

ÔNG NHỚ ĐỂ MẤY LINK DRAWIO CỦA ÔNG VÀO FOLDER CÁ NHÂN TASK 3 NHA

TIỆN VÀO XEM HƠN Á :3

Ngoài ra thì bên dưới, những cái nào khi ông viết ra (ví dụ description đồ) mà có nhắc tới tên mà có trong diagram (giả dụ như “User View” thì từ “**User View**” ông in đậm lên nha)

# III. Architecture Design

## 1. MVC Architecture

The team has chosen the MVC (Model - View - Controller) architecture, one of the most popular architectures, to design the HCMUT\_SPSS system.

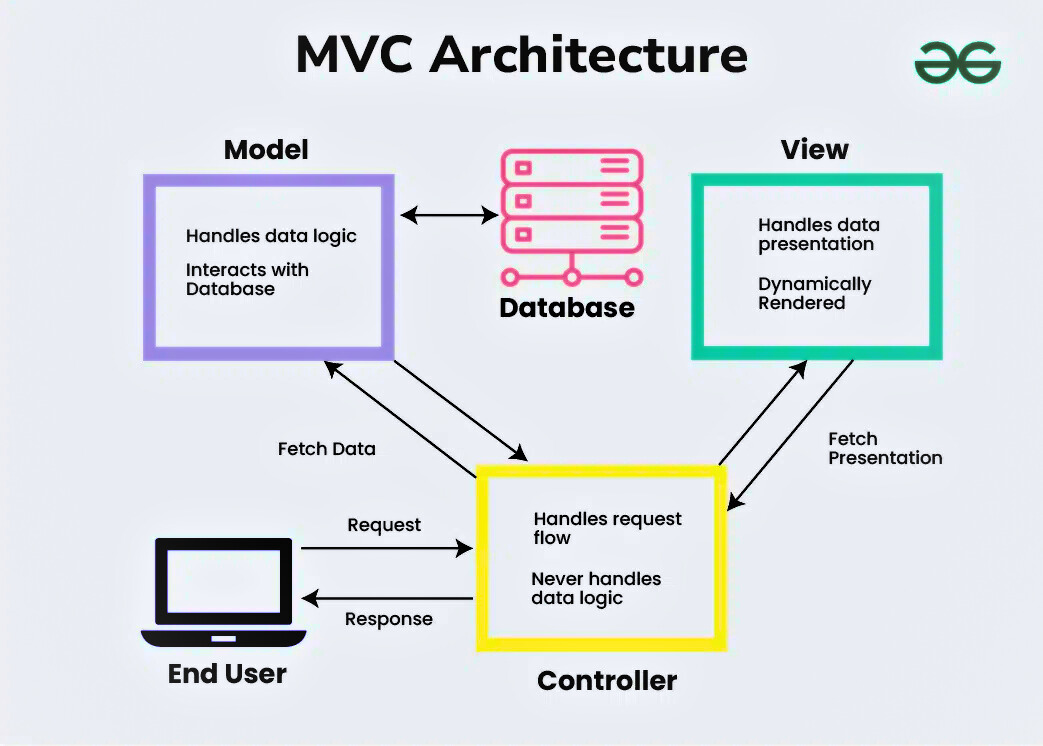


Figure ..: MVC Architectural Model (Source: GeeksforGeeks.org)

