Lesson 01 Demo 01

Generating Architecture Diagram Using Generative AI

Objective: To generate the architecture diagram for design components of an e-commerce app using Generative AI for enhanced scalability and performance

Tools required: ChatGPT account

Prerequisites: NA

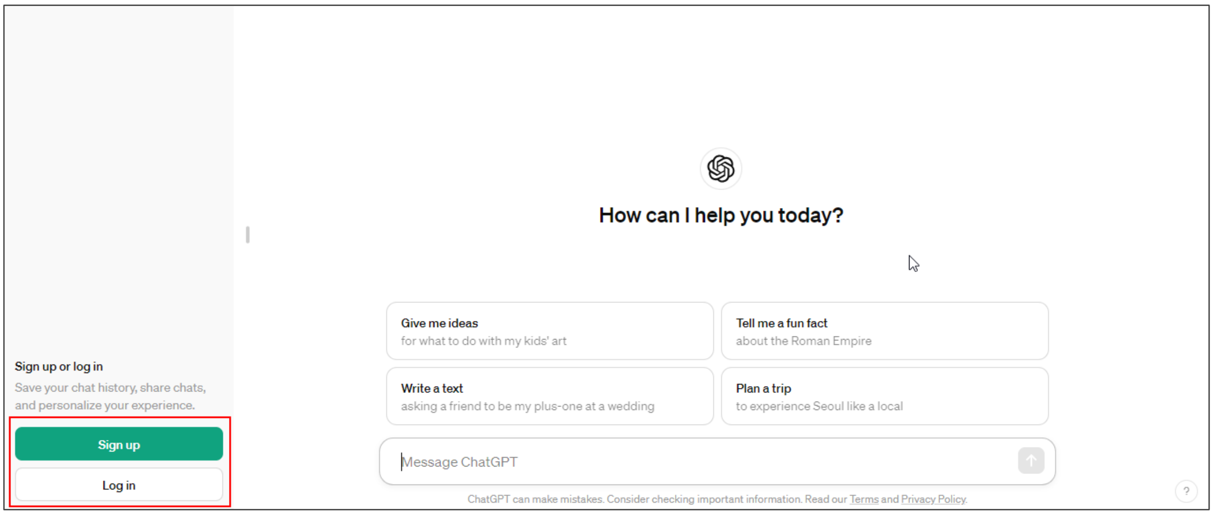
Steps to be followed:

1. Generate architecture for the login page, add to the cart page, payment page, and purchase page using prompts

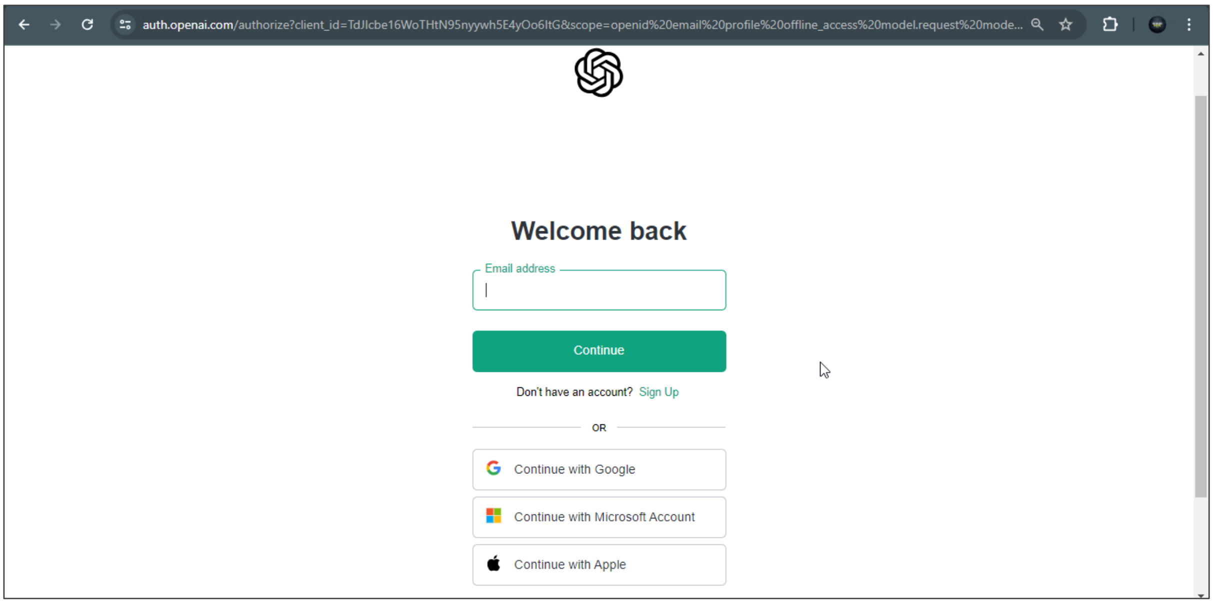
Note: Please be advised that ChatGPT, an artificial intelligence tool, can produce varied outputs even when presented with similar prompts.

Step 1: Generate architecture for the login page, add to the cart page, payment page, and purchase page using prompts

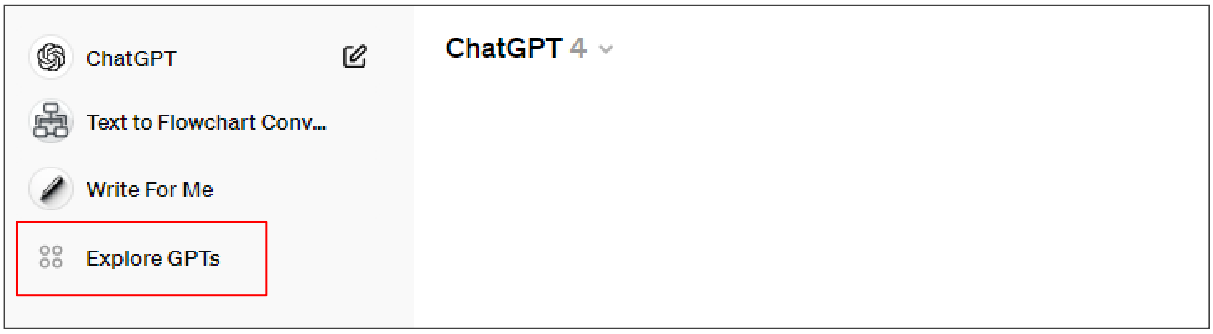
1.1 Go to **https://chat.openai.com** and log in to your account


How can I help you today?
Give me ideas
for what to do with my kids' art
Tell me a fun fact
about the Roman Empire
Write a text
asking a friend to be my plus-one at a wedding
Plan a trip
to experience Seoul like a local
Sign up or log in
Save your chat history, share chats, and personalize your experience.
Sign up
Log in
Message ChatGPT
ChatGPT can make mistakes. Consider checking important information. Read our Terms and Privacy Policy.

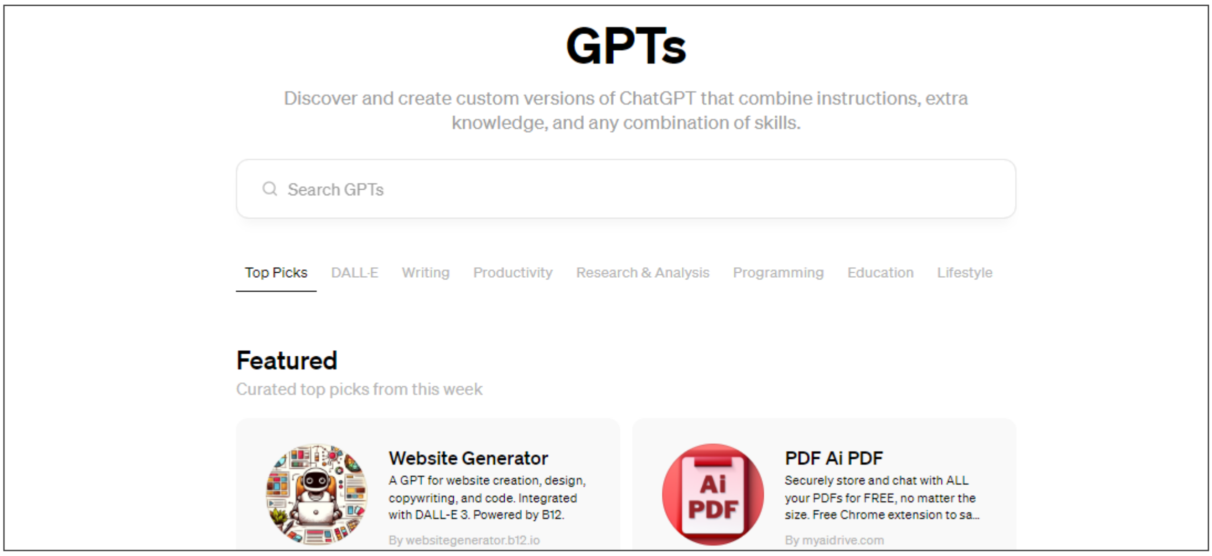
Note: Sign up if you do not have an account


auth.openai.com/authorize?client_id=TdJIcbe16WoTHtN95nyywh5E4yOo6ltG&scope=openid%20email%20profile%20offline_access%20model.request%20mode...
Welcome back
Email address
Continue
Don't have an account? Sign Up
OR
Continue with Google
Continue with Microsoft Account
Continue with Apple

