WATERPROOFING MANAGEMENT SYSTEM

THE PROJECT REPORT

SUBMITTED TO

THE APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

MASTER OF COMPUTER APPLICATIONS

by

NIYAS MOHAMMED BASHEER

(**SNG23MCA-2064**)

UNDER THE GUIDANCE OF

Ms. MEERA K CHANDRAN

ASSISTANT PROFESSOR

Department of Computer Applications

Sree Narayana Gurukulam College of Engineering (SNGCE)

Kadayiruppu

May, 2025

PROJECT REPORT

WATERPROOFING MANAGEMENT SYSTEM

Submitted by
NIYAS MOHAMMED BASHEER
Reg. No (**SNG23MCA-2064**)

under the guidance of

Ms Meera K Chandran
Asst.Professor

to

the APJ Abdul Kalam Technological University
in partial fulfilment of the requirements for the award of the Degree

of

Master of Computer Applications

Department of Computer Applications

Sree Narayana Gurukulam College of Engineering (SNGCE)
Kadayiruppu

May, 2025

DECLARATION

I, the undersigned, hereby declare that the project report titled “**Waterproofing Management System**”, submitted in partial fulfillment of the requirements for the award of the degree of Master of Computer Applications of APJ Abdul Kalam Technological University, Kerala, is a bonafide work carried out by me under the supervision of Asst. Prof. Meera K. Chandran.

This submission represents my original work and ideas. Where ideas, data, or words of others have been used, they have been duly cited and referenced. I affirm that I have upheld the principles of academic honesty and integrity and have not misrepresented or fabricated any data, fact, source, or idea in this report.

I understand that any violation of the above will be subject to disciplinary action by the institution and/or the University, and may also attract legal consequences from the rightful sources not properly acknowledged or cited. I further declare that this report has not been submitted previously for the award of any degree, diploma, or similar title in any other university or institution.

Place :	Signature
Date :	NIYAS MOHAMMED BASHEER

DEPARTMENT OF COMPUTER APPLICATIONS
SREE NARAYANA GURUKULAM COLLEGE OF ENGINEERING
(SNGCE)
KADAYIRUPPU.

CERTIFICATE

This is to certify that the report entitled "**WATERPROOFING MANAGEMENT SYSTEM**" submitted by "**NIYAS MOHAMMED BASHEER**" to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications is a bonafide record of the project work carried out by him/her under my/our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

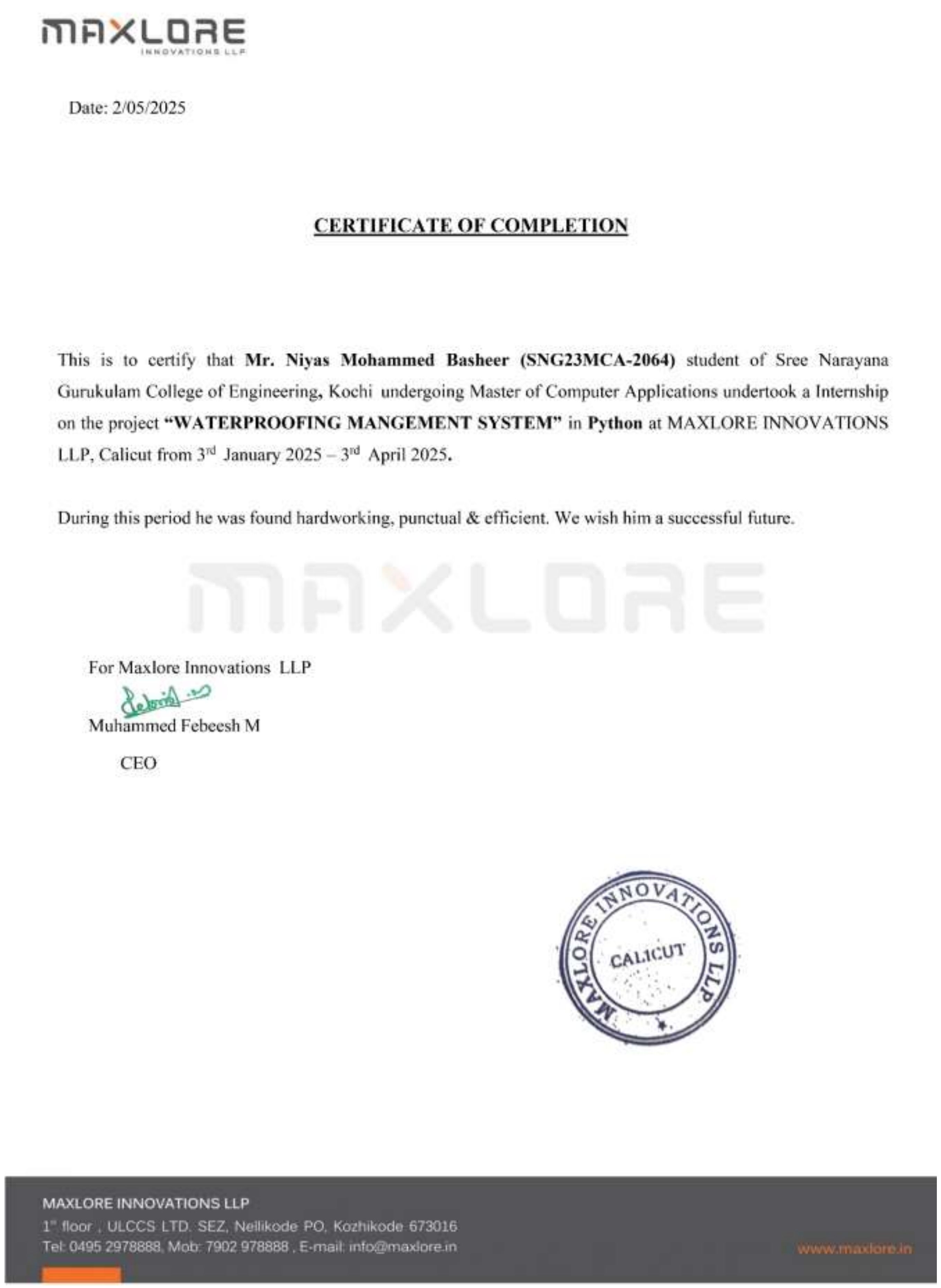
External Supervisor   Internal Supervisor (SNG)

Project Coordinator   Head Of The Department

Submitted to ______________examinations held at ___________________________on_________

Internal Examiner   External Examiner

(Seal)



MAXLORE
INNOVATIONS LLP

Date: 2/05/2025

CERTIFICATE OF COMPLETION

This is to certify that **Mr. Niyas Mohammed Basheer (SNG23MCA-2064)** student of Sree Narayana Gurukulam College of Engineering, Kochi undergoing Master of Computer Applications undertook a Internship on the project **"WATERPROOFING MANGEMENT SYSTEM"** in **Python** at MAXLORE INNOVATIONS LLP, Calicut from 3rd January 2025 – 3rd April 2025.

During this period he was found hardworking, punctual & efficient. We wish him a successful future.

For Maxlore Innovations LLP

[illegible]

Muhammed Febeesh M

CEO

MAXLORE INNOVATIONS LLP
CALICUT

MAXLORE INNOVATIONS LLP
1st floor, ULCCS LTD. SEZ, Nellikode PO, Kozhikode 673016
Tel: 0495 2978888, Mob: 7902 978888 , E-mail: info@maxlore.in

www.maxlore.in

Contents

Contents

VISION:

1. Unequaled Services with Unfolding Technologies

MISSION:

Our mission is to:

1. Foster a culture of innovation, creativity, and entrepreneurship
2. Engage with industry and other stakeholders to adopt latest technological trends and do research.
3. Imbibing professional ethics, values and life skills

PEOs:

PEO 1:

Graduates will establish themselves as competent IT professionals, software developers, system analysts, or entrepreneurs by applying their technical knowledge, problem-solving skills, and innovative thinking in the field of computing and information technology.

PEO 2:

Graduates will uphold responsible research practices, promote innovative solutions and address societal challenges through technology-driven research, fostering inclusive development.

PEO 3:

Graduates will adhere and promote the ethical standards, cybersecurity principles, and sustainable computing practices while developing technology solutions for societal and business needs.

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to all those who have contributed to the successful completion of our academic mini project, "Development of a Task Management Application," undertaken as part of our curriculum at SNGCE affiliated with Kerala Technological University (KTU).

First and foremost, we wish to extend our sincere thanks to **Dr S Jose**, Principal of SNGCE, for fostering an environment that encourages innovation and learning. Their commitment to academic excellence has provided us with the resources and opportunities necessary for our growth.

We are also deeply grateful to, **Dr. Sandhya R**, Head of the Department of the Department of Computer Applications, for their unwavering support and guidance throughout this project. Their leadership and vision have greatly inspired us to strive for excellence.

Our special thanks go to our project guide, **Asst.Prof Meera K Chandran**, Head of the Department of the Department of Computer Applications whose expertise and mentorship were invaluable throughout the development process. Their insights and constructive feedback have helped us refine our approach and achieve our project objectives.

We would like to acknowledge the contributions of other faculty members, including, who provided us with essential knowledge and support during various stages of our project. Their dedication to teaching has significantly enriched our learning experience.

We are profoundly thankful to **God Almighty** for granting us the strength, wisdom, and perseverance needed to complete this project. We believe that faith has played a crucial role in guiding us through challenges. Lastly, we wish to express our heartfelt appreciation to our parents for their unconditional love, encouragement, and support throughout our academic journey. Their belief in us has been a constant source of motivation.

This project has not only enhanced our technical skills but also taught us valuable lessons in teamwork and project management. We are excited to carry these experiences forward into our future endeavours.

ABSTRACT

The **Waterproofing Management System** has been conceptualized and developed as a comprehensive **Enterprise Resource Planning (ERP) solution**, specifically tailored for companies operating in the waterproofing industry. Recognizing the increasing importance of construction and infrastructure protection, the demand for efficient waterproofing services has grown substantially. Manual methods of managing operations in this sector often result in inefficiencies, delays, mismanaged inventories, and unsatisfactory customer service. This ERP system addresses these challenges by offering an integrated platform that centralizes customer relationship management, job allocation, inventory control, and communication between various stakeholders.

The system incorporates four principal user roles: **Administrator**, **Administrative Staff**, **Dealers**, and **Workers**, each provided with a customized dashboard and role-specific functionalities. The Administrator is responsible for overseeing the complete system, including user verification, product inventory management, complaint and feedback resolution, and the assignment of workers to job requests. Administrative Staff support these operations by managing inventory updates, handling worker assignments, and monitoring the statuses of purchase and job orders. Dealers interact with the system by registering, viewing and ordering products, requesting waterproofing services, and utilizing the complaint and feedback modules. Workers, on the other hand, receive task assignments, update job statuses in real-time, and track their daily operational activities through the system.

By deploying this ERP solution, waterproofing companies can substantially improve their operational efficiency, reduce manual errors, optimize resource allocation, and deliver superior customer service Overall, the Waterproofing Management System as an ERP framework offers a modernized, scalable, and efficient digital infrastructure to meet the evolving operational needs of the waterproofing industry.

LIST OF FIGURES

ABBREVIATIONS

Abbreviation	Full Form
ERP	Enterprise Resource Planning
SDLC	Software Development Life Cycle
UI	User Interface
UAT	User Acceptance Testing
VCS	Version Control System
Git	Global Information Tracker
API	Application Programming Interface
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
DB	Database
MySQL	My Structured Query Language
CRUD	Create, Read, Update, Delete
SMTP	Simple Mail Transfer Protocol
IDE	Integrated Development Environment
VS Code	Visual Studio Code
UML	Unified Modeling Language
CI	Continuous Integration
DoD	Definition of Done
PR	Pull Request
REQ	Requirement

NOTATION

Symbol	**Notation / Meaning**
○	Start of activity (Initial Node)
●	End of activity (Final Node)
▭	Activity (Action or Process Step)
◆	Decision/Branch (Conditional Flow)
→	Control Flow (Arrow indicates sequence)

SCRUM PROJECT

INTRODUCTION TO SCRUM

Scrum is an Agile framework designed for managing complex projects, particularly in software development. It emphasizes iterative progress through a series of time-boxed iterations known as sprints. Scrum promotes collaboration, accountability, and continuous improvement.

OVERVIEW OF ACADEMIC MINI PROJECTS USING SCRUM AT APJA-KTU

Project Structure: Students typically form small teams to work on a mini project that addresses a specific problem or need within their domain of study. The projects can range from software development to research initiatives, allowing students to apply theoretical knowledge in practical scenarios.

Roles and Responsibilities: Each team member assumes specific roles within the Scrum framework:

- **Product Owner:** If there is a customer for the project then he/she will be the Product Owner (External Guide/ HoD) and a faculty from the department will be the Internal Guide. If there is no such customer then the Internal Guide himself/herself shall act as the Product Owner. This role is usually filled by those who represents the stakeholders' interests, prioritizes the project backlog, and defines the project vision.
- **Scrum Master: A faculty / technical staff shall act as the Scrum Master to continuously monitor the project development. Periodic meetings, of less than 15 minutes, at the convenience of the Scrum Master are to be highly encouraged. SM facilitates Scrum practices, helps resolve impediments, and ensures the team adheres to Agile principles.**
- **Development Team:** Comprised of students with diverse skill sets, this group collaborates to deliver project increments during each sprint.

