CH 6
Table of Contents

Introduction

This document explores essential architectural design concepts, focusing on the role and organization of software architectures in system development. A well-defined architecture is crucial for managing system complexity, ensuring modularity, and supporting scalability and adaptability. This chapter examines three primary architectural patterns Layered Architecture, Pipe and Filter Architecture, and Client-Server Architecture each serving different functional needs and offering unique advantages and limitations. Understanding these architectures enables effective decision-making, allowing systems to be designed with a clear structure, maintainability, and flexibility for future expansion or modification.

Purpose

The purpose of this document is to analyze and justify the choice of client-server architecture as the most suitable structure for my project's requirements. This architecture's inherent flexibility, scalability, and support for distributed services provide a strong foundation for handling variable loads and accommodating a broad user base. By outlining the advantages and limitations of client-server architecture and comparing it to other architectural models, this document supports an informed design choice that aligns with the project's need to efficiently manage data, support concurrent users, and enable seamless functionality across multiple services.

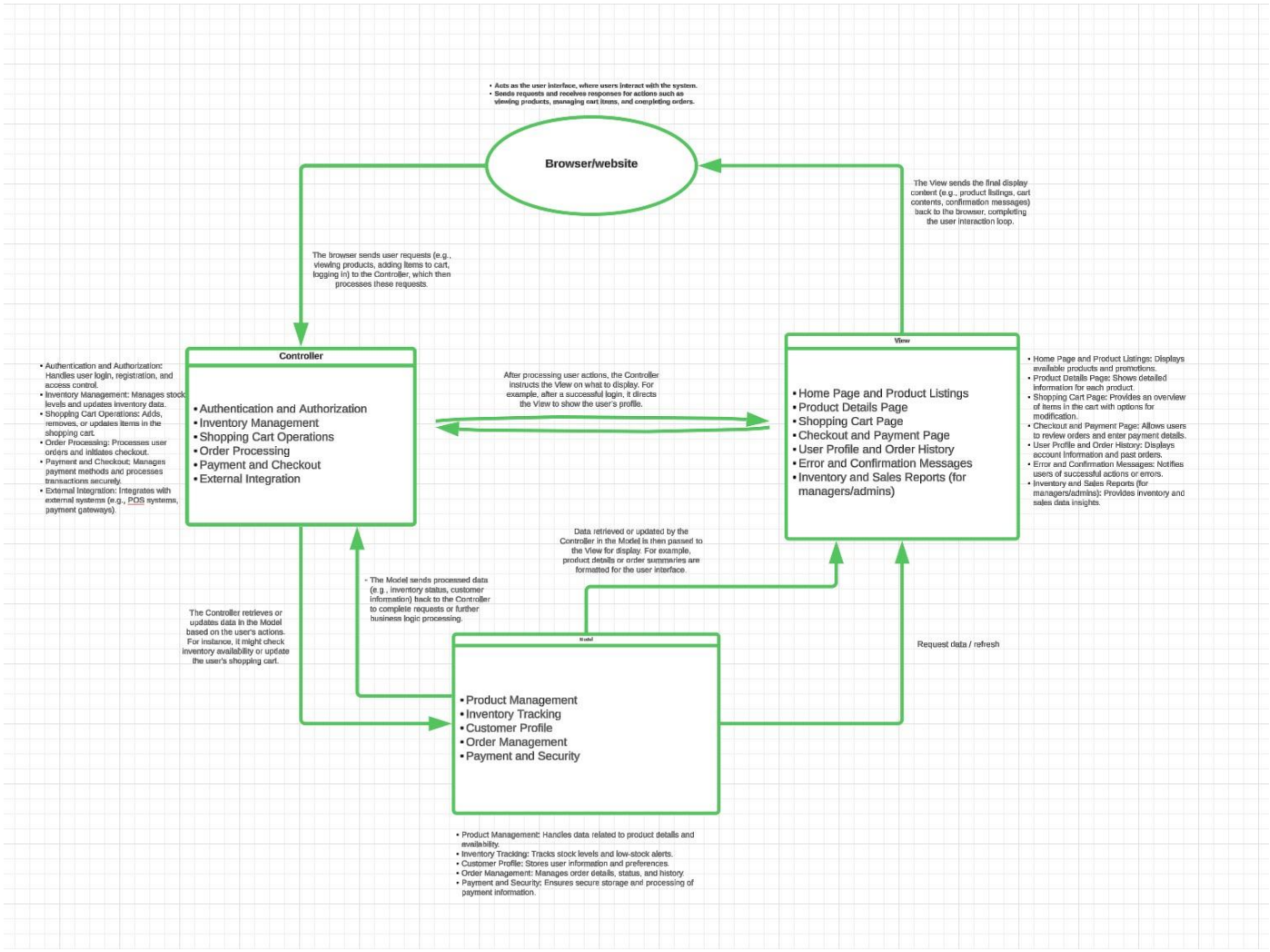
This MVC implementation demonstrates how the **Advanced Smart Inventory Stock Management System** is structured to manage interactions and streamline functionality. The diagram illustrates how the **Controller**, **View**, and **Model** components work together to create an efficient and well-organized application.

