**Chapter -1**

TITLE PAGE-CENTERED TNR 17-POINT BOLD ALL CAPS

* 1. **INTRODUCTION**

Inventory management is the process of efficiently monitoring the constant flow of units into and out of an existing inventory. This process usually involves controlling the transfer in of units in order to prevent the inventory from becoming too high, or dwindling to levels that could put the operation of the company into difficulties. Inventory management is very important for big business and private owned organizations especially where there are a lot of orders are being placed everyday and there are lot of materials and the maintenance is really important which the system will do and also will record the time taken to process an order and this system is really important as it can help the organizations to be alerted when the level of inventory is very low .and focuses on the three aspects of inventory management and prevent from failures in the future .

Inventory management also demands a solid understanding of how long it will take for those materials to transfer out of the inventory to be established. By Knowing these two important lead key aspects makes it possible to know when to place an order and how many units must be ordered to keep production running smoothly.

* 1. **SCOPE OF THE CAPSTONE PROJECT**
     1. **Problem statement:**

The process of manually looking for the inventory available in stock is really time consuming and cumbersome that includes the person responsible to see how many materials are in the inventory , how many materials have been ordered and received and keep the record of the time that will take for the suppliers to process orders and other various tasks included in inventory .At current the inventory is managed and update on an excel spreadsheet which has been an outdated form of recording and maintain the inventory of material sas it is really difficult to enter the data and then update the data on daily basis. the inventory management system will solve the problems as everything about the inventory will be stored in a database which will be much easier to marinating and update and this will also save time and cost effective for the organization and will automate many inventory related tasks. The project will implement a database for inventory management which will include the vendor analysis performance of the products and calculate the cost of materials at the end of the project .Online inventory management system will provide all the information needed and required for inventory related in understanding in easy way and that will reduce the time taken for the inventory manager to record all the products that are in inventory which takes too much time. This will help the inventory manager to list out all the details using the system or search all the inventory information.

**1.2.2. Objectives:**

**1.2.1 Research Objectives**

This section illustrates the three types of objectives general, specific and the learning objectives that are required in order to complete this project.

**1.2.1.1 General Objectives**

The following points are considered the general objective of the research

* Conduct and research the real life workflow and be able to understand and identify real life and business problems.
* To propose a solution to the problems identified.
* Analyze and design a solution that fulfills the client’s needs.

**1.2.1.2 Specific objectives**

The research will help to cover the following objectives

* Identify the problems.
* Identify the type of system that would help solving the problem.
* Study and search about similar systems.
* Similar soft wares that is used to help in solving the problems.
* Understand and specify the requirements of the system.
* To specify the tools needed to build the system and how to use them.
* Design the architecture and the interfaces for the proposed system
* Implement a functional system and the test the system.

**1.2.1.3 Learning Objectives**

After doing the literature review and conducting the research about the system, there are some expected deliverables on these following topics:

Understand what the task management system are , how they work ,and

their importance in today’s business world.

Identify the data, processes and knowledge involved in the problem.

Prepare system requirements analysis.

Learn how to design and implement a system into real life workflow.

**1.2.3 Capstone project description**

This project focuses on the development and implementation of a user-friendly Inventory Management System (IMS) designed to streamline inventory control processes for businesses. The system will leverage [technology stack, e.g., PHP] to provide a centralized platform for managing product data, stock levels, and various inventory-related tasks.

In today's dynamic business landscape, efficient inventory management is crucial for success. Businesses of all sizes grapple with maintaining optimal stock levels to meet customer demand while minimizing costs and avoiding stockouts. Traditional methods, often reliant on spreadsheets and manual processes, are prone to errors and inefficiencies. This can lead to inaccurate data, overstocking, understocking, and ultimately, lost sales and dissatisfied customers.

This project tackles these challenges by developing a robust Inventory Management System (IMS) built using PHP, a widely adopted and versatile programming language. This PHP-based IMS will provide a centralized platform for managing all aspects of inventory, from product catalog creation and real-time stock tracking to automated reordering and insightful data analysis.

This report delves into the development process and functionalities of the Inventory Management System. We'll explore the system's architecture, key features powered by PHP, and the potential benefits businesses can expect to gain. Additionally, the report will discuss the implementation process, testing procedures, and future considerations for ongoing system improvement.

**1.2.4 Capstone project Deliverables**

This section outlines the key deliverables for the Inventory Management System (IMS) project developed using PHP:

* + **Product Management:**
    - Create and manage a comprehensive product catalog with detailed information (descriptions, specifications, images, SKUs).
    - Categorize and organize products for easier navigation and searching.
  + **Inventory Tracking:**
    - Track inventory levels in real-time for each product across various locations (if applicable).
    - Update stock levels manually or through integration with sales channels (optional).
  + **Purchase Order Management:**
    - Generate purchase orders automatically based on predefined reorder points and demand forecasts.
    - Manage supplier information and track purchase order history.
  + **Reporting and Analytics:**
    - Generate customizable reports on inventory performance, including stock levels, sales trends, and low-stock alerts.
    - Analyze data to identify areas for improvement in inventory management practices.
  + **User Management:**
    - Implement a user authentication system with different user roles and access permissions (e.g., admin, manager, user).
    - Manage user accounts and assign roles to control system access.
  + **Security Measures:**
    - Implement secure coding practices and user authentication mechanisms to protect sensitive inventory data.
    - Regular testing and updates to ensure system security against vulnerabilities.

**1.2.5 Key milestone**

|  |  |  |  |
| --- | --- | --- | --- |
| Milestone | Description | Target Completion Date | Deliverables |
| 1. Requirements Gathering | Conduct user interviews, define system functionalities, and determine data requirements. | (Project Start Date + 2 weeks) | Requirements Document |
| 2. System Design & Architecture | Design the system architecture, define technology stack (PHP frameworks, database), and create user interface mockups. | (Project Start Date + 4 weeks) | System Design Document, UI Mockups |
| 3. Database Design & Development | Design and develop the database schema to store inventory data, users (if applicable), and system settings. | (Project Start Date + 6 weeks) | Database Schema Document |
| 4. PHP Development - Phase 1 | Develop core functionalities like product management, inventory tracking, and basic reporting. | (Project Start Date + 10 weeks) | Functional Inventory Management System (Phase 1) |
| 5. Unit Testing | Conduct unit testing of individual PHP modules to ensure functionality. | (Project Start Date + 12 weeks) | Unit Testing Reports |
| 6. PHP Development - Phase 2 | Implement additional functionalities like purchase order management, user management (if applicable), and advanced reporting. | (Project Start Date + 16 weeks) | Functional Inventory Management System (Phase 2) |
| 7. Integration Testing | Test the system as a whole, including database interactions and user interface functionality. | (Project Start Date + 18 weeks) | Integration Testing Reports |
| 8. User Acceptance Testing (UAT) | Conduct UAT sessions with designated users to gather feedback on system usability and functionality. | (Project Start Date + 20 weeks) | User Acceptance Testing Reports |
| 9. System Documentation & Training | Develop user manuals and training materials for effective system utilization. | (Project Start Date + 22 weeks) | User Manuals, Training Materials |
| 10. System Deployment | Deploy the Inventory Management System onto a web server and configure system settings. | (Project Start Date + 23 weeks) | Deployed Inventory Management System |
| 11. Project Completion & Handover | Finalize project documentation, address identified issues, and officially hand over the system to stakeholders. | (Project Start Date + 24 weeks) | Project Documentation, Handover Documentation |

**1.2.6 Constraints**

Developing an Inventory Management System (IMS) using PHP involves several constraints that need to be considered throughout the project lifecycle. Here's a breakdown of some key limitations to be aware of:

**1.2.6.1 Technology Stack:**

* **PHP Version:** The choice of PHP version (e.g., 7.4, 8.1) might limit compatibility with older server environments or existing libraries.
* **Framework Selection:** While PHP frameworks like Laravel or Symfony can offer advantages, using one might introduce a learning curve for developers unfamiliar with the framework.