Execution Process

The project is executed in iterative cycles called sprints, typically lasting 2-4 weeks. Each sprint begins with planning sessions where user stories from the backlog are selected for development. Daily stand-up meetings foster communication and allow team members to discuss progress and challenges. At the end of each sprint, a review meeting is held to demonstrate completed work and gather feedback from stakeholders.

LEARNING OUTCOMES

By using Scrum for their mini projects, students at KTU gain hands-on experience in Agile methodologies, enhance their teamwork skills, and improve their ability to manage time and resources effectively. This practical application of Scrum not only prepares them for future careers but also reinforces their understanding of project management principles.

Course Outcomes: After the completion of the course the student will be able to

CO No.	Course Outcome (CO)	Bloom's Category Level
CO 1	Identify a real-life project that is useful to society/industry	Level 2: Understand
CO 2	Interact with people to identify the project requirements	Level 3: Apply
CO 3	Apply suitable development methodology for the development of the product/project	Level 3: Apply
CO 4	Analyse and design a software product/project	Level 4: Analyse
CO 5	Test the modules at various stages of project development	Level 5: Evaluate
CO 6	Build and integrate different software modules	Level 6: Create
CO 7	Document and deploy the product/project	Level 3: Apply

Mapping of course outcomes with program outcomes

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO 1	2	3	3	3	1	2	3	3	3	3	3	3
CO 2	2	3	2	3	2	3	2	1	3	2	3	
CO 3	3	3	3	3	3	1	3	3	1	2		
CO 4	3	3	3	3	3	3	3	1	1	2		
CO 5	3	3	3	3	3	2	3	1				
CO 6	3	3	3	3	3	2	3	3	2	3	3	
CO 7	1	1	3	3	3	2	3	3	2	1	2	

3/2/1: High/Medium/Low

TEAM ROLES AND RESPONSIBILITIES

In an academic project utilizing the Scrum framework, the roles and responsibilities of the team members are crucial for ensuring effective collaboration and successful project outcomes. The Scrum team is composed of three primary roles: **Product Owner**, **Scrum Master**, and **Development Team**.

Product Owner

The Product Owner acts as the voice of the stakeholders and is responsible for defining the vision of the project. This role involves managing and prioritizing the product backlog, which is a dynamic list of features, enhancements, and fixes that need to be addressed. The Product Owner ensures that the team focuses on delivering maximum value by clearly articulating user stories and acceptance criteria, thus guiding the Development Team on what needs to be built next.

Scrum Master

The Scrum Master serves as a facilitator and coach for the Scrum team. Their primary responsibilities include ensuring that Scrum practices are followed, helping to remove any impediments that may hinder the team's progress, and fostering an environment conducive to collaboration and continuous improvement. The Scrum Master also protects the team from external distractions and encourages open communication among team members, ensuring that everyone understands their roles within the Scrum process.

Development Team

The Development Team consists of cross-functional members who are responsible for delivering potentially shippable increments of the product at the end of each sprint. This team is self-organizing, meaning they have the autonomy to decide how best to accomplish their work. Members typically include software developers, designers, testers, and other specialists who collaborate closely to meet sprint goals. The Development Team is accountable for maintaining high-quality standards and adhering to the "Definition of Done" for each increment.

List of Team Members

- **Product Owner: NAVEEN JHON THOMAS**
  (Senior Developer – Maxlore Innovation , Calicut)
- **Scrum Master: MEERA K CHANDRAN**
  (Asst.Professor Department Of Computer Application)
- **Development Team Members: NIYAS MOHAMMED BASHEER**

CHAPTER 1. INTRODUCTION

1.1 PROJECT OVERVIEW

Brief Description of the Project

The project entitled **"Waterproofing Management System"** is a comprehensive enterprise solution developed to streamline and automate the operational activities of companies in the waterproofing industry. This ERP system integrates critical functionalities to assist waterproofing service providers in effectively managing customer data, job allocations, inventory tracking, worker coordination, and dealer interactions. The platform offers a centralized framework where administrators, administrative staff, dealers, and workers can collaborate seamlessly.

The primary objective of this system is to enhance service delivery by increasing operational transparency, minimizing manual errors, optimizing material management, and improving customer satisfaction. The ERP system supports real-time tracking of job progress, accurate inventory monitoring, warranty management, and promotes effective communication among all stakeholders involved in the waterproofing services lifecycle.

1.2 OBJECTIVES

- Centralized Management of Waterproofing Services
- Customer Relationship Management (CRM) Integration
- Dealer Registration, Product Ordering, and Service Request Handling
- Worker Allocation and Real-time Job Status Updates
- Inventory and Material Tracking
- Complaint and Feedback Management System
- Role-Based Access Control (RBAC) for Secure Operation
- Interactive and Intelligent Dashboard for Administrators and Staff

1.3 SCOPE

The **Waterproofing Management System** is specifically designed to streamline the operational workflows of waterproofing companies by providing centralized tools for managing customer interactions, job allocations, material inventories, dealer registrations, and worker assignments. The system ensures secure and efficient operations through role-based access control, providing differentiated access to administrators, administrative staff, dealers, and workers.

Furthermore, the platform is scalable to support multiple dealers, projects, and workforce management, and is capable of integration with future modules such as payment gateways, real-time location tracking, and automated notification systems. The ERP system significantly reduces manual effort, enhances data security, and improves decision-making processes, ultimately contributing to superior service delivery and organizational growth.

CHAPTER 2. REQUIREMENTS/ SURVEYS

Before initiating the development of the **Waterproofing Management System**, a comprehensive requirements analysis was conducted to understand the operational needs and expectations of all stakeholders, including administrators, administrative staff, dealers, and workers.

2.1PROBLEM SCENARIO

In the traditional management of waterproofing services, numerous challenges arise due to the reliance on manual tracking methods, inefficient material management, and the absence of real-time collaboration among stakeholders. Administrators and staff often encounter difficulties in monitoring job progress, managing inventory levels, and coordinating worker assignments across various service locations. Additionally, dealers and workers face delays in communication, task allocations, and service tracking due to disjointed operational workflows. These inefficiencies contribute to increased operational costs, resource wastage, customer dissatisfaction, and delays in service delivery.

2.2 HARDWARE REQUIREMENTS

- A computer with at least an Intel Core i3 processor or higher.
- Minimum 4 GB RAM.
- Minimum 500 GB Hard Disk Drive (HDD) or SSD.
- A 14-inch LCD monitor or larger.
- Standard keyboard and mouse.
- Android-supported mobile device for field operations.
- Stable internet connection for deployment, testing, and live synchronization.

2.3 SOFTWARE REQUIREMENTS

- Operating System: Windows 8 or above.
- Frontend Technologies: HTML, CSS, Bootstrap.
- Backend Technologies: Python, Java.
- Database Management System: MySQL.
- Development Tools: Android Studio, JetBrains PyCharm, MySQL Workbench.

2.4 FUNCTIONAL REQUIREMENTS

- User registration and login functionality for Admin, Administrative Staff, Dealers, and Workers.
- Dealer ability to view products, place orders, and request waterproofing services.
- Administrative Staff capability to update stock, manage workers, and update job order statuses.
- Admin functionalities to approve registrations, allocate workers, and manage feedback and complaints.
- Worker ability to view assigned tasks and update job statuses in real time.Complaint and feedback modules for seamless communication.

2.5 NON-FUNCTIONAL REQUIREMENTS

- High performance and low latency under traffic.
- Secure communication with HTTPS and token-based authentication.
- System scalability to handle increased user base.
- Cross-platform compatibility (desktop, mobile browsers).
- 99.9% system uptime.

2.6 PRODUCT FUNCTIONS

The ERP system will allow users to:

- Register and authenticate based on their roles (Admin, Staff, Dealer, Worker).
- View, browse, and order waterproofing products.
- Submit job requests and track the progress of service delivery.
- Allocate and manage workers for specific job assignments.
- Monitor real-time inventory and material usage.
- Manage complaints and feedback from dealers and customers.
- Generate reports related to inventory status, job progress, and customer interactions.

CHAPTER 3. ANALYSIS

Examination of the Problem Scenario Traditional methods of managing waterproofing services often lack efficiency, real-time tracking capabilities, and seamless communication between stakeholders. While certain business management tools exist, they generally fail to address the unique operational demands of waterproofing companies, such as material tracking, job allocation, warranty management, and dealer interactions. As a result, companies encounter challenges in ensuring timely service delivery, effective resource utilization, and customer satisfaction.

Identification of Key Stakeholders and Their Needs

- **Dealers:** Require an intuitive interface to view products, place orders, and track service requests easily.
- **Administrative Staff:** Need efficient tools for managing stock, approving service requests, and assigning tasks to workers.
- **Workers:** Seek a streamlined method to view assigned jobs, update work progress, and report issues.
- **Admins:** Require complete oversight of operations, including user verification, inventory management, complaint handling, and report generation.

3.1 FEASIBILITY ANALYSIS

A thorough feasibility study was conducted, highlighting the need for an integrated waterproofing management system capable of supporting real-time operations, secure communication, and role-based access. The feasibility aspects are summarized as follows:

- **Technical Feasibility:** The technologies selected (Python, MySQL) are reliable, scalable, and well-suited to developing a robust ERP solution.

- **Operational Feasibility:** The proposed system aligns with the day-to-day operational needs of waterproofing companies and improves coordination among multiple user groups.
- **Economic Feasibility:** The system minimizes manual workload and errors, thus reducing overall operational costs and enhancing profitability.
- **Schedule Feasibility:** The development timeline is practical, given the modular design approach and familiar development tools.

3.2 PROBLEM STORY

Waterproofing companies frequently face operational issues such as untracked inventory usage, delayed job scheduling, inefficient allocation of workers, and a lack of centralized warranty and service history management. Without a comprehensive system, administrative staff and dealers encounter difficulty in communicating requirements and resolving complaints promptly, leading to operational bottlenecks and diminished customer service quality.

3.3 USE CASE STORY

An Admin logs into the system to access a dashboard displaying key metrics such as pending dealer verifications, active complaints, available product inventory, and worker availability. They verify a newly registered dealer, approve their product order, and allocate a worker for a requested waterproofing service. Simultaneously, Administrative Staff update stock levels after reviewing dealer purchase requests. Dealers browse available products, place orders, and submit service requests through an intuitive interface. Workers view their assigned jobs on a mobile application, mark task progress, and report job completion updates to the system.

3.4 USE CASE DIAGRAMS

The use case diagram visually represents the interactions between the system and its primary actors:

- Admin: Manage users, approve requests, assign workers, respond to feedback and complaints.
- Administrative Staff: Manage stock, view and update job statuses, oversee purchases.
- Dealer: Browse products, place orders, request services, submit complaints, and feedback.
- Worker: View assigned tasks and update work status.

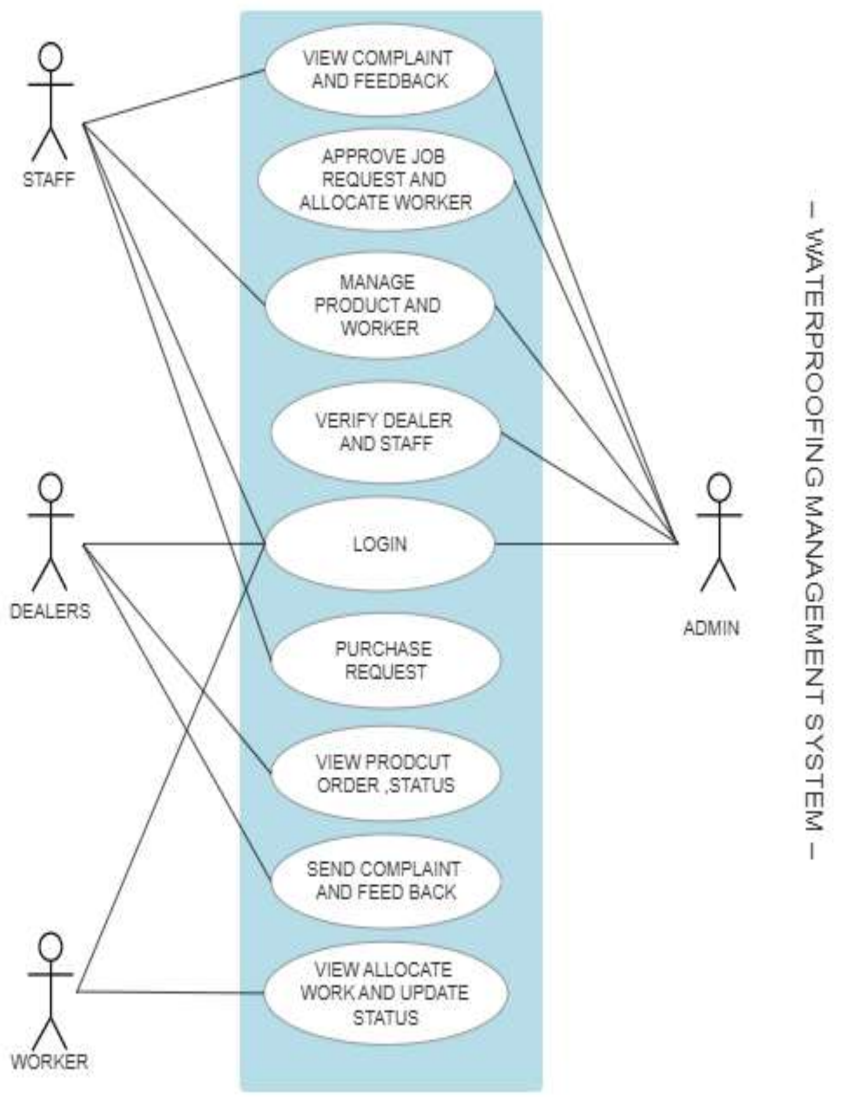

STAFF
DEALERS
WORKER
ADMIN
VIEW COMPLAINT AND FEEDBACK
APPROVE JOB REQUEST AND ALLOCATE WORKER
MANAGE PRODUCT AND WORKER
VERIFY DEALER AND STAFF
LOGIN
PURCHASE REQUEST
VIEW PRODCUT ORDER ,STATUS
SEND COMPLAINT AND FEED BACK
VIEW ALLOCATE WORK AND UPDATE STATUS
– WATERPROOFING MANAGEMENT SYSTEM –

Fig.3.1(use case diagram)

3.5USER STORIES

High-level system requirements, expressed through user stories, include:

Admin User Stories

1. **As an Admin**, I want to log in securely, so that I can manage and monitor the system.
2. **As an Admin**, I want to verify dealers and staff, so that only authorized users can access system features.
3. **As an Admin**, I want to add and manage product listings, so that accurate inventory is available for dealers.
4. **As an Admin**, I want to add and manage workers, so that jobs can be assigned effectively.
5. **As an Admin**, I want to view and respond to complaints, so that user issues are addressed promptly.
6. **As an Admin**, I want to view feedback, so that I can improve system quality.
7. **As an Admin**, I want to approve job requests and assign them to workers, so that work is executed efficiently.

