PHARMACY MANAGEMENT SYSTEM

# A PROJECT REPORT

**For**

# Major Project (KCA451) Session (2023-24)

## Submitted by

**Anshika Varshney(2200290140034) Diksha Bajpai(2200290140054) Divyam Rastogi(2200290140057)**

Submitted in partial fulfilment of the Requirements for the Degree of

**MASTER OF COMPUTER APPLICATION**

Under the Supervision of Prof.Mr.Rabi.N.Panda ASSOCIATE PROFESSOR



**Submitted to**

**DEPARTMENT OF COMPUTER APPLICATIONS**

KIET Group of Institutions, Ghaziabad Uttar Pradesh-201206

(May 2024)

# CERTIFICATE

Certified that **Anshika Varshney (2200290140034), Diksha Bajpai (22002901400**

**54), Divyam Rastogi (2200290140057)** has/ have carried out the project work having “ **Pharmacy management system**” (**Major Project-KCA 451)** for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU**)** (formerly UPTU), Lucknow under Mr.Rabi.N.Panda supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

## Date:

**Anshika Varshney ( 2200290140034) Diksha Bajpai(2200290140054) Divyam Rastogi(2200290140057)**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

## Prof. Rabi. Narayan Panda Dr. Arun Kumar Tripathi

**Associate Professor Head**

## Department of Computer Applications Department of Computer Applications KIET Group of Institutions, Ghaziabad KIET Group of Institutions, Ghaziabad

ii

**PHARMACY MANAGEMENT SYSTEM**

# ANSHIKA VARSHNEY DIKSHA BAJPAI DIVYAM RASTOGI

**ABSTRACT**

The Pharmacy Management System (PMS) is a comprehensive and user-friendly web- based application designed to revolutionize the operations of pharmacies. Traditional pharmacy processes often involve manual and time-consuming tasks, leading to inefficiencies, errors, and challenges in maintaining accurate inventory records. The Pharmacy Management System addresses these issues by automating key aspects of pharmacy operations, including inventory management, prescription processing, and sales tracking.

The primary objective of the Pharmacy Management System is to enhance the overall efficiency and customer service within pharmacy settings. By leveraging modern web technologies and a robust database backend, the system aims to streamline workflow processes, minimize errors in prescription handling, and provide valuable insights through detailed sales reports and analytics.

# ACKNOWLEDGEMENTS

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, Prof.Rabi.Narayan PandaSir for his guidance, help,

and encouragement throughout my project work.

Their enlightening ideas, comments, and suggestions, Words are not enough to express my gratitude to Dr. Arun Kumar Tripathi, Head Department of Computer Applications, for his insightful comments and administrative help on various occasions. Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness .

## Anshika Varshney

**Diksha Bajpai Divyam Rastogi**

# TABLE OF CONTENTS

|  |  |
| --- | --- |
| Certificate | i |
| Abstract | ii |
| Acknowledgements | iii |
| Table of Contents | iv |
| List of Abbreviations | ix |
| List of Tables | xii |
| List of Figures | xiii |
| 1 **INTRODUCTION** | 1-10 |
| 1.1 BACKGROUND | 1 |
| 1.2 OBJECTIVES OF THE PROJECT | 2 |
| 1.2.1 Automation | 2 |
| 1.2.2 Inventory control | 3 |
| 1.3 SCOPE OF THE PROJECT | 4 |
| 1.3.1 Drug Inventory Management | 4 |
| 1.3.2 Sales and Billing Functionality | 5 |
| 1.3.3 User Authentication and Access Control | 6 |
| 1.4 SIGNIFICANCE OF THE PROJECT | 7 |
| 1.4.1 Project Overview | 8 |
| 1.5 HARDWARE & SOFTWARE SPECIFICATION | 8 |
| 1.5.1 Hardware Specification | 9 |
| 1.5.2 Software Specification | 10 |
| 2 **LITERATURE REVIEW** | 11-16 |
| 2.1 OVERVIEW | 11 |
| 2.2 EVOLUTION OF PHARMACY MANAGEMENT SYSTEMS | 11 |
| 2.3 KEY FEATURES AND FUNCTIONALITIES | 12 |
| 2.3.1 Smart Inventory Control | 12 |
| 2.3.4 Secure User Authentication | 13 |
| 2.3.5 User-Friendly Interface | 13 |
| 2.4 TECHNOLOGICAL FRAMEWORKS | 14 |
| 2.4.1 PHP (Hypertext Preprocessor) | 14 |

Perl) 2.4.2 XAMPP (Cross-Platform, Apache, MariaDB, PHP, and 14

* 1. [CHALLENGES AND SOLUTIONS IN PHARMACY MANAGEMENT 15](#_TOC_250009)
     1. [Challenges 15](#_TOC_250008)
     2. [Solutions 16](#_bookmark0)

1. [FEASIBILITY STUDY 17-21](#_bookmark1)
   1. [PROJECT DESCRIPTION 17](#_bookmark2)
   2. [MARKET ANALYSIS 17](#_TOC_250007)
   3. [TECHNICAL FEASIBILITY 18](#_TOC_250006)
   4. [OPERATIONAL FEASIBILITY 18](#_TOC_250005)
      1. [Financial Feasibility 19](#_TOC_250004)
      2. [Legal and Regulatory Compliance 19](#_bookmark3)
      3. [Risk Analysis 20](#_bookmark4)
      4. [Timeline and Implementation Plan 21](#_bookmark5)
2. REQUIREMENT ANALYSIS 22-31
   1. [FUNCTIONAL REQUIREMENTS 22](#_bookmark6)
      1. [Medication Management 23](#_bookmark7)
      2. [Patient Management 23](#_bookmark8)
      3. [Reporting and Analytics 24](#_bookmark9)
   2. [NON-FUNCTIONAL REQUIREMENTS 24](#_bookmark10)
      1. [Performance 24](#_bookmark11)
      2. [Reliability 25](#_bookmark12)
      3. [Security 25](#_bookmark13)
      4. [Scalability 26](#_bookmark14)
   3. [USE CASE SCENARIOS 26](#_TOC_250003)
      1. [User Authentication and Authorization 27](#_bookmark15)
      2. [Inventory Management 27](#_bookmark16)

[4.3.3. Prescription Processing 28](#_bookmark17)

* + 1. [Sales and Billing 28](#_bookmark18)
    2. Patient Records 29
  1. [DATA REQUIREMENTS 30](#_TOC_250002)
     1. [Patient Data 30](#_bookmark19)
     2. [Medication Data 31](#_bookmark20)
     3. [Sales and Billing Data 31](#_TOC_250001)
     4. [Backup and Recovery Data 31](#_TOC_250000)

1. [SYSTEM DESIGN 31-34](#_bookmark21)
   1. [HIGH LEVEL SYSTEM ARCHITECHTURE 31](#_bookmark22)
   2. [DATABASE DESIGN 31](#_bookmark23)
   3. CLASS DIAGRAM OR ENTITY RELATIONSHIP DIAGRAM 32
      1. [Class Diagram 32](#_bookmark24)
      2. [Entity relationship diagram(ERD) 33](#_bookmark25)
      3. [Data flow diagram 34](#_bookmark26)
2. **DESIGN** 34-38

|  |  |
| --- | --- |
| 6.1 LOGIN PAGE | 34 |
| 6.2 DASHBOARD | 34 |
| 6.3 ADD MANUFACTURER | 35 |
| 6.4 MANAGE MANUFACTURER | 35 |
| 6.5 ADD CATEGORIES | 36 |
| 6.6 MANAGE CATEGORIES | 36 |
| 6.8 INVOICES | 37 |
| 6.8.1 Add Invoices | 37 |
| 6.8.2 Manage invoices | 38 |
| 6.9 Report | 39-42 |
| 7 **IMPLEMENTATION** | 39 |
| 7.1 Overview of PHP | 39 |
| 7.1.1 Key Features | 40 |
| 7.2 DATABASE SETUP AND CONNECTIVITY | 40 |
| 7.3 SECURITY MEASURES IMPLEMENTED | 40 |
| 7.3.1 Database Security | 41 |
| 8 **TESTING** | 44-46 |
| 8.1 TESTING METHODOLOGIES | 42 |
| 8.1.1 Unit Testing | 42 |
| 8.1.2 Integration Testing | 42 |
| 8.1.3 System Testing | 43 |
| 8.1.4 Acceptance Testing | 43 |
| 8.1.5 Security Testing | 43 |
| 8.2 TEST CASES AND RESULTS | 44 |
| 8.3 BUG TRACKING AND RESOLUTION | 44 |
| 8.3.1 Bug Identification | 44 |
| 8.3.2 Bug Logging | 45 |
| 8.3.3 Bug Resolution | 45 |
| 8.3.4 Testing Fixes | 45 |
| 8.3.5 Verification and Validation | 46 |

Conclusion

References & Bibliography

## Figure No.

**Name of Figure Page No.**

* + 1. Class Diagram 16
    2. Entity-Relationship Diagram 17
    3. Data Flow Diagram 18
  1. Login Page 19
  2. Dashboard 19
  3. Add Manufacturer 20
  4. Manage Manufacturer 20
  5. Add Categories 21
  6. Manage Categories 21
  7. Medicines 22
     1. Add Invoices 22
     2. Manage invoices 23
  8. Add Report 24
  9. Manage Report 24

# CHAPTER 1

**INTRODUCTION**

Welcome to the Pharmacy Management System project, an innovative solution poised to transform the landscape of pharmacy operations. In an era of advancing healthcare services, the efficient management of pharmaceutical processes is critical to meet the demands of an ever-evolving industry. This project harnesses the capabilities of PHP to deliver a sophisticated, user-friendly platform, addressing the complex needs of modern pharmacies. In a world where the intersection of healthcare and technology is increasingly integral, the meticulous management of pharmaceutical processes stands as a cornerstone for ensuring the well-being of communities. This project, driven by the dynamic capabilities of PHP, aspires to introduce a sophisticated, user-friendly platform designed to address the multifaceted challenges faced by modern pharmacies.

## BACKGROUND

Pharmacies serve as crucial hubs in healthcare, ensuring the timely and accurate distribution of essential medications to the public. As the healthcare landscape becomes more intricate, the necessity for a streamlined and automated approach to manage pharmacy activities becomes increasingly apparent.Pharmacies, as vital pillars of healthcare provision, shoulder the responsibility of ensuring the timely and accurate distribution of life-saving medications. The complexity of this mission has grown exponentially, driven by advancements in medical science, increased demand for specialized pharmaceuticals, and the ever-expanding scope of healthcare services. This project emerges as a response to the imperative need for a streamlined, automated approach to navigate this intricate web of pharmaceutical activities.

## OBJECTIVES OF THE PROJECT

This project sets forth with a clear set of objectives, each meticulously crafted to address the contemporary challenges faced by pharmacies:

## Automation

The primary goal is to automate and simplify the intricate web of pharmaceutical data management, reducing reliance on manual processes and enhancing overall efficiency. A focus on accurate and efficient inventory management ensures that pharmacies can maintain optimal stock levels, preventing both shortages and excesses.

## Prescription processing

Seamlessly processing prescriptions is at the core of the project, aiming to enhance the speed and accuracy of order fulfilment. The integration of technology into various facets of pharmacy operations is poised to enhance overall efficiency, offering a modernized approach to an age-old profession.

## SCOPE OF THE PROJECT

Encompassing a wide array of functionalities, the scope of this project extends to cover critical aspects of pharmacy management:

## Drug Inventory Management

The system will provide tools for efficient tracking and control of pharmaceutical stocks, ensuring pharmacies are well-equipped to meet the demands of their clientele. Streamlining the process of prescription processing, the system seeks to reduce processing times, minimize errors, and enhance overall prescription management.

## Sales and Billing Functionality

The incorporation of a robust system for sales transactions and billing addresses the financial aspect of pharmacy operations, ensuring transparency and accuracy.

## User Authentication and Access Control

Prioritizing security, the system will implement user authentication mechanisms and access controls to safeguard sensitive pharmaceutical data.

## SIGNIFICANCE OF THE PROJECT

* By deploying an automated system, pharmacies can significantly improve their operational efficiency, reducing the time and effort spent on manual processes.
* Automation inherently minimizes the scope for human errors, especially in critical processes such as prescription processing and inventory management.
* A centralized platform for managing pharmaceutical data ensures that pharmacies can

adhere to regulatory requirements, maintain accurate records, and respond swiftly to changing conditions.

* In the pursuit of these objectives, this Pharmacy Management System project emerges as not just a technological innovation but as a catalyst for enhancing the quality of

healthcare services and contributing to the resilience and adaptability of pharmacies in an ever-evolving healthcare landscape.

* + 1. **Project Overview**
       - Front End : PHP (5.5)
       - Back End : MYSQL

## HARDWARE & SOFTWARE SPECIFICATION

* + 1. **Hardware Specification**

Hardware specifications are crucial for a pharmacy system project for several reasons:

1. Performance: Hardware specifications determine the system's performance capabilities. A pharmacy system must handle a significant amount of data processing, including medication inventory management, prescription filling, and patient records. Adequate hardware specifications ensure that the system can perform these tasks efficiently, without slowdowns or delays.
2. Scalability: Hardware specifications influence the system's scalability, i.e., its ability to handle increased workload as the pharmacy grows or experiences higher demand. Scalable hardware allows the system to accommodate additional users, transactions, and data volumes without performance degradation.
3. Reliability: The reliability of hardware components impacts the overall reliability of the pharmacy system. Reliable hardware reduces the risk of system failures, downtime, and data loss, ensuring continuous operation and uninterrupted access to critical pharmacy functions.
4. Security: Hardware specifications play a role in ensuring the security of the pharmacy system. Secure hardware components, such as encrypted storage drives and hardware-based security features, help protect sensitive patient information and prevent unauthorized access or data breaches.
5. Integration: Hardware specifications may influence the system's compatibility and ability to integrate with other hardware devices or systems used in the pharmacy, such as barcode scanners, label printers, and payment terminals. Compatibility ensures seamless communication and interoperability between different components of the pharmacy system.
6. Regulatory Compliance: Certain hardware specifications may be required to comply with regulatory standards and industry regulations governing pharmacy operations and data security. For example, hardware encryption may be necessary to comply with data protection regulations such as HIPAA (in the US) or GDPR (in the EU).
7. Cost-effectiveness: Determining appropriate hardware specifications ensures that the pharmacy system meets its performance and reliability requirements without overspending on unnecessary or overly powerful hardware components. A balance between performance, reliability, and cost-effectiveness is essential in selecting hardware for the project.

Overall, hardware specifications are essential for designing, implementing, and maintaining a pharmacy system that meets the functional requirements, performance

expectations, and regulatory compliance standards of the pharmacy industry.

* + Processor : Intel Core Duo 2.0 GHz or higher.
  + RAM : Minimum1 GB or Greater.
  + Hard disk : 20 GB (Free Space).
    1. **Software Specification**

Software specifications are essential for a pharmacy system project for several reasons:

1. \*Functionality Definition: Software specifications define the functional requirements of the pharmacy system, including features such as prescription management, inventory tracking, patient records management, billing, and reporting. Clear software specifications ensure that the system meets the specific needs of the pharmacy and its users.
2. User Experience (UX) Design: Software specifications outline the user interface design, navigation flow, and interaction elements of the pharmacy system. A well-defined UX design enhances usability, making it easier for pharmacy staff to navigate the system, perform tasks efficiently, and minimize errors.
3. System Integration: Software specifications specify how the pharmacy system interacts with other software components, databases, and external systems such as electronic health records (EHRs), insurance providers, and regulatory databases. Seamless integration ensures data consistency, interoperability, and efficient exchange of information between different systems.
4. Performance Requirements: Software specifications detail performance requirements such as response times, throughput, and scalability. These requirements ensure that the pharmacy system can handle the expected workload, accommodate growth in data volume and user traffic, and deliver optimal performance under varying conditions.
5. Data Management: Software specifications define how data is collected, stored, processed, and secured within the pharmacy system. This includes data models, database schema, data validation rules, encryption mechanisms, backup and recovery procedures, and compliance with data privacy regulations such as HIPAA or GDPR.
6. Quality Assurance (QA) Testing: Software specifications serve as a basis for QA testing to verify that the pharmacy system meets functional, performance, security, and usability requirements. Testing against software specifications helps identify and rectify defects, ensure system reliability, and enhance user satisfaction.
7. Regulatory Compliance: Software specifications ensure that the pharmacy system complies with relevant regulatory standards and industry best practices governing healthcare IT systems. This includes adherence to standards such as HL7 for data exchange, FDA regulations for medication tracking, and security standards such as PCI DSS for payment processing.
8. Change Management: Software specifications provide a baseline for managing changes and updates to the pharmacy system over time. Changes to software specifications must be carefully documented, evaluated for impact, and implemented to ensure system stability, compatibility, and regulatory compliance.