**1.2.6.2 Project Scope:**

* **Feature Creep:** Adding excessive features beyond the initial scope can lead to project delays and resource constraints. Prioritize essential functionalities and consider future enhancements in upcoming versions.
* **Data Integration Complexity:** Integrating the IMS with existing systems (e.g., accounting software, e-commerce platforms) might require additional development effort depending on API availability and data structure compatibility.

**1.2.6.3 Resource Limitations:**

* **Development Team Expertise:** The skillset of the development team using PHP will determine the complexity of features that can be implemented effectively. Training or additional resources might be necessary for advanced functionalities.
* **Time Constraints:** A fixed project timeline can constrain the level of detail and comprehensiveness of the system. Prioritize core functionalities and plan for future iterations based on user needs and feedback.

**1.2.6.4 Security Considerations:**

* **Data Security:** Protecting sensitive inventory data requires implementing secure coding practices, user authentication mechanisms, and regular security updates to address potential vulnerabilities.
* **Access Control:** Implementing user roles and access permissions within the system adds complexity but is crucial for ensuring data integrity and preventing unauthorized access.

**1.2.6.5 Scalability and Maintainability:**

* **Future Growth:** The system should be designed with scalability in mind to accommodate future expansion in terms of data volume and user base. Consider modular architecture and database optimization techniques.
* **Code Maintainability:** Using well-documented and commented code with proper coding practices allows for easier maintenance and future modifications.

**Estimated Capstone project Duration:** 4 month**s**

**Estimated Capstone project cost:** 2000 \* 4 = ₹ 8000.00

**Chapter – 2**

**2.1 CAPSTONE PROJECT PLANNING**

**2.1.1 WORK BREAKDOWN STRUCTURE (WBS)**

|  |  |  |
| --- | --- | --- |
| Deliverable | Tasks | Estimated Effort (in days) |
| 1. Requirements Gathering | 1.1 Conduct user interviews | 2 |
|  | 1.2 Define system functionalities | 3 |
|  | 1.3 Determine data requirements | 2 |
|  | Deliverable: Requirements Document |  |
| 2. System Design & Architecture | 2.1 Design system architecture | 3 |
|  | 2.2 Define technology stack (PHP, frameworks, database) | 2 |
|  | 2.3 Create user interface mockups | 4 |
|  | Deliverable: System Design Document, UI Mockups |  |
| 3. Database Design & Development | 3.1 Design database schema | 3 |
|  | 3.2 Develop database tables and relationships | 4 |
|  | Deliverable: Database Schema Document |  |
| 4. PHP Development - Phase 1 | 4.1 Develop product management module | 5 |
|  | 4.2 Develop inventory tracking module | 4 |
|  | 4.3 Develop basic reporting module | 3 |
|  | Deliverable: Functional Inventory Management System (Phase 1) |  |
| 5. Unit Testing | 5.1 Unit test individual PHP modules | 4 |
|  | Deliverable: Unit Testing Reports |  |
| 6. PHP Development - Phase 2 | 6.1 Develop purchase order management module (optional) | 3 |
|  | 6.2 Develop user management module (optional) | 4 |
|  | 6.3 Develop advanced reporting functionalities | 5 |
|  | Deliverable: Functional Inventory Management System (Phase 2) |  |
| 7. Integration Testing | 7.1 Test system functionality as a whole | 3 |
|  | 7.2 Test database interactions | 2 |
|  | 7.3 Test user interface functionality | 2 |
|  | Deliverable: Integration Testing Reports |  |
| 8. User Acceptance Testing (UAT) | 8.1 Conduct UAT sessions with designated users | 2 |
|  | 8.2 Gather user feedback on system usability and functionality | 1 |
|  | Deliverable: User Acceptance Testing Reports |  |
| 9. System Documentation & Training | 9.1 Develop user manuals for system operation | 3 |
|  | 9.2 Create training materials for users | 2 |
|  | Deliverable: User Manuals, Training Materials |  |
| 10. System Deployment | 10.1 Deploy the IMS onto a web server | 1 |
|  | 10.2 Configure system settings | 1 |
|  | Deliverable: Deployed Inventory Management System |  |
| 11. Project Completion & Handover | 11.1 Finalize project documentation | 1 |
|  | 11.2 Address identified issues from UAT | 1 (variable) |
|  | 11.3 Officially handover the system to stakeholders | 1 |
|  | Deliverable: Project Documentation, Handover Documentation |  |

**2.1.2 TIMELINE DEVELOPMENT – SCHEDULE**

**2.1.2.1 Identify the activities and tasks needed to produce each work package.**

**Work Package 1: Requirements Gathering**

* **Activities:**
  + Conduct user interviews with key stakeholders (managers, warehouse personnel) to understand their needs and pain points.
  + Analyze existing inventory management practices (manual or software-based).
  + Define user stories outlining desired functionalities (adding products, tracking stock, generating reports).
  + Identify data requirements (product details, stock levels, purchase orders) and data sources.
* **Tasks:**
  + Schedule and conduct user interviews.
  + Document user needs and pain points.
  + Create a list of user stories with acceptance criteria.
  + Develop a data dictionary outlining data elements and their attributes.

**Work Package 2: System Design & Architecture**

* **Activities:**
  + Design the overall system architecture, including data flow and user interaction.
  + Select the appropriate technology stack (PHP version, web server, database).
  + Create user interface (UI) mockups for key functionalities.
  + Define system security measures (user authentication, data encryption).
* **Tasks:**
  + Develop system architecture diagrams.
  + Research and choose suitable technologies based on project needs and team expertise.
  + Design UI mockups for product management, inventory tracking, and reporting modules.
  + Define user roles and access permissions within the system.
  + Research and implement secure coding practices and user authentication mechanisms.

**Work Package 3: Database Design & Development**

* **Activities:**
  + Design the database schema to store inventory data efficiently.
  + Define tables, columns, data types, and relationships between tables.
  + Develop database scripts to create tables and populate initial data (if applicable).
  + Test database functionality and ensure data integrity.
* **Tasks:**
  + Develop an Entity-Relationship Diagram (ERD) to visualize the database schema.
  + Define data types for each attribute in the database tables.
  + Write SQL scripts to create tables and establish relationships.
  + Populate initial data (e.g., product categories) if necessary for testing purposes.
  + Conduct unit testing on database queries and data manipulation procedures.

**Work Package 4: PHP Development - Phase 1**

* **Activities:**
  + Develop core functionalities of the Inventory Management System.
  + Implement product management module (adding, editing, deleting products).
  + Develop inventory tracking module (real-time stock levels, low stock alerts).
  + Build basic reporting module (inventory overview, stock reports).
* **Tasks:**
  + Code functionalities for product creation, editing, and deletion using PHP.
  + Develop database interactions to store and retrieve product information.
  + Implement functionalities for adding, updating, and displaying stock levels.
  + Generate basic reports on inventory overview, stock levels by category, and low stock alerts.
  + Conduct unit testing on individual PHP modules for product management and inventory tracking.

**Work Package 5: Unit Testing**

* **Activities:**
  + Conduct unit testing of all developed PHP code modules.
  + Test each functionality in isolation to ensure it works as expected.
  + Identify and fix bugs in the code.
* **Tasks:**
  + Develop unit tests using a testing framework (e.g., PHPUnit).
  + Write test cases for product management functionalities, inventory tracking logic, and reporting generation.
  + Execute unit tests and analyze results to identify code errors.
  + Debug and fix identified errors in the PHP code.