Administrative Staff User Stories

1. **As administrative staff**, I want to register and wait for approval, so that I can use the system after validation.
2. **As administrative staff**, I want to log in to the system, so that I can perform my management duties.
3. **As administrative staff**, I want to update product stock, so that inventory levels stay current.
4. **As administrative staff**, I want to manage worker records, so that assignments can be organized.
5. **As administrative staff**, I want to view and update purchase and job request statuses, so that work and orders progress smoothly.
6. **As administrative staff**, I want to respond to complaints, so that customer satisfaction is maintained.
7. **As administrative staff**, I want to view feedback, so that I understand user experiences.

Dealer User Stories

1. **As a dealer**, I want to register and get verified, so that I can access and use the platform.
2. **As a dealer**, I want to log in, so that I can view products and place orders.
3. **As a dealer**, I want to view product listings and order materials, so that I can get supplies for my work.
4. **As a dealer**, I want to check the status of my product orders, so that I can track delivery.
5. **As a dealer**, I want to send job requests to waterproofing companies, so that work can be initiated.
6. **As a dealer**, I want to check the status of my job requests, so that I can monitor progress.
7. **As a dealer**, I want to track assigned workers, so that I can know who is working on my projects.
8. **As a dealer**, I want to send complaints and receive replies, so that my issues are addressed.
9. **As a dealer**, I want to send feedback, so that I can help improve service quality.

Worker User Stories

1. **As a worker**, I want to log in to the system, so that I can access my job assignments.
2. **As a worker**, I want to view the details of my allocated jobs, so that I know what work to do.
3. **As a worker**, I want to update the job status after completing work, so that the system stays updated.

3.6 TECHNICAL DOCUMENTATION

The technical implementation details for the Waterproofing Management ERP System are as follows:

- **Backend Development**: Python .
- **Frontend Development**: HTML, CSS, Bootstrap for web interface.
- **Database**: MySQL, structured with entities such Dealers, Workers, Products, Complaints, and Job Requests.
- **Version Control**: GitHub .
- **Hosting and Deployment**: :Localhost

CHAPTER 4. PLANNING

Effective planning plays a vital role in the success of any enterprise software development project. For the Waterproofing Management ERP System, the development process was broken down into distinct sprints to ensure systematic execution, improved clarity, and timely delivery. Each phase was carefully aligned with stakeholder needs and system objectives.

4.1 PRODUCT BACKLOG

The product backlog encompasses all the essential features and services intended to be delivered throughout the development cycle. These backlog items were formulated based on stakeholder interviews, requirement analysis, and operational workflows. Each item is prioritized according to business impact and technical feasibility.

Sample Product Backlog Items:

1. User registration, login, and authentication
2. Role-based dashboards (Admin, Dealer, Worker, Staff)
3. Inventory and product management module
4. Job request
5. Worker assignment
6. Complaint handling system
7. Feedback handling system
8. Admin monitoring and reporting panel

4.2 SPRINT PLANNING DOCUMENTS

Sprint planning involved identifying user stories from the product backlog that were achievable within the sprint duration. Each sprint focused on delivering a functional module that contributed to the larger system, ensuring measurable outcomes and consistent development flow.

Sample Sprint Plan – Sprint 1

- **Sprint Goal:** Implement core user authentication and onboarding
- **Selected Backlog Items:**
    - Setup login and registration
    - Design and develop user profile management interface

4.3 SPRINT BACKLOG

The sprint backlog outlines the focused objectives and key deliverables within each sprint, including major development tasks and intended functionality.

Sprint	**Goal**	**Key Tasks**
Sprint 1	Define system goals and setup authentication	- Finalize project scope - Setup registration, login modules - Implement session handling
Sprint 2	Requirements finalization and stakeholder analysis	- Conduct requirement sessions - Identify stakeholders - Create use case diagrams - Build initial product backlog
Sprint 3	Design database and implement role-based user access	- Structure database schema - Define roles (Admin, Staff, Dealer, Worker) - Implement basic role-based access control
Sprint 4	Develop dealer dashboard and product inventory system	- Create interface for product listing and orders - Enable inventory tracking
Sprint 5	Implement Product Management and Tracking Module	- Develop product catalog for waterproofing materials - Enable product addition, editing, and categorization
Sprint 6	Build job assignment and project monitoring module	- Assign tasks to workers - Track job completion and status - Admin analytics for task progress
Sprint 7	Testing, delivery scheduling, and deployment	- Implement delivery scheduling interface

Table 4.1(Sprint Backlog)

4.4 PROJECT PLAN

The full project lifecycle spanned from **January to April 2025**, structured into the following milestones:

- **Sprint 1**: User authentication (Week 1–2)
- **Sprint 2**: Requirement gathering and database design (Week 3–4)
- **Sprint 3**: Design database schema and implement user authentication (Week 5–6)
- **Sprint 4**: Develop vendor dashboard and material management system (Week 7–8)
- **Sprint 5**: Implement product management module (Week 9–10)
- **Sprint 6**: Build project management dashboard and admin monitoring (Week 11–12)
- **Sprint 7**: Testing and deployment (Week 13–14)

Resource Allocation:

- **Backend**: Niyas Mohammed Basheer
- **Frontend/UI**: Niyas Mohammed Basheer
- **Database & Testing**: Niyas Mohammed Basheer
- **Documentation & Deployment**: Niyas Mohammed Basheer

4.5 BURNDOWN CHART

The ideal burndown chart visually illustrates the progression of sprint goals versus the time spent, helping track task completion efficiency. A smooth downward trajectory in the chart indicates that tasks are being completed at a consistent pace. Irregularities in the curve suggest potential delays or underestimated tasks that require attention.

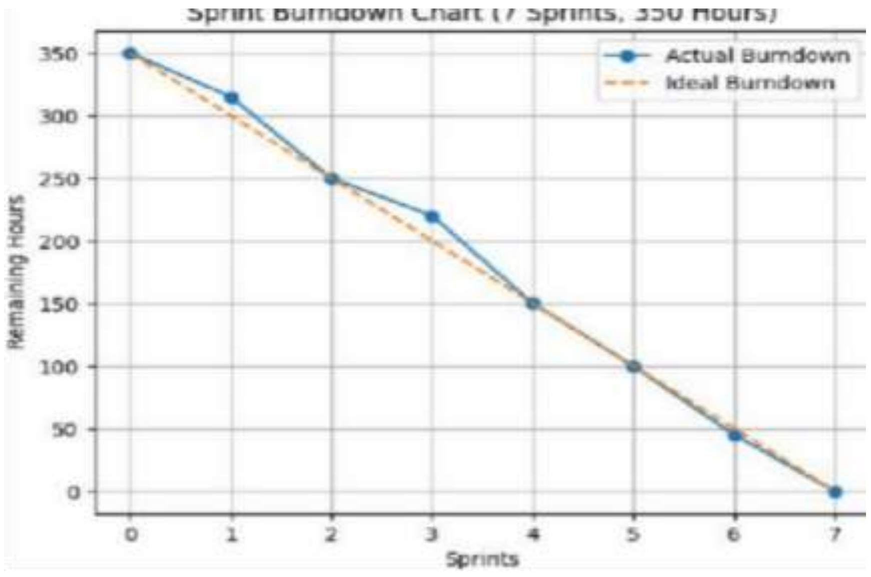


Fig.4.1(Burn downchart)

CHAPTER 5: DESIGNS

System design is a crucial phase that outlines how the software will function in terms of structure, flow, and user interaction. It serves as the bridge between requirement analysis and system development. For the **Waterproofing Management System**, several design strategies were implemented to represent the architecture of the application clearly and efficiently. These include the design of the user interface, database schema, process flows, and system behaviors.

5.1 USER INTERFACE DESIGN (UID)

The User Interface Design is structured to ensure the system is user-friendly, responsive, and functional for all user roles such as admins, dealers, and workers. Wireframes were developed to lay out the interface elements and demonstrate the navigation across various modules.

UID Highlights:

- **Home Page**: Displays company branding, a list of available waterproofing services, login options for dealers, workers, and admin.
- **Dealer Dashboard**: Enables dealers to submit job and purchase requests, view work progress, and lodge complaints or feedback.
- **Worker Panel**: Allows assigned workers to view task details, mark attendance, update task status, and manage their personal information.
- **Admin Panel**: Provides administrative access to allocate work, review job requests, manage users, and monitor payment records and system usage.
- **Work Allocation Form**: Simplified interface to assign tasks to workers based on job requests, including start and end dates.

5.2 DATA DESIGN DOCUMENTATION

The **Data Design** describes how data is structured, stored, and related throughout the system. The database was odelled using Django ORM and captured using an **Entity-Relationship Diagram (ERD)**, reflecting actual system entities and their relationships.

Key Entities from ERD:

- **login_table**: Stores all user login credentials and their roles (admin, dealer, worker).
- **dealers_table**: Captures dealer-specific data including contact details and ID proof.
- **workers_table**: Maintains worker profiles, skills, availability, and proof documentation.
- **product_table**: Stores inventory items including pricing, description, and stock levels.
- **job_request_table**: Submitted by dealers to request specific construction tasks.
- **allocate_work**: Tracks work assignments to workers along with dates and current status.
- **purchase_request_table**: Dealer-submitted requests for materials.
- **purchase_request_more_table**: Links multiple products and quantities to a purchase request.
- **feedback_table**: Stores feedback provided by dealers for products.
- **complaint_table**: Logs service or system-related complaints submitted by dealers.
- **payment**: Logs purchase request payments along with timestamps and status.

1. **login_table**

Field	**Type**	**Description**
id	AutoField	Primary key
username	CharField	Username for login
password	CharField	Hashed password
type	CharField	User role (admin, dealer, worker)

Table 5.1

2. dealers_table

Field	Type	Description
id	AutoField	Primary key
LOGIN	ForeignKey	Linked login_table entry
name	CharField	Full name
contact	BigIntegerField	Contact number
email	CharField	Email address
place, post, pin	CharFields/Int	Address fields
id_proof_number	BigIntegerField	National ID or similar
image	FileField	Uploaded ID image

Table 5.2

3. job_request_table

Field	Type	Description
id	AutoField	Primary key
DEALEARS	ForeignKey	Linked dealer
work	CharField	Work title
description	TextField	Detailed request info
date	DateField	Submission date

Table 5.3

4. allocate_work

Field	Type	Description
id	AutoField	Primary key
JOB_REQUEST	ForeignKey	Linked job request
WORKER	ForeignKey	Assigned worker
start_date	DateField	Work start date
end_date	DateField	Expected completion date
allocation_date	DateField	Date of assignment
status	CharField	Current work status

Table 5.4

5. admin_table

Field Name	Type	Description	Constraints
id	AutoField	Unique identifier	Primary Key
LOGIN	ForeignKey	Linked to login_table	Not Null, FK(login_table.id)
name	CharField	Admin's name	Not Null
contact	BigIntegerField	Phone number	Not Null
email	CharField	Email address	Not Null, Unique

Table 5.5

6. workers_table

Field Name	Type	Description	Constraints
id	AutoField	Unique identifier	Primary Key
LOGIN	ForeignKey	Linked to login_table	Not Null, FK(login_table.id)
name	CharField	Worker's full name	Not Null
contact	BigIntegerField	Phone number	Not Null
email	CharField	Email address	Not Null, Unique
place	CharField	Address	Not Null
post	CharField	Postal area	Not Null
pin	IntegerField	Pin code	Not Null
idproof	FileField	ID proof document upload	Optional

Table 5.6

7. product_table

Field Name	Type	Description	Constraints
id	AutoField	Unique identifier	Primary Key
product	CharField	Name of the product	Not Null
details	TextField	Description	Not Null
rate	IntegerField	Price	Not Null
image	FileField	Product image upload	Optional

Table 5.7

8. purchase_request_table

Field Name	Type	Description	Constraints
id	AutoField	Unique identifier	Primary Key
DEALERS	ForeignKey	Dealer making the purchase request	Not Null, FK(dealers_table.id)
date	DateField	Request date	Not Null

Table 5.8

9. purchase_request_more_table

Field Name	Type	Description	Constraints
id	AutoField	Unique identifier	Primary Key
PURCHASE_REQUEST	ForeignKey	Related purchase request	Not Null, FK(purchase_request_table.id)
PRODUCT	ForeignKey	Product requested	Not Null, FK(product_table.id)
quantity	IntegerField	Quantity requested	Not Null

Table 5.9

10. feedback_table

Field Name	**Type**	**Description**	**Constraints**
id	AutoField	Unique identifier	Primary Key
DEALERS	ForeignKey	Dealer giving feedback	Not Null, FK(dealers_table.id)
PRODUCT	ForeignKey	Product being reviewed	Not Null, FK(product_table.id)
feedback	TextField	Textual feedback	Not Null
date	DateField	Date of submission	Not Null

Table 5.10

11. complaint_table

Field Name	**Type**	**Description**	**Constraints**
id	AutoField	Unique identifier	Primary Key
DEALERS	ForeignKey	Dealer submitting complaint	Not Null, FK(dealers_table.id)
complaint	TextField	Complaint text	Not Null
date	DateField	Date submitted	Not Null

Table 5.11

5.3 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of work flows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e, work flows), as well as the data flows intersecting with the related activities.

