A logo of a globe and a graduation cap

Description automatically generated

**Digital Egypt Pioneers Initiative**

**.Net Back-End Development**

# **Koala Inventory Management System**

**SUPERVISED BY**

**Eng. Fatma Gaber Tolpa**

**PREPARED BY**

|  |  |
| --- | --- |
| **Muhammed Salama Muhammed** | **Mohamed Hazem Mohamed** |
| **Arsany Nashat** | **Mostafa Alaa Eldeen** |
| **Ahmed Tarek** | **Omar Ayman** |

Acknowledgement

It is a pleasure to acknowledge many people who knowingly and unwittingly helped us, to complete our project. First of all let us thank God for all the blessings, which carried us through all these years.

We express our gratitude to

**ENG. Fatema Gaber Tolpa**

for here cooperation and encouragement during the completion of this project.

Also we want to thank here as our project guide who has always stood by our side and guided, appreciated and encouraged us to get into more and more ventures. Continuing the same, she enlightened us in the various stages during the development of this project and provided us with many insights and useful examples, which proved to be of immense help in successful completion of this project.

We extend our deep gratitude to our supervisors who made unforgettable contributions. We also thank the management of the DEPI initiative, which contributed to providing this opportunity for us to develop our skills in the field of Information Technology and web development.

Abstract

Inventory management is a fundamental challenge faced by nearly every organization. In the pre-digital era, companies relied heavily on paper-based tables and manual documentation to manage their inventories. These methods proved to be inefficient, consuming substantial amounts of time and requiring dedicated personnel solely for this aspect of operations. During this time, many companies lacked effective solutions for inventory management, leading to high error rates due to reliance on manual processes. The absence of efficient logging systems made it nearly impossible to track inventory losses, and processes were often plagued by inefficiencies.

With the advent of the computer age, organizations began transitioning from paper-based methods to electronic systems. This shift allowed for the integration of various processes into a digital environment, revolutionizing the way businesses manage their inventories. The introduction of software-based solutions has enabled organizations to maintain greater control over their inventory with reduced effort and fewer employees. These advancements not only streamline operations but also enhance accuracy and traceability in inventory management.

In light of these developments, this documentation presents a effective designed and implemented Inventory Management System (IMS) ,This system leverages modern technology to address the challenges of traditional inventory management practices. By integrating efficient tracking and logging functionalities, the IMS aims to minimize human error and improve overall operational efficiency. The implementation of this system marks a significant step towards optimizing inventory management processes, providing a robust solution that meets the evolving needs of the warehouses.

Contents

[**Koala Inventory Management System** 1](#_Toc180077765)

[Introduction 6](#_Toc180077766)

[1.1 Introduction to project 7](#_Toc180077767)

[1.2 Objective 7](#_Toc180077768)

[1.2.1 Inefficiency: 7](#_Toc180077769)

[1.2.2 Stockouts and Overstocks: 7](#_Toc180077770)

[1.2.3 Inaccurate Data: 7](#_Toc180077771)

[1.2.4 Poor Visibility: 7](#_Toc180077772)

[1.2.5 Time-Consuming Reporting: 8](#_Toc180077773)

[1.3 Technologies Used 8](#_Toc180077774)

[1.4 Target Audience 8](#_Toc180077775)

[System requirement and analysis 9](#_Toc180077776)

[2.1 Introduction 9](#_Toc180077777)

[2.2 Analysis model 9](#_Toc180077778)

[2.3 Study of the system 10](#_Toc180077779)

[2.3.1 GUI’S 10](#_Toc180077780)

[2.3.2 Number of modules 11](#_Toc180077781)

[2.3.3 Entities Involved in the Project: 12](#_Toc180077782)

[2.4 Hardware specifications 12](#_Toc180077783)

[2.4.1 Hardware requirements: 12](#_Toc180077784)

[2.4.2 Software requirements: 13](#_Toc180077785)

[2.5 Software requirement specification 13](#_Toc180077786)

[2.5.2 Functional requirements: 13](#_Toc180077787)

[3 . System design 14](#_Toc180077788)

[3.1 Introduction 15](#_Toc180077789)

[3.2 E – R Diagrams 15](#_Toc180077790)

[3.3 Database Design 17](#_Toc180077791)

[3.4.3 Data dictionary 17](#_Toc180077792)

[4 . Used software 20](#_Toc180077793)

[4.1 Introduction to .net framework 20](#_Toc180077794)

[4.2 Features of the common language runtime 22](#_Toc180077795)

[4.3 .Net framework class library 23](#_Toc180077796)

[4.4 Client application development 24](#_Toc180077797)

[4.5 ASP.NET Core 25](#_Toc180077798)

[4.5.1 Server application development 25](#_Toc180077799)

[4.5.2 Server-side managed code 25](#_Toc180077800)

[4.5.3 Active server pages.net 27](#_Toc180077801)

[4.5.4 Language support 29](#_Toc180077802)

[4.6 Introduction to asp.net server controls 30](#_Toc180077805)

[4.7 C#.NET 31](#_Toc180077806)

[4.7.1 Entity Framework Core overview 31](#_Toc180077807)

[4.7.2 DbSet: 34](#_Toc180077808)

[4.8 SQL SERVER 37](#_Toc180077809)

[4.8.1 Sql server tables 37](#_Toc180077810)

[4.8.2 Primary key 37](#_Toc180077811)

[4.8.3 Relational database 37](#_Toc180077812)

[4.8.4 Foreign key 38](#_Toc180077813)

[4.8.5 Referential integrity 38](#_Toc180077814)

[4.8.6 Data abstraction 38](#_Toc180077815)

[4.8.7 Advantages of RDBMS 38](#_Toc180077816)

[4.8.8 Disadvantages of DBMS 39](#_Toc180077817)

[4.8.9 Features of SQL server (RDBMS) 39](#_Toc180077818)

[4.8.10 Sophisticated concurrency control 41](#_Toc180077819)

[5. Project Overview 42](#_Toc180077820)

[5.1. Sidebar 42](#_Toc180077821)

[5.2. Home Page 45](#_Toc180077822)

[5.3 Dashboard Overview 50](#_Toc180077823)

[5.3.2 Warehouse Navigation Buttons 50](#_Toc180077824)

[5.3.3 Low Stock Products Overview 52](#_Toc180077825)

[5.3.4 Average Stock by Category 54](#_Toc180077826)

[5.3.5 Average Stock by Category 55](#_Toc180077827)

[5.3.6 Profits By Category 56](#_Toc180077828)

[5.4 Inventory Page 58](#_Toc180077829)

[5.4.5 Product Editing and Deletion 61](#_Toc180077830)

[5.5 Suppliers and Categories Page 62](#_Toc180077831)

[5.6 Sales Page 65](#_Toc180077832)

[5.7 Admin Board 66](#_Toc180077833)

[5.8 Account Management 68](#_Toc180077834)

[6 . System testing 71](#_Toc180077835)

[6.1 Introduction 71](#_Toc180077836)

[6.2 Strategic approach to software testing 71](#_Toc180077837)

[6.3 The levels of testing 72](#_Toc180077838)

[6.3.1 Unit Testing 74](#_Toc180077839)

[6.4 Methodology 76](#_Toc180077840)

[6.4.1 Pros of using the methodology 77](#_Toc180077841)

[6.4.2 Cons of using the methodology 77](#_Toc180077842)

[6.5 Interface Testing 77](#_Toc180077843)

[6.5.1 Functional requirement 77](#_Toc180077844)

[6.5.2 Test Cases 79](#_Toc180077845)

[6.6 Results 84](#_Toc180077846)

[7 . System Security 86](#_Toc180077847)

[7.1 Introduction 87](#_Toc180077848)

[7.2 Security in software 88](#_Toc180077849)

[7.2.1 Client side validation 88](#_Toc180077850)

[7.2.2 Server side validation 88](#_Toc180077851)

[8 . Conclusion and future work 91](#_Toc180077852)

[8.1 Conclusion 91](#_Toc180077853)

[8.2 Benefits: 92](#_Toc180077854)

[8.3 Future work: 93](#_Toc180077855)

[9 References 96](#_Toc180077856)

Chapter1

# Introduction

## 1.1 Introduction to project

This inventory management system is designed to streamline the tracking and management of inventory levels, orders, sales, and deliveries. It offers a comprehensive solution for real-time tracking of stock levels, locations, and movements, making it an essential tool for businesses that deal with large quantities of products. Key features include order management, automated stock replenishment alerts, reporting and analytics, multi-location support, and integration capabilities with other systems like accounting, e-commerce, and supply chain management.

## Objective

The system addresses several key challenges faced by warehouses:

### Inefficiency:

* Manual inventory management is time-consuming and error-prone. By automating processes, the system significantly improves operational efficiency.

### 1.2.2 Stockouts and Overstocks:

* Maintaining optimal stock levels is a constant struggle for businesses. The system provides real-time data and alerts to help prevent stockouts or overstocks, which can lead to lost sales or increased carrying costs

### 1.2.3 Inaccurate Data:

* Inconsistent or outdated inventory records often lead to errors in order fulfillment. This system ensures accurate, up-to-date information, minimizing discrepancies.

### 1.2.4 Poor Visibility:

* Managing inventory across multiple locations can be challenging without a centralized system. The solution offers comprehensive visibility, regardless of location.

### 1.2.5 Time-Consuming Reporting:

* Manually generating reports takes considerable time. The system automates reporting, providing quick access to insights and data analysis.

## 1.3 Technologies Used

* The project is built using ASP.NET Core 8 and follows the Repository Design Pattern. Other technologies used include SQL Server for database management, AJAX for asynchronous communication, and jQuery for enhancing the user interface.

## 1.4 Target Audience

The primary users of this system are warehouse operators who need to efficiently manage inventory across various locations.

Chapter 2

# System requirement and analysis

## 2.1 Introduction

After analyzing the requirements of the task to be performed, the next step is to analyze the problem and understand its context. The first activity in the phase is studying the existing system and other is to understand the requirements and domain of the new system. Both the activities are equally important, but the first activity serves as a basis of giving the functional specifications and then successful design of the proposed system.

Understanding the properties and requirements of a new system is more difficult and requires creative thinking and understanding of existing running system is also difficult, improper understanding of present system can lead diversion from solution.

## 2.2 Analysis model

The model being followed is the AGILE MODEL, which is structured in an iterative and incremental manner. Unlike traditional models like Waterfall, Agile promotes flexibility and rapid delivery through continuous feedback and collaboration. The process is broken down into short, manageable iterations called sprints, each focusing on delivering functional parts of the system.

The development cycle begins with requirement gathering, but instead of defining all requirements upfront, Agile encourages ongoing collaboration with stakeholders to adapt to changing needs. The key phases in an Agile project are:

* **Sprint Planning & Requirement Gathering:** At the start of each sprint, the team collaborates with stakeholders to prioritize features and define the goals of the sprint.
* **Design & Development:** Development and design occur simultaneously. During the sprint, the team continuously refines designs and writes code to build working features incrementally.
* **Testing:** Testing is not a separate phase, as it occurs continuously throughout the sprint. Testers collaborate with developers to perform unit tests, integration tests, and end-user testing to ensure functionality aligns with requirements.
* **Review & Retrospective:** At the end of each sprint, a review meeting is held to present the completed work to stakeholders. Feedback is gathered and incorporated into future sprints. A retrospective is also conducted where the team discusses what went well, what didn’t, and how processes can be improved.
* **Iteration**: After each sprint, based on feedback and lessons learned, the team adjusts its course for the next sprint. This iterative approach allows for flexibility and adaptation to changing project requirements or priorities.

Agile was chosen because it allows us to rapidly respond to feedback and deliver functional features in short cycles, making it ideal for projects where requirements may evolve over time or are not fully known upfront. Additionally, Agile fosters collaboration and ensures that the system being developed stays aligned with business goals through continuous feedback.

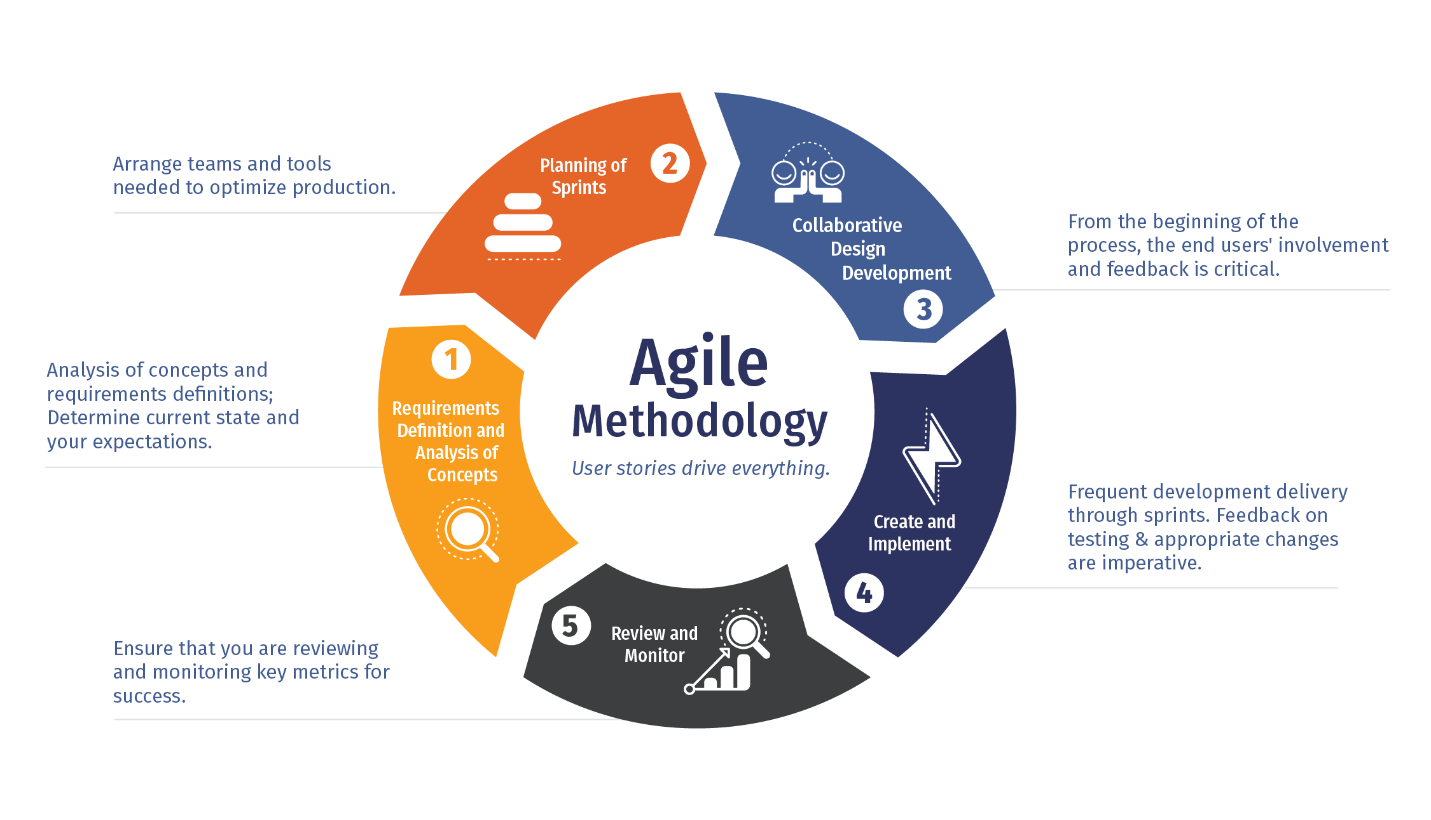


Fig 2.1

## 2.3 Study of the system

### 2.3.1 GUI’S

In the flexibility of the uses the interface has been developed a graphics concept in mind, associated through a browses interface. The GUI'S at the top level have been categorized as 1. Administrative user interface 2. The operational or generic user interface

The administrative user interface concentrates on the consistent information that is practically, part of the organizational activities and which needs proper authentication for the data collection.

The interfaces help the administrations with all the transactional states like Data insertion, Data deletion and Date updating along with the extensive data search capabilities.

The operational or generic user interface helps the users upon the system in transactions through the existing data and required services. The operational user interface also helps the ordinary users in managing their own information helps the ordinary users in managing their own information in a customized manner as per the assisted flexibilities.