1.2 Open the ChatGPT interface to access its features and options, and select **Explore GPTs** to learn more about different generative pre-trained transformers for designing microservice architecture



You will see the following features and options:

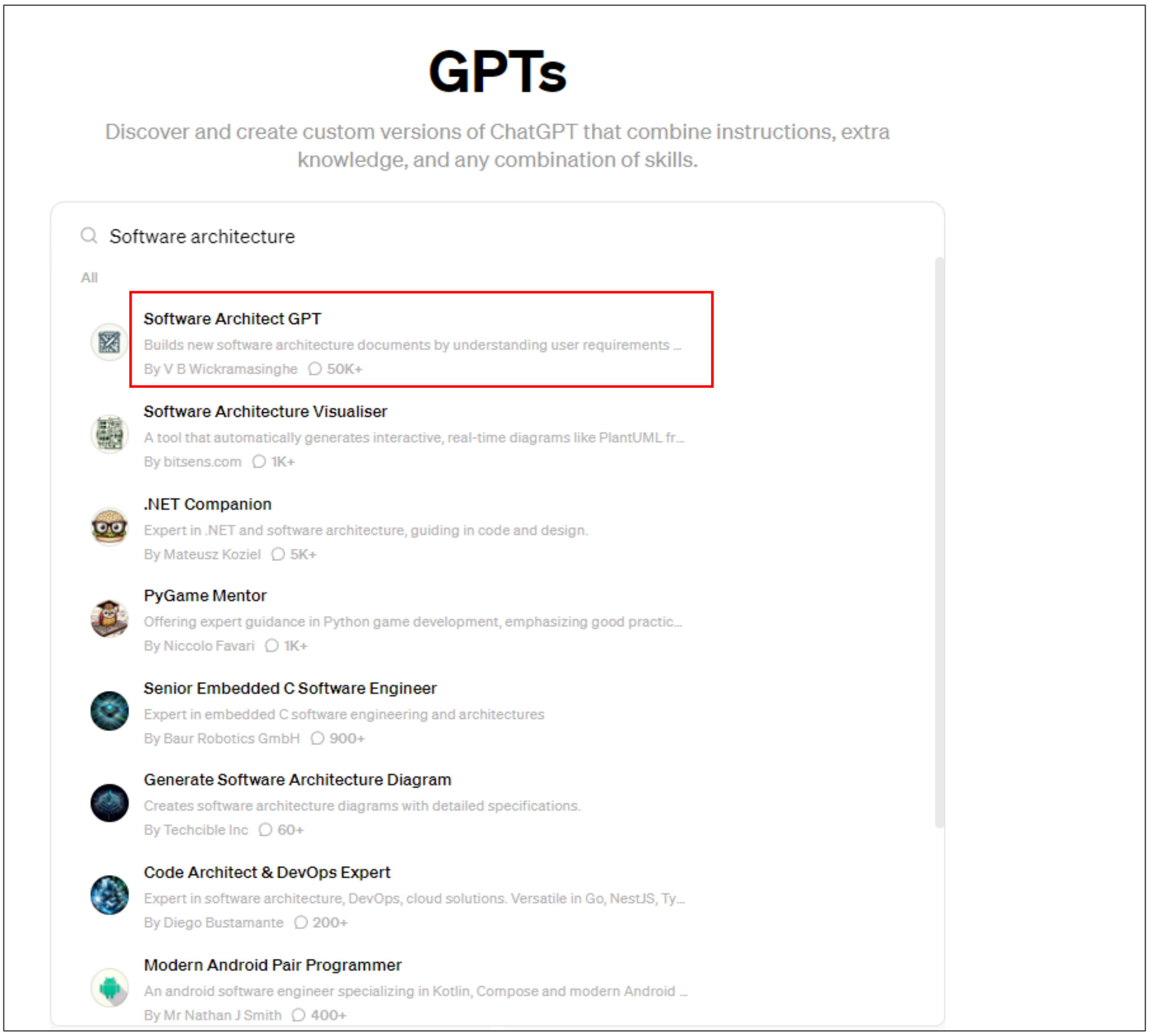


Discover and create custom versions of ChatGPT that combine instructions, extra knowledge, and any combination of skills.

A GPT for website creation, design, copywriting, and code. Integrated with DALL-E 3. Powered by B12.

Securely store and chat with ALL your PDFs for FREE, no matter the size. Free Chrome extension to sa...

1.3 In the search box, type **Software architecture**. From the suggested list, select **Software Architecture GPT**.



GPTs

Discover and create custom versions of ChatGPT that combine instructions, extra knowledge, and any combination of skills.

Software architecture

All

Software Architect GPT
Builds new software architecture documents by understanding user requirements ...
By V B Wickramasinghe 50K+

Software Architecture Visualiser
A tool that automatically generates interactive, real-time diagrams like PlantUML fr...
By bitsens.com 1K+

.NET Companion
Expert in .NET and software architecture, guiding in code and design.
By Mateusz Koziel 5K+

PyGame Mentor
Offering expert guidance in Python game development, emphasizing good practic...
By Niccolo Favari 1K+

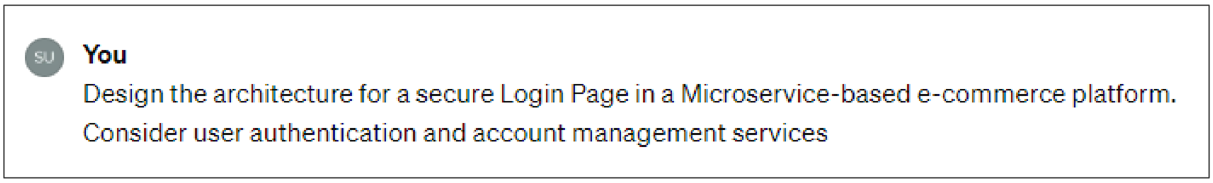
Senior Embedded C Software Engineer
Expert in embedded C software engineering and architectures
By Baur Robotics GmbH 900+

Generate Software Architecture Diagram
Creates software architecture diagrams with detailed specifications.
By Techcible Inc 60+

Code Architect & DevOps Expert
Expert in software architecture, DevOps, cloud solutions. Versatile in Go, NestJS, Ty...
By Diego Bustamante 200+