### 1.1. Advantages and Disadvantages of MVC

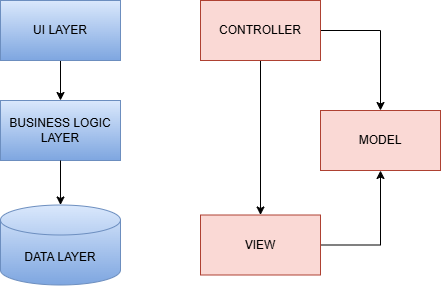
**Advantages of MVC:**

1. **Clear Separation of Responsibilities**: MVC divides the application into three distinct components - Model, View, and Controller - each with well-defined roles. This separation simplifies code organization, making it easier to manage and maintain.
2. **Enhanced Scalability and Maintainability**: By isolating each component, the MVC architecture allows for easy modifications to individual parts without impacting the entire system. This modularity promotes scalability and long-term maintenance as the application evolves.
3. **Parallel Development Capability**: The MVC pattern allows multiple developers to work on different layers simultaneously - one can work on the View while another focuses on the Controller, and a third works on the Model. This concurrent development reduces overall development time and prevents code conflicts.
4. **Flexible Integration of New Features**: MVC supports easy swapping or modification of components. Developers can add or replace features with minimal impact on the other parts of the application, facilitating smooth integration of new functionalities.
5. **Improved Testability**: MVC’s modular structure enables testing of individual components without having to execute the entire application, which enhances testing efficiency and helps identify issues early.
6. **Reusable Components**: MVC encourages the development of reusable components. Views and Models, for instance, can often be repurposed in other applications, improving development efficiency and consistency across projects.

**Disadvantages of MVC:**

1. **Increased Complexity**: The MVC architecture can introduce additional complexity, as developers need to manage the relationships and data flow between Models, Views, and Controllers.
2. **Challenging for Large Applications**: As the application scales, the number of Model, View, and Controller files grows significantly. Without a well-structured organization, managing and navigating these files can become cumbersome, impacting maintainability.
3. **Higher Learning Curve for New Developers**: For developers unfamiliar with MVC, the structure may be challenging to grasp at first, especially in understanding how each component interacts and the conventions that must be followed.
4. **Potential for Performance Overhead**: In applications where each component frequently communicates with the others, there can be performance overhead due to the additional layers of abstraction. This may require optimization to maintain efficient performance.
5. **Increased Initial Development Effort**: Setting up an application in MVC involves creating and configuring separate components, which can require more initial development time and effort compared to simpler, monolithic architectures.

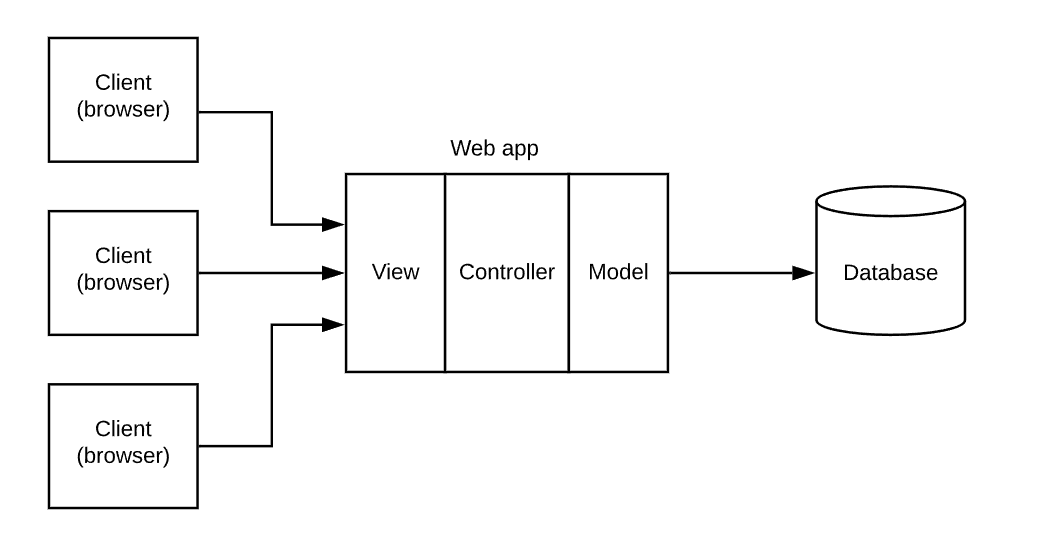
### 1.2. Reasons for Choosing MVC over Layered Architecture



Figure…: Difference between 3-tier and MVC architecture

After analyzing the MVC and Layered Architecture models, the team opted for MVC for the following reasons:

* ***Easy Task Allocation***: The MVC architecture divides the application into three main components—Model, View, and Controller. This separation simplifies task distribution within the team, enabling each member to focus on a distinct component without overlapping responsibilities. Unlike the Layered Architecture, where each layer depends sequentially on the lower layers, MVC allows for clear task delineation, reducing interdependence among team members.
* ***Concurrent Development****:* MVC supports concurrent development as the Model, View, and Controller can be developed independently, promoting collaboration among multiple developers. This is especially advantageous for medium-scale projects like the SPSS system, where parallel workstreams can reduce development time. In contrast, the Layered Architecture’s dependency chain between layers makes simultaneous work challenging, as testing upper layers often requires completed lower layers.
* ***Support from Frameworks and Libraries:*** MVC is widely supported by a variety of frameworks and libraries, such as ASP.NET, AngularJS, and Ruby on Rails, making it easier to adopt and implement for modern web applications. This broad support accelerates development and provides readily available resources, community support, and documentation. While the Layered Architecture has its advantages in enterprise-level applications, MVC's popularity makes it an efficient choice for smaller, adaptable projects like SPSS.
* ***Adaptability for Project Scale***: The SPSS project is a medium-sized system, where MVC’s structure is particularly suitable. MVC offers flexibility by allowing components to be developed and tested individually, which simplifies management and reduces complexity. In a Layered Architecture, the tight dependency among layers could complicate updates or modifications, which is less desirable for a system that requires agility.



Figure…: Web MVC Architecture

* ***Learning and Practical Application***: Given that MVC is a commonly used architecture pattern in web development, implementing it in the SPSS project is an excellent opportunity for the team to deepen their skills. This familiarity with MVC prepares team members for future projects, as many organizations utilize MVC for its balance of separation and integration of concerns.
* ***Scalability and Flexibility***: MVC allows for straightforward scalability and adaptation to new requirements. Each component (Model, View, Controller) can be modified independently as new features or changes are introduced. This is particularly beneficial for projects expected to evolve over time, as the SPSS system might. The Layered Architecture can also be scalable, but often at the cost of complex refactoring when requirements change.
* ***Efficient Handling of Data and User Interaction***: In MVC, the Controller acts as a mediator between the user input (View) and the data (Model), handling requests and passing data seamlessly. This clear interaction path enhances responsiveness and user experience. In a Layered Architecture, user requests might need to pass through multiple layers, which can introduce latency and increase processing complexity.

By selecting MVC over Layered Architecture, the team aligns the SPSS project’s goals with an architecture that supports efficient development, task separation, adaptability, and real-time user interaction—qualities that are essential for a responsive and scalable application.

## 2. Architectural Diagram and Deployment Diagram

### 2.1. Architectural Diagram for the overall design of HCMUT-SSPS system

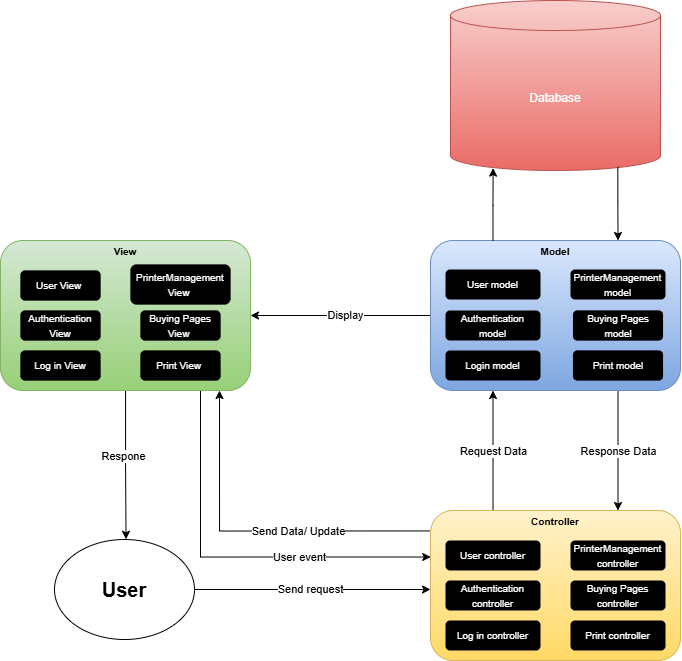


Figure …: Architectural diagram

**Presentation Strategy**

The presentation strategy for the HCMUT-SSPS system emphasizes providing a user-friendly and intuitive interface through distinct View components. Core screens such as User View, Authentication View, Log in View, and Printer Management View allow students to easily navigate and access functions like account management, printing services, and page purchasing. By clearly delineating each interface, the system enhances user experience and reduces the time required to locate information. Using frontend frameworks like React or Angular would further streamline the development of these Views due to their flexibility and efficiency.