### 2.3.2 Number of modules

The system after careful analysis has been identified to be presented with the following modules:

This project is divided into 7 modules:

1. Registration Module
2. Category Module
3. Supplier Module
4. Products Browse Module
5. Products Search Module
6. Warehouse Module
7. Admin User Management Module
8. Admin Warehouse Management Module

### 2.3.3 Entities Involved in the Project:

1. Products
2. Suppliers
3. Categories
4. Orders
5. System Users and Rules
6. Warehouses

* **Products**: Contains all relevant information about the products in the inventory, such as name, quantity, and specifications.
* **Suppliers**: Holds details of suppliers providing the products.
* **Categories**: Organizes products into different categories for easier management and reporting.
* **Orders**: Tracks purchase and sales orders, including details on product quantities and transaction dates.
* **Users**: Stores user information, including roles and access levels, as the system uses role-based authentication.
* **Warehouses**: Manages information about the various locations where inventory is stored.

## 2.4 Hardware specifications

### 2.4.1 Hardware requirements:

* 2.8 GHz Processor and Above
* RAM 1024MB and Above
* HDD 20 GB Hard Disk Space and Above

### 2.4.2 Software requirements:

* WINDOWS OS (XP / 8 / 7 / 2003, 2008, 2012 Server etc.)
* Visual Studio .Net 2022 Community Edition
* Internet Information Server (IIS)
* Visual Studio .Net Core (Minimal for Deployment)
* SQL Server 2019 Enterprise Edition

## 2.5 Software requirement specification

The software, Site Explorer is designed for management of system from a remote location.

### 2.5.1 Functional requirements:

##### 2.5.1.1 Error avoidance

At this stage care is to be taken to ensure that input data remains accurate form the stage at which it is recorded up to the stage in which the data is accepted by the system. This can be achieved only by means of careful control each time the data is handled.

##### 2.5.1.2 Error detection

Even though every effort is make to avoid the occurrence of errors, still a small proportion of errors is always likely to occur, these types of errors can be discovered by using validations to check the input data.

##### 2.5.1.3 Data validation

Procedures are designed to detect errors in data at a lower level of detail. Data validations have been included in the system in almost every area where there is a possibility for the user to commit errors. The system will not accept invalid data. Whenever an invalid data is keyed in, the system immediately prompts the user and the user has to again key in the data and the system will accept the data only if the data is correct. Validations have been included where necessary.

The system is designed to be a user friendly one. In other words the system has been designed to communicate effectively with the user. The system has been designed with popup menus.

Chapter 3

# 3 . System design

## 3.1 Introduction

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm and area of application. Design is the first step in the development phase for any engineered product or system. The designer’s goal is to produce a model or representation of an entity that will later be built. Beginning, once system requirement have been specified and analyzed, system design is the first of the three technical activities design, code and test that is required to build and verify software.

During design, progressive refinement of data structure, program structure, and procedural details are developed reviewed and documented. System design can be viewed from either technical or project management perspective. From the technical point of view, design is comprised of four activities architectural design, data structure design, interface design and procedural design.

## 3.2 E – R Diagrams

The relation upon the system is structure through a conceptual ER-Diagram, which not only specifics the existential entities but also the standard relations through which the system exists and the cardinalities that are necessary for the system state to continue.

The entity Relationship Diagram (ERD) depicts the relationship between the data objects. The ERD is the notation that is used to conduct the date modeling activity the attributes of each data object noted is the ERD can be described resign a data object descriptions.

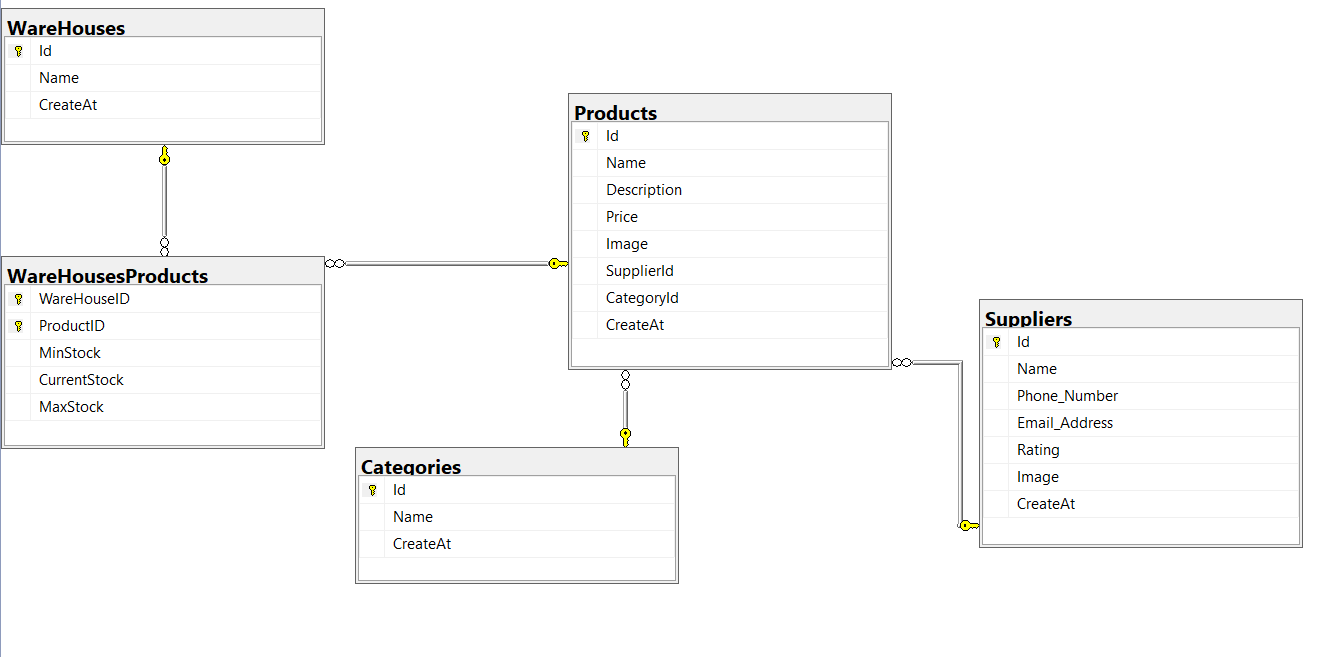
The set of primary components that are identified by the ERD are:

* Data object
* Relationships
* Attributes
* Various types of indicators.

The primary purpose of the ERD is to represent data objects and their relationships.

ERD Design

Fig 3.1



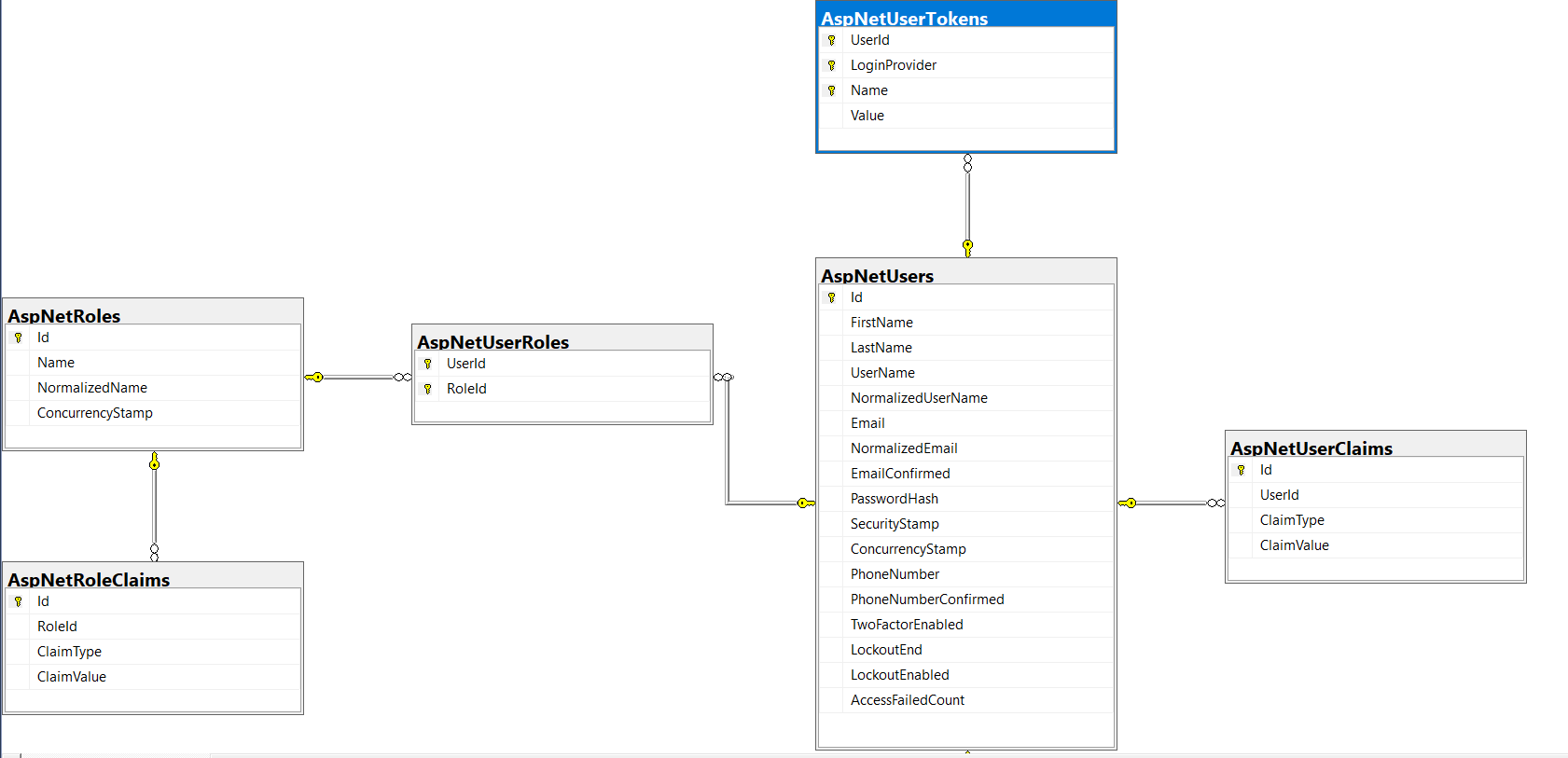


Fig 3.2

## 3.3 Database Design

Database Design

The database is designed with several key tables, each serving a crucial role in managing inventory and operations:

• Products: Contains all relevant information about the products in the inventory, such as name, quantity, and specifications.

• Suppliers: Holds details of suppliers providing the products.

• Categories: Organizes products into different categories for easier management and reporting.

• Orders: Tracks purchase and sales orders, including details on product quantities and transaction dates.

• Users: Stores user information, including roles and access levels, as the system uses role-based authentication.

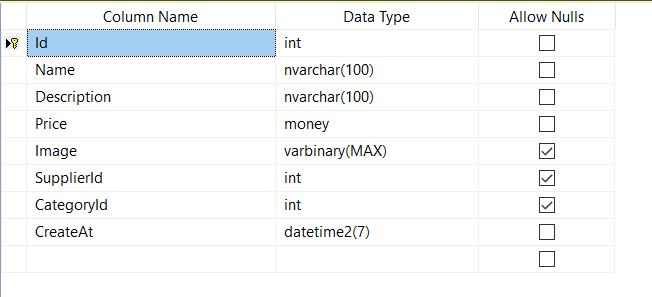
• Warehouses: Manages information about the various locations where inventory is stored.

The system includes a many-to-many relationship between products and warehouses. This relationship is represented through a junction table called warehouseProducts, which links products to multiple warehouses and vice versa. Additionally, there are one-to-many relationships between products and suppliers (a product can have one supplier, but a supplier can provide many products) and between products and categories (a product belongs to one category, but a category can contain many products).

### 3.4 Data dictionary

After carefully understanding the requirements of the system, the entire data storage requirements are divided into tables. The below are the database tables:

**Table: Products**



**Table: Suppliers**

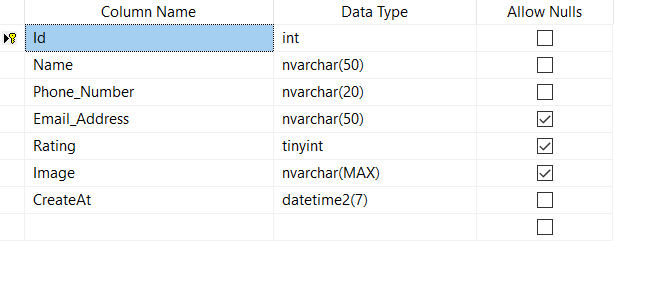


Table: Categories

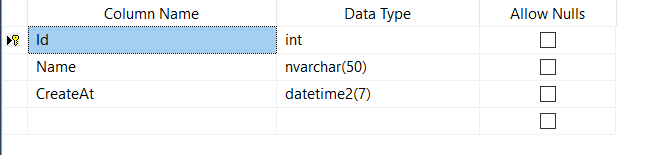


Table: Warehouses

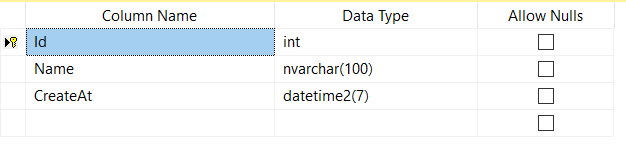


Table: WarehousePorduct

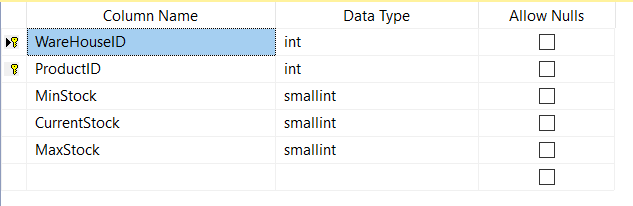


Table: users

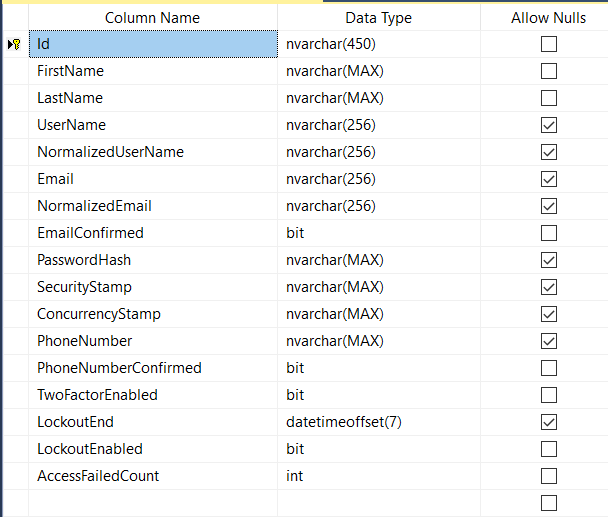
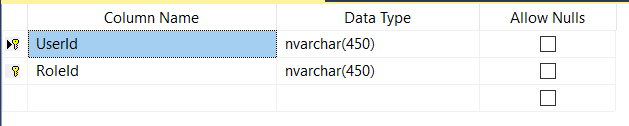


Table: User Rules



Chapter 4

# 4 . Used software

## 4.1 Introduction to .NET Core 8

.NET Core 8 is a modern, open-source, cross-platform framework that enables developers to build high-performance applications across various environments, including the web, cloud, and IoT. .NET Core 8 is designed to achieve the following objectives:

- To offer a unified development experience that runs seamlessly on Windows, Linux, and macOS, providing flexibility in cloud-native and containerized environments.

- To deliver a modular framework that supports lightweight application deployment, reducing conflicts and minimizing footprint.

- To provide a secure and reliable code execution environment, ensuring safe operation for applications, even when interacting with untrusted third-party code.

- To maximize performance by optimizing runtime execution, surpassing the limitations of interpreted or virtualized environments.

- To streamline the development process for various types of applications, from web APIs to microservices, with a unified toolset and architecture.

- To embrace industry standards and open protocols, ensuring that .NET Core 8 applications can integrate easily with a diverse range of technologies.

.NET Core 8 has two fundamental components: the runtime and the extensive class library. The runtime is at the heart of .NET Core, managing critical tasks such as memory allocation, threading, and execution of managed code. It ensures type safety, enforces security, and optimizes code performance. Managed code refers to code executed under the control of the .NET runtime, while unmanaged code runs outside this environment. The class library provides a robust set of reusable types, enabling developers to build a wide range of applications, from console apps to sophisticated web services using ASP.NET Core.