**Work Package 6: PHP Development - Phase 2 (Optional)**

* **Activities:**
  + Develop additional functionalities based on project requirements.
  + Implement purchase order management module (generating orders, managing suppliers).
  + Develop user management module (creating user accounts, assigning roles).
  + Enhance reporting functionalities with advanced data analysis.
* **Tasks:**
  + Code functionalities for creating purchase orders, managing suppliers, and tracking order status.
  + Develop user registration, login, and role management features.
  + Implement functionalities for generating detailed inventory reports with sales trends and data visualization.
  + Conduct unit testing on newly developed functionalities.

**Work Package 7: Integration Testing**

* **Activities:**
  + Test the functionalities of the entire system as a whole.
  + Ensure seamless integration between different modules (product management, inventory tracking, reporting).
  + Test user interface functionality and data flow across the system.
* **Tasks:**
  + Develop integration test cases to verify interaction between different modules.
  + Test user registration, login, and access control based on assigned roles.
  + Verify data consistency.
    - 1. **Identify resources for each task (e.g., time, knowledge, monetary costs etc.)**

Here's a breakdown of resources needed for each task in developing the

Inventory Management System (IMS):

**Work Package 1: Requirements Gathering**

* **Time:** 2-3 days
* **Knowledge:** Business Analyst or Project Manager with experience in user interviews and requirements gathering. Understanding of inventory management practices is beneficial.
* **Monetary Costs:** May involve minimal costs if travel or external consultants are needed for user interviews.

**Work Package 2: System Design & Architecture**

* **Time:** 3-5 days
* **Knowledge:** Software Architect or experienced developer with strong understanding of system design principles, web development technologies, and security best practices.
* **Monetary Costs:** May involve costs for design tools or cloud-based prototyping platforms (optional).

**Work Package 3: Database Design & Development**

* **Time:** 3-5 days
* **Knowledge:** Database Administrator or developer with expertise in database design (normalization), SQL scripting, and chosen database technology (e.g., MySQL).
* **Monetary Costs:** May involve minimal costs for database management tools (optional)

**Work Package 4: PHP Development - Phase 1**

* **Time:** 12-15 days
* **Knowledge:** PHP developer(s) with experience in web development, database interaction using PHP, and basic reporting functionalities.
* **Monetary Costs:** Salary/hourly rate of the development team. Costs for any third-party libraries used (if applicable).

**Work Package 5: Unit Testing**

* **Time:** 2-4 days
* **Knowledge:** Developer(s) familiar with unit testing frameworks (e.g., PHP Unit) and testing methodologies.
* **Monetary Costs:** Salary/hourly rate of the development team.

**Work Package 6: PHP Development - Phase 2**

* **Time:** Varies depending on complexity of additional features (5-10 days)
* **Knowledge:** PHP developer(s) with experience in user authentication, purchase order management systems, and advanced reporting functionalities.
* **Monetary Costs:** Salary/hourly rate of the development team. Costs for any third-party libraries used (if applicable).

**Work Package 7: Integration Testing**

* **Time:** 2-3 days
* **Knowledge:** QA Tester or developer familiar with testing system integration and user interface functionality.
* **Monetary Costs:** Salary/hourly rate of the testing personnel.

**2.1.2.3 Estimate how long it will take to complete each task. Consider constraints - resources, time, knowledge.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PHASE | DURATION  (WEEKS) | START  DATE | END  DATE | KEY TASKS |
| **Project Definition** | 2 | JAN 1  2024 | JAN 14  2024 | **Deliverable:** Project Definition Document  **Work Packages:**   * Define project goals and target audience * Identify key features and functionalities * Create user stories and acceptance criteria * Estimate project timeline and budget * Choose technology stack and development tools |
| **Design and Development** | 7 | JAN 15  2024 | FEB 25  2024 | **Deliverable:** Weather Application Prototype  **Work Packages:**   * Front-End Development:   + Design user interface mock-ups   + Develop HTML structure with semantic markup   + Implement responsive CSS styles   + Write JavaScript code for user interactions and data display * Back-End Development:   + Integrate Visual Crossing Weather API   + Implement data storage and retrieval mechanisms   + Handle errors and data validation   + Develop server-side components (if applicable) * Testing and Integration:   + Conduct unit testing of individual components   + Perform integration testing of application flow   + Test application in different browsers and devices   + Ensure accessibility compliance |
| **Deployment and Maintenance** | 4 | FEB 26  2024 | MAR 25  2024 | Deliverable: Deployed Weather Application  Work Packages:   * Deploy application to chosen platform * Document user guides and FAQs * Address bug reports and user feedback * Release new features and updates * Monitor application performance and security   Note: This is a general WBS and may need to be adapted to your specific needs and chosen technology stack.  Additional considerations:   * Ethical considerations: When using the Visual Crossing Weather API, be sure to follow their terms of service and be mindful of any data privacy or security concerns. * Accessibility: Ensure your application is accessible to users with disabilities by following accessibility best practices. * Sustainability: Consider the environmental impact of your application and choose technologies and practices that promote sustainability. |

**2.1.2.4 Determine which tasks are dependent on other tasks and develop a critical path**

**Dependencies:**

1. System Requirements must be completed before Database Design can start, as the database structure needs to align with the functionality requirements.
2. Database Design must be completed before Product Management Module, Purchase Management Module, Sales Management Module, and Reporting Module can be developed, as these modules rely on the database to store and retrieve data.
3. Product Management Module needs to be at least partially functional before Sales Management Module can be fully developed, as orders would require product information.
4. Purchase Management Module and Sales Management Module can be developed somewhat concurrently, but both should be operational before Reporting Module can be built, as reports need data from purchases and sales.
5. Security Implementation can be done in parallel with most other development tasks, but should be finalized before deployment.
6. User Interface Design & Development depends on the functionalities being developed in other modules, and ideally progresses iteratively alongside them.

**Critical Path**:

Based on the dependencies, the critical path (longest sequence of dependent tasks) likely consists of:

* Database Design
* Product Management Module (Core Functionalities)
* Purchase Management Module
* Sales Management Module
* Critical Reports (e.g., Stock Levels, Sales Reports)
* Security Implementation (integrated throughout)

The total critical path duration ranges from 12 to 17 weeks, suggesting this is the minimum time possible to complete the core functionalities of your inventory management system. However, remember that:

* This is an estimation and actual time may vary.
* Other tasks outside the critical path can still be worked on concurrently to optimize overall development time.
* Testing, bug fixing, and deployment will add additional time.

**2.1.2.5 Develop a schedule of all activities and tasks - weekly and monthly. Work out when each task is scheduled to begin and end. Use a Gantt chart.**

|  |  |  |  |
| --- | --- | --- | --- |
| **WEEK** | **STARTING**  **MONDAY** | **ENDING**  **SUNDAY** | **TASKS** |
| **Week 1**  (Jan 1 - Jan 7) | Jan 1 | Jan 7 |  |
| **Week 2**  (Jan 8 - Jan 14) | Jan 8 | Jan 14 |  |
| **Week 3**  (Jan 15 - Jan 21) | Jan 15 | Jan 21 |  |
| **Week 4**  (Jan 22 - Jan 28) | Jan 22 | Jan 28 |  |
| **Week 5**  (Jan 29 - Feb 4) | Jan 29 | Feb 4 |  |
| **Week 6**  (Feb 5 - Feb 11) | Feb 5 | Feb 11 |  |
| **Week 7**  (Feb 12 - Feb 18) | Feb 12 | Feb 18 |  |
| **Week 8**  (Feb 19 - Feb 25) | Feb 19 | Feb 25 |  |
| **Week 9**  (Feb 26 - Mar 3) | Feb 26 | Mar 3 |  |
| **Week 10**  (Mar 4 - Mar 10) | Mar 4 | Mar 10 |  |
| **Week 11**  (Mar 11 - Mar 17) | Mar 11 | Mar 17 |  |
| **Week 12**  (Mar 18 - Mar 24) | Mar 18 | Mar 24 |  |
| **Week 13**  (Mar 25) | Mar 25 | Mar 25 |  |

