficiently. By emphasizing the significance of a secure hiring process, the platform incorporates robust verification measures to establish the credibility of workers, ensuring a trustworthy and reliable workforce. The system introduces a seamless and streamlined booking management process, centralizing document handling, and establishing a communication hub among stakeholders, including agents, police, employers, and administrators. With distinct modules catering to the unique roles of agents, police personnel, employers, and administrators, the platform offers tailored functionalities to enhance user experience and operational efficiency. Through the integration of Django and SQLite for the backend and HTML, CSS, JS, and Bootstrap for the frontend, the system ensures a user-friendly interface. Additionally, the inclusion of RazorPay facilitates secure payment transactions. This comprehensive solution empowers employers to engage seamlessly with verified workers, ultimately fostering a safer and more productive working environment.

### ADVANTAGES OF PROPOSED SYSTEM

* **Secure Hiring Process:**

The platform prioritizes a secure hiring process by implementing robust verification measures, ensuring the credibility of migratory workers. This feature enhances the overall integrity of the workforce and mitigates the risk of hiring unverified individuals.

* **Efficient Booking Management:**

The system introduces a streamlined booking management process, allowing employers to request and manage workers seamlessly. This efficiency reduces administrative overhead, accelerates the hiring timeline, and enhances overall productivity.

* **Centralized Document Handling:**

"SafeHire Migration" centralizes document handling, providing a secure repository for important documents such as identification, permits, and certifications. This centralized approach ensures easy accessibility, reduces the risk of document loss, and enhances overall document management efficiency.

* **User-Friendly Interface:**

The system employs a user-friendly interface with HTML, CSS, JS, and Bootstrap, ensuring a positive user experience. This accessibility is crucial for users with varying technical expertise, promoting widespread adoption and ease of use.

* **Role-Based Modules:**

Tailored modules for agents, police, employers, and administrators ensure that each user role has access to functionalities relevant to their responsibilities. This modular approach optimizes the user experience, making the platform more intuitive and efficient for different stakeholders.

* **Scalability and Adaptability:**

Built on Django and SQLite for the backend, the system is designed to be scalable and adaptable to changing requirements. This flexibility allows the platform to evolve with the dynamic nature of migratory worker management, ensuring long-term viability and relevance.

* **Secure Payment Integration:**

The inclusion of RazorPay for payment transactions enhances the security of financial transactions, providing a reliable and trusted payment gateway. This feature ensures that employers can make payments for worker bookings with confidence.

* **Comprehensive User Verification:**

The system incorporates a thorough user verification process, including the verification of agents, police personnel, and workers. This comprehensive approach contributes to a trustworthy and reliable network of users within the platform.

* **Enhanced Work Environment:**

By facilitating a secure and efficient hiring process, the "SafeHire Migration" system contributes to the creation of a safer and more productive work environment. Employers can confidently engage with verified workers, fostering a positive atmosphere for both employers and employees.

# CHAPTER 3

# REQUIREMENT ANALYSIS

## FEASIBILITY STUDY

When complex problem and opportunities are to be defined, it is generally desirable to conduct a preliminary investigation called a feasibility study. A feasibility study is conduct to obtain an overview of the problem and to roughly assess whether feasible solution exists prior to committing substantial resources to a project.

### Economical Feasibility

A systems development project may be regarded as economically feasible or ‘good value’ to the organization if its anticipated benefits outweigh its estimated costs. However, many of the organizational benefits arising from record keeping projects are intangible and may be hard to quantify. In contrasts, many development costs are easier to identify.

The economic feasibility of the SafeHire Migration system is highly promising. While initial development costs are significant, the system's benefits in terms of improved labor migration oversight, ethical hiring practices, and reduced labor disputes are substantial. The return on investment (ROI) is expected to be favorable, with a relatively short payback period. Additionally, the system's scalability and potential for long-term cost savings make it economically viable. However, ongoing operational costs, regulatory compliance, and the need for effective risk management must be carefully considered to ensure continued economic success.

### Technical Feasibility

Technical feasibility is a study of function , performance and constraints that may affect the ability to achieve an acceptable system. During technical analysis, the analyst evaluates the technical merits of the system , at the same time collecting additional information about performance, reliability, maintainability and productivity . Technical feasibility is frequently the most difficult areas to assess. The main technical issue raised during feasibility is the existence of necessary technology and whether the proposed equipment has the capacity to hold required data. The technical guarantee of accuracy, reliability, ease and data were also investigated.

The technical feasibility of the SafeHire Migration system is robust. With advancements in web-based technologies, secure data encryption, and biometric authentication, the system can be efficiently developed and deployed. It can leverage existing databases and integration capabilities with government agencies. The scalability of cloud-based infrastructure ensures it can handle increasing data volumes. While challenges like user training and ongoing technical support exist, the overall technical foundation is sound, making the implementation of this innovative labor migration system both feasible and efficient.

### Behavioral Feasibility

Operation feasibility is a measure of how people feel about the system. Operational Feasibility criteria measure the urgency of the problem or the acceptability of a solution. Operational Feasibility is dependent upon determining human resources for the project . It refers to projecting whether the system will operate and be used once it is installed . If the ultimate users are comfortable with the present system and they see no problem with its continuance, then resistance to its operation will be zero. Behaviorally also the proposed system is feasible. A particular application may be technically and but may fail to produce the forecasted benefits . For the system, it is not necessary that the user must be a computer expert, but any computer operator given a little bit of knowledge and training can easily operate.

**3.1.4 Feasibility Study Questionnaire**

**Project Overview?**

The project aims to create an automated workforce management system that streamlines the profiles of agents, workers, and police, ensuring efficient communication, secure document verification, and transparent transactions between agents and employers.

**To what extent the system is proposed for?**

The system is proposed to cover end-to-end workforce management, from profile creation and document verification to communication and financial transactions, catering to the needs of agents, police, and employers.

**Specify the Viewers/Public which is to be involved in the System?**

The system is designed to involve agents, police officers, and employers as primary users. Additionally, administrators will have access to oversee and manage the entire platform.

**List the Modules included in your System?**

a. User Management

b. Profile Creation and Verification

d. Payment Integration

e. Reporting and Analytics

i. Multi-Level Authentication

j. Job Opportunity Submission

**Identify the users in your project?**

a. Agents

b. Police Officers

c. Employers

d. Administrators

**Who owns the system?**

Administrators

**System is related to which firm/industry/organization?**

The system is related to the workforce management sector, providing a solution for labor agencies, law enforcement, and employers seeking a reliable platform for hiring workers.

**Details of the person that you have contacted for data collection?**

Contacted Suresh ASI for data collection regarding the existing manual system and user requirements.

**Is the Required Technology Available?**

Yes, the required technology for the SafeHire Migration system is available. It leverages web-based platforms, which are widely accessible and supported by various devices. Additionally, the system incorporates features like document verification, face recognition authentication, and data encryption, all of which rely on established technologies. The integration with relevant government agencies and communication tools can be facilitated using existing technologies. Overall, the technology needed to develop and operate the SafeHire Migration system is readily available and accessible.

**Can the project Scale to Handle User Load?**

The scalability of the SafeHire Migration project is contingent on several factors including the underlying technology stack, architecture, and infrastructure. To ensure it can handle increasing user loads, the system should be designed with scalability in mind from the outset. Scalability measures such as load balancing, database sharding, and cloud-based hosting can be implemented to accommodate growing user numbers. Regular performance testing and capacity planning will be crucial to identify and address potential bottlenecks. With proper planning and the use of scalable technologies, the SafeHire Migration project can be designed to handle a significant user load as needed.

**Integration with Backend Systems?**

The integration of the SafeHire Migration system with backend systems is a fundamental aspect of its technical feasibility. To ensure seamless data exchange and functionality, the system should be designed to integrate with various backend systems, such as government databases for document verification, payment gateways for financial transactions, and communication systems for notifications.

APIs (Application Programming Interfaces) and web services can facilitate these integrations, allowing the SafeHire Migration platform to securely communicate and share data with external systems. The choice of integration technologies and protocols should align with industry standards and security best practices.

**Security and Data Protection?**

Security and data protection in the SafeHire Migration system are of paramount importance. The system employs state-of-the-art encryption for data both in transit and at rest, ensuring that sensitive information remains confidential. Robust user authentication and access control mechanisms prevent unauthorized access, and comprehensive audit trails track user activities for security monitoring. Regular security audits, compliance with data privacy regulations, and employee training further enhance system security. Document storage is secure, and disaster recovery measures are in place to ensure data availability in case of emergencies. With these measures, the system safeguards user data and ensures a high level of security throughout its operations.

**Support for Different Devices and OS Versions?**

The SafeHire Migration system is designed with flexibility in mind, offering support for a wide range of devices and operating system (OS) versions. It includes responsive web design, making it accessible on various devices such as desktop computers, laptops, tablets, and smartphones. The system is compatible with major web browsers, ensuring functionality across different platforms and OS versions, including Windows, macOS, Android, and iOS. This approach maximizes accessibility for users, agents, and administrators, allowing them to access the system seamlessly regardless of their preferred device or OS.

## SYSTEM SPECIFICATION

### Hardware Specification

Processor - AMD Ryzen 5

RAM - 4GB

Hard disk - 256GB

### Software Specification

Front End - HTML/CSS

Back End - Python Django

Database - Sqlite3

Client on PC - Windows 7 and above.

Technologies used - JS, HTML5, AJAX, J Query, PHP, CSS

## SOFTWARE DESCRIPTION

**3.3.1 DJANGO FRAMEWORK:**

The Django Framework is a popular and robust web framework for Python developers. It is

revered for its simplicity, clean code, and rapid development capabilities. Django is built on the

Model-Template-Views (MTV) architectural pattern, which shares similarities with the ModelView-Controller (MVC) pattern used in other frameworks. One of its standout features is the Object-Relational Mapping (ORM) system, simplifying database interactions by representing database tables as Python objects. This abstraction eliminates the need for writing raw SQL queries, making database operations more straightforward.

Django also offers a built-in administrative interface, making content management a breeze. Its

URL routing system allows developers to define clean and user-friendly URLs for their web

applications. Furthermore, Django provides comprehensive support for form handling, data

validation, and user authentication, reducing the complexity of common web development tasks.

Security is a top priority in Django, with built-in protections against common web vulnerabilities

like Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF). With a thriving

community and an array of reusable packages, Django is a versatile choice for web development projects of varying sizes and complexities.

**3.3.2 SQLite:**

SQLite is a lightweight, self-contained, and serverless relational database management system. Unlike traditional databases, it does not require a separate server but is embedded directly within the application. One of its standout features is the self-contained nature of SQLite databases; the entire database is stored in a single file, simplifying management, backups, and transfers. Despite its lightweight design, SQLite is ACID-compliant, meaning it ensures data integrity by supporting transactions, making it suitable for multi-user environments.

SQLite is cross-platform and compatible with various operating systems, making it a versatile choice for applications targeting different platforms. Its small code size and minimal resource usage make it suitable for resource-constrained environments, such as mobile devices. SQLite is known for its speed and efficiency, especially in read-heavy workloads. It is commonly used in mobile applications, desktop applications, and embedded systems, where a full-fledged database server might be excessive, and portability is essential.

# CHAPTER 4

# SYSTEM DESIGN

* 1. **INTRODUCTION**

Design is the first step into the development phase for any engineered product or system. Design is a creative process. A good design is the key to effective system. The term “design” is defined as “the process of applying various techniques and principles for the purpose of defining a process or a system in sufficient detail to permit its physical realization”. It may be defined as a process of applying various techniques and principles for the purpose of defining a device, a process or a system in sufficient detail to permit its physical realization. Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm that is used. The system design develops the architectural detail required to build a system or product. As in the case of any systematic approach, this software too has undergone the best possible design phase fine tuning all efficiency, performance and accuracy levels. The design phase is a transition from a user-oriented document to a document to the programmers or database personnel. System design goes through two phases of development: Logical and Physical Design.

## UML DIAGRAM

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997. UML is a pictorial language used to make software blueprints. UML can be described as a general-purpose visual modeling language to visualize, specify, construct, and document software system. UML is not a programming language but tools can be used to generate code in various languages using UML diagrams. UML has a direct relation with object-oriented analysis and design. All the elements, relationships are used to make a complete UML diagram and the diagram represents a system. The visual effect of the UML diagram is the most important part of the entire process. All the other elements are used to make it complete. They enhance communication by providing a visual representation that is easily understood by technical and non-technical stakeholders alike. They aid in analysis and design by capturing system requirements, relationships, and interactions. UML diagrams also facilitate software development by serving as a blueprint for developers to implement and test systems.

UML includes the following nine diagrams.

• Use case diagram

• Sequence diagram

• State chart diagram

• Activity diagram

• Class diagram

• Object diagram

• Component diagram

• Deployment diagram

## USE CASE DIAGRAM

A use case diagram is a graphic depiction of the interactions among the elements of a system. A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. In this context, the term "system" refers to something being developed or operated, such as a mail-order product sales and service Web site. Use case diagrams are employed in UML (Unified Modeling Language), a standard notation for the modeling of real-world objects and systems. System objectives can include planning overall requirements, validating a hardware design, testing and debugging a software product under development, creating an online help reference, or performing a consumer-service-oriented task. For example, use cases in a product sales environment would include item ordering, catalog updating, payment processing, and customer relations. A use case diagram contains four components.

• The boundary, which defines the system of interest in relation to the world around it.

• The actors, usually individuals involved with the system defined according to their roles.

• The use cases, which are the specific roles are played by the actors within and around the system.

• The relationships between and among the actors and the use cases.

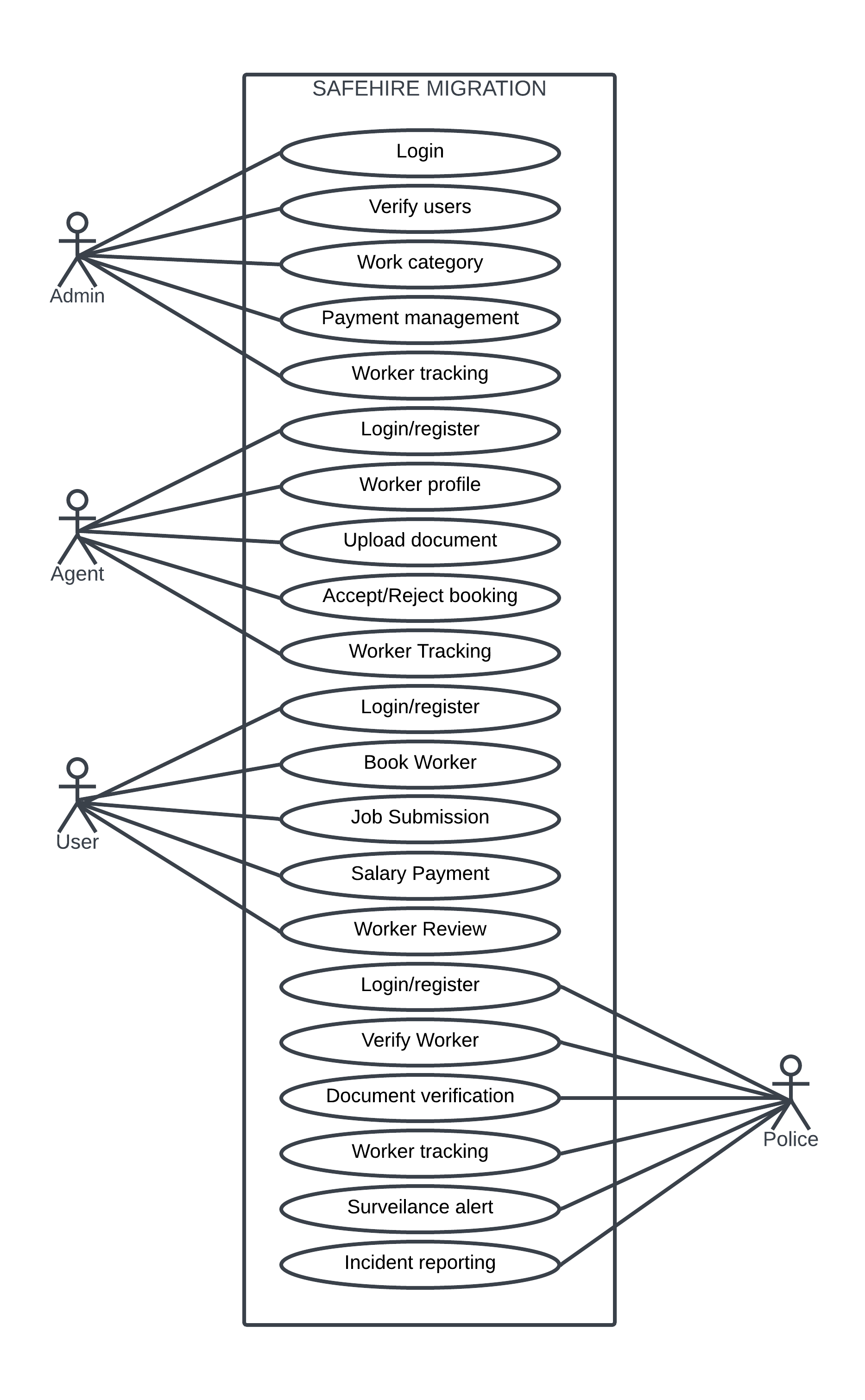


Figure 1:Use Case Diagram

## SEQUENCE DIAGRAM

The chronological order of interactions between various system components is shown in a

sequence diagram, a form of interaction diagram. It demonstrates how several things

communicate with one another over the course of a series of messages. These images are

sometimes referred to as event scenarios or event scenarios diagrams. In software engineering,

sequence diagrams are frequently used to describe and comprehend the needs of both new and

old systems. They support the visualization of object control relationships and the detection of

systemic issues.