.NET Core 8 can also be hosted by external components that integrate the runtime within their processes, creating a flexible development ecosystem that benefits from both managed and unmanaged capabilities. Hosting scenarios include support for third-party runtimes, ensuring seamless extensibility.

For instance, ASP.NET Core uses the .NET Core runtime to offer a scalable, high-performance environment for web applications and APIs. It works closely with the runtime to enable fast, secure web development, supporting innovations like Blazor and microservices architecture.

Another example is the use of browser-based applications that can leverage the .NET Core runtime through technologies like WebAssembly, enabling managed code to run in the browser with enhanced security and isolation. This approach allows for robust web application development with the safety and performance benefits of managed code.

The following illustration demonstrates how .NET Core 8 runtime and class library interact with your applications, showcasing how managed code fits into the broader system architecture.

## 4.2 Features of the common language runtime

The Common Language Runtime (CLR) in .NET Core is responsible for managing memory, executing threads, handling code execution, and ensuring code safety and compilation. These features are fundamental to managed code that runs on the CLR.

In terms of security, the CLR assigns different levels of trust to managed components based on their origin, such as from the Internet, a network, or a local machine. This ensures that managed components may have restricted access to sensitive operations like file access or registry modifications, even if used in the same application.

The CLR enforces code access security, so users can trust that a web-based executable may display animations or play media without compromising their personal data or file system. This security model allows Internet-based software to be feature-rich while maintaining safety.

The CLR also guarantees code robustness through its strict type- and code-verification system known as the Common Type System (CTS). The CTS ensures that all managed code is self-describing and follows consistent type rules, enabling different languages and compilers to interact seamlessly and safely.

The managed environment of the CLR eliminates common software issues like memory leaks and invalid memory references. It automatically handles memory allocation, manages object lifecycles, and releases memory when objects are no longer in use, providing automatic memory management.

CLR boosts developer productivity by supporting multiple programming languages, allowing developers to write applications in their preferred language while leveraging the power of the runtime and class libraries. Language compilers targeting the CLR make it easy to migrate existing code and take full advantage of .NET Core's capabilities.

Additionally, the CLR supports backward compatibility through interoperability between managed and unmanaged code, allowing developers to use existing COM components and DLLs within their managed applications.

The CLR is optimized for performance, offering just-in-time (JIT) compilation, which converts managed code to native machine code for execution. This eliminates the need for code interpretation, ensuring efficient execution. The CLR's memory manager reduces memory fragmentation and improves locality of reference to further enhance performance.

Finally, the CLR can be hosted by high-performance server applications, such as Microsoft SQL Server and Internet Information Services (IIS). This allows developers to write business logic using managed code while benefiting from the performance of enterprise-grade servers that support CLR hosting.

## 4.3 .Net Core class library

Client application development in .NET Core is designed to provide a robust, flexible, and cross-platform solution for building modern desktop and web applications. The framework ensures the following key objectives:

- To offer a unified development model for creating interactive user interfaces across desktop and web environments, including Windows, macOS, Linux, and mobile platforms.

- To enable high-performance, responsive applications with optimized support for asynchronous programming and efficient resource management.

- To promote the use of open standards and modern web technologies such as HTML5, CSS, JavaScript, and WebAssembly.

- To ensure seamless integration with backend services and APIs, allowing client applications to easily communicate with cloud or on-premises systems.

- To provide a wide range of tools and libraries that simplify the development of rich user interfaces and complex workflows.

- To support modern deployment models, including containerization and cloud hosting, enabling client applications to be delivered efficiently at scale.

Client application development in .NET Core is facilitated through technologies like Blazor, Windows Forms, WPF, and Xamarin. Blazor, a key part of ASP.NET Core, allows developers to build rich web applications using C# instead of JavaScript, leveraging WebAssembly for near-native performance. Windows Forms and WPF provide powerful frameworks for building desktop applications with intuitive graphical interfaces. Xamarin extends client application development to mobile platforms, enabling the creation of cross-platform mobile apps using a shared .NET codebase.

With .NET Core, developers can create client applications that run across platforms, ensuring a seamless user experience and leveraging shared business logic and services. The client-side components integrate closely with server-side technologies, enabling full-stack development within a single ecosystem.

By leveraging .NET Core for client applications, developers benefit from a consistent development experience, enhanced productivity, and the ability to build scalable, modern applications that meet the needs of today's users.

## 4.4 Client application development

Client application development in .NET Core focuses on building high-performance, cross-platform applications for desktop, web, and mobile environments. The framework is designed to meet the following objectives:

- To enable developers to create interactive, feature-rich applications that run seamlessly on multiple platforms, including Windows, macOS, Linux, iOS, and Android.

- To support both desktop and web applications with modern, responsive user interfaces using technologies like Windows Presentation Foundation (WPF), Windows Forms, and Blazor.

- To facilitate the integration of client-side applications with backend services, APIs, and cloud resources, ensuring connectivity and data synchronization across platforms.

- To leverage modern web standards like WebAssembly in Blazor for building client-side applications using C#, avoiding the need for JavaScript in certain scenarios.

- To promote efficient memory management and performance optimization, resulting in responsive and scalable client applications.

- To offer tools and libraries for creating complex user interfaces, handling asynchronous operations, and managing application state effectively.

Client application development in .NET Core empowers developers to use frameworks like WPF and Windows Forms for desktop applications, delivering powerful graphical interfaces and native Windows experiences. For web-based client applications, ASP.NET Core Blazor allows C# developers to create interactive web apps that run on the client browser through WebAssembly or on the server. For cross-platform mobile development, Xamarin provides the ability to share code across iOS, Android, and other platforms, allowing for the reuse of business logic and a consistent user experience.

.NET Core simplifies the client application development process by providing a unified development environment that integrates with Visual Studio, making it easier to manage projects and libraries across different platforms. Developers can build, test, and deploy client applications with flexibility, whether targeting desktop, web, or mobile users.

## 4.5 ASP.NET Core

### 4.5.1 Server application development

ASP.NET server application development in .NET Core is a modern, versatile framework for building high-performance, scalable web and cloud-based applications. It focuses on achieving the following objectives:

- To provide a unified framework for developing server-side applications, enabling full control over HTML markup, HTTP requests, and server-side logic.

- To support cross-platform development, allowing applications to run on Windows, Linux, and macOS, ensuring broad deployment flexibility.

- To integrate with cloud services, making it easier to build cloud-native applications that leverage scalability, high availability, and distributed systems.

- To offer advanced security mechanisms like authentication, authorization, and secure data transfer, protecting sensitive information and preventing common web vulnerabilities.

- To promote modularity and extensibility by using middleware components to handle requests and responses, making the pipeline flexible and configurable.

- To provide a productive developer experience with tools like Visual Studio, making it easier to create, debug, and deploy server-side applications efficiently.

ASP.NET Core supports several models for server-side application development, including MVC (Model-View-Controller), Razor Pages, and Web APIs. The MVC pattern is ideal for complex applications requiring separation of concerns between user interfaces, data models, and business logic, while Razor Pages offer a simpler page-based model for lightweight applications. Web APIs enable developers to create RESTful services that interact with client applications and other systems, facilitating data exchange in distributed environments.

The framework ensures high performance through features like asynchronous programming, connection pooling, and caching, making it ideal for handling large-scale, concurrent web traffic. ASP.NET Core also leverages dependency injection, a key feature that improves code maintainability, testability, and flexibility. With built-in support for modern web standards, such as WebSockets and SignalR, developers can create real-time, interactive web applications with ease.

ASP.NET server application development provides robust tools for building and deploying server-side applications, whether for the web, APIs, or cloud-based solutions. With its cross-platform capabilities and cloud integration, developers can create scalable, secure, and high-performing applications for a wide range of use cases.

### 4.5.2 Server-side managed code

ASP.NET is the hosting environment that enables developers to use the .NET Framework to target Web-based applications. However, ASP.NET is more than just a runtime host; it is a complete architecture for developing Web sites and Internet-distributed objects using managed code. Both Web Forms and XML Web services use IIS and ASP.NET as the publishing mechanism for applications, and both have a collection of supporting classes in the .NET Framework.

XML Web services, an important evolution in Web-based technology, are distributed, server-side application components similar to common Web sites. However, unlike Web-based applications, XML Web services components have no UI and are not targeted for browsers such as Internet Explorer and Netscape Navigator. Instead, XML Web services consist of reusable software components designed to be consumed by other applications, such as traditional client applications, Web-based applications, or even other XML Web services. As a result, XML Web services technology is rapidly moving application development and deployment into the highly distributed environment of the Internet.

If you have used earlier versions of ASP technology, you will immediately notice the improvements that ASP.NET and Web Forms offers. For example, you can develop Web Forms pages in any language that supports the .NET Framework. In addition, your code no longer needs to share the same file with your HTTP text (although it can continue to do so if you prefer). Web Forms pages execute in native machine language because, like any other managed application, they take full advantage of the runtime. In contrast, unmanaged ASP pages are always scripted and interpreted. ASP.NET pages are faster, more functional, and easier to develop than unmanaged ASP pages because they interact with the runtime like any managed application.

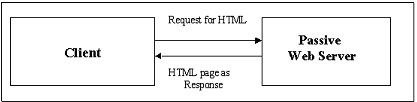
The .NET Framework also provides a collection of classes and tools to aid in development and consumption of XML Web services applications. XML Web services are built on standards such as SOAP (a remote procedure-call protocol), XML (an extensible data format), and WSDL ( the Web Services Description Language). The .NET Framework is built on these standards to promote interoperability with non-Microsoft solutions.

For example, the Web Services Description Language tool included with the .NET Framework SDK can query an XML Web service published on the Web, parse its WSDL description, and produce C# or Visual Basic source code that your application can use to become a client of the XML Web service. The source code can create classes derived from classes in the class library that handle all the underlying communication using SOAP and XML parsing. Although you can use the class library to consume XML Web services directly, the Web Services Description Language tool and the other tools contained in the SDK facilitate your development efforts with the .NET Framework.

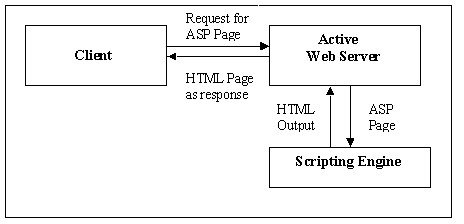
If you develop and publish your own XML Web service, the .NET Framework provides a set of classes that conform to all the underlying communication standards, such as SOAP, WSDL, and XML. Using those classes enables you to focus on the logic of your service, without concerning yourself with the communications infrastructure required by distributed software development.

Finally, like Web Forms pages in the managed environment, your XML Web service will run with the speed of native machine language using the scalable communication of IIS.

**A Passive Web Server stores static Web Pages**



**An active Web Server generates dynamic Web Pages**



### 4.5.3 Active server pages.net

ASP.NET is a programming framework built on the common language runtime that can be used on a server to build powerful Web applications. ASP.NET offers several important advantages over previous Web development models:

**Enhanced Performance.** ASP.NET is compiled common language runtime code running on the server. Unlike its interpreted predecessors, ASP.NET can take advantage of early binding, just-in-time compilation, native optimization, and caching services right out of the box. This amounts to dramatically better performance before you ever write a line of code.

**World-Class Tool Support.** The ASP.NET framework is complemented by a rich toolbox and designer in the Visual Studio integrated development environment. WYSIWYG editing, drag-and-drop server controls, and automatic deployment are just a few of the features this powerful tool provides.

**Power and Flexibility**. Because ASP.NET is based on the common language runtime, the power and flexibility of that entire platform is available to Web application developers. The .NET Framework class library, Messaging, and Data Access solutions are all seamlessly accessible from the Web. ASP.NET is also language-independent, so you can choose the language that best applies to your application or partition your application across many languages. Further, common language runtime interoperability guarantees that your existing investment in COM-based development is preserved when migrating to ASP.NET.

**Simplicity**. ASP.NET makes it easy to perform common tasks, from simple form submission and client authentication to deployment and site configuration. For example, the ASP.NET page framework allows you to build user interfaces that cleanly separate application logic from presentation code and to handle events in a simple, Visual Basic - like forms processing model. Additionally, the common language runtime simplifies development, with managed code services such as automatic reference counting and garbage collection.

**Manageability.** ASP.NET employs a text-based, hierarchical configuration system, which simplifies applying settings to your server environment and Web applications. Because configuration information is stored as plain text, new settings may be applied without the aid of local administration tools. This "zero local administration" philosophy extends to deploying ASP.NET Framework applications as well. An ASP.NET Framework application is deployed to a server simply by copying the necessary files to the server. No server restart is required, even to deploy or replace running compiled code.

**Scalability and Availability**. ASP.NET has been designed with scalability in mind, with features specifically tailored to improve performance in clustered and multiprocessor environments. Further, processes are closely monitored and managed by the ASP.NET runtime, so that if one misbehaves (leaks, deadlocks), a new process can be created in its place, which helps keep your application constantly available to handle requests.

**Customizability and Extensibility**. ASP.NET delivers a well-factored architecture that allows developers to "plug-in" their code at the appropriate level. In fact, it is possible to extend or replace any subcomponent of the ASP.NET runtime with your own custom-written component. Implementing custom authentication or state services has never been easier.

**Security**. With built in Windows authentication and per-application configuration, you can be assured that your applications are secure.

### 4.5.4 Language support

The Microsoft .NET Platform currently offers built-in support for three languages: C#, Visual Basic, and F#.

## 4.6 Introduction to asp.net server controls

In addition to (or instead of) using <% %> code blocks to program dynamic content, ASP.NET page developers can use ASP.NET server controls to program Web pages. Server controls are declared within an .aspx file using custom tags or intrinsic HTML tags that contain a runat="server" attributes value. Intrinsic

HTML tags are handled by one of the controls in the

**System.Web.UI.HtmlControls** namespace. Any tag that doesn't explicitly map to one of the controls is assigned the type of

**System.Web.UI.HtmlControls.HtmlGenericControl**.

Server controls automatically maintain any client-entered values between round trips to the server. This control state is not stored on the server (it is instead stored within an **<input type="hidden">** form field that is round-tripped between requests). Note also that no client-side script is required.

In addition to supporting standard HTML input controls, ASP.NET enables developers to utilize richer custom controls on their pages. For example, the following sample demonstrates how the **<asp:adrotator>** control can be used to dynamically display rotating ads on a page.

1. ASP.NET Web Forms provide an easy and powerful way to build dynamic Web UI.
2. ASP.NET Web Forms pages can target any browser client (there are no script library or cookie requirements).
3. ASP.NET Web Forms pages provide syntax compatibility with existing ASP pages.
4. ASP.NET server controls provide an easy way to encapsulate common functionality.
5. ASP.NET ships with 45 built-in server controls. Developers can also use controls built by third parties.
6. ASP.NET server controls can automatically project both up level and down level HTML.
7. ASP.NET templates provide an easy way to customize the look and feel of list server controls.
8. ASP.NET validation controls provide an easy way to do declarative client or server data validation.

## 4.7 C#.NET

### 4.7.1 Entity Framework Core

Entity Framework Core (EF Core) is a modern ORM (Object-Relational Mapping) framework that simplifies data access in .NET applications, allowing developers to work with databases using .NET objects. Designed with performance and flexibility in mind, EF Core supports multiple database providers and includes features for lazy loading, change tracking, and migrations.

EF Core uses the concept of a `DbContext`, which acts as a bridge between the application and the database. Key components of EF Core include `DbSet<T>`, which represents a collection of entities, and `EntityTypeConfiguration`, which allows for configuration of entity properties and relationships.

A significant advantage of EF Core over traditional data access technologies is its ability to work with strongly typed entities that reflect the underlying database schema. This enables developers to query and manipulate data using LINQ (Language Integrated Query), making data access more intuitive. EF Core provides a set of conventions and a fluent API for configuring models, which helps enforce data integrity and relationships among entities.

The `DbContext` serves as a unit of work, allowing developers to perform CRUD (Create, Read, Update, Delete) operations on entities and automatically track changes. This marks a shift from connection-oriented data access, moving towards a model where the application can interact with data in a more abstract and manageable way. EF Core uses `IQueryable` and `IEnumerable` to enable efficient data retrieval and manipulation.

EF Core's change tracking feature automatically monitors changes to entities, allowing for batch updates and saves. This is achieved through a concept called "change tracker," which detects changes and prepares the appropriate SQL commands to persist data back to the database.