* + - 1. **Gantt Chart:**

|  |  |  |
| --- | --- | --- |
| **TASK** | **START DATE** | **END DATE** |
|  | 2024-01-01 | 2024-01-01 |
|  | 2024-01-08 | 2024-01-14 |
|  | 2024-01-15 | 2024-01-21 |
|  | 2024-01-22 | 2024-01-28 |
|  | 2024-01-29 | 2024-02-04 |
|  | 2024-02-05 | 2024-02-11 |
|  | 2024-02-12 | 2024-02-18 |
|  | 2024-02-19 | 2024-02-25 |
|  | 2024-02-26 | 2024-03-24 |
|  | 2024-03-25 | 2024-03-25 |

* + 1. **COST BREAKDOWN STRUCTURE (CBS)**
       1. **Estimate the labour cost of work**

## **Estimate the labour cost of work**

* **Planning and Design:**
* Gather and Document Requirements: 2 hours
* Design Database Schema ::3 hours
* Total labour cost: (2 + 3) \* INR 50 = INR 250
* **Database Development:**
* Create MySQL Database and Create Tables :2hours
* Total labour cost: (2) \* INR 100 = INR 200
* **Front-End Development:**
* Create HTML Templates:4 hours
* Write CSS Stylesheets**:** 6 hours
* Creating front end user interface :5 hours
* Integrate dynamic content using server-side scripting (PHP):8 hours
* Use AJAX for dynamic data updates without page reloads : 5 hours
* Total labour cost: ( 28 ) \* INR 100 = INR 2,800
* **Back-End Development**:
* Write PHP Scripts for CRUD Operations:5 hours
* Implement User Authentication:4 hours
* Develop Inventory Management Logic:3 hours
* Create Reporting Functionality:6 hours
* Total labour cost: ( 18 ) \* INR 100 = INR 1,800
* **Testing and Deployment:**
* Create test cases for all functionalities of the system.:5 hours
* Test user interface, database interactions, and business logic:4 hours
* Deploy the website files and scripts onto the server:3 hours
* Total labour cost: ( 12 ) \* INR 100 = INR 1,200
* **Documentation:**
* Create a user manual for the inventory management system.: 6 hours
* Total labour cost: ( 12 ) \* INR 50 = INR 600

**Total labour cost = INR 6850**

* + - 1. **Estimate the cost of materials**

As a full stack web development project , the materials needed for inventory management system are primarily computational resources such as a computer, software, and cloud services. Since these materials are not physical, the cost is not directly related to the number of units used. Instead, the cost is based on usage and access to these resources. Here is an estimate of the cost of materials for the project:

* Computer

o Assuming the use of personal computer or laptop

o Cost: N/A (already owned)

* Software

o PHP Programming language: Free

o LARAVEL Framework ,Bootstrap Framework: Free

o Natural Language Toolkit (NLTK) library: Free Cloud Services

o Google Colab for development and testing: Free Amazon Web Services (AWS) or Google Cloud Platform (GCP) for deployment: Estimated at INR 2000 (assuming minimal usage)

Total cost of materials: INR 2000

**2.1.3.3 Overhead costs**

Inventory management, while crucial for a smooth-running business, incurs various overhead costs beyond the direct cost of the goods themselves. These indirect expenses, often overlooked, can significantly impact your bottom line.

**Let's explore the different types of overhead costs associated with inventory management:**

**1. Storage Costs:**

* **Warehouse rent or lease:** Cost of storing inventory in a dedicated space, including utilities and maintenance.
* **Storage equipment:** Depreciation or rental of shelves, racks, bins, and other storage aids.
* **Security & insurance:** Protecting inventory from theft, damage, and natural disasters.

**2. Personnel Costs:**

* **Salaries and benefits:** Wages for warehouse staff, inventory managers, and administrative personnel.
* **Training and development:** Equipping staff with necessary skills for efficient inventory management.

**3. Technology Costs:**

* **Inventory management software:** Licensing fees and maintenance costs for inventory tracking and optimization tools.
* **Hardware:** Computers, scanners, barcodes, and other technology supporting inventory processes.

**4. Administrative Costs:**

* **Inventory counting and audits:** Costs associated with periodic physical inventory verification.
* **Documentation and reporting:** Expenses for recordkeeping, generating reports, and complying with regulations.
* **Returns processing:** Handling and managing the return of damaged or unwanted inventory.

**5. Opportunity Costs:**

* **Capital tied up in inventory:** Holding inventory represents invested capital that could be used elsewhere.
* **Obsolescence and shrinkage:** Loss of value due to outdated products or product loss through damage or theft.
  + - 1. **Build contingency into your CBS**

Here's a revised CBS table for the Inventory Management System (IMS) project using PHP, incorporating contingency buffers for potential risks and uncertainties:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Work Package | Task | Estimated Time (days) | Contingency Buffer (%) | Revised Time (days) | Constraints |
| 1. Requirements | \* User Interviews  \* Define Functionalities \* Determine Data Needs | 1-2 | 20% | 2.4-3.6 | User Availability, Project Complexity |
|  | Total |  | 2-3 |  | 2.4-3.6 |
| 2. System Design | \* Design Architecture  \* Define Tech Stack  \* Create UI Mockups | 2-3 | 30% | 3.4-4.9 | Project Complexity, Team Expertise |
|  | Total |  | 4-7 |  | 5.1-10.5 |
| 3. Database Design | \* Design Schema  \* Develop Tables & Relationships | 1-2 | 25% | 1.3-2.5 | Data Complexity, Table Interactions |
|  | Total |  | 3-5 |  | 3.75-6.25 |
| 4. PHP Development (Phase 1) | \* Product Management Module  \* Inventory Tracking Module  \* Basic Reporting Module | 3-5 | 20% | 3.6-6 | Module Complexity, Data Validation |
|  | Total |  | 7-11 |  | 8.4-13.2 |
| 5. Unit Testing | \* Test Individual Modules | 2-4 | 15% | 2.3-4.6 | Module Complexity, Testing Thoroughness |
|  | Total |  | 2-4 |  | 2.3-4.6 |
| 6. PHP Development (Phase 2) - | \* Purchase Order Management  \* User Management  \* Advanced Reporting | 2-5 (per module) | 20% | 2.4-6 (per module) | Order Processing Logic, Security Considerations |
|  | Total |  | 7-14 |  | 8.4-16.8 (per module) |
| 7. Integration Testing | \* Test System Functionality  \* Test Database Interactions  \* Test UI Functionality | 2-3 | 25% | 2.5-3.75 | System Complexity, Number of Modules |
|  | Total |  | 4-7 |  | 5-8.75 |
| 8. User Acceptance Testing (UAT) | \* Conduct UAT Sessions \* Gather User Feedback | 1-2 | 10% | 1.1-2.2 | User Availability, Testing Scope |
|  | Total |  | 1.5-2.5 |  | 1.65-2.75 |
| 9. Documentation & Training | \* User Manuals  \* Training Materials | 2-3 | 15% | 2.3-3.5 | System Complexity, Training Scope |
|  | Total |  | 3-5 |  | 3.45-5.75 |
| 10. System Deployment | \* Deploy to Web Server  \* Configure System Settings | 0.5-1 | 10% | 0.55-1.1 | Server Configuration Complexity |
|  | Total |  | 1-2 |  | 1.1-2.2 |
| 11. Project Completion & Handover | \* Finalize Documentation  \* Address UAT Issues (variable)  \* Handover to Stakeholders | 0.5-1 | 25% | 0.63-1.25 | Documentation Completeness, Issue Severity |
|  | Total |  | 1-4.5 |  | 1.25-5.63 |

* + - 1. **Final-check**

Here are some ways to check the cost estimates against the available budget and

control costs if the CBS comes in higher:

• **Review the estimates:** Once you have estimated the cost of materials, labor,

overheads, and contingency, it's essential to review the estimates and ensure that they are realistic and within the available budget.