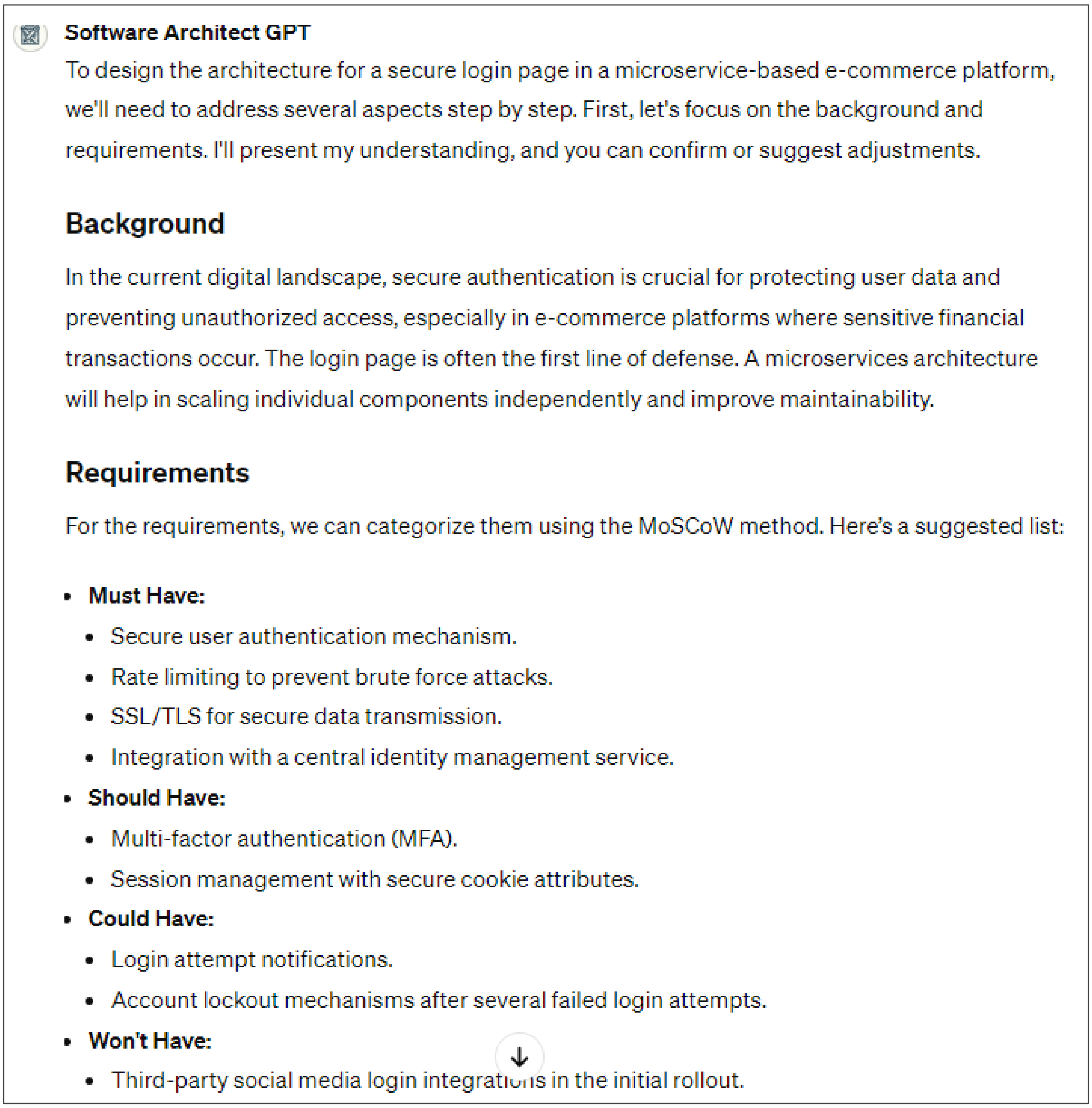
Modern Android Pair Programmer
An android software engineer specializing in Kotlin, Compose and modern Android ...
By Mr Nathan J Smith 400+

1.4 Provide the following prompt to generate descriptions of the secure login page for the e-commerce platform:
Design the architecture for a secure Login Page in a Microservice-based e-commerce platform. Consider user authentication and account management services.



You
Design the architecture for a secure Login Page in a Microservice-based e-commerce platform. Consider user authentication and account management services

The Software Architecture GPT tool generates the following information for creating a login page:



Software Architect GPT

To design the architecture for a secure login page in a microservice-based e-commerce platform, we'll need to address several aspects step by step. First, let's focus on the background and requirements. I'll present my understanding, and you can confirm or suggest adjustments.

Background

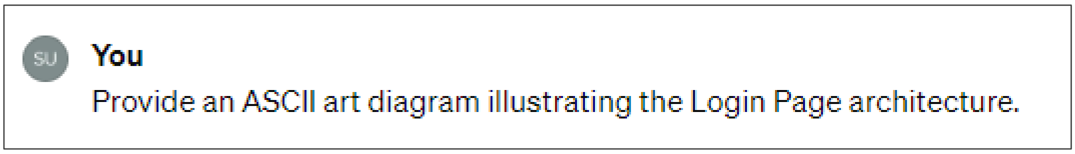
In the current digital landscape, secure authentication is crucial for protecting user data and preventing unauthorized access, especially in e-commerce platforms where sensitive financial transactions occur. The login page is often the first line of defense. A microservices architecture will help in scaling individual components independently and improve maintainability.

Requirements

For the requirements, we can categorize them using the MoSCoW method. Here's a suggested list:

- **Must Have:**
  - Secure user authentication mechanism.
  - Rate limiting to prevent brute force attacks.
  - SSL/TLS for secure data transmission.
  - Integration with a central identity management service.
- **Should Have:**
  - Multi-factor authentication (MFA).
  - Session management with secure cookie attributes.
- **Could Have:**
  - Login attempt notifications.
  - Account lockout mechanisms after several failed login attempts.
- **Won't Have:**
  - Third-party social media login integrations in the initial rollout.

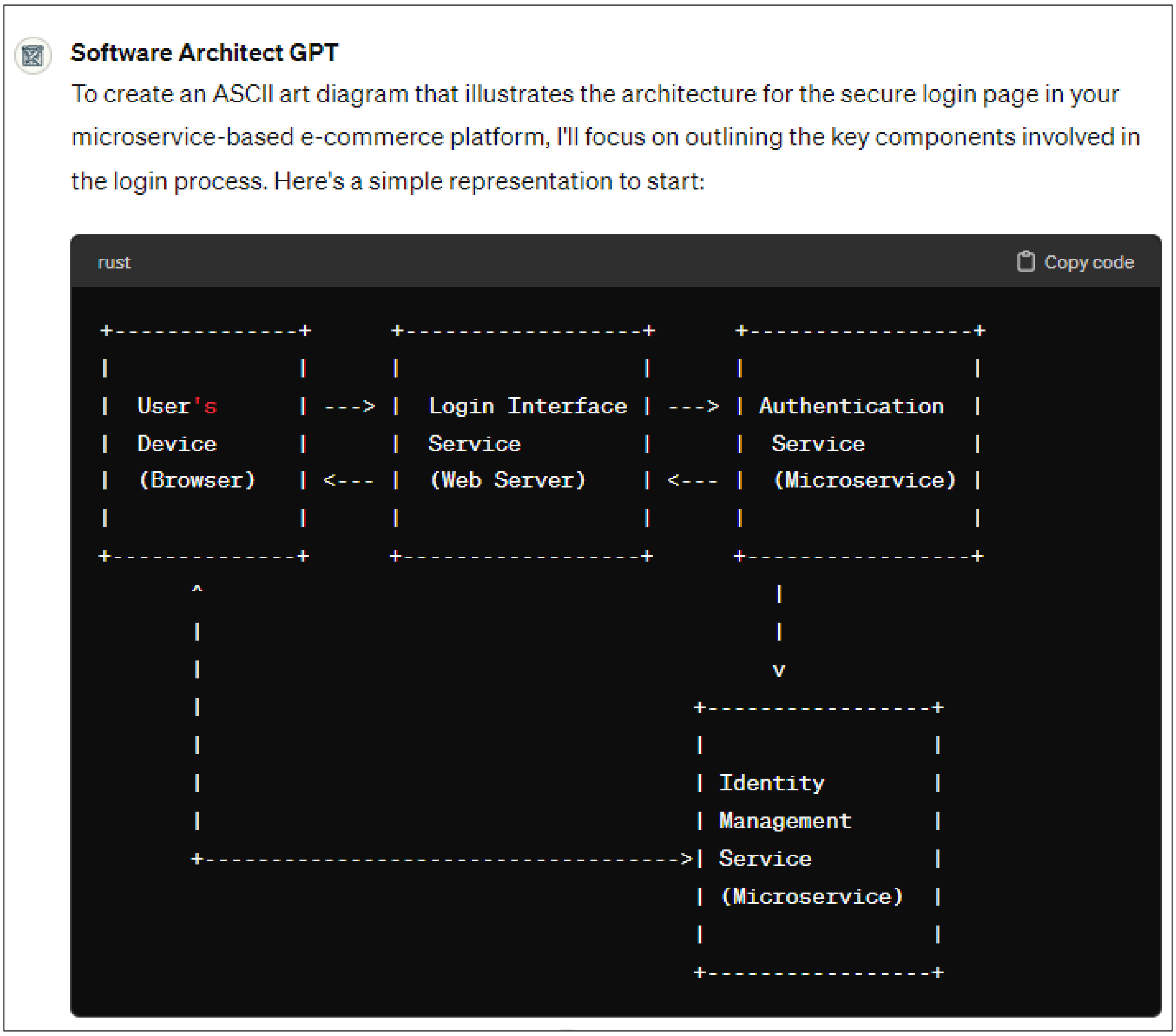
1.5 Generate an ASCII art diagram for the login page using the following prompt: **Provide an ASCII art diagram illustrating the Login Page architecture.**



You

Provide an ASCII art diagram illustrating the Login Page architecture.