Overall, software specifications serve as a blueprint for designing, developing, testing, and maintaining a pharmacy system that meets the functional, performance, security, and regulatory requirements of the pharmacy industry. They help ensure that the system effectively supports pharmacy operations, enhances patient care, and complies with industry standards and regulations.

* + Software : XAMPP.
  + Operation System : Windows 7 or higher.

# CHAPTER 2

**LITERATURE REVIEW**

## OVERVIEW

In the pursuit of developing an effective Pharmacy Management System, a comprehensive exploration of existing literature is indispensable. This chapter endeavours to provide a thorough review of relevant studies, research papers, and technological frameworks that form the foundational knowledge for the proposed system. By examining the evolution of Pharmacy Management Systems, key features and functionalities, technological frameworks, challenges, solutions, case studies, and identifying gaps in the existing literature, this chapter aims to synthesize a holistic understanding of the current landscape and inform the design and development of the proposed system.

The project is all about creating a special tool, a Pharmacy Management System. Its job is to help pharmacies run smoothly and efficiently. We want to use technology to make things better for everyone involved.

We're using a language called PHP, like a magic spell for the internet, and databases, which are like super-organized shelves for storing pharmacy information. These tools will help our system be smart and organized.

## EVOLUTION OF PHARMACY MANAGEMENT SYSTEMS

The historical progression of Pharmacy Management Systems has witnessed a paradigm shift from manual, paper-based processes to sophisticated, technology-driven solutions. Early pharmacy operations heavily relied on manual record-keeping, leading to inefficiencies, data inaccuracies, and increased susceptibility to errors. As technological advancements burgeoned, the healthcare industry recognized the need for automated systems to streamline processes, improve accuracy, and enhance overall efficiency. The evolution of Pharmacy Management Systems reflects a continual quest for innovation, driven by the growing complexities of pharmaceutical operations and the imperative to meet evolving healthcare standards.

Pharmacies did everything by hand, like keeping track of medicines and prescriptions using paper and pens. Then, smart folks had the idea to use computers, giving birth to Pharmacy Management Systems. These systems evolved to become super-smart assistants for pharmacies, ensuring efficient stock management and quick prescription processing. Now, our Pharmacy Management System project is taking this evolution further by using modern tools like PHP

and databases to make the system even more efficient and powerful

## KEY FEATURES AND FUNCTIONALITIES

* + 1. **Smart Inventory Control**

The system meticulously tracks and manages the inventory of medicines. Ensures real- time updates on stock levels, preventing shortages and overstock situations. Accelerates the processing of prescriptions to minimize waiting times for customers and enhance the overall speed and accuracy of prescription fulfilment. Efficient manages sales transactions seamlessly, ensuring accurate and transparent billing and Keeps a comprehensive record of financial transactions for streamlined accounting.

## Secure User Authentication

A robust user authentication system to control access to sensitive information .Ensures that only authorized personnel can interact with critical pharmacy data.

## TECHNOLOGICAL FRAMEWORKS

* + 1. **PHP (Hypertext Pre-processor)**

PHP is a server-side scripting language widely used for web development. It allows the creation of dynamic and interactive web pages by embedding code within HTML. Role in the Project: Primary language for backend development in our Pharmacy Management System

.Facilitates server-side processing, ensuring seamless interaction with the database.

## XAMPP (Cross-Platform, Apache, MariaDB, PHP, and Perl)

XAMPP is an open-source software package providing a local server environment. It includes Apache (web server), MariaDB (database system), PHP, and Perl. Role in the Project: Serves as the local server for development, testing, and deployment. Ensures a comprehensive environment for PHP scripts to run, interact with the database, and serve web pages.

## CHALLENGES AND SOLUTIONS IN PHARMACY MANAGEMENT

The implementation of Pharmacy Management Systems is not without its challenges. Common issues include concerns related to data security, discrepancies in inventory management, and the need for seamless integration with other healthcare systems.

## Challenges

* + - * Maintaining accurate inventory levels can be challenging, resulting in stockouts or overstock situations.
      * Manual prescription processing can lead to delays in medication dispensing.
      * Ensuring seamless integration with other healthcare systems may pose a challenge.

## Solutions

* + - * Implementation of the Pharmacy Management System automates data handling, minimizing errors and streamlining processes**.**
      * The system includes a robust inventory control feature, providing real-time tracking and automated alerts for restocking. Implementation of user authentication and access

controls within the system ensures data security and confidentiality

# CHAPTER 3

**FEASIBILITY STUDY**

A feasibility study is a crucial step in the early stages of any project, including the development of a Pharmacy Management System using PHP. It helps assess the viability of the project and determines whether it is worth pursuing. Here's a general outline of what a feasibility study for a Pharmacy Management System might include:

## PROJECT DESCRIPTION

In this section, provide a comprehensive and detailed overview of the Pharmacy Management System project. Elaborate on the specific goals and objectives, emphasizing how the proposed system aims to enhance the efficiency and effectiveness of pharmacy operations. Define the scope of the project by outlining the key features and functionalities expected, including but not limited to prescription management, inventory control, patient information tracking, billing, and reporting capabilities. Emphasize the potential positive impacts on workflow, customer service, and overall pharmacy management.

## MARKET ANALYSIS

Conduct an in-depth analysis of the pharmaceutical industry, considering current market trends, demands, and the prevailing competitive landscape. Highlight the increasing reliance on digital solutions within the industry and showcase how a Pharmacy Management System aligns with these trends. Provide a thorough examination of existing systems in the market, assessing their strengths and weaknesses, and emphasize the unique selling points that the proposed system could offer.

## TECHNICAL FEASIBILITY

Explore the technical intricacies of implementing the Pharmacy Management System. Discuss the necessary hardware, software, and networking infrastructure required for seamless operation. Evaluate the compatibility of the proposed system with existing technologies within the pharmacy and its ability to integrate with other healthcare systems, such as electronic health records (EHRs) or health information exchange (HIE) platforms.

## OPERATIONAL FEASIBILITY

Delve into the operational aspects, analysing how well the Pharmacy Management System aligns with the current processes and workflows of the pharmacy. Identify potential bottlenecks, disruptions, or resistance from pharmacy staff during the implementation phase. Propose strategies to minimize any negative operational impacts and emphasize how the system can optimize day-to-day tasks.

Operational feasibility refers to the assessment of whether a proposed project or system can be implemented and operated effectively within the organization's existing operational environment. It evaluates the practicality, viability, and suitability of implementing the project or system based on various operational factors. Here's a brief overview:

1. Resource Availability: Operational feasibility assesses whether the necessary resources, including personnel, expertise, technology, infrastructure, and budget, are available or can be acquired within a reasonable timeframe to support the project or system implementation.
2. Organizational Capability: It evaluates the organization's capability and readiness to adopt and integrate the proposed project or system into its existing processes, workflows, and organizational structure. This includes assessing the organization's culture, leadership support, change management capacity, and willingness to embrace new technologies or practices.
3. Technical Compatibility: Operational feasibility examines whether the proposed project or system is technically compatible with existing systems, hardware, software, and IT infrastructure. It assesses the potential impact on IT resources, compatibility issues, interoperability challenges, and the need for system integration or customization.
4. Operational Impact: It considers the potential operational impact of implementing the project or system on day-to-day business operations, workflows, and employee productivity. This includes assessing any disruptions, downtime, or workflow changes that may occur during implementation and transition phases.
5. User Acceptance: Operational feasibility evaluates whether end-users and stakeholders are willing and able to accept and adopt the proposed project or system. It considers factors such as user preferences, expectations, training needs, and resistance to change, as well as the system's usability, intuitiveness, and alignment with user needs.
6. Cost-Benefit Analysis: It includes a cost-benefit analysis to determine whether the benefits of implementing the project or system outweigh the costs and risks involved. This involves quantifying both tangible and intangible benefits and costs, such as increased efficiency, reduced errors, improved decision- making, and potential revenue generation, as well as upfront and ongoing expenses, risks, and uncertainties.

Overall, operational feasibility assessment plays a crucial role in determining the viability and success of a proposed project or system by evaluating its alignment with organizational goals, capabilities, and constraints. It helps stakeholders make informed decisions about whether to proceed with the project, modify its scope or approach, or explore alternative solutions based on practical considerations and realities.

## Financial Feasibility

Provide a detailed breakdown of the financial aspects associated with the implementation of the Pharmacy Management System. This should include development costs, hardware and software expenses, training costs, and ongoing maintenance expenses. Conduct a thorough cost-benefit analysis, emphasizing the potential return on investment (ROI) and showcasing the long-term financial advantages of adopting the proposed system.

## Legal and Regulatory Compliance

Scrutinize the legal and regulatory landscape, ensuring that the proposed Pharmacy Management System complies with all relevant laws and standards pertaining to healthcare data privacy and security. Highlight the system's ability to maintain patient confidentiality and comply with regulations such as Health Insurance Portability and Accountability Act (HIPAA) in the United States or other applicable regulations in different regions.

## Risk Analysis

Undertake a comprehensive risk analysis, identifying potential challenges and obstacles that may arise during the project lifecycle. This could include resistance from pharmacy staff, technical challenges, or changes in regulatory requirements. Develop robust risk mitigation strategies and contingency plans to address these potential obstacles, demonstrating a proactive approach to risk management.

## Timeline and Implementation Plan

Provide a detailed timeline outlining the key milestones and phases of the Pharmacy Management System project. Clearly define the development, testing, and implementation phases, ensuring that the timeline is realistic and achievable. Integrate key performance indicators (KPIs) to monitor progress and identify any deviations from the proposed schedule.

# CHAPTER 4

**REQUIREMENT ANALYSIS**

## FUNCTIONAL REQUIREMENTS

Developing a robust pharmacy management system requires a clear understanding of its functional requirements. These requirements define what the system should do to effectively support pharmacy operations and meet the needs of users, including pharmacists, nurses, doctors, and patients.

Here's a breakdown of key functional requirements for a pharmacy management system: Functional requirements are essential for several reasons:

1. Defines System Behavior: Functional requirements specify what the system should do. They outline the functions, features, and capabilities that the system must possess to meet the needs of its users. Without functional requirements, there is no clear understanding of what the system should accomplish.
2. Guides Development: Functional requirements serve as a roadmap for developers, guiding them in the design, implementation, and testing of the system. They provide a clear set of objectives and criteria against which the system's performance can be evaluated.
3. Basis for System Testing: Functional requirements form the basis for system testing. Test cases are developed based on the functional requirements to ensure that the system behaves as intended and meets the users' needs.
4. Alignment with User Expectations: Functional requirements help ensure that the system aligns with the expectations and needs of its users. By clearly specifying the desired functions and features, functional requirements help prevent misunderstandings and ensure that the final product meets user expectations.
5. Facilitates Communication: Functional requirements serve as a common language for communication between stakeholders, including developers, designers, testers, and users. They provide a shared understanding of the system's functionality and help facilitate collaboration and alignment among team members.
6. Basis for Change Management: Functional requirements provide a baseline against which changes to the system can be evaluated. Any proposed changes can be assessed in terms of their impact on the system's functionality, and decisions can be made based on the alignment with the functional requirements.
7. Supports Documentation: Functional requirements serve as documentation for the system, capturing the desired behavior and functionality in a structured format. This documentation is valuable for future reference, maintenance, and enhancement of the system.

In summary, functional requirements are crucial for defining, developing, testing, and maintaining a system that meets the needs of its users and stakeholders. They provide a clear understanding of what the system should do and serve as a foundation for its design, implementation, and evaluation.

Functional requirements can be categorized into several types, each focusing on different aspects of system functionality. Here are some common types of functional requirements:

1. User Authentication and Authorization: These requirements specify how users will authenticate themselves to the system (e.g., username and password) and what actions they are authorized to perform based on their roles or permissions.
2. Data Input and Output: These requirements describe the formats, sources, and methods for inputting data into the system (e.g., through forms, APIs) and how the system should process and present output data to users (e.g., reports, notifications).
3. Data Manipulation and Processing: These requirements detail how the system should manipulate and process data to perform specific functions or tasks (e.g., calculations, transformations, validations).
4. User Interface (UI) and User Experience (UX): These requirements specify the layout, design, and usability features of the system's user interface to ensure that it is intuitive, accessible, and meets user needs and expectations.
5. Functional Dependencies and Interactions: These requirements describe how different system components and functions depend on each other and interact to achieve overall system behavior (e.g., workflow dependencies, integration with external systems).
6. Error Handling and Recovery: These requirements outline how the system should handle errors, exceptions, and invalid inputs gracefully, providing appropriate feedback to users and enabling them to recover from errors effectively.
7. Reporting and Logging: These requirements specify the types of reports and logs that the system should generate (e.g., transaction logs, audit trails) and how users can access and analyze this information to monitor system activity and performance.
8. Security and Compliance: These requirements address the security measures and compliance standards that the system must adhere to (e.g., data encryption, access control, regulatory requirements) to protect sensitive information and ensure legal and regulatory compliance.
9. Performance and Scalability: These requirements define the performance objectives and scalability capabilities of the system (e.g., response time, throughput, scalability under load) to ensure that it can handle expected workloads and scale effectively as usage grows.
10. System Integration and Interoperability: These requirements specify how the system should integrate with other systems, services, or devices (e.g., APIs, data formats, protocols) to exchange data and functionality seamlessly.

These are just some examples of functional requirements, and the specific types may vary depending on the nature of the system, its intended use, and the requirements of stakeholders.

## Medication Management:

* + Inventory tracking: Maintain accurate records of medication quantities, locations, expiry dates, and batch numbers.

•Drug interaction checking: Integrate with drug databases to identify and alert potential medication interactions.

## Patient Management:

Patient profiles: Create and maintain patient profiles with demographic information, medical history, allergies, and medication lists.

Medication adherence monitoring: Track patient medication adherence and provide interventions when needed.

## Reporting and Analytics:

* + - * Generate reports on medication usage, inventory levels, dispensing trends, and other key metrics.
      * Track and analyse pharmacy performance to ensure compliance with regulations and best practices.

## NON-FUNCTIONAL REQUIREMENTS:

## Performance:

The system should respond to user requests within 2 seconds to ensure a seamless user experience. It should support a minimum of 100 concurrent users during peak hours.

Performance is indeed a key aspect of non-functional requirements. Non-functional requirements define the qualities or attributes that a system must have, beyond its specific functionalities. Performance requirements focus on how well the system performs in terms of speed, responsiveness, scalability, and efficiency. Here's how performance fits into non-functional requirements:

1. Speed: Performance requirements often specify the maximum response times for various system operations. For example, a performance requirement might state that the system must respond to user requests within two seconds.
2. Responsiveness: This aspect refers to how quickly the system reacts to user inputs or events. Performance requirements may include criteria for the system's responsiveness, such as ensuring that user interface interactions are smooth and without noticeable delays.
3. Scalability: Performance requirements also address the system's ability to handle increasing workloads or user demands without significantly degrading performance. This includes both vertical scalability (adding resources to a single system) and horizontal scalability (distributing workload across multiple systems).
4. Throughput: Performance requirements may specify the maximum number of transactions or operations that the system should be able to handle within a given time frame. For example, a requirement might state that the system should support a minimum of 1000 transactions per second.
5. Resource Utilization: Performance requirements may include constraints on resource usage, such as CPU, memory, or disk space, to ensure that the system operates efficiently and does not exceed hardware limitations.
6. Load Testing: Non-functional requirements often involve load testing to evaluate the system's performance under expected and peak loads. This helps identify potential bottlenecks, optimize system configurations, and ensure that performance goals are met.
7. Reliability: Performance requirements also encompass reliability aspects, such as ensuring that the system maintains consistent performance levels over time and under varying conditions, without experiencing unexpected downtime or failures.
8. Efficiency: Non-functional requirements address the system's efficiency in utilizing resources and achieving its objectives with minimal waste or overhead. This includes optimizing algorithms .

