***Abstract***

This project presents the development of ServiceHub MVM, a scalable and modular multi-vendor service marketplace designed to streamline service listings, bookings, and transactions between vendors and customers in a centralized ecosystem. The system allows vendors to register, authenticate, and manage their service offerings, while customers can seamlessly browse categorized services, make real-time bookings, and manage personal profiles and order histories. Additionally, an administrator dashboard facilitates system-wide oversight, including user management, performance analytics, and order regulation.

ServiceHub MVM is engineered to function independently or integrate with popular content management systems (CMS) such as WordPress, making it highly extensible and compatible with existing web infrastructures. The platform emphasizes modularity, maintainability, and role-based access control through the implementation of custom user roles and dedicated dashboards for Admin, Vendor, and Customer tiers.

Core features include a dynamic service booking system with real-time form validation and asynchronous submission, custom order generation and tracking mechanisms, role-specific dashboards for streamlined user experiences, advanced filtering and search capabilities based on taxonomies and price ranges, secure Stripe payment integration for transactional operations, administrative dashboard widgets to monitor platform metrics, and automated email notifications for booking confirmations and vendor alerts. The system also ensures robust security through role-based authentication and session validation. Built using modern development practices, ServiceHub MVM provides a robust foundation for launching a commercial-grade service marketplace connecting digital service providers with end-users.

The system architecture follows modern web development paradigms using a mix of backend and frontend technologies, ensuring scalability, extensibility, and performance. This project stands as a blueprint for building a commercial-grade online service marketplace that bridges the gap between digital service providers and consumers.

***Chapter 1: Introduction***

In today’s digital age, the way people find and book services has changed dramatically. From hiring electricians to booking photography sessions, users now expect fast, convenient, and secure ways to connect with professionals. Traditional service models often involve time-consuming phone calls or visits, while many local businesses struggle with limited online presence. To address this gap, online service marketplaces have become increasingly popular, acting as a bridge between customers and service providers.

This project presents a modern solution: a Multi-Vendor Service Marketplace that simplifies the entire process for all stakeholders. Service providers (vendors) can register, showcase their offerings, manage orders, and receive payments. At the same time, customers can explore services, apply filters to find relevant options, and book services with ease. The platform is user-friendly, scalable, and designed to support integration with widely-used content management systems, including WordPress.

**1.1 Background**

In today’s digital age, service-based businesses are increasingly moving online to reach more customers and streamline operations. While product-based eCommerce has seen massive growth and support from major platforms, service-based businesses—especially local and independent providers—have limited access to structured, scalable platforms. Many vendors still rely on traditional communication methods like calls or physical visits to get leads, schedule appointments, and handle payments. This results in inefficiencies, limited reach, and lack of automation.

To bridge this gap, a comprehensive digital marketplace that can bring together vendors from different industries under a unified system is needed. A Multi-Vendor Service Marketplace empowers individual service providers to register, showcase their services, manage bookings, and receive payments, all within a controlled environment. It also gives customers a centralized platform to compare services, check availability, and place bookings. This concept has been implemented in some global platforms, but customized, affordable, and CMS-compatible solutions are still lacking, especially for regional and small-scale use.

**1.2 Problem Statement**

Despite the rapid digitization of commerce and services, there is a significant gap in the availability of structured, user-friendly platforms tailored for service-based businesses—especially those that support multiple vendors. Most existing online solutions are either tailored to product-based eCommerce, require extensive technical knowledge, or demand high development costs, making them inaccessible for small to mid-sized service providers.

Local service providers often lack the technical infrastructure to create their own websites or manage online bookings, payments, and customer communications. As a result, they remain dependent on outdated methods like phone bookings or word-of-mouth promotions, leading to limited market reach, inefficient operations, and lost revenue opportunities.

On the other hand, customers frequently encounter fragmented experiences when attempting to discover and book services. There is no unified system where users can explore services by category, location, or price, compare providers, view availability in real time, and complete bookings with integrated payment options. This results in a lack of transparency, trust, and convenience.

Moreover, many existing multi-vendor platforms do not provide personalized dashboards for different user roles (e.g., admin, vendor, customer), leading to confusion and poor user engagement. They also lack essential features like order tracking, automated notifications, or secure payment integration.

