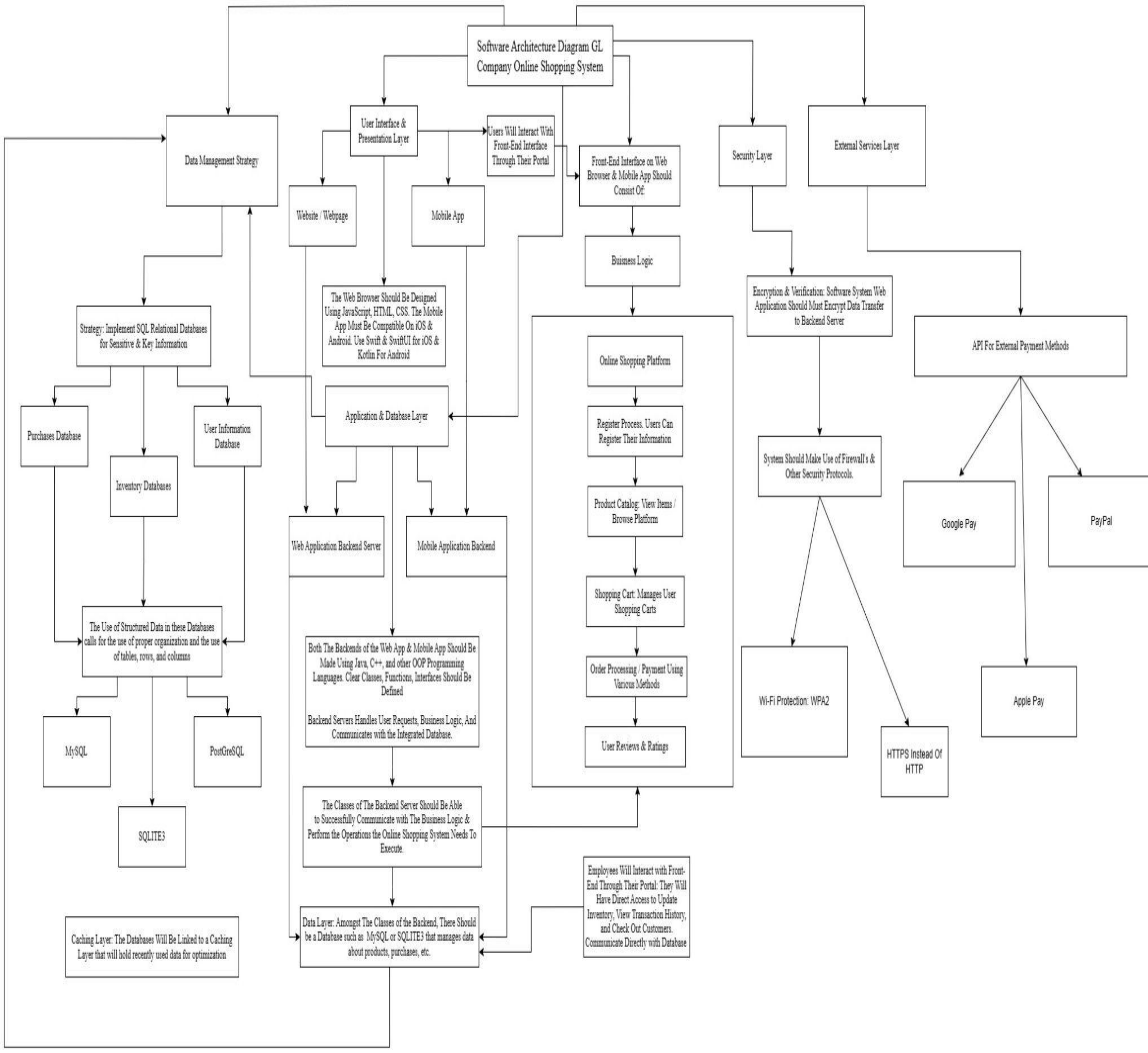
Software Design 2.0: Online Shopping System

Aman Ambastha, Kyle Cristobal, Samantha Velazquez

Software Design Specification

Link: Architectural Diagram (Clicking on the link and opening with diagrams.net would be better)