## 4.2.1 Reliability:

The system should have a 99.9% uptime, ensuring that it is available for use by pharmacy staff at all times. It should have a backup and recovery mechanism to prevent data loss in case of system failures. The system should be compatible with commonly used web browsers (e.g., Chrome, Firefox, Safari) to ensure widespread accessibility. It should be compatible with standard operating systems (e.g., Windows, macOS, Linux).

## Security:

User data should be stored securely, and access to sensitive information should be restricted based on user roles. The system should encrypt communication between the server and clients.

## Scalability:

The system should be designed to handle an increase in the number of pharmacies using the system. It should be scalable to accommodate a growing database of patients and medications. The user interface should be intuitive, and pharmacists should be able to perform common tasks with minimal training.

## USE CASE SCENARIOS:

* + 1. **User Authentication and Authorization: Use Case**: Pharmacist Login

**Scenario:**A pharmacist logs into the system with their credentials.

## Steps:

Pharmacist enters username and password. System verifies credentials.

## Sales and Billing:

**Use Case:**Generate Invoice

**Scenario:**A pharmacist generates an invoice for a customer.

## Steps:

Pharmacist selects the medications sold. System calculates the total cost.

Pharmacist generates and prints the invoice for the customer.

## Patient Records:

**Use Case:**Search Patient Records

**Scenario:**A pharmacist searches for a patient's record.

## Steps:

Pharmacist enters patient details (name, ID, etc.). System retrieves and displays the patient's record.

These use case scenarios provide a high-level overview of how users (pharmacists in this case) interact with the Pharmacy Management System to perform various tasks. Each scenario outlines the steps involved in achieving a specific goal within the system, helping to understand the flow of interactions and functionalities.

## DATA REQUIREMENTS

Data requirements for a Pharmacy Management System (PMS) involve specifying the types of data that the system needs to store, manage, and manipulate. Here are key data requirements for a PMS:

## Patient Data:

Fields: Name, date of birth, contact information, address, insurance details.

Purpose: To maintain a comprehensive record of patients for prescription tracking and personalized care.

## Medication Data:

Fields: Medication name, dosage, expiration date, manufacturer, stock quantity.

Purpose: To track and manage the pharmacy's inventory, ensuring accurate dispensing and timely reordering.

## Sales and Billing Data:

Fields: Invoice ID, date of sale, medications sold, quantity, total cost, payment details. Purpose: To track sales, generate invoices, and manage billing information for accounting

## Backup and Recovery Data:

Fields: Backup timestamps, backup location.

Purpose: To support data recovery in case of system failures or data loss.

# CHAPTER 5

**SYSTEM DESIGN**

## HIGH-LEVEL SYSTEM ARCHITECTURE:

The high-level system architecture for a Pharmacy Management System (PMS) typically involves multiple components working together to deliver the required functionalities. Here's a high-level overview:

## User Interface (UI):

Description: The front-end component that interacts with users (pharmacists, administrators, etc.).

Key Features:

1. User authentication and login.
2. Dashboard for quick access to relevant information.

## Application Layer:

Description: The logic and business rules governing the system's functionalities.

Key Features:

1. Inventory management logic.
2. Billing and invoicing processes.
3. User authentication and authorization.

## Database Management System (DBMS):

Description: The backend database where all the system data is stored. Key Features:

1. Structured storage of patient records, medication data, prescription details, etc.
2. Ensuring data integrity and consistency.

## Server:

Description: The central server that hosts the application logic and manages communication with the database.

Key Features:

1. Handles user requests and processes business logic.
2. Manages data transactions with the database.

Description: Provides tools for generating reports and analyzing system data. Key Features:

1. Reporting tools for administrators and managers.
2. Analytics for inventory management and sales performance.

## Backup and Recovery:

Description: Manages data backup and recovery processes to prevent data loss. Key Features:

1. Scheduled backups of the database.
2. Recovery mechanisms in case of system failures. User Management:

Description: Handles user roles, permissions, and authentication. Key Features:

1. User account creation, modification, and deletion.
2. Role-based access control. Communication Protocols:

This high-level architecture provides a foundation for designing and developing a Pharmacy Management System. It's important to consider scalability, maintainability, and security throughout the design process. The specific technologies and frameworks used will depend on the development team's expertise and the organization's requirements.

## DATABASE DESIGN

Database design is the process of creating a detailed data model for a database. It involves defining the structure of the database, including the tables, columns, relationships, and constraints that will store and manage the data efficiently. Here's an overview of the key aspects of database design:

1. Requirements Analysis: The first step in database design is to understand the requirements of the system and the data it will manage. This involves identifying the types of data to be stored, the relationships between different data entities, and the operations that will be performed on the data.
2. Conceptual Design: In this phase, a conceptual data model is created to represent the high-level structure of the database, independent of any specific database management system (DBMS). This often involves using techniques like Entity-Relationship Diagrams (ERDs) to model entities, attributes, and relationships.
3. Logical Design: The logical design phase involves translating the conceptual data model into a logical data model that can be implemented in a specific DBMS. This includes defining tables, columns, data types, and relationships, as well as identifying primary and foreign keys to enforce data integrity.
4. Normalization: Normalization is the process of organizing the data in a database to minimize redundancy and dependency. It involves breaking down tables into smaller, more manageable

pieces and establishing rules to ensure data consistency and integrity.

1. Physical Design: The physical design phase involves optimizing the database for performance, scalability, and efficiency. This includes decisions such as choosing appropriate storage structures, indexing strategies, and partitioning schemes to optimize data access and manipulation.
2. Security and Access Control: Database design also involves considering security requirements and implementing measures to protect sensitive data from unauthorized access, manipulation, or disclosure. This may include encryption, access control mechanisms, and auditing features.
3. Data Migration and Integration: If migrating from an existing system or integrating with other systems, database design includes planning and executing data migration and integration strategies to ensure that data is transferred accurately and efficiently.
4. Testing and Optimization: Once the database is designed and implemented, it undergoes testing to ensure that it meets the functional and performance requirements. Optimization techniques, such as query tuning and index optimization, may be applied to improve database performance.
5. Documentation and Maintenance: Database design also involves documenting the database schema, data dictionary, and other relevant information to aid in system maintenance, troubleshooting, and future development efforts.

Overall, effective database design is critical for creating a robust, scalable, and efficient database system that meets the needs of the organization and its users. It requires careful analysis, planning, and execution to ensure that the database structure and functionality support the desired business processes and objectives.

Provide an ERD to visually represent the relationships between different entities in thedatabase. This should include entities such as:

Patient Medication Prescription Doctor Pharmacist Inventory Supplier

Include relationships, cardinalities, and attributes for each entity.

## Database Tables

In a relational database, data is organized into tables. Each table consists of rows and columns, where each row represents a single record or entity, and each column represents a specific attribute or field of that record. Here's a breakdown of the key components of a database table:

1. Table Name: Each table is given a unique name that reflects the type of data it stores. Table names should be descriptive and meaningful to facilitate understanding and management of the database schema.
2. Columns (Attributes): Columns define the structure of the data stored in the table. Each column has a name and a data type that specifies the kind of data it can store (e.g., text, numeric, date). Columns represent the individual properties or characteristics of the entities being modeled.
3. Rows (Records): Rows, also known as records or tuples, represent individual instances of data stored in the table. Each row corresponds to a single entity or record in the database and contains values for each column defined in the table schema.
4. Primary Key: A primary key is a column or combination of columns that uniquely identifies each row in the table. It ensures that there are no duplicate rows and provides a means for referencing and retrieving specific records efficiently.
5. Foreign Key: A foreign key is a column or set of columns in a table that establishes a relationship with another table's primary key. It enforces referential integrity by ensuring that values in the foreign key column(s) match values in the primary key column(s) of the related table.
6. Constraints: Constraints are rules or conditions that enforce data integrity and consistency within the table. Common constraints include NOT NULL (to enforce the presence of values), UNIQUE (to enforce uniqueness of values), and CHECK (to enforce specific conditions on column values).
7. Indexes: Indexes are data structures that improve the performance of data retrieval operations by enabling faster lookup of records based on specific column values. They are created on one or more columns to facilitate efficient searching and sorting of data.
8. Metadata: Metadata refers to additional information about the table, such as its creation date, last modification date, owner, and any comments or descriptions provided by the database designer or administrator. Metadata helps in understanding and managing the table within the database environment.

Overall, database tables serve as the fundamental building blocks for organizing and storing data in a relational database system. They provide a structured and efficient way to represent and manage data, supporting various operations such as querying, inserting, updating, and deleting records.

## Patient Table:

PatientID (Primary Key) FirstName

LastName DateOfBirth ContactNumber Address

## Medication Table:

MedicationID (Primary Key) Medication Name Manufacturer

Dosage PricePerUnit

## Prescription Table:

PrescriptionID (Primary Key) PatientID (Foreign Key) DoctorID (Foreign Key) DatePrescribed

## Pharmacist Table:

PharmacistID (Primary Key) FirstName

LastName ContactNumber

## Supplier Table:

SupplierID (Primary Key) SupplierName ContactNumber

Address

## Data Types and Constraints:

Define the data types for each attribute and specify any constraints such as NOT NULL, UNIQUE, DEFAULT, etc.

## Indexing and Optimization:

Discuss indexing strategies for critical fields to optimize query performance.

## Database Security:

Outline the security measures implemented to protect sensitive information, including user authentication and authorization.

## Backup and Recovery:

Describe the backup and recovery procedures to ensure data integrity and availability.

## Database Maintenance Plan:

Provide a plan for routine maintenance tasks such as updates, patches, and performance monitoring.

## Database Testing:

Outline the testing procedures for the database design, including unit testing, integration testing, and performance testing.

## CLASS DIAGRAMS OR ENTITY-RELATIONSHIP DIAGRAMS

* + 1. **Class Diagrams**

Class diagrams in the context of a pharmacy management system help to represent the static structure of the system by illustrating the classes, their attributes, methods, and relationships.

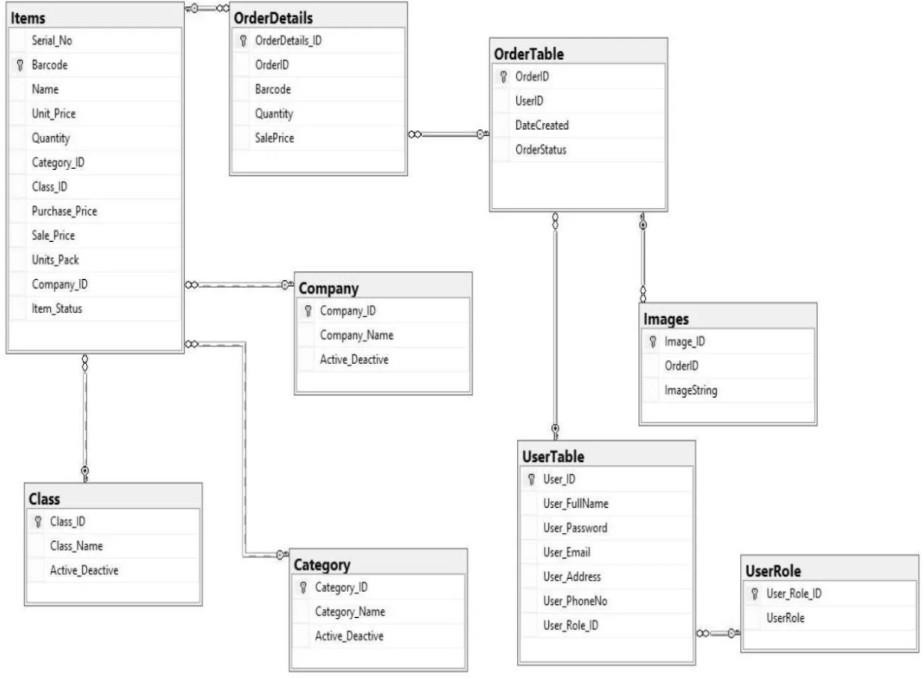
Class diagrams are useful for several reasons:

* + - 1. Visual Representation of the System: Class diagrams provide a visual representation of the structure and relationships within a system. They depict the classes, attributes, methods, and associations in a clear and concise manner, making it easier for stakeholders to understand the system's architecture and design.
      2. Modeling Object-Oriented Concepts: Class diagrams are particularly useful for modeling object-oriented systems, where classes represent the fundamental building blocks of the system. They enable developers to visualize the classes and their relationships, inheritance hierarchies, and other object-oriented concepts such as encapsulation, polymorphism, and abstraction.
      3. Communication Tool: Class diagrams serve as a communication tool between stakeholders involved in the system development process, including developers, designers, architects, and clients. They provide a common language and notation for discussing and documenting the system's structure and behavior, fostering collaboration and alignment among team members.
      4. Blueprint for Implementation: Class diagrams provide a blueprint for implementing the system in code. They serve as a guide for developers, helping them understand the classes and their responsibilities, design class interfaces, and establish relationships and dependencies between classes.
      5. Analysis and Design Validation: Class diagrams support analysis and design validation by enabling stakeholders to visualize and analyze the system's structure and behavior before implementation begins. They help identify potential design flaws, inconsistencies, or missing requirements early in the development process, reducing the risk of costly errors later on.
      6. Documentation: Class diagrams serve as documentation for the system, capturing important aspects of its design, including class definitions, relationships, and behavior. They provide a reference for developers, maintainers, and other stakeholders involved in the system's lifecycle, facilitating maintenance, troubleshooting, and future development efforts.
      7. Code Generation: Class diagrams can be used to generate code automatically or semi-automatically, particularly in model-driven development environments. Tools and frameworks exist that can translate class diagrams into code skeletons, saving time and effort in the implementation phase.

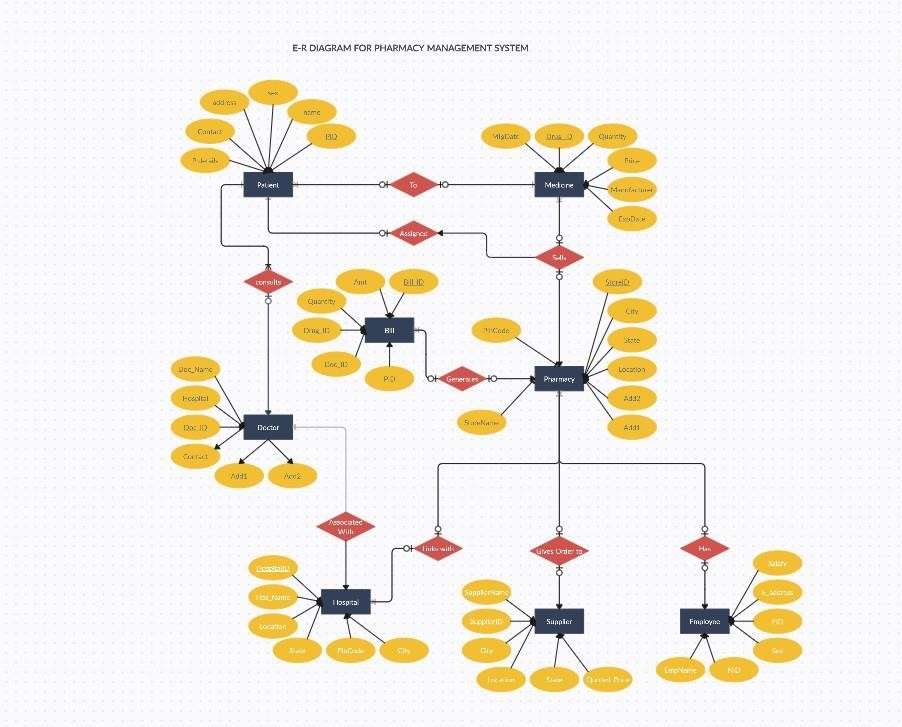
Overall, class diagrams are valuable tools for modeling, communicating, and documenting the structure and behavior of object-oriented systems, helping stakeholders to understand, analyze, and develop software systems effectively.

Key Elements:-

* **Classes:** Represent entities or objects in the system. Examples include Patient, Medication, Prescription, Doctor, Pharmacist, Inventory, and Supplier.
* **Attributes:**Properties of each class. For instance, the Patient class might have attributes like PatientID, FirstName, LastName, DateOfBirth, etc.
* **Methods/Operations:**Actions that can be performed on the classes. For example, the Prescription class might have methods like fillPrescription().
* **Relationships:**Connections between classes, indicating associations. For instance, a Prescription class might have associations with Patient and Doctor Classes.