Therefore, there is a clear need for a dedicated, scalable, and CMS-compatible Multi-Vendor Service Marketplace—a platform that can streamline vendor onboarding, service discovery, booking management, and secure transactions in a single ecosystem. This project aims to address this gap by offering an extensible, role-based solution that enhances both vendor capabilities and customer experience.

**1.3 Objectives**

The primary aim of this project is to develop a functional Multi-Vendor Service Marketplace that simplifies service discovery, booking, and vendor management using a web-based system. The platform should be flexible enough to support a variety of service categories and user interactions while ensuring usability, performance, and security.

The key objectives of this system include the creation of dedicated dashboards for each user type—Admin, Vendor, and Customer—along with support for real-time service listings, booking forms, order generation, and payment processing. The platform must also provide a responsive interface, advanced service filtering (by location, type, and price), and role-based access controls. Admins should have oversight of user registrations, orders, and vendor approvals. The system should be extensible and compatible with popular content management frameworks to ensure broader usability.

**1.4 Scope**

This project focuses on developing a full-stack web application that can support multiple service vendors across various industries. The platform will allow vendors to register, list their services with relevant details and media, and manage their own orders and availability. Customers will be able to explore these services, place bookings, track orders, and manage their profiles from a personalized dashboard.

The project also includes a role-based access system, meaning different interfaces and permissions are available based on user type. It incorporates features like service filtering, secure user authentication, order status management, and integrated email notifications. Although the project is built to be CMS-compatible, it is developed in a way that the core logic can be adapted for non-CMS platforms as well. Features such as review systems, messaging, or mobile apps may be considered outside the current scope but could be added in future expansions.

**1.5 Methodology**

To ensure a structured and efficient development process, this project followed a combination of the Waterfall and Iterative methodologies. The Waterfall model was applied during the initial stages of the project to define the requirements clearly, plan the development phases, and establish a complete understanding of the project's goals and scope. This helped in maintaining a clear roadmap and ensuring that each core feature—such as the user roles, service listings, and booking mechanism—was addressed systematically. Once the foundation was laid, the Iterative model was used to refine, improve, and test functionalities through cycles of development, testing, and feedback.

The development was divided into key modules, including the creation of custom user roles (Admin, Vendor, Customer), implementation of service posting features, dynamic booking systems, and secure payment integration. Each module was developed separately and tested individually before being integrated into the main system. Backend logic and database management were handled using PHP and MySQL, with custom post types and metadata structures ensuring flexibility and scalability. Frontend design was done using responsive frameworks to provide an optimal user experience on both desktop and mobile devices.

Testing and validation were a critical part of the methodology. Functional testing was conducted at each phase to ensure that all features worked as intended. Role-based testing ensured that each user had access only to relevant data and actions. Edge cases, such as invalid form submissions, duplicate user registrations, and payment failures, were also tested thoroughly. The methodology allowed for real-time feedback and continuous improvements throughout the development cycle. As a result, the system is not only functionally robust but also flexible enough for future enhancements and scalability.

***Chapter 2: Literature Review***

A solid understanding of prior research and related systems is essential for developing a well-informed and technically sound software solution. This chapter investigates existing literature, tools, and technologies that are relevant to the development of a multi-vendor service marketplace. The review focuses on platforms that enable service listing, user role management, booking systems, and transaction handling. Additionally, the research includes existing CMS platforms, plugin architectures, and service-oriented portals to highlight both the strengths and limitations of current systems.

Several online platforms facilitate service-based transactions between vendors and customers, such as Fiverr, TaskRabbit, and UrbanClap. These systems follow a core principle of service aggregation, enabling users to access various services under a single unified portal. While these platforms provide end-to-end functionality, they are proprietary and lack customization freedom for developers seeking tailored solutions for niche industries. Open-source alternatives, including various WordPress-based systems, offer a modular approach where features can be integrated via plugins, making them flexible and extensible.

In terms of architecture, many of the successful systems rely on a component-based modular framework where different user roles such as admin, vendor, and customer interact through distinct yet interconnected dashboards. Payment gateways like Stripe or PayPal are integrated to streamline financial transactions and commissions. Literature also highlights the use of REST APIs for real-time data synchronization between users and services, ensuring a smooth and responsive user experience.