In this setup, the **Controller** handles essential actions such as adding items to the shopping cart, managing user logins and registrations, validating credentials, updating inventory quantities, and processing checkout requests. It serves as the central mediator, processing HTTP requests and coordinating between the View and Model to ensure smooth user interactions.

The **View** component manages the user interface, displaying elements such as the page layout, popular product widgets, shopping cart buttons, and language selection options. It responds to user input and presents relevant data from the Model, providing a seamless and user-friendly experience.

The **Model** is responsible for all data management, including the shopping cart details (product names, prices, quantities), the item database, and the customer database with secure user information. It communicates with the Controller to update or retrieve information, using hashing algorithms to protect sensitive data.

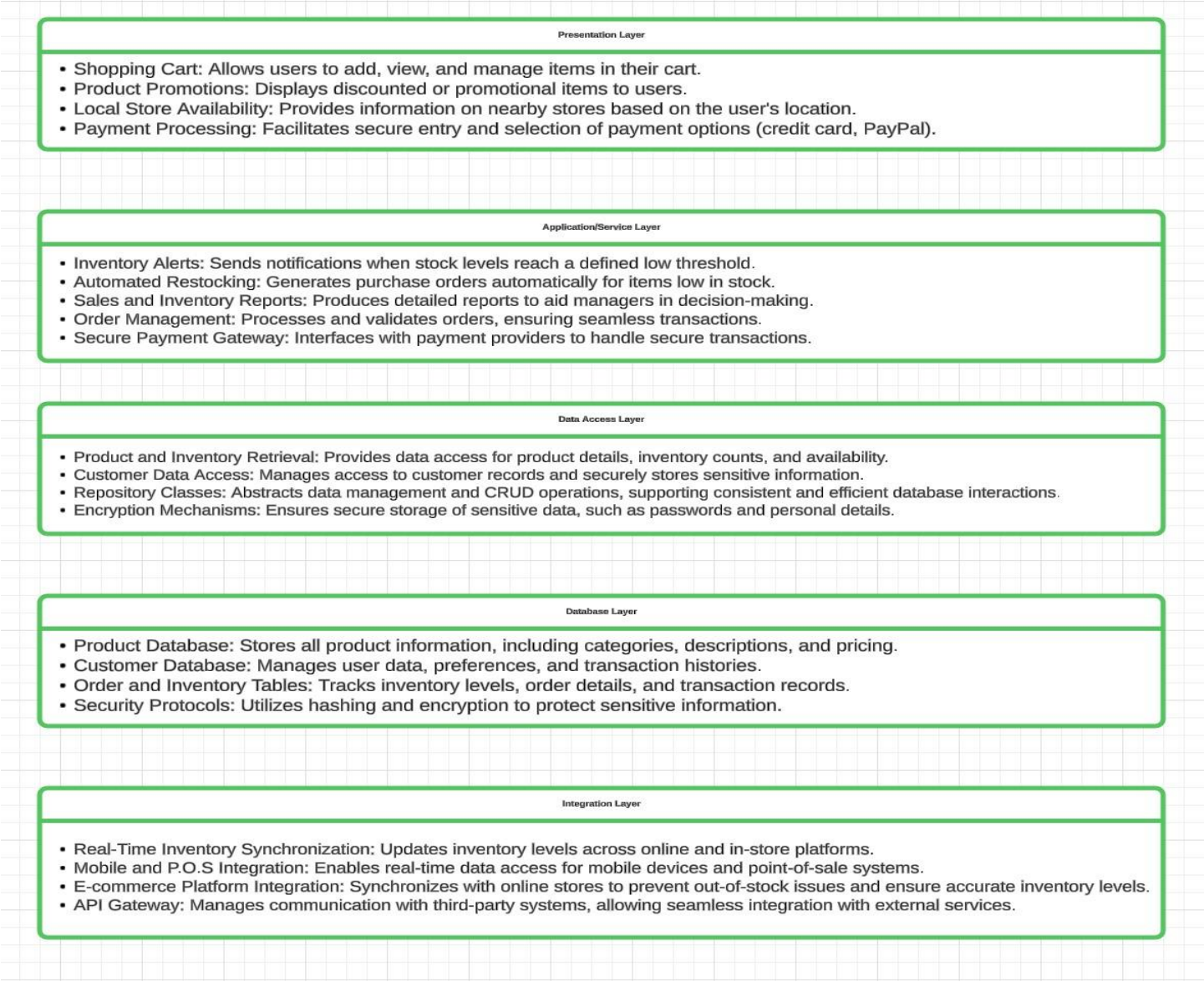
By following this MVC architecture, the system will achieves enhanced modularity, allowing each component to be updated independently without affecting the others. This approach enables the **Advanced Smart Inventory Stock Management System** to be scalable, maintainable, and focused on delivering a smooth user experience.