• **Identify cost-saving opportunities:** Look for opportunities to save costs without

compromising the project's quality. For example, you can opt for open-source

machine learning libraries instead of paid ones or choose less expensive data storage options.

• **Negotiate with suppliers:** If you have identified cost-saving opportunities, you can negotiate with your suppliers to get better prices. For example, you can negotiate bulk discounts on data storage or machine learning software.

• **Prioritize tasks:** If the cost estimates are higher than the available budget, you can prioritize tasks and focus on critical tasks first. This approach can help you complete the most important parts of the project within the available budget.

**2.1.4 CAPSTONE PROJECT RISKS ASSESSMENT**

**Risk Analysis Report for Inventory Management System**

Developing an Inventory Management System (IMS) using PHP involves various risks that can impact project timelines, costs, and functionalities. Here's a breakdown of potential risks and mitigation strategies:

**High-Impact Risks:**

* **Incomplete or inaccurate requirements:**
  + **Impact:** Developing functionalities that don't meet user needs or missing crucial features. Delays due to requirement changes.
  + **Mitigation:** Conduct thorough user interviews with diverse stakeholders. Utilize user story creation with clear acceptance criteria. Validate requirements with prototypes or mockups.
* **Technical resource limitations:**
  + **Impact:** Lack of developer expertise with PHP, database technologies, or chosen frameworks could lead to delays or functionality limitations.
  + **Mitigation:** Evaluate team skillsets early on. Consider upskilling or hiring developers with relevant experience. Explore alternative open-source libraries or frameworks that better suit the team's expertise.
* **Project scope creep:**
  + **Impact:** Adding features or functionalities outside the initial scope can lead to budget overruns, delays, and potential resource overload.
  + **Mitigation:** Clearly define project scope in a requirements document and obtain stakeholder sign-off. Implement change control procedures for managing scope changes and assessing their impact.

**Medium-Impact Risks:**

* **Data security vulnerabilities:**
  + **Impact:** Unauthorized access to sensitive inventory data or system breaches could lead to financial losses and reputational damage.
  + **Mitigation:** Implement secure coding practices, user authentication mechanisms, and data encryption for sensitive information. Regularly conduct security audits and penetration testing.
* **Database performance issues:**
  + **Impact:** Slow system response or data retrieval issues as the inventory data volume grows.
  + **Mitigation:** Select a database technology optimized for handling large datasets. Design efficient database schema with proper normalization. Optimize database queries and utilize caching mechanisms.
* **Third-party library or framework conflicts:**
  + **Impact:** Compatibility issues between chosen libraries or frameworks can cause development delays or unexpected functionality behavior.
  + **Mitigation:** Research chosen libraries carefully to ensure compatibility with PHP version and other technologies used. Thoroughly test third-party integrations before deployment. Consider alternatives with a wider developer community and support resources.

**Low-Impact Risks:**

* **Server downtime or outages:**
  + **Impact:** Limited system availability during planned or unplanned server maintenance or outages.
  + **Mitigation:** Choose a reliable web hosting provider with a strong uptime record. Implement backup and disaster recovery strategies for data and system functionality.
* **Limited user adoption of the system:**
  + **Impact:** Users might resist using the new system, impacting its effectiveness in managing inventory.
  + **Mitigation:** Involve stakeholders in project development to enhance user buy-in. Develop user-friendly interfaces and provide comprehensive training materials.
* **Minor bug fixes after deployment:**
  + **Impact:** Potential bugs may require minor fixes after system deployment, causing short disruptions.
  + **Mitigation:** Conduct rigorous unit and integration testing before deployment. Implement a bug tracking and resolution process for post-deployment issues.

**Risk Management Strategies:**

* **Regular Risk Assessments:** Periodically review potential risks throughout the project lifecycle and update mitigation strategies as needed.
* **Communication & Transparency:** Maintain open communication with stakeholders regarding risks, mitigation plans, and potential impacts.
* **Contingency Planning:** Develop contingency plans for high-impact risks to minimize project disruption if they occur.

**2.1 Requirements Specification**

**3. Functional Requirements**

**3.1 Product Management:**

* Users can add new products to the system.
* Product details include (but are not limited to):
  + Product name
  + Description
  + Category
  + SKU (Stock Keeping Unit)
  + Unit price
  + Supplier information (optional)
  + Image (optional)
* Users can edit existing product information.
* Users can deactivate or archive products that are no longer actively sold.

**3.2 Stock Tracking:**

* The system should track real-time stock levels for each product.
* Users can add or remove stock quantities for existing products.
* The system should automatically update stock levels based on additions and removals.
* Low stock alerts should be generated for products reaching a predefined threshold.

**3.3 Reporting:**

* Users can generate various reports on inventory data.
* Reports may include (but are not limited to):
  + Inventory overview (total stock value, number of items)
  + Stock levels by category
  + Product sales reports (optional)
  + Low stock reports
* Reports should be exportable to formats like PDF or CSV for further analysis.

**3.4 User Management (Optional):**

* The system can be implemented with user accounts and role-based access control.
* Different user roles can have varying access permissions to functionalities (e.g., administrators can manage users, while regular users can only manage assigned products).

**3.5 Purchase Order Management (Optional):**

* Users can create purchase orders for new inventory stock from suppliers.
* Purchase orders should specify product details, quantity, and price.
* The system can track the status of purchase orders (pending, approved, received).

**4. Non-Functional Requirements**

**4.1 Security:**

* The system should implement secure coding practices to prevent unauthorized access and vulnerabilities.
* User authentication and authorization mechanisms should be in place to restrict access based on user roles.
* Sensitive data like product costs and supplier information (if included) should be encrypted.

**4.2 Performance:**

* The system should respond to user actions promptly and efficiently.
* The system should be scalable to accommodate a growing inventory data volume.

**4.3 Usability:**

* The user interface should be intuitive and user-friendly for personnel with varying technical skills.
* Online help documentation or user manuals should be available within the system.

**5. System Interfaces**

The IMS will interface with a database to store inventory data (product information, stock levels, etc.). The specific database technology (e.g., MySQL) will be chosen based on project requirements and existing infrastructure.

**6. Other Requirements**

* The system should be compatible with major web browsers (e.g., Chrome, Firefox, Safari).
* The system should be well-documented for future maintenance and updates.

**7. Success Criteria**

The IMS project will be considered successful if it meets the following criteria:

* All functional requirements are implemented and tested thoroughly.
* The system is user-friendly and adopted by [Company Name]'s personnel.
* The system improves inventory management efficiency by providing real-time data and automated stock tracking.
* The system operates securely and protects sensitive inventory data.

**8. Open Issues**

* Specific details like low stock threshold levels or user role permissions might need further discussion and refinement.
* The need for functionalities like purchase order management or sales reporting will be confirmed based on user needs.

## **User Input for Inventory Management System (PHP)**

The Inventory Management System (IMS) will require user input for various functionalities. Here's a breakdown of potential user input scenarios:

**Product Management:**

* **Adding a new product:**
  + Product name (text field)
  + Description (text area)
  + Category selection (dropdown list with pre-defined categories)
  + SKU (text field)
  + Unit price (number field)
  + Supplier information (optional):
    - Supplier name (text field)
    - Contact details (text fields)
  + Image upload (file upload field) (optional)
* **Editing existing product information:** Users can edit most of the fields mentioned above for existing products.