A recurring limitation observed in many platforms is the lack of personalized dashboards for different user roles. Vendors often lack comprehensive tools to manage bookings, while customers experience limited tracking options post-purchase. The administrative dashboards are often generic, without real-time performance indicators or analytics. These gaps present an opportunity for systems that not only centralize marketplace features but also focus on the experience and functionality unique to each user type.

Several academic papers and technical journals also examine the use of taxonomies and metadata in filtering services, which is especially relevant to this project’s search and filter functionality. Moreover, case studies around Stripe Connect demonstrate its capability to facilitate direct vendor payouts in multi-vendor setups, eliminating the need for complex manual intervention. This provides a solid technical foundation for implementing secure and automated payment flows.

In conclusion, the literature review reflects a growing demand for flexible, modular, and role-specific service marketplaces. The reviewed platforms and research works have influenced the design choices in this project, especially in the implementation of features such as dynamic dashboards, booking systems, custom user roles, and scalable transaction mechanisms. This chapter provides the contextual basis for decisions made in system architecture, user flow, and feature prioritization in the development phase.

***Chapter 3: Requirements Specification***

A well-defined requirements specification is essential to the success of any software project. It establishes a clear understanding between the development team and stakeholders by outlining what the system must do, how it should behave, and the constraints it must operate within. This chapter documents all the functional and non-functional requirements for the **Multi-Vendor Service Marketplace** project. These requirements serve as a foundation for design, development, testing, and validation phases.

The requirements have been categorized based on their purpose and scope, and each is assigned a unique identifier for easy tracking. Functional requirements define what the system should do, such as user registration, service booking, and order management. Non-functional requirements specify the quality attributes of the system, such as performance, security, and usability.

**Requirement ID Format**

Each requirement is assigned an ID with three components, structured as follows:

* **Requirement Type**: A two-letter code indicating the nature of the requirement.
* FR: Functional Requirement
* NR: Non-Functional Requirement
* **Group Index**: A two-digit number representing the group or category.
* **Requirement Index**: A three-digit serial number for individual tracking within the group.

**Example**: FR-01-002 refers to a Functional Requirement under Group 01 (e.g. User Management), and it is the second requirement in that category.

All requirements are assigned a **priority level** from 1 to 3:

* **Priority 1**: Critical – Must be implemented.
* **Priority 2**: Important – Should be implemented.
* **Priority 3**: Optional – May be skipped if time does not allow.

**3.1 System Overview**

The purpose of the **Multi-Vendor Service Marketplace** system is to provide a robust and scalable platform that allows users to seamlessly interact with a network of service providers. The system enables vendors to register and offer their services to a broader customer base while customers can browse, filter, and book these services as per their preferences. The platform also empowers administrators to manage users, services, and platform-wide operations through a centralized dashboard.

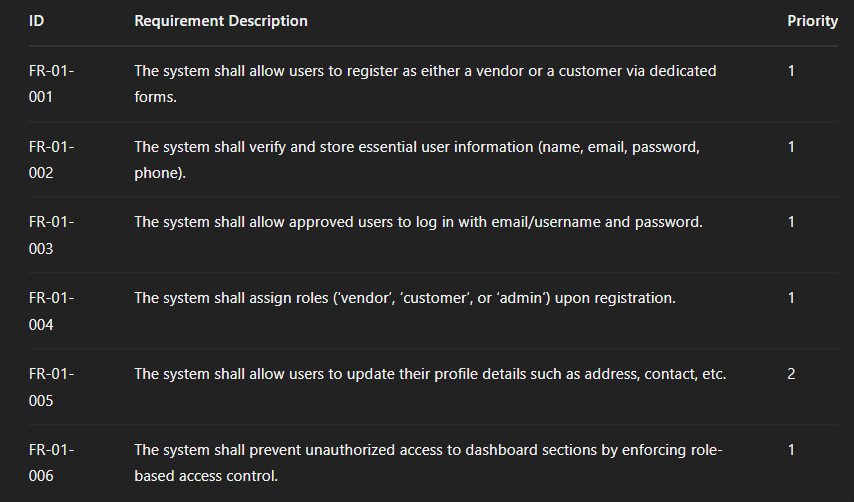
This platform offers a smooth onboarding experience for both vendors and customers, supports real-time order handling, and ensures secure financial transactions through integrated payment gateways. The entire ecosystem is designed to be modular, secure, and user-friendly—capable of scaling as the service demand and vendor base grow.