The GPT generates the following ASCII art diagram:

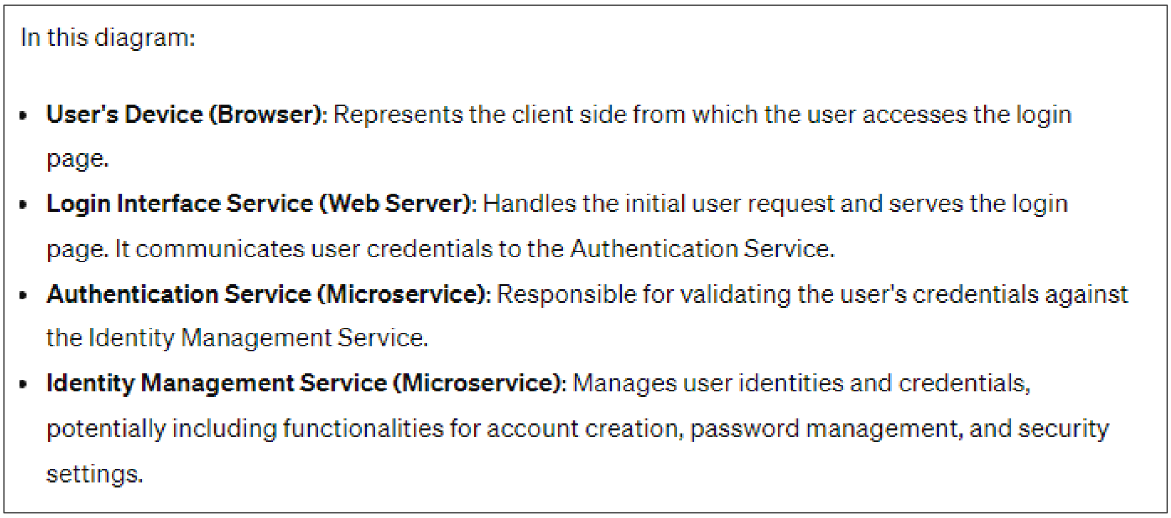


Software Architect GPT

To create an ASCII art diagram that illustrates the architecture for the secure login page in your microservice-based e-commerce platform, I'll focus on outlining the key components involved in the login process. Here's a simple representation to start:

```rust
+--------------+      +------------------+      +-----------------+
|              |      |                  |      |                 |
|  User's      | ---> |  Login Interface | ---> | Authentication  |
|  Device      |      |  Service         |      |  Service        |
|  (Browser)   | <--- |  (Web Server)    | <--- |  (Microservice) |
|              |      |                  |      |                 |
+--------------+      +------------------+      +-----------------+
       ^                                           |
       |                                           |
       |                                           v
       |                                     +-----------------+
       |                                     |                 |
       |                                     | Identity        |
       |                                     | Management      |
       +------------------------------------>| Service         |
                                             | (Microservice)  |
                                             |                 |
                                             +-----------------+
```

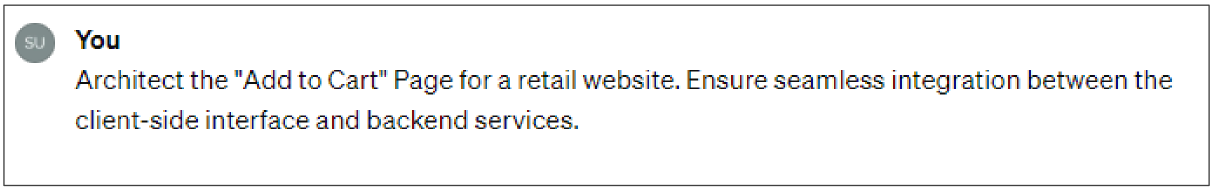
The description of the components of the diagram is as follows:



In this diagram:

- **User's Device (Browser):** Represents the client side from which the user accesses the login page.
- **Login Interface Service (Web Server):** Handles the initial user request and serves the login page. It communicates user credentials to the Authentication Service.
- **Authentication Service (Microservice):** Responsible for validating the user's credentials against the Identity Management Service.
- **Identity Management Service (Microservice):** Manages user identities and credentials, potentially including functionalities for account creation, password management, and security settings.

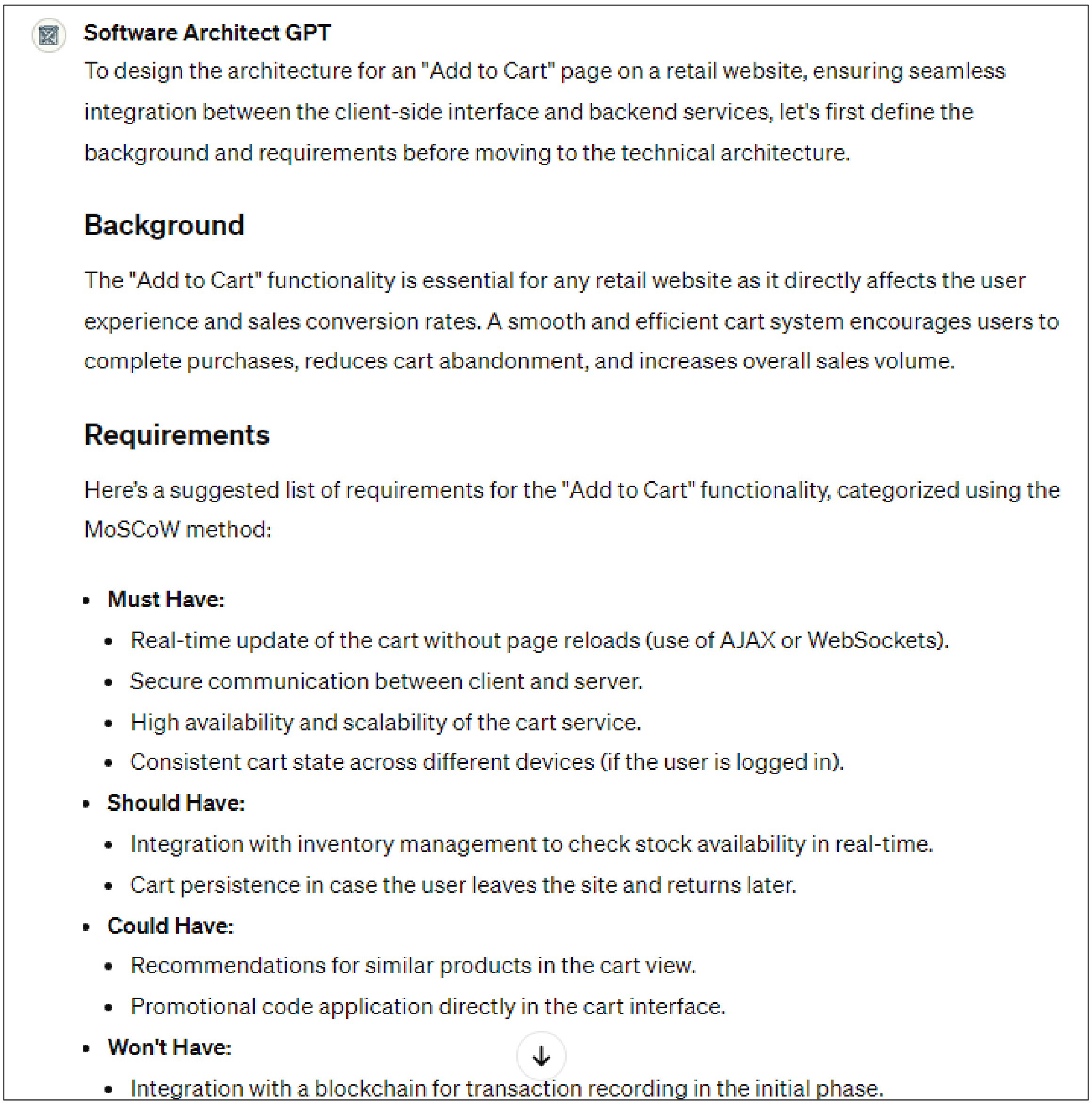
1.6 Use the following prompt to generate the requirements for an **Add to Cart** page: **Architect the "Add to Cart" Page for a retail website. Ensure seamless integration between the client-side interface and backend services.**



You

Architect the "Add to Cart" Page for a retail website. Ensure seamless integration between the client-side interface and backend services.