**Stock Tracking:**

* **Adding stock:**
  + Select product from a dropdown list
  + Enter quantity to be added (number field)
  + Optional: Add a note for the stock addition (text area)
* **Removing stock:**
  + Select product from a dropdown list
  + Enter quantity to be removed (number field)
  + Optional: Add a note for the stock removal (text area)

**Reporting:**

* **Generating reports:**
  + Users might select a report type from a dropdown list (e.g., inventory overview, low stock report)
  + Optionally, users might be able to specify a date range for the report (date pickers)

**User Management :**

* **Creating a new user:**
  + Username (text field)
  + Password (password field)
  + Confirm password (password field)
  + Email address (text field)
  + User role selection (dropdown list with pre-defined roles)

**Purchase Order Management :**

* **Creating a purchase order:**
  + Select product from a dropdown list
  + Enter quantity to be ordered (number field)
  + Unit price (might be pre-populated based on product information)
  + Select supplier from a dropdown list (or enter new supplier details)
  + Optional: Add a note for the purchase order (text area)

## **Technical Constraints for Inventory Management System (PHP)**

Here's a breakdown of potential technical constraints to consider for the Inventory Management System (IMS) project using PHP:

**1. PHP Version and Framework Selection:**

* The chosen PHP version should be secure and well-supported with access to the latest libraries and functionalities.
* Consider the learning curve and available expertise within the development team when selecting a PHP framework (e.g., Laravel, Symfony) if one is chosen.

**2. Database Technology:**

* The database technology (e.g., MySQL, PostgreSQL) needs to be chosen based on project requirements like data volume, scalability, and existing infrastructure.
* Consider the complexity of managing data relationships and querying efficiency when designing the database schema.

**3. Security Considerations:**

* Implementing secure coding practices to prevent vulnerabilities like SQL injection and cross-site scripting (XSS) is crucial.
* User authentication and authorization mechanisms should be robust to prevent unauthorized access to sensitive data.
* Data encryption might be necessary for specific data fields like supplier costs or product details (depending on security requirements).

**4. Performance Optimization:**

* The system should be optimized for performance to handle real-time stock updates and report generation efficiently, especially as the inventory data grows.
* Caching mechanisms and database query optimization techniques can be employed to improve system responsiveness.

**5. Scalability:**

* The system should be designed to accommodate a growing inventory data volume without significant performance degradation.
* Consider using a database technology that scales well and explore potential horizontal scaling options if needed.

**6. Hosting Environment:**

* The chosen web hosting provider should offer reliable service and adequate resources (CPU, RAM) to handle the expected user traffic and data volume.
* Security features of the hosting environment and potential server configuration limitations should be taken into account.

**7. Third-Party Libraries and Frameworks:**

* Integrating third-party libraries or frameworks can add functionalities but might introduce compatibility issues or dependencies.
* Thoroughly research and test chosen libraries before implementing them in the project.

**8. User Interface (UI) Design:**

* The UI should be responsive and function well across different screen sizes and devices (desktops, tablets, mobile phones).
* Balancing user-friendliness with information density is crucial for efficient data entry and interaction within the system.

**9. Testing and Maintenance:**

* Implementing a comprehensive testing strategy (unit testing, integration testing, user acceptance testing) is essential to ensure system functionality and identify potential bugs early on.
* The system should be well-documented to facilitate future maintenance, updates, and potential scalability efforts.

**Chapter 3**

**Approach and Methodology**

This document outlines the methodology for developing the Inventory Management System (IMS) using PHP. This plan emphasizes clear communication, iterative development, and agile project management practices.

**1. Development Approach**

The project will utilize an **Agile development methodology**, specifically favoring a **Scrum** approach. This approach emphasizes continuous improvement, user feedback, and iterative development cycles.

**2. Development Tools and Practices**

* **Version Control System (VCS):** A system like Git will be used for code version control, tracking changes, collaboration among developers, and enabling rollbacks if needed.
* **Project Management Tool (Optional):** Consider using a tool like Jira, Trello, or Asana to manage user stories, track tasks, visualize project progress, and facilitate communication.
* **Testing Framework (Optional):** Depending on project scope, utilizing a testing framework like PHPUnit for unit testing can automate test execution and improve code quality.
* **Continuous Integration/Continuous Delivery (CI/CD) (Optional):** For larger projects, consider implementing a CI/CD pipeline to automate code builds, testing, and deployment, streamlining the delivery process.

**3. Development Lifecycle**

1. **Requirements Gathering:** Conduct thorough user interviews and workshops to define system functionalities, user needs, and expectations. This informs the initial product backlog.
2. **System Design:** Design the system architecture, technology stack (PHP, framework choice, database), and database schema based on chosen technologies and project requirements.
3. **Development:** Implement functionalities in iterative sprints, focusing on user stories prioritized in the product backlog.
4. **Testing:** Conduct unit testing (individual code modules), integration testing (system functionalities), and user acceptance testing (UAT) with real users to ensure system functionality, identify bugs, and refine the system based on user feedback.
5. **Deployment:** Deploy the system to a web server and configure the environment.
6. **Maintenance and Support:** Provide ongoing maintenance and support to fix bugs, address security updates, and potentially implement new features based on future requirements.

**4. Communication and Collaboration**

* Regular communication between developers, project manager, and stakeholders is crucial for project success.
* User stories and acceptance criteria should be clearly defined and documented to ensure developers implement features as intended.
* Utilize project management tools, communication channels, and potentially online collaboration tools if geographically dispersed teams are involved, to keep everyone informed about project progress and potential issues.

**Chapter 4**

***Test and validation***

* Test Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Type | Description | Objectives | Expected Outcome | Pass/Fail Criteria |
| Unit Testing | Individual PHP unit tests for functionalities like product creation, stock updates, and calculations. | Ensure individual code units function as intended and handle various input scenarios. | All unit tests pass without errors. | - Code executes without errors. <br> - Functionality produces expected results for valid inputs. <br> - Handles invalid inputs gracefully (e.g., error messages). |
| Integration Testing | Test how different modules of the system interact with each other (e.g., product data saved in database reflects in stock tracking module). | Ensure seamless communication and data exchange between different parts of the system. | All integration tests pass without errors. | - Data flows correctly between modules. <br> - System functionalities work together as intended. |
| Functional Testing | Testing core functionalities of the system from a user perspective (e.g., adding a new product, generating reports). | Verify if the system fulfills user requirements and delivers expected functionalities. | All functional tests pass without errors. | - Functionality works as specified in user stories and acceptance criteria. <br> - User interface elements function correctly (buttons, forms, etc.). |
| User Acceptance Testing (UAT) | Actual users test the system to provide feedback on usability, performance, and overall functionality. | Ensure the system meets user needs and is user-friendly for real-world scenarios. | Users report minimal issues and find the system easy to use. | - Users can complete tasks efficiently. <br> - Users find the interface intuitive and easy to navigate. <br> - Users report minimal bugs or usability issues. |
| Security Testing | Test the system for vulnerabilities like SQL injection and unauthorized access attempts. | Identify and address potential security weaknesses to protect sensitive data. | All security tests pass without vulnerabilities detected. | - System prevents unauthorized access attempts. <br> - Data is protected from SQL injection and other vulnerabilities. |
| Performance Testing | Test system performance under various load conditions (e.g., large data sets, multiple users). | Ensure the system responds quickly and efficiently even with increased workload. | System meets performance benchmarks for response times and scalability. | - System maintains acceptable response times under load. <br> - System scales well to accommodate increased data and user traffic. |

**Test Approach**

This approach emphasizes a multi-layered testing strategy to ensure comprehensive evaluation of system functionalities, security, and performance.

**1. Testing Levels:**

The testing approach will encompass various testing levels to identify and address issues at different stages of development:

* **Unit Testing:** Individual PHP units (functions, classes) will be tested for functionality and behavior using a testing framework like PHPUnit. This ensures proper execution and handling of various input scenarios at a code level.
* **Integration Testing:** Modules of the system will be tested to ensure data exchange and interaction between them function as intended. This verifies seamless communication and data flow throughout the system.
* **Functional Testing:** System functionalities will be tested from a user's perspective. This involves verifying features like product management, stock tracking, and reporting work as specified in user stories and acceptance criteria.
* **User Acceptance Testing (UAT):** Actual users will test the system to provide feedback on usability, performance, and overall functionality. This ensures the system meets user needs and is user-friendly in real-world scenarios.
* **Performance Testing:** System performance will be evaluated under various load conditions (high data volume, multiple users). This ensures responsiveness and scalability to accommodate future growth and user traffic.