**3.2 Functional Requirements**

Functional requirements define the core capabilities and operations the system must perform to meet the expectations of its users. These requirements are categorized into logical groups, and each is assigned a unique identifier that aligns with the format specified in Figure 3-1 of this report (e.g., FR-01-001). Additionally, every requirement is associated with a priority level (1 = High, 2 = Medium, 3 = Low), allowing efficient resource allocation during implementation.

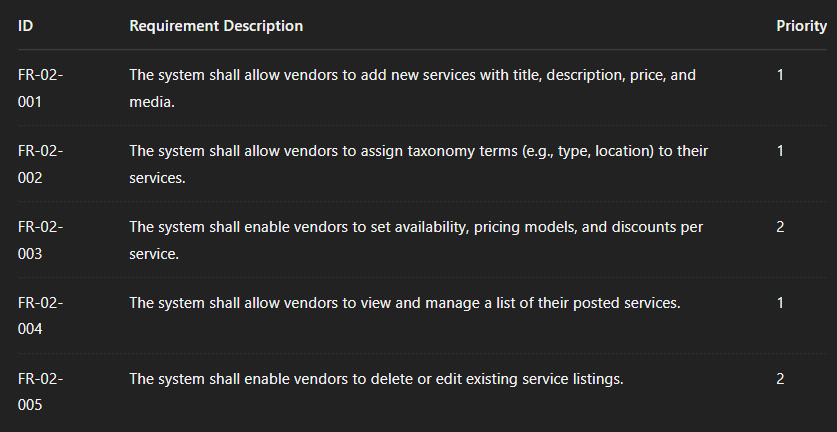
**3.2.1 Category 1 – User Management**

This category addresses user-related operations including registration, login, role assignment, and profile management.



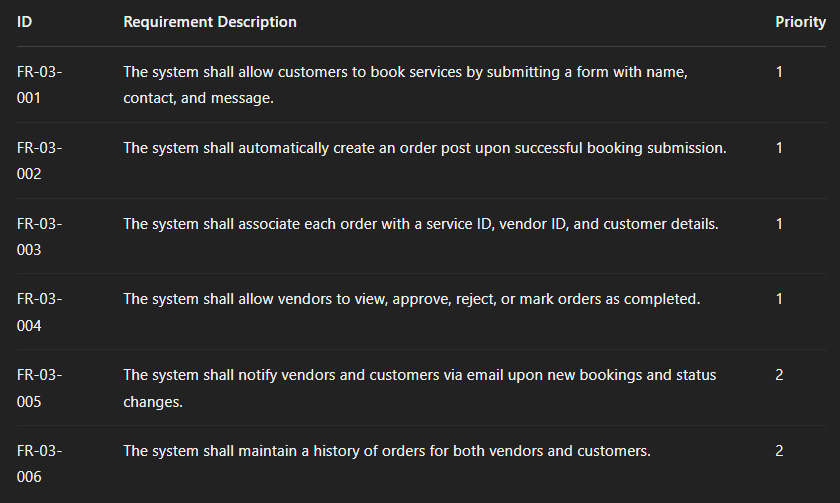
**3.2.2 Category 2 – Service Management**

This category focuses on vendor capabilities such as adding, editing, and managing service listings.