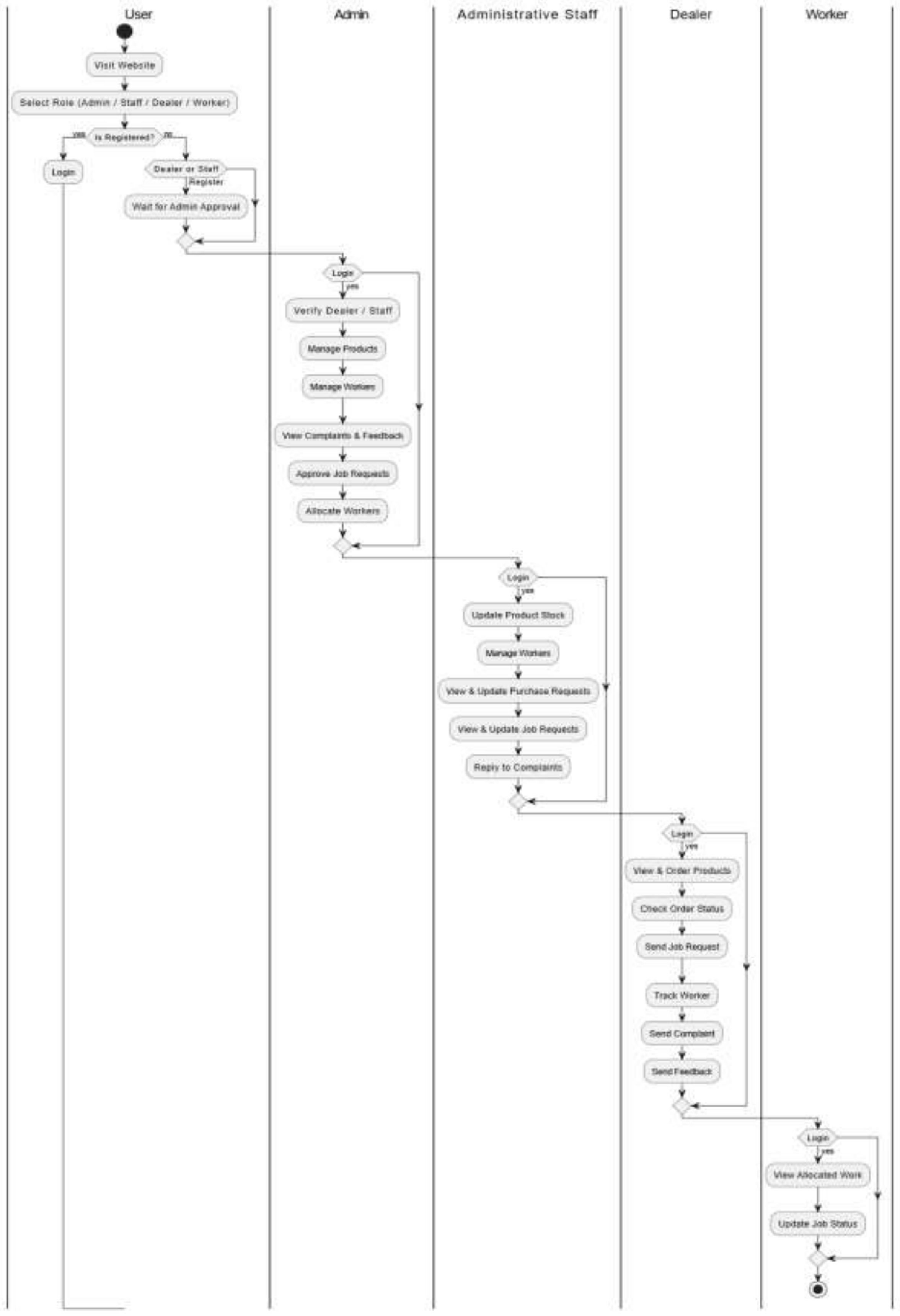


Fig 5.1 Activity Diagram

CHAPTER 6. CODING / DEVELOPMENT

The development phase is where system designs are transformed into a functional and usable application. The **Waterproofing Management System** was built using the **Python Django framework** for the backend, **MySQL** as the relational database system, and **HTML, CSS, Bootstrap, and JavaScript** for the frontend. This chapter outlines the development methodology followed, delivery approach, and code implementation samples that were part of the system's lifecycle.

Development was structured around **Agile methodology**, with tasks grouped into **iterative sprints**, each focusing on delivering working functionality that could be tested and refined incrementally.

6.1 INCREMENT DEFINITION

Each sprint culminated in the delivery of a functional module or enhancement referred to as an **increment**. To ensure consistent quality, a **Definition of Done (DoD)** was established for each increment.

Definition of Done (DoD):

- All associated user stories and acceptance criteria have been fulfilled.
- Code has been reviewed and merged into the main branch.
- Corresponding unit and integration tests pass without failure.
- Feature is integrated and deployed to the test/staging server.
- Relevant documentation and usage instructions are updated.

Delivered Configuration Items / Incremental Milestones

Sprint	Deliverables (Increments)
Sprint 1	Implemented login system and user role differentiation (Admin, Staff, Dealer, Client) with session management.
Sprint 2	Developed project module to allow admins to add job requests, allocate workers, and manage status.
Sprint 3	Finalized the MySQL database schema, added core models (Users, Projects, Workers, Dealers, Complaints).
Sprint 4	Built the feedback and complaint submission modules and connected them with user accounts.
Sprint 5	Developed inventory and purchase request modules used by dealers and admin to manage materials.
Sprint 6	Integrated work allocation features and role-based dashboards for Admins, Dealers, and Workers.
Sprint 7	Conducted end-to-end testing, optimized backend queries, fixed bugs, and deployed to the hosting environment.

Table 6.1

Each sprint increment contributed a working feature set that was validated before moving forward to the next.

6.2 CODE SAMPLES

Below are some key code snippets used in the **Waterproofing Management System**, showcasing coding conventions, modular architecture, and clean implementation practices:

url.py

```
from django.contrib import admin
from django.urls import path

from myapp import views

urlpatterns = [
    #admin
    path('',views.login),
    path('register',views.register),
    path('dealer_register',views.dealer_register),
    path('dealer_register_post',views.dealer_register_post),
    path('administrative_staff_register',views.administrative_staff_register),
    path('administrative_staff_register_post',views.administrative_staff_register_post),
    path('add_manage_worker',views.add_manage_worker),
    path('add_product',views.add_product),
    path('admin_home',views.admin_home),
    path('allocate_works/<id>',views.allocate_works),
    path('allocate_works_post',views.allocate_works_post),
    path('approve_job_request',views.approve_job_request),
    path('delete_work/<id>',views.delete_work),
    path('send_reply/<id>',views.send_reply),
    path('verify_administrativestaff',views.verify_administrativestaff),
    path('verify_dealer',views.verify_dealer),
    path('view_complaint',views.view_complaint),
    path('view_feedback',views.view_feedback),
    path('specify_feedback/<id>',views.specify_feedback),
    path('view_job_allocation',views.view_job_allocation),
    path('view_product',views.view_product),
    path('view_worker',views.view_worker),
    path('send_reply_post',views.send_reply_post),
    path('product_update_admin/<id>',views.product_update_admin),
    path('product_update_admin_post',views.product_update_admin_post),
    path('edit_worker/<id>',views.edit_worker),
    path('edit_worker_post',views.edit_worker_post),
    path('delete_worker/<id>',views.delete_worker),
    path('delete_product/<id>',views.delete_product),
    path('search_product',views.search_product),



    #administrtive staff
    path('add_manage_worker_administrative', views.add_manage_worker_administrative),
    path('administrative_staff_home', views.administrative_staff_home),
    path('product_update/<id>', views.product_update),
    path('send_reply_administrative/<id>', views.send_reply_administrative),
    path('view_complaint_administrative', views.view_complaint_administrative),
    path('view_feedback_administrative', views.view_feedback_administrative),
    path('view_job_allocation_administrative', views.view_job_allocation_administrative),
    path('view_job_request', views.view_job_request),
    path('view_product_administrative', views.view_product_administrative),
    path('view_job_request', views.view_job_request),
    path('view_product_administrative', views.view_product_administrative),
    path('view_purchase_request', views.view_purchase_request),
    path('view_purchase_request_more', views.view_purchase_request_more),
    path('view_worker_administrative', views.view_worker_administrative),
    path('accept_dealers/<id>', views.accept_dealers),
    path('reject_dealers/<id>', views.reject_dealers),
    path('accept_administrativestaff/<id>', views.accept_administrativestaff),
    path('reject_administrativestaff/<id>', views.reject_administrativestaff),
    path('add_product_post',views.add_product_post),
    path('add_manage_worker_post',views.add_manage_worker_post),
    path('add_product_administrative',views.add_product_administrative),
    path('add_product_administrative_post',views.add_product_administrative_post),
path('send_reply_administrative_post',views.send_reply_administrative_post),
    path('indexad',views.indexad),
    path('indexadm',views.indexadm),


    #DEALERS
    path('cart/<id>',views.cart),
    path('order',views.order),
    path('view_cart',views.view_cart),
    path('view_order',views.view_order),
    path('send_feedback/<id>',views.send_feedback),
    path('send_feedback_post',views.send_feedback_post),
    path('view_order_more',views.view_order_more),
    path('view_product_dealer',views.view_product_dealer),
    path('view_product_more_dealer/<id>',views.view_product_more_dealer),
    path('dealer_home',views.dealer_home),
    path('orderscode',views.orderscode),
    path('add_to_cart',views.add_to_cart),
    path('send_request',views.send_request),
    path('check_order',views.check_order),
    path('check_order_more/<id>',views.check_order_more),
    path('send_request_post',views.send_request_post),
    path('delete_cart/<cartid>',views.delete_cart),
    path('send_complaint',views.send_complaint),
    path('send_complaint_post',views.send_complaint_post),
    path('view_replay_dealer',views.view_replay_dealer),
    path('view_feedback_dealer',views.view_feedback_dealer),
    path('check_status',views.check_status),
    path('search_date',views.search_date),
    path('contact',views.contact),
    path('about',views.about),
    # path('user_pay_proceed/<id>/<amt>',views.user_pay_proceed),
    # path('on_payment_success',views.on_payment_success),
    path('buy_post',views.buy_post),



    #WORKER
    path('worker_home', views.worker_home),
    path('view_work', views.view_work),
    path('status_update/<id>', views.status_update),
    path('status_update_post', views.status_update_post),
    path('index', views.index),
    path('dealer_index', views.dealer_index),

    #actions
    path('login_post',views.login_post),
]
```