Software Architectures Layered Architecture

Layered architecture is ideal for structuring subsystems through defined layers, each providing a unique set of services. This structure supports incremental development, as changes are isolated to specific layers, limiting their impact on other parts of the system. Layered architecture is commonly applied when new functionality needs to be added to existing systems.

- **Advantages:**
  - Entire layers can be replaced if the interface remains unchanged.
  - Redundant components within each layer can improve system reliability.
  - Simplifies changes since all elements are centralized, enabling easy modifications.
- **Disadvantages:** o Maintaining clear boundaries between layers can be challenging due to complex layer interactions. o Performance may suffer due to multiple levels of interpretation across layers.



Pipe and Filter Architecture

Pipe and filter architecture utilizes a central repository for data, which all components can access without direct interaction among them. Often used in data-driven systems, it supports scenarios where substantial data must be stored and accessed over extended periods.

- **Advantages:**

  o Components operate independently and need not be aware of each other. o

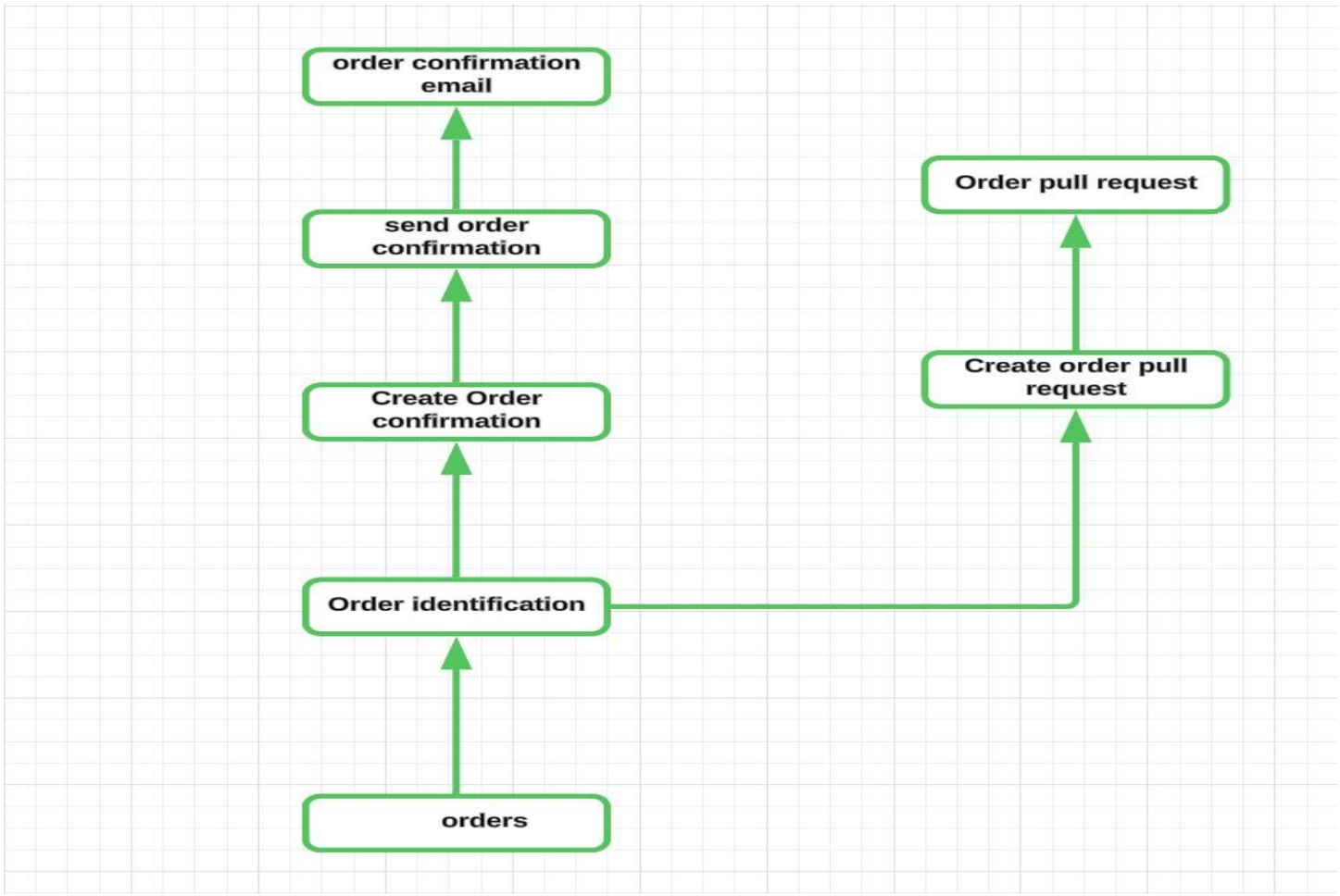
  Modifications by one component are reflected across all components.

  - Centralized data management ensures consistency.

- **Disadvantages:**

  The repository is a single point of failure, making the entire system vulnerable.