While EF Core abstracts the underlying database provider, it also provides an extensible architecture that allows for custom providers. This flexibility enables developers to work with different types of databases seamlessly. The framework includes various methods and tools for managing database migrations, seeding data, and testing the application against different environments.

In the following sections, we will explore the key components of EF Core and how to leverage them for efficient data access in your applications. These components include:

- DbContext: For managing database connections and performing operations on entities.

- DbSet<T>: For representing collections of entities and enabling LINQ queries.

- EntityTypeConfiguration: For configuring entity properties and relationships.

- Change Tracking: For monitoring changes to entities and persisting updates.

- Migrations: For managing schema changes and evolving the database alongside your application.

In this guide, we will primarily use SQL Server as our database provider, leveraging the power of EF Core to interact with data in a seamless and efficient manner.

**4.7.1.1 Connections:**

In Entity Framework Core, connections to databases are managed through the `DbContext`, which encapsulates the database connection and provides access to the database. Instead of using provider-specific classes like `SqlConnection`, EF Core abstracts this with a `DbContext` that is configured to work with a specific database provider, such as SQL Server.

Commands are issued through the `DbSet<T>` collections within the `DbContext`, allowing developers to execute queries and commands using LINQ or raw SQL. The results are returned as strongly typed entities, which can be directly manipulated in the application. EF Core handles the mapping of these entities to database records, streamlining the process of retrieving and saving data.

**4.7.1.2 Commands:**

In Entity Framework Core, commands are represented by LINQ queries or raw SQL executed through the `DbSet<T>` collections within a `DbContext`. Instead of using provider-specific classes like `SqlCommand`, EF Core allows you to define your commands using strongly typed entities. Commands can include operations such as inserting, updating, or deleting records, and they can also call stored procedures or execute raw SQL.

You can utilize input parameters by directly setting properties on your entity objects, and EF Core manages the underlying SQL commands, including any return values or output parameters. The example below demonstrates how to issue an INSERT operation using EF Core with the `DbSet<T>` representing the Northwind database's entities.

using (var context = new NorthwindContext())

{

var newOrder = new Order

{

CustomerId = "ALFKI",

OrderDate = DateTime.Now,

// Set other properties as needed

};

context.Orders.Add(newOrder);

context.SaveChanges(); // This executes the INSERT command

}

### 4.7.2 DbSet:

DbSet and Entity Framework Core

In Entity Framework Core, the `DbSet<T>` class is analogous to the ADO.NET `DataSet`, but it provides a more powerful and intuitive way to work with data. One significant distinction is that `DbSet<T>` operates in a connected manner, allowing developers to interact directly with the database context through strongly typed entities. It represents a collection of entities that map to a database table, including properties for columns, relationships, and constraints.

Unlike a `DataSet`, which is a disconnected model, `DbSet<T>` maintains a connection to the database through the `DbContext`. This enables developers to work with a programming model that is consistent and reflects the current state of the database. Data can come from a variety of sources—whether from a database, an XML file, or user input—and can be manipulated through the entity framework. Changes made to the entities tracked by `DbContext` can be validated and then persisted back to the database by calling `SaveChanges()`.

The `DbContext` serves as the bridge between the application and the database, managing the retrieval and saving of data. In EF Core, when changes are made to the entities, the context tracks those changes. When `SaveChanges()` is called, EF Core automatically generates and executes the necessary SQL commands to update the database. You don’t need to manually manage updates as you would with a `DataAdapter`.

For example, consider the following code that demonstrates how to insert a new record:

using (var context = new NorthwindContext())

{

var newOrder = new Order

{

CustomerId = "ALFKI",

OrderDate = DateTime.Now,

// Set other properties as needed

};

context.Orders.Add(newOrder);

context.SaveChanges(); // This executes the INSERT command

}

EF Core also supports executing raw SQL commands directly against the database if needed, providing flexibility for performing inserts, updates, and deletes without relying on intermediate caching.

Performance Considerations:

In EF Core, you can also optimize performance by using asynchronous methods for data retrieval and updates, which helps to keep the application responsive. Additionally, you can define relationships, constraints, and data annotations on your entity classes to enforce data integrity.

EF Core allows for the generation of migrations to manage database schema changes over time. This feature enables developers to evolve the database schema alongside their application, making it easier to maintain and scale.

Summary

1. DbSet<T>: Represents collections of entities and enables a connected data access model.

2. DbContext: Manages the connection to the database and tracks changes to entities.

3. Entity Framework Core: Designed for n-tier architectures and supports various data sources.

4. Direct Commands: You can execute commands directly without needing intermediate steps, enhancing performance.

5. Change Tracking: Automatically tracks changes made to entities, facilitating easier updates to the database.

In EF Core, the process of connecting to the database and retrieving data is simplified through the use of the `DbContext`, which abstracts away the complexities involved in data access, making development more efficient and intuitive.

## 4.8 SQL SERVER

A database management, or DBMS, gives the user access to their data and helps them transform the data into information. Such database management systems include dBase, paradox, IMS, SQL Server and SQL Server. These systems allow users to create, update and extract information from their database.

A database is a structured collection of data. Data refers to the characteristics of people, things and events. SQL Server stores each data item in its own fields. In SQL Server, the fields relating to a particular person, thing or event are bundled together to form a single complete unit of data, called a record (it can also be referred to as raw or an occurrence). Each record is made up of a number of fields. No two fields in a record can have the same field name.

During an SQL Server Database design project, the analysis of your business needs identifies all the fields or attributes of interest. If your business needs change over time, you define any additional fields or change the definition of existing fields.

### 4.8.1 Sql server tables

SQL Server stores records relating to each other in a table. Different tables are created for the various groups of information. Related tables are grouped together to form a database.

### 4.8.2 Primary key

Every table in SQL Server has a field or a combination of fields that uniquely identifies each record in the table. The Unique identifier is called the Primary Key, or simply the Key. The primary key provides the means to distinguish one record from all other in a table. It allows the user and the database system to identify, locate and refer to one particular record in the database.

### 4.8.3 Relational database

Sometimes all the information of interest to a business operation can be stored in one table. SQL Server makes it very easy to link the data in multiple tables. Matching an employee to the department in which they work is one example. This is what makes SQL Server a relational database management system, or RDBMS. It stores data in two or more tables and enables you to define relationships between the table and enables you to define relationships between the tables.

### 4.8.4 Foreign key

When a field is one table matches the primary key of another field is referred to as a foreign key. A foreign key is a field or a group of fields in one table whose values match those of the primary key of another table.

### 4.8.5 Referential integrity

Not only does SQL Server allow you to link multiple tables, it also maintains consistency between them. Ensuring that the data among related tables is correctly matched is referred to as maintaining referential integrity.

### 4.8.6 Data abstraction

A major purpose of a database system is to provide users with an abstract view of the data. This system hides certain details of how the data is stored and maintained. Data abstraction is divided into three levels.

**Physical level:** This is the lowest level of abstraction at which one describes how the data are actually stored.

**Conceptual Level:** At this level of database abstraction all the attributed and what data are actually stored is described and entries and relationship among them.

**View level:** This is the highest level of abstraction at which one describes only part of the database.

### 4.8.7 Advantages of RDBMS

* Redundancy can be avoided
* Inconsistency can be eliminated
* Data can be Shared
* Standards can be enforced
* Security restrictions ca be applied
* Integrity can be maintained
* Conflicting requirements can be balanced - Data independence can be achieved.

### 4.8.8 Disadvantages of DBMS

A significant disadvantage of the DBMS system is cost. In addition to the cost of purchasing of developing the software, the hardware has to be upgraded to allow for the extensive programs and the workspace required for their execution and storage. While centralization reduces duplication, the lack of duplication requires that the database be adequately backed up so that in case of failure the data can be recovered.

### 4.8.9 Features of SQL server (RDBMS)

SQL SERVER is one of the leading database management systems (DBMS) because it is the only Database that meets the uncompromising requirements of today's most demanding information systems. From complex decision support systems (DSS) to the most rigorous online transaction processing (OLTP) application, even application that require simultaneous DSS and OLTP access to the same critical data, SQL Server leads the industry in both performance and capability

SQL SERVER is a truly portable, distributed, and open DBMS that delivers unmatched performance, continuous operation and support for every database.

SQL SERVER RDBMS is high performance fault tolerant DBMS which is specially designed for online transactions processing and for handling large database application.

SQL SERVER with transactions processing option offers two features which contribute to very high level of transaction processing throughput, which are

**4.8.9.1 Enterprise wide data sharing**

The unrivaled portability and connectivity of the SQL SERVER DBMS enables all the systems in the organization to be linked into a singular, integrated computing resource.

**4.8.9.2 Portability**

SQL SERVER is fully portable to more than 80 distinct hardware and operating systems platforms, including UNIX, MSDOS, OS/2, Macintosh and dozens of proprietary platforms. This portability gives complete freedom to choose the database sever platform that meets the system requirements.

**4.8.9.3 Open systems**

SQL SERVER offers a leading implementation of industry standard SQL. SQL Server's open architecture integrates SQL SERVER and non SQL SERVER DBMS with industries most comprehensive collection of tools, application, and third party software products SQL Server's Open architecture provides transparent access to data from other relational database and even nonrelational database.

**4.8.9.4 Distributed data sharing**

SQL Server’s networking and distributed database capabilities to access data stored on remote server with the same ease as if the information was stored on a single local computer. A single SQL statement can access data at multiple sites. You can store data where system requirements such as performance, security or availability dictate.

**4.8.9.5 Unmatched performance**

The most advanced architecture in the industry allows the SQL SERVER DBMS to deliver unmatched performance.

### 4.8.10 Sophisticated concurrency control

Real World applications demand access to critical data. With most database Systems application becomes contention bound which performance is limited not by the CPU power or by disk I/O, but user waiting on one another for data access . SQL Server employs full, unrestricted row-level locking and contention free queries to minimize and in many cases entirely eliminates contention wait times.

**4.8.10.1 No i/o bottlenecks**

SQL Server’s fast commit groups commit and deferred write technologies dramatically reduce disk I/O bottlenecks. While some database write whole data block to disk at commit time, SQL Server commits transactions with at most sequential log file on disk at commit time, On high throughput systems, one sequential writes typically group commit multiple transactions. Data read by the transaction remains as shared memory so that other transactions may access that data without reading it again from disk. Since fast commits write all data necessary to the recovery to the log file, modified blocks are written back to the database independently of the transaction commit, when written from memory to disk.

**Chapter 5**

# 5. Project Overview

### 5.1. Sidebar

The sidebar in **Koala Inventory Management System** is a collapsible navigation menu that provides access to various features based on user roles. The sidebar layout and available options vary depending on the login status of the user. Below are the descriptions for both logged-in and not-logged-in states, as well as the minimized views.

#### 5.1.1. Sidebar (Logged In)

In the logged-in state, the sidebar provides comprehensive navigation for the system's core features. The user’s profile information is displayed at the bottom, indicating the currently logged-in user.

* **Navigation Icons (Minimized State)**: When the sidebar is minimized, it displays only icons representing each feature for a compact look, enabling more space for content on the main page.
  + **Home**: Icon resembling a house.
  + **Dashboard**: A chart icon for data visualization.
  + **Inventory**: A storage building icon for inventory management.
  + **Suppliers**: A delivery truck icon representing supplier information.
  + **Sales**: A shopping cart icon for the sales module.
  + **Admin Board**: A user group icon for administrative tasks.
  + **Reports**: An attachment icon for accessing reports.
* **Expanded State (Full Menu)**: When expanded, the sidebar shows both icons and text labels, enhancing clarity for users.
  + **Home**: Directs users to the homepage for an overview of the system.
  + **Dashboard**: Provides key insights and metrics regarding inventory, suppliers, and sales.
  + **Inventory**: Accesses the inventory management section for tracking products and stock levels.
  + **Suppliers**: Contains details of suppliers and their relationships with the organization.
  + **Sales**: Tracks sales records, allowing users to monitor transactions.
  + **Admin Board**: Available only to users with administrative privileges, this option allows management of roles and system settings.
  + **Reports**: Allows users to generate and view various reports about system performance.
* **Profile Section**: Located at the bottom, it shows the logged-in user's name and profile picture (or placeholder if no picture is uploaded). Users can log out from this section.

A screenshot of a computer

Description automatically generated

Figure i-sidebar

Figure ii- sidebar minimized

#### 5.1.2. Sidebar (Not Logged In)

When the user is not logged in, the sidebar displays a reduced set of features. This state is typically meant for guest users with limited access.

* **Minimized State**:
  + **Home**: Icon for the homepage.
  + **Inventory**: Limited access to view general inventory data.
* **Expanded State**:
  + **Home**: Directs users to the system homepage.
  + **Inventory**: Allows viewing general inventory data, but with restricted access to detailed management functions.
* A blue and white rectangle

  Description automatically generated**Profile Section**: Instead of the user’s name, the profile section indicates "Not Logged In." Guest users have the option to log in from here to gain full access to the system’s features.

Figure iii-sidebar not logged in

Figure iv-sidebar not logged in minimized



### 5.2. Home Page

The **Home Page** of the Koala Inventory Management System offers an informative and visually engaging introduction to the platform. It provides a high-level overview of the system’s capabilities and key statistics about the business.

#### 5.2.1 Overview Section

The main panel of the home page introduces the system’s core functionality and highlights how Koala helps optimize inventory management. The introduction includes:

* **Welcome Text**: A brief statement outlining the purpose of Koala, emphasizing real-time tracking, analytics, and seamless integration.
* **Key Features**:
  + Streamlined operations to reduce human error and increase efficiency.
  + Real-time insights for quicker decision-making and inventory optimization.
  + Secure data with encryption and compliance with industry standards.
* **App Store Links**: Buttons leading to the system’s mobile applications on Google Play and the App Store (if applicable).

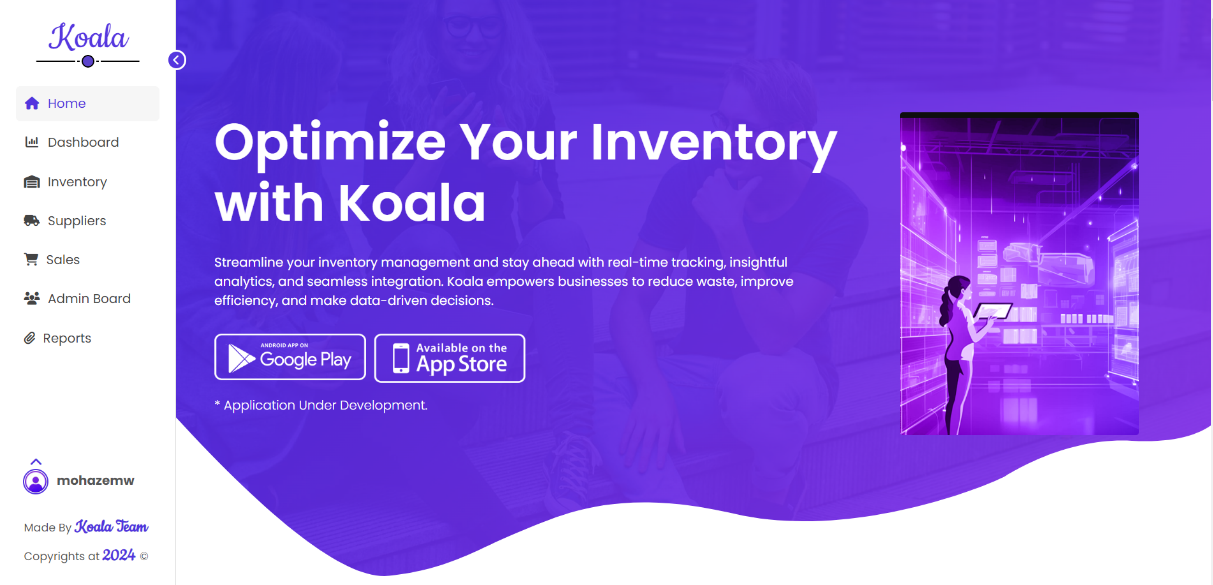


Figure v-home 1

#### 5.2.2 Statistics

The next section displays key business statistics relevant to the user’s inventory and sales data:

* **10K Sales**: Total number of sales made through the system.
* **23K Products**: Count of products available in the inventory.
* **9M Customers**: Number of customers using or impacted by the system.
* **5 Developers**: Information about the development team behind Koala.

This section serves as a quick snapshot of the business’s operational scale and system impact.