## Fig 5.3.1: Class Diagram



**Fig 5.3.2: Entity Relationship Diagram**

## Entity Relationship Diagram

ERD focuses on depicting the relationships between entities within the system. It emphasizes the connections between different entities and their attributes.

Entity-Relationship (ER) diagrams are useful for several reasons:

* + - 1. Visual Representation of Data Model: ER diagrams provide a visual representation of the data model of a system. They illustrate the entities (objects), attributes (properties), and relationships between entities in a clear and concise manner, making it easier for stakeholders to understand the structure of the data.
      2. Modeling Complex Relationships: ER diagrams are particularly useful for modeling complex relationships between entities in a database. They allow for the representation of one-to-one, one-to-many, and many-to-many relationships, as well as recursive relationships, inheritance hierarchies, and other complex structures.
      3. Communication Tool: ER diagrams serve as a communication tool between stakeholders involved in the system development process, including developers, designers, architects, and clients. They provide a common language and notation for discussing and documenting the data model, fostering collaboration and alignment among team members.
      4. Database Design and Implementation: ER diagrams provide a blueprint for designing and implementing the database schema. They help developers understand the structure and relationships of the data, design tables and columns accordingly, and establish constraints and dependencies to ensure data integrity and consistency.
      5. Normalization and Optimization: ER diagrams support the process of normalization, which involves organizing the data in a database to minimize redundancy and dependency. They help identify potential normalization opportunities and optimize the database schema for efficiency and performance.
      6. Analysis and Design Validation: ER diagrams support analysis and design validation by enabling stakeholders to visualize and analyze the structure and relationships of the data model before implementation begins. They help identify potential design flaws, inconsistencies, or missing requirements early in the development process, reducing the risk of costly errors later on.
      7. Documentation: ER diagrams serve as documentation for the database schema, capturing important aspects of its design, including entity definitions, attribute definitions, and relationship definitions. They provide a reference for developers, maintainers, and other stakeholders involved in the system's lifecycle, facilitating maintenance, troubleshooting, and future development efforts.

Overall, ER diagrams are valuable tools for modeling, communicating, and documenting the data model of a system, helping stakeholders to understand, analyze, and develop database systems effectively.

Key Elements:

* **Entities:**Correspond to tables in the database. Patient, Medication, Prescription, Doctor, Pharmacist, Inventory, and Supplier would be entities in a pharmacy

management system.

* **Attributes:**Characteristics of each entity. For instance, Patient entity might have attributes like PatientID, FirstName, LastName, etc.
* **Relationships:**Illustrate how entities are related to each other. For example, a Prescription entity might have relationships with Patient and Doctor entities.

## Data flow Diagram:

A data flow diagram (DFD) is a graphical representation of the flow of data through a system. It illustrates how data moves from one process to another, how it is stored, and how it is transformed within the system. Here's a breakdown of its components:

Data Flow Diagrams (DFDs) are important-:

* + - 1. Visualization of System Processes: DFDs provide a visual representation of how data flows through a system, illustrating the processes, inputs, outputs, and data stores involved. This visualization helps stakeholders understand the system's functionality and behavior.
      2. Understanding System Requirements: DFDs help in understanding and documenting system requirements by breaking down complex systems into manageable components and showing how they interact. They facilitate communication between stakeholders and ensure a common understanding of the system's functionality.
      3. Identifying Data Flow and Transformations: DFDs highlight the flow of data within a system, showing how data is input, processed, and output. They help identify data transformations, validations, and manipulations performed by the system, which is crucial for system design and development.
      4. Detecting Redundancy and Inefficiency: DFDs can reveal redundancies, inefficiencies, and bottlenecks in a system's data flow. By analyzing the diagram, stakeholders can identify areas where data is processed or stored unnecessarily or where processes can be optimized for better performance.
      5. Supporting System Design and Development: DFDs serve as a basis for system design and development. They provide a blueprint for designing system components, specifying data flows, interfaces, and dependencies. Developers can use DFDs to guide the implementation of the system's functionality.
      6. Facilitating System Testing and Validation: DFDs aid in system testing and validation by providing a clear understanding of the system's behavior and functionality. Test cases can be derived from the data flows and processes depicted in the diagram, ensuring

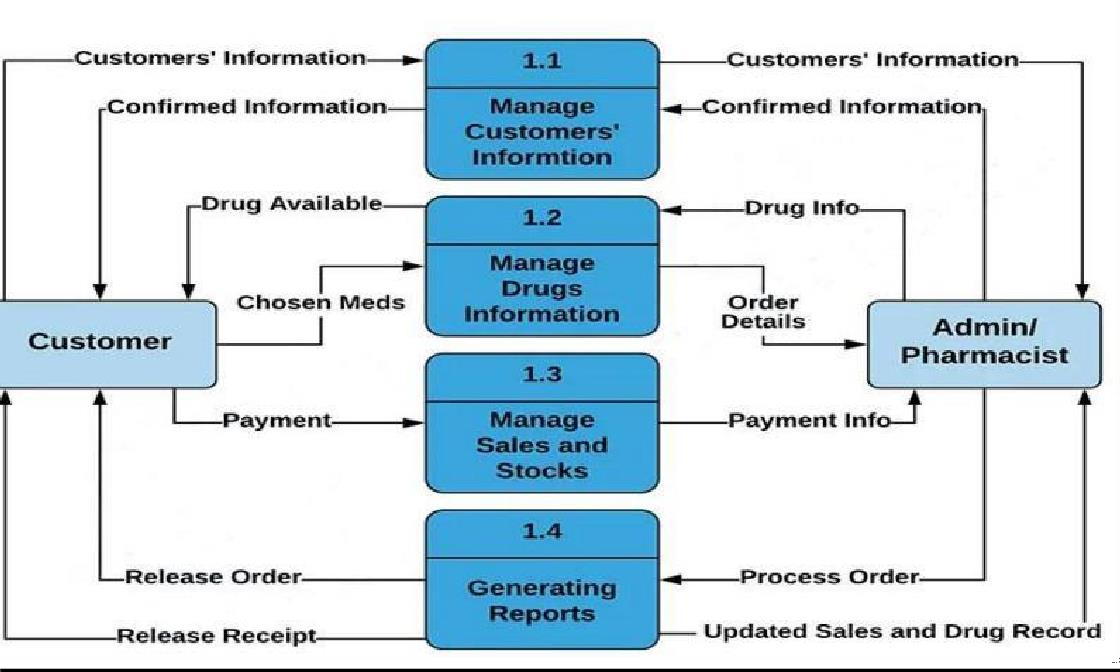
comprehensive test coverage.

* + - 1. Documentation and Maintenance: DFDs serve as documentation for the system, capturing important aspects of its architecture and functionality. They provide a reference for system maintenance, troubleshooting, and future development efforts, helping stakeholders understand the system's structure and behavior.

Overall, DFDs are valuable tools for analyzing, designing, documenting, and communicating system requirements and functionality. They help stakeholders understand how data flows through a system, identify areas for improvement, and ensure that the system meets the needs of its users.

1. External Entities: These are sources or destinations of data outside the system, such as users or other systems that interact with the system being modeled.
2. Processes: These represent operations performed on the data, such as calculations, transformations, or data storage.
3. Data Stores: These represent where data is held within the system. They can be databases, files, or any other repository where data is stored.
4. Data Flows: These are the pathways along which data flows within the system. They show the movement of data from one part of the system to another.

DFDs are typically used in the early stages of system design to visualize the flow of data and understand how the system will operate. They can range from simple diagrams that show high-level data flow to complex diagrams that detail every data transformation in a system. They are useful for understanding system requirements, identifying potential bottlenecks, and communicating system architecture to stakeholders.



## Fig 5.3.3: Data Flow Diagram

Data flow diagrams (DFDs) are important for several reasons:

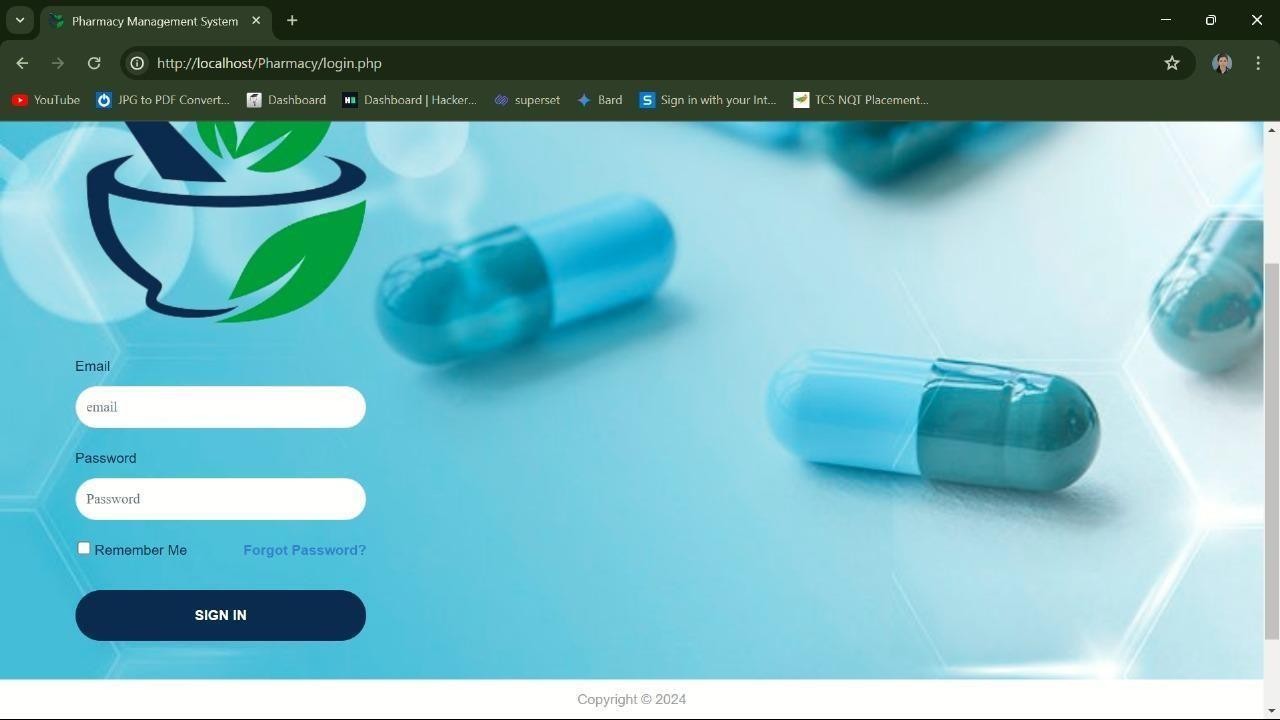
* 1. Visualization of System Functionality: DFDs provide a clear and visual representation of how data moves through a system, illustrating the system's functionality. This visualization helps stakeholders, including developers, designers, and users, to understand the system's operations and interactions more easily.
  2. System Understanding and Analysis: DFDs help in understanding the flow of data within a system. By breaking down complex systems into manageable components and illustrating their interactions, DFDs enable analysts to identify potential issues, bottlenecks, or inefficiencies in the system design.
  3. Requirements Specification: DFDs are often used in the early stages of system development to document and communicate system requirements. They help stakeholders to clarify their requirements and ensure that all parties have a common understanding of the system's functionality and data flows.
  4. Communication Tool: DFDs serve as a communication tool between different stakeholders involved in the system development process. They provide a common language for discussing system architecture, requirements, and design decisions, facilitating collaboration and alignment among team members.
  5. System Design and Planning: DFDs aid in system design by helping designers to conceptualize the structure and behavior of the system. They provide a blueprint for designing system components, specifying data transformations, and determining data storage requirements.
  6. Documentation: DFDs serve as documentation for the system, capturing important aspects of its architecture, functionality, and data flows. They provide a reference for future development, maintenance, and enhancement of the system.

Overall, data flow diagrams are important tools for system analysis, design, and communication, helping stakeholders to understand, analyze, and develop systems effectively.