Software Architecture Diagram GL Company Online Shopping System
Data Management Strategy
User Interface & Presentation Layer
Users Will Interact With Front-End Interface Through Their Portal
Front-End Interface on Web Browser & Mobile App Should Consist Of:
Security Layer
External Services Layer
Website / Webpage
Mobile App
Buisness Logic
The Web Browser Should Be Designed Using JavaScript, HTML, CSS. The Mobile App Must Be Compatible On iOS & Android. Use Swift & SwiftUI for iOS & Kotlin For Android
Encryption & Verification: Software System Web Application Should Must Encrypt Data Transfer to Backend Server
API For External Payment Methods
Strategy: Implement SQL Relational Databases for Sensitive & Key Information
Online Shopping Platform
Application & Database Layer
Purchases Database
User Information Database
Register Process. Users Can Register Their Information
System Should Make Use of Firewall's & Other Security Protocols.
Inventory Databases
Product Catalog: View Items / Browse Platform
Google Pay
PayPal
Web Application Backend Server
Mobile Application Backend
Shopping Cart: Manages User Shopping Carts
The Use of Structured Data in these Databases calls for the use of proper organization and the use of tables, rows, and columns
Both The Backends of the Web App & Mobile App Should Be Made Using Java, C++, and other OOP Programming Languages. Clear Classes, Functions, Interfaces Should Be Defined
Backend Servers Handles User Requests, Business Logic, And Communicates with the Integrated Database.
Order Processing / Payment Using Various Methods
Wi-Fi Protection: WPA2
Apple Pay
MySQL
PostGreSQL
User Reviews & Ratings
HTTPS Instead Of HTTP
SQLITE3
The Classes of The Backend Server Should Be Able to Successfully Communicate with The Business Logic & Perform the Operations the Online Shopping System Needs To Execute.
Employees Will Interact with Front-End Through Their Portal: They Will Have Direct Access to Update Inventory, View Transaction History, and Check Out Customers. Communicate Directly with Database
Caching Layer: The Databases Will Be Linked to a Caching Layer that will hold recently used data for optimization
Data Layer: Amongst The Classes of the Backend, There Should be a Database such as MySQL or SQLITE3 that manages data about products, purchases, etc.

Description Of Architectural Components / Attributes:

1. **User Interface & Presentation Layer**: The front-end interface where users will communicate with the online shopping system will comprise a webpage/website and a Mobile App compatible with both Android & iOS Operating Systems. The user interface should be visually appealing to its customers. The software development and engineering team should use programming languages such as HTML, CSS, and JavaScript for the web page user interface. For the mobile application, the software developers should use Swift programming language for the app on iOS devices while using Kotlin for Android-based devices.

2. **Business Logic Layer:** The software system users should be able to do the following activities on the web and mobile apps. Users should be able to make accounts by registering with GL Company, browsing the inventory, adding items to their shopping cart, making purchases, and leaving reviews and ratings about their experience on the website/mobile app. The front-end and back-end of the software system should collectively be able to support these actions by the user.

3. **Application Backend Layer:** The backends of both the website and mobile app must be programmed using OOP Programming principles and OOP languages such as Java, C++, Python, and other programming languages. Clear classes, functions, and interfaces should be accurately described and explained. The classes, interfaces, and functions aim to handle the business logic user requests and communicate with the integrated database. Several classes make up the backend server. More details about the classes and interfaces are provided in the UML Diagram in the next section as the description of classes and operations.

4. **Database Layer / Database Management Strategy:** An SQL database such as MySQL, PostgreSQL, or SQLITE3 should be used and integrated within the backend server and the classes. The database is responsible for keeping track of product information, transaction history, and registered users. Employees can log in through the user interface to their portal and be able to access, change, and update information stored in the database. Employees can directly have communication access to the database. Along with the database later, an extra caching layer could be added that helps with accessing frequently used data quicker to improve optimization and performance.

    - **Caching Layer**: The caching layer will store the most commonly used/accessed data so that the backend server does not always need to retrieve information from the databases and instead relies on the caching layer. This can improve the server's performance by enhancing time efficiency and memory usage.

- **Database Structure:** The structure of the databases will be mainly SQL to store the data in a logical and orderly fashion, given that the database contains sensitive and private information. More details on why we chose SQL over NoSQL and other alternative database forms are provided in the section below.

5. **Security Layer:** The software system should have enhanced security measures and protocols to protect data transferred from the backend to the frontend and vice versa. Having data encryption and verification is very important for the software system to implement. Examples of secure measures include using HTTPS instead of HTTP to establish a secure connection between the web server and the website. Other examples include using firewalls and using WPA2 Wifi Access. Ensure that the website is only accessible to people with safe and secure internet connections.