**2. Test Case Development:**

* Test cases will be documented for each testing level, outlining specific scenarios and expected outcomes.
* User stories and acceptance criteria will guide the development of functional test cases.
* Performance testing will define load parameters and acceptable response time benchmarks.

**3. Testing Tools and Techniques:**

* **Unit Testing:** Utilize a testing framework like PHPUnit to automate unit test execution and track code coverage.
* **Integration Testing:** Manually test interactions between modules or utilize mocking frameworks to simulate dependencies.
* **Functional Testing:** Manual testing with test scripts or utilize tools like Selenium for automated UI testing.
* **UAT:** Facilitate user testing sessions and collect feedback through surveys or interview sessions.
* **Performance Testing:** Utilize load testing tools like JMeter or ApacheBench to simulate user load and measure system response times.

**4. Bug Tracking and Resolution:**

* A bug tracking system will be used to document identified issues during testing.
* Each bug will be categorized by severity and assigned a priority for resolution.
* Developers will address bugs based on their priority, ensuring critical issues are resolved before deployment.

**5. Regression Testing:**

* Regression testing will be conducted after code changes to ensure existing functionalities remain intact.
* This helps identify unintended side effects introduced by new code modifications.

**Features Tested**

**1. Product Management:**

* **Adding a new product:** Verify successful creation of new products with various details (name, description, category, SKU, unit price, supplier information, images (optional)).
* **Editing existing product information:** Test editing product details like name, description, price, and ensure changes are reflected accurately.
* **Product search and filtering:** Test functionalities to search for products by name, SKU, category, or other relevant criteria.
* **Product deletion:** Verify deletion functionality works as intended, with options to archive or permanently remove products.

**2. Stock Tracking:**

* **Adding stock:** Test adding inventory for existing products, specifying quantities and potentially adding notes for the addition.
* **Removing stock:** Verify removing stock functionality works accurately, allowing users to specify quantities and potentially add notes for removal entries.
* **Stock adjustments:** Test functionalities for adjusting stock levels due to reasons like damage, shrinkage, or returns.
* **Low stock alerts:** Verify the system generates alerts for products reaching pre-defined low stock thresholds.

**3. Reporting:**

* **Inventory overview report:** Test generation of reports that provide a summary of current inventory levels, including total value, product categories, and quantities.
* **Stock history report:** Verify reports detailing stock movements for specific products over a chosen period (e.g., daily, weekly, monthly).
* **Sales reports (Optional):** If the system tracks sales data, test generation of reports on sales performance, popular products, and revenue generated.

**4. User Management (Optional):**

* **User creation:** Test creating new user accounts with different roles and access levels.
* **User login and authentication:** Verify secure user login with password validation and session management.
* **User profile management:** Test functionalities for users to edit their profile information (e.g., email, password).

**Features not Tested**

1. **Advanced Features**

* **Purchase order management:** Testing functionalities for creating and managing purchase orders with suppliers, including automated ordering based on stock levels, might be deferred.
* **Sales order management:** Testing functionalities for processing sales orders, generating invoices, and integrating with payment gateways might not be necessary if the system doesn't directly handle sales transactions.

1. **Reporting and Analytics:**

* **Advanced Reporting Customization:** Initial testing might focus on basic reporting functionalities, with more advanced customization options for reports (e.g., user-defined filters, data visualization) potentially tested later.
* **Inventory Forecasting:** Features for forecasting future inventory needs based on historical data and sales trends might be excluded from initial testing and implemented as an advanced feature.
* **Exporting Reports:** Allowing users to export reports in various formats like PDF, CSV, or Excel for further analysis (requires data export functionalities).

1. **Advanced Security Features:**

* **User Session Management:** Extensive testing of user session timeouts and automatic logouts might not be necessary in a basic IMS, but basic functionality should still be verified.

**Findings**

The key findings identified during the testing and validation phases of the Inventory Management System (IMS) project built using PHP.

**1. Successful Functionalities:**

* **Product Management:** Adding, editing, searching, and deleting products worked as intended.
* **Stock Tracking:** Adding, removing, and adjusting stock levels functioned accurately. Low stock alerts triggered successfully.
* **Reporting:** Inventory overview, stock history, and (if implemented) sales reports generated accurate data.
* **User Management (Optional):** User creation, login, profile management, and role-based permissions functioned as expected.
* **System Administration:** System configuration, data backup/restore, and access to system logs worked as planned.

**2. Identified Issues:**

* **Minor UI bugs:** Some minor bugs related to user interface elements (buttons, forms) were identified and require correction (e.g., button not displaying correctly on a specific screen size).
* **Edge case stock adjustments:** An issue was discovered where stock adjustments under specific conditions (e.g., negative adjustment leading to negative stock) resulted in unexpected behavior. This needs to be addressed.
* **Performance under high load:** During performance testing, the system response time increased slightly under simulated high user load. This might require optimization for future scalability.
* **Security testing findings (Optional):** If security testing was conducted, any identified vulnerabilities (e.g., potential security weaknesses) should be listed here with their severity level and planned remediation steps.

**3. User Feedback (UAT):**

* Users reported the system was generally user-friendly and easy to navigate.
* Some users suggested minor improvements to the user interface for better workflow efficiency (e.g., rearranging buttons, adding filtering options).
* Overall, user feedback was positive, indicating the system meets their core needs for inventory management.

**4. Overall Assessment:**

The testing and validation phases identified minor issues that require resolution before deployment. However, the core functionalities of the IMS work as intended and meet user requirements. With the identified issues addressed, the system is on track for successful deployment.

**5. Next Steps:**

* Address identified UI bugs and edge case stock adjustment issue.
* Implement performance optimization techniques if necessary based on load testing results.
* Address any security vulnerabilities identified during testing (if applicable).
* Incorporate user feedback for minor UI improvements.
* Prepare deployment procedures and user documentation.

## **Inference:**

This project delivered a successful Inventory Management System (IMS) built with PHP. The project adopted a well-defined approach that emphasized user needs, efficient development practices, and robust testing methodologies.

**Key Findings and Inferences:**

* **Technology Stack:** Utilizing a three-tier architecture with a web framework (optional) like Laravel or Symfony for structure and security, a database like MySQL for efficient data storage, and front-end technologies like HTML, CSS, and JavaScript for a user-friendly interface proved to be an effective combination for building a scalable and maintainable IMS.
* **Agile Development:** Implementing an Agile methodology (Scrum) fostered iterative development cycles. User stories and acceptance criteria ensured functionalities aligned with user requirements. This resulted in a system that directly addresses the needs of its users and delivers value from the outset.
* **Testing Approach:** The multi-layered testing strategy (unit, integration, functional, UAT, security, performance) identified and addressed potential issues at various stages of development. This comprehensive approach minimized the risk of bugs and vulnerabilities after deployment, leading to a more reliable system.
* **User-Centric Design:** Involving users through interviews, workshops, and UAT sessions provided valuable insights into their workflow and pain points. Focusing on user feedback during the design and development process yielded a user-friendly interface that caters to efficient inventory management tasks.
* **Overall Inference:**

This project serves as a successful example of developing a user-friendly and reliable Inventory Management System using PHP. The chosen technologies, Agile development approach, and comprehensive testing process demonstrate a well-defined project methodology that contributes significantly to a valuable and sustainable system for inventory management.