Sequence Diagram Notations –

**i. Actors** - In UML, a role that interacts with the system and its objects is represented by an

actor. Actors frequently exist outside of the system that the UML diagram is intended to

portray. Actors can play a variety of roles, including those of external topics or human

users. A stick person notation is used in UML diagrams to represent actors. Depending on

the situation that is being modelled; a sequence diagram may have more than one actor.

**ii. Lifelines** - A lifeline in a sequence diagram is a vertical dashed line that represents the

lifespan of an object participating in the interaction. Each lifeline represents an individual

participant in the sequence of events and is labeled with the name of the participant. The

lifeline shows the timeline of events for the participant and is drawn as a vertical line

extending from the participant's activation point to its deactivation point.

**iii. Messages** - Messages are a key component of sequence diagrams, representing the

interactions and communication between objects or components in a system. They can be

categorized into synchronous and asynchronous messages, create, and delete messages,

self-messages, reply messages, found messages, and lost messages. Guards are also used

to model conditions and restrictions on message flow.

**iv. Guards** - Guards in UML are used to model conditions and are employed to restrict the

flow of messages when a certain condition is met. This feature is essential for letting

software developers know about any constraints or limitations associated with a system

or a particular process.

Uses of sequence diagram –

• Modeling and visualizing the logic of complex functions, operations, or procedures.

• Showing details of UML use case diagrams.

• Understanding the detailed functionality of current or future systems.

• Visualizing how messages and tasks move between objects or components in a system.

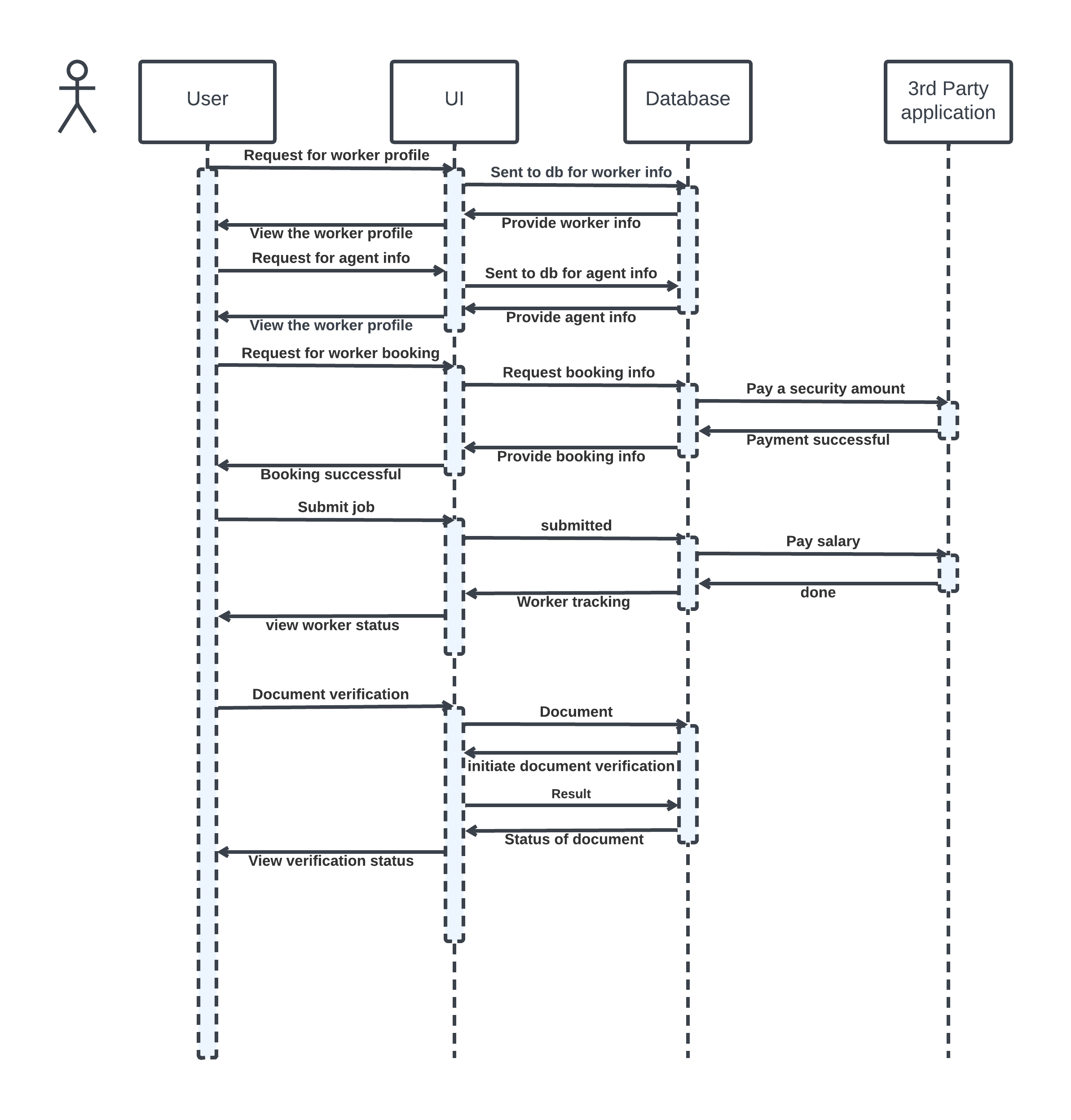


Figure 2:Sequence Diagram

## 4.2.2 State Chart Diagram

A state diagram is a visual representation, often created using the Unified Modeling Language (UML), that shows the different states that an object can exist in and how it can transition between those states. It is also referred to as a state machine diagram or state chart diagram. The State Chart Diagram is a behavioral diagram in UML that describes the behavior of a system or object over time. It includes various elements such as:

• Initial State - This state represents the starting point of the system or object and is denoted by a solid black circle.

• State - This element describes the current state of the system or object at a specific point in time and is represented by a rectangle with rounded corners.

• Transition - This element shows the movement of the system or object from one state to another and is represented by an arrow.

• Event and Action - An event is a trigger that causes a transition to occur, and an action is the behavior or effect of the transition.

• Signal - A message or trigger caused by an event that is sent to a state, causing a transition to occur.

• Final State - The State Chart Diagram ends with a Final State element, which is represented by a solid black circle with a dot inside. It indicates that the behavior of the system or object has completed.

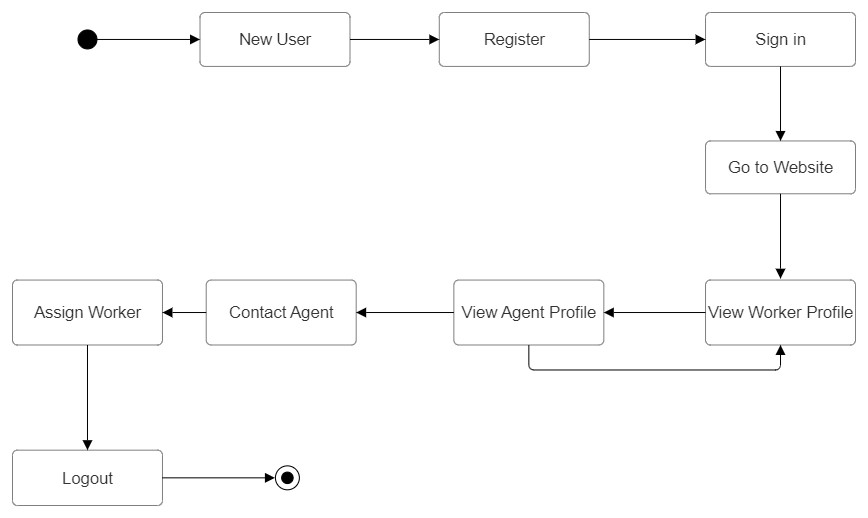


Figure 3:State Chart Diagram

## Activity Diagram

Activity diagrams depict how different levels of abstraction of activities are linked to provide a service. Typically, an event should be completed by some activities, particularly when the activity is intended to do multiple separate goals that need coordination. Another typical requirement is how the events in a single use case interact with one another, particularly in use cases where operations may overlap and require coordination. It may also be used to show how a collection of interrelated use cases interacts to reflect business operations. In an activity diagram, activities are represented as rounded rectangles, and arrows indicate the flow of control between activities. Decision points are depicted as diamonds, where different paths or branches can be taken based on conditions or criteria. Forks and joins represent parallel or concurrent execution of activities. Swim lanes can also be used to group related activities performed by specific actors or roles. Activity diagrams are valuable for modelling and analysing complex processes or workflows. They help to visualize the sequence of steps, dependencies, and decision points within a system, providing a clear understanding of how the process operates. This aids in identifying potential bottlenecks, inefficiencies, or areas for optimization.

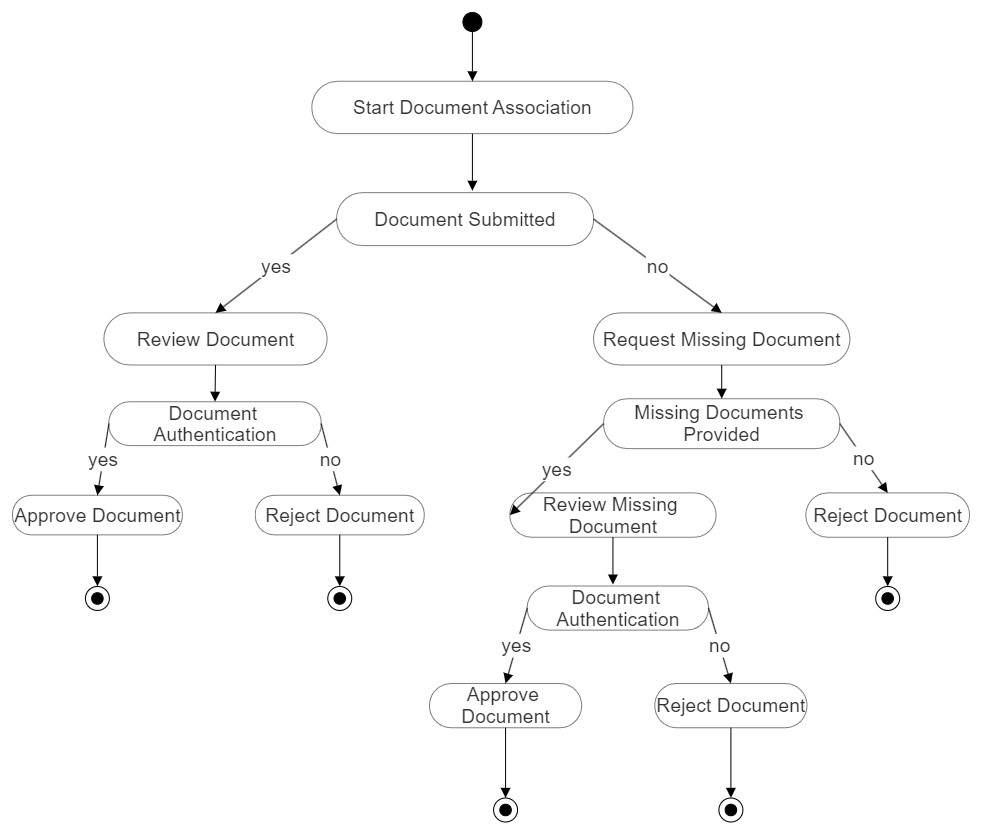


Figure 4:Activity Diagram

## Class Diagram

Class diagram is a static diagram. It represents the static view of the application. Class diagrams are useful for visualizing, describing, and documenting various system components as well as for writing executable code for software applications. A class diagram describes the constraints imposed on the system together with the properties and operations of a class. The only UML diagrams that can be directly converted into object-oriented languages are class diagrams, which are extensively utilized in the designing of object-oriented systems. An assortment of classes, interfaces, affiliations, partnerships, and limitations are displayed in a class diagram. It also goes by the name "structural diagram”. Class diagrams are widely used in software development to design and document the structure of object-oriented systems. They help to identify and organize classes, their attributes, and behaviours, providing a blueprint for software implementation. Class diagrams also assist in understanding and communicating the relationships and interactions between different classes. Key elements in a class diagram include associations, which represent relationships between classes, such as one-to-one, one-to-many, or many-to-many associations. Inheritance is depicted using arrows to indicate the "is-a" relationship, where one class inherits attributes and methods from another. Aggregation and composition relationships are used to represent whole-part relationships between classes. Class diagrams are an essential tool for visualizing the static structure of a system and capturing its essential elements. They aid in system design, modelling, and documentation, allowing software developers and stakeholders to understand the system's architecture, dependencies, and responsibilities of different classes.

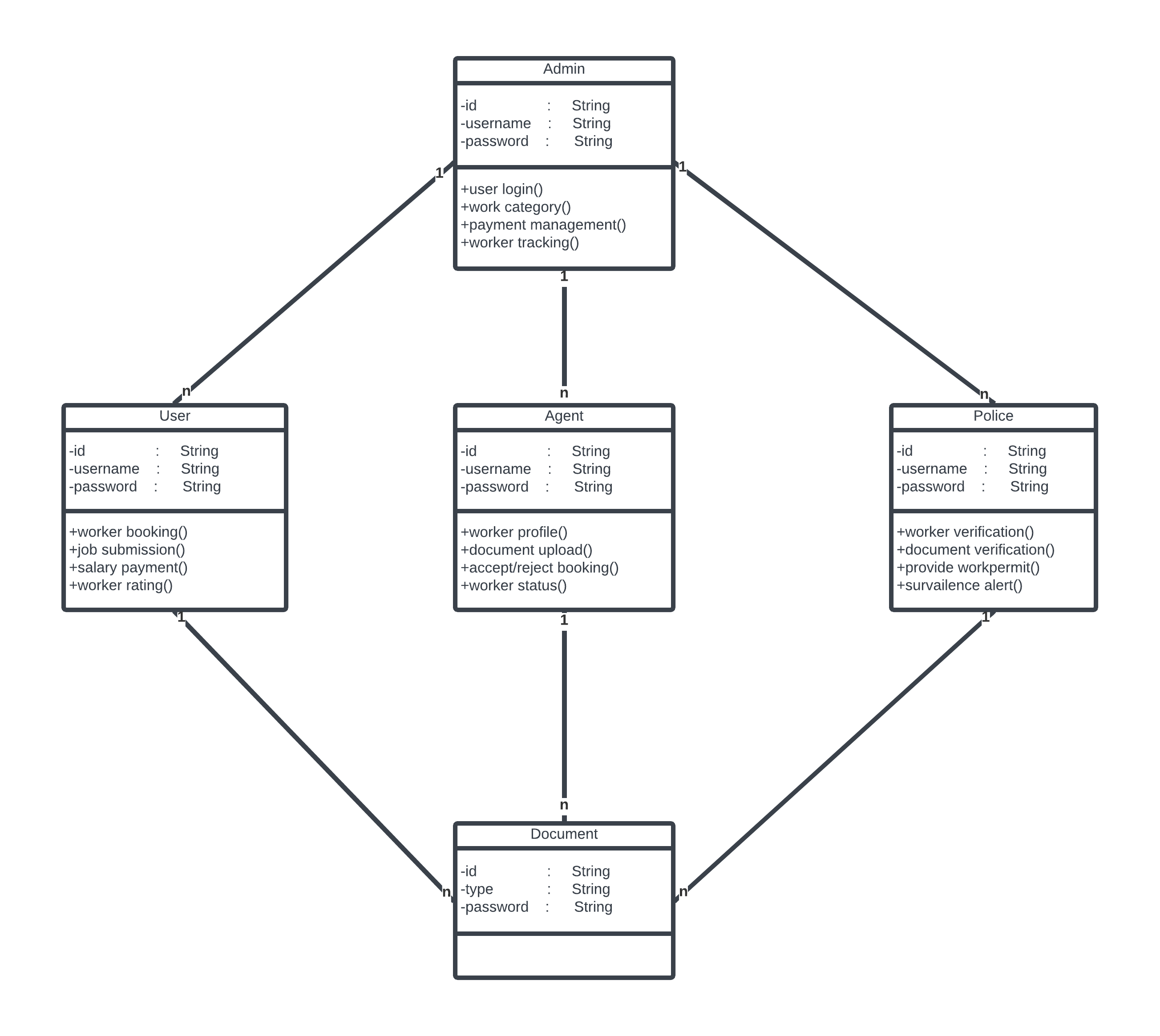


Figure 5:Class Diagram

## Object Diagram

Class diagrams and object diagrams are closely related in object-oriented modeling. Object diagrams are instances of class diagrams, which represent a snapshot of the system at a given moment in time. Both types of diagrams use the same concepts and notation to represent the structure of a system. While class diagrams are used to model the structure of the system, including its classes, attributes, and methods, object diagrams represent a group of objects and their connections at a specific point in time. An object diagram is a type of structural diagram in UML that shows instances of classes and their relationships. The main components of an object diagram include: • Object: An object is an instance of a class that represents a specific entity in the system. It is represented as a rectangle with the object name inside. • Class: A class is a blueprint or template for creating objects that defines its attributes and methods. It is represented as a rectangle with three compartments for the class name, attributes, and methods. • Link: A link is a relationship between two objects that represents a connection or association. It is represented as a line connecting two objects with optional labels. • Attribute: An attribute is a property or characteristic of an object that describes its state. It is represented as a name-value pair inside the object rectangle. • Value: A value is a specific instance or setting of an attribute. It is represented as a value inside the attribute name-value pair. • Operation: An operation is a behavior or action that an object can perform. It is represented as a method name inside the class rectangle. • Multiplicity: Multiplicity represents the number of instances of a class that can be associated with another class. Object diagrams help to visualize the relationships between objects and their attributes in a system. They are useful for understanding the behavior of a system at a specific point in time and for identifying potential issues or inefficiencies in the system.