A screenshot of a website

Description automatically generated

Figure vi-home 2

#### 5.2.3 Why Choose Koala?

This section explains the unique value propositions of Koala, organized into three key features:

* **Streamlined Operations**: Describes how Koala automates inventory workflows, reducing manual errors and improving efficiency.
* **Real-Time Insights**: Details the importance of gaining immediate access to critical data, helping businesses make faster, more informed decisions.
* **Secure Data**: Emphasizes the system’s robust data protection measures, including encryption and compliance with industry regulations.

A screenshot of a blue and white screen

Description automatically generated

Figure vii-home 3

#### 5.2.4 Customer Testimonials Section

The Customer Testimonials Section is strategically positioned on the Home Page to provide social proof and enhance trust in the Koala Inventory Management System. It includes:

* **Heading**: "What Our Clients Say About Koala"
* **Overview**: This section showcases feedback from satisfied clients, emphasizing the platform's effectiveness in addressing their inventory management needs.
* **Design Elements**: Each testimonial is presented in a visually appealing format, featuring summaries of user experiences and satisfaction levels to build credibility among potential users.

Here’s an overview for the **Pricing Page** and the **FAQ Page** of the Koala Inventory Management System, based on the provided images.

A screenshot of a website

Description automatically generated

Figure viii-home 5

**5.2.5 Pricing Page**

The Pricing Page of the Koala Inventory Management System presents users with flexible subscription plans tailored to meet varying business needs. This page is designed to help users understand the value of the service while offering clear comparisons between different plans.

A screenshot of a website

Description automatically generated

Figure ix-home 6

**5.2.6 Subscription Plans**

This section outlines the various subscription options available, detailing the key features of each plan to assist users in making informed decisions:

* **Basic Plan**:
  + **Price**: $49
  + **Features**:
    - **Storage**: 5GB of data storage for inventory management.
    - **Support**: 24/7 customer support to address user inquiries and issues.
    - **Automated Backups**: Regular backups to ensure data security and availability.
* **Pro Plan**:
  + **Price**: $120
  + **Features**:
    - **Storage**: 10GB of data storage, suitable for larger inventories.
    - **Databases**: Management of up to 50 databases, allowing for extensive inventory tracking.
    - **Unlimited Emails**: Enables users to communicate freely within the system.
    - **Enhanced Backups**: More frequent automated backups for increased data protection.

**5.2.7 Value Proposition**

The Pricing Page emphasizes the value each plan provides, showcasing both the cost-effectiveness and the comprehensive support offered with each subscription. The clear presentation helps potential customers assess which plan aligns best with their inventory management needs, facilitating an easier decision-making process.

**5.2.8 FAQ Page**

The FAQ Page of the Koala Inventory Management System is designed to address common queries from users, providing clarity and support to enhance their experience with the platform.

**Common Questions**

This section provides solutions to frequently asked questions, helping users navigate the system more effectively:

* **What is Koala Inventory Management?**
  + Koala is a comprehensive inventory management system designed to simplify tracking, managing, and analyzing inventory processes to enhance operational efficiency.
* **Why use Koala over other inventory systems?**
  + Koala stands out due to its user-friendly interface, integration capabilities with existing systems, and features that help businesses optimize stock management and reduce operational waste.
* **How secure is my data with Koala?**
  + Data security is paramount for Koala. The platform uses industry-standard encryption protocols to protect user records and secure access controls to safeguard sensitive information.

**Customization and Integration**

This subsection addresses inquiries related to the customization and integration capabilities of Koala:

* **Can I customize features in Koala?**
  + Users can tailor certain features to meet specific business needs, ensuring that the system aligns with individual operational requirements.
* **How can Koala integrate with other systems?**
  + Koala supports integration with various business tools, including ERP systems, allowing users to streamline operations and gather comprehensive insights across platforms.

A screenshot of a computer

Description automatically generated

Figure x-home 6

### 5.3 Dashboard Overview

The Dashboard serves as the central hub for users, providing essential insights and analytics related to inventory management. Accessible to all managerial roles, including managers and admins, the Dashboard is designed to enhance decision-making through visual data representation.

#### 5.3.1 Key Features

* **Products By Category**:
  + This section features a pie chart that visually represents the distribution of products across various categories such as Accessories, Clothing, Electronics, Furniture, and Groceries. This allows users to quickly assess which categories have the most inventory and identify areas for potential growth or reduction.
* **Products By Supplier**:
  + Displayed as a circular chart, this feature highlights the number of products supplied by each vendor. For example, the chart indicates that the supplier "Hazlam" provides 4 products. This visualization aids in evaluating supplier performance and inventory diversification.

A screenshot of a dashboard

Description automatically generated

Figure xi-dashboard-Warehouses overview

### 5.3.2 Warehouse Navigation Buttons

The Warehouse Navigation Buttons are an essential feature of the Dashboard, allowing users to easily access specific warehouse data based on their role within the organization. This ensures that users see only the information relevant to their responsibilities while maintaining security and operational efficiency.

#### Role-Specific Access

* **Dynamic Button Display**: The number of buttons displayed on the Dashboard changes according to the user's role (e.g., manager or admin). This customization prevents unauthorized access to sensitive warehouse data, thereby promoting data security.
* **User Permissions**: Managers may have access to multiple warehouses, while admins might see all available warehouses. This tailored approach ensures that users can focus on the warehouses they manage or oversee without being overwhelmed by irrelevant information.

#### Warehouse Buttons

* **Individual Warehouse Representation**: Each button corresponds to a specific warehouse, labeled accordingly (e.g., "SwiftStore," "Nexus Dist," "PrimeSpace," "AgileHub"). This clear labeling helps users quickly identify the warehouse they wish to view.
* **Interactive Functionality**: Clicking on a warehouse button triggers an update on the Dashboard, displaying detailed information relevant to that particular warehouse. This includes:
  + **Inventory Levels**: Users can see current stock levels, which helps in making quick decisions about restocking or managing inventory turnover.
  + **Product Breakdown**: Users gain insights into the types of products stored within the warehouse, organized by categories such as Accessories, Clothing, Electronics, Furniture, and Groceries.
  + **Supplier Information**: Data on suppliers who provide products to the selected warehouse can also be displayed, allowing users to evaluate supplier performance and relationships.

#### Dynamic Content Display

* **Real-Time Updates**: When a warehouse button is clicked, the main content area of the Dashboard updates dynamically to reflect the selected warehouse's data. This real-time change allows for seamless navigation and a user-friendly experience.
* **Visual Analytics Integration**: The displayed data may include additional visual aids, such as charts and graphs, providing a comprehensive overview of the warehouse’s inventory status and performance metrics.

This feature enhances the usability of the Dashboard, allowing users to drill down into specific areas of interest quickly and efficiently, thereby improving their ability to manage and optimize inventory processes.

A screenshot of a graph

Description automatically generated

Figure xii-dashboard each warehouse details

### 5.3.3 Low Stock Products Overview

This section of the Dashboard displays crucial information regarding low stock products specific to the selected warehouse. It is designed to help users quickly identify items that need replenishment, ensuring that inventory levels are maintained and stock-outs are minimized.

#### Key Features of the Low Stock Products Section

* **Table Format**: The low stock products are presented in a clear and organized table format, making it easy for users to scan and understand the data at a glance. The table includes the following columns:
  + **ID**: A unique identifier for each product, facilitating easy reference and management.
  + **Name**: The name of the product, allowing users to quickly identify which items are running low.
  + **Price**: The cost of each product, which helps in evaluating the financial implications of restocking.
  + **Minimum Stock**: The minimum quantity that should be maintained in inventory to prevent stock-outs. This serves as a crucial benchmark for inventory management.
  + **Current Stock**: The present quantity of the product available in the warehouse. This immediate insight allows users to assess the urgency of replenishing stock.
  + **Optimal Stock**: The ideal quantity recommended for maintaining inventory levels, balancing supply and demand efficiently.

#### Example Data

For instance, in the shown table, the following low stock products are highlighted for the "Nexus Dist" warehouse:

| **ID** | **Name** | **Price** | **Minimum Stock** | **Current Stock** | **Optimal Stock** |
| --- | --- | --- | --- | --- | --- |
| 76 | Reusable Water Bottle | 20 | 13 | 5 | 48 |
| 86 | Fast Charging Charger | 18 | 10 | 3 | 42 |
| 91 | Office Desk Organizer | 15 | 10 | 2 | 30 |

#### Importance of Low Stock Monitoring

* **Proactive Inventory Management**: By highlighting low stock items, the Dashboard enables users to take proactive measures to restock before running out, ensuring that customer demand can be met without interruption.
* **Strategic Planning**: Understanding the current stock levels in relation to minimum and optimal stock levels supports strategic planning and inventory optimization. Users can analyze trends, adjust orders, and manage supplier relationships more effectively.
* **Cost Management**: Monitoring low stock products allows businesses to manage costs associated with urgent restocking, such as expedited shipping fees or missed sales opportunities due to stock-outs.

This section effectively aids warehouse managers and admins in maintaining optimal inventory levels and ensuring a smooth operational flow within the warehouse.

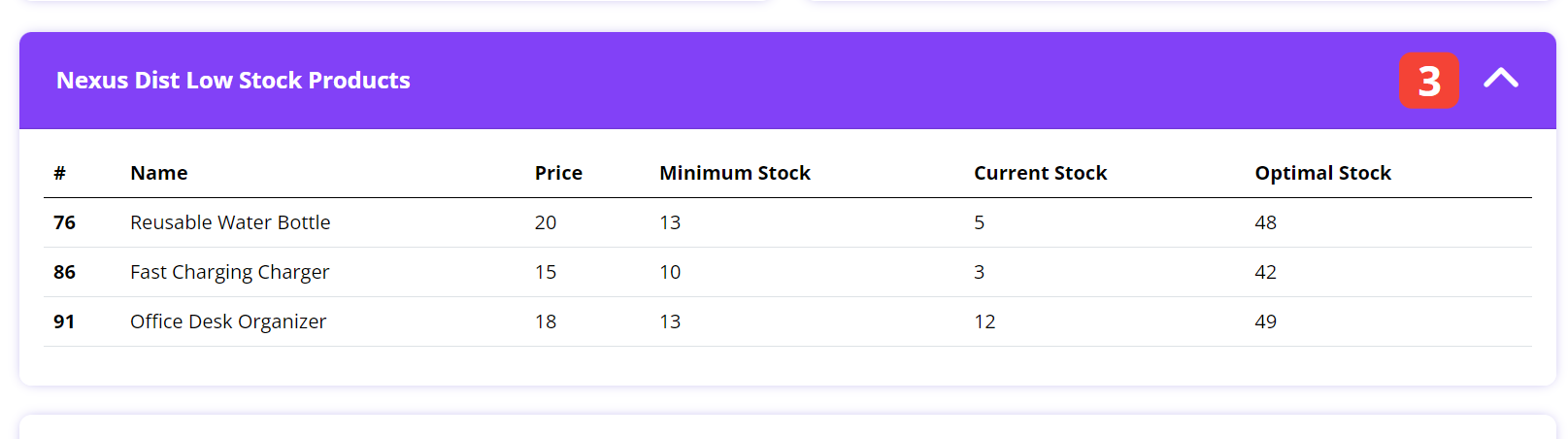


Figure xiii-dashboard low stock products details

### 5.3.4 Average Stock by Category

The **Average Stock by Category** section of the Dashboard provides a visual representation of inventory levels across different product categories. This feature is vital for understanding how stock is distributed and can help inform inventory management strategies.

#### Key Features of the Average Stock by Category Section

* **Bar Chart Visualization**: The data is presented in a bar chart format, which allows users to easily compare average stock levels across various categories. Each category is represented by two bars:
  + **Current Average Stock**: Shown in a darker shade, this bar indicates the average quantity of products currently in stock for that category.
  + **Optimal Average Stock**: Represented in a lighter shade, this bar signifies the ideal average quantity that should be maintained to meet customer demand effectively.

#### Categories Represented

The following categories are typically included in this analysis:

1. **Accessories**: Items that complement main products, such as tech accessories or fashion jewelry.
2. **Clothing**: Apparel items, which can vary widely from casual wear to professional attire.
3. **Electronics**: This category encompasses a range of gadgets and devices, reflecting the demand for tech-related products.
4. **Furniture**: Essential items for homes and offices, including desks, chairs, and storage solutions.
5. **Groceries**: Everyday consumables, which are critical for maintaining household and personal needs.

#### Insights from the Average Stock Analysis

* **Inventory Assessment**: By comparing the current average stock to the optimal average stock, users can quickly identify which categories may be overstocked or understocked. For example:
  + If the current average stock for groceries is significantly below the optimal level, it may indicate a need for restocking to meet customer needs.
  + Conversely, a high current average stock in clothing might suggest an opportunity to run promotions or discounts to clear excess inventory.
* **Strategic Decision-Making**: This visual data aids managers in making informed decisions regarding purchasing, marketing strategies, and inventory allocation. Understanding category performance helps in optimizing stock levels and reducing carrying costs.
* **Trend Analysis**: Over time, tracking average stock levels by category can reveal trends in consumer behavior, helping businesses adjust their inventory strategies to align with demand patterns.

This section enhances the Dashboard's functionality by providing users with essential insights into inventory management, enabling them to take proactive steps to optimize stock levels across all categories.

A screenshot of a computer

Description automatically generated

Figure xiv-low stock products

### 5.3.5 Average Stock by Category

The **Average Stock by Category** section visually displays the average inventory levels across different product categories, providing valuable insights for effective inventory management.

#### Visualization Overview

* **Bar Chart Format**: The data is represented in a bar chart, which includes two bars for each category:
  + **Current Average Stock**: Displayed in a darker shade, indicating the average quantity of products presently in stock.
  + **Optimal Average Stock**: Shown in a lighter shade, representing the ideal quantity that should be maintained to meet customer demand efficiently.

#### Categories Analyzed

1. **Accessories**: Complementary items such as tech gadgets and fashion accessories.
2. **Clothing**: Various apparel categories ranging from casual to formal wear.
3. **Electronics**: Gadgets and devices that are essential for modern living.
4. **Furniture**: Items needed for home and office settings, including tables and chairs.
5. **Groceries**: Daily consumables that are fundamental for households.

#### Insights Gained

* **Inventory Assessment**: Users can assess which categories are overstocked or understocked:
  + **Understocked Categories**: If the current average stock is below the optimal level, it indicates a need for replenishment.
  + **Overstocked Categories**: High current stock levels compared to optimal levels may suggest the need for sales promotions or inventory adjustments.
* **Strategic Planning**: This analysis supports better decision-making regarding purchasing and inventory allocation, allowing businesses to align their stock levels with customer demand.
* **Trend Monitoring**: Over time, tracking these averages helps identify trends, enabling proactive adjustments to inventory strategies.

This section is crucial for warehouse managers and inventory professionals, providing a clear overview of stock levels by category and helping to optimize inventory management practices.

A graph of different colored squares

Description automatically generated with medium confidence

### 5.3.6 Profits By Category

The **Profits By Category** section of the Dashboard provides a dynamic view of profit trends across different product categories over a specified time period. This feature is essential for understanding the financial performance of various segments and informing strategic business decisions.

#### Visualization Overview

* **Line Graph Format**: Profits are represented in a line graph, allowing for easy tracking of profit trends over time. The graph displays multiple lines, each corresponding to a different product category, with the following characteristics:
  + **Layered Design**: The lines may overlap, creating a visually engaging representation of profits across categories, showing how each category's profitability changes over time.

#### Insights Gained

* **Profit Trends**: Users can observe how profits fluctuate over the months, identifying peaks and troughs in profitability for each category.
  + **Seasonal Trends**: Recognizing patterns can help businesses anticipate busy periods, such as holidays or seasonal changes, where certain categories may experience increased demand.
* **Category Performance**: By comparing the profitability of different categories, businesses can determine which segments are performing well and which may need adjustments in strategy.
  + For instance, if electronics consistently show higher profits compared to clothing, resources can be allocated accordingly to capitalize on this trend.
* **Strategic Decision-Making**: This analysis supports informed decision-making regarding product promotions, inventory levels, and marketing strategies based on profit performance across categories.
* **Financial Health Monitoring**: Tracking profits over time allows businesses to assess their financial health, ensuring they are on track to meet revenue goals.

This section is crucial for financial analysts and business managers, providing a comprehensive overview of profitability by category and enabling effective planning and forecasting.