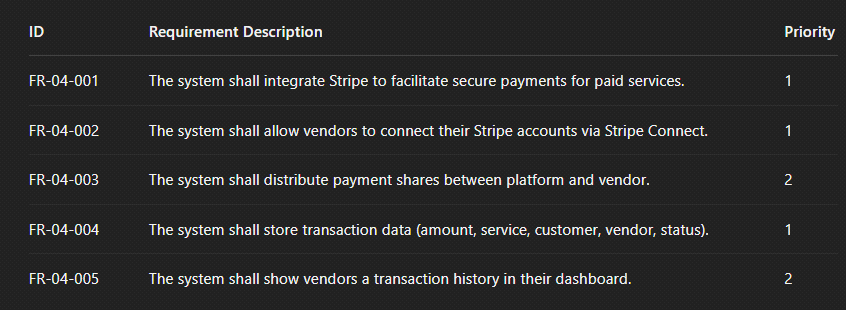
**3.2.3 Category 3 – Booking & Order Management**

This category outlines how customers interact with services through bookings, and how vendors manage received orders.



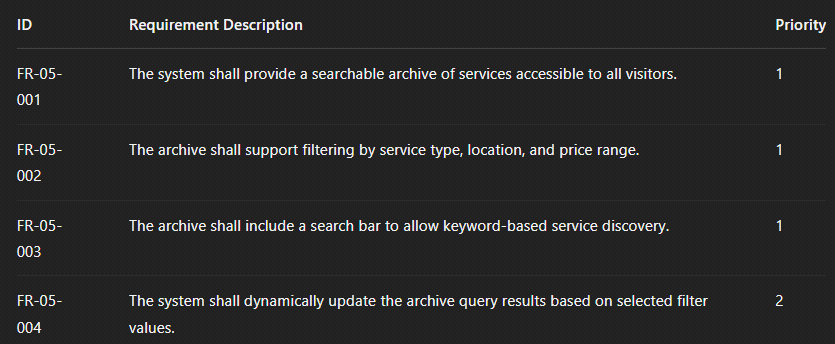
**3.2.4 Category 4 – Payment Integration**

This category includes functionality for processing payments through Stripe and managing transaction data.



**3.2.5 Category 5 – Search and Filtering**

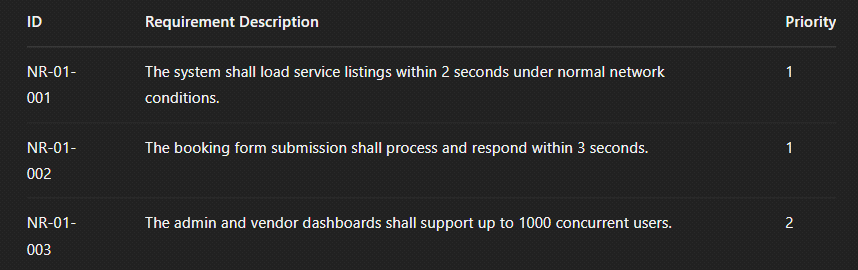
This category covers service discovery by users through dynamic filters and keyword search.



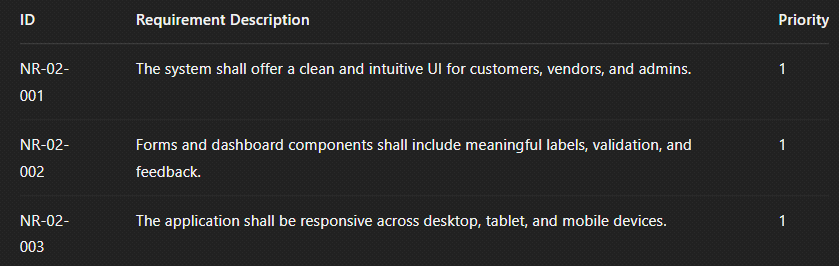
**3.3 Non-Functional Requirements**

Non-functional requirements (NFRs) define the quality attributes and operational constraints of the system rather than specific functionalities. These ensure the system is reliable, secure, and user-friendly under real-world usage.