6. **External Services Layer:** The above diagram explains the most critical architectural components of the software system; a possible extension of the current architectural setup is using advanced API to use external payment gateways and methods via third-party services. Payment is made through the backend server using the getAmount() and voidPurchase() methods. External third-party services, including integrating PayPal, Google Pay, and Apple Pay, could provide users with more payment options for their items.

<u>Data Management Strategy</u>

1) **Database Organization (SQL/NoSQL):** After much consideration, our team ultimately decided on organizing our systems' database using SQL. With the system handling clothing store purchases and inventory it made sense to implement three databases: inventory, user information, and purchases. Each database has their own respective diagrams pictured below.

Inventory:

ID	Item Name	Size	Price	Stock
0001	Green Hoodie	M	$40	50
0002	Blue Hat	One Size	$15	12

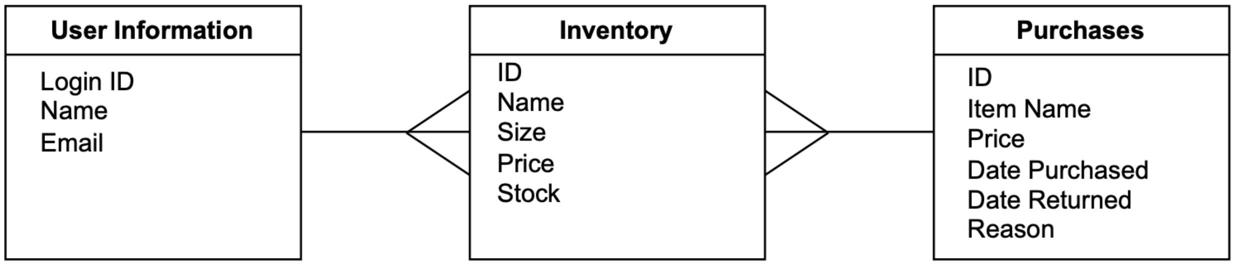
User Information:

Login ID	Name	Email
0001	Billy Bob	billybob01@gmail.com

Purchases:

ID	Item Name	Price	Date Purchased	Return Date	Reason
0002	Blue Hat	$15	11/10	N/A	N/A
0008	Black T-Shirt	$15	08/01	08/05	Wrong Size

2) **Databases Used & How Is The Data Split Logically:** Now pictured is how the databases would interact with one another throughout the systems' process. As discussed before the goal of the system is to efficiently guide users through their shopping experience, nonetheless the database would aid in the process. Therefore, it only made sense to implement three databases as the bulk of the data the system handles consists of three subjects: users, inventory and purchases, which as we established previously is only viewable by the administrator. Essentially, the databases interact with each other much in the same way information would be passed in the real world setting of shopping in-store, from a customer looking around the store to picking up an item, then later purchases with an associate.



3) **Possible Alternative Database Choices:** Although we chose SQL as our primary database management strategy, several other possible alternatives were considered before we decided to move forward with SQL. Our first alternative was to implement NoSQL databases such as MongoDB or CouchDB. NoSQL databases provide the benefit of having less pre-organization and pre-planning because of their unstructured style. An unstructured style offers the benefit of having various data types and structures organized non-relationally. Additionally, NoSQL can scale horizontally by adding servers as load increases. NoSQL was a choice for us that we had to consider before deciding. The reasoning behind our choice of SQL over NoSQL is provided in the following section. In addition, we also looked at forms of database management. While SQL databases represent an example of Relational Database Management Systems (RDMS), we also

looked at other forms, such as Object-Oriented Database Management Systems (OODMS), which use the principles of OOP Programming in database management systems. In comparing these various forms, we stuck with RDMS and SQL because they best fit our software system's needs.

4) **Tradeoffs Between Our Choice & Alternative Databases:** For our data management strategy, our choice for SQL data management comes with advantages and tradeoffs compared to alternatives. We felt SQL, known for its well-structured and relational nature, worked well in handling complex functions and transactions while maintaining the integrity of our system. In managing inventory, sales transactions, and customer information, this data management strategy was well-suited for managing our data. However, there are some tradeoffs we considered. Obviously, for SQL data management, we know that it needs help to handle unstructured or semi-structured data. This could be a challenge when implementing unique data types, whereas other data management strategies could take diverse data types. NoSQL data strategies may offer more flexibility and scalability; however, considering that the data for our system is more structured and more defined, we still felt that SQL was a better choice. Also, considering the relational nature of SQL and continuously moving and altering data with relatively few data types, SQL is a far better choice for our data management strategy.