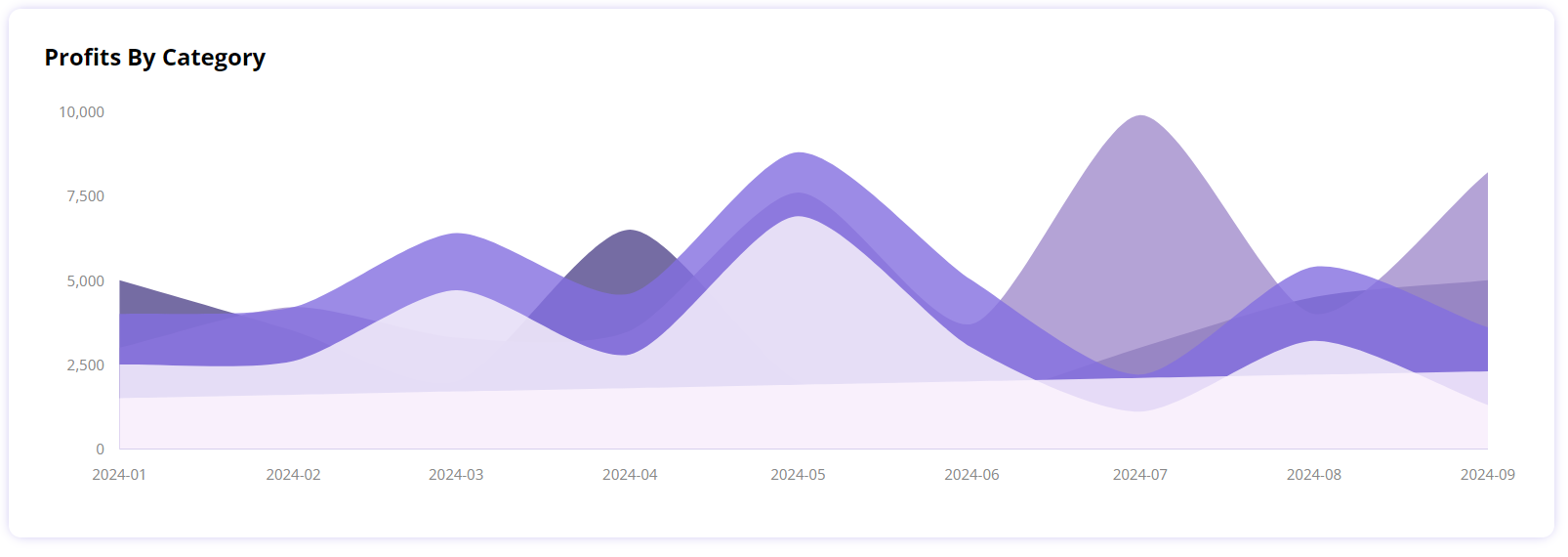
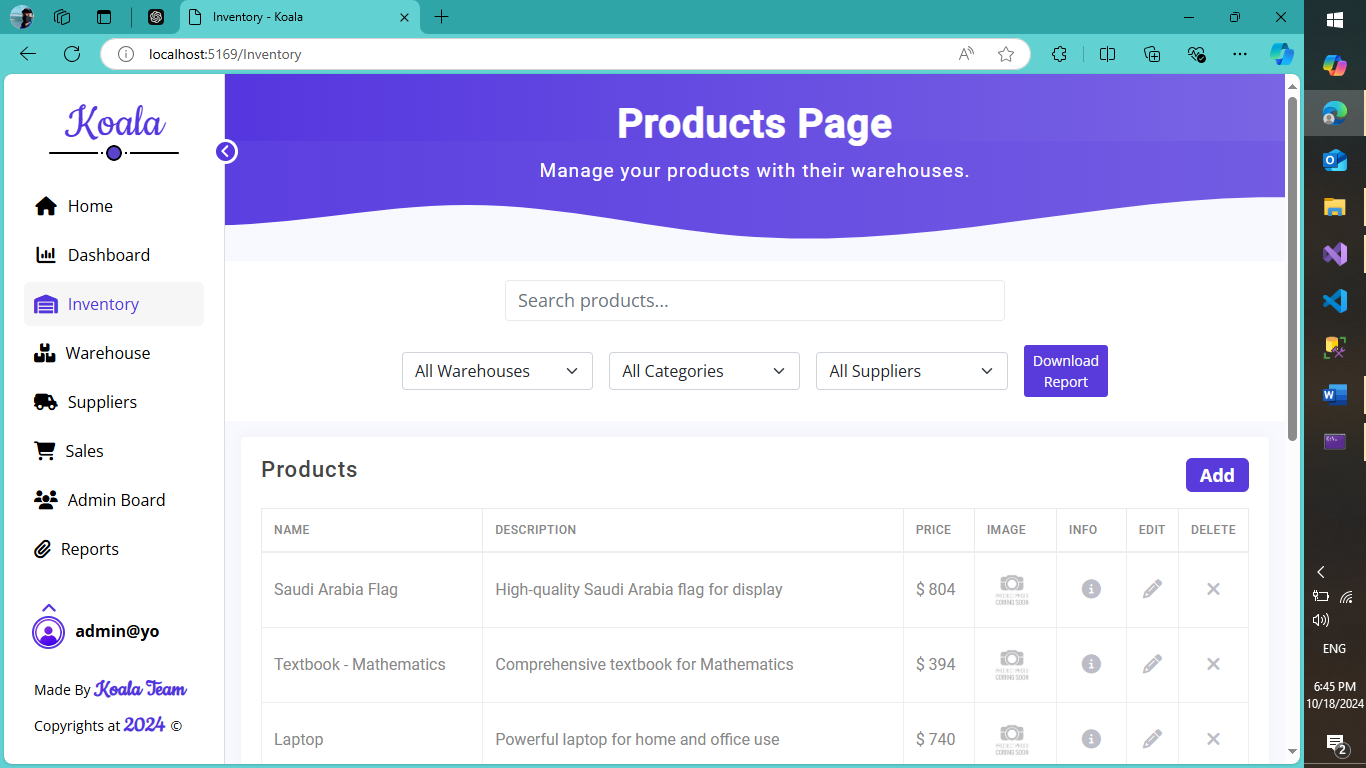


Figure xv-dashboard profits

### 5.4 Inventory Page

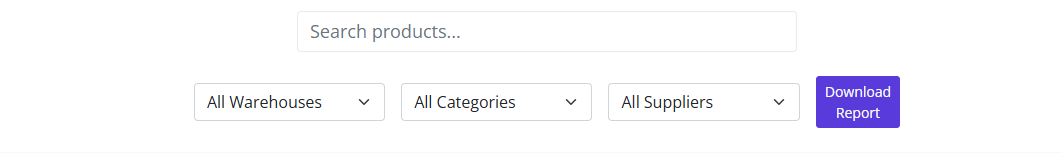
The Inventory page provides a comprehensive view of all products within the Koala Inventory Management System. It allows for detailed management of individual products, including viewing, editing, and deleting entries. The page is designed for efficiency and ease of use, allowing for quick access to critical inventory information.



#### 5.4.1 Product Listing and Search

This section displays a comprehensive list of all products within the system. Key features include:

* **Product Information:** Each product entry includes at least the following fields: Name, Description, Price, Image, Info (additional details), Edit, and Delete options. The "Info" section may contain additional product specifications or details. The inclusion of an image helps in quick identification. Edit and Delete buttons allow for direct management of product entries.
* **Search Functionality:** A search bar allows users to quickly locate specific products based on name, description, or other searchable fields. This significantly reduces the time required to find specific items within a large inventory.
* **Filtering and Sorting:** The system should allow users to filter products by warehouse, category, and supplier. Sorting options should include sorting by name, price, and other relevant fields. These features enable users to quickly narrow down their search and organize the product list according to their needs.
* **Pagination:** For large inventories, pagination is crucial to prevent overwhelming the user interface. The system should implement pagination to display products in manageable chunks.

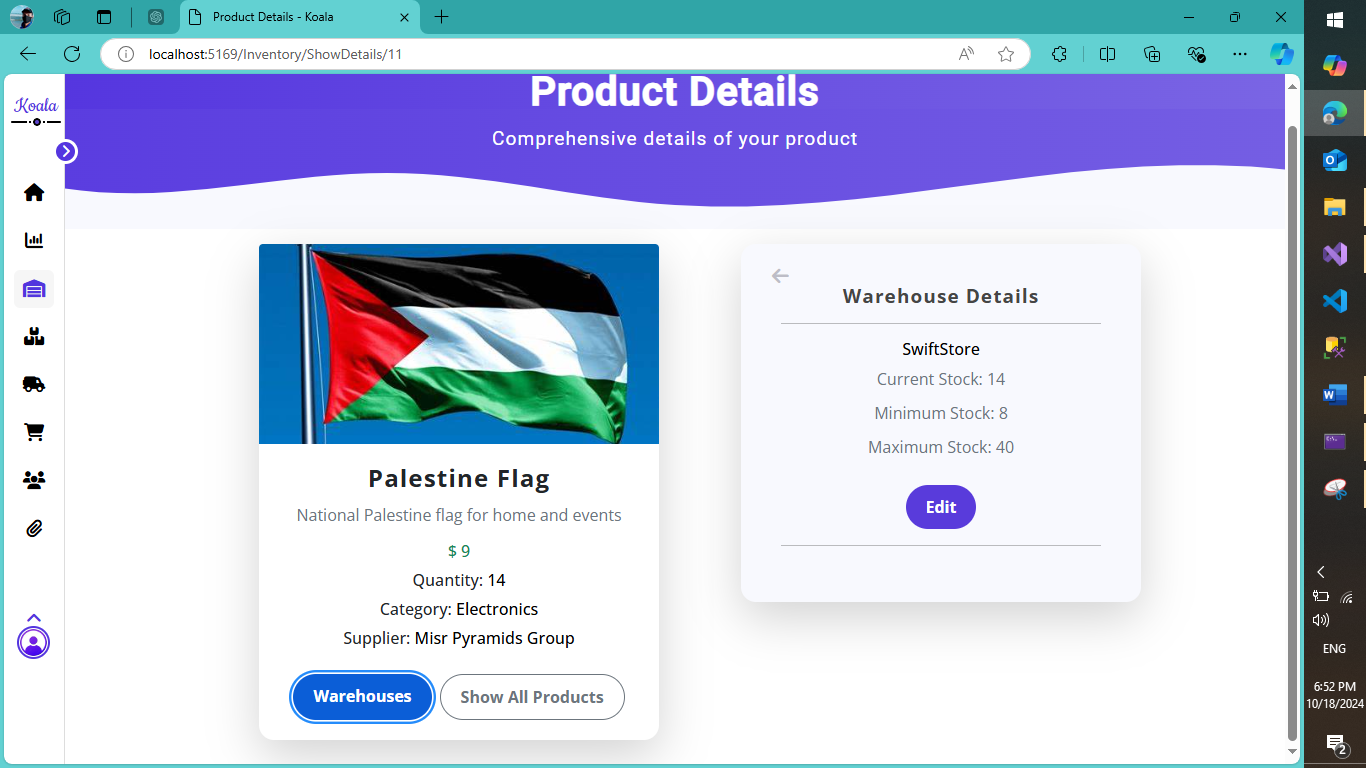


#### 5.4.2 Product Details View

Selecting a product from the listing opens a detailed view, providing comprehensive information about the selected item. This might include:

* **Expanded Product Information:** This section expands upon the information shown in the product listing, potentially including more detailed specifications, supplier information, purchase history, and other relevant data.
* **Inventory Management Tools:** This section allows for direct management of the product's inventory, including updating stock levels, setting minimum and optimal stock levels, and generating stock alerts.





#### 5.4.3 Add New Product Functionality

This section allows authorized users to add new products to the inventory. Key features include:

* **Input Fields:** A comprehensive set of input fields allows users to enter all necessary product information, including name, description, price, image upload, category selection, supplier selection, and initial stock quantity.
* **Data Validation:** The system should include data validation to ensure data integrity and prevent errors. This might include required fields, data type validation, and range checks.
* A screenshot of a computer

  Description automatically generated**Image Upload:** The ability to upload product images is crucial for easy identification and catalog management. The system should support common image formats and sizes.

#### 5.4.4 Download Report Functionality

This feature allows users to download a report containing the inventory data. Key aspects include:

* **Report Customization:** Users should be able to customize the report's content, selecting specific fields to include and choosing the report format (e.g., CSV, PDF, Excel).
* A screenshot of a computer

  Description automatically generated**Data Export:** The system should allow users to export the report data in a format suitable for further analysis or integration with other systems.

### 5.4.5 Product Editing and Deletion

This section details the procedures for modifying existing product information and removing products from the inventory. These functionalities are crucial for maintaining data accuracy and managing inventory effectively.

#### 5.4.5.1 Product Editing

The product editing functionality allows authorized users to update existing product information. This ensures that the inventory data remains accurate and up-to-date.

* **Access:** The edit function is typically accessible through an "Edit" button associated with each product in the product listing or within the detailed product view.
* **Editable Fields:** Users can modify various fields, including:
  + Product Name
  + Description
  + Price
  + Image (allowing for image replacement or deletion)
  + Category
  + Supplier
  + Stock Levels (Current, Minimum, Optimal)
* **Data Validation:** Similar to the "Add New Product" functionality, data validation is essential to ensure data integrity. This includes checks for required fields, data types, and range constraints.
* A screenshot of a product

  Description automatically generated**Update Confirmation:** An update confirmation message should be displayed to the user upon successful completion of the edit operation.

#### 5.4.5.2 Product Deletion

The product deletion functionality allows authorized users to remove products from the inventory. This is a critical function for managing obsolete or discontinued items.

* **Confirmation Dialog:** Before deleting a product, the system should display a confirmation dialog to prevent accidental deletions. This dialog should clearly state the product being deleted and prompt the user to confirm the action. The dialog should also include "Close" and "Delete" buttons.
* **Deletion Confirmation:** Upon confirmation, the product should be removed from the inventory database. A success message should be displayed to the user.
* A screenshot of a computer

  Description automatically generated**Data Integrity:** The system should handle deletion in a manner that maintains data integrity, ensuring that any related data (e.g., sales records, purchase orders) is appropriately handled. This might involve archiving or flagging the deleted product for auditing purposes.

### 5.5 Suppliers and Categories Page

This page provides a centralized interface for managing suppliers and their associated product categories. It allows for the addition, editing, and deletion of both suppliers and categories, ensuring efficient management of the supply chain.

A screenshot of a website

Description automatically generated

#### 5.5.1 Suppliers Management

This section focuses on managing the list of suppliers within the system.

* **Supplier Listing:** The page displays a list of all registered suppliers, including their name, phone number, email address, and a rating (likely a visual representation, such as a progress bar). Each supplier entry includes "Edit" and "Delete" buttons for direct management.
* **Add Supplier Functionality:** An "Add" button allows authorized users to add new suppliers to the system. A modal dialog appears, prompting for the supplier's name, phone number, email address, and rating (using a slider for input). Data validation should be implemented to ensure data integrity.
* **Edit Supplier Functionality:** The "Edit" button opens a modal dialog to modify existing supplier information. The dialog pre-populates the fields with existing data, allowing users to update the name, phone number, email address, and rating. Data validation is also crucial here.
* A screenshot of a phone

  Description automatically generatedA screenshot of a computer

  Description automatically generatedA screenshot of a contact form

  Description automatically generated**Delete Supplier Functionality:** The "Delete" button triggers a confirmation dialog to prevent accidental deletions. The dialog clearly states the supplier to be deleted and prompts for confirmation. Upon confirmation, the supplier is removed from the system, and any associated data should be handled appropriately (e.g., archiving, flagging for auditing).

#### 5.5.2 Categories by Suppliers Management

This section focuses on managing the association between suppliers and the product categories they supply.

* **Category Listing:** The page displays a list of categories, showing which suppliers provide products within each category. Each category entry includes "Edit" and "Delete" buttons.
* **Add Category Functionality:** An "Add" button opens a modal dialog to add new categories. The dialog prompts for the category name. Data validation should ensure a unique category name.
* **Edit Category Functionality:** The "Edit" button opens a modal dialog to modify existing category names. Data validation should ensure the name remains unique.
* **Delete Category Functionality:** The "Delete" button triggers a confirmation dialog before deleting a category. Similar to supplier deletion, the dialog should clearly state the category to be deleted and prompt for confirmation. Upon confirmation, the category is removed from the system, and associated data should be handled appropriately.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

#### 5.5.3 Data Integrity and Validation

Throughout the Suppliers and Categories page, data validation is crucial to ensure data integrity. This includes:

* **Unique Supplier Names:** The system should prevent the addition of duplicate supplier names.
* **Unique Category Names:** Similarly, duplicate category names should be prevented.
* **Data Type Validation:** Appropriate data types should be enforced for phone numbers and email addresses.
* **Required Fields:** All necessary fields should be marked as required to prevent incomplete entries.