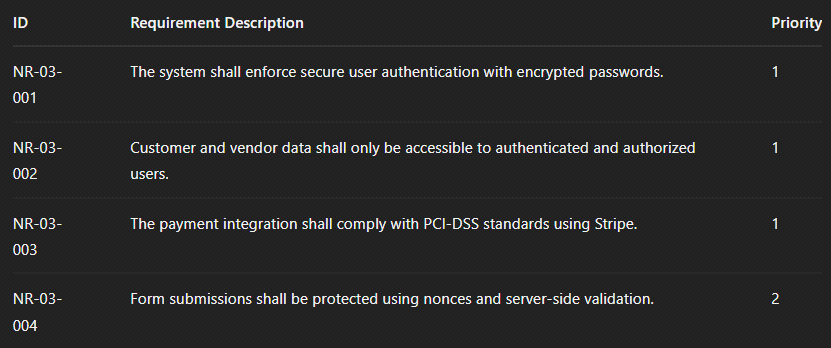
**3.3.1 Performance Requirements**



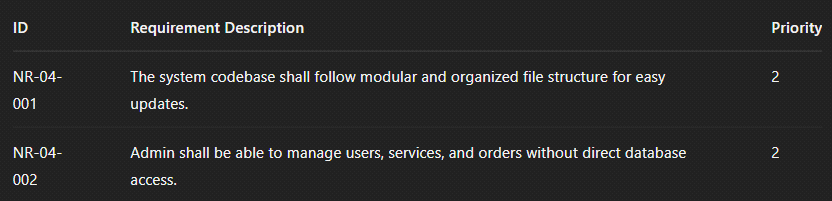
**3.3.2 Usability Requirements**



**3.3.3 Security Requirements**



**3.3.4 Maintainability Requirements**



**5.1 Tools and Technologies**

This project leverages a combination of widely-used technologies and platforms to provide an efficient, scalable, and secure web-based software solution. While the platform is independent and can be adapted to any CMS, for this specific implementation, it has been developed and tested using WordPress due to its robust ecosystem and widespread use for building dynamic websites.

**1. Development Environment**

The development of this platform was primarily carried out using Visual Studio Code (VS Code), a powerful and lightweight code editor, known for its extensive feature set and integration with version control systems like Git. GitHub was used to manage the project's source code, allowing for seamless version control and collaboration. The software was developed in a local environment using XAMPP and Laragon, both popular solutions for running PHP, MySQL, and Apache locally. This setup made testing, debugging, and local development easier and faster.

**2. Frontend Development**   
 The frontend was developed using HTML, CSS, JavaScript, and jQuery. Bootstrap was employed for responsive design, ensuring that the user interface is both aesthetically appealing and functional on a wide range of devices. To add more styling flexibility, Sass (Syntactically Awesome Stylesheets) was utilized to make the CSS code more maintainable and modular. These technologies ensured that the platform is easy to use, responsive, and accessible.

**3. Backend Development**  
The backend of the web-based software was developed using **PHP**, one of the most widely used server-side scripting languages in the industry. PHP was chosen due to its versatility, ease of integration with various systems, and its ability to handle server-side processing for dynamic web applications. This choice enables the application to handle tasks such as form submissions, user authentication, data processing, and server-side rendering of content.

For database management, **MySQL** was utilized, a reliable and efficient relational database management system. The database was administered via **phpMyAdmin**, a powerful and user-friendly interface that simplifies database creation, modification, and maintenance. This tool enabled smooth interactions with the database, including running queries, managing tables, and performing data backup and restoration operations.

In terms of data storage, the platform leverages the custom tables and custom tables to efficiently manage and store data related to the plugin. For example, user-specific data is stored in **wp\_usermeta**, which holds information about customers, vendors, and their respective roles and permissions. Configuration settings are stored in **wp\_options**, which provides a central location for managing various site-wide settings, such as API keys or default preferences.

The core plugin data, including **service listings**, **orders**, and other platform-specific entities, are stored in the **wp\_posts** and **wp\_postmeta** tables. The use of these custom tables ensures compatibility with the overall system, maintaining consistency across the application while benefiting from the built-in features of WordPress, such as post revisions, metadata storage, and custom post types.

In addition to using the custom tables, a custom database called **service\_marketplace** was created to store plugin-specific data. This custom database houses important data related to services, customer orders, payment details, and additional settings. By creating dedicated tables in the **service\_marketplace** database, the plugin is able to manage and track services offered by vendors, customer orders, and vendor-specific data separately, without conflicting with the core data. This also ensures that the plugin can scale and evolve without impacting the site's overall performance.