Link: [Reference](https://reactjs.org/).

**Data Storage Approach**

In designing the data storage strategy for the HCMUT-SSPS system, the team is undertaking a comparative evaluation of **SQL** and **NoSQL** databases to identify the most suitable solution. The SQL option under consideration is **MySQL**, a well-established relational database management system (RDBMS), whereas **MongoDB** represents the NoSQL category as a document-oriented database. Below, we outline the primary advantages and limitations of each approach, followed by a conclusion based on their alignment with the system’s data requirements.

***SQL (MySQL)***

* **Advantages**:
* **Data Integrity and ACID Compliance**: MySQL ensures adherence to ACID properties (*Atomicity, Consistency, Isolation, and Durability*). Thereby providing a high degree of data reliability, essential for applications requiring precise transactional integrity.
* **Structured Data Management**: As a relational database, MySQL is optimized for handling structured data within a well-defined schema. It is particularly effective for applications that require complex join operations and precise data relationships.
* **Mature Ecosystem and Tooling**: MySQL, as a widely adopted SQL database, benefits from a robust ecosystem of tools, extensive documentation, and a supportive community. This maturity facilitates integrations, maintenance, and reliable performance monitoring.
* **Limitations**:
* **Scalability Constraints**: MySQL typically relies on vertical scaling, which may not be optimal for applications that need to scale horizontally. This limitation can pose challenges when handling rapidly expanding data volumes or data that require distributed storage across multiple nodes.
* **Schema Rigidity**: MySQL’s strict schema enforcement can hinder flexibility, as altering table structures can be complex and time-consuming. This constraint may become a disadvantage in systems where data requirements are dynamic and subject to frequent changes.

***NoSQL (MongoDB)***

* **Advantages**:
* **Horizontal Scalability and High Availability**: MongoDB is designed with horizontal scaling capabilities, allowing it to distribute data across multiple servers (sharding). This design is well-suited for handling large and continuously growing datasets in distributed environments.
* **Schema Flexibility and Agility**: MongoDB’s schema-less structure permits flexible document-based storage, making it ideal for storing heterogeneous data that may not conform to a rigid schema. This flexibility is advantageous in rapidly evolving systems where data models may change frequently.
* **Optimized for Large Datasets and High-Performance Read/Write Operations**: MongoDB’s document-oriented storage architecture enables it to efficiently handle high-velocity data and large datasets. Its architecture facilitates fast read and write operations, enhancing performance in applications that process substantial volumes of unstructured or semi-structured data.
* **Alignment with Agile Development Practices**: MongoDB’s adaptability to store complex, hierarchical data structures directly within documents aligns well with agile development methodologies, supporting rapid prototyping and iterative development cycles.
* **Limitations**:
* **Potential for Reduced Consistency**: MongoDB, like many NoSQL databases, operates on the principles of eventual consistency, prioritizing availability and partition tolerance. As a result, it may not provide the same level of transactional integrity as MySQL, which could be a drawback for applications requiring strict data consistency.
* **Limited Support for Complex Join Operations**: Unlike relational databases, MongoDB is less suited for complex querying involving multiple collections (equivalent to tables). This limitation could impact analytical workloads that require intricate data joins or aggregations.

***Conclusion***

Based on the comparative analysis, MongoDB appears more suitable for the HCMUT-SSPS system’s anticipated data needs. The system is expected to handle diverse, potentially unstructured data at a significant scale, making MongoDB’s schema flexibility, horizontal scalability, and efficient handling of large datasets advantageous. Consequently, the team is inclined to proceed with MongoDB in the subsequent stages of implementation, recognizing that its adaptability will support both the system's current requirements and its projected growth.

**API Management**

API management in the HCMUT-SSPS system will follow **RESTful standards**, ensuring seamless and structured communication between **Controller** and **Model** components, as depicted in the diagram. RESTful principles enable the APIs to provide well-defined endpoints that facilitate data retrieval and updates across different system components. Each endpoint is organized around specific resources, aligning with HTTP methods such as GET, POST, PUT, and DELETE, which handle operations like fetching user profiles, updating printer settings, or logging print activities.