It generates detailed information for building the page.



Software Architect GPT

To design the architecture for an "Add to Cart" page on a retail website, ensuring seamless integration between the client-side interface and backend services, let's first define the background and requirements before moving to the technical architecture.

Background

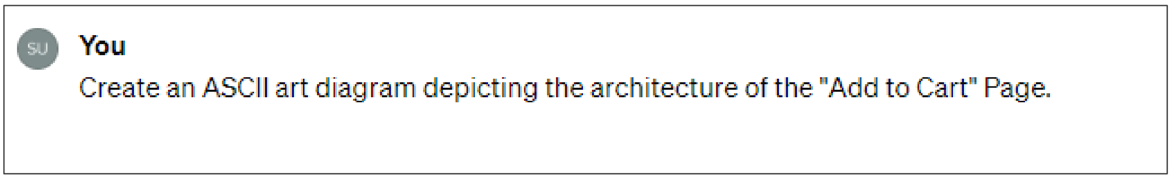
The "Add to Cart" functionality is essential for any retail website as it directly affects the user experience and sales conversion rates. A smooth and efficient cart system encourages users to complete purchases, reduces cart abandonment, and increases overall sales volume.

Requirements

Here's a suggested list of requirements for the "Add to Cart" functionality, categorized using the MoSCoW method:

- **Must Have:**
  - Real-time update of the cart without page reloads (use of AJAX or WebSockets).
  - Secure communication between client and server.
  - High availability and scalability of the cart service.
  - Consistent cart state across different devices (if the user is logged in).
- **Should Have:**
  - Integration with inventory management to check stock availability in real-time.
  - Cart persistence in case the user leaves the site and returns later.
- **Could Have:**
  - Recommendations for similar products in the cart view.
  - Promotional code application directly in the cart interface.
- **Won't Have:**
  - Integration with a blockchain for transaction recording in the initial phase.

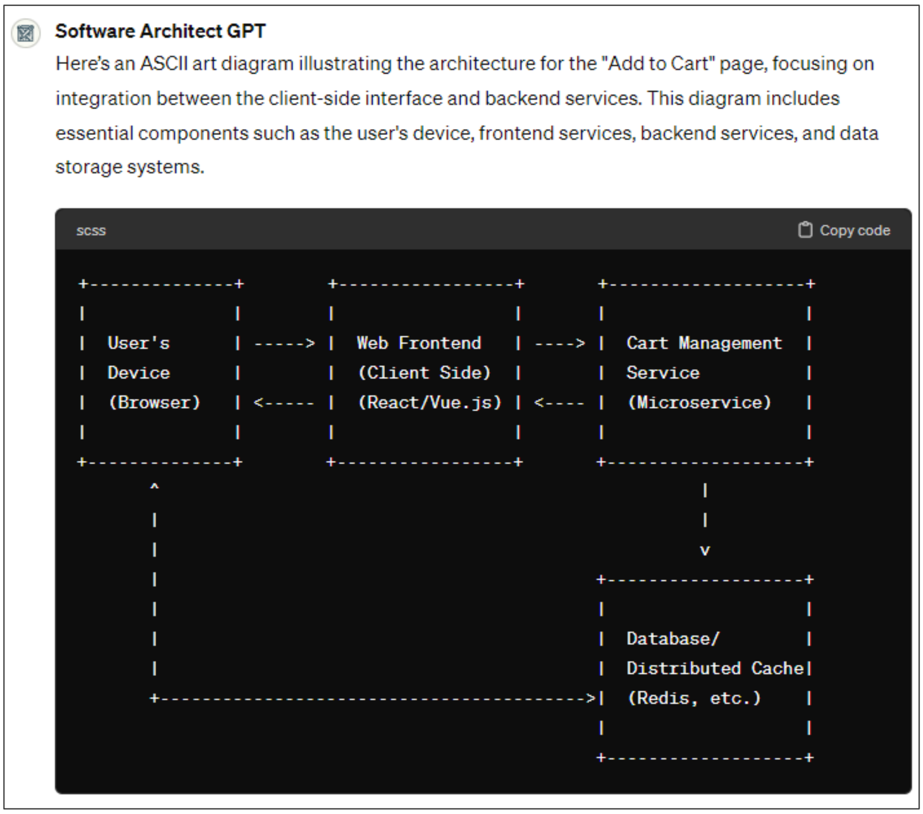
1.7 Generate an ASCII art diagram for the Add to Cart page using the following prompt:
Create an ASCII art diagram depicting the architecture of the "Add to Cart" Page.



You

Create an ASCII art diagram depicting the architecture of the "Add to Cart" Page.

The ASCII art diagram for the Add to Cart is as follows:

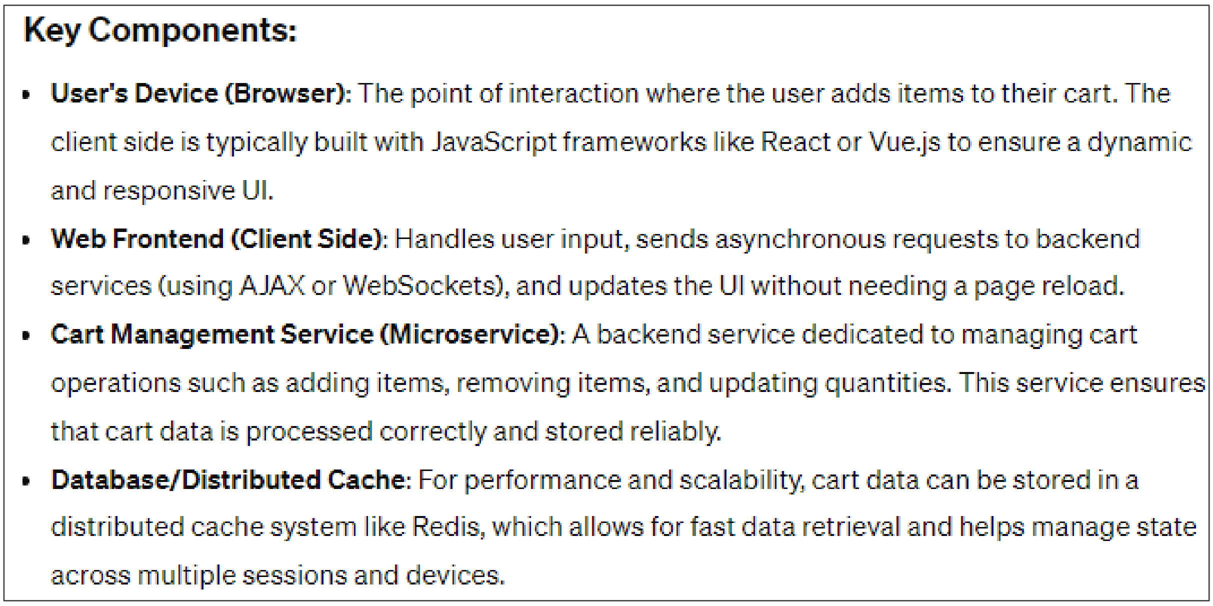


Software Architect GPT

Here's an ASCII art diagram illustrating the architecture for the "Add to Cart" page, focusing on integration between the client-side interface and backend services. This diagram includes essential components such as the user's device, frontend services, backend services, and data storage systems.

```
+-------------+       +----------------+      +------------------+
|             |       |                |      |                  |
|  User's     | -----> |  Web Frontend  | ----> |  Cart Management |
|  Device     |       |  (Client Side) |      |  Service         |
|  (Browser)  | <----- |  (React/Vue.js)| <---- |  (Microservice)  |
|             |       |                |      |                  |
+-------------+       +----------------+      +------------------+
      ^                                                  |
      |                                                  |
      |                                                  v
      |                                       +------------------+
      |                                       |                  |
      |                                       |  Database/       |
      |                                       |  Distributed Cache|
      +-------------------------------------->|  (Redis, etc.)   |
                                              |                  |
                                              +------------------+
```