A key advantage of using custom tables within the existing database structure is that it allows for seamless integration while maintaining optimal performance. Custom queries can be run against the plugin’s tables, allowing for efficient retrieval and manipulation of the service and order data. Additionally, utilizing the default database structure enables the plugin to benefit from built-in security features, such as user permissions and role management, which helps in protecting sensitive data.

This approach ensures that the plugin’s data is well-organized, avoids redundancy, and integrates smoothly with the existing framework. It also makes use of wp’s flexible and powerful infrastructure, allowing the plugin to function as a cohesive and scalable web-based software solution.

By carefully selecting and structuring the database schema, the backend of the web-based software not only facilitates seamless data storage and retrieval but also ensures that the plugin performs optimally, even as it grows in complexity and user load.

**4. Payment Integration**  
For payment processing, the project integrates **Stripe**, a leading online payment service. Using **Stripe's API**, the system handles payments for service bookings securely and efficiently. Customers can pay directly through the platform, and their transactions are securely processed via Stripe’s infrastructure, ensuring a seamless payment experience.

**5. Email Configuration**  
The project uses **Google's SMTP server (smtp.gmail.com)** for handling email communications between customers, vendors, and administrators. This was integrated through **WP Mail SMTP**, a tool that ensures reliable email delivery, bypassing issues associated with WordPress’ default PHP mail function. Nonces are implemented to secure email requests and prevent unauthorized access to user data during interactions.

**6. Security Features**  
Security is of utmost importance in the development of this platform. All user inputs are carefully sanitized to prevent malicious actions such as **SQL injections**, **Cross-Site Scripting (XSS)**, and other common attacks that target web applications. By utilizing **prepared statements** and ensuring that only sanitized input is processed and stored in the database, the risk of data breaches and vulnerabilities is significantly reduced. This approach ensures that no malicious data can manipulate the database or execute unintended actions.

To further enhance the security of the platform, **nonces** are used to verify that a request has been intentionally made by a valid user. Nonces are unique tokens that are generated for each request, making sure that forms or actions are not submitted by unauthorized third parties. This prevents attacks like **Cross-Site Request Forgery (CSRF)**, where a malicious user could attempt to trigger an action on behalf of another user without their consent.

In addition to these protections, user passwords are stored securely using **password hashing techniques**. This ensures that even if a database breach were to occur, sensitive data such as user passwords remains encrypted and inaccessible. Only the hashed version of the password is stored, and during login, the entered password is hashed and compared with the stored hash, preventing the exposure of the original passwords.

The platform also makes use of **role-based access control (RBAC)** to manage user permissions effectively. By assigning specific roles to users—such as admin, customer, or vendor—the system ensures that users can only access data and features relevant to their role. This minimizes the chances of unauthorized access to sensitive data or administrative functions, effectively preventing privilege escalation attacks.

Further security measures include the implementation of **SSL/TLS encryption** to secure data transmitted between the user’s browser and the server, especially for sensitive information like passwords and payment details. All communication is encrypted to prevent interception by malicious entities.

To prevent brute-force attacks, measures like **login attempt throttling** and **two-factor authentication (2FA)** can be integrated, although they are not part of the initial setup. Additionally, regular **security audits** and the use of **firewalls** can be employed to detect and block suspicious activity, adding an extra layer of defense against external threats.

In summary, this platform incorporates robust security practices such as input sanitization, password hashing, nonces for action validation, secure data transmission, and role-based access control, ensuring that user data and system integrity are protected at all times.

**7. Local Development Environment**  
For local development and testing, **XAMPP** and **Laragon** were used to simulate a real server environment. These tools provide a robust local server with support for Apache, MySQL, and PHP, enabling smooth local development and testing. Once the platform is ready for deployment, it can be easily migrated to a live server environment.

**8. Version Control and Collaboration**  
The entire codebase was managed and stored in a **GitHub** repository, ensuring proper version control, collaboration, and traceability of code changes. GitHub also made it easier to collaborate with other team members, allowing for code review, feature branches, and issue tracking throughout the development process.

**9. Database Management**  
For managing the platform’s database, **phpMyAdmin** was used. It provided a user-friendly interface for managing database tables, running queries, and performing backups. The data related to services, users, orders, and other important details are stored in the default WordPress tables, ensuring seamless integration with the WordPress ecosystem.