To secure these APIs, the system will implement robust authentication protocols, such as **JWT (JSON Web Token)** or **OAuth2**, to ensure that only authorized users can access sensitive data and perform specific actions. This security layer prevents unauthorized access and enhances data integrity across the **Controller** and **Model** layers.

Furthermore, the RESTful standards promote **stateless interactions**, meaning each API request from the client to the server is independent, carrying all necessary information without relying on previous interactions. This approach allows for easier scalability and faster response times, as no session state is stored on the server. The APIs will be further optimized to minimize latency, employing techniques such as caching for frequently requested data and reducing payload sizes, which will enhance overall performance and user experience in real-time interactions.

**List of Models in the Diagram:**

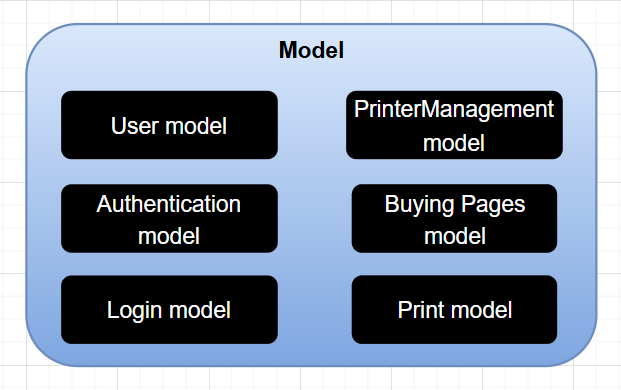


Figure: List of Models

* **User Model** - Manages information and data related to users.
* **Authentication Model** - Handles functions related to user authentication.
* **Log in Model** - Manages data associated with user login activities.
* **Printer Management Model** - Manages information and the status of printers.
* **Buying Pages Model** - Processes information related to purchasing additional printing pages.
* **Print Model** - Manages data related to printing requests.

These models support various functionalities of the HCMUT-SSPS system.

**List of Views in the Diagram:**

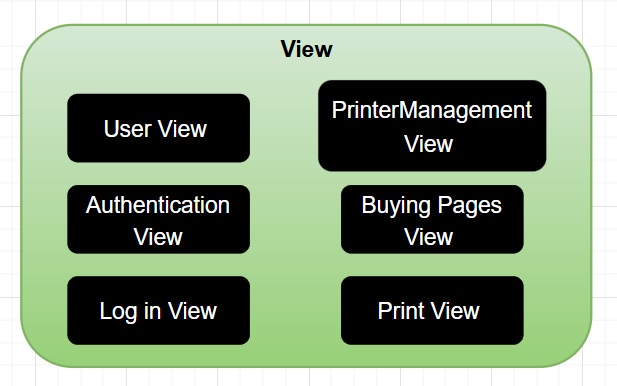


Figure x: List of Views

* **User View** - Displays information and interface elements related to user data.
* **Authentication View** - Interface for user authentication.
* **Log in View** - Interface for user login to the system.
* **Printer Management View** - Interface for managing printer details and status.
* **Buying Pages View** - Interface for purchasing additional printing pages.
* **Print View** - Interface for executing and managing print requests.

Each of these Views is designed to provide user access to different functionalities of the HCMUT-SSPS system.

**List of Controllers in the Diagram:**

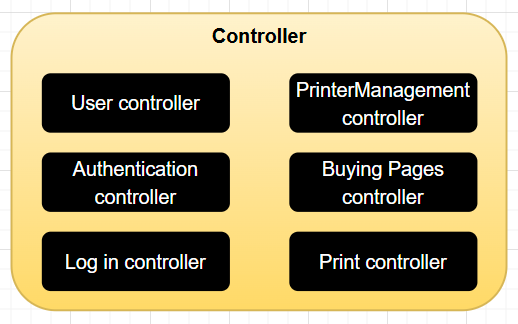


Figure: List of Controllers

* **User Controller** - Controls functionalities related to user management.
* **Authentication Controller** - Manages activities related to user authentication.
* **Log in Controller** - Handles requests related to user login.
* **Printer Management Controller** - Controls printer management processes.
* **Buying Pages Controller** - Manages requests related to the purchase of additional printing pages.
* **Print Controller** - Controls and manages print request operations.