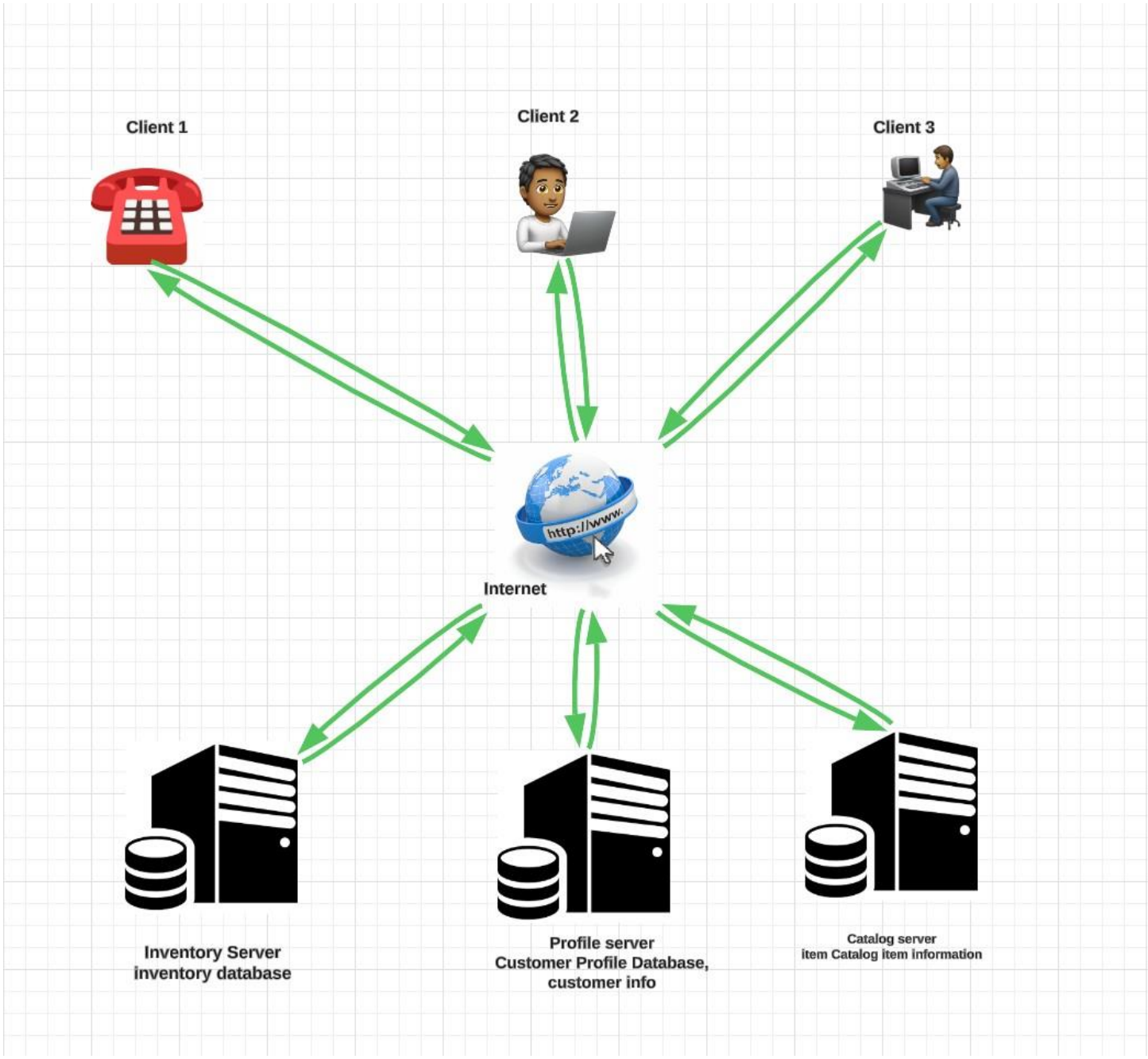
  - Communication through the repository may reduce efficiency.
  - Distributing the repository across components can be challenging.



Client-Server Architecture

In client-server architecture, services are structured as separate servers, with clients accessing these services as needed. This setup suits scenarios where data in a shared database must be accessible across different locations. Servers can be replicated to accommodate varying system loads.

- **Advantages:** o Servers can be distributed over a network. o Clients can access functionalities without needing to implement all services.
    - Supports data security, authorization, authentication, scalability, easy management, and accessibility.
- **Disadvantages:**
    - Each service introduces a single point of failure, increasing susceptibility to DoS attacks. o Network and system dependencies can lead to unpredictable performance. o Managing servers across organizations may pose difficulties.
    - The cost of setup and maintenance is high, and resources on the client's end are often limited, such as printing from the web or modifying data stored on the client's device.



Selection Justification