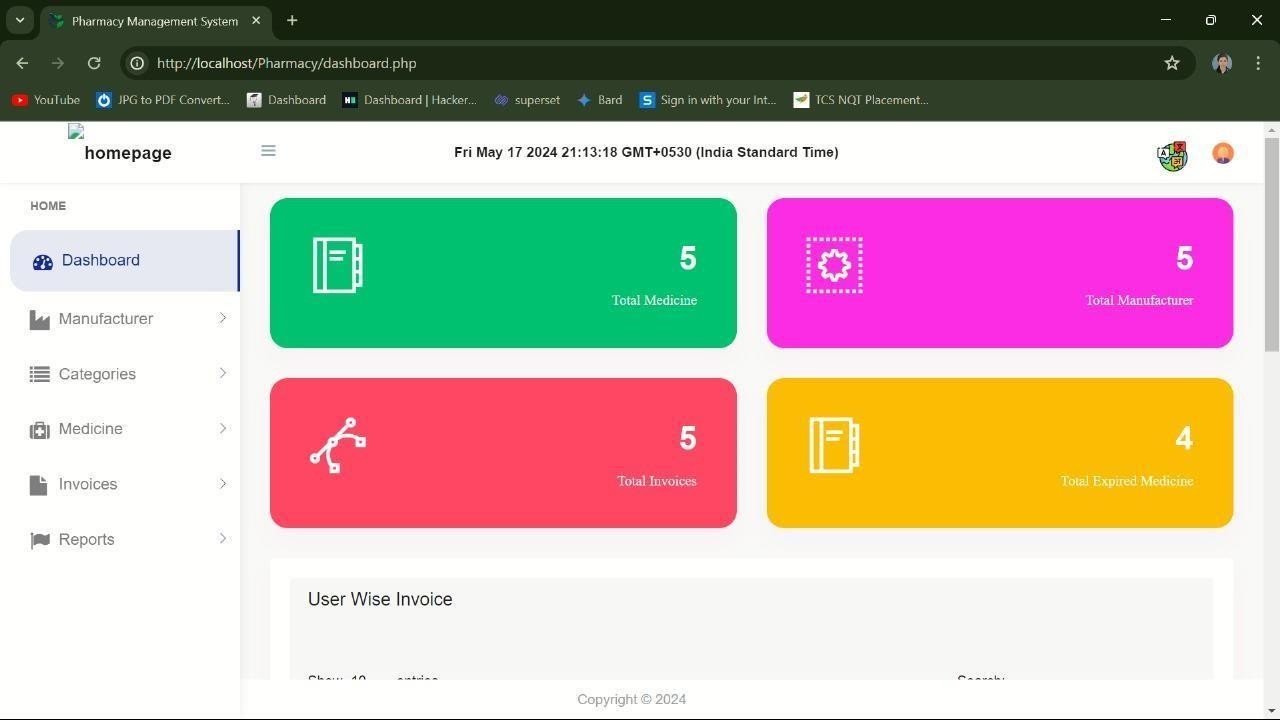
# CHAPTER 6

**DESIGN**

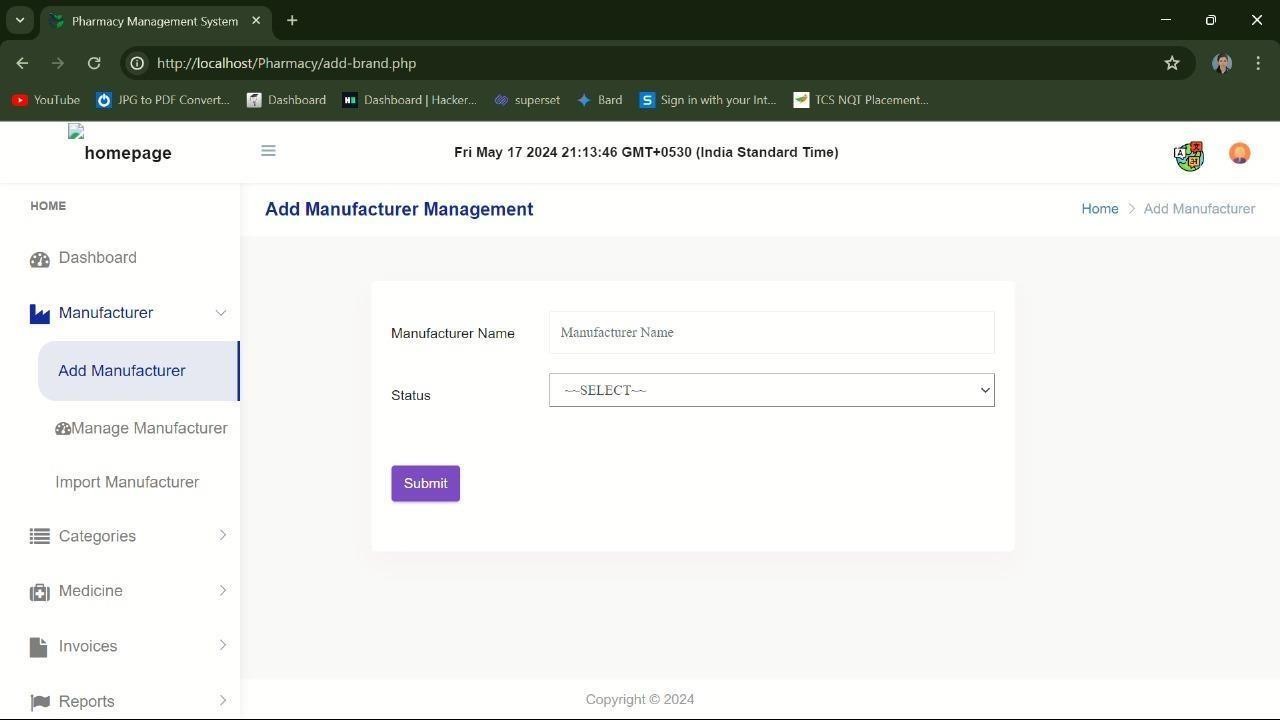
# Login Page-



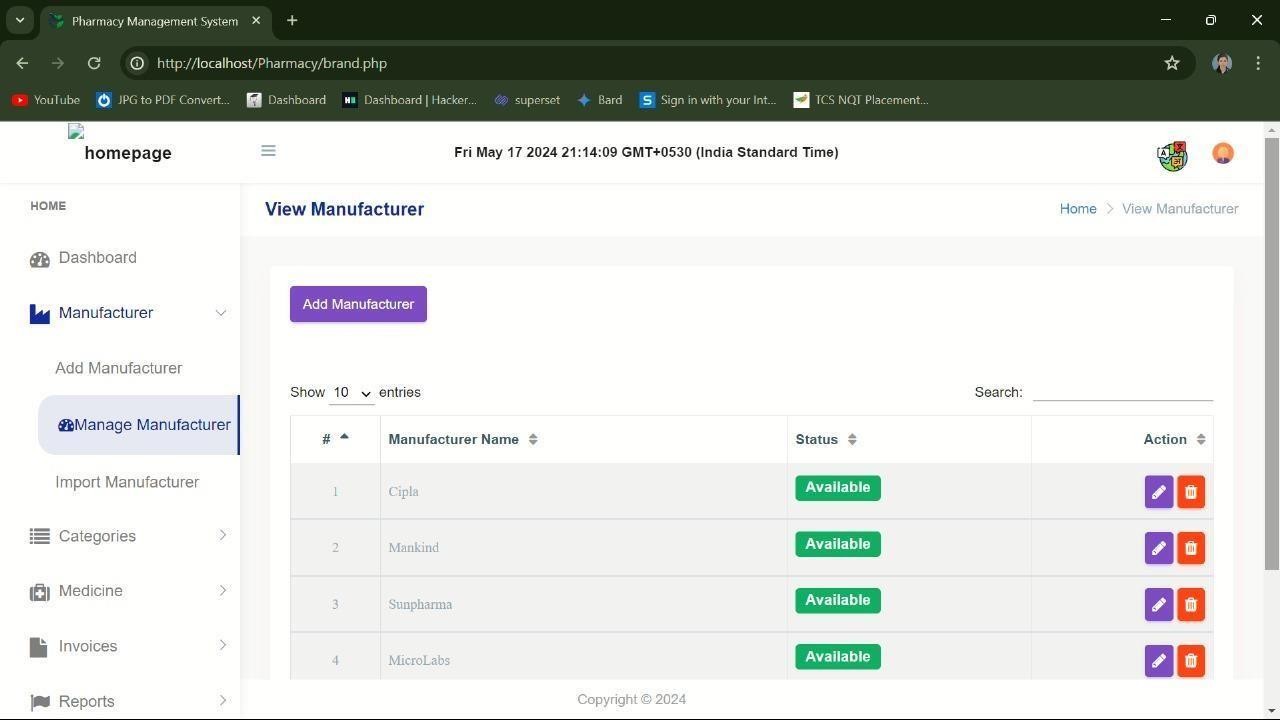
## Fig 6.1: Login Page



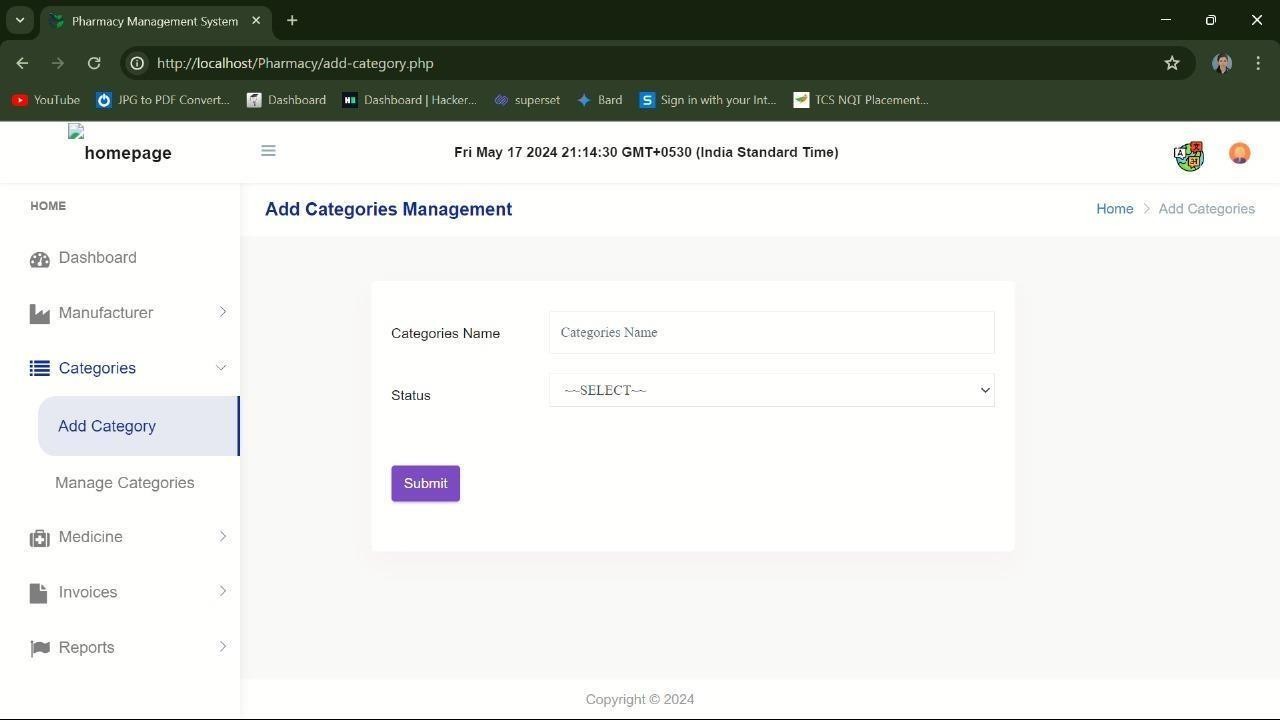
**Fig 6.2: Dashboard**



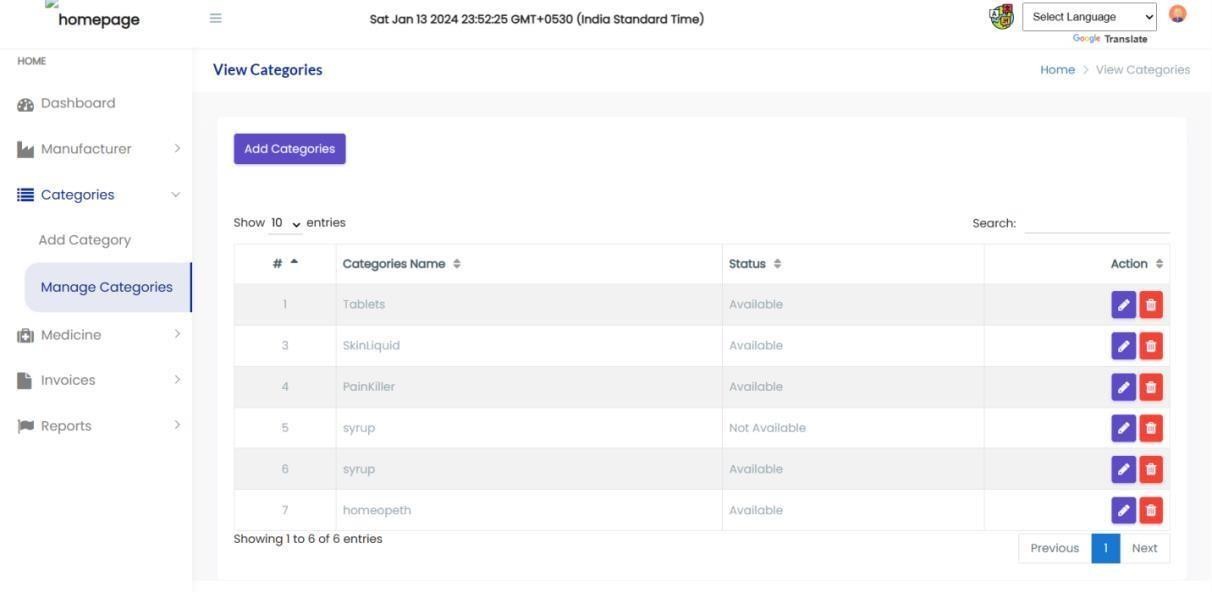
## Fig 6.3: Add Manufacturer



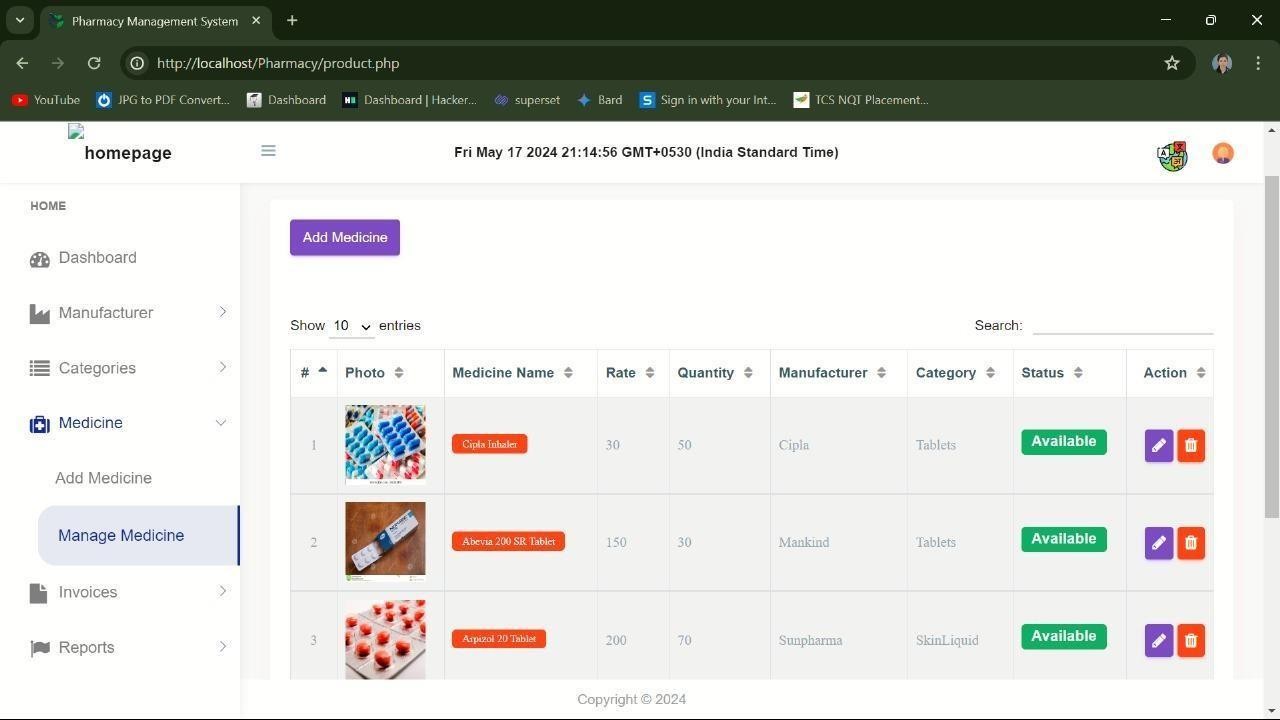
**Fig 6.4: Manage Manufacturer**



## Fig 6.5: Add Categories

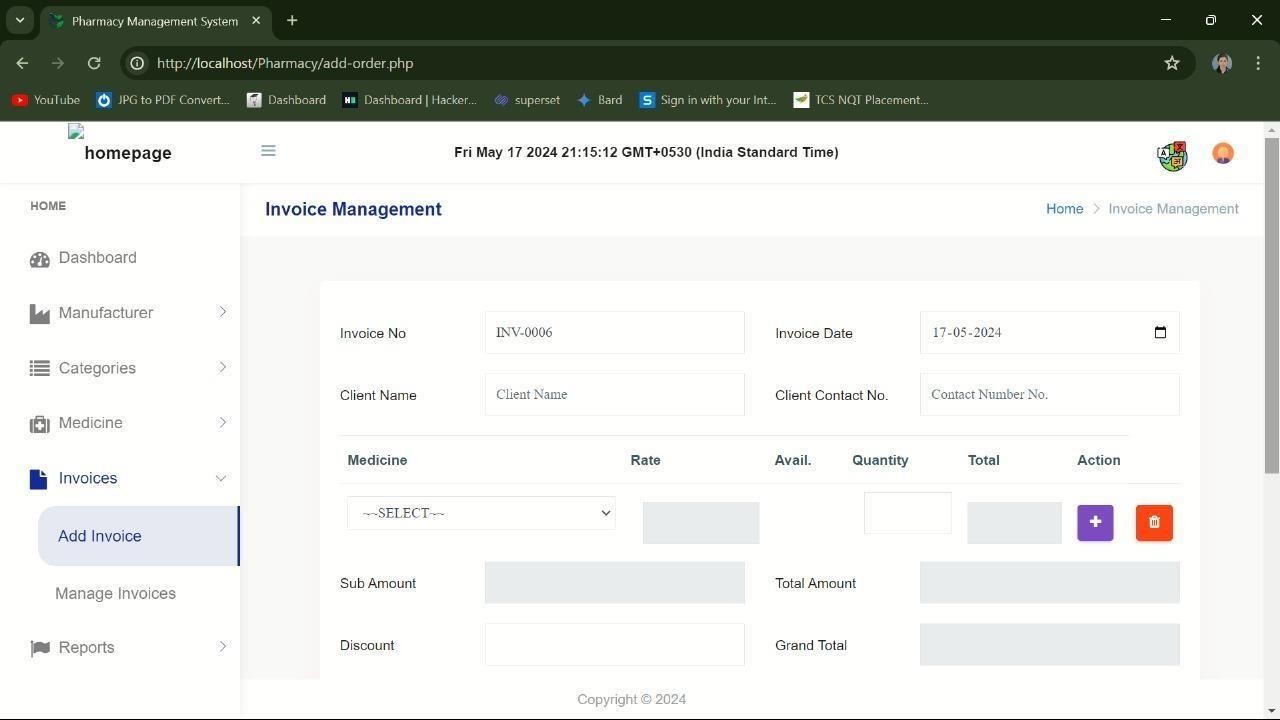


**Fig 6.6: Manage Categories**

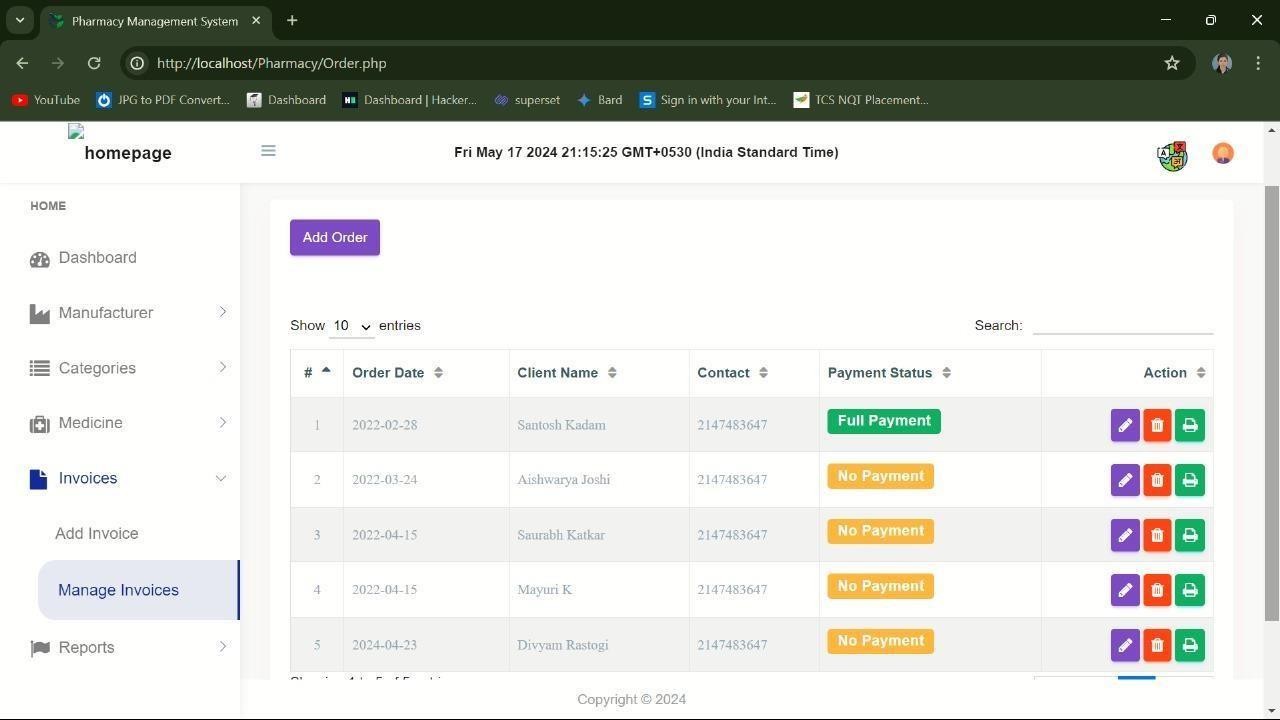


## Fig 6.7: Medicines

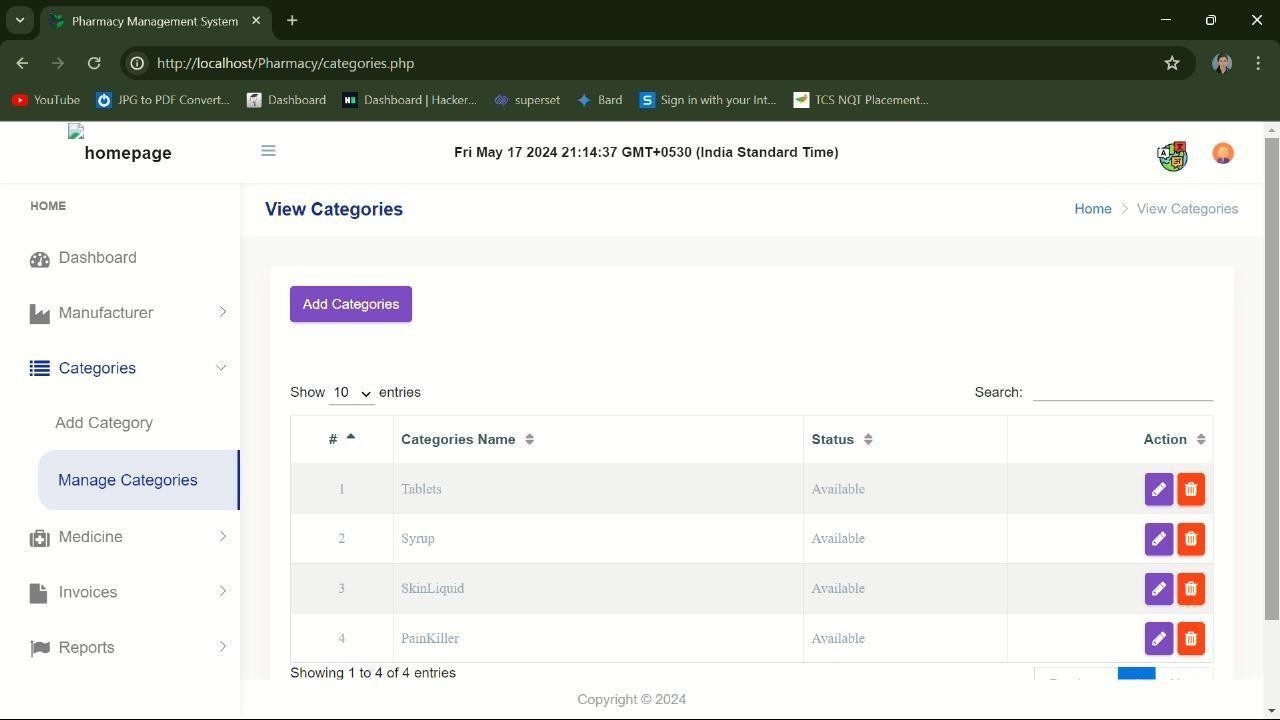
**6.8 INVOICES:**



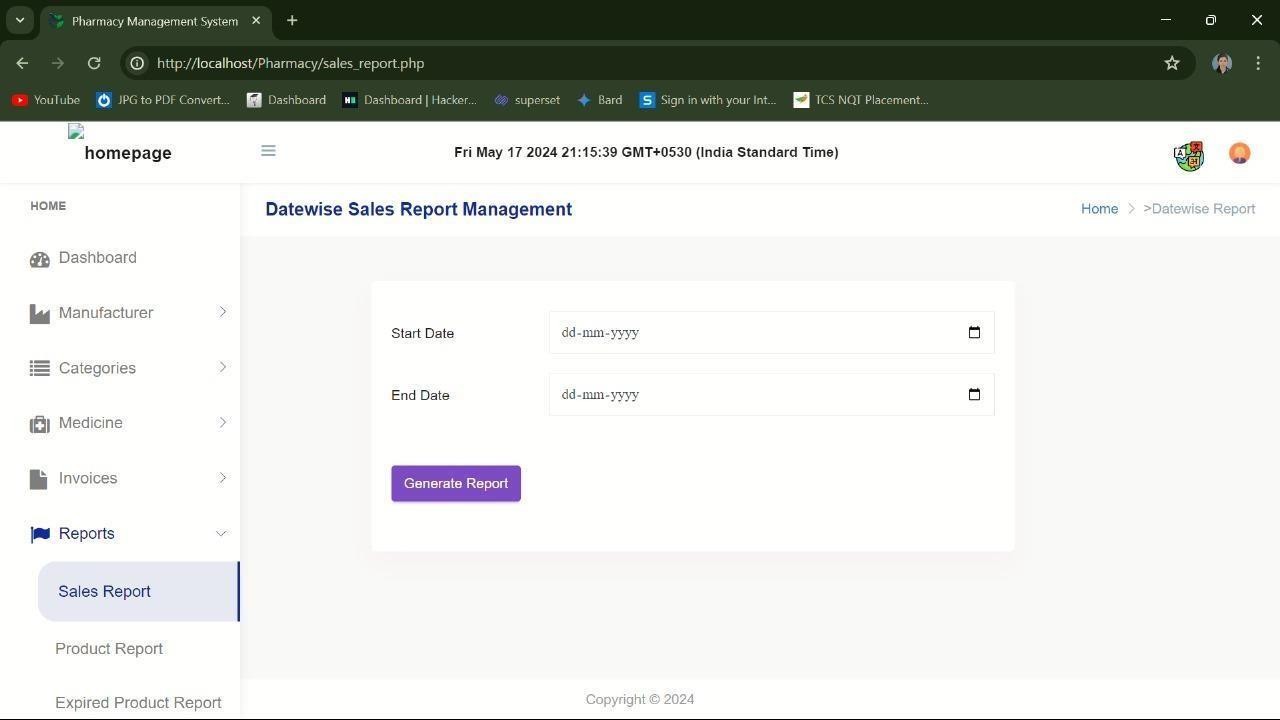
## Fig 6.8.1: Add Invoices



**Fig 6.8.2: Manage invoices**



## Fig 6.9: Manage Category



**Fig 6.10: Report**

# IMPLEMENTATION

## OVERVIEW OF PHP

PHP (Hypertext Pre-processor) is a widely used, server-side scripting language designed for web development. It is particularly well-suited for creating dynamic web pages and applications. PHP code is embedded within HTML, allowing developers to seamlessly integrate dynamic content into static web pages.

PHP (Hypertext Preprocessor) is a widely-used, open-source scripting language primarily designed for web development. It has several key features that make it popular among developers:

* + 1. Easy to Learn and Use: PHP syntax is similar to other C-based languages, making it relatively easy for developers to learn and use. Its simplicity and flexibility make it accessible to beginners while providing powerful features for advanced developers.
    2. Platform Independence: PHP runs on various operating systems, including Windows, Linux, macOS, and Unix. It is compatible with most web servers, such as Apache, Nginx, and Microsoft IIS, making it a versatile choice for web development.
    3. Integration with HTML: PHP code can be embedded directly within HTML, allowing developers to create dynamic web pages easily. This integration simplifies the process of building dynamic content and interacting with databases and other server-side components.
    4. Server-Side Scripting: PHP is a server-side scripting language, meaning that PHP code is executed on the server before the resulting HTML is sent to the client's browser. This allows for dynamic content generation, form processing, session management, and other server-side functionalities.
    5. Extensive Library Support: PHP has a vast ecosystem of libraries, frameworks, and extensions that extend its functionality and simplify common tasks. Popular PHP frameworks like Laravel, Symfony, and CodeIgniter provide tools for building robust web applications efficiently.
    6. Database Support: PHP offers built-in support for various databases, including MySQL,

PostgreSQL, SQLite, and others. It provides APIs for connecting to databases, executing queries, and managing database transactions, making it easy to build database-driven web applications.

* + 1. Community and Documentation: PHP has a large and active community of developers who contribute to its development, share knowledge, and provide support through forums, blogs, and online resources. The PHP manual and documentation provide comprehensive guidance on language features, functions, and best practices.
    2. Security Features: PHP includes built-in security features to help developers prevent common vulnerabilities, such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). Best practices for secure PHP programming include input validation, parameterized queries, and using secure coding practices.
    3. Performance Optimization: PHP offers various features and techniques for optimizing performance, such as opcode caching, code optimization, and asynchronous processing. Opcode caching tools like OPcache improve PHP performance by caching compiled bytecode and reducing execution time.
    4. Continuous Development: PHP is actively developed and maintained by the PHP community, with regular releases introducing new features, improvements, and bug fixes. The PHP development team follows a predictable release cycle, ensuring stability and compatibility for developers.

## Key Features

* + - * **Open Source**: PHP is an open-source language, making it freely available for developers to use, modify, and distribute.
      * **Server-Side Scripting**: PHP is primarily used on the server side of web development. When a user requests a PHP page, the server processes the PHP code and sends the

output (usually HTML) to the client's browser.

* + - * **Cross-Platform Compatibility** : PHP is compatible with various operating systems, including Windows, Linux, and macOS. This ensures flexibility and ease of

deployment.

* + - * **Versatility:** PHP supports a wide range of web development tasks, from simple scripting to more complex object-oriented programming.

Reasons for Choosing PHP in the Project

* + - * PHP is specifically designed for web development, making it a natural choice for building web-based applications.
      * Given its strong support for database connectivity, PHP is well-suited for projects requiring interaction with databases, such as a pharmacy management system.