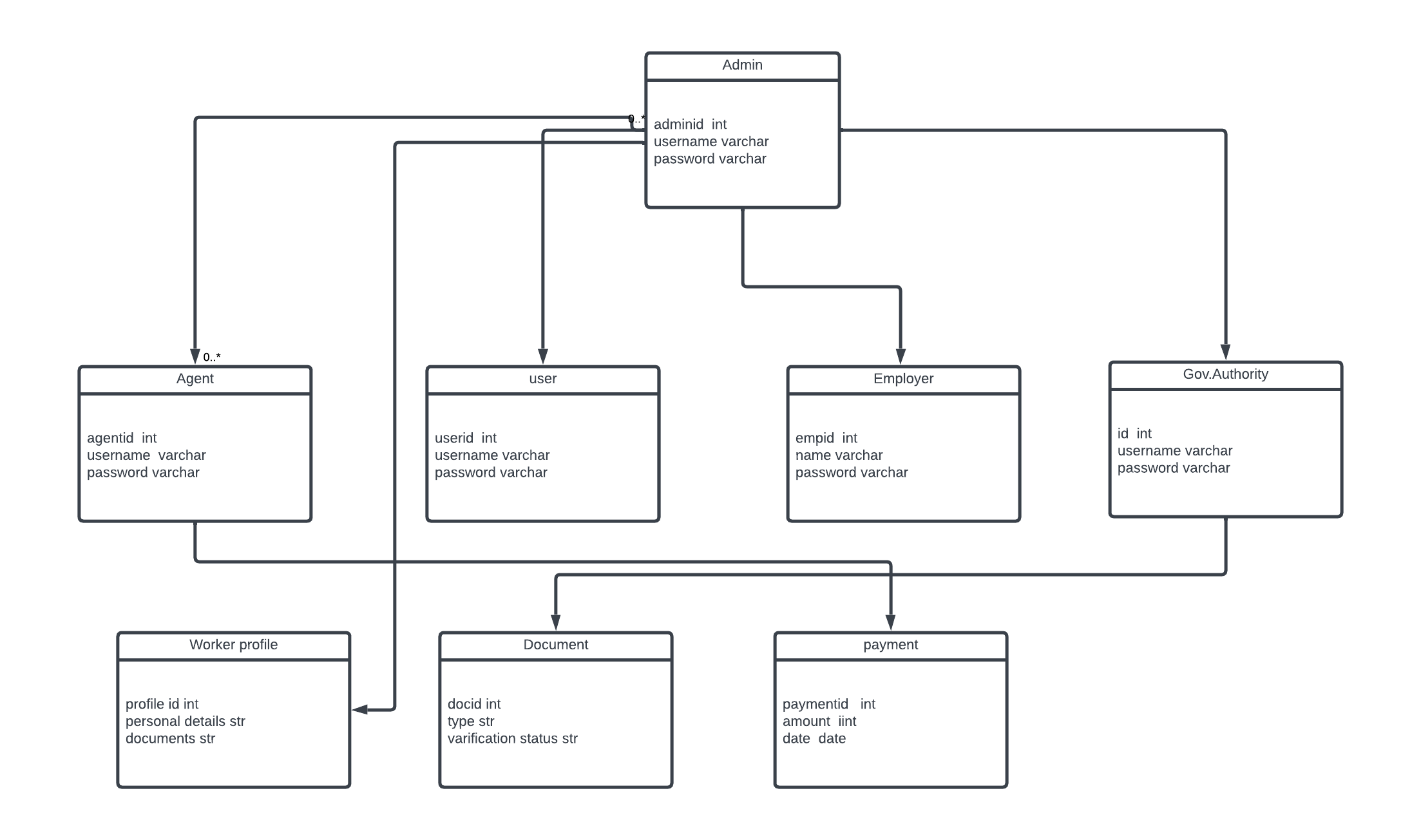


Figure 6:Object Diagram

## Component Diagram

A component diagram in UML illustrates how various components are interconnected to create larger components or software systems. It is an effective tool for representing the structure of complex systems with multiple components. By using component diagrams, developers can easily visualize the internal structure of a software system and understand how different components work together to accomplish a specific task. Its key components include:

• Component: A modular and encapsulated unit of functionality in a system that offers interfaces to interact with other components. It is represented as a rectangle with the component name inside.

• Interface: A contract between a component and its environment or other components, specifying a set of methods that can be used by other components. It is represented as a circle with the interface name inside.

• Port: A point of interaction between a component and its environment or other components. It is represented as a small square on the boundary of a component.

• Connector: A link between two components that enables communication or data exchange. It is represented as a line with optional adornments and labels.

• Dependency: A relationship between two components where one component depends on another for its implementation or functionality. It is represented as a dashed line with an arrowhead pointing from the dependent component to the independent component.

• Association: A relationship between two components that represents a connection or link. It is represented as a line connecting two components with optional directionality, multiplicity, and role names.

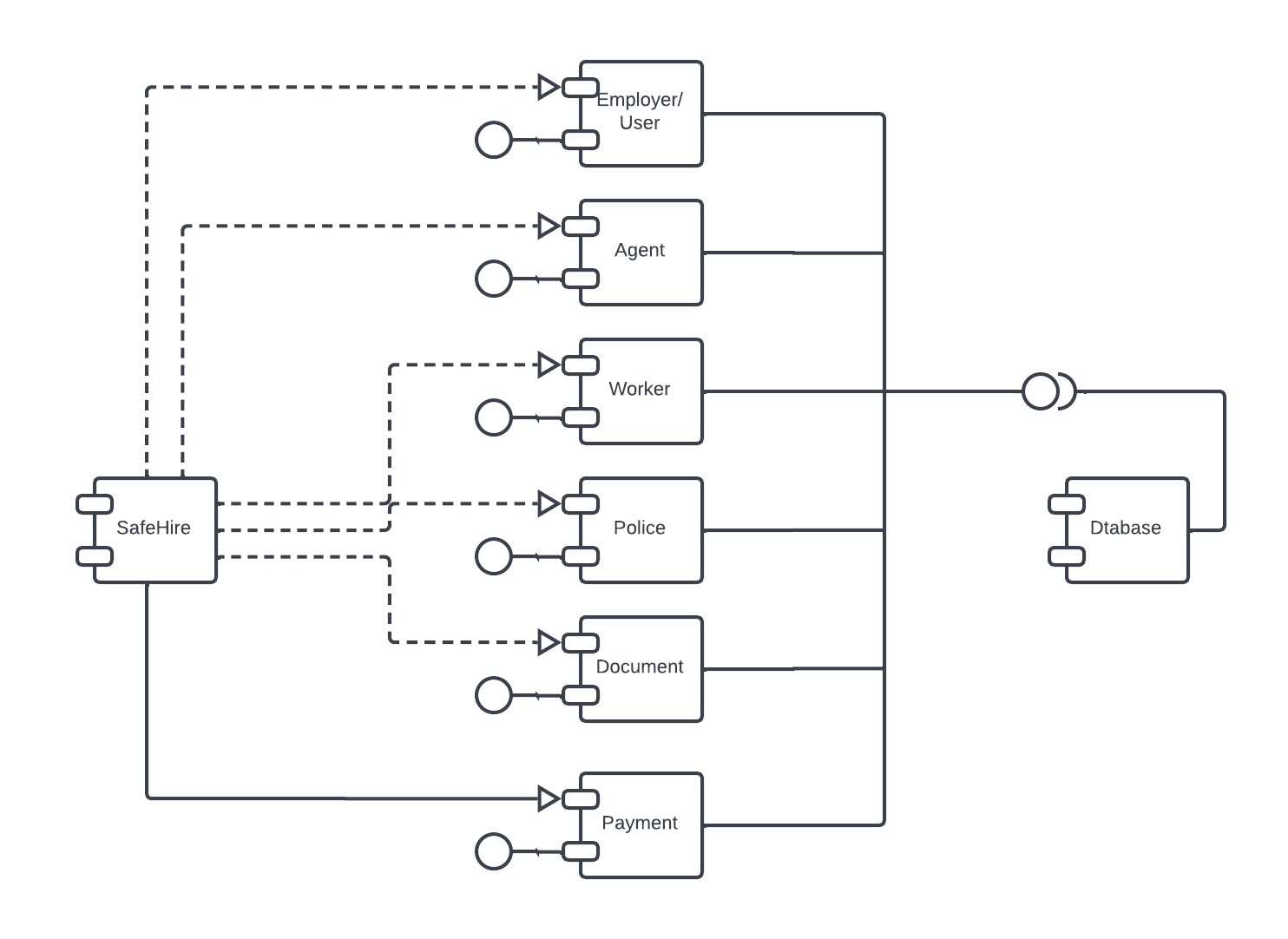
• Provided/Required Interface: A provided interface is an interface that a component offers to other components, while a required interface is an interface that a component needs from other components to function properly. These are represented by lollipops and halfcircles respectively. Component diagrams are useful for modeling the architecture of a software system, and can help identify potential issues and improvements in the design. They can also be used to communicate the structure and behavior of a system to stakeholders, such as developers and project managers.

Figure 7:Component Diagram

**4.2.8 Deployment Diagram**

A deployment diagram is a type of UML diagram that focuses on the physical hardware used to deploy software. It provides a static view of a system's deployment and involves nodes and their relationships. The deployment diagram maps the software architecture to the physical system architecture, showing how the software will be executed on nodes. Communication paths are used to illustrate the relationships between the nodes. Unlike other UML diagram types, which focus on the logical components of a system, the deployment diagram emphasizes the hardware topology. The key components of a deployment diagram are: • Node - A node is a physical or virtual machine on which a component or artifact is deployed. It is represented by a box with the node's name inside. • Component - A component is a software element that performs a specific function or provides a specific service. It is represented by a rectangle with the component's name inside. • Artifact - An artifact is a physical piece of data that is used or produced by a component. It is represented by a rectangle with the artifact's name inside. • Deployment Specification - A deployment specification describes how a component or artifact is deployed on a node. It includes information about the location, version, and configuration parameters of the component or artifact. • Association - An association is a relationship between a node and a component or artifact that represents a deployment dependency. It is represented by a line connecting the two components with optional directionality, multiplicity, and role names. • Communication Path - A communication path represents the connection between nodes, such as a network connection or communication channel. It is represented by a line with optional labels and adornments. Deployment diagrams help in visualizing the physical architecture of a system and identifying any potential issues or bottlenecks in the deployment process. They also aid in planning the deployment strategy and optimizing the use of hardware resources.

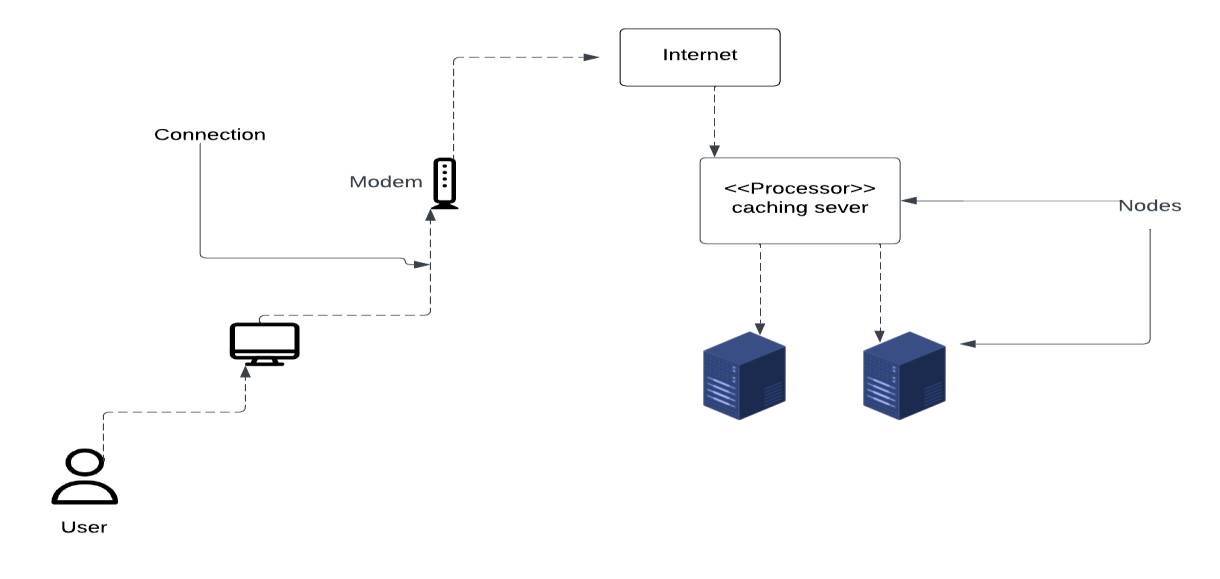


Figure 8:Deployment Diagram

**4.2.9 Collaboration Diagram**

A collaboration diagram is a diagram that is used to represent the relationships between objects in a system. It is like a sequence diagram in that it represents the same information, but it does so in a different way. Instead of showing the flow of messages between objects, it depicts the structure of the objects in the system. This is because collaboration diagrams are based on objectoriented programming, where objects have various attributes and are connected to each other. Thus, collaboration diagrams are a visual representation of the object architecture in a system. A component diagram includes the following components: • Objects: Objects are represented by symbols with their name and class underlined, separated by a colon. In a collaboration diagram, objects are used to represent a class instance and specify its name and class. It is not necessary for every class to have an object representation, and a single class may have multiple objects. Objects are created first, and their class is specified afterwards. Naming objects is important to differentiate them from one another.

**• Actors:** Actors play a key role in the collaboration diagram as they invoke the interaction.

Each actor has its own name and role. In the diagram, one actor initiates the use case.

• **Links:** Links are instances of association that connect objects and actors. They represent

the relationship between objects through which messages are sent. Links are represented

by solid lines and help objects to navigate to other objects.