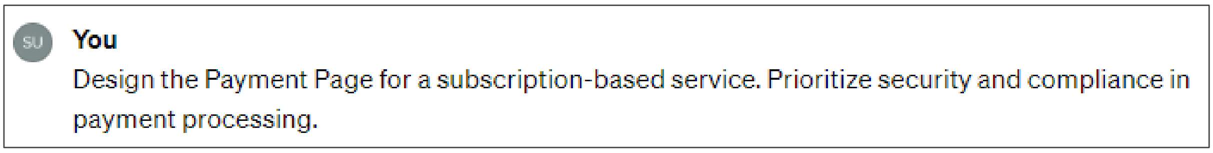
The component description for the ASCII art diagram is as follows:



Key Components:

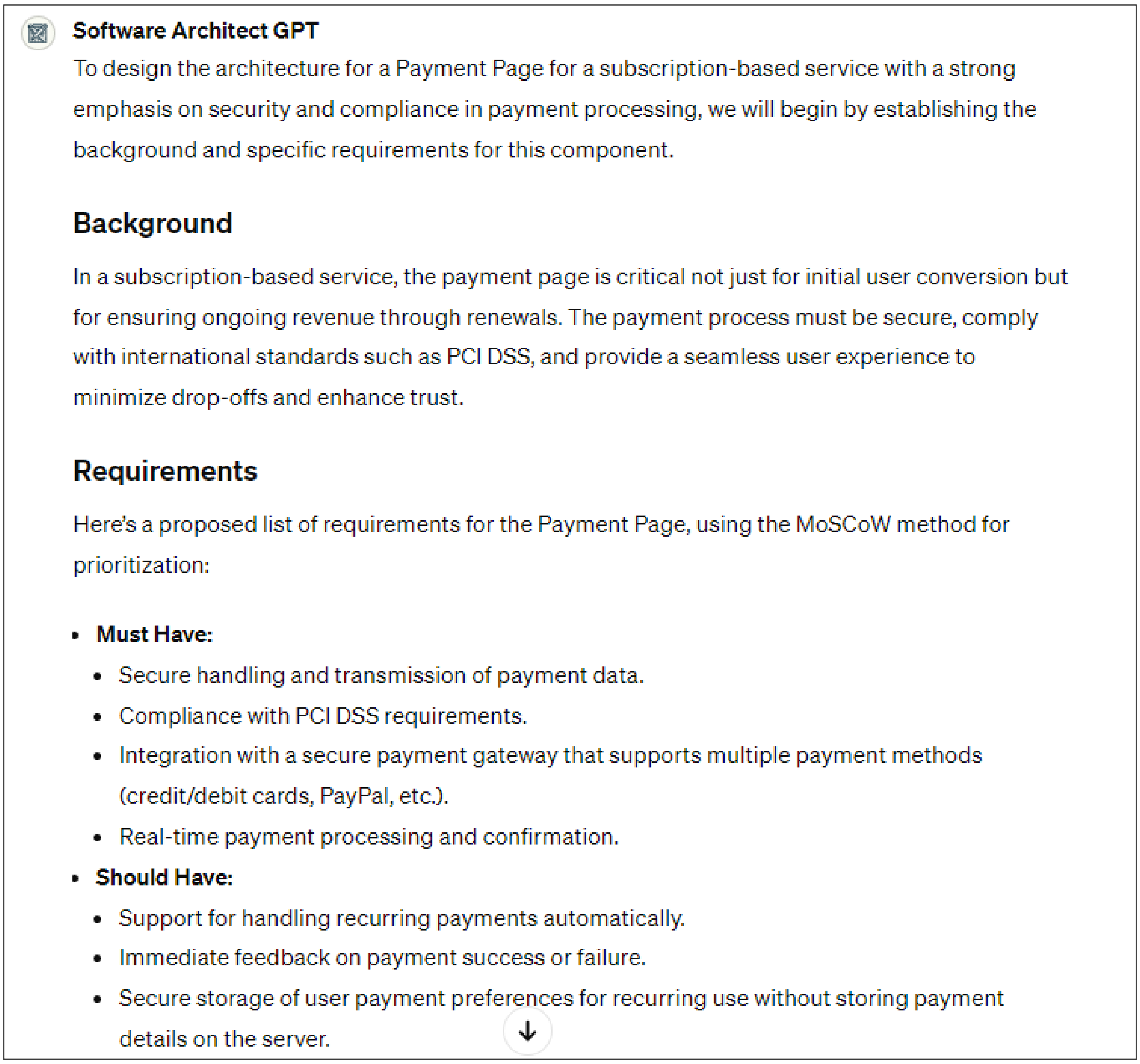
- **User's Device (Browser)**: The point of interaction where the user adds items to their cart. The client side is typically built with JavaScript frameworks like React or Vue.js to ensure a dynamic and responsive UI.
- **Web Frontend (Client Side)**: Handles user input, sends asynchronous requests to backend services (using AJAX or WebSockets), and updates the UI without needing a page reload.
- **Cart Management Service (Microservice)**: A backend service dedicated to managing cart operations such as adding items, removing items, and updating quantities. This service ensures that cart data is processed correctly and stored reliably.
- **Database/Distributed Cache**: For performance and scalability, cart data can be stored in a distributed cache system like Redis, which allows for fast data retrieval and helps manage state across multiple sessions and devices.

1.8 Use the following prompt to generate descriptions and requirements for designing the payment page:
Design the Payment Page for a subscription-based service. Prioritize security and compliance in payment processing.



You
Design the Payment Page for a subscription-based service. Prioritize security and compliance in payment processing.

The detailed description for building the payment page is as follows:



Software Architect GPT

To design the architecture for a Payment Page for a subscription-based service with a strong emphasis on security and compliance in payment processing, we will begin by establishing the background and specific requirements for this component.

Background

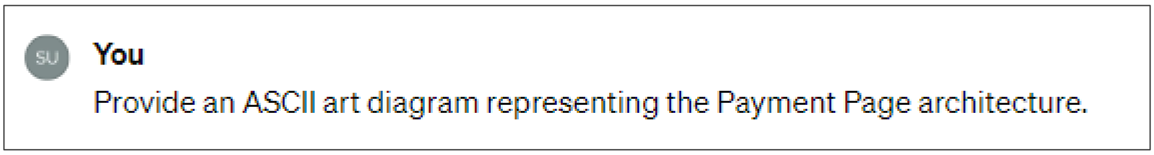
In a subscription-based service, the payment page is critical not just for initial user conversion but for ensuring ongoing revenue through renewals. The payment process must be secure, comply with international standards such as PCI DSS, and provide a seamless user experience to minimize drop-offs and enhance trust.

Requirements

Here's a proposed list of requirements for the Payment Page, using the MoSCoW method for prioritization:

- **Must Have:**
  - Secure handling and transmission of payment data.
  - Compliance with PCI DSS requirements.
  - Integration with a secure payment gateway that supports multiple payment methods (credit/debit cards, PayPal, etc.).
  - Real-time payment processing and confirmation.
- **Should Have:**
  - Support for handling recurring payments automatically.
  - Immediate feedback on payment success or failure.
  - Secure storage of user payment preferences for recurring use without storing payment details on the server.