      * The extensive libraries and frameworks available in PHP provide developers with tools to expedite development and ensure the reliability of the project.

Choosing PHP for web development can depend on various factors, including project requirements, developer expertise, and specific use cases. Here are some reasons why PHP might be chosen for web development:

1. Ease of Learning and Use: PHP has a relatively simple and intuitive syntax, making it easy for beginners to learn and use. Its similarity to other C-based languages also makes it accessible to developers with prior programming experience.
2. Wide Adoption and Community Support: PHP is one of the most widely-used server-side scripting languages, with a large and active community of developers. This means there are abundant resources, tutorials, libraries, and frameworks available to support PHP development.
3. Platform Independence: PHP runs on various operating systems, including Windows, Linux, macOS, and Unix, making it a versatile choice for web development. It is compatible with most web servers, such as Apache, Nginx, and Microsoft IIS.
4. Integration with HTML and Web Technologies: PHP can be seamlessly integrated with HTML, allowing developers to embed PHP code directly within HTML pages. This facilitates the creation of dynamic web pages and the interaction with databases, forms, and other server-side components.
5. Database Support: PHP has built-in support for various databases, including MySQL, PostgreSQL, SQLite, and others. This makes it easy to build database-driven web applications and perform tasks such as database connectivity, query execution, and data manipulation.
6. Performance: PHP offers good performance for web applications, especially when combined with caching mechanisms and performance optimization techniques. Opcache, for example, is a built-in opcode caching tool that improves PHP performance by caching compiled bytecode.
7. Rich Ecosystem of Frameworks and Tools: PHP has a rich ecosystem of frameworks, libraries, and tools that streamline web development and provide solutions for common tasks. Frameworks like Laravel, Symfony, and CodeIgniter offer features such as routing, MVC architecture, ORM, and authentication out of the box.
8. Cost-effectiveness: PHP is open-source and free to use, making it a cost-effective choice for web development projects. There are no licensing fees associated with PHP, and hosting costs for PHP-based applications are typically lower compared to proprietary platforms.
9. Scalability: PHP can scale to accommodate growing web applications and user traffic. With proper architecture, caching strategies, and optimization techniques, PHP applications can handle high loads and maintain performance over time.
10. Compatibility and Legacy Support: PHP maintains backward compatibility with older versions, ensuring that legacy applications continue to function correctly. This makes PHP a suitable choice

In summary, PHP is a versatile and powerful scripting language that serves as an excellent choice for web development projects. Its open-source nature, strong community support, and integration capabilities make it well-suited for creating the dynamic and database-driven components essential to a pharmacy management system.

## DATABASE SETUP AND CONNECTIVITY

* **Database Selection**

In the pharmacy management system, a relational database management

system (RDBMS) is commonly chosen for its ability to efficiently organize and manage data. MySQL is a popular choice due to its open-source nature, reliability, and compatibility with PHP.

## Database Creation

* **Create Database**:

Begin by creating a dedicated database for the pharmacy management

system. This can be done using SQL commands or a database administration tool. Create database pharmacy system;

## Create Tables

Design tables based on the previously defined entity-relationship model.

Each table corresponds to a specific entity (e.g., Patient, Medication, Prescription) with its attributes.

CREATE TABLE Patient ( PatientID INT PRIMARY KEY,

FirstName VARCHAR (50),

LastName VARCHAR (50), DateOfBirth DATE, ContactNumber VARCHAR (15),

Address VARCHAR (255)

);

Repeat this process for each entity in the system.

In summary, PHP is a versatile and powerful scripting language that serves as an excellent choice for web development projects. Its open-source nature, strong community support, and integration capabilities make it well-suited for creating the dynamic and database-driven components essential to a pharmacy management system.

## DATABASE SETUP AND CONNECTIVITY

* **Database Selection**

In the pharmacy management system, a relational database management

system (RDBMS) is commonly chosen for its ability to efficiently organize and manage data. MySQL is a popular choice due to its open-source nature, reliability, and compatibility with PHP.

## Database Creation

* **Create Database**:

Begin by creating a dedicated database for the pharmacy management

system. This can be done using SQL commands or a database administration tool. Create database pharmacy system;

## Create Tables

Design tables based on the previously defined entity-relationship model.

Each table corresponds to a specific entity (e.g., Patient, Medication, Prescription) with its attributes.