• **Messages:** Messages represent communication between objects that carry information and

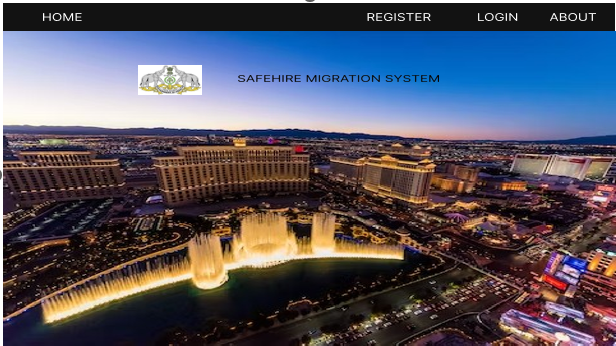
are identified by a sequence number. They are represented by labeled arrows placed near

the link and sent from the sender to the receiver. The direction must be navigable in that

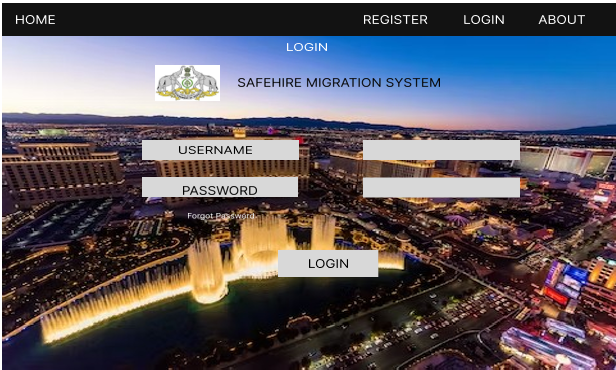
specific direction, and the receiver must understand the message.

## 4.3 USER INTERFACE DESIGN USING FIGMA

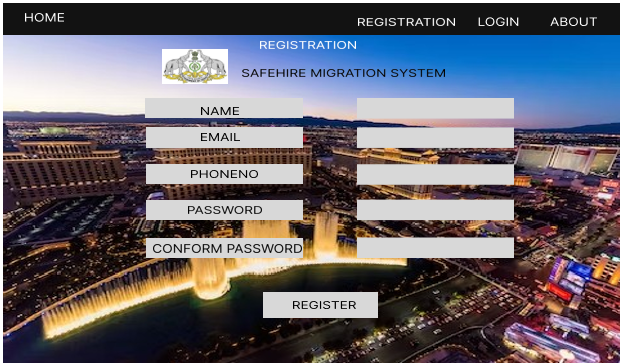
**Form Name: Home Page**



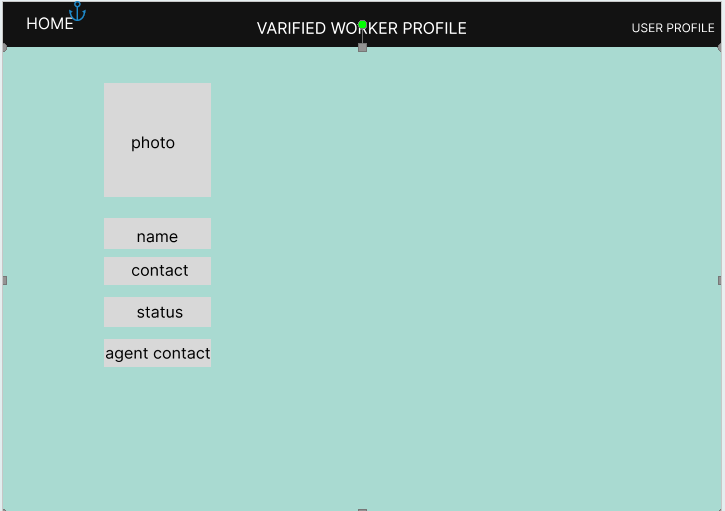
**Form Name: Login Page**



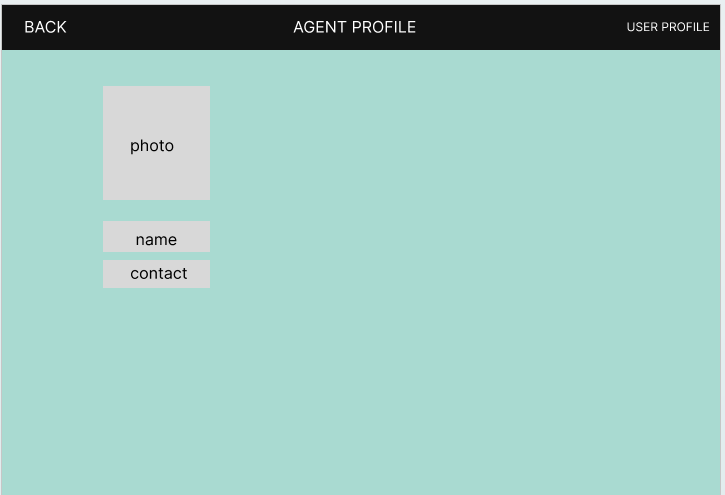
**Form Name: Registration Page**



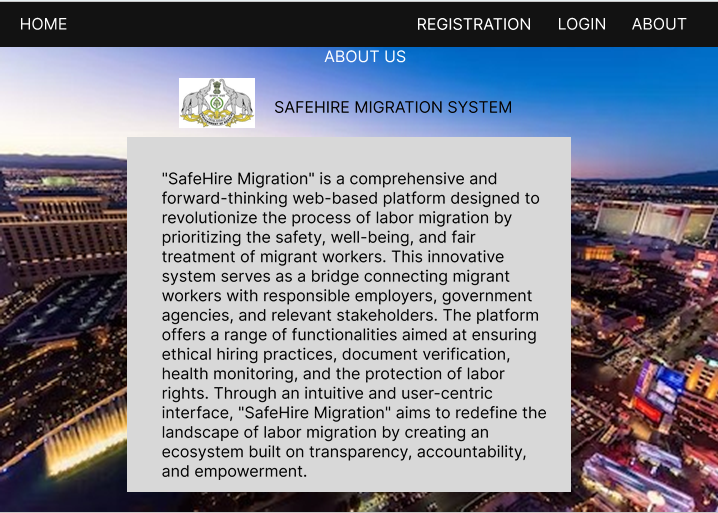
**Form Name: Worker Profile Page**

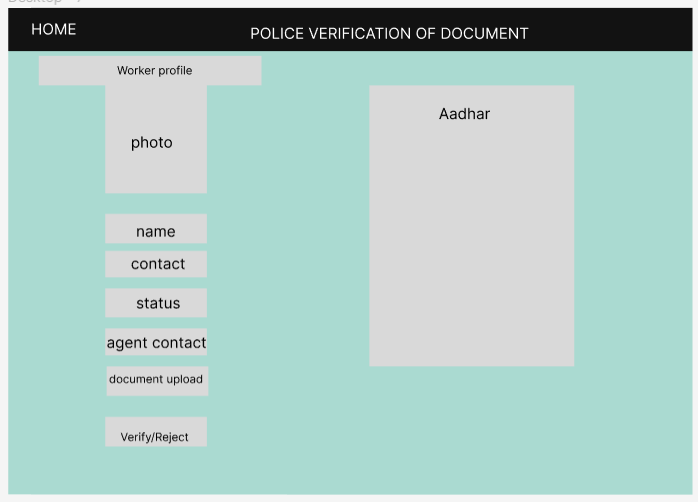
****

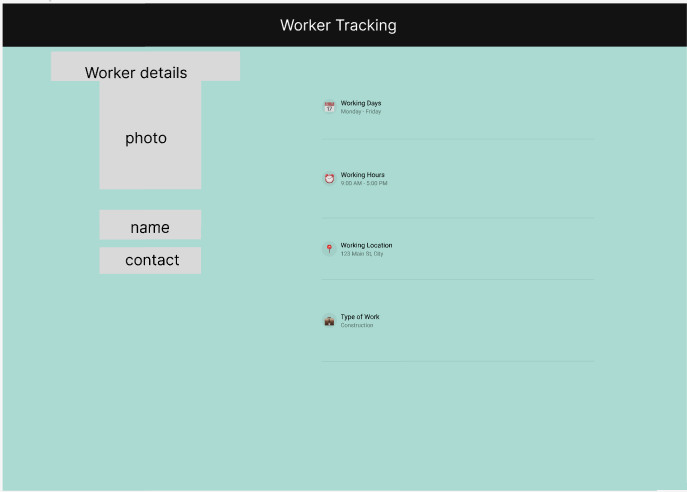
**Form Name: Agent Profile Page**

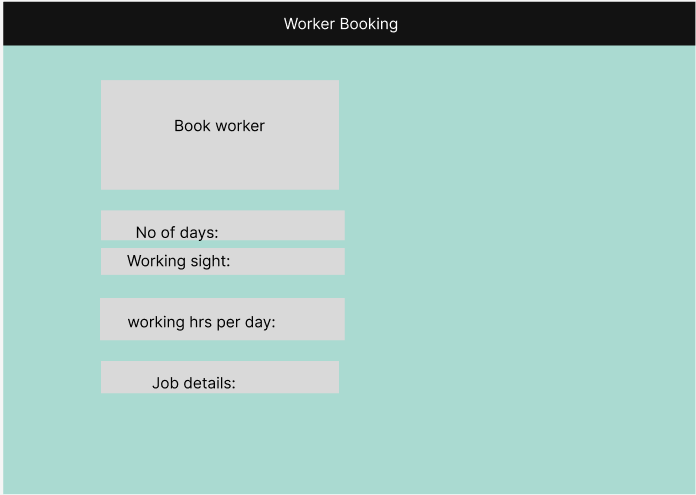
****

**Form Name: About Page**

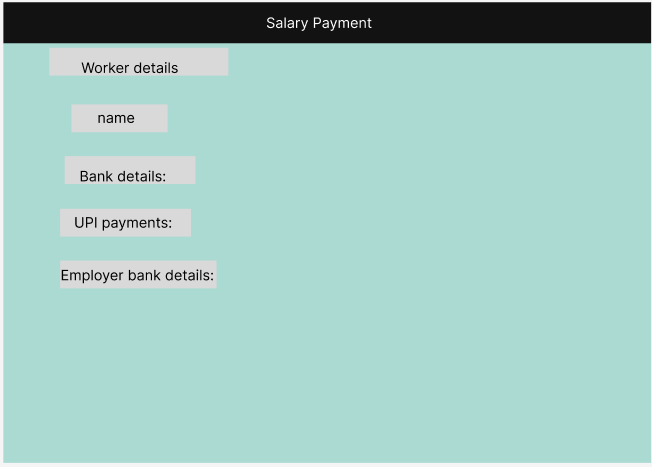
****

****

****

****

****

****

## 4.4 DATABASE DESIGN

A database is an organized collection of information that's organized to enable easy accessibility, administration, and overhauls. The security of information could be a essential objective of any database. The database design process comprises of two stages. In the first stage, user requirements are gathered to create a database that meets those requirements as clearly as possible. This is known as information-level design and is carried out independently of any DBMS. In the second stage, the design is converted from an information-level design to a specific DBMS design that will be used to construct the system. This stage is known as physical-level design, where the characteristics of the specific DBMS are considered. Alongside system design, there is also database design, which aims to achieve two main goals: data integrity and data independence.

### 4.4.1 Relational Database Management System (RDBMS)

Data is represented via a relational model as a set of relationships. Every relationship is like a table of values or datasets. In relational model terminology, rows are called tuples, column headings are called attributes, and tables are relations. A relational file consists of tables, each of which is givena unique name. A row in the chart represents a group of related values.

### 4.4.2 Normalization

The simplest possible grouping of data is used to put them together so that future changes can be made with little influence on the data structures. The formal process of normalizing data structures in a way that reduces duplication and fosters integrity. Using the normalization technique, superfluous fields are removed and a huge table is divided into several smaller ones. Anomalies in insertion, deletion, and updating are also prevented by using it. Keys and relationships are two notions used in the standard form of data modelling. A row in a table is uniquely identified by a key. Primary keys and foreign keys are two different kinds of keys. Primary key is an element, or set of components, in a table that serves as a means of distinguishing between records from the same table. A column in a table known as a foreign key is used to uniquely identify records from other tables. Up to the third normal form, all tables have been normalized. Normalization is a process in database design that aims to organize data into proper tables and columns, making it easily correlated to the data by the user. This process eliminates data redundancy that can be a burden on computer resources. The main steps involved in normalization include:

• Normalizing the data

• Choosing appropriate names for tables and columns

• Choosing the correct names for the data

By following these steps, a developer can create a more efficient and organized database that is easier to manage and maintain.

### First Normal Form

The First Normal Form (1NF) requires that each attribute in a table must contain only atomic or indivisible values. It prohibits the use of nested relations or relations within relations as attribute values within tuples. To satisfy 1NF, data must be moved into separate tables where the data is of similar type in each table, and each table should have a primary key or foreign key as required by the project. This process eliminates repeating groups of data and creates new relations for each non-atomic attribute or nested relation. A relation is in 1NF only if it satisfies the constraints that contain the primary key only

### Second Normal Form

Second normal form (2NF) is a rule in database normalization that states that non-key attributes should not be functionally dependent on only the part of the primary key in a relation that has a composite primary key. In other words, each non-key attribute should depend on the entire primary key, not just a part of it. To achieve this, we need to decompose the table and create new relationships for each subkey along with their dependent attributes. It is important to maintain the relationship with the original primary key and all attributes that are fully functionally dependent on it. A relation is said to be in 2NF only if it satisfies all the 1NF conditions for the primary key and every non-primary key attribute of the relation is fully dependent only on the primary key

### Third Normal Form

Third normal form (3NF) requires that a relation have no non-key attribute that is functionally determined by another non-key attribute or set of non-key attributes. This means that there should be no transitive dependency on the primary key. To achieve 3NF, we decompose the relation and set up a new relation that includes non-key attributes that functionally determine other non-key attributes. This helps eliminate any dependencies that do not just rely on the primary key. A relation is considered a relation in 3NF if it satisfies the conditions of 2NF and, moreover, the non-key attributes of the relation are not dependent on any other non-key attribute

### 4.4.3 Sanitization

Data sanitization entails removing or erasing data from a storage device on purpose and permanently to make sure it cannot be restored. Normally, when data is removed from storage media, the medium is not truly wiped, and an attackerwho gains access to the device can recover the data. Serious questions about security and data privacy are raised by this. Sanitization involves cleaning storage media so that no data remains on the device and that no data can be recovered even with cutting-edge forensic techniques.

**4.4.4 Indexing**

By reducing the number of disc accesses needed when a query is completed, indexing helps a database perform better. It is a data structure method used to locate and access data in a database rapidly. Several database columns are used to generate indexes. • The search key, which is the first column, contains a duplicate of the table's primary key or potential primary key. These values are kept in sorted order so that it is easy to get the corresponding data. Note: The data may or may not be kept in sorted order. • The address of the disc block on which that specific key value is stored is held by a series ofpointers in the second column, which is designated as the Data Reference or Pointer.