Client-server architecture best suits our project due to its adaptability with load management and range. With multiple users potentially engaging in various activities placing orders, creating accounts, updating payment details the architecture is well-suited for handling multiple requests from different sources concurrently. For instance, users checking inventory would connect to the server managing stock data, while others can use distinct services on separate servers without interference. Shared databases in client-server setups are advantageous for serving users across regions, allowing for server replication and load distribution. Although potential downsides include DoS vulnerabilities and server performance inconsistencies, the benefits especially flexibility and scalability far outweigh these concerns.

Reference

"**MVC Framework Introduction**." *GeeksforGeeks*, 8 July 2024, https://www.geeksforgeeks.org/mvc-framework-introduction/. Accessed 4 Nov. 2024.

GeeksforGeeks. "Layered Architecture in Computer Networks." *GeeksforGeeks*, https://www.geeksforgeeks.org/layered-architecture-in-computer-networks/. Accessed 4 Nov. 2024.

Microsoft Learn. "Pipes and Filters Pattern." *Microsoft Learn*, https://learn.microsoft.com/en-us/azure/architecture/patterns/pipes-and-filters. Accessed 4 Nov. 2024.

GeeksforGeeks. "Client-Server Model." *GeeksforGeeks*, https://www.geeksforgeeks.org/client-server-model/. Accessed 4 Nov. 2024.