A screenshot of a phone

Description automatically generated

### 5.6 Sales Page

The Sales page provides a comprehensive overview of all sales transactions within the Koala Inventory Management System. It allows for easy tracking of sales data, including the ability to add new sales records.

#### 5.6.1 Sales Listing and Navigation

This section displays a list of all sales transactions.

* **Sales Information:** Each entry includes the product name, warehouse name, quantity sold, total price, and date of sale. A "Delete" button allows for removing individual sales records.
* A screenshot of a website

  Description automatically generated**Pagination:** For a large number of sales transactions, pagination is implemented to display sales in manageable chunks, allowing for easy navigation through the sales history using "Previous" and "Next" buttons. A count of total sales and the current page is also displayed (e.g., "5 out of 5").

#### 5.6.2 Add New Sale Functionality

This section allows authorized users to add new sales transactions.

* **Input Fields:** A modal dialog appears with input fields for selecting the warehouse, entering the product name, quantity sold, and price. The total price is automatically calculated.
* **Data Validation:** Data validation is essential to ensure accuracy. This includes:
  + Required fields for all input fields.
  + Product suggestion list is displayed based on the warehouse the user has selected and the text that was written inside product’s name input field which maintains data integrity across the database.
  + Data type validation (e.g., ensuring quantity is a number).
  + Checks to ensure the entered quantity does not exceed the available stock and is a valid number (greater than 0). An error message should be displayed if the quantity exceeds stock.
  + Automatic calculation of the total price (quantity \* price).
  + User must select a warehouse, otherwise the form will not be submitted.
* **Stock Update:** Upon successful addition of a sale, the system should automatically update the stock level of the corresponding product in the relevant warehouse
* A screenshot of a login form

  Description automatically generated**Table Update:** Upon successful addition of a sale, the table will be updated automatically utilizing ajax method in JavaScript which sends a request to server asynchronously, the server responds with the updated data from the database, which is received at the client browser and the table is updated accordingly. Focusing on keeping the functionality of pagination is the same as it is, adding a new page number if necessary.

#### 5.6.3 Delete Sale Functionality

This section allows authorized users to remove sales records.

* **Confirmation Dialog:** A confirmation dialog should appear before deleting a sales record to prevent accidental deletions. The dialog should clearly state the sale to be deleted and prompt for confirmation.
* A screenshot of a computer

  Description automatically generated**Data Integrity:** Upon confirmation, the sales record is deleted. The table is then updated after the record has been deleted by utilizing Ajax Requests to fetch data from the database while maintaining a correct pagination process. The system should handle this deletion in a way that maintains data integrity, potentially archiving the deleted record for auditing purposes.

### 5.7 Admin Board

The Admin Board page provides administrative functionalities for managing users and their roles within the Koala Inventory Management System.

#### 5.7.1 User Management

This section allows administrators to manage users within the system.

* **User Listing:** The page displays a list of all users, including their name, role, and associated actions.
* **User Details:** A "User Details" button opens a modal dialog to view and edit user details, such as first name, last name, email, phone number, and role.
* **Update User Role:** An "Update User Role" button opens a modal dialog to change a user's role. A checklist of available roles is presented, allowing the administrator to select the appropriate role.
* **Delete User:** A "Delete User" button triggers a confirmation dialog before deleting a user. The dialog should clearly state the user to be deleted and prompt for confirmation. Upon confirmation, the user is removed from the system, and any associated data should be handled appropriately.

A screenshot of a contact form

Description automatically generatedA screenshot of a computer

Description automatically generated

#### 5.7.2 Role Management

This section (implied by the "Update User Role" functionality) allows administrators to manage user roles. While not explicitly shown, it's implied that roles can be added, edited, and potentially deleted. The system should allow for defining the permissions associated with each role.

A screenshot of a computer

Description automatically generated

### A screenshot of a login box Description automatically generated5.8 Account Management

This section details the functionalities related to user accounts within the Koala Inventory Management System, encompassing login, registration, password reset, and logout procedures. Robust account management ensures secure access and maintains data integrity.

#### 5.8.1 Sign-In (Login)

The Sign-In page allows registered users to access the system.

* **Input Fields:** The page features input fields for email and password. A "Remember Me" checkbox allows users to persist their login credentials for future sessions.
* **Authentication:** Upon submission, the system authenticates the provided credentials against the stored user data. If the credentials are valid, the user is granted access to the system, redirecting them to the Dashboard.
* **Error Handling:** If the credentials are invalid (incorrect email or password), an appropriate error message should be displayed, guiding the user to correct their input.
* **Forgot Password Link:** A "Forgot Password" link provides a pathway for users to reset their password if they have forgotten it.
* **Guest Access:** A "Continue as Guest" option (if implemented) allows limited access to certain functionalities without requiring a login. The scope of guest access should be clearly defined.

A screenshot of a computer

Description automatically generated

#### 5.8.2 Registration (Create Account)

The Registration page allows new users to create accounts within the system.

* **Input Fields:** The page requires various input fields, including first name, last name, email address, phone number, password, and password confirmation.
* **Password Complexity:** The system should enforce password complexity requirements, such as minimum length, inclusion of uppercase and lowercase letters, numbers, and special characters. Clear instructions should be provided to the user regarding these requirements. Real-time feedback on password strength is highly recommended.
* **Terms and Conditions:** A checkbox requiring agreement to the terms and conditions is essential for legal compliance.
* **Data Validation:** Comprehensive data validation is crucial to ensure data integrity and prevent errors. This includes:
  + Required fields for all input fields.
  + Data type validation (e.g., email format).
  + Unique email address validation (preventing duplicate accounts).
  + Password confirmation matching.
* A screenshot of a login form

  Description automatically generated**Account Creation:** Upon successful submission and validation, a new user account is created, and the user is redirected to the Sign-In page or another appropriate location. A confirmation email might be sent to the registered email address.

#### 5.8.3 Password Reset

The Password Reset functionality allows users to recover their passwords if forgotten.

* **Email Input:** The user provides their registered email address.
* **Email Verification:** The system verifies the provided email address against existing user accounts.
* **Password Reset Email:** If the email address is valid, the system generates a temporary password or a password reset link and sends it to the user's email address.
* **Password Update:** The user follows the instructions in the email to update their password, potentially setting a new password that meets the system's complexity requirements.
* **Error Handling:** Appropriate error messages should be displayed if the email address is invalid or if any other issues occur during the password reset process.

A screenshot of a login form

Description automatically generated

#### 5.8.4 Sign Out (Logout)

The Sign Out functionality allows users to securely terminate their current session.

* **Session Termination:** Upon clicking the "Sign Out" button, the system invalidates the user's session, effectively logging them out.
* **Redirection:** The user is typically redirected to the Sign-In page or a landing page.
* A screenshot of a computer

  Description automatically generated**Security:** The logout process should be designed to ensure secure session termination, preventing unauthorized access. Consider using secure session management techniques.

#### 5.8.5 User Profile

A user profile section (accessible after login) allows users to view and modify their personal information. This might include:

* **View/Edit Personal Information:** Users can view and update their first name, last name, email address, and phone number.
* **Change Password:** Users can change their password, adhering to the system's password complexity requirements.

**Chapter 6**

# 6 . System testing

## 6.1 Introduction

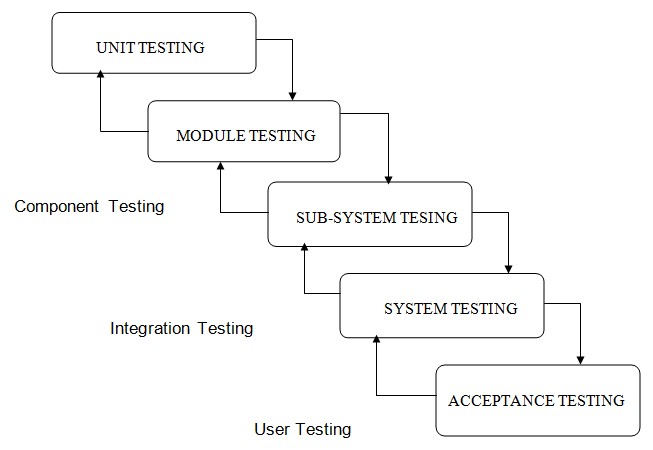
Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. In fact, testing is the one step in the software engineering process that could be viewed as destructive rather than constructive.

A strategy for software testing integrates software test case design methods into a well-planned series of steps that result in the successful construction of software. Testing is the set of activities that can be planned in advance and conducted systematically. The underlying motivation of program testing is to affirm software quality with methods that can economically and effectively apply to both strategic to both large and small-scale systems.

## 6.2 Strategic approach to software testing

The software engineering process can be viewed as a spiral. Initially system engineering defines the role of software and leads to software requirement analysis where the information domain, functions, behavior, performance, constraints and validation criteria for software are established. Moving inward along the spiral, we come to design and finally to coding. To develop computer software we spiral in along streamlines that decrease the level of abstraction on each turn.

A strategy for software testing may also be viewed in the context of the spiral. Unit testing begins at the vertex of the spiral and concentrates on each unit of the software as implemented in source code. Testing progress by moving outward along the spiral to integration testing, where the focus is on the design and the construction of the software architecture. Talking another turn on outward on the spiral we encounter validation testing where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally we arrive at system testing, where the software and other system elements are tested as a whole.



## 6.3 The levels of testing

The levels of testing that software goes through are:

**ACCEPTANCE TESTING**: Testing to verify a product meets customer specified requirements. A customer usually does this type of testing on a product that is developed externally.



**BLACK BOX TESTING.** Testing without knowledge of the internal workings of the item being tested. Tests are usually functional.



**COMPATIBILITY TESTING.** Testing to ensure compatibility of an application or Web site with different browsers, OSs, and hardware platforms. Compatibility testing can be performed manually or can be driven by an automated functional or regression test suite.



**CONFORMANCE TESTING.** Verifying implementation conformance to industry standards. Producing tests for the behavior of an implementation to be sure it provides the portability, interoperability, and/or compatibility a standard defines.

**FUNCTIONAL TESTING**. Validating an application or Web site conforms to its specifications and correctly performs all its required functions. This entails a series of tests which perform a feature by feature validation of behavior, using a wide range of normal and erroneous input data. This can involve testing of the product's user interface, APIs, database management, security, installation, networking, etc testing can be performed on an automated or manual basis using black box or white box methodologies.



**INTEGRATION TESTING.** Testing in which modules are combined and tested as a group. Modules are typically code modules, individual applications, client and server applications on a network, etc. Integration Testing follows unit testing and precedes system testing.



**LOAD TESTING**. Load testing is a generic term covering Performance Testing and Stress Testing.



**PERFORMANCE TESTING.** Performance testing can be applied to understand your application or WWW site's scalability, or to benchmark the performance in an environment of third party products such as servers and middleware for potential purchase. This sort of testing is particularly useful to identify performance bottlenecks in high use applications. Performance testing generally involves an automated test suite as this allows easy simulation of a variety of normal, peak, and exceptional load conditions.



**REGRESSION TESTING.** Similar in scope to a functional test, a regression test allows a consistent, repeatable validation of each new release of a product or Web site. Such testing ensures reported product defects have been corrected for each new release and that no new quality problems were introduced in the maintenance process. Though regression testing can be performed manually an automated test suite is often used to reduce the time and resources needed to perform the required testing.



**SMOKE TESTING.** A quick-and-dirty test that the major functions of a piece of software work without bothering with finer details. Originated in the hardware testing practice of turning on a new piece of hardware for the first time and considering it a success if it does not catch on fire.



**STRESS TESTING.** Testing conducted to evaluate a system or component at or beyond the limits of its specified requirements to determine the load under which it fails and how. A graceful degradation under load leading to non-catastrophic failure is the desired result.

Often Stress Testing is performed using the same process as Performance Testing but employing a very high level of simulated load.



**SYSTEM TESTING.** Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

### 6.3.1 Unit Testing

Unit testing focuses verification effort on the smallest unit of software design, the module. The unit testing we have is white box oriented and some modules the steps are conducted in parallel.

#### 6.3.1.1 White box testing

This type of testing ensures that:

* All independent paths have been exercised at least once
* All logical decisions have been exercised on their true and false sides
* All loops are executed at their boundaries and within their operational bounds
* All internal data structures have been exercised to assure their validity.

To follow the concept of white box testing we have tested each form .we have created independently to verify that Data flow is correct, All conditions are exercised to check their validity, All loops are executed on their boundaries.

#### 6.3.1.2 Conditional testing

In this part of the testing each of the conditions were tested to both true and false aspects. And all the resulting paths were tested. So that each path that may be generate on particular condition is traced to uncover any possible errors.

#### 6.3.1.3 Data flow testing

This type of testing selects the path of the program according to the location of definition and use of variables. This kind of testing was used only when some local variable were declared. The definition-use chain method was used in this type of testing. These were particularly useful in nested statements.

#### 6.3.1.4 Loop testing

In this type of testing all the loops are tested to all the limits possible. The following exercise was adopted for all loops:

* All the loops were tested at their limits, just above them and just below them.
* All the loops were skipped at least once.
* For nested loops test the inner most loop first and then work outwards.
* For concatenated loops the values of dependent loops were set with the help of connected loop.
* Unstructured loops were resolved into nested loops or concatenated loops and tested as above.

Each unit has been separately tested by the development team itself and all the input have been validated.

## 6.4 Methodology

With this testing approach, all the specs were ready for a prototype, and a plan was already built to be shown; the tester started writing his or her code and saw if he or she could obtain the same results that the specs mentioned. This way, the specs were tested on each prototype, and continuous testing was applied. This also helped to minimize the testing that would have to be implemented at the end of the software lifecycle. In the process, all aspects of the software were tested.

Steps to follow while implementing the methodology are as follows:

1. Start with a base function that you want to implement.
2. Create a document with the detailed requirement definition, an activity diagram with a description of the flow, database tables to be used, a component diagram, and a description of each component with the precondition and tables that would be affected by the component.
3. Give the document to the tester, and work with the tester while he or she writes the code to check if the steps in the document can be implemented and if the result of each use case can be achieved.
4. If the tester finds a step difficult to implement or thinks he or she is missing additional information to implement the functionality, then go to step 2; otherwise, go to step 3.
5. Ask the tester to log on all the errors and difficulties he or she encountered while working on the prototype implementation.
6. Once the prototype is done and the results between the developer’s prototype and tester's prototype match, work on the other requirement, and expand the prototype to final software.
7. When the testing approach was implemented, the following pros and cons regarding the testing approach were realized.

### 6.4.1 Pros of using the methodology

* Helps give a better understanding about the requirements.
* Better design at the end of the cycle.
* Reduced testing to be performed at the end of the cycle
* Documents produced would be of higher quality.

### 6.4.2 Cons of using the methodology

* The person working on the document should be experienced.
* There are increased time and money involved with testing.
* Different viewpoints for the same problem can lead to varying results.

## 6.5 Interface Testing

### 6.5.1 Functional requirement

This section lists the functional requirements used for creating the test-case table, the test cases that were used to verify the interface table, and the results for the test-cases table.