### 4.5 TABLE DESIGN

**1.Tbl\_CustomUser**

Primary key: **user\_id**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No:** | **Field name** | **Datatype (Size)** | **Key Constraints** | **Description of the field** |
| 1 | user\_id | Int(5) | Primary Key | User ID |
| 2 | name | CharField(25) | Not null | User’s name |
| 3 | email | EmailField | Unique | User’s email id |
| 4 | phone | CharField(12) | Not null | User’s phone:no |
| 5 | adhar\_number | CharField(12) | Not null | User’s adhar:no |
| 6 | liscence\_number | CharField(10) | Not null | User’s liscence:no |
| 7 | police\_id | CharField(6) | Not null | User’s police id |
| 8 | upload\_file | FileField | Not null | User’s related file upload |
| 9 | is\_varified | BoolianField | Default:False | If user is verified |
| 10 | is\_rejected | BoolianField | Default:False | If user is rejected |
| 11 | user\_type | CharField | Choice | Type of user |
| 12 | is\_employer | BoolianField | Default:False | If user is an employer |
| 13 | is\_agent | BoolianField | Default:False | If user is an agent |
| 14 | is\_police | BoolianField | Default:False | If user is a police |

**2.Tbl\_UserProfile**

Primary key: **user\_id**

Foreign Key:**user\_id** reference **Tbl\_CustomUser**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No:** | **Field name** | **Datatype (Size)** | **Key Constraints** | **Description of the field** |
| 1 | profile\_id | Int(5) | PrimaryKey | Profile ID |
| 2 | gender | CharField(5) | Not null | User’s gender |
| 3 | add\_pf | FileField | Not null | Additional file upload |
| 4 | User\_id | Int(5) | ForeignKey | User’s id |

**3.Tbl\_WorkCategory**

Primary key: **work\_id**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No:** | **Field name** | **Datatype (Size)** | **Key Constraints** | **Description of the field** |
| 1 | work\_id | Int(5) | PrimaryKey | User ID |
| 2 | name | CharField(100) | Not null | Work name |
| 3 | description | TextField | Not null | Description |

**4.Tbl\_MigratoryWorker**

Primary key: **worker\_id**

Foreign Key: **category\_id** references **Tbl\_WorkCategory**, **police\_id** references **Tbl\_CustomUser**, **agent\_id** references **Tbl\_CustomUser,** **employer\_id** references **Tbl\_CustomUser**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No:** | **Field name** | **Datatype (Size)** | **Key Constraints** | **Description of the field** |
| 1 | worker\_id | Int(5) | PrimaryKey | Worker’s ID |
| 2 | name | CharField(25) | Not null | Worker’s name |
| 3 | dob | DateField | Not null | Worker’s dob |
| 4 | gender | CharField(15) | Choice | Worker’s gender |
| 5 | nationality | CharField(12) | Not null | Worker’s nationality |
| 6 | address | TextField | Not null | Worker’s address |
| 7 | Contact\_number | CharField(12) | Not null | Worker’s ph:no |
| 8 | Adhar\_number | CharField(12) | Not null | Worker’s adhar:no |
| 9 | Profile\_image | FileField | Not null | Worker’s photo |
| 10 | document | FileField | Not null | Worker’s doc |
| 11 | Is\_verified | BoolianField | Default:False | If worker is varified |
| 12 | Is\_rejected | BoolianField | Default:False | If worker is rejected |
| 13 | Agent\_id | Int(5) | ForeignKey | Reference id of agent |
| 14 | Category\_id | Int(5) | ForeignKey | Reference id of work category |
| 15 | Workpermit\_varified | Not null | Default:False | Workpermit varified |

**5.Tbl\_BookingWorker**

Primary key: **booking\_id**

Foreign Key: **employer\_id** references **Tbl\_CustomUser, agent\_id** references **Tbl\_CustomUser, worker\_id** references **Tbl\_MigratoryWorker**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No:** | **Field name** | **Datatype (Size)** | **Key Constraints** | **Description of the field** |
| 1 | booking\_id | Int(5) | PrimaryKey | User ID |
| 2 | Employer\_id | Int(5) | ForeignKey | Employer’s id |
| 3 | Agent\_id | Int(5) | ForeignKey | Agent id |
| 4 | Worker\_id | Int(5) | ForeignKey | Worker id |
| 5 | status | CharField(10) | Choice | Booking status |
| 6 | duration | CharField(10) | choice | Duration of booking |
| 7 | Is\_accepted | BoolianField | Default:False | If booking accepted |
| 8 | Is\_rejected | BoolianField | Default:False | If booking rejected |

**3.Tbl\_Payment**

Primary key: **payment\_id**

Foreign Key: **employer\_id** references **Tbl\_CustomUser**, **worker\_id** references **Tbl\_MigratoryWorker,** **booking\_id** references **Tbl\_BookingWorker**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No:** | **Field name** | **Datatype (Size)** | **Key Constraints** | **Description of the field** |
| 1 | payment\_id | Int(5) | PrimaryKey | Payment ID |
| 2 | amount | DecimalField | Not null | Amount |
| 3 | Payment\_receipt | FileField | Not null | Receipt |
| 4 | Employer\_id | Int(5) | ForeignKey | Employer id |
| 5 | Worker\_id | Int(5) | ForeignKey | Worker id |
| 6 | Booking\_id | Int(5) | ForeignKey | Booking id |
| 7 | Razorpay\_payment\_id | CharField(30) | Not null | Rozerpay id |
| 8 | Is\_paid | BoolianField | Default:False | Payment status |
| 9 | date | DateField | Not null | Payment date |

**7.Tbl\_Worker tracking**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No:** | **Field name** | **Datatype (Size)** | **Key Constraints** | **Description of the field** |
| 1 | worker\_id | Int(5) | ForeignKey | Worker id |
| 2 | Employer\_id | Int(5) | ForeignKey | Employer id |
| 3 | Work\_hr | Int(5) | Primarykey | Working hour |
| 4 | Work\_location | CharField (15) | Not null | Working location |
| 5 | Salary\_id | Int(5) | ForeignKey | Salary id |
| 6 | Work\_status | CharField (15) | Not null | Work status |
| 7 | Work\_review | CharField(30) | Not null | Work review |

**8.Tbl\_Document authentication**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No:** | **Field name** | **Datatype (Size)** | **Key Constraints** | **Description of the field** |
| 1 | Authent\_id | Int(5) | PrimaryKey | Authentication id |
| 2 | document\_id | Int(5) | ForeignKey | Document id |
| 3 | Authent\_status | CharField (15) | Not null | Authentication status |
| 4 | Authent\_date | date | Not null | Authentication date |

**9.Tbl\_JobSubmission**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No:** | **Field name** | **Datatype (Size)** | **Key Constraints** | **Description of the field** |
| 1 | Work\_type | Int(5) | PrimaryKey | Authentication id |
| 2 | Work\_location | CharField (15) | Not null | Work location |
| 3 | duration | CharField (15) | Not null | Work duration |
| 4 | Qualif\_required | CharField (15) | Not null | Qualification required |
| 5 | Worker\_id | Int(5) | ForeignKey | Worker id |
| 6 | Employer\_id | Int(5) | ForeignKey | Employer id |

**9.Tbl\_review**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No:** | **Field name** | **Datatype (Size)** | **Key Constraints** | **Description of the field** |
| 1 | Review\_id | Int(5) | PrimaryKey | Review id |
| 2 | Worker\_id | Int(5) | ForeignKey | Worker id |
| 3 | Employer\_id | Int(5) | ForeignKey | Employer id |
| 4 | Review\_content | CharField (15) | Not null | Review content |
| 5 | rating | Int(5) | ForeignKey | Rating worker |
| 6 | Review\_date | date | ForeignKey | Review date |

**9.Tbl\_salary**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No:** | **Field name** | **Datatype (Size)** | **Key Constraints** | **Description of the field** |
| 1 | Salary\_id | Int(5) | PrimaryKey | Review id |
| 2 | Worker\_id | Int(5) | ForeignKey | Worker id |
| 3 | Employer\_id | Int(5) | ForeignKey | Employer id |
| 5 | amount | Int(5) | Not null | Amount |
| 6 | payment\_date | date | Not null | payment date |

# CHAPTER 5

# SYSTEM TESTING

* 1. **INTRODUCTION**

Software Testing is the process of executing software in a controlled manner, in order to answer the question - Does the software behave as specified? Software testing is often used in association with the terms verification and validation. Validation is the checking or testing of items, includes software, for conformance and consistency with an associated specification. Software testing is just one kind of verification, which also uses techniques such as reviews, analysis, inspections, and walkthroughs. Validation is the process of checking that what has been specified is what the user actually wanted.

## TEST PLAN

An extensive document that describes the testing strategy and approach for a specific project or product is called a test plan. Selenium is one of the most popular tools for automating tests when it comes to web application testing. A test plan suggests a number of required steps that need be taken in order to complete various testing methodologies. Therefore, the test plan should include information on the mean time to failure, the cost to locateand correct the flaws, the residual defect density or frequency of occurrence, and the number of test work hours required for each regression test. The testing levels include:

• Unit testing

• Integration Testing

• Data validation Testing

• Output Testing

Overall, defining the scope, identifying the test environment, defining the test cases, generating the test scripts, running the tests, recording the results, analyzing the results, and reporting the results are all steps in the Selenium test plan creation process. By doing the following actions, you can make sure that your testing is thorough and efficient and that you are able to spot and fix any issues before they become serious ones.

### Unit Testing

Unit testing is a software testing technique that focuses on verifying individual components or modules of the software design. The purpose of unit testing is to test the smallest unit of software design and ensure that it performs as intended. Unit testing is typically white-box focused, and multiple components can be tested simultaneously. The component-level design description is used as a guide during testing to identify critical control paths and potential faults within the module's perimeter. During unit testing, the modular interface is tested to ensure that data enters and exits the software unit under test properly. The local data structure is inspected to ensure that data temporarily stored retains its integrity during each step of an algorithm's execution. Boundary conditions are tested to ensure that all statements in a module have been executed at least once, and all error handling paths are tested to ensure that the software can handle errors correctly. Before any other testing can take place, it is essential to test data flow over a module interface. If data cannot enter and exit the system properly, all other tests are irrelevant. Another crucial duty during unit testing is the selective examination of execution pathways to anticipate potential errors and ensure that error handling paths are set up to reroute or halt work when an error occurs. Finally, boundary testing is conducted to ensure that the software operates correctly at its limits. In the SafeHire System, unit testing was carried out by treating each module as a distinct entity and subjecting them to a variety of test inputs. Any issues with the internal logic of the modules were fixed, and each module was tested and run separately after coding. Unused code was eliminated, and it was confirmed that every module was functional and produced the desired outcome.

### Integration Testing

Integration testing is a systematic approach that involves creating the program structure while simultaneously conducting tests to identify interface issues. The objective is to construct a program structure based on the design that uses unit-tested components. The entire program is then tested. Correcting errors in integration testing can be challenging due to the size of the overall program, which makes it difficult to isolate the causes of the errors. As soon as one set of errors is fixed, new ones may arise, and the process may continue in an apparently endless cycle. Once unit testing is complete for all modules in the system, they are integrated to check for any interface inconsistencies. Any discrepancies in program structures are resolved, and a unique program structure is developed.

### Validation Testing or System Testing

The final stage of the testing process involves testing the entire software system as a whole, including all forms, code, modules, and class modules. This is commonly referred to as system testing or black box testing. The focus of black box testing is on testing the functional requirements of the software. A software engineer can use this approach to create input conditions that will fully test each program requirement. The main types of errors targeted by black box testing include incorrect or missing functions, interface errors, errors in data structure or external data access, performance errors, initialization errors, and termination errors.

### Output Testing or User Acceptance Testing

User approval of the system under consideration is tested; in this case, it must meet the needs of the company. When developing, the program should stay in touch with the user and perspective system to make modifications as needed. With regard to the following points, this was done: • Input Screen Designs, • Output Screen Designs, The aforementioned testing is carried out using a variety of test data. The preparation of test data is essential to the system testing process. The system under investigation is then put to the test using the prepared test data. When testing the system, test data issues are found again and fixed using the testing procedures described above. The fixes are also logged for use in the future.

* + 1. **Automation Testing**

A test case suite is executed using specialized automated testing software tools as part of the software testing technique known as automation testing. The test stages are meticulously carried out by a human performing manual testing while seated in front of a computer. Additionally, the automation testing software may generate thorough test reports, compare expected and actual findings, and enter test data into the System Under Test. Software test automation necessitates significant financial and material inputs. Repeated execution of the same test suite will be necessary during subsequent development cycles.

* + 1. **Selenium Testing**

Selenium is a free (open-source) automated testing framework used to verify web applications across different browsers and platforms. You can use multiple programming languages to create Selenium test scripts like Java, C#, Python, etc. Testing done using Selenium testing tool is usually referred to as Selenium testing Since Selenium is a collection of different tools, it also had different developers. Below are the key people who have made significant contributions to the Selenium project. Moreover, Selenium's extensibility allows integration with other testing frameworks, tools, and technologies. It can be combined with testing frameworks like TestNG or JUnit for advanced test management and reporting. It integrates well with popular build tools, source control systems, and defect tracking systems.

**Example:**

**Test Case 1:Login**

**Code**

from django.test import TestCase

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

class LoginTestCase(TestCase):

def setUp(self):

self.driver = webdriver.Chrome()

self.driver.implicitly\_wait(10)

def tearDown(self):

self.driver.quit()

def test\_login(self):

self.driver.get('http://127.0.0.1:8000/login/')

username\_input = self.driver.find\_element(By.NAME, 'username')

password\_input = self.driver.find\_element(By.NAME, 'password')

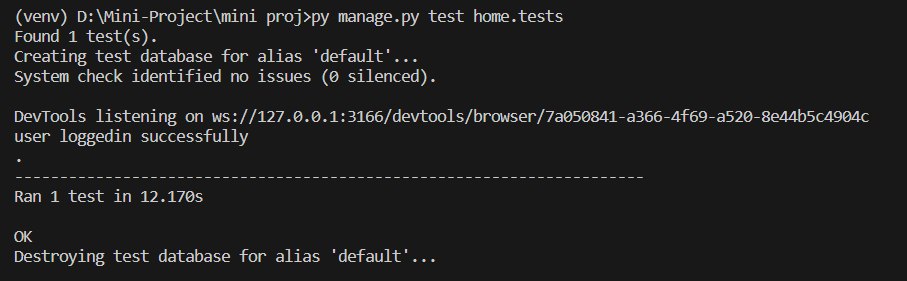
login\_button = self.driver.find\_element(By.ID, 'login')

username\_input.send\_keys('gayathri')

password\_input.send\_keys('gayathri@123')

login\_button.click()

**Screenshot**

****

**Test Report**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case 1** | | | | | |
| **Project Name: SAFEHIRE MIGRATION** | | | | | |
| **Login Test Case** | | | | | |
| **Test Case ID: Test\_1** | | | **Test Designed By: GAYATHRI UNNIKRISHNAN** | | |
| **Test Priority(Low/Medium/High):** | | | **Test Designed Date: 2-12-2023** | | |
| **Module Name**: **Login** | | | **Test Executed By : Ms. Sona Maria Sebastian** | | |
| **Test Title :** Login | | | **Test Execution Date: 2-12-2023** | | |
| **Description:** Verify login with username and password | | |  | | |
| **Pre-Condition :**User has valid username and password | | | | | |
| **Step** | **Test Step** | **Test Data** | **Expected Result** | **Actual Result** | **Status(Pass/**  **Fai l)** |
| 1 | Navigate to login page |  | Dashboard should be displayed | Login page displayed | Pass |
| 2 | Provide valid username | Username =  gayathri | User should be able to login | User Logged in and navigated to User Dashboard | Pass |
| 3 | Provide valid password | Password =  gayathri@123 |
| 4 | Click on login button |  |
| **Post-Condition:** User is validated with database and successfully login into account. The Account session details are logged in database. | | | | | |

**Test Case 2: Add Worker**

**Code**

from django.test import TestCase

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

class WorkerRegistrationTest(TestCase):

def setUp(self):

self.selenium = webdriver.Chrome()

super(WorkerRegistrationTest, self).setUp()

def tearDown(self):

self.selenium.quit()

super(WorkerRegistrationTest, self).tearDown()

def login(self, username, password):

self.selenium.get('http://127.0.0.1:8000/login/')

wait = WebDriverWait(self.selenium, 10)

username\_input = wait.until(EC.presence\_of\_element\_located((By.NAME, 'username')))

password\_input = wait.until(EC.presence\_of\_element\_located((By.NAME, 'password')))

login\_button = wait.until(EC.presence\_of\_element\_located((By.ID, 'login')))

username\_input.send\_keys(username)

password\_input.send\_keys(password)

login\_button.click()

wait.until(EC.url\_contains("/agentpage/"))

def test\_worker\_registration\_after\_login(self):

self.login(username="goutham", password="goutham@123")

self.selenium.get('http://127.0.0.1:8000/addworker')

wait = WebDriverWait(self.selenium, 10)

name\_input = wait.until(EC.presence\_of\_element\_located((By.ID, "name")))

name\_input.send\_keys("John Doe")

nationality\_input = wait.until(EC.presence\_of\_element\_located((By.ID, "nationality")))

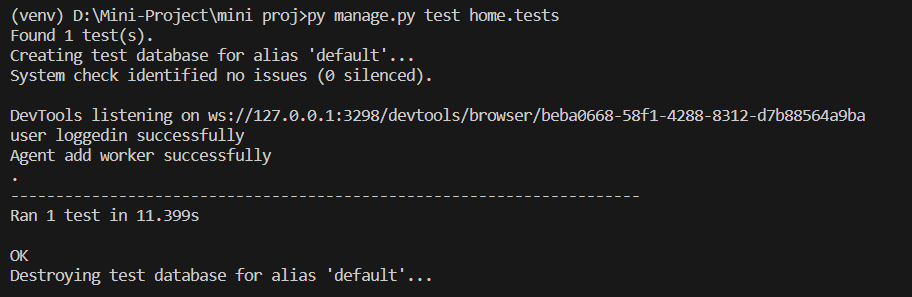
nationality\_input.send\_keys("Country")

registration\_form = wait.until(EC.presence\_of\_element\_located((By.ID, "registrationForm")))

registration\_form.submit()

self.selenium.get('http://127.0.0.1:8000/viewworker')