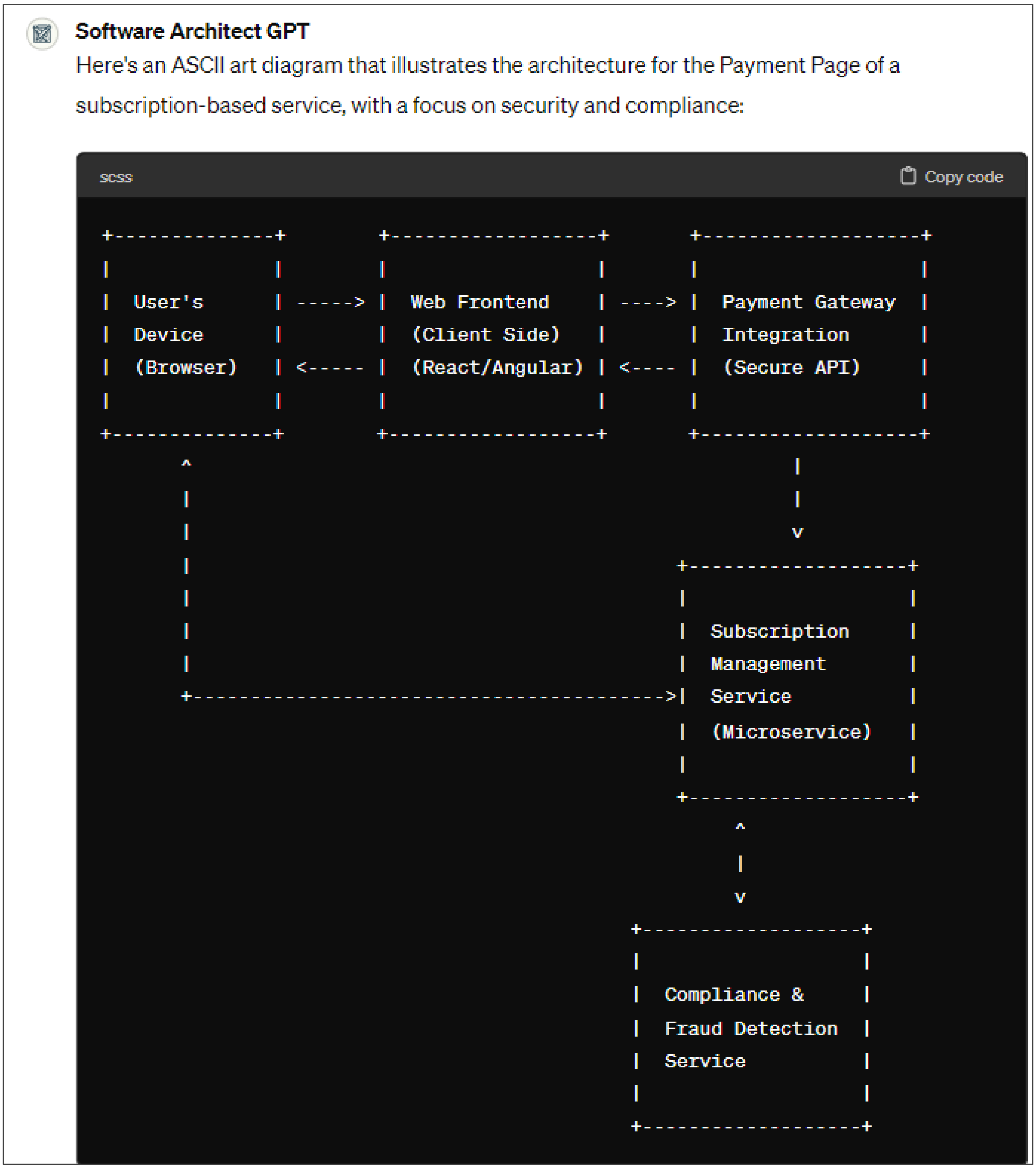
1.9 Use the following prompt to generate an ASCII art diagram for the payment page:
Provide an ASCII art diagram representing the Payment Page architecture.



You

Provide an ASCII art diagram representing the Payment Page architecture.

The following diagram shows the ASCII art architecture for the payment page:

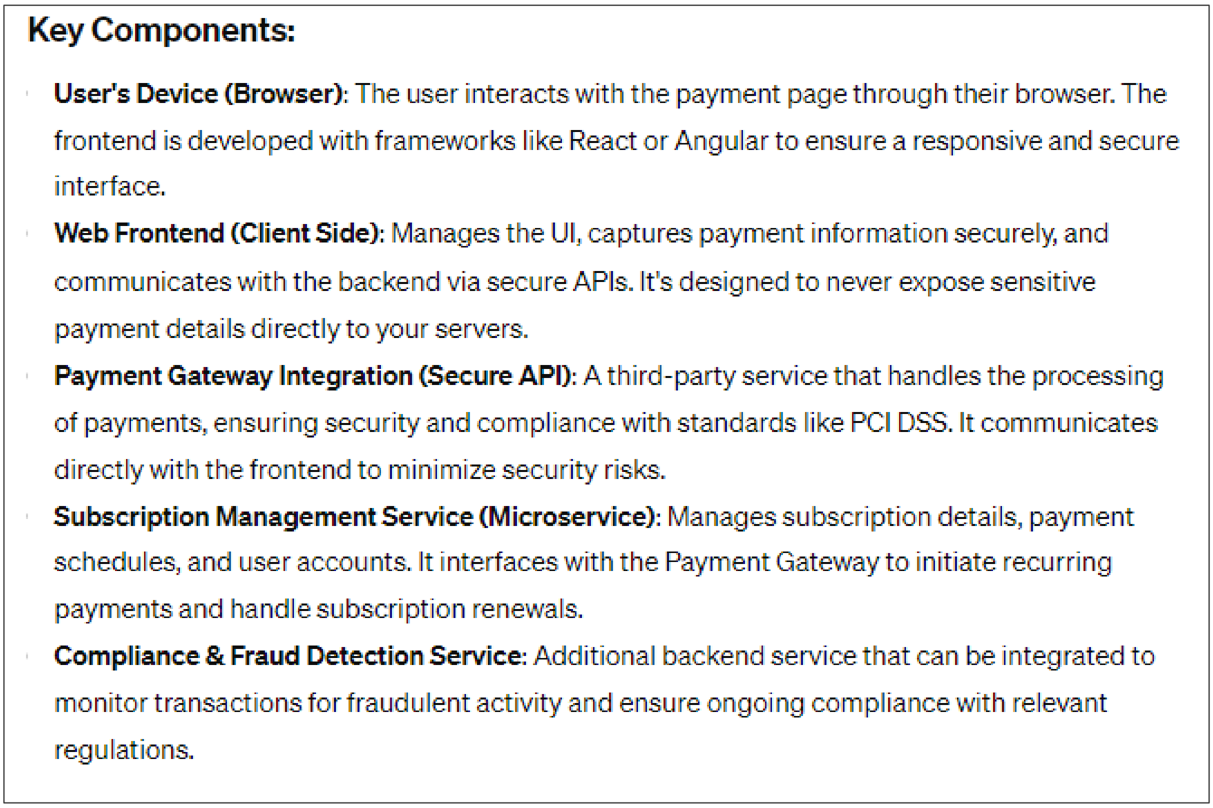


Software Architect GPT

Here's an ASCII art diagram that illustrates the architecture for the Payment Page of a subscription-based service, with a focus on security and compliance:

```
+--------------+        +------------------+       +-------------------+
|              |        |                  |       |                   |
|  User's      | -----> |  Web Frontend    | ----> |  Payment Gateway  |
|  Device      |        |  (Client Side)   |       |  Integration      |
|  (Browser)   | <----- |  (React/Angular) | <---- |  (Secure API)     |
|              |        |                  |       |                   |
+--------------+        +------------------+       +-------------------+
       ^                                                    |
       |                                                    |
       |                                                    v
       |                                          +-------------------+
       |                                          |                   |
       |                                          |  Subscription     |
       |                                          |  Management       |
       +----------------------------------------->|  Service          |
                                                  |  (Microservice)   |
                                                  |                   |
                                                  +-------------------+
                                                       ^
                                                       |
                                                       v
                                              +-------------------+
                                              |                   |
                                              |  Compliance &     |
                                              |  Fraud Detection  |
                                              |  Service          |
                                              |                   |
                                              +-------------------+
```