Table 1 lists the functional requirements for the interface built for the online shopping-cart application, along with a short description of each requirement

|  |  |  |  |
| --- | --- | --- | --- |
| **Functional Requirement Number** | | | **Functional Requirement Short Description** |
| **R1** | | | The web portal application shall have two types of authentication: User authentication and Admin authentication. |
| **R2** | | | The web portal application shall be accessible to all the users to browse all the categories and the items. |
| **R3** | | | The users shall be able to view the items they added to the cart. |
| **R6** | | | The Admin shall be able to add new items as well as to add/modify the categories. |
| **R7** | | | The Admin shall be able to view all the users registered in the system. |
| **R9** | | | The Admin shall be able to view all the information about users who placed an order from the cart. |
| **R9** | | | The users shall not be able to check out with an |
|  | empty cart. | | |
| **R10** | The users shall not be able to place an order without providing valid information for all rows in the order form. | | |
| **R11** | The users shall not be able to place an order if any of the columns in the order form are left empty. | | |

**Table 1**

### 6.5.2 Test Cases

Table 2 shows the functional requirements used to write the test cases along with the test-case numbers for each test case and a short description of the test cases.

|  |  |  |
| --- | --- | --- |
| **Functional Requirement Number** | **Test Case No.** | **Test-Case Short Description** |
| **R1** | T1 | To test the Login/Authentication interface for the Admin |
| **R2** | T2 | To test the Login/Authentication interface for the users |
| **R3** | T3 | To test, users can view the items they add in the shopping cart |
| **R6** | T4 | To test, Admin can upload new/revised categories. |
| **R7** | T5 | To test, Admin can upload new/revised items. |
| **R8** | T6 | To test, Admin can view all the users registered in the system  To test, Admin can view the information about all the users who successfully placed an order. |
| **R9** | T7 | To test that users cannot check out with an empty shopping cart. |
| **R10** | T8 | To test that users are not able to submit an order form if the information in any of the fields is not valid. |
| **R11** | T9 | To test that users are not able to submit an order form if the information in any of the fields is left blank. |

**Table 2**

The following list includes the steps that should be taken by the user, the conditions that should be met for the successful execution of the test case, and the end result that should be met for the test cases to pass.

1. T1: To test the Login/Authentication interface
   * + Input: Username and Password
     + Output: Valid Destination Page
     + Valid Range: User Name Alphanumeric, Password  Alphanumeric - End Messages/Result
   1. If (User == Valid User), an order form appears to complete the checkout process

* 1. If (User != Valid User), an error message is displayed on the Login interface.

1. T2: To test, the users can view the items they add to the shopping cart.
   * + Input: The user adds an item to the cart from any of the available categories.
     + Output: The cart page pops up, showing the item that is added by the user. - End messages/Result
   1. If (Selection == Item and document == exists), the user is able to add that item to the cart, and the item shows up in the shopping cart, prompting user to delete the item, to continue shopping, or to check out the item.

* 1. If (Selection = Item and Selection = View Cart), an empty shopping cart pops up with buttons to check out or to continue shopping.

1. T3: To test, the Admin can upload new/revised categories and items.

- Input

i. User=Admin ii. Selection=Items iii. Selection=Categories

- Output: New or modified items in the cart. - End messages/Result

* 1. If (User type = "Admin" &Selection = (Items || Category)&& Item/Category =existing), then display the modified items in the cart.

* 1. If (User type == "Admin" &Selection == (Items || Category) & Item/ Category=existing), then display newly added items in the cart.

1. T4: To test, the Admin can view all the users registered in the system.

- Input

i. User Name Alphanumeric, Password  Alphanumeric ii. User==Admin iii. Selection==View Database

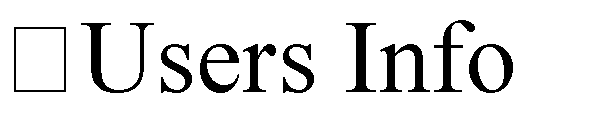
* + - Output: User List
    - End messages/Result
  1. If( login type == "Admin" & Database.clicked = "true" and list.clicked=true and userlist.exists=true), then display users.

* 1. If (login type == "Admin" &Database.clicked = "true" and list.clicked=true and userlist.exists=false), then display the empty database.

1. T5: To test, the Admin can view the information about all users who successfully placed an order.

- Input

i. User Name Alphanumeric, Password  Alphanumeric ii. User==Admin iii. Selection==Database



- Output: A database with the user’s information or an empty database. - End messages/Result

* 1. If (login type == "Admin" & Checkout.clicked = "true" and Place

Order.clicked=true and userlist.exists=true), then display the database containing the user's personal information.

* 1. If (login type = "Admin" &Checkout.clicked = "true" and

PlaceOrder.clicked=true and userlist.exists = false), then display the empty database.

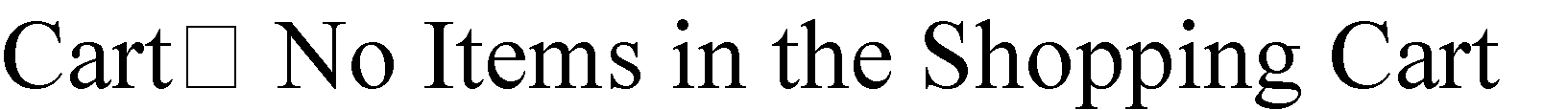
* 1. If (login type = "Admin" & Checkout.clicked = "true" and

PlaceOrder.clicked=false and userlist.exists=true), then do not update the row in the database.

1. T6: To test that users cannot checkout with an empty shopping cart.

- Input

* + - 1. User Name Alphanumeric, Password  Alphanumeric
      2. User==Users iii. Selection==View



* + - Output: Disabled checkout button.
    - End messages/Result

* 1. If (login type == "Users" & Items.AddToCart = "false"

&ViewCart.clicked="true"), then display the empty shopping cart with no items and a disabled checkout button.

* 1. If (login type == "Users" & Items.AddToCart = "true"

&ViewCart.clicked="true" &Checkout.clicked=="rue"), then display items in the shopping cart with the checkout button enabled so that users can check out.

1. T7: To test that users are not able to submit an order form if the information in any of the fields is invalid.

- Input

i. User Name Alphanumeric, Password  Alphanumeric ii. User==Users iii. Selection==Checkout Order Form  Place Order

* + - Output: User successfully or unsuccessfully places the order.
    - End messages/Result

* 1. If (login type == "User" &CheckoutButton.clicked = "true"

andOrderFormInformation.Valid=="false" or

OrderFormInformation.Invalid=="false"&& PlaceOrder.clicked=true), then display an error message after the place order button is clicked.

* 1. If (login type == "User" &CheckoutButton.clicked = "true"

andOrderFormInformation.Valid=="true" and

OrderFormInformation.Invalid=="true"PlaceOrder.clicked=true), then successfully place the order and display the success message.

## 6.6 Results

This section lists the results that were produced by running the test cases. Table 3 lists the test cases that were used while testing the interface along with the expected result and the actual results for each test case.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case Number** | **Expected result** |  | **Actual result** |
| **T1** | Pass |  | Pass |
| **T2** | Pass |  | Pass |
| **T3** | Pass |  | Pass |
| **T4** | Pass |  | Pass |
| **T5** | Pass |  | Pass |
| **T6** | Pass |  | Pass |
| **T7** | Pass |  | Pass |

**Table 3**

**Chapter 7**

# 7 . System Security

## 7.1 Introduction

The protection of computer based resources that includes hardware, software, data, procedures and people against unauthorized use or natural Disaster is known as System Security.

System Security can be divided into four related issues:

* Security
* Integrity
* Privacy
* Confidentiality

**SYSTEM SECURITY** refers to the technical innovations and procedures applied to the hardware and operation systems to protect against deliberate or accidental damage from a defined threat.

**DATA SECURITY** is the protection of data from loss, disclosure, modification and destruction.

**SYSTEM INTEGRITY** refers to the power functioning of hardware and programs, appropriate physical security and safety against external threats such as eavesdropping and wiretapping.

**PRIVACY** defines the rights of the user or organizations to determine what information they are willing to share with or accept from others and how the organization can be protected against unwelcome, unfair or excessive dissemination of information about it.

**CONFIDENTIALITY** is a special status given to sensitive information in a database to minimize the possible invasion of privacy. It is an attribute of information that characterizes its need for protection.

## 7.2 Security in software

System security refers to various validations on data in form of checks and controls to avoid the system from failing. It is always important to ensure that only valid data is entered and only valid operations are performed on the system. The system employees two types of checks and controls:

### 7.2.1 Client side validation

Various client side validations are used to ensure on the client side that only valid data is entered. Client side validation saves server time and load to handle invalid data. Some checks imposed are:

* VBScript in used to ensure those required fields are filled with suitable data only. Maximum lengths of the fields of the forms are appropriately defined.
* Forms cannot be submitted without filling up the mandatory data so that manual mistakes of submitting empty fields that are mandatory can be sorted out at the client side to save the server time and load.
* Tab-indexes are set according to the need and taking into account the ease of user while working with the system.

### 7.2.2 Server side validation

Some checks cannot be applied at client side. Server side checks are necessary to save the system from failing and intimating the user that some invalid operation has been performed or the performed operation is restricted. Some of the server side checks imposed is:

* Server side constraint has been imposed to check for the validity of primary key and foreign key. A primary key value cannot be duplicated. Any attempt to duplicate the primary value results into a message intimating the user about those values through the forms using foreign key can be updated only of the existing foreign key values.
* User is intimating through appropriate messages about the successful operations or exceptions occurring at server side.
* Various Access Control Mechanisms have been built so that one user may not agitate upon another. Access permissions to various types of users are controlled according to the organizational structure. Only permitted users can log on to the system and can have access according to their category. User- name, passwords and permissions are controlled o the server side.
* Using server side validation, constraints on several restricted operations are imposed.

**Chapter 8**

# 8 . Conclusion and future work

## 8.1 Conclusion

The Internet has become a major resource in modern business, thus electronic shopping has gained significance not only from the entrepreneur’s but also from the customer's point of view. For the entrepreneur, electronic shopping generates new business opportunities and for the customer, it makes comparative shopping possible. As per a survey, most consumers of online stores are impulsive and usually make a decision to stay on a site within the first few seconds. Website design is like a shop interior. If the shop looks poor or like hundreds of other shops the customer is most likely to skip to the other site. Hence we have designed the project to provide the user with easy navigation, retrieval of data and necessary feedback as much as possible.

In this project, the user is provided with an web portal site that can be used to buy Products online. To implement this as a web application we used ASP.NET as the Technology. ASP.NET has several advantages such as enhanced performance, scalability, built- in security and simplicity. To build any web application using ASP.NET we need a programming language such as C#, VB.NET, J# and so on. C# was the language used to build this application. For the client browser to connect to the ASP.NET engine we used Microsoft's Internet Information Services (IIS) as the Web Server. ASP.NET uses ADO.NET to interact with the database as it provides in-memory caching that eliminates the need to contact the database server frequently and it can easily deploy and maintain an ASP.NET application. SQL Server was used as back-end database since it is one of the most popular open source databases, and it provides fast data access, easy installation and simplicity.

A good shopping cart design must be accompanied with user-friendly shopping cart application logic. It should be convenient for the customer to view the contents of their cart and to be able to remove or add items to their cart. The shopping cart application described in this project provides a number of features that are designed to make the customer more comfortable.

This project helps in understanding the creation of an interactive web page and the technologies used to implement it. The design of the project which includes Data Model and Process Model illustrates how the database is built with different tables, how the data is accessed and processed from the tables. The building of the project has given me a precise knowledge about how ASP.NET is used to develop a website, how it connects to the database to access the data and how the data and web pages are modified to provide the user with a shopping cart application.

It has been a great pleasure for me to work on this exciting and challenging project. This project proved good for me as it provided practical knowledge of not only programming in ASP.NET and C#.NET web based application and no some extent Windows Application and SQL Server, but also about all handling procedure related with our secure web portal for E-marketing. It also provides knowledge about the latest technology used in developing web enabled application and client server technology that will be great demand in future. This will provide better opportunities and guidance in future in developing projects independently.

## 8.2 Benefits:

The project is identified by the merits of the system offered to the user. The merits of this project are as follows:

* It's a web-enabled project.
* This project offers user to enter the data through simple and interactive forms. This is very helpful for the client to enter the desired information through so much simplicity.
* The user is mainly more concerned about the validity of the data, whatever he is entering. There are checks on every stages of any new creation, data entry or updating so that the user cannot enter the invalid data, which can create problems at later date.
* Sometimes the user finds in the later stages of using project that he needs to update some of the information that he entered earlier. There are options for him by which he can update the records. Moreover there is restriction for his that he cannot change the primary data field. This keeps the validity of the data to longer extent.
* User is provided the option of monitoring the records he entered earlier. He can see the desired records with the variety of options provided by him.
* From every part of the project the user is provided with the links through framing so that he can go from one option of the project to other as per the requirement. This is bound to be simple and very friendly as per the user is concerned. That is, we can sat that the project is user friendly which is one of the primary concerns of any good project.
* Data storage and retrieval will become faster and easier to maintain because data is stored in a systematic manner and in a single database.
* Decision making process would be greatly enhanced because of faster processing of information since data collection from information available on computer takes much less time then manual system.
* Allocating of sample results becomes much faster because at a time the user can see the records of last years.
* Easier and faster data transfer through latest technology associated with the computer and communication.
* Through these features it will increase the efficiency, accuracy and transparency,

## 8.3 Future work:

There are some limitations for the current system to which solutions can be provided as a future development:

1. The system is not configured for multi- users at this time. The concept of transaction can be used to achieve this.

1. The Website is not accessible to everyone. It can be deployed on a web server so that everybody who is connected to the Internet can use it.

1. Credit Card validation is not done. Third party proprietary software can be used for validation check.

1. The Administrator of the web site can be given more functionality, like looking at a specific customer's profile, the books that have to be reordered, etc.

1. Multiple Shopping carts can be allowed.

1. This System being web-based and an undertaking of Cyber Security Division, needs to be thoroughly tested to find out any security gaps.

1. A console for the data centre may be made available to allow the personnel to monitor on the sites which were cleared for hosting during a particular period.

1. Moreover, it is just a beginning; further the system may be utilized in various other types of auditing operation viz. Network auditing or similar process/workflow based applications...

1. Mobile application for the project can also be created to make our project more reliable

# 9- References

1. Chen, L. (2000). Enticing Online Consumers: A Technology Acceptance

Perspective Research- in-Progress, ACM Proceedings, SIGCPR

1. Diwakar, H., Marathe, M. (2000). The architecture of a one-stop web-window shop. December, ACM SIGecom Exchanges, Volume 2 Issue 1.
2. Morrison, M, Morrison, J., and Keys, A. Integrating Web Sites and Databases. Communications of the ACM, September, Volume 45, Issue 9
3. Kubilus, N. J. (2000). Designing an e-commerce site for users. September 2000, Crossroads, Volume 7 Issue 1.
4. Anderson, R., Francis, B., Homer, A., Howard, R., Sussman, D. and Watson. (2001) Professional ASP.NET. Wrox Press Ltd.
5. Wagner, R., Daniels, K., Griffin, G., Haddad, C. and Nasr, J. (1997) JavaScript Unleashed. SAMS Net.
6. Wiley, Y. M. J. & Sons. (1997) Creating the Virtual Store: Taking Your Web Site from Browsing to Buying.
7. http://encyclopedia.laborlawtalk.com/IIS for information on IIS
8. http://aspnet.4guysfromrolla.com/articles/020404-1.aspx for relationship between IIS and ASP.NET.

10.

http://216.15.201.66/dpec/course.htm?fullpg=http%3A//216.15.201.66/dpec/cours es/wac312/wah006.htm&acro=wac312 for security authentication in ASP.NET

1. http://samples.gotdotnet.com/quickstart/aspplus/doc/mtstransactions.aspx for information on Transactions in ASP.NET.
2. http://www.x-cart.com/articles/design\_development.html for online customer behavior.
3. http://aspnet.4guysfromrolla.com/articles/011404-1.aspx for relation between IIS and ASP.NET.
4. http://www.informatik.uni-bremen.de/uniform/gdpa\_d/methods/m-fctd.htm for definition of Functional Decomposition.
5. http://www.agilemodeling.com/artifacts/dataFlowDiagram.htm for definition of Data Flow Diagram.
6. [http://www.startvbdotnet.com/ado/default.aspx for information on 17.](http://www.startvbdotnet.com/ado/default.aspx%20for%20information%20on%2017) 17. http://mypage.iusb.edu/~hhakimza/505/index.html for ADO.NET objects.
7. http://msdn.microsoft.com for ADO.NET objects.
8. King, Stephen F. en Juhn-ShiuanLiou (2004), A framework for internet channel evaluation. International Journal of Information & Management 24:473488.
9. Falk, Louis K., Sockel, Hy, and Chen, Kuanchin. (2005). E-Commerce and consumer's expectations: what makes a website work. Journal of Website Promotion, 1:65–75.
10. Elliot, Steve, and Fowell, Sue. (2000). Expectations versus reality: a snapshot of consumer experiences with Internet retailing", International Journal of Information Management 20: 323-336.
11. Java vs.C#, Retrieved on 2013.

"http://www.cs.yale.edu/homes/hudak/CS112F06/java.html"

12. Java Shopping Cart Powered by SoftSlate., Retrieved on 2013.

"http://www.softslate.com/featureList.html"