The Controllers coordinate with the Views and Models to ensure smooth and efficient system operations within the HCMUT-SSPS.

### **2.2.** **Deployment** D**iagram for the overall design of HCMUT-SSPS system**

#### 2.2.1 Deployment Diagram

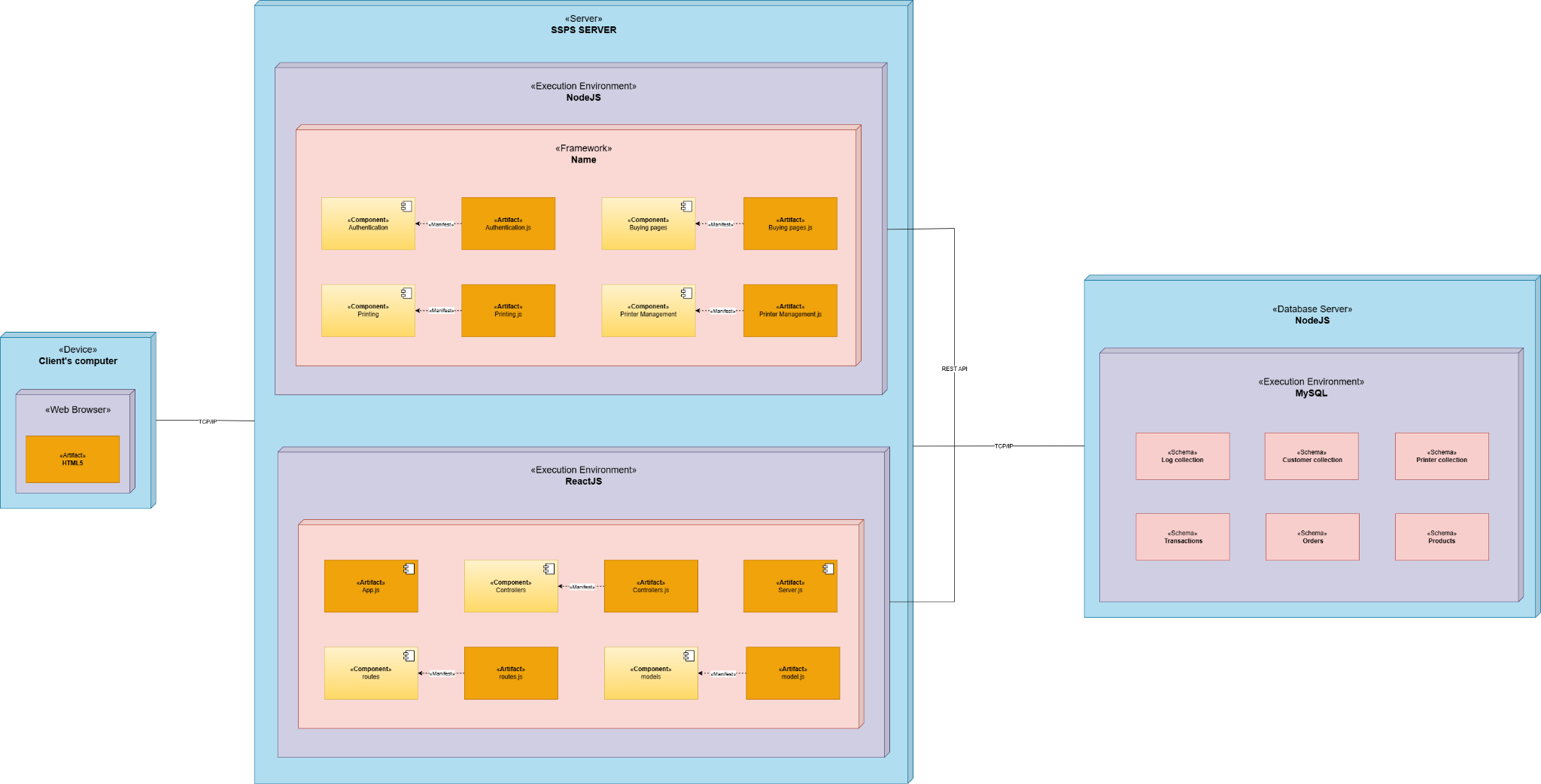


Figure …: Deployment diagram

#### 2.2.2. Deployment Diagram for description

The deployment diagram for the Student Smart Printing Service (SSPS) system illustrates the allocation of components across distinct hardware and software environments. This architecture is structured into three primary nodes: **Client’s Computer**, **SSPS Server**, and **Database Server**. Each node hosts specific components and execution environments, contributing unique roles to fulfill the overall system functionality.