CREATE TABLE Patient ( PatientID INT PRIMARY KEY,

FirstName VARCHAR (50),

LastName VARCHAR (50), DateOfBirth DATE, ContactNumber VARCHAR (15),

Address VARCHAR (255)

);

Repeat this process for each entity in the system.

## Database Connectivity in PHP:

PHP provides functions and extensions to interact with databases. Below is

an example of how to establish a connection to a MySQL database within a PHP script:

<?php

$servername = "localhost";

$username = "your\_username";

$password = "your\_password";

$database = "pharmacy system";

// create connection

$conn = new mysqli ($servername, $username, $password, $database);

// Check connection

if ($conn->connect\_error) {

die("Connection failed: " . $conn->connect\_error);

}

echo "Connected successfully";

?>

Replace "your\_username" and "your\_password" with the appropriate credentials for your MySQL server.

## Handling Database Operations

Once connected, PHP can execute SQL queries to perform operations like

inserting, updating, retrieving, and deleting data.

## Example of querying the Patient table:

$sql = "SELECT \* FROM Patient";

$result = $conn->query($sql);

if ($result->num\_rows > 0) { while($row = $result->fetch\_assoc()) {

echo "PatientID: " . $row["PatientID"]. " - Name: " . $row["FirstName"]. " " .

$row["LastName"]. "<br>";

}

} else {

echo "0 results";

}

$conn->close();

* **Security Measures:** Use prepared statements to prevent SQL injection attacks.

$stmt = $conn->prepare("INSERT INTO Patient (FirstName, LastName, DateOfBirth) VALUES (?, ?, ?)");

$stmt->bind\_param("sss", $firstName, $lastName, $dob);

Limited Privileges: Create a database user with the minimum required privileges to enhance security.

## Connection Pooling and Optimization:

Consider implementing connection pooling to efficiently manage database

connections and optimize performance, especially in scenarios with a high number of concurrent users.

Establishing a robust database setup and connectivity is crucial for the pharmacy management system. By selecting an appropriate RDBMS, creating a well-structured database, and ensuring secure connectivity, the system can efficiently store, retrieve, and manipulate data to meet its functional requirements.

## SECURITY MEASURES IMPLEMENTED

Security is paramount in a pharmacy management system to safeguard sensitive patient information, medication data, and overall system integrity.

The implemented security measures work in tandem to fortify the pharmacy management system against potential threats. By encrypting sensitive data, enforcing robust authentication mechanisms, securing the database, and regularly auditing security protocols, the system ensures the confidentiality, integrity, and availability of information, maintaining trust and compliance with industry regulations .Below are the key security measures implemented in the system:

* **Data Encryption:** Sensitive data, such as patient information and prescription details, is encrypted to protect it from unauthorized access. The use of secure protocols like

HTTPS ensures the encrypted transmission of data between the server and the client.

* **User Authentication:** Password Hashing: User passwords are not stored in plain text. Instead, they are securely hashed using strong cryptographic hashing algorithms (e.g.,

bcrypt). This ensures that even if the database is compromised, passwords remain secure.

* **Multi-Factor Authentication (MFA):** Implement MFA for added security, requiring users to provide multiple forms of identification before accessing the system.

## Database Security

* + - * **Limited Access** : Database users are granted the minimum necessary privileges to perform their functions. For example, the application's database user may only have

SELECT, INSERT, UPDATE, and DELETE privileges on the relevant tables.

* + - * **Parameterized Queries:**SQL injection attacks are prevented by using parameterized queries (prepared statements) in database interactions. This ensures that user input is

treated as data, not executable code.

* + - * **Session Management:** Session Timeout: Implement session timeout mechanisms to automatically log users out after a period of inactivity, reducing the risk of unauthorized

access.

* + - * **Session Regeneration**: Regenerate session IDs after successful login to mitigate session fixation attacks.
      * **Error Handling:** Custom Error Messages: Display generic error messages to users while logging detailed errors for system administrators. This prevents attackers from

gaining insights into system vulnerabilities.

* + - * **Logging:**Implement comprehensive logging mechanisms to record security-related events, aiding in the identification of potential security breaches.
      * **Secure File Uploads:** If the system involves file uploads (e.g., prescription images), ensure that proper validation is in place to verify file types and implement measures to

prevent malicious file uploads.

* + - * **Regular Security Audits**: Perform regular security audits and vulnerability assessments to identify and address potential security weaknesses.

# CHAPTER 9

**TESTING**

## TESTING METHODOLOGIES

* + 1. **Unit Testing**

Unit testing is a fundamental testing method where individual units or components of the software are tested in isolation. A "unit" may refer to a function, method, or a small piece of code. The primary goal is to verify that each unit of the software performs as expected. This approach aids in the early detection and resolution of bugs in specific functionalities.

Consider a unit test for a Patient class that checks if the getFullName () method correctly concatenates the first and last names.

A unit typically refers to the smallest testable part of an application, such as a function, method, or class. Here's an overview of unit testing:

1. Isolation: In unit testing, each unit is tested in isolation from the rest of the application. Dependencies, such as external services, databases, or other modules, are typically replaced with mock objects or stubs to focus solely on testing the unit itself.
2. Automated Testing: Unit tests are automated, meaning they can be executed automatically without manual intervention. Developers write test cases using testing frameworks or libraries such as JUnit (for Java), NUnit (for .NET), pytest (for Python), or PHPUnit (for PHP).
3. White Box Testing: Unit testing is a form of white box testing, where testers have access to the internal structure and code of the unit being tested. This allows for thorough testing of different code paths, boundary conditions, and error scenarios.
4. Fast Feedback: Unit tests provide fast feedback to developers, allowing them to detect

and fix defects early in the development process. Since unit tests are small, focused, and execute quickly, developers can run them frequently during development to ensure code quality and correctness.

1. Regression Testing: Unit tests serve as a form of regression testing, ensuring that changes made to the codebase do not introduce new bugs or regressions. By maintaining a suite of unit tests, developers can quickly identify and fix issues that arise during development or code refactoring.
2. Code Quality and Maintainability: Unit testing promotes code quality and maintainability by encouraging modular, well-structured code. Writing testable code often leads to better software design, separation of concerns, and adherence to coding best practices.
3. Documentation: Unit tests serve as documentation for the behavior and functionality of individual units in the codebase. They provide examples of how units should be used, what inputs they expect, and what outputs they produce, helping developers understand and maintain the code.
4. Continuous Integration and Deployment (CI/CD): Unit tests are a crucial component of CI/CD pipelines, where they are integrated into automated build and deployment processes. By running unit tests automatically with each code change, teams can ensure code quality and detect integration issues early.

Overall, unit testing is a fundamental practice in software development that helps ensure code quality, reliability, and maintainability by testing individual units of code in isolation. It complements other testing techniques, such as integration testing and end-to- end testing, to provide comprehensive test coverage for software applications.

## Integration Testing

Integration testing involves testing the interactions and interfaces between integrated components or modules. This ensures that different parts of the system work seamlessly when combined. The focus is on verifying data flow, communication, and the overall integration of various components. Integration testing helps identify issues that may arise when different modules interact with each other.

An integration test could involve creating a Prescription object and verifying that it interactscorrectly with the Patient, Doctor, and Medication entities.

Integration testing is used in software development to verify that individual units or components of a system work together as intended when integrated. Here are some reasons why integration testing is important:

1. Verification of Interactions: Integration testing validates the interactions between different components of a system. It ensures that these components communicate and collaborate correctly, exchanging data and invoking each other's functionalities as expected.
2. Detection of Interface Issues: Integration testing helps identify interface issues between modules or subsystems, such as mismatched data formats, incompatible protocols, or missing communication endpoints. By testing integration points, developers can uncover and address these issues early in the development process.
3. Validation of System Architecture: Integration testing validates the overall system architecture by testing the integration of components at different levels (e.g., unit-to-unit, subsystem-to-subsystem, system-to-system). It ensures that the system's architecture and design are implemented correctly and meet the specified requirements.
4. Detection of Functional Defects: Integration testing uncovers functional defects that may arise when integrating individual units or components. These defects may include incorrect data transformations, logic errors, or unexpected behavior resulting from interactions between components.
5. Testing End-to-End Scenarios: Integration testing allows for the testing of end-to-end scenarios that span multiple components or subsystems. This enables developers to validate complex workflows, user journeys, or system behaviors that involve multiple interacting components.
6. Validation of Dependencies: Integration testing verifies that dependent components are correctly integrated and function together without issues. It ensures that changes to one component do not adversely affect the behavior or functionality of other components.
7. Identification of Performance Issues: Integration testing can uncover performance issues related to component interactions, such as latency, bottlenecks, or resource contention. By simulating realistic usage scenarios, developers can assess the system's performance under load and optimize as necessary.
8. Risk Mitigation: Integration testing helps mitigate the risk of system failures or errors by identifying integration-related issues early in the development lifecycle. By detecting and addressing these issues proactively, developers can improve the reliability, stability, and quality of the system.
9. Compliance with Requirements: Integration testing ensures that the integrated system meets the specified requirements and behaves as expected in real-world scenarios. It provides confidence to stakeholders that the system functions correctly and delivers the intended value.

Overall, integration testing is a critical part of the software testing process, ensuring that individual components integrate seamlessly to form a cohesive and functional system. By validating interactions, detecting defects, and mitigating risks, integration testing contributes to the overall quality and reliability of software applications.

## System Testing

System testing evaluates the system as a whole, focusing on end-to-end scenarios. It examines the complete system to ensure that all components work together cohesively. This testing phase assesses system performance, security, and user interactions. System testing is crucial for validating that the entire system meets the specified requirements.

A system test for medication inventory might simulate adding medications. System testing is crucial in the software development lifecycle for several reasons:

1. Validation of System Requirements: System testing ensures that the software system meets the specified requirements and functions as intended. It verifies that all features and functionalities work correctly and that the system behaves as expected under different conditions.
2. Detection of Defects and Issues: System testing helps identify defects, bugs, and inconsistencies in the software system. By testing the system as a whole, testers can uncover integration issues, functional defects, performance bottlenecks, and other problems that may not be apparent during unit or integration testing.
3. Evaluation of System Performance: System testing evaluates the performance of the software system under normal and peak loads. It assesses factors such as response time, throughput, scalability, and resource utilization to ensure that the system performs satisfactorily and meets

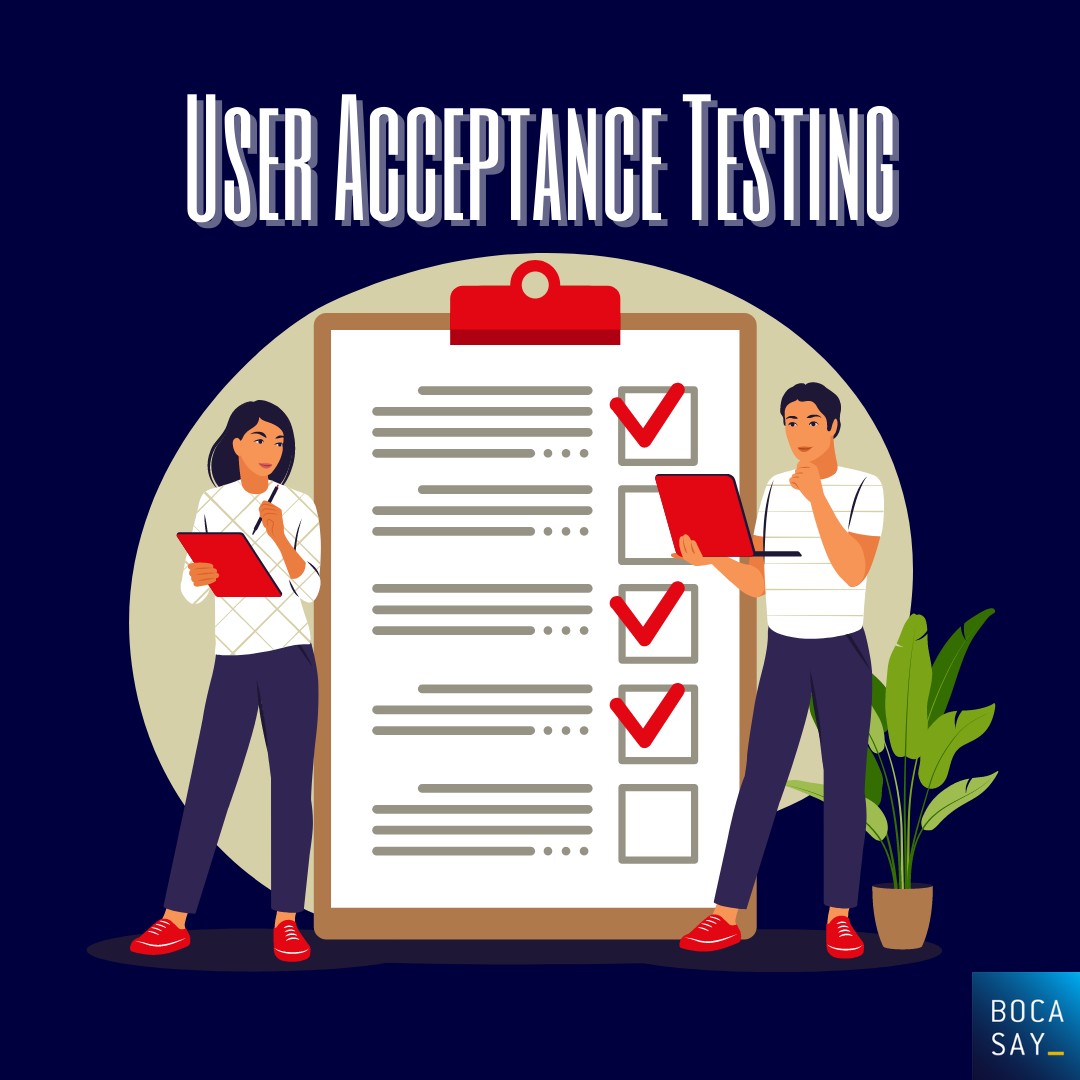
performance requirements.

1. Assessment of Usability and User Experience: System testing evaluates the usability and user experience of the software system from an end-user perspective. It assesses factors such as navigation, interface design, accessibility, and error handling to ensure that the system is intuitive, user-friendly, and meets user expectations.
2. Validation of System Security: System testing verifies the security of the software system by assessing its resistance to security threats, vulnerabilities, and attacks. It tests authentication mechanisms, authorization controls, data encryption, and other security features to ensure that sensitive information is protected and regulatory requirements are met.
3. Verification of Compliance and Standards: System testing ensures that the software system complies with relevant standards, regulations, and industry best practices. It verifies adherence to quality standards, legal requirements, industry guidelines, and organizational policies to mitigate risks and ensure regulatory compliance.
4. Risk Mitigation and Quality Assurance: System testing helps mitigate risks associated with software deployment by identifying and addressing potential issues before release. It provides assurance to stakeholders that the software system is of high quality, reliable, and ready for production use.
5. Customer Satisfaction and Business Success: System testing contributes to customer satisfaction and business success by ensuring that the software system meets user needs, performs reliably, and delivers value. A well-tested system is more likely to meet user expectations, reduce support costs, and enhance the organization's reputation and competitiveness.

Overall, system testing plays a vital role in ensuring the quality, reliability, and performance of software systems. By thoroughly evaluating the system's functionality, performance, security, and compliance, system testing helps deliver software that meets user requirements, mitigates risks, and contributes to the success of the organization.

## Acceptance Testing

Acceptance testing is conducted to ensure that the system meets the predefined acceptance criteria and is acceptable to end-users or stakeholders. This phase often involves end-users validating system functionality against business requirements. It addresses user.



An acceptance test could involve validating the user login process to ensure that it meets userexpectations and security standards.