views.py

```
import json
from datetime import datetime

from django.core.files.storage import FileSystemStorage
from django.db.models import Q
from django.http.response import HttpResponse
from django.shortcuts import render

# Create your views here.
from myapp.models import *


def login(request):
   return render(request,'login.html')
def login_post(request):
   username=request.POST['textfield']
   pswd=request.POST['textfield2']

   a=login_table.objects.get(username=username,password=pswd)
   if a.type=='admin':
       return  HttpResponse('''<script>alert('ADMIN
LOGINED');window.location='/admin_home'</script>''')
   elif a.type=='administrative_staff':
       return HttpResponse('''<script>alert('ADMINISTRATIVE
LOGINED');window.location='/administrative_staff_home'</script>''')
   elif a.type=='dealer':
       request.session['lid']=a.id
       return HttpResponse('''<script>alert('DEALER
LOGINED');window.location='/dealer_home'</script>''')
   elif a.type=='worker':
       request.session['lid'] = a.id
       return HttpResponse('''<script>alert('WORKER
LOGINED');window.location='/worker_home'</script>''')
   else:
       return HttpResponse('''<script>alert('INVALID
LOGIN');window.location='/'</script>''')

def register(request):
   return render(request,'register.html')

#DEALER REGISTRATION:

def dealer_register(request):
   return render(request,'dealer_register.html')
```

```
def dealer_register_post(request):
    username = request.POST['textfield8']
    password = request.POST['textfield10']
    name = request.POST['textfield']
    contact = request.POST['textfield2']
    email = request.POST['textfield3']
    place = request.POST['textfield4']
    post = request.POST['textfield5']
    pin = request.POST['textfield6']
    id_proof_number = request.POST['textfield7']
    image = request.FILES['file']

    fs = FileSystemStorage()
    fp = fs.save(image.name, image)

    l=login_table()
    l.username = username
    l.password = password
    l.type = 'PENDING'
    l.save()




    d=dealers_table()
    d.name = name
    d.LOGIN = l
    d.contact = contact
    d.email = email
    d.place = place
    d.post = post
    d.pin = pin
    d.id_proof_number = id_proof_number
    d.image = fp
    d.save()
    return HttpResponse('''<script>alert('REGISTER REQUEST
SEND');window.location='/register'</script>''')

#ADMINISTRATIVE STAFF REGISTRATION

def administrative_staff_register(request):
    return render(request,'administrative_staff_register.html')


def administrative_staff_register_post(request):
    username = request.POST['textfield8']
    password = request.POST['textfield10']
    name = request.POST['textfield']
    contact = request.POST['textfield2']
    email = request.POST['textfield3']
    role = request.POST['textfield4']
    id_proof_number = request.POST['textfield7']
    image = request.FILES['file']

    fs = FileSystemStorage()
    fp = fs.save(image.name, image)

    l = login_table()
    l.username = username
    l.password = password
    l.type = 'PENDING'
    l.save()

    a=administrative_staff_table()

    a.name = name
    a.LOGIN = l
    a.contact = contact
    a.email = email
    a.role = role
    a.id_proof_number = id_proof_number
    a.image = fp
    a.save()

    return HttpResponse('''<script>alert('REGISTER REQUEST
SEND');window.location='/register'</script>''')



#ADD WORKER

def add_manage_worker(request):
    return render(request,'admin/add_manage_worker.html')

def add_manage_worker_post(request):
    name=request.POST['textfield']
    contact = request.POST['textfield3']
    email = request.POST['textfield2']
    skills = request.POST['textfield4']
    availability = request.POST['textfield5']
    image=request.FILES['file']
    id_proof_number = request.POST['textfield6']
    username=request.POST['textfield7']
    password=request.POST['textfield8']


    l=login_table()
    l.username=username
    l.password=password
    l.type='worker'
    l.save()



    fs=FileSystemStorage()
    fp=fs.save(image.name,image)

    c=workers_table()
    c.name = name
    c.contact = contact
    c.email = email
    c.skills = skills
    c.availability = availability
    c.image = fp
    c.id_proof_number = id_proof_number
    c.LOGIN = l
    c.save()

    return HttpResponse('''<script>alert('WORKER
ADDED');window.location='/view_worker'</script>''')
    return render(request,'admin/add_manage_worker.html')

#add product


def add_product(request):
    return render(request,'admin/add_product.html')


def add_product_post(request):
    name=request.POST['textfield']
    description=request.POST['textfield2']
    price = request.POST['textfield3']
    stock = request.POST['textfield4']
    image = request.FILES['file']

    fs=FileSystemStorage()
    fp=fs.save(image.name,image)

    c=product_table()
    c.name=name
    c.description=description
    c.price=price
    c.stock=stock
    c.image=fp
    c.save()
    return HttpResponse('''<script>alert('PRODUCT
ADDED');window.location='/view_product'</script>''')
    return render(request,'admin/add_product.html')

def admin_home(request):
    obusers=dealers_table.objects.filter(LOGIN__type="dealer")
    obworkers = workers_table.objects.all()
    oborder = purchase_request_more_table.objects.all()
    obfeed = feedback_table.objects.all()


    request.session["dealers"]=len(obusers)
    request.session["workers"]=len(obworkers)
    request.session["order"]=len(oborder)
    request.session["feed"]=len(obfeed)

    return
render(request,'admin/index.html',{"dealers":len(obusers),"workers":len(obworkers),"order":l
en(oborder),"feed":len(obfeed)})


#PRODUCT UPDATE

def product_update_admin(request,id):
    request.session['pid'] = id
    return  render(request,'admin/product_update_admin.html')

def product_update_admin_post(request):
    stock=request.POST['textfield']
    r = request.session['pid']

    p=product_table.objects.get(id=r)
    p.stock = stock
    p.save()
    return HttpResponse('''<script>alert('STOCK
UPDATED');window.location='/view_product'</script>''')



def approve_job_request(request):
    a=job_request_table.objects.all()
    return render(request,'admin/approve_job_request.html',{'data':a})

def allocate_works(request,id):
    request.session['jid'] = id # job request id
    a=workers_table.objects.all()
    return render(request, 'admin/allocate_works.html', {'data':a})

def allocate_works_post(request):
    from datetime import date
    name=request.POST['select']

    r = request.session['jid']
    start_date = request.POST['startdate']
    end_date = request.POST['enddate']


    a=allocate_work()
    a.JOB_REQUEST_id=r
    a.start_date = start_date
    a.allocation_date = date.today()
    a.end_date = end_date
    a.WORKER = workers_table.objects.get(id=name)
    a.save()
    return HttpResponse('''<script>alert('WORK
ALLOCATED');window.location='/approve_job_request'</script>''')

def delete_work(request,id):
    work = allocate_work.objects.get(id=id)
    work.delete()
    return HttpResponse('''<script>alert('WORK
DELETED');window.location='/approve_job_request'</script>''')
def send_reply(request,id):
    request.session['cid']=id
    return render(request,'admin/send_reply.html')

def send_reply_post(request):
    replay=request.POST['textfield']
    r=request.session['cid']

    c=complaint_table.objects.get(id=r)
    c.replay = replay
    c.save()
    return
HttpResponse('''<script>alert('REPLIED');window.location='/view_complaint'</script>''')

#admin administartive staff
def verify_administrativestaff(request):
    a=administrative_staff_table.objects.all()
    return render(request,'admin/verify_administrativestaff.html',{'data':a})

def accept_administrativestaff(request,id):
    ob=login_table.objects.get(id=id)
    ob.type='administrative_staff'
    ob.save()
    return
HttpResponse('''<script>alert('ACCEPTED');window.location='/verify_administrativestaff'</s
cript>''')
```

```
def reject_administrativestaff(request,id):
    ob=login_table.objects.get(id=id)
    ob.type='rejected'
    ob.save()
    return
HttpResponse('''<script>alert('REJECTED');window.location='/verify_administratiestaff'</scr
ipt>''')


#dealer verification
def verify_dealer(request):
    a=dealers_table.objects.all()
    return render(request,'admin/verify_dealer.html',{'data':a})

def accept_dealers(request,id):
    ob=login_table.objects.get(id=id)
    ob.type='dealer'
    ob.save()
    return
HttpResponse('''<script>alert('ACCEPTED');window.location='/verify_dealer'</script>''')

def reject_dealers(request,id):
    ob=login_table.objects.get(id=id)
    ob.type='rejected'
    ob.save()
    return
HttpResponse('''<script>alert('REJECTED');window.location='/verify_dealer'</script>''')




def view_complaint(request):
    a=complaint_table.objects.all()
    return render(request,'admin/view_complaint.html',{'data':a})

def view_feedback(request):
    a=feedback_table.objects.all()
    return render(request,'admin/view_feedback.html',{'data':a})

def specify_feedback(request,id):

    # request.session["productid"] = id
    a = product_table.objects.get(id=id)

    feed = feedback_table.objects.filter(PRODUCT=id)
    return render(request,'admin/specify_feedback.html',{'i':a,'feed':feed})


def view_job_allocation(request):
    a = allocate_work.objects.all()
    return render(request,'admin/view_job_allocation.html',{'data':a})


def view_product(request):
    a=product_table.objects.all()
    return render(request,'admin/view_product.html',{'data':a})

def search_product(request):
    search = request.POST['textfield']
    a=product_table.objects.filter(name__icontains=search)
    return render(request,'admin/view_product.html',{'data':a})




def view_worker(request):
    a=workers_table.objects.all()
    return render(request,'admin/view_worker.html',{'data':a})

def edit_worker(request,id):
    worker= workers_table.objects.get(id=id)
    request.session['wid'] = id
    return render(request,'admin/edit_worker.html',{'data':worker})

def edit_worker_post(request):
    r = request.session['wid']
    name = request.POST['textfield']
    contact = request.POST['textfield3']
    email = request.POST['textfield2']
    skills = request.POST['textfield4']
    availability = request.POST['textfield5']
    id_proof_number = request.POST['textfield6']

    w = workers_table.objects.get(id=r)


    if 'file' in request.FILES:
        image=request.FILES['file']
        fs = FileSystemStorage()
        fp = fs.save(image.name, image)
        w.image = fp
        w.save()

    w.name = name
    w.contact = contact
    w.email = email
    w.skills = skills
    w.availability = availability
    w.id_proof_number = id_proof_number
    w.save()
    return HttpResponse('''<script>alert('WORKER
UPDATED');window.location='/view_worker'</script>''')

def delete_worker(request,id):
    worker = workers_table.objects.get(id=id)
    worker.delete()
    return HttpResponse('''<script>alert('WORKER
DELETED');window.location='/view_worker'</script>''')

def delete_product(request,id):
    product = product_table.objects.get(id=id)
    product.delete()
    return HttpResponse('''<script>alert('PRODUCT
DELETED');window.location='/view_worker'</script>''')




#administrative staff

def add_manage_worker_administrative(request):
    return render(request,'administrative_staff/add_manage_worker_administrative.html')


#ADD PRODUCT ADMINISTRATIVE

def add_product_administrative(request):
    return render(request,'administrative_staff/add_product_administrative.html')

def add_product_administrative_post(request):
    name = request.POST['textfield']
    description =  request.POST['textfield2']
    price = request.POST['textfield3']
    stock = request.POST['textfield4']
    image = request.FILES['file']

    fs = FileSystemStorage()
    fp = fs.save(image.name,image)

    c = product_table
    c.name = name
    c.description = description
    c.price = price
    c.stock = stock
    c.image = fp
    c.save()
    return  HttpResponse('''<script>aleret('PRODUCT
ADDED');window.location='/view_product_admnistrative'</script>''')
    return render(request,'administrative_staff/add_product_administrative.html')

def administrative_staff_home(request):
    return render(request,'administrative_staff/indexad.html')

#PRODUCT UPDATE

def product_update(request,id):
    request.session['pid'] = id
    return  render(request,'admin/product_update_admin.html')

def product_update_admin_post(request):
    stock=request.POST['textfield']
    r = request.session['pid']

    p=product_table.objects.get(id=r)
    p.stock = stock
    p.save()
    return HttpResponse('''<script>alert('STOCK
UPDATED');window.location='/view_product'</script>''')

    # return  render(request,'admin/product_update_admin.html')

def administrative_staff(request):
    return render(request,'administrative_staff/adminstrative_staff_home.html')
#ADDMINISTARTIVE REPLY COMPLAINT

def send_reply_administrative(request,id):
    request.session['cid'] = id
    return render(request,'administrative_staff/send_reply_administrative.html')


def send_reply_administrative_post(request):
    r = request.session['cid']
    replay = request.POST['textfield']

    s = complaint_table.objects.get(id=r)
    s.replay = replay
    s.save()


    return
HttpResponse('''<script>alert('REPLAIED');window.location='/view_complaint_administrati
ve'</script>''')
```

```
def view_complaint_administrative(request):
    a=complaint_table.objects.all()
    return
render(request,'administrative_staff/view_complaint_administrative.html',{'data':a})

def view_feedback_administrative(request):
    a = feedback_table.objects.all()
    return render(request,'administrative_staff/view_feedback_administrative.html',{'data':a})

def view_job_allocation_administrative(request):
    a = allocate_work.objects.all()
    return
render(request,'administrative_staff/view_job_allocation_administrative.html',{'data':a})

def view_job_request(request):
    a = job_request_table.objects.all()
    return render(request,'administrative_staff/view_job_request.html',{'data':a})

def view_product_administrative(request):
    a=product_table.objects.all()
    return render(request,'administrative_staff/view_product_administrative.html',{'data':a})

def view_purchase_request(request):
    a=purchase_request_table.objects.all()
    return render(request,'administrative_staff/view_purchase_request.html',{'data':a})

def view_purchase_request_more(request):
    a= purchase_request_more_table.objects.all()
    return render(request,'administrative_staff/view_purchase_request_more.html',{'data':a})

def view_worker_administrative(request):
    a=workers_table.objects.all()
    return render(request,'administrative_staff/view_worker_administrative.html',{'data':a})



#DEALERS

def cart(request,id):
    request.session['pid']=id
    return render(request,'dealers/cart.html')

def dealer_home(request):
    products = product_table.objects.all()

    return render(request,'dealers/index.html',{'products':products})


def order(request):
    return render(request,'dealers/order.html')

def contact(request):
    return render(request,'dealers/contact.html')

def view_cart(request):

a=purchase_request_more_table.objects.filter(PURCHASE__status="CART",PURCHASE__
DEALEARS__LOGIN=request.session["lid"])
    amt=0
    for i in a:
        amt=amt+i.amount
    return render(request,'dealers/view_cart.html',{'data':a,"total":amt})

def view_order(request):
    return render(request,'dealers/view_order.html')

def view_order_more(request):
    return render(request,'dealers/view_order_more.html')

def view_product_dealer(request):

    a = product_table.objects.all()
    return render(request, 'dealers/view_product_dealer.html', {'data':a})

def send_request(request):
    return render(request,'dealers/send_request.html')

def send_request_post(request):
    work=request.POST['textfield']
    description=request.POST['textfield2']
    import datetime

    a=job_request_table()
    a.work=work
    a.DEALEARS=dealers_table.objects.get(LOGIN=request.session["lid"])
    a.description = description
    a.date = datetime.datetime.today().now()
    a.save()
    return HttpResponse('''<script>alert('JOB REQUEST SEND
SUCCESSFULLY');window.location='/send_request'</script>''')


def view_product_more_dealer(request,id):
    request.session['proid']=id
    a = product_table.objects.get(id=id)

    feed=feedback_table.objects.filter(PRODUCT=id)
    return render(request, 'dealers/view_product_more_dealer.html', {'i':a,'feed':feed})
```

```
def orderscode(request):

    pro_id = request.session['proid']
    qty = request.POST['textfield']
    lid = request.session['lid']

    ob = product_table.objects.get(id=pro_id)
    tt = int(ob.price) * int(qty)
    stock = ob.stock
    nstk = int(stock) - int(qty)
    if int(stock)>= int(qty):
        up = product_table.objects.get(id=pro_id)
        up.stock = nstk
        up.save()
        ob = purchase_request_table()
        ob.status = 'ORDER'
        from datetime import datetime

        ob.date = datetime.today().date()
        ob.DEALEARS = dealers_table.objects.get(LOGIN__id=lid)
        ob.amount = tt
        ob.save()
        obj = purchase_request_more_table()
        obj.PURCHASE = ob
        obj.PRODUCT=product_table.objects.get(id=pro_id)
        obj.quantity = qty
        obj.status = 'ORDER'
        obj.date=datetime.now()
        obj.amount = tt
        obj.save()

        obx = payment()
        obx.date = datetime.today()
        obx.time = datetime.now()
        obx.PURCHASE_REQUEST_TABLE = purchase_request_table.objects.get(id=ob.id)
        obx.status = 'ordered'
        obx.save()

        # return
HttpResponse("<script>window.location='/user_pay_proceed/"+str(ob.id)+"/"+str(tt)+"'</scri
pt>")
        return
HttpResponse("<script>alert('Ordered');window.location='/view_product_dealer'</script>")
    else:
        return HttpResponse('''<script>alert('OUT OF
STOCK');window.location='/view_product_dealer'</script>''')
```

```
def add_to_cart(request):
    qty = request.POST['textfield']
    pid = request.session['pid']
    lid = request.session['lid']

    qq = product_table.objects.get(id=pid)
    tt = int(qq.price) * int(qty)
    stock = int(qq.stock)
    nstk = int(stock) - int(qty)
    from datetime import datetime


    if stock >= int(qty):
        up = product_table.objects.get(id=pid)
        up.stock = nstk
        up.save()
        q
=purchase_request_table.objects.filter(DEALEARS=dealers_table.objects.get(LOGIN__id=li
d), status='CART')
        if len(q) == 0:
            qt = purchase_request_table()


            qt.date = datetime.today().date()
            qt.DEALEARS = dealers_table.objects.get(LOGIN=lid)
            qt.status = 'CART'
            qt.amount = tt
            qt.save()

            qty1 = purchase_request_more_table()
            qty1.quantity = qty
            qty1.PRODUCT = product_table.objects.get(id=pid)
            qty1.PURCHASE = qt
            qty1.amount=tt

            qty1.date = datetime.today()
            qty1.save()

            return HttpResponse('''<script>alert('ADDED
SUCCESSFULLY');window.location='/view_product_dealer'</script>''')

        else:
            total = int(q[0].amount) + int(tt)
            qt = purchase_request_table.objects.get(id=q[0].id)
            qt.amount = total
            qt.save()
            qty1 = purchase_request_more_table.objects.filter(PRODUCT__id=pid,
PURCHASE__id=q[0].id)
            if len(qty1) == 0:
                qqt = purchase_request_more_table()
                qqt.PURCHASE = q[0]
                qqt.PRODUCT = product_table.objects.get(id=pid)
                qqt.quantity = qty
                qqt.amount=tt
                qqt.date = datetime.now().date()
                qqt.save()
            else:
                qry1 = purchase_request_more_table.objects.get(id=qty1[0].id)
                quty = int(qty1[0].quantity) + int(qty)
                amt=int(qty1[0].amount)+int(tt)
                qry1.quantity = quty
                qry1.amount=amt
                qry1.save()
                return HttpResponse('''<script>alert('ADDED
SUCCESSFULLY');window.location='/view_product_dealer'</script>''')

    else:
        return HttpResponse('''<script>alert('OUT OF
STOCK');window.location='/view_product_dealer'</script>''')

    return HttpResponse('''<script>alert('ADDED
SUCCESSFULLY');window.location='/view_product_dealer'</script>''')

def delete_cart(request,cartid):
    ob=purchase_request_more_table.objects.get(id=cartid)
    amt=ob.amount
    qty=ob.quantity
    pid=ob.PRODUCT.id

    obx=purchase_request_table.objects.get(id=ob.PURCHASE.id)
    tot=int(obx.amount)-int(amt)
    obx.amount=tot
    obx.save()

    obx1=product_table.objects.get(id=pid)
    totqty=int(obx1.stock)+int(qty)
    obx1.stock=totqty
    obx1.save()

    ob.delete()
    return HttpResponse('''<script>alert('REMOVED FROM
CART');window.location='/view_cart'</script>''')

def check_order(request):
    lid = request.session['lid']
    a =
purchase_request_table.objects.filter(~Q(status="CART"),DEALEARS=dealers_table.objects
.get(LOGIN__id=lid))
    return render(request,'dealers/check_order.html',{'data':a})
```

```
def check_order_more(request,id):
    a=purchase_request_more_table.objects.filter(PURCHASE=id)
    return render(request, 'dealers/check_order_more.html', {'data': a})

def send_complaint(request):
    return render(request,'dealers/send_complaint.html')

def send_complaint_post(request):
    description = request.POST['textfield']

    a = complaint_table()
    a.DEALEARS = dealers_table.objects.get(LOGIN=request.session["lid"])
    a.description = description
    a.replay = 'PENDING'
    a. date = datetime.datetime.today().now()
    a.save()
    return HttpResponse('''<script>alert('COMPLAINT SEND
SUCCESSFULLY');window.location='/view_replay_dealer'</script>''')

def view_replay_dealer(request):
    lid = request.session['lid']
    a =
complaint_table.objects.filter(DEALEARS=dealers_table.objects.get(LOGIN__id=lid))
    return render(request,'dealers/view_replay_dealer.html',{'data':a})


def send_feedback(request,id):

    request.session["productid"]=id

    return render(request,'dealers/send_feedback.html')




def send_feedback_post(request):

    print(request.POST)
    comments = request.POST['textfield']
    rating = request.POST['rating']

    a = feedback_table()
    a.DEALEARS = dealers_table.objects.get(LOGIN=request.session["lid"])
    a.PRODUCT_id=request.session["productid"]
    a.comments =comments
    a.rating = rating
    a.date = datetime.now().date()
    a.save()
    return HttpResponse('''<script>alert('FEEDBACK SEND
SUCCESSFULLY');window.location='/view_product_dealer'</script>'")

def search_date(request):
   if request.method == "POST":
      fromdate = request.POST.get('fromdate')
      todate = request.POST.get('todate')



      if fromdate and todate:
         records = purchase_request_more_table.objects.filter(date__range=[fromdate, todate])
      else:
         records = purchase_request_more_table.objects.all()

      return render(request, 'dealers/check_order.html', {'data': records})

   return render(request, 'dealers/check_order.html', {'data': []})


def view_feedback_dealer(request):
   lid = request.session['lid']
   a = feedback_table.objects.filter(DEALEARS=dealers_table.objects.get(LOGIN__id=lid))
   return  render(request,'dealers/view_feedback_dealer.html',{'data':a})


def check_status(request):
   b=
allocate_work.objects.filter(JOB_REQUEST__DEALEARS__LOGIN_id=request.session['li
d'])
   return render(request,'dealers/check_status.html',{'data':b})


#WORKER

def worker_home(request):
   return render(request,'worker/worker_home.html')

def view_work(request):
   a = allocate_work.objects.filter(WORKER__LOGIN__id=request.session['lid'])
   return render(request,'worker/view_work.html',{'data': a })

def status_update(request,id):
   request.session['aid'] =id
   return render(request,'worker/status_update.html')


def status_update_post(request):
   status = request.POST['select']
   i = request.session['aid']
```

```
    a=allocate_work.objects.get(id=i)
    a.status = status
    a.save()
    return HttpResponse('''<script>alert('STATUS
UPDATED');window.location='/view_work'</script>''')


def index(request):

    return render(request,'admin/index.html')


def dealer_index(request):
    return render(request,'dealers/index.html')

def about(request):
    return render(request,'dealers/about.html')


def indexad(request):
    return render(request,'administrative_staff/indexad.html')

def indexadm(request):
    return render(request,'administrative_staff/indexadm.html')
```