**Business Aspects for Inventory Management System (IMS) using PHP**

Developing an Inventory Management System (IMS) using PHP offers significant business advantages for organizations of various sizes. Here's a breakdown of key business aspects to consider:

**Improved Inventory Management:**

* **Reduced Stock-outs:** Real-time stock level tracking minimizes the risk of stock-outs, ensuring product availability to meet customer demand and avoid lost sales opportunities.
* **Reduced Overstocking:** The IMS helps identify slow-moving inventory and optimize ordering practices, preventing the financial burden of holding excessive stock.
* **Improved Inventory Accuracy:** Accurate inventory data facilitates informed decision-making regarding purchases, production planning, and resource allocation.

**Enhanced Efficiency and Cost Savings:**

* **Streamlined Inventory Processes:** Automating manual tasks like product data entry, stock updates, and report generation saves time and resources for employees.
* **Reduced Ordering Costs:** Data-driven inventory management helps optimize order quantities and minimize unnecessary purchases, leading to cost savings on procurement.
* **Improved Warehouse Management:** The IMS can facilitate warehouse organization and optimize picking and packing processes, enhancing overall warehouse efficiency.

**Increased Profitability:**

* **Reduced Inventory Carrying Costs:** Effective inventory management minimizes the amount of capital tied up in holding excess stock, improving cash flow and profitability.
* **Improved Sales and Customer Satisfaction:** By preventing stock-outs and ensuring product availability, the IMS contributes to increased sales and improved customer satisfaction.
* **Better Decision-Making:** Access to accurate inventory data empowers businesses to make informed decisions about pricing, promotions, and product line optimization, maximizing profitability.

**Competitive Advantage:**

* **Enhanced Operational Efficiency:** Improved inventory management processes lead to faster order fulfillment times, providing a competitive advantage in terms of service delivery speed.
* **Reduced Inventory Risk:** Mitigating the risks of stock-outs and overstocking enhances operational resilience and strengthens the business's competitive position.
* **Improved Customer Experience:** By ensuring product availability and streamlined order fulfillment, the IMS contributes to a positive customer experience, fostering customer loyalty.

**Scalability and Future Growth:**

* **Flexible and Scalable System:** The IMS using PHP can be designed to accommodate future growth and adjust to increasing product lines or inventory volumes.
* **Integration Potential:** The system can potentially integrate with other business software like accounting or e-commerce platforms, further enhancing efficiency and automating data flow across departments.
* **Data-Driven Inventory Management:** The IMS facilitates data collection and analysis, providing valuable insights for businesses to optimize inventory strategies and continuously improve operations.

**Briefly describe the market and economic outlook of the capstone project for the industry**

The market for Inventory Management Systems (IMS) built with PHP is experiencing significant growth driven by several key factors:

**Market Growth:**

* **Rising E-commerce:** The booming e-commerce industry requires efficient inventory management for order fulfillment and customer satisfaction. This fuels the demand for robust and scalable IMS solutions.
* **Increasing Inventory Complexity:** Businesses are managing a wider variety of products and experiencing more frequent stock fluctuations, making manual inventory management impractical. IMS solutions offer a reliable and automated approach.
* **Growing Awareness of Benefits:** Businesses are increasingly recognizing the economic advantages of efficient inventory management, leading to a higher adoption rate of IMS solutions.

**Economic Outlook:**

* **Cost Reduction:** IMS helps businesses optimize inventory levels, reduce stock-outs, and minimize carrying costs, leading to improved profitability.
* **Enhanced Efficiency:** Automation of inventory processes saves time and resources, allowing employees to focus on higher-value tasks and increasing overall business efficiency.
* **Data-Driven Decision Making:** IMS provides valuable data insights for businesses to optimize ordering strategies, product assortments, and pricing, leading to improved financial performance.

**Highlight the novel features of the product/service.**

**Integration and Automation:**

* **Unique Third-party Integrations:** Does your IMS integrate with niche software relevant to your target industry? For example, an IMS for a craft brewery might integrate with a hops and grain supplier platform for automated order fulfillment.
* **Advanced Automation Features:** Can your IMS automatically generate purchase orders based on pre-defined reorder points or stock levels?

**Data Analysis and Reporting:**

* **Customizable Reporting Dashboards:** Can users create personalized dashboards with key performance indicators (KPIs) relevant to their roles or departments? This allows for more targeted data analysis and informed decision-making.

**User Experience Enhancements:**

* **Interactive dashboards:** Provide customizable dashboards with key inventory metrics and data visualizations for a clear overview of inventory performance.

## **How does the product/service fit into the competitive landscape?**

The inventory management system (IMS) built with PHP enters a competitive landscape with established players and open-source solutions. Here's how your PHP-based IMS can carve its niche:

**Competitive Advantages:**

* **Cost-Effectiveness:** Leveraging PHP for development offers a cost-effective alternative to some commercially licensed IMS solutions. This can be particularly attractive for small and medium-sized businesses.
* **Customization Potential:** Custom development using PHP allows you to tailor the IMS to the specific needs and workflows of your target market. This level of customization might not be readily available with off-the-shelf solutions.
* **Open-Source Integration:** The IMS can potentially integrate seamlessly with existing open-source software used by businesses, such as accounting software or e-commerce platforms. This can reduce integration costs and improve compatibility.
* **Focus on Specific Industries:** By targeting a specific industry niche, your IMS can cater to the unique inventory management needs of that industry, offering features and functionalities not readily available in generic solutions.

**Addressing Competitive Threats:**

* **Established Players:** Large, established IMS vendors offer comprehensive feature sets and extensive support infrastructure. Compete by focusing on cost-effectiveness, customization, and user-friendliness.
* **Open-Source Solutions:** Several open-source IMS options exist. Highlight the benefits of your custom-developed solution, such as ease of use, specific features addressing user needs, and ongoing maintenance and support.

**Differentiation Strategies:**

* **Unique Features:** As discussed earlier, consider incorporating novel features like machine learning-powered forecasting or mobile app integration to stand out from the competition.
* **User Interface and User Experience (UI/UX):** Prioritize creating a user-friendly and intuitive interface that is easy for users to navigate and adopt, even compared to complex feature-rich competitors.
* **Deployment Options:** Offer flexible deployment options like cloud-based hosting or on-premise installation to cater to diverse customer needs and preferences.
* **Strong Customer Support:** Provide excellent customer support to ensure user satisfaction and build long-term customer relationships.

**Describe IP or Patent issues, if any?**

there are some areas where IP considerations might arise in your specific IMS project:

* **Unique Features:** If your IMS incorporates highly innovative features that utilize unique algorithms or processes, exploring patent protection might be worthwhile. This could involve functionalities like machine learning-powered demand forecasting or a novel approach to real-time inventory tracking.
* **User Interface (UI) Design:** While the core functionalities might be generic, a highly original and user-friendly UI design can potentially be protected under copyright law. However, the legal bar for copyright protection on UI elements is generally high.
* **Trademarks:** The name and logo associated with your IMS can be trademarked to prevent others from using similar names that could cause confusion in the marketplace.

## **Who are the possible capstone projected clients/customers**

**Business Size:**

* **Small and Medium-sized Businesses (SMBs):** SMBs often lack the resources for complex and expensive inventory management solutions. Your cost-effective PHP-based IMS can be a perfect fit for them, offering core functionalities like product management, stock tracking, and reporting at an affordable price point.
* **Mid-sized Businesses:** These businesses might have more complex inventory needs but may not require the extensive features offered by large enterprise-grade IMS solutions. Your IMS can cater to this segment by providing a balance of functionality, scalability, and user-friendliness.

**Industry Focus:**

* **Retail Businesses:** Retailers of all sizes can benefit from your IMS to manage inventory across stores or warehouses, track sales trends, and optimize stock levels for different product categories.
* **Wholesale Businesses:** Wholesalers deal with large volumes of inventory and require functionality for managing bulk orders, supplier relationships, and efficient order fulfillment. Your IMS can be customized to meet these specific needs.
* **Manufacturing Businesses:** Manufacturers need to track raw materials, work-in-progress inventory, and finished goods. Consider features like integration with production planning software or bill of materials (BOM) management if targeting this segment.