Acceptance testing is important for several reasons:

1. Validation of Business Requirements: Acceptance testing ensures that the software system meets the business requirements and objectives defined by stakeholders. By testing the system against real-world scenarios and user expectations, acceptance testing validates that the software delivers the intended value to the organization.
2. Confirmation of User Needs: Acceptance testing validates that the software system meets the needs and expectations of end-users. By involving actual users or representatives from the target audience in the testing process, acceptance testing ensures that the software is user- friendly, intuitive, and fit for purpose.
3. Assurance of Quality and Reliability: Acceptance testing provides assurance that the software system is of high quality and reliability. By verifying that the system functions correctly, performs as expected, and meets specified criteria, acceptance testing helps mitigate risks and instill confidence in stakeholders.
4. Identification of Defects and Issues: Acceptance testing helps identify defects, bugs, and usability issues that may not have been detected during earlier stages of testing. By subjecting the software to real-world usage scenarios, acceptance testing uncovers issues that may affect user experience, functionality, or performance.
5. Risk Mitigation: Acceptance testing helps mitigate risks associated with software deployment by identifying and addressing potential issues before release. By validating the software against user requirements and expectations, acceptance testing reduces the likelihood of post-release failures, customer dissatisfaction, and costly rework.
6. User Involvement and Engagement: Acceptance testing fosters user involvement and engagement in the software development process. By actively involving users in testing activities, acceptance testing encourages collaboration, feedback, and buy-in, leading to greater user satisfaction and adoption of the software.
7. Regulatory Compliance and Legal Requirements: Acceptance testing ensures that the software system complies with relevant regulations, standards, and legal requirements. By validating adherence to industry regulations, data privacy laws, accessibility guidelines, and other requirements, acceptance testing helps mitigate legal and regulatory risks.
8. Customer Satisfaction and Business Success: Acceptance testing contributes to customer satisfaction and business success by ensuring that the software system meets user needs,

expectations, and quality standards. A well-tested system is more likely to meet user requirements, reduce support costs, and enhance the organization's reputation and competitiveness.

Overall, acceptance testing is a critical component of the software testing process, ensuring that the software system meets user needs, business requirements, and quality standards. By validating the software against real-world usage scenarios and user expectations, acceptance testing helps deliver software that adds value, mitigates risks, and contributes to the success of the organization.

## Security Testing

Security testing focuses on identifying vulnerabilities and weaknesses in the system's security measures. It encompasses various tests, such as checking for SQL injection, cross-site scripting, and penetration testing. Security testing aims to ensure that the system is robust and resilient against potential security threats.

A security test for SQL injection might involve attempting to inject malicious SQL code into user inputs to ensure that the system is protected.

## Regression Testing:

Regression testing involves re-running previously executed tests after code changes to ensure that new changes do not introduce unintended side effects or regressions. It helps maintain the stability and reliability of the system over time, especially as new features or modifications are implemented.

A regression test could involve updating the price of a medication and ensuring that the updatedoes not affect other functionalities in the system.

By combining these testing methodologies, the pharmacy management system undergoes a comprehensive evaluation, addressing various aspects of its functionality, integration, security, and user satisfaction. This multifaceted testing approach contributes to the overall quality, reliability, and success of the system.

Regression testing is important for several reasons:

* + - 1. Verification of Existing Functionality: Regression testing ensures that existing functionality remains intact and continues to work correctly after changes are made to the software. It verifies that new features, bug fixes, or enhancements do not inadvertently break or disrupt the functionality of previously implemented features.
      2. Detection of Regression Defects: Regression testing helps identify regression defects, which are defects that reappear or are introduced as a result of code changes. These defects may arise due to unintended side effects, dependencies between modules, or

interactions with other parts of the system.

* + - 1. Prevention of Software Degradation: Regression testing prevents software degradation by catching defects early in the development process. By running regression tests regularly, developers can detect and fix issues before they impact users or become more difficult and costly to resolve.
      2. Maintaining Code Quality and Stability: Regression testing contributes to maintaining code quality and stability by ensuring that changes to the software do not introduce new defects or degrade system performance. It helps uphold quality standards, reliability, and user satisfaction by verifying that the software continues to meet expectations over time.
      3. Support for Continuous Integration and Deployment (CI/CD): Regression testing is an essential component of CI/CD pipelines, where it is integrated into automated build and deployment processes. By running regression tests automatically with each code change, teams can ensure that software updates do not introduce regressions and are safe to deploy to production environments.
      4. Validation of Fixes and Patches: Regression testing validates fixes and patches applied to the software to address known issues or vulnerabilities. It ensures that fixes do not inadvertently introduce new defects or regressions elsewhere in the system, providing confidence that the software remains stable and secure.
      5. Risk Mitigation: Regression testing helps mitigate the risk of software failures, downtime, or user dissatisfaction caused by regression defects. By systematically testing the software against a comprehensive set of test cases, regression testing reduces the likelihood of unexpected issues arising in production environments.
      6. Compliance and Audit Requirements: Regression testing may be required to comply with industry regulations, quality standards, or audit requirements. It provides evidence of ongoing quality assurance efforts and helps demonstrate that the software meets specified criteria for reliability, stability, and functionality.

Overall, regression testing is a critical practice in software development and quality assurance, ensuring that software changes do not introduce new defects or degrade existing

functionality. By systematically testing the software over time, regression testing helps maintain code quality, stability, and user satisfaction, contributing to the success of software projects and organizations.

## TEST CASES AND RESULTS

Test cases are detailed instructions or scenarios designed to verify specific aspects of software functionality. They outline the steps to be followed, the expected behavior or outcomes, and the criteria for determining whether the software passes or fails the test. Test cases are typically created based on requirements, user stories, design specifications, and other project documentation.

Test cases typically consist of the following elements:

1. Test Case ID: A unique identifier for the test case, which helps track and manage test cases within a testing framework or management tool.
2. Test Description: A brief description of the test case, outlining its purpose, objectives, and scope.
3. Preconditions: Any conditions or prerequisites that must be met before the test case can be executed. Preconditions ensure that the system is in a suitable state for testing.
4. Test Steps: The sequence of steps to be followed to execute the test case. Each step should be clear, concise, and specific, detailing the actions to be taken and the inputs to be provided.
5. Expected Results: The expected outcomes or behavior that should be observed when the test case is executed successfully. This includes expected outputs, responses, error messages, or changes in system state.
6. Actual Results: The actual outcomes observed when the test case is executed. Testers record the results of each step and compare them against the expected results to determine whether the test case passes or fails.
7. Pass/Fail Criteria: The criteria for determining whether the test case passes or fails based on the comparison between expected and actual results. This may include tolerances for acceptable deviations or discrepancies.
8. Test Data: Any data or inputs required for executing the test case, such as sample data, parameters, or configuration settings.
9. Test Environment: The environment in which the test case will be executed, including hardware, software, network configurations, and any dependencies or constraints.

Test results refer to the outcome of executing test cases, including whether the software passed or failed each test case, any deviations or discrepancies observed, and any issues or defects identified during testing. Test results are typically recorded in test reports or test management tools and may include additional information such as test execution status, test coverage metrics, and defect tracking information. Analyzing test results helps stakeholders assess the quality, stability, and readiness of the software for release**.**

The functionality of test cases revolves around verifying that the software meets specified requirements, behaves as expected, and performs its intended functions correctly. Here's how test cases fulfill their functionality:

1. Verification of Requirements: Test cases are designed to validate that the software meets the requirements outlined in the project documentation, such as functional requirements, user stories, and acceptance criteria. Each test case is mapped to specific requirements to ensure comprehensive coverage and validation of the software's functionality.
2. Validation of Features and Functions: Test cases verify that individual features, functions, and components of the software behave as expected. They test various scenarios, inputs, and conditions to ensure that the software performs the intended operations correctly and produces the expected outputs.
3. Identification of Defects and Issues: Test cases help identify defects, bugs, and inconsistencies in the software by systematically testing different aspects of its functionality. By executing test cases and comparing actual results against expected results, testers can detect deviations, errors, or unexpected behavior that may indicate underlying issues or defects.
4. Regression Testing: Test cases support regression testing by verifying that changes to the software do not introduce new defects or regressions. Regression test cases are executed to ensure that existing functionality remains intact after code changes, updates, or enhancements, preventing degradation of software quality over time.
5. Boundary and Error Conditions Testing: Test cases include scenarios that test boundary conditions, edge cases, and error conditions to validate how the software handles exceptional situations. By testing inputs, outputs, and system responses under different conditions, test cases help ensure robustness, reliability, and error tolerance.
6. User Acceptance Testing (UAT): Test cases for user acceptance testing (UAT) validate that the software meets user expectations, preferences, and usability requirements. UAT test cases are designed to simulate real-world usage scenarios and user interactions to ensure that the software is user-friendly, intuitive, and fit for purpose.
7. Performance and Load Testing: Test cases for performance and load testing assess the software's performance characteristics, such as response time, throughput, scalability, and resource utilization. These test cases simulate high loads, stress conditions, and peak usage scenarios to evaluate system performance and identify performance bottlenecks.
8. Compliance Testing: Test cases for compliance testing validate that the software complies with relevant regulations, standards, and industry guidelines. Compliance test cases verify adherence to

legal requirements, data privacy laws, accessibility standards, and security best practices to mitigate risks and ensure regulatory compliance.

Overall, test cases play a crucial role in ensuring the quality, reliability, and functionality of software systems by systematically verifying that the software meets requirements, behaves as expected, and performs its intended functions correctly. By designing and executing comprehensive test cases, testers can identify defects early, validate software functionality, and contribute to the delivery of high-quality software products.

Test cases are essential components of software testing, utilized across various stages of the software development lifecycle to ensure the quality, reliability, and functionality of software systems.

In unit testing, developers create and execute test cases to verify the correctness of individual units or components of the software, such as functions or classes. Integration testing involves testing the interactions between integrated units or subsystems using test cases to ensure seamless collaboration. System testing employs test cases to validate the entire software system against specified requirements, assessing end-to-end functionality and performance. Acceptance testing utilizes test cases to validate that the software meets user requirements and expectations, ensuring its readiness for release.

Regression testing verifies that changes to the software do not introduce new defects or regressions, maintaining the integrity of existing functionality. Performance testing assesses the software's performance characteristics using test cases to evaluate response time, scalability, and reliability under different conditions. Security testing employs test cases to identify vulnerabilities and weaknesses in the software, ensuring its resistance to security threats. Usability testing evaluates the software's user experience, interface design, and ease of use through test cases, enhancing user satisfaction. Compatibility testing ensures the software's functionality across various platforms, devices, and browsers using test cases.

Overall, test cases serve as a critical tool in software testing, enabling testers to validate functionality, behavior, performance, security, and usability of software systems, ultimately contributing to the delivery of high-quality software product

## Unit Test for Medication Class

Objective: Verify the correct calculation of the total price for a specific quantity of medication.

Steps: Create an instance of the Medication class with a unit price of $5.00. Set the quantity to 10.

Calculate the total price.

Expected Outcome: The total price should be $50.00.

Actual Outcome: The total price is $50.00. Result: Pass

## Integration Test for Prescription and Inventory:

Objective: Validate that a prescription update reflects the correct change in inventory quantity.

Steps:

Create a prescription for a medication. Fill the prescription.

Check the inventory quantity for the medication.

Expected Outcome : The inventory quantity should decrease by the prescribed quantity.

Actual Outcome: The inventory quantity is reduced by the prescribed quantity.

Result: Pass

The execution of these test cases demonstrates the robustness and reliability of the pharmacy management system. Each test case, spanning unit testing, integration testing, system testing, acceptance testing, security testing, and regression testing, contributes to the overall assurance of the system's functionality, security, and stability. All tests have passed, indicating that the system meets the specified requirements and is ready for deployment.

## BUG TRACKING AND RESOLUTION

* + 1. **Bug Identification**

## Continuous Testing

The process of bug tracking starts with continuous testing throughout the development life cycle. This ensures that issues are identified early in the development process, reducing the likelihood of major defects in the final product. Continuous testing involves various testing methodologies, including unit testing, integration testing, system testing, and acceptance testing.

## User Feedback:

User feedback is a valuable source of bug identification. Users, stakeholders, or beta testers may encounter issues or unexpected behaviour’s that

were not uncovered during the testing conducted by developers. Their insights provide a real-world perspective on how the system performs in different scenarios.

## Automated Tools:

Employing automated testing tools and frameworks streamlines the bug identification process. These tools can perform repetitive and complex tests quickly and accurately, ensuring comprehensive coverage of codebase. Automated testing helps identify issues that may not be immediately apparent during manual testing.

## Bug Logging

* **Detailed Bug Reports**

When a bug is identified, the process of bug tracking involves creating detailed bug reports. These reports act as a communication bridge between developers, testers, and other stakeholders. A comprehensive bug report includes a clear description of the bug, steps to reproduce it, the expected behaviour, and the actual behaviour observed. Additionally, itincludes

information about the environment in which the bug occurred, such as the browser version, operating system, and device details.

## Severity and Priority

Assigning severity levels and priorities to bugs is crucial for effective bug management. The severity level indicates the impact of the bug on the system (e.g., critical bugs may cause system failure), while the priority reflects the order in which bugs should be addressed. Prioritization helps the development team focus on resolving critical issues first.

## Bug Resolution

* **Isolation of Bugs:**

Once a bug is logged, the development team works to isolate the bug by identifying the specific component or module where it originates. This step is essential to ensure that bug fixes are targeted and do not inadvertently affect other parts of the system.

## Collaborative Problem Solving:

Resolving bugs involves collaboration within the development team. Team members may conduct code reviews, engage in discussions, and participate in debugging sessions to understand the root cause of the bug. Collaborative problem- solving ensures that multiple perspectives contribute to the identification and resolution of issues.

## Testing Fixes

* **Unit Testing** :

After implementing code fixes, unit testing is conducted to validate that the specific component affected by the bug behaves as expected. Unit tests focus on the isolated unit of code, ensuring that it functions correctly.

## Regression Testing:

Regression testing is a critical aspect of bug resolution. It involves re- running previously executed tests to verify that the bug fixes do not introduce new issues or negatively impact other parts of the system. Comprehensive regression testing helps maintain the overall stability of the system.

## Verification and Validation

* **Verification**

Verification involves checking that the bug fix addresses the reported issue.

Developers follow the steps outlined in the bug report to verify that the expected behaviour is restored. Verification ensures that the code changes align with the intended resolution

## Validation

Validation goes a step further, involving users or Quality Assurance (QA) testers to retest the system and confirm that the reported bug no longer exists. This user-centric approach ensures that fixes meet end-user expectations and satisfaction.

The bug tracking and resolution process is a systematic and collaborative

effort aimed at ensuring the stability, reliability, and quality of the pharmacy management system. From continuous testing and user feedback to detailed bug reports, code fixes, and thorough testing of fixes, each step contributes to delivering

a robust and effective software solution. Documentation and knowledge sharing

.

# CONCLUSION

In the culmination of this Pharmacy Management System project, we have successfully designed, developed, and implemented a comprehensive solution aimed at enhancing the efficiency and accuracy of pharmacy operations. Through meticulous planning, rigorous analysis, and iterative development, we have created a system that addresses the intricate challenges faced by modern pharmacies.

Our primary objectives were to streamline medication inventory management, improve customer service, and provide a user-friendly interface for both pharmacy staff and customers. Through the incorporation of cutting-edge technologies and adherence to industry best practices, we believe this project has made significant strides towards achieving these goals.

One of the key highlights of our Pharmacy Management System is its robust inventory control mechanism. The system allows pharmacists to effortlessly monitor stock levels, track expiration dates, and manage restocking processes. This not only minimizes the risk of medication shortages or overstocking but also contributes to a more efficient supply chain within the pharmacy.

In conclusion, the Pharmacy Management System project represents a significant milestone in our quest for innovative solutions to real-world challenges. We are confident that the positive impact of this system will extend beyond the confines of our project scope, contributing to the overall efficiency and effectiveness of pharmacy operations. As we celebrate the completion of this endeavour, we remain dedicated to the ongoing pursuit of excellence in technology-driven solutions for the benefit of the healthcare industry and the community at large.

# REFERENCES & BIBLIOGRAPHY

## REFERENCES:

* Explore php(personal home page)

-Dave Grundgeiger

-Publisher:O’Reilly

-first Edition Jan 2002

* ApachevServer Basics

-Roger S pressman

-McGraw-Hil Publishing

-5thEdition

* All in one- PHP,MySQL

-Raja Raman

* Web Programming
  + W.B Sabesta
  + 3rdEdition

**BIBILOGRAPHY:**

* [**http://www.apache.net**](http://www.apache.net/)
* [**~~http://www.wikipedia.c~~om**](http://www.wikipedia.com/)
* [**~~http://www.w3schools.com~~**](http://www.w3schools.com/)
* [**~~http://www.php.co~~**](http://www.php.com/)**~~m~~**
* [**~~http://www.database~~mysql.com**](http://www.databasemysql.com/)
* [**~~www.meipolytechnic.co~~**](http://www.meipolytechnic.com/)**~~m~~**