models.py

```
from django.db import models

# Create your models here.
class login_table(models.Model):
    username=models.CharField(max_length=100)
    password=models.CharField(max_length=100)
    type=models.CharField(max_length=100)

class dealers_table(models.Model):
    LOGIN=models.ForeignKey(login_table,on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    contact = models.BigIntegerField()
    email = models.CharField(max_length=100)
    place = models.CharField(max_length=100)
    post = models.CharField(max_length=100)
    pin = models.IntegerField()
    id_proof_number = models.BigIntegerField()
    image = models.FileField()
```

```
class administrative_staff_table(models.Model):
    LOGIN = models.ForeignKey(login_table, on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    contact = models.BigIntegerField()
    email = models.CharField(max_length=100)
    role = models.CharField(max_length=100)
    id_proof_number = models.BigIntegerField()
    image = models.FileField()

class product_table(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField()
    price = models.IntegerField()
    stock = models.IntegerField()
    image = models.FileField()

class workers_table(models.Model):
    LOGIN = models.ForeignKey(login_table, on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    contact = models.BigIntegerField()
    email = models.CharField(max_length=100)
    skills = models.TextField()
    availability = models.CharField(max_length=100)
    image = models.FileField()
    id_proof_number = models.BigIntegerField()

class complaint_table(models.Model):
    DEALEARS = models.ForeignKey(dealers_table,on_delete=models.CASCADE)

  description = models.TextField()
    replay = models.CharField(max_length=100)
    date = models.DateField()

class feedback_table(models.Model):
    DEALEARS = models.ForeignKey(dealers_table, on_delete=models.CASCADE)
    PRODUCT = models.ForeignKey(product_table, on_delete=models.CASCADE)
    comments = models.TextField()
    rating = models.IntegerField()

    date = models.DateField()

class job_request_table(models.Model):
    DEALEARS = models.ForeignKey(dealers_table, on_delete=models.CASCADE)
    work = models.CharField(max_length=100)
    description = models.TextField()
    date = models.DateField()
    # amount = models.IntegerField()

class purchase_request_table(models.Model):
    DEALEARS = models.ForeignKey(dealers_table,
on_delete=models.CASCADE)
    status = models.CharField(max_length=100)
    date = models.DateField()
    amount = models.IntegerField()


class purchase_request_more_table(models.Model):
    PURCHASE =
models.ForeignKey(purchase_request_table,on_delete=models.CASCADE)
    PRODUCT = models.ForeignKey(product_table,on_delete=models.CASCADE)
    quantity = models.IntegerField()
    status = models.CharField(max_length=100)
    date = models.DateField()
    amount = models.IntegerField()

class allocate_work(models.Model):
    JOB_REQUEST = models.ForeignKey(job_request_table,on_delete=models.CASCADE)
    WORKER = models.ForeignKey(workers_table,on_delete=models.CASCADE)
    start_date = models.DateField()
    end_date = models.DateField()
    allocation_date = models.DateField()
    status = models.CharField(max_length=100)

class payment(models.Model):

    PURCHASE_REQUEST_TABLE =
models.ForeignKey(purchase_request_table,on_delete=models.CASCADE)
    date = models.DateField()
    time = models.TimeField()
    status = models.CharField(max_length=100)
```

CHAPTER 7. TESTING

Testing is an essential phase of the Software Development Life Cycle (SDLC), ensuring that each feature operates correctly and that the system as a whole satisfies its functional and non-functional requirements. In the **Waterproofing Management System**, comprehensive testing was carried out during and after each development sprint. This ensured system stability, data consistency, user satisfaction, and platform security.
Testing activities followed **Agile-based validation**, with a strong emphasis on incremental feedback and improvement using unit, integration, and user acceptance testing.