**Client’s Computer**

**Role**: The Client’s Computer serves as the user interface, facilitating interactions between the user and the SSPS system.

* **Components**:
* **Web Browser**: Acts as the main access point, allowing users to connect to the SSPS system and initiate interactions such as logging in, managing print jobs, and purchasing additional pages.
* **HTML5 Artifact**: Represents the front-end interface rendered within the web browser, providing users with a responsive and accessible application experience.
* **Communication Protocol**: The Client’s Computer connects to the SSPS Server via the **TCP/IP** protocol. Through this setup, user requests (e.g., login or print commands) are transmitted to the server, which processes them and returns responses (e.g., printing status updates) for display on the client-side interface.

**SSPS Server**

**Role**: The SSPS Server is responsible for handling application logic, managing user interactions, and coordinating the system's printing functionalities. It is divided into two primary execution environments to support both back-end processing and front-end interface logic.

***NodeJS Environment (Back-End)***

* **Purpose**: This environment handles the core back-end functionalities, such as user authentication, print job management, and page purchasing processes.
* **Framework**: The back-end processing leverages the **Express.js** framework, which is instrumental in developing RESTful APIs and managing routing efficiently.
* **Components**:
* **Authentication Component**: Implements user authentication processes, ensuring secure access to the system, and is represented by Authentication.js.
* **Buying Pages Component**: Manages the purchasing of additional printing pages for users, implemented within BuyingPages.js.
* **Printing Component**: Oversees the processing and management of print jobs, represented by Printing.js.
* **Printer Management Component**: Handles configuration and monitoring of printer status, contained within PrinterManagement.js.
* **Implementation**: Each component in this environment is encapsulated in a JavaScript artifact (file) dedicated to handling specific back-end operations. This modular approach enhances code maintainability, allowing each component to perform discrete tasks without interdependency.

***ReactJS Environment (Front-End)***

* **Purpose**: The ReactJS environment manages client-side logic, including data routing and front-end data modeling, to support a dynamic and interactive user interface.
* **Framework**: The front-end environment utilizes **Material UI** alongside ReactJS to design a cohesive, accessible, and responsive user interface.
* **Components**:
* **Controllers Component**: Encapsulates front-end control logic, providing business logic processing for each module, contained in Controllers.js.
* **Routes Component**: Manages application routing, defining pathways between different views, implemented within Routes.js.
* **Models Component**: Defines front-end data models to interact with and represent the server-side data, implemented as Models.js.
* **App Component**: Acts as the main entry point for the ReactJS application, configuring Express and connecting routes, represented by App.js.
* **Server Component**: Facilitates communication between the front-end and back-end environments, implemented in Server.js.
* **Implementation**: Each front-end component is implemented as a modular JavaScript file, providing a structured and organized approach to managing the client-side application logic.
* **Communication**: The SSPS Server establishes communication with the Database Server through **REST API** over the TCP/IP protocol. This RESTful architecture supports modular interactions between the server and the database, ensuring data consistency and efficiency in data handling operations.

**Database Server**

**Role**: The Database Server is responsible for storing and managing all system data, including user profiles, printing logs, printer configurations, and transactional data.

* **Environment**: The SSPS system utilizes **MongoDB Atlas Cloud Database** as its primary data storage solution, chosen for its scalability, high availability, and remote accessibility.
* **Database Features**:
* **Remote Connection**: The MongoDB Atlas cloud environment facilitates remote data management, removing the necessity for on-premises storage and enabling distributed access for enhanced scalability.
* **Data Management**: MongoDB Atlas provides comprehensive data management tools, allowing administrators to view, query, and delete data directly from the online interface, supporting flexible data manipulation.
* **Schemas**: The database organizes data into various collections, including Log Collection, Customer Collection, Printer Collection, Transactions, Orders, and Products. Each collection is structured to support specific data types required by the SSPS system and to maintain relational consistency among data elements.