**Screenshot**



**Test report**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case 2** | | | | | |
| **Project Name: SAFEHIRE MIGRATION** | | | | | |
| **Add Test Case** | | | | | |
| **Test Case ID: Test\_2** | | | **Test Designed By: GAYATHRI UNNIKRISHNAN** | | |
| **Test Priority(Low/Medium/High):** | | | **Test Designed Date: 2-12-2023** | | |
| **Module Name**: **Add Worker** | | | **Test Executed By : Ms. Sona Maria Sebastian** | | |
| **Test Title :** Add Worker | | | **Test Execution Date: 2-12-2023** | | |
| **Description:** add worker using name and nationality | | |  | | |
| **Pre-Condition :**Worker is added with name and nationality | | | | | |
| **Step** | **Test Step** | **Test Data** | **Expected Result** | **Actual Result** | **Status(Pass/**  **Fai l)** |
| 1 | Navigate to login page |  | Dashboard should be displayed | Login page displayed | Pass |
| 2 | Provide valid username | Username =  gayathri | User should be able to login | User Logged in and navigated to User Dashboard | Pass |
| 3 | Provide valid password | Password =  gayathri@123 |
| 4 | Click on login button |  |
| 5 | Navigate to agentpage |  | Dashboard of agent displayed |  | Pass |
| 6 | Click on addworker |  |  |  | pass |
| 7 | Navigate to worker registration page |  | Registration page displayed |  | pass |
| 8 | Provide name | Name=John |  |  | pass |
| 9 | Provide nationality | Nationality=  Indian |  | Worker get registered | Pass |
| 10 | Click on submit |  |  |  | pass |
| **Post-Condition:** Agent successfully login into account. Successfully added the worker | | | | | |

**Test Case 3: Booking Worker**

**Code**

import time

import unittest

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

class BookingSeleniumTest(unittest.TestCase):

def setUp(self):

self.driver = webdriver.Chrome()

self.driver.get("http://127.0.0.1:8000/login")

def tearDown(self):

self.driver.quit()

def login(self, username, password):

username\_input = self.driver.find\_element(By.NAME, "username")

password\_input = self.driver.find\_element(By.NAME, "password")

login\_button = self.driver.find\_element(By.ID, "login")

username\_input.send\_keys(username)

password\_input.send\_keys(password)

login\_button.click()

WebDriverWait(self.driver, 10).until(EC.url\_contains("/accounts/profile"))

def navigate\_to\_agent\_request\_page(self, worker\_id, agent\_id):

agent\_link = self.driver.find\_element(By.XPATH, f"//a[@href='/agent\_contact/{agent\_id}/{worker\_id}/']")

agent\_link.click()

WebDriverWait(self.driver, 10).until(EC.url\_contains("/agent\_contact"))

def test\_booking\_process(self):

self.login(username="gayathri", password="gayathri@123")

user\_page\_link = self.driver.find\_element(By.ID, "worker-list-link")

user\_page\_link.click()

WebDriverWait(self.driver, 10).until(EC.url\_contains("/worker\_list"))

set\_agent\_button = self.driver.find\_element(By.ID,"worker")

set\_agent\_button.click()

send\_request\_button = self.driver.find\_element(By.ID, "sendMessageBtn")

send\_request\_button.click()

WebDriverWait(self.driver, 10).until(EC.presence\_of\_element\_located((By.ID, "bookingModal")))

duration\_input = self.driver.find\_element(By.ID, "duration")

duration\_input.send\_keys("5")

duration\_unit\_select = self.driver.find\_element(By.ID, "durationUnit")

duration\_unit\_select.send\_keys("Months")

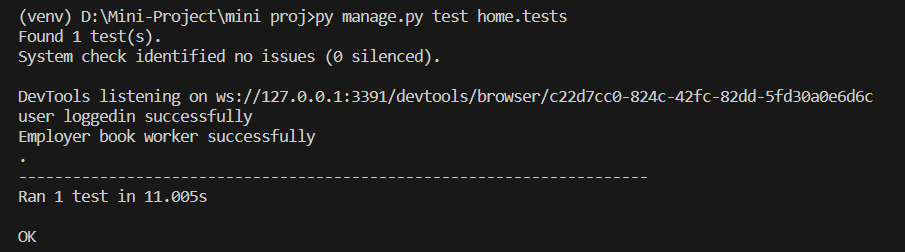
book\_worker\_button = self.driver.find\_element(By.ID, "book")

book\_worker\_button.click()

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

**Screenshot**

****

**Test report**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case 3** | | | | | |
| **Project Name: SAFEHIRE MIGRATION** | | | | | |
| **Booking Test Case** | | | | | |
| **Test Case ID: Test\_3** | | | **Test Designed By: GAYATHRI UNNIKRISHNAN** | | |
| **Test Priority(Low/Medium/High):** | | | **Test Designed Date: 2-12-2023** | | |
| **Module Name**: **Booking Worker** | | | **Test Executed By : Ms.Sona Maria Sebastian** | | |
| **Test Title :** Booking Worker | | | **Test Execution Date: 2-12-2023** | | |
| **Description:** Booking worker by entering duration and unit | | |  | | |
| **Pre-Condition :**worker is booked with duration and duration unit | | | | | |
| **Step** | **Test Step** | **Test Data** | **Expected Result** | **Actual Result** | **Status(Pass/**  **Fai l)** |
| 1 | Navigate to login page |  | Dashboard should be displayed | Login page displayed | Pass |
| 2 | Provide valid username | Username =  goutham | User should be able to login | User Logged in and navigated to User Dashboard | Pass |
| 3 | Provide valid password | Password =  goutham@123 |
| 4 | Click on login button |  |
| 5 | Navigate to Employer page |  | User dashboard displayed |  | pass |
| 6 | Click on worker list |  |  |  | pass |
| 7 | Navigate to worker list page |  | Worker list page displayed |  | pass |
| 8 | Click on agent |  |  |  | pass |
| 9 | Navigate to agent contact page |  | Agent contact page displayed | User able to navigate to agent contact | pass |
| 10 | Click on sent request |  |  |  | pass |
| 11 | Provide booking duration | Duration = 5 |  |  | pass |
| 12 | Provide duration unit | Duration unit= months |  |  | pass |
| 13 | Click on book worker |  | User booked the worker | User is able to book the worker | pass |
| **Post-Condition:** Employer login successfully and booked the worker. | | | | | |

**Test Case 4: Verify Worker**

**Code**

import time

import unittest

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

class PoliceSeleniumTest(unittest.TestCase):

def setUp(self):

self.driver = webdriver.Chrome()

self.driver.get("http://127.0.0.1:8000/login")

def tearDown(self):

self.driver.quit()

def login\_to\_police\_page(self, username, password):

username\_input = self.driver.find\_element(By.NAME, "username")

password\_input = self.driver.find\_element(By.NAME, "password")

login\_button = self.driver.find\_element(By.ID, "login")

username\_input.send\_keys(username)

password\_input.send\_keys(password)

login\_button.click()

WebDriverWait(self.driver, 10).until(EC.url\_contains("/policepage"))

def test\_verify\_worker(self):

self.login\_to\_police\_page(username="gokul", password="gokul@123")

user\_page\_link = self.driver.find\_element(By.ID, "worker-list-link")

user\_page\_link.click()

WebDriverWait(self.driver, 10).until(EC.url\_contains("/workerprofile"))

view\_button = self.driver.find\_element(By.ID, "view")

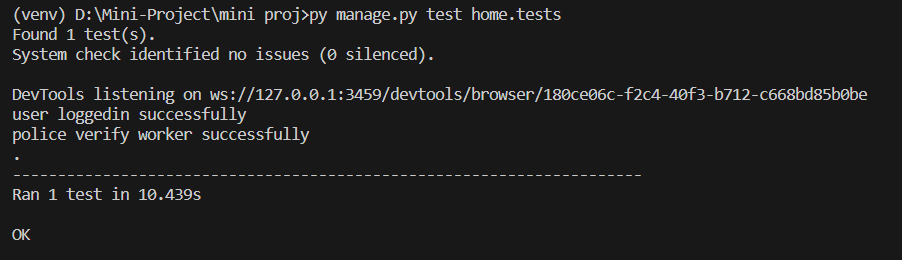
view\_button.click()

WebDriverWait(self.driver, 10).until(EC.url\_contains("/viewprofile"))

verify\_button = self.driver.find\_element(By.ID, "veri")

verify\_button.click()

**Screenshot**



**Test report**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case 4** | | | | | |
| **Project Name: SAFEHIRE MIGRATION** | | | | | |
| **Verification Test Case** | | | | | |
| **Test Case ID: Test\_4** | | | **Test Designed By: GAYATHRI UNNIKRISHNAN** | | |
| **Test Priority(Low/Medium/High):** | | | **Test Designed Date: 2-12-2023** | | |
| **Module Name**: **Verify Worker** | | | **Test Executed By : Ms.Sona Maria Sebastian** | | |
| **Test Title :** Verify Worker | | | **Test Execution Date: 2-12-2023** | | |
| **Description:** Verify login with username and password | | |  | | |
| **Pre-Condition :**User has valid username and password | | | | | |
| **Step** | **Test Step** | **Test Data** | **Expected Result** | **Actual Result** | **Status(Pass/**  **Fai l)** |
| 1 | Navigate to login page |  | Dashboard should be displayed | Login page displayed | Pass |
| 2 | Provide valid username | Username =  gokul | User should be able to login | User Logged in and navigated to User Dashboard | Pass |
| 3 | Provide valid password | Password =  gokul@123 |
| 4 | Click on login button |  |
| 5 | Navigate to police dashboard |  | Police dashboard is displayed | User able navigate to dashboard | pass |
| 6 | Click on workers list |  |  |  | pass |
| 7 | Navigate to worker  profile page |  | Worker profile is displayed | User is able to navigate to worker profile page | pass |
| 8 | Click on view profile |  |  |  | pass |
| 9 | Navigate to view profile page |  | View profile page is displayed | User is able to navigate to view profile page | pass |
| 10 | Click on verify |  | Police verify the worker | Police is able to verify worker | pass |
| **Post-Condition:** Police logged in successfully and verify the worker. | | | | | |

# CHAPTER 6

# IMPLEMENTATION

## INTRODUCTION

The implementation phase of the "SafeHire Migration" project marks the transition from planning to execution. This stage involves turning the project design into a fully functional system. The goal is to ensure that the platform effectively addresses the challenges faced by employers in managing migratory workers. This chapter provides an overview of the implementation process, detailing the steps taken to bring the envisioned system to life.

## IMPLEMENTATION PROCEDURES

Implementation procedures refer to the step-by-step process of putting a plan or design into action. In the context of a project or system development, implementation procedures involve the actual execution of the proposed solution or system. This phase follows the planning and design stages and precedes the testing and maintenance phases in the software development life cycle.

Key elements of implementation procedures typically include:

**Coding/Programming:** Writing the actual code or programming scripts based on the design specifications. This involves translating the logical design into a programming language that the computer can execute.

**Database Creation:** If the project involves a database, the implementation phase may include setting up the database structure and populating it with initial data.

**Integration:** Combining different components, modules, or subsystems of the system to ensure they work together seamlessly. This may involve integrating software with hardware components.

**Testing:** Conducting various tests to ensure that individual components and the entire system function as intended. Testing helps identify and address any issues or bugs before the system is deployed.

**Deployment:** Rolling out the system or solution for use by end-users. Deployment can be gradual or all at once, depending on the project requirements.

**User Training:** Providing training to end-users or stakeholders on how to use the new system. This step is crucial to ensure a smooth transition and effective utilization of the implemented solution.

**Documentation:** Creating and updating documentation that describes the system's architecture, functionality, and usage. Documentation is essential for future maintenance and troubleshooting.

**Monitoring and Evaluation:** Continuously monitoring the system's performance after deployment and evaluating its effectiveness. This step helps identify any post-implementation issues and ensures ongoing improvements.

### User Training

User training is a critical component of the implementation process. It involves educating relevant stakeholders, including agents, police, employers, and administrators, on how to effectively use the SafeHire Migration platform. Training sessions are designed to familiarize users with the features and functionalities, ensuring they can navigate the system seamlessly. Emphasis is placed on security measures, data handling, and efficient utilization of the platform.

### Training on the Application Software

In this section, we delve into the specific training provided on the application software. Users are guided through the registration process, profile creation, document verification, and communication functionalities. Special attention is given to the booking management system and the secure payment integration through RazorPay. The training aims to empower users with the knowledge needed to utilize the software to its full potential.

### System Maintenance

System maintenance is an ongoing process to ensure the platform's reliability, security, and optimal performance. This involves regular updates, bug fixes, and enhancements based on user feedback and changing requirements. The chapter discusses the strategies and protocols in place for system maintenance, highlighting the commitment to delivering a robust and continuously improving solution.

# CHAPTER 7

# CONCLUSION AND FUTURE SCOPE

## CONCLUSION

In conclusion, the mini project lays the foundation for a user management system with features such as profile creation, worker addition, and document verification. While it provides basic functionalities for agents, police, and employers, there is room for improvement in terms of user interface and scalability. The integration of RazorPay for payment processing is a positive step, but security measures, such as data encryption, could be further detailed. Future developments might focus on refining the user experience, enhancing security features, and expanding the system's capabilities to meet the growing demands of users.

* 1. **FUTURE SCOPE**

Looking forward, the main project has a lot of potential for improvements. We aim to make the system more user-friendly, secure, and scalable. We're exploring using advanced encryption for better security and considering incorporating machine learning for smarter analytics and fraud detection. There's also the possibility of using blockchain for secure data handling. Regular updates will ensure that the system stays in line with labor and immigration rules, and we're thinking about creating a mobile app for easier access. Collaborating with authorities, gathering feedback, and making the system more internationally friendly are all part of our plan to make the project more advanced, user-focused, and applicable worldwide in the field of workforce management.

# CHAPTER 8

# BIBLIOGRAPHY

### REFERENCES:

* + - Gary B. Shelly, Harry J. Rosenblatt, “System Analysis and Design”, 2009.
    - Roger S Pressman, “Software Engineering”, 1994
    - IEEE Std 1016 Recommended Practice for Software Design Descriptions.

### WEBSITES:

* www.w3schools.com
* <https://getbootstrap.com/>
* <https://www.djangoproject.com/>
* https://jquery.com/
* https://chat.openai.c

# CHAPTER 9

# APPENDIX

## Sample Code

**Login.html**

{% load socialaccount %}

{% load static %}

<!DOCTYPE html>

<html>

<head>

<!-- Basic -->

<meta charset="utf-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<!-- Mobile Metas -->

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

<!-- Site Metas -->

<meta name="keywords" content="" />

<meta name="description" content="" />

<meta name="author" content="" />

<link rel="shortcut icon" href="{% static 'images/favicon.png'%}" type="image/x-icon">

<title>SafeHire Migration</title>

<!-- fonts style -->

<link href="https://fonts.googleapis.com/css?family=Open+Sans:400,700|Poppins:400,600,700&display=swap" rel="stylesheet" />

<!-- Custom styles for this template -->

<link href="{% static 'css/style.css'%}" rel="stylesheet" />

<!-- responsive style -->

<link href="{% static 'css/responsive.css'%}" rel="stylesheet" />

<!-- Bootstrap CSS -->

<link rel="stylesheet" type="text/css" href="{% static 'css/bootstrap.css'%}" />

</head>

<body class="sub\_page">

<!-- Header Section -->

<header class="header\_section">

<div class="header\_bottom">

<div class="container-fluid">

<nav class="navbar navbar-expand-lg custom\_nav-container">

<a class="navbar-brand" href="index.html">

<span>

SafeHire Migration

</span>

</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

<span class=""></span>

</button>

<div class="collapse navbar-collapse ml-auto" id="navbarSupportedContent">

<ul class="navbar-nav ">

<li class="nav-item active">

<a class="nav-link" href="{% url 'index' %}"> HOME <span class="sr-only">(current)</span></a>

</li>

<li class="nav-item">

<a class="nav-link" href="{% url 'about' %}"> ABOUT </a>

</li>

<li class="nav-item">

<a class="nav-link" href="{% url 'services' %}"> SERVICES </a>

</li>

<li class="nav-item">

<a class="nav-link" href="{% url 'register' %}"> REGISTER </a>

</li>

<li class="nav-item">

<a class="nav-link" href="{% url 'login' %}"> LOGIN </a>

</li>

</ul>

</div>

</nav>

</div>

</div>

</header>

{% if messages %}

{% for message in messages %}

<div class="alert" role="alert">

{{ message }}

</div>

<div class="alert" role="alert" style="color: white;">

{{ message }}

</div>

{% endfor %}

{% endif %}

<form method="POST" action="/login/">

{% csrf\_token %}

<div class="forms">

<div class="form-content">

<div class="login-form">

<div class="title">Login</div>

<form action="#">

<div class="input-boxes">

<div class="input-box">

<i class="fas fa-envelope"></i>

<input type="text" name="username" id="username" placeholder="Enter your username" oninput="validateUsername()">