7.1 TESTING DOCUMENTATION

The testing process was structured based on the acceptance criteria established during sprint planning. Various testing methods were employed to cover functionality, performance, and user experience:

7. **Unit Testing**

Individual components such as Django views, models, and utility functions were tested in isolation using Django’s built-in TestCase framework.

b. Integration Testing

Connected modules like job request submission and worker allocation were tested together to ensure correct flow between components.

c. System Testing

The full application was tested as an integrated product. Each module (login, dashboard, feedback, complaints, allocation, inventory) was verified to ensure cross-module functionality.

d. User Acceptance Testing (UAT)

Peer and mentor feedback was collected to evaluate the platform’s usability, clarity of features, and visual design from an end-user’s perspective.

e. Manual Testing

The UI was manually tested on various browsers (Chrome, Edge, Firefox) and screen resolutions to verify responsiveness, layout consistency, and interactive behavior.

f. API Testing

Postman was used to test Django REST API endpoints related to login, data fetch, and submission. Response validation, error messages, and security headers were checked.

7.2 TEST PLANS DERIVED

Below is a snapshot of selected test cases corresponding to the development sprints:

Sprint	**Test Case**	**Expected Outcome**	**Status**
Sprint 1	Register admin/staff with valid data	User is created and stored in database	Passed
Sprint 1	Attempt login with incorrect credentials	Error message is shown	Passed
Sprint 2	View available job requests as a Admin	List of jobs displayed	Passed
Sprint 3	Assign workers to job requests	Allocation record is saved	Passed
Sprint 4	Submit feedback as client	Feedback is stored and viewable to admin	Passed
Sprint 5	Submit and view purchase requests by dealers	Request recorded and shown to admin	Passed
Sprint 6	View allocated work in worker dashboard	Assigned tasks visible to specific worker	Passed
Sprint 7	Complete system test and deployment	All modules functional and stable in test server	Passed

Table 7.1

All bugs and issues found during testing were logged and assigned for resolution within the sprint cycle.

7.3 RESULTS FROM TESTING

Final testing confirmed that the **Waterproofing Management System** was reliable, responsive, and user-ready:

- All key modules received full test coverage.
- UI tested across major browsers and mobile devices for responsiveness.
- Query execution and page load times were within acceptable limits.
- All identified bugs were resolved prior to deployment.
- UAT feedback in the last sprint confirmed project readiness for live deployment.

Tools Used

- **Postman** – For backend API request/response testing
- **Google Chrome DevTools** – For frontend layout and console error tracking
- **Excel Sheets** – Manual test case planning and issue tracking

CHAPTER 8. DEPLOYMENT STRATEGY

Deployment represents the final phase of the project development lifecycle. It transitions the system from the development and testing environment to a stable, accessible setup that simulates real-world usage. For the **Waterproofing Management System**, the deployment strategy was designed to ensure stability, scalability, and easy recovery while also adhering to academic requirements.

8.1 VERSION CONTROL AND GIT

To ensure consistent code management, team collaboration, and backup reliability, **Git** was used as the version control system, with **GitHub** serving as the remote repository platform for storing, syncing, and reviewing the codebase.

8.1.1 IMPORTANCE OF VERSION CONTROL AND GIT

1. **Team Collaboration**

Git facilitated efficient teamwork by allowing multiple contributors to work on isolated branches without affecting the main project structure. Feature branches were used to develop new modules (e.g., job allocation, attendance) and later merged through pull requests.

2. **Change Tracking**

Git maintained a complete history of every code change, providing visibility into what was changed, when, and by whom. This made it easier to trace bugs, analyze development progress, and conduct retrospectives after each sprint.

3. **Rollback Capabilities**

In case a bug or error was introduced in recent commits, Git's revert functionality allowed the team to quickly restore previous versions, minimizing downtime and rework.

4. **Improved Code Quality through Peer Reviews**

The team used Git pull requests for peer reviews before merging changes. This encouraged code accountability and mutual learning while maintaining high-quality code standards.

5. **Reliable Backup and Recovery**

With all project code securely stored in a GitHub repository, accidental loss of local files did not affect project continuity. Team members could easily restore their environments by cloning the central repo.

6. **Task Tracking Integration**

Git branches were linked with sprint tasks and backlog items. This integration allowed better traceability of user stories and ensured development aligned with the Scrum plan.

8.2 RELEASE NOTES

The following summarizes the final stable release submitted as part of the academic deliverables:

Version 1.0 – Final Academic Release

Major Features Implemented:

- Secure login/logout with role-based access (admin, contractor, client, worker, dealer)
- Client and contractor dashboards
- Job request submission and contractor assignment
- Work allocation and status tracking
- Feedback and complaint modules
- Dealer portal for submitting purchase requests
- Admin panel for monitoring all modules
- Inventory and material tracking
- Bootstrap-integrated responsive frontend

Known Issues:

- Occasional lag when loading job allocation data under poor internet conditions
- Feedback module could benefit from text analytics in future versions
- Export-to-PDF functionality for reports is planned but not included

Repository Details:

- **Platform**: GitHub
- **Repository Name**: waterproofing-management-system
- **Visibility**: Private (restricted to academic use)
- **Branching Structure**: main, development, feature/*, bugfix/*

CHAPTER 9. SUMMARY

The **Waterproofing Management System** project was developed to provide a centralized platform that streamlines key operations such as job requests, contractor assignments, attendance tracking, dealer inventory handling, and user communication. Throughout the lifecycle of the project, a structured and Agile-based approach was followed—covering requirement analysis, design, development, testing, and deployment.
This chapter presents a summary of project experiences, key takeaways, and reflective insights gathered during the course of the development journey.

9.1 RETROSPECTIVE NOTES

To align with Scrum methodology, sprint retrospectives were held after the completion of each iteration. These reflective sessions provided an opportunity for the team to evaluate the sprint outcomes, identify roadblocks, and propose improvements.

Lessons Learned:

- **Team Collaboration Drives Progress:** The sprints that started with clear roles and a shared understanding of objectives had smoother execution and higher productivity.
- **Regular Syncing is Essential:** A few delays were observed when updates between the frontend (UI) and backend teams were not aligned in real-time.
- **Early Code Reviews Matter:** Reviewing each other's code helped catch logic errors early and ensured better system stability.
- **Timely Documentation Helps:** Lack of timely updates in system documentation led to minor confusion during testing and final integration phases.

Actionable Improvements:

- Introduce **daily stand-ups** or short sync meetings to keep everyone on the same page.
- Leverage tools like **Trello or Jira** more rigorously to track sprint progress visually and collaboratively.
- Integrate **automated testing and deployment tools** in future versions to improve CI/CD processes.
- Maintain a centralized and updated **technical documentation hub** to support integration and future handovers.

These reflections and continuous improvements not only helped enhance the quality of this project but also equipped the development team with practical insights for future academic and professional software projects.

APPENDICES

1. SREENSHOT OF ACTUAL UIDS

Signup:

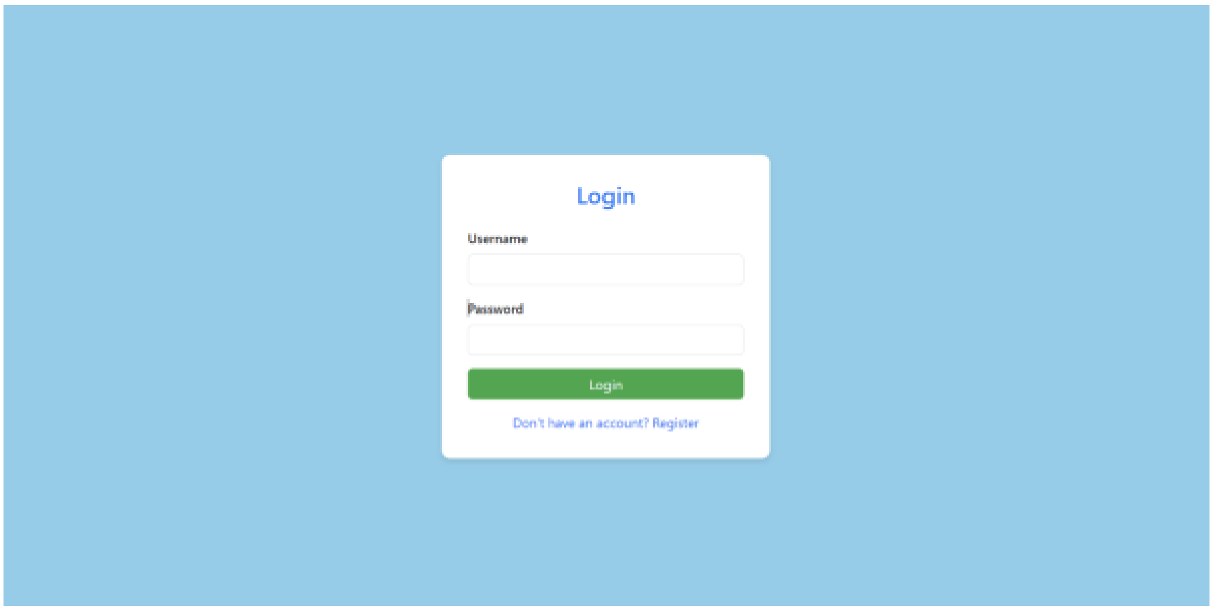


Fig.10.1(Signup)

Admin Panel:

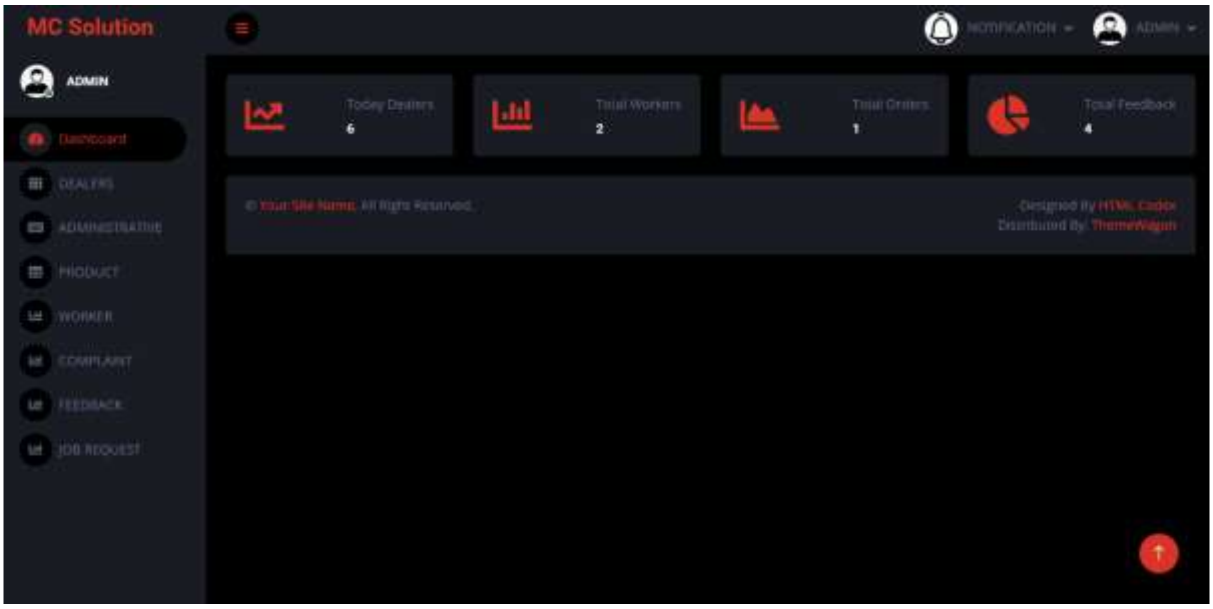


Fig.10.2(Admin Panel)

Dealer Dashboard:

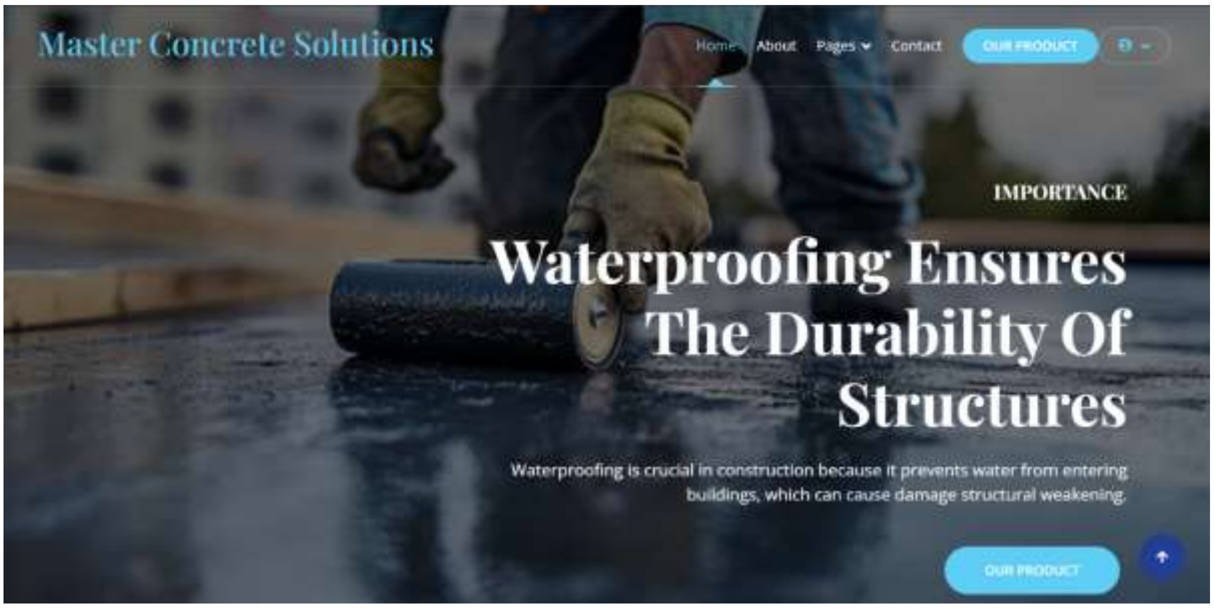
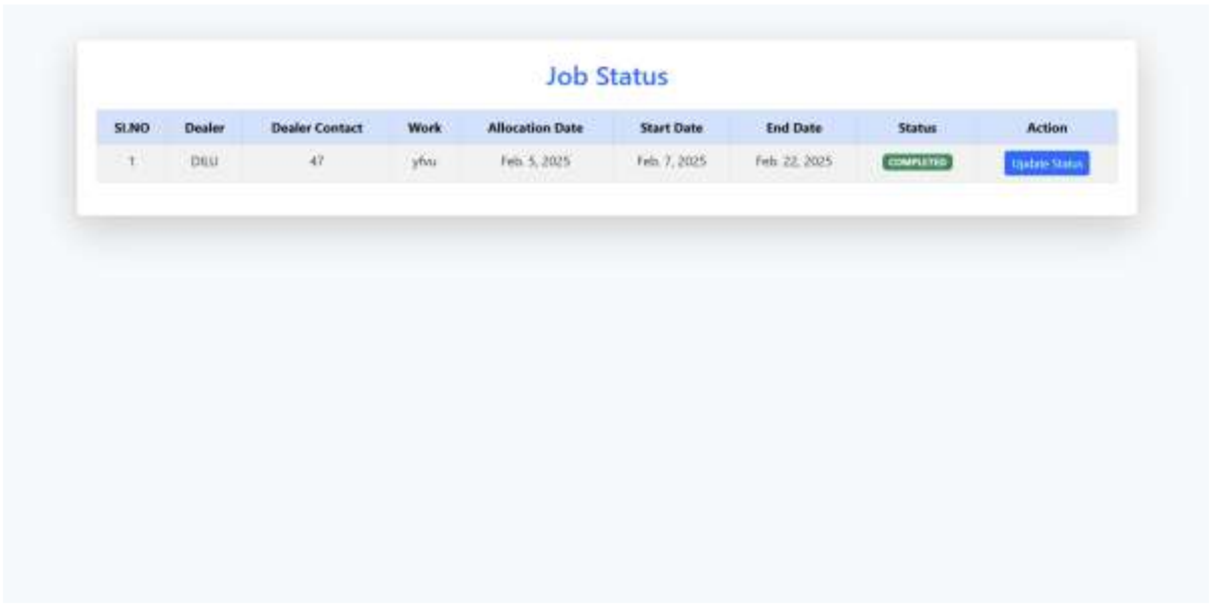

Master Concrete Solutions
Home
About
Pages
Contact
OUR PRODUCT
IMPORTANCE
Waterproofing Ensures The Durability Of Structures
Waterproofing is crucial in construction because it prevents water from entering buildings, which can cause damage structural weakening.
OUR PRODUCT

Fig.10.3(Dealer Dashboard)

Worker View Allocation:



Job Status

Sl.NO	Dealer	Dealer Contact	Work	Allocation Date	Start Date	End Date	Status	Action
1	DILU	47	yfvu	Feb. 5, 2025	Feb. 7, 2025	Feb. 22, 2025	COMPLETED	Update Status

Fig.10.4(Worker Allocated Work)

Dealer Product View:

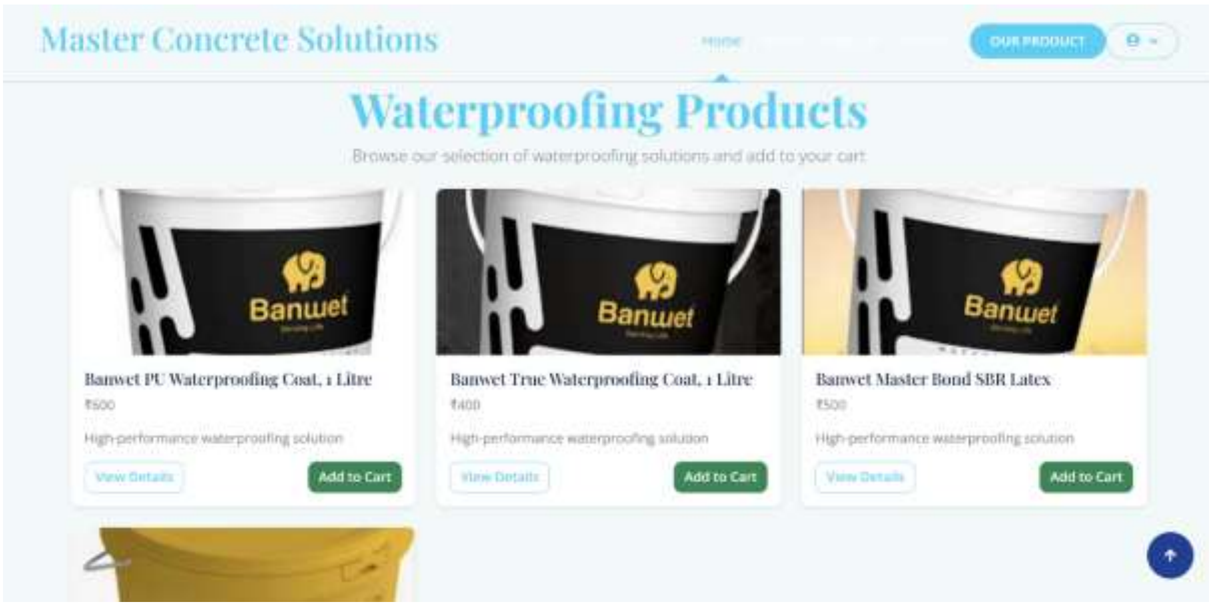
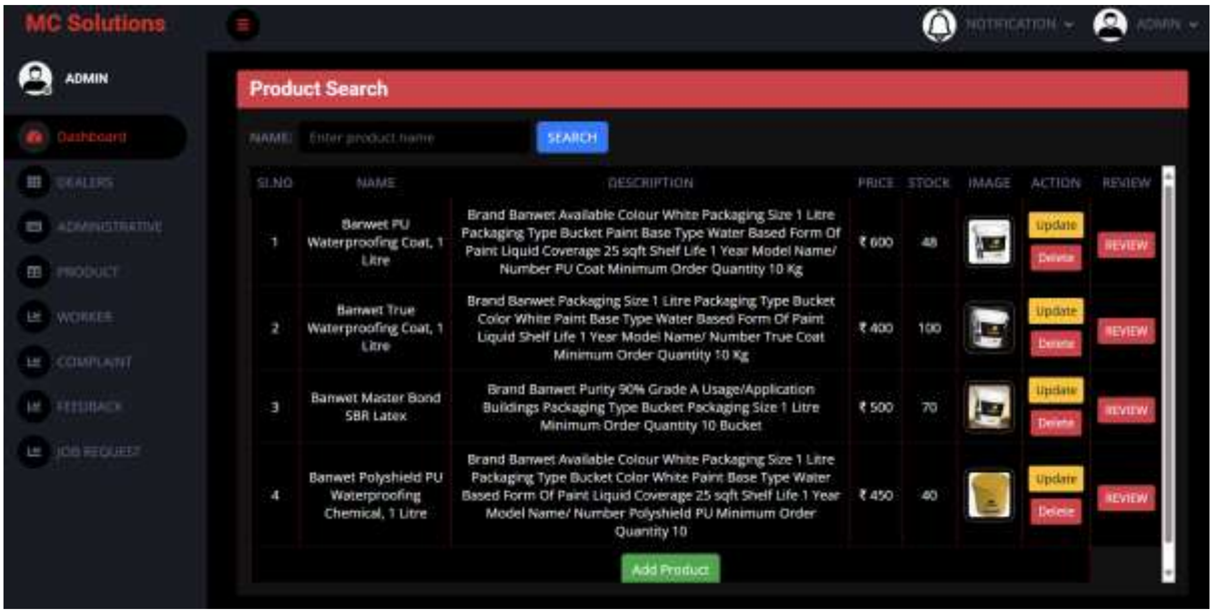

Master Concrete Solutions
OUR PRODUCT
Waterproofing Products
Browse our selection of waterproofing solutions and add to your cart
Banwet PU Waterproofing Coat, 1 Litre
₹500
High-performance waterproofing solution
View Details
Add to Cart
Banwet True Waterproofing Coat, 1 Litre
₹400
High-performance waterproofing solution
View Details
Add to Cart
Banwet Master Bond SBR Latex
₹500
High-performance waterproofing solution
View Details
Add to Cart

Fig.10.5(Dealer View Product)

Admin Product View:



SL.NO	NAME	DESCRIPTION	PRICE	STOCK	IMAGE	ACTION	REVIEW
1	Banwet PU Waterproofing Coat, 1 Litre	Brand Banwet Available Colour White Packaging Size 1 Litre Packaging Type Bucket Paint Base Type Water Based Form Of Paint Liquid Coverage 25 sqft Shelf Life 1 Year Model Name/ Number PU Coat Minimum Order Quantity 10 Kg	₹ 600	48		Update Delete	REVIEW
2	Banwet True Waterproofing Coat, 1 Litre	Brand Banwet Packaging Size 1 Litre Packaging Type Bucket Color White Paint Base Type Water Based Form Of Paint Liquid Shelf Life 1 Year Model Name/ Number True Coat Minimum Order Quantity 10 Kg	₹ 400	100		Update Delete	REVIEW
3	Banwet Master Bond SBR Latex	Brand Banwet Purity 90% Grade A Usage/Application Buildings Packaging Type Bucket Packaging Size 1 Litre Minimum Order Quantity 10 Bucket	₹ 500	70		Update Delete	REVIEW
4	Banwet Polyshield PU Waterproofing Chemical, 1 Litre	Brand Banwet Available Colour White Packaging Size 1 Litre Packaging Type Bucket Color White Paint Base Type Water Based Form Of Paint Liquid Coverage 25 sqft Shelf Life 1 Year Model Name/ Number Polyshield PU Minimum Order Quantity 10	₹ 450	40		Update Delete	REVIEW

Fig.10.6(Admin View Product)

I. GLOSSARY OF TERMS

Term	**Definition**
Scrum	An Agile framework used for managing iterative software development.
Sprint	A fixed duration within which a defined set of tasks or features are completed.
Product Backlog	A prioritized list of project features, requirements, and fixes.
Burn Down Chart	A visual graph showing remaining work against time for sprint tracking.
Repository	A version-controlled storage location (e.g., GitHub) for maintaining source code.
Pull Request	A method of submitting code changes from one branch to another after review.
ER Diagram	A graphical representation of entities and their relationships in a database.
Django	A Python-based web framework used for rapid backend development.
MySQL	A relational database management system used for storing structured data.
Bootstrap	A frontend toolkit for designing responsive web interfaces using CSS/JS.
Postman	A tool for testing and validating RESTful APIs.
HTML/CSS/JS	Standard web technologies used for structuring, styling, and scripting pages.

II. ADDITIONAL RESOURCES

The following resources and tools contributed to the development, testing, and deployment of the **Waterproofing Management System**:

- **Django Documentation** – For backend development and model-based design.
- **MySQL Documentation** – For implementing and managing the project database.
- **Scrum Guide** – For iterative development and sprint planning under Agile methodology.
- **Bootstrap** – For designing user-friendly, responsive interfaces.
- **Postman** – For API response validation and error handling.
- **Git & GitHub** – For version control, collaboration, and backup.
- **Lucidchart / draw.io** – For visualizing ER diagrams and data flows.
- **VS Code** – As the primary integrated development environment (IDE).
- **Excel** – For creating manual test checklists and documenting results.

REFERENCES

1. Django Documentation, Django Software Foundation. [Online]. Available: https://docs.djangoproject.com
2. MySQL Documentation, Oracle Corporation. [Online]. Available: https://dev.mysql.com/doc/
3. Scrum Guide, Ken Schwaber & Jeff Sutherland, Scrum.org, 2020. [Online]. Available: https://scrumguides.org
4. Bootstrap Documentation, Twitter, Inc. [Online]. Available: https://getbootstrap.com
5. Git & GitHub Docs, GitHub Inc. [Online]. Available: https://docs.github.com
6. Postman API Testing Tool, Postman, Inc. [Online]. Available: https://www.postman.com
7. draw.io (diagrams.net) [Online]. Available: https://app.diagrams.net
8. Lucidchart, Lucid Software Inc. [Online]. Available: https://www.lucidchart.com
9. Visual Studio Code, Microsoft Corporation. [Online]. Available: https://code.visualstudio.com
10. G.E. Davis, *A Handbook of Chemical Engineering*, 2nd Ed., Vol. II, Davis Bros., Manchester, 1903.

MAY 2025 WATERPROOFING MANAGEMENT SYSTEM NIYAS MOHAMMED BASHEER(SNG23MCA-2064)