<span id="usernameError" class="error"></span>

</div>

<div class="input-box">

<i class="fas fa-lock"></i>

<input type="password" name="password" id="password" placeholder="Enter your password" oninput="validatePassword()">

<label for="showPassword"></label>

<span id="passwordError" class="error"></span>

</div>

<div class="text"><a href="{% url 'password\_reset' %}">Forgot password?</a></div>

<script>

document.getElementById("showPassword").addEventListener("change", function() {

var passwordInput = document.getElementById("password");

if (this.checked) {

passwordInput.type = "text"; // Show password

} else {

passwordInput.type = "password"; // Hide password

}

});

</script>

<div class="button input-box">

<input id="login"type="submit" value="Submit">

</div>

<div class="text sign-up-text">Don't have an account? <a href="{% url 'index' %}">Register</a></div>

</div>

<button><a href="{% provider\_login\_url 'google' %}">Sign in with Google</a></button>

</form>

</div>

</div>

</div>

</form>

</body>

<!-- Bootstrap and jQuery scripts should be included here -->

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

<script>

function validateUsername() {

var username = document.getElementById("username").value.trim();

var usernameError = document.getElementById("usernameError");

var usernamePattern = /^[A-Za-z0-9\_]{3,20}$/; // Allow letters, numbers, and underscores; length between 3 and 20 characters

if (username === "") {

usernameError.textContent = "Please enter your username.";

usernameError.style.color = "red"; // Set error message color to red

return false;

} else {

usernameError.textContent = "";

return true;

}

}

function validatePassword() {

var password = document.getElementById("password").value;

var passwordError = document.getElementById("passwordError");

if (password === "") {

passwordError.textContent = "Please enter a Password.";

passwordError.style.color = "red"; // Set error message color to red

return false;

} else {

passwordError.textContent = "";

}

}

</script>

<section class="info\_section">

<div class="container">

<div class="info\_logo">

<a class="navbar-brand" href="index.html">

<span>

<p>&copy; 2023 SafeHire Migration</p>

</span>

</a>

</div>

</div>

</section>

<style>

.header\_section {

background-color: #090909;

color: #fff;

padding: 15px 0;

}

.header\_bottom {

display: flex;

justify-content: space-between;

align-items: center;

}

/\* Style for the navigation links \*/

.navbar-nav .nav-link {

color: #fff;

margin-right: 20px;

font-weight: bold;

}

/\* Style for the active navigation link \*/

.navbar-nav .nav-item.active .nav-link {

background-color: #0056b3;

border-radius: 4px;

}

/\* Style for the logo text \*/

.navbar-brand span {

font-size: 24px;

}

/\* Style for the navigation toggler button (hamburger icon) \*/

.navbar-toggler {

border: 1px solid #fff;

color: #fff;

}

/\* Style for the navigation toggler icon (bars icon) \*/

.navbar-toggler-icon {

background-color: #fff;

}

/\* Style for the login form container \*/

.forms {

display: flex;

justify-content: center;

align-items: center;

min-height: 80vh;

}

/\* Style for the login form content \*/

.form-content {

background-color: #fff;

padding: 20px;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

text-align: center;

width: 100%;

max-width: 400px;

}

/\* Style for the login form title \*/

.title {

font-size: 24px;

font-weight: bold;

margin-bottom: 20px;

}

/\* Style for the input boxes \*/

.input-box {

position: relative;

margin-bottom: 20px;

}

/\* Style for the input icons \*/

.input-box i {

position: absolute;

top: 50%;

left: 16px;

transform: translateY(-50%);

color: #555;

}

/\* Style for the input fields \*/

.input-box input {

width: 100%;

padding: 10px 16px;

border: 1px solid #ccc;

border-radius: 4px;

font-size: 16px;

}

/\* Style for the show password checkbox \*/

.input-box label {

position: absolute;

top: 50%;

right: 16px;

transform: translateY(-50%);

cursor: pointer;

}

/\* Style for the error messages \*/

.error {

color: red;

}

/\* Style for the "Forgot password?" link \*/

.text a {

color: #007bff;

text-decoration: none;

}

/\* Style for the submit button \*/

.button input[type="submit"] {

background-color: #007bff;

color: #fff;

padding: 10px 20px;

border: none;

border-radius: 4px;

font-size: 16px;

cursor: pointer;

transition: background-color 0.3s;

}

.button input[type="submit"]:hover {

background-color: #0056b3;

}

/\* Style for the "Don't have an account?" link \*/

.sign-up-text a {

color: #007bff;

text-decoration: none;

}

/\* Style for the Google sign-in button \*/

.button button {

background-color: #fff;

color: #333;

border: 1px solid #ccc;

padding: 10px 20px;

border-radius: 4px;

font-size: 16px;

margin-top: 20px;

cursor: pointer;

transition: background-color 0.3s;

}

.button button:hover {

background-color: #f5f5f5;

}

/\* Style for the footer section \*/

.info\_section {

background-color: #333;

color: #fff;

padding: 20px 0;

text-align: center;

}

.info\_logo p {

margin: 0;

font-size: 14px;

}

</style>

</body>

</html>

**Addworker.html**

{% extends "agentpage.html" %}

{% block title %}Worker{% endblock %}

{% block content %}

{% if messages %}

{% for message in messages %}

<div class="alert" role="alert">

{{ message }}

</div>

{% endfor %}

{% endif %}

<h1>Registration Form for Migratory Worker</h1>

<form id="registrationForm" action="{% url 'addworker' %}" method="POST" enctype="multipart/form-data" onsubmit="return validateForm(event);">

{% csrf\_token %}

<!-- Your other form elements -->

<label for="first\_name">Name:</label>

<input type="text" name="first\_name" id="name" oninput="validateName('name')">

<span id="nameError" class="error"></span>

<!-- Other fields -->

<label for="nationality">Nationality:</label>

<input type="text" id="nationality" name="nationality" oninput="validateNationality('nationality')">

<span id="nationalityError" class="error"></span><br><br>

<!-- Other fields -->

<label for="adhar\_number">Adhar Number:</label>

<input type="text" id="adhar\_number" name="adhar\_number" oninput="validateAdharNumber('adhar\_number')">

<span id="adharError" class="error"></span><br><br>

<label for="dob\_0">Date of Birth:</label>

<input type="date" id="dob\_0" name="dob\_0" required onblur="validateDateOfBirth()"><br><br>

<span id="dobError" class="error"></span>

<label for="gender">Gender:</label>

<select id="gender" name="gender">

<option value="male">Male</option>

<option value="female">Female</option>

<option value="other">Other</option>

</select>

<span id="genderError" class="error"></span><br><br>

<label for="address">Address:</label>

<textarea id="address" name="address" oninput="validateAddress()"></textarea><br><br>

<span id="addressError" class="error"></span>

<label for="contact\_number">Contact Number:</label>

<input type="tel" id="contact\_number" name="contact\_number" oninput="validateContactNumber()"><br><br>

<span id="contactError" class="error"></span>

<label for="work\_assign">Assign Work:</label>

<select id="work\_assign" name="work\_assign">

<option value="select\_work">Select Work</option>

{% for category in categories %}

<option value="{{ category.id }}">{{ category.name }}</option>

{% endfor %}

</select>

<br><br>

<label for="profile\_image">Profile Image:</label>

<input type="file" id="profile\_image" name="profile\_image" accept="image/\*"><br><br>

<label for="document">Upload Document or Government ID:</label>

<input type="file" id="document" name="document" accept=".pdf, .doc, .docx, image/\*"><br><br>

<input type="submit" value="Submit">

</form>

<style>

body, h1, h2, h3, p, ul, li {

margin: 0;

padding: 0;

}

body {

font-family: Arial, sans-serif;

background-color: #f8f9fa;

}

h1 {

text-align: center;

margin: 20px 0;

}

form {

width: 50%;

margin: 0 auto;

background-color: #fff;

padding: 20px;

border: 1px solid #ccc;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

label {

display: block;

font-weight: bold;

margin-top: 10px;

}

input[type="text"],

input[type="date"],

input[type="tel"],

input[type="email"],

input[type="password"],

select,

textarea {

width: 100%;

padding: 10px;

margin: 5px 0;

border: 1px solid #ccc;

border-radius: 3px;

box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);

}

select {

height: 35px;

}

input[type="submit"] {

background-color: #007BFF;

color: #fff;

padding: 10px 20px;

border: none;

border-radius: 3px;

cursor: pointer;

margin-top: 10px;

}

.error-border {

border: 1px solid red;

}

.error {

color: red;

}

input[type="submit"]:hover {

background-color: #0056b3;

}

</style>

<script>

function validateName(inputID) {

const nameInput = document.getElementById(inputID);

const name = nameInput.value.trim();

const nameError = document.getElementById("nameError");

if (name === "") {

nameError.textContent = "Name is required";

nameInput.classList.add("error-border"); // Add a CSS class for styling

return false;

} else {

nameError.textContent = "";

nameInput.classList.remove("error-border"); // Remove the CSS class

return true;

}

}

function validateDateOfBirth() {

const dobInput = document.getElementById("dob\_0");

const dob = dobInput.value.trim();

const dobError = document.getElementById("dobError");

if (dob === "") {

dobError.textContent = "Date of Birth is required";

dobInput.classList.add("error-border"); // Add a CSS class for styling

return false;

} else {

dobError.textContent = "";

dobInput.classList.remove("error-border"); // Remove the CSS class

return true;

}

}

function validateGender() {

const genderInput = document.getElementById("gender");

const selectedGender = genderInput.value;

const genderError = document.getElementById("genderError");

if (selectedGender === "select\_gender") {

genderError.textContent = "Please select a gender";

genderInput.classList.add("error-border");

return false;

} else {

genderError.textContent = "";

genderInput.classList.remove("error-border");

return true;

}

}

function validateAddress() {

const addressInput = document.getElementById("address");

const address = addressInput.value.trim();

const addressError = document.getElementById("addressError");

if (address === "") {

addressError.textContent = "Address is required";

addressInput.classList.add("error-border");

return false;

} else {

addressError.textContent = "";

addressInput.classList.remove("error-border");

return true;

}

}

function validateNationality(inputID) {

const nationalityInput = document.getElementById(inputID);

const nationality = nationalityInput.value.trim();

const nationalityError = document.getElementById("nationalityError");

if (nationality === "") {

nationalityError.textContent = "Nationality is required";

nationalityInput.classList.add("error-border");

return false;

} else {

nationalityError.textContent = "";

nationalityInput.classList.remove("error-border");

return true;

}

}

function validateAdharNumber(inputID) {

const adharNumberInput = document.getElementById(inputID);

const adharNumber = adharNumberInput.value.trim();

const adharError = document.getElementById("adharError");

// Regular expression for validating Aadhar number

const adharRegex = /^\d{12}$/;

if (adharNumber === "") {

adharError.textContent = "Adhar Number is required";

adharNumberInput.classList.add("error-border");

return false;

} else if (!adharRegex.test(adharNumber)) {

adharError.textContent = "Invalid Adhar Number";

adharNumberInput.classList.add("error-border");

return false;

} else {

adharError.textContent = ""; // Clear the error message

adharNumberInput.classList.remove("error-border");

return true;

}

}

function validateContactNumber() {

const contactNumberInput = document.getElementById("contact\_number");

const contactNumber = contactNumberInput.value.trim();

const contactError = document.getElementById("contactError");

// Regular expression for validating a 10-digit phone number

const phoneRegex = /^\d{10}$/;

if (contactNumber === "") {

contactError.textContent = "Contact Number is required";

contactNumberInput.classList.add("error-border");

return false;

} else if (!phoneRegex.test(contactNumber)) {

contactError.textContent = "Invalid Contact Number";

contactNumberInput.classList.add("error-border");

return false;

} else {

contactError.textContent = "";

contactNumberInput.classList.remove("error-border");

return true;

}

}

function validateWorkAssign() {

const workAssignSelect = document.getElementById("work\_assign");

const workAssignError = document.getElementById("workAssignError");

if (workAssignSelect.value === "select\_work") {

workAssignError.textContent = "Please select a valid work assignment";

workAssignSelect.classList.add("error-border");

return false;

} else {

workAssignError.textContent = "";

workAssignSelect.classList.remove("error-border");

return true;

}

}

function validateFile(inputID) {

const fileInput = document.getElementById(inputID);

const errorSpan = document.getElementById(inputID + "Error");

if (fileInput.files.length === 0) {

errorSpan.textContent = "Please select a file";

fileInput.classList.add("error-border");

return false;

} else {

errorSpan.textContent = "";

fileInput.classList.remove("error-border");

return true;

}

}

function validateForm(event) {

// Call all validation functions

const isNameValid = validateName("name");

const isDobValid = validateDateOfBirth();

const isGenderValid = validateGender();

const isAddressValid = validateAddress();

const isNationalityValid = validateNationality("nationality");

const isAdharNumberValid = validateAdharNumber("adhar\_number");

const isContactNumberValid = validateContactNumber();

const isWorkAssignValid = validateWorkAssign();

const isProfileImageValid = validateFile("profile\_image");

const isDocumentValid = validateFile("document");

// Check if all validations pass

if (

isNameValid &&

isDobValid &&

isGenderValid &&

isAddressValid &&

isNationalityValid &&

isAdharNumberValid &&

isContactNumberValid &&

isWorkAssignValid &&

isProfileImageValid &&

isDocumentValid

) {

// All validations passed, allow form submission

return true;

} else {

// Prevent form submission

event.preventDefault();

return false;

}

}

</script>

{% endblock content %}

**Viewworker.html**

{% extends "agentpage.html" %}

{% block title %}Registered Workers{% endblock %}

{% block content %}

<div class="container">

<h1 class="mt-4 mb-4">Registered Workers</h1>

<table class="table table-bordered table-striped">

<thead>

<tr>

<th>Photo</th>

<th>Name</th>

<th>Date of Birth</th>

<th>Nationality</th>

<th>Contact Number</th>

<th>Adhar Number</th>

<th>Gender</th> <!-- New column for Gender -->

<th>Assigned Work</th>

<th>Document</th> <!-- New column for Document -->

<th>Actions</th> <!-- New column for buttons -->

</tr>

</thead>

<tbody>

{% for worker in workers %}

<tr>

<td>

{% if worker.profile\_image %}

<img src="{{ worker.profile\_image.url }}" alt="{{ worker.first\_name }}'s photo" style="max-width: 100px; max-height: 100px;">

{% else %}

<p>No Profile for {{ worker.first\_name }}</p>

{% endif %}

</td>

<td>{{ worker.first\_name }}</td>

<td>{{ worker.dob }}</td>

<td>{{ worker.nationality }}</td>

<td>{{ worker.contact\_number }}</td>

<td>{{ worker.adhar\_number }}</td>

<td>{{ worker.gender }}</td> <!-- Display Gender -->

<td>

{% if worker.category %}

{{ worker.category.name }}

{% else %}

No assigned work

{% endif %}

</td>

<td>

{% if worker.document %}

<a href="{{ worker.document.url }}" target="\_blank">View Document</a>

{% else %}

No document uploaded

{% endif %}

</td>

<td>

<a href="{% url 'update\_worker' worker.id %}" class="btn btn-primary">Update</a>

<a href="{% url 'delete\_worker' worker.id %}" class="btn btn-danger">Delete</a>

</td>

</tr>

{% empty %}

<tr>

<td colspan="10">No workers registered yet.</td>

</tr>

{% endfor %}

</tbody>

</table>

<a href="{% url 'addworker' %}" class="btn btn-primary">Add Worker</a>

</div>

</body>

<style>

/\* Global Styles \*/

body {

font-family: Arial, sans-serif;

background-color: #f8f9fa;

margin: 0;

padding: 0;

}

/\* Container Styles \*/

.container {

width: 80%;

margin: 0 auto;

padding: 20px;

background-color: #fff;

border: 1px solid #ccc;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

margin-top: 20px;

}

/\* Table Styles \*/

.table {

width: 100%;

}

.table th,

.table td {

text-align: left;

padding: 8px;

}

.table th {

background-color: #

color: #fff;

}

.table tr:nth-child(even) {

background-color: #f2f2f2;

}

/\* Button Style \*/

.btn {

display: inline-block;

padding: 10px 20px;

text-decoration: none;

border: none;

border-radius: 3px;

cursor: pointer;

margin-top: 10px;

}

.btn-primary {

background-color: #007BFF;

color: #fff;

}

.btn-primary:hover {

background-color: #0056b3;

}

.btn-danger {

background-color: #FF0000;

color: #fff;

}

.btn-danger:hover {

background-color: #CC0000;

}

</style>

{% endblock %}

**Viewprofile.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Worker Profile</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h1 class="mt-4 mb-4">Worker Profile</h1>

<div class="worker-profile">

<div class="profile-details">

<p>Name: <span>{{ worker.first\_name }}</span></p>

<p>Date of Birth: <span>{{ worker.dob }}</span></p>

<p>Nationality: <span>{{ worker.nationality }}</span></p>

<p>Contact Number: <span>{{ worker.contact\_number }}</span></p>

<p>Adhar Number: <span>{{ worker.adhar\_number }}</span></p>

<p>Work Assigned: <span>{{ worker.category }}</span></p>

<p>Agent:<a href="/workeragent/{{ worker.agent.id }}" class="btn btn-secondary">View Agent Profile</a>

</div>

<div class="profile-image">

{% if worker.profile\_image %}

<img src="{{ worker.profile\_image.url }}" alt="{{ worker.first\_name }}'s photo" style="max-width: 200px;">

{% else %}

<p>No Profile Image for {{ worker.first\_name }}</p>

{% endif %}

</div>

</div>

<div class="document">

<h4>View Documents</h4>

<ul class="document">

{% if worker.document %}

<li>

<a href="{{ worker.document.url }}" >View Document</a>

<a href="{{ worker.document.url }}" download="{{ worker.document.name }}">Download</a>

</li>

{% else %}

<p>No document uploaded</p>

{% endif %}

</ul>

</div>

<div class="action-buttons">

<form action="/verify\_worker/{{ worker.id }}/" method="post">

{% csrf\_token %}

<button class="btn btn-primary" type="submit">Verify</button>

</form>

<form action="/reject\_worker/{{ worker.id }}/" method="post">

{% csrf\_token %}

<button class="btn btn-danger" type="submit">Reject</button>

</form>

</div>

<div class="work-permit">

<h4>Work Permit</h4>

{% if worker.work\_permit\_verified %}

<a href="{% url 'generate\_work\_permit\_pdf' worker.id %}" class="btn btn-primary" download>Download Work Permit</a>

{% else %}

<p>No work permit generated or not verified</p>

{% endif %}

</div>

<script src="script.js"></script>

</body>

<style>

/\* Global Styles \*/

body {

font-family: Arial, sans-serif;

background-color: #f8f9fa;

margin: 0;

padding: 0;

}

/\* Container Styles \*/

.container {

width: 80%;

margin: 0 auto;

padding: 20px;

background-color: #fff;

border: 1px solid #ccc;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

margin-top: 20px;

}

/\* Worker Profile Styles \*/

.worker-profile {

display: flex;

justify-content: space-between;

align-items: center;

}

.profile-details {

flex: 1;

}

.profile-details p {

margin-bottom: 10px;

}

.profile-details span {

font-weight: bold;

}

.profile-image {

flex: 1;

text-align: center;

}

.profile-image img {

max-width: 200px;

border: 1px solid #ccc;

border-radius: 5px;

}

/\* Document Styles \*/

.document-upload {

margin-top: 20px;

}

.document-upload h2 {

margin-bottom: 10px;

}

.document-list {

list-style: none;

padding: 0;

}

.document-list li {

margin-bottom: 5px;

}

.document-list a {

text-decoration: none;

color: #007BFF;

}

/\* Action Buttons Styles \*/

.action-buttons {

margin-top: 20px;

}

.btn {

display: inline-block;

padding: 10px 20px;

text-decoration: none;

border: none;

border-radius: 3px;

cursor: pointer;

margin-right: 10px;

}

.btn-primary {

background-color: #007BFF;

color: #fff;

}

.btn-primary:hover {

background-color: #0056b3;

}

.btn-danger {

background-color: #FF0000;

color: #fff;

}

.btn-danger:hover {

background-color: #CC0000;

}

</style>

<script>

function viewAgentProfile() {

window.location.href = `/agent/${agent\_id}`; // Modify the URL as per your app's routing

}

// Assign the function to the Agent ID field

document.querySelector('span a').addEventListener('click', viewAgentProfile);

// Function to display worker details

function displayWorkerDetails() {

const workerDetails = document.getElementById("workerDetails");

workerDetails.innerHTML = `

<div class="profile-details">

<p>ID: <span>${worker.id}</span></p>

<p>Name: <span>${worker.first\_name}</span></p>

<p>Date of Birth: <span>${worker.dob}</span></p>

<p>Nationality: <span>${worker.nationality}</span></p>

<p>Contact Number: <span>${worker.contact\_number}</span></p>

<p>Adhar Number: <span>${worker.adhar\_number}</span></p>

</div>

<div class="profile-image">

<img src="${worker.profile\_image}" alt="${worker.first\_name}'s photo" style="max-width: 200px;">

</div>

`;

}

function downloadDocument(url, name) {

// Function to download the document

const link = document.createElement('a');

link.href = url;

link.download = name;

link.click();

}

function verifyWorker() {

console.log("Worker verified");

}

function rejectWorker() {

console.log("Worker rejected");

}

</script>

</html>

**Worker\_list.html**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Workers Listing</title>

<style>

body {

font-family: Arial, Helvetica, sans-serif;

margin: 0;

padding: 0;

background-color: #f4f4f4;

}

header {

background-color: #333;

color: #fff;

text-align: center;

padding: 20px;

}

.container {

width: 80%;

margin: 0 auto;

padding: 20px;

background-color: #fff;

border: 1px solid #ccc;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

margin-top: 20px;

}

/\* Table Styles \*/

.table {

width: 100%;

}

.table th,

.table td {

text-align: left;

padding: 8px;

}

.table th {

background-color: #

color: #fff;

}

.table tr:nth-child(even) {

background-color: #f2f2f2;

}

.worker-list {

display: flex;

flex-wrap: wrap;

justify-content: space-between;

}

.worker-item {

background-color: #fff;

box-shadow: 0px 0px 5px rgba(0, 0, 0, 0.2);

margin: 10px 0;

padding: 20px;

border-radius: 5px;

width: calc(32% - 20px);

text-align: center;

}

.worker-item img {

max-width: 100%;

height: auto;

}

.worker-item h3 {

font-size: 20px;

margin: 10px 0;

}

.worker-item p {

color: #777;

}

</style>

</head>

<body>

<header>

<h1>Workers Listing</h1>

</header>

<div class="container">

<h1 class="mt-4 mb-4">Verified Workers</h1>

<input type="text" id="searchInput" onkeyup="searchWorkers()" placeholder="Search for workers..." title="Type in a name">

<table class="table">

<thead>

<tr>

<th>Photo</th>

<th>Name</th>

<th>Date of Birth</th>

<th>Nationality</th>

<th>Contact Number</th>

<th>Adhar Number</th>

<th>Work Assigned</th>

<th>Agent Contact</th>

<th>Work Permit</th>

</tr>

</thead>

<tbody>

{% for worker in workers %}

{% if worker.is\_verified %}

<tr>

<td>

{% if worker.profile\_image %}

<img src="{{ worker.profile\_image.url }}" alt="{{ worker.first\_name }}'s photo" style="max-width: 100px; max-height: 100px;">

{% else %}

No Profile

{% endif %}

</td>

<td>{{ worker.first\_name }}</td>

<td>{{ worker.dob }}</td>

<td>{{ worker.nationality }}</td>

<td>{{ worker.contact\_number }}</td>

<td>{{ worker.adhar\_number }}</td>

<td>

{% if worker.category %}

{{ worker.category.name }}

{% else %}

No assigned work

{% endif %}

</td>

<td>

{% if worker.agent %}

<a href="{% url 'agent\_contact' agent\_id=worker.agent.id worker\_id=worker.id %}" id="worker">{{ worker.agent.first\_name }}Agent</a>

{% else %}

No assigned agent

{% endif %}

</td>

<td>

{% if worker.work\_permit\_verified %}

<a href="{% url 'generate\_work\_permit\_pdf' worker.id %}" class="btn btn-primary" download>Download Work Permit</a>

{% endif %}

</td>

</tr>

{% endif %}

{% empty %}

<tr>

<td colspan="10">No registered workers yet.</td>

</tr>

{% endfor %}

</tbody>

</table>

</div>

</body>

<script>

function searchWorkers() {

// Declare variables

var input, filter, table, tr, td, i, txtValue;

input = document.getElementById("searchInput");

filter = input.value.toUpperCase();

table = document.querySelector(".table");

tr = table.getElementsByTagName("tr");

// Loop through all table rows and hide those that don't match the search query

for (i = 0; i < tr.length; i++) {

td = tr[i].getElementsByTagName("td");

for (var j = 0; j < td.length; j++) {

if (td[j]) {

txtValue = td[j].textContent || td[j].innerText;

if (txtValue.toUpperCase().indexOf(filter) > -1) {

tr[i].style.display = "";

break;

} else {

tr[i].style.display = "none";

}

}

}

}

}

</script>

</html>

**Agent\_contact.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Agent Contact</title>

<style>

body {

font-family: Arial, Helvetica, sans-serif;

margin: 0;

padding: 0;

background-color: #f4f4f4;

}

header {

background-color: #333;

color: #fff;

text-align: center;

padding: 20px;

}

.container {

width: 80%;

margin: 0 auto;

padding: 20px;

background-color: #fff;

border: 1px solid #ccc;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

margin-top: 20px;

}

.agent-details {

text-align: center;

margin-bottom: 20px;

}

.agent-details img {

max-width: 100px;

max-height: 100px;

border-radius: 50%;

}

.agent-details h2 {

font-size: 24px;

margin: 10px 0;

}

.agent-details p {

color: #777;

}

.chat-box {

border: 1px solid #ccc;

padding: 10px;

margin-top: 20px;

}

.chat-box textarea {

width: calc(100% - 20px);

padding: 10px;

margin-top: 10px;

resize: none;

}

.chat-box button {

background-color: #333;

color: #fff;

padding: 10px;

border: none;

cursor: pointer;

}

/\* Modal Styles \*/

#bookingModal {

display: none;

position: fixed;

top: 50%;

left: 50%;

transform: translate(-50%, -50%);

width: 300px;

padding: 20px;

background-color: #fff;

border: 1px solid #ccc;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

z-index: 1;

}

.modal-content {

text-align: center;

}

.modal-content label {

display: block;

margin-bottom: 10px;

font-weight: bold;

}

.modal-content input {

width: calc(100% - 20px);

padding: 10px;

margin-bottom: 15px;

border: 1px solid #ccc;

border-radius: 3px;

}

.modal-content select {

width: calc(100% - 20px);

padding: 10px;

margin-bottom: 15px;

border: 1px solid #ccc;

border-radius: 3px;

}

.modal-content button {

width: 100%;

background-color: #007BFF;

color: #fff;

padding: 10px;

border: none;

border-radius: 3px;

cursor: pointer;

}

</style>

</head>

<body>

<header>

<h1>Agent Contact</h1>

</header>

<div class="container">

<div class="agent-details">

<h2>{{ agent.name }}</h2>

<p>Contact: {{ agent.phone }}</p>

<p>Email: {{ agent.email }}</p>

</div>

<div class="chat-box">

<h5>Sent request to Agent For booking Worker</h5>

<button id="sendMessageBtn">Sent Request</button>

</div>

</div>

<!-- Modal -->

<div id="bookingModal">

<div class="modal-content">

<span class="close" id="closeModal">&times;</span>

<h2>Book Worker</h2>

<form id="bookingForm" action="{% url 'book\_worker' agent.id worker.id %}" method="post">

{% csrf\_token %}

<label for="duration">Duration:</label>

<input type="number" name="duration" id="duration" required>

<label for="durationUnit">Duration Unit:</label>

<select name="durationUnit" id="durationUnit" required>

<option value="days">Days</option>

<option value="months">Months</option>

</select>

<button type="submit">Book Worker</button>

</form>

</div>

</div>

<script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>

<script>

$(document).ready(function() {

$('#sendMessageBtn').click(function() {

$('#bookingModal').css('display', 'block');

});

$('#closeModal').click(function() {

$('#bookingModal').css('display', 'none');

});

});

</script>

</body>

</html>

**Workcategory.html**

{% extends "adminpanel.html" %}

{% block title %}Worker{% endblock %}

{% block content %}

<!DOCTYPE html>

<html>

<head>

<title>Work Categories</title>

<link rel="stylesheet" type="text/css" href="styles.css">

</head>

<body>

<div class="container">

<h1>Work Categories</h1>

<div class="work-categories">

<form id="addCategoryForm" method="post" >

{% csrf\_token %}

<div class="form-group">

<input type="text" name="category-name" id="category-name" class="form-control" placeholder="Category Name">

</div>

<div class="form-group">

<textarea name="category-description" id="category-description" class="form-control" placeholder="Category Description"></textarea>

</div>

<button type="submit" class="btn btn-primary">Add Category</button>

</form>

<table class="table">

<thead>

<tr>

<th>Category Name</th>

<th>Description</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

{% for category in categories %}

<tr>

<form method="post" action="{% url 'edit\_category' category.id %}">

{% csrf\_token %}

<td><input type="text" name="name" value="{{ category.name }}"></td>

<td><input type="text" name="description" value="{{ category.description }}"></td>

<td>

<button type="submit" class="btn btn-sm btn-info">Edit</button>

</td>

</form>

<td>

<a href="{% url 'delete\_category' category.id %}" class="btn btn-sm btn-danger">Delete</a>

</td>

</tr>

{% endfor %}

</tbody>

</table>

</div>

</div>

<script src="scripts.js"></script>

</body>

<style>

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

background-color: #f4f4f4;

}

.container {

width: 80%;

margin: 0 auto;

background-color: #fff;

padding: 20px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

h1 {

color: #333;

margin-bottom: 20px;

}

.form-group {

margin-bottom: 15px;

}

.form-control {

width: 100%;

padding: 8px;

font-size: 16px;

border: 1px solid #ccc;

border-radius: 4px;

}

button {

padding: 10px 20px;

background-color: #337ab7;

color: #fff;

border: none;

border-radius: 4px;

cursor: pointer;

transition: background-color 0.3s;

}

button:hover {

background-color: #23527c;

}

.work-categories ul {

list-style: none;

padding: 0;

}

.work-categories ul li {

background-color: #f9f9f9;

padding: 10px;

margin-bottom: 5px;

border-radius: 4px;

box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);

}

</style>

</html>

{% endblock content %}

**Verify\_user.html**

{% extends "adminpanel.html" %}

{% block title %}Worker{% endblock %}

{% block content %}

<!DOCTYPE html>

<html>

<head>

<title>User Profile Verification</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 20px;

}

.container {

max-width: 800px;

margin: auto;

background: #f9f9f9;

padding: 20px;

border-radius: 8px;

}

h1 {

text-align: center;

margin-bottom: 20px;

}

h2 {

margin-bottom: 10px;

}

form {

margin-top: 20px;

}

.document-link {

display: block;

margin-top: 10px;

text-decoration: underline;

color: blue;

}

</style>

</head>

<body>

<div class="container">

<h1>User Profile Verification</h1>

<p>Username: <span>{{ user.username }}</span></p>

<p>Email: <span>{{ user.email }}</span></p>

<p>Contact Number: <span>{{ user.phone }}</span></p>

<!-- Display other user information as needed -->

<div>

{% if user.is\_employer %}

Employer

{% endif %}

{% if user.is\_agent %}

Agent

{% endif %}

{% if user.is\_police %}

Police

{% endif %}

</div>

<div>

<h3>Document:</h3>

{% if user.uploaded\_file %}

<li>

<a href="{{ user.uploaded\_file.url }}">View Document</a>

<a href="{{ user.uploaded\_file.url }}" download="{{ user.uploaded\_file.name }}">Download</a>

</li>

{% endif %}

</div>

<div class="action-buttons">

<form action="/verify\_user/{{ user.id }}/" method="post">

{% csrf\_token %}

<button class="btn btn-primary" type="submit">Verify</button>

</form>

<form action="/rejectuser/{{ user.id }}/" method="post">

{% csrf\_token %}

<button class="btn btn-danger" type="submit">Reject</button>

</form>

</div>

</div>

</body>

</html>

{% endblock content %}

## Screen Shots

## Home page

## 

## 1.Employer

## 1.1 userpage

## 

## 1.2 worker\_list page

## 

## 1.3 Agent contact(booking worker)page

## 

## 1.4 bookings list page

## 

## 2.Agent

## 2.1 agent page

## 

## 2.2 worker registration

## 

## 2.3 view workers

## 

## 2.4 notifications of booking

## 

## 3.Police

## 3.1 police page

## 

## 3.2 workers list

## 

## 3.3 view profile (for verification)

## 

## 3.4 incidence

## 

## 3.5 Active officers list

## 

## 4.Admin

## 4.1 admin panel

## 

## 4.2 User verification

## 

## 4.3 Adding work category

## 