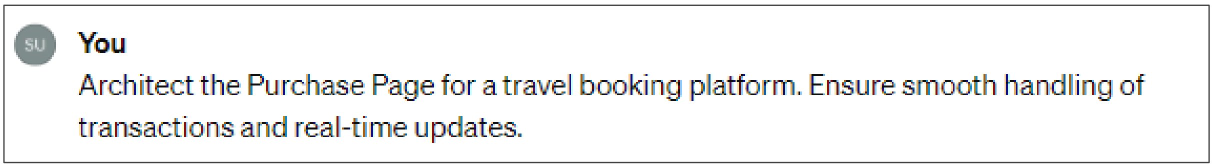
The key component description of the ASCII art diagram is as follows:



Key Components:

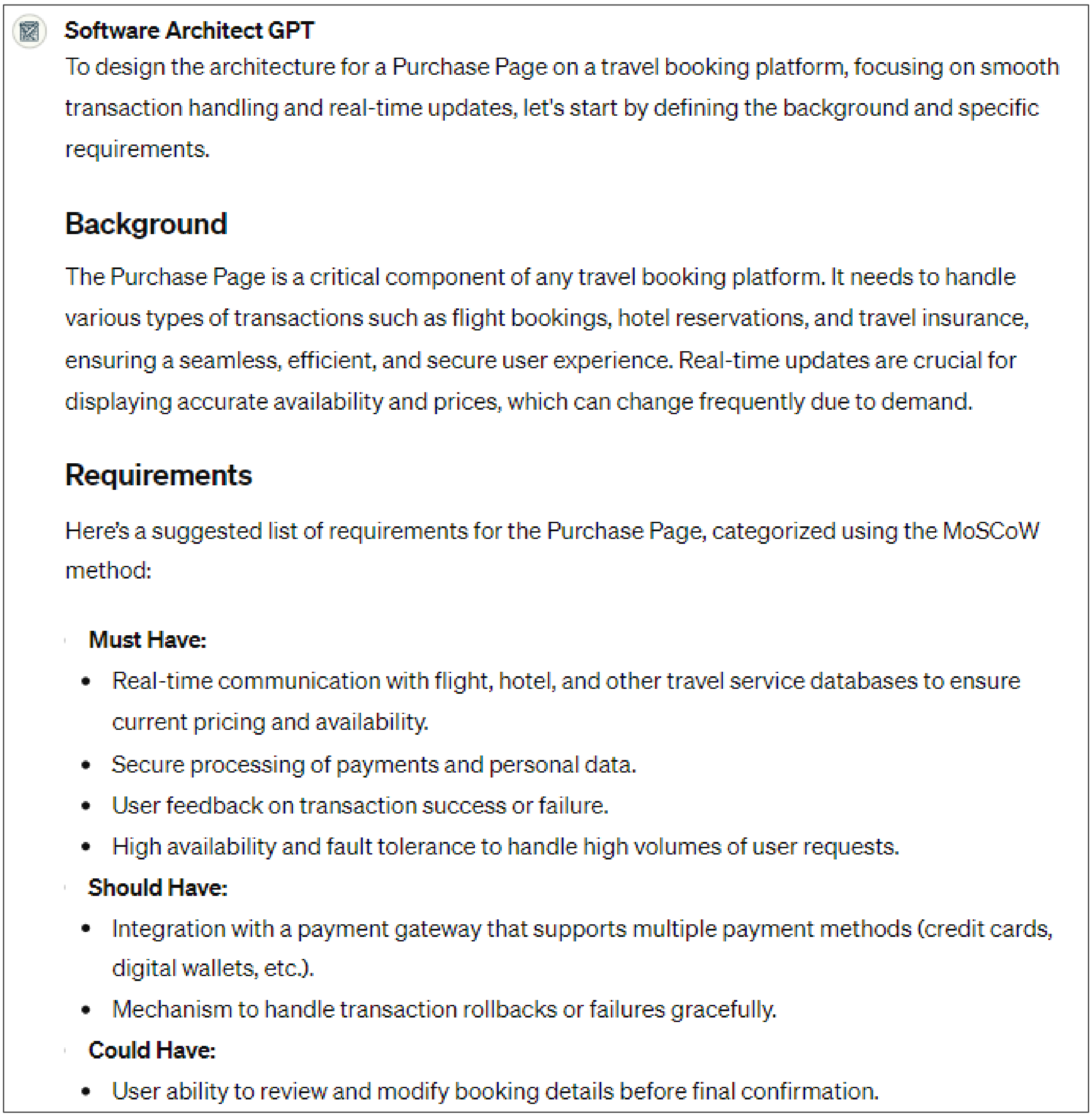
- **User's Device (Browser)**: The user interacts with the payment page through their browser. The frontend is developed with frameworks like React or Angular to ensure a responsive and secure interface.
- **Web Frontend (Client Side)**: Manages the UI, captures payment information securely, and communicates with the backend via secure APIs. It's designed to never expose sensitive payment details directly to your servers.
- **Payment Gateway Integration (Secure API)**: A third-party service that handles the processing of payments, ensuring security and compliance with standards like PCI DSS. It communicates directly with the frontend to minimize security risks.
- **Subscription Management Service (Microservice)**: Manages subscription details, payment schedules, and user accounts. It interfaces with the Payment Gateway to initiate recurring payments and handle subscription renewals.
- **Compliance & Fraud Detection Service**: Additional backend service that can be integrated to monitor transactions for fraudulent activity and ensure ongoing compliance with relevant regulations.

1.10 Generate a purchase page for the travel booking platform using the following prompt:
Architect the Purchase Page for a travel booking platform. Ensure smooth handling of transactions and real-time updates.



You
Architect the Purchase Page for a travel booking platform. Ensure smooth handling of transactions and real-time updates.

The Software Architecture GPT tool generates the following information for building the purchase page:



Software Architect GPT

To design the architecture for a Purchase Page on a travel booking platform, focusing on smooth transaction handling and real-time updates, let's start by defining the background and specific requirements.

Background

The Purchase Page is a critical component of any travel booking platform. It needs to handle various types of transactions such as flight bookings, hotel reservations, and travel insurance, ensuring a seamless, efficient, and secure user experience. Real-time updates are crucial for displaying accurate availability and prices, which can change frequently due to demand.

Requirements

Here's a suggested list of requirements for the Purchase Page, categorized using the MoSCoW method:

Must Have:

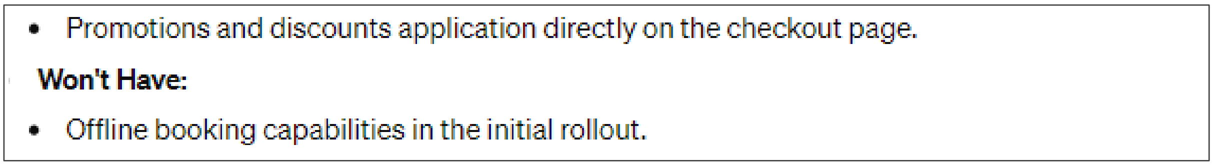
- Real-time communication with flight, hotel, and other travel service databases to ensure current pricing and availability.
- Secure processing of payments and personal data.
- User feedback on transaction success or failure.
- High availability and fault tolerance to handle high volumes of user requests.

Should Have:

- Integration with a payment gateway that supports multiple payment methods (credit cards, digital wallets, etc.).
- Mechanism to handle transaction rollbacks or failures gracefully.

Could Have:

- User ability to review and modify booking details before final confirmation.

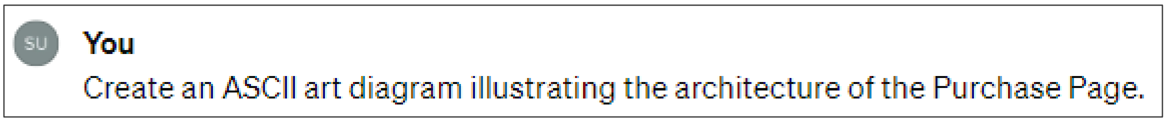


- Promotions and discounts application directly on the checkout page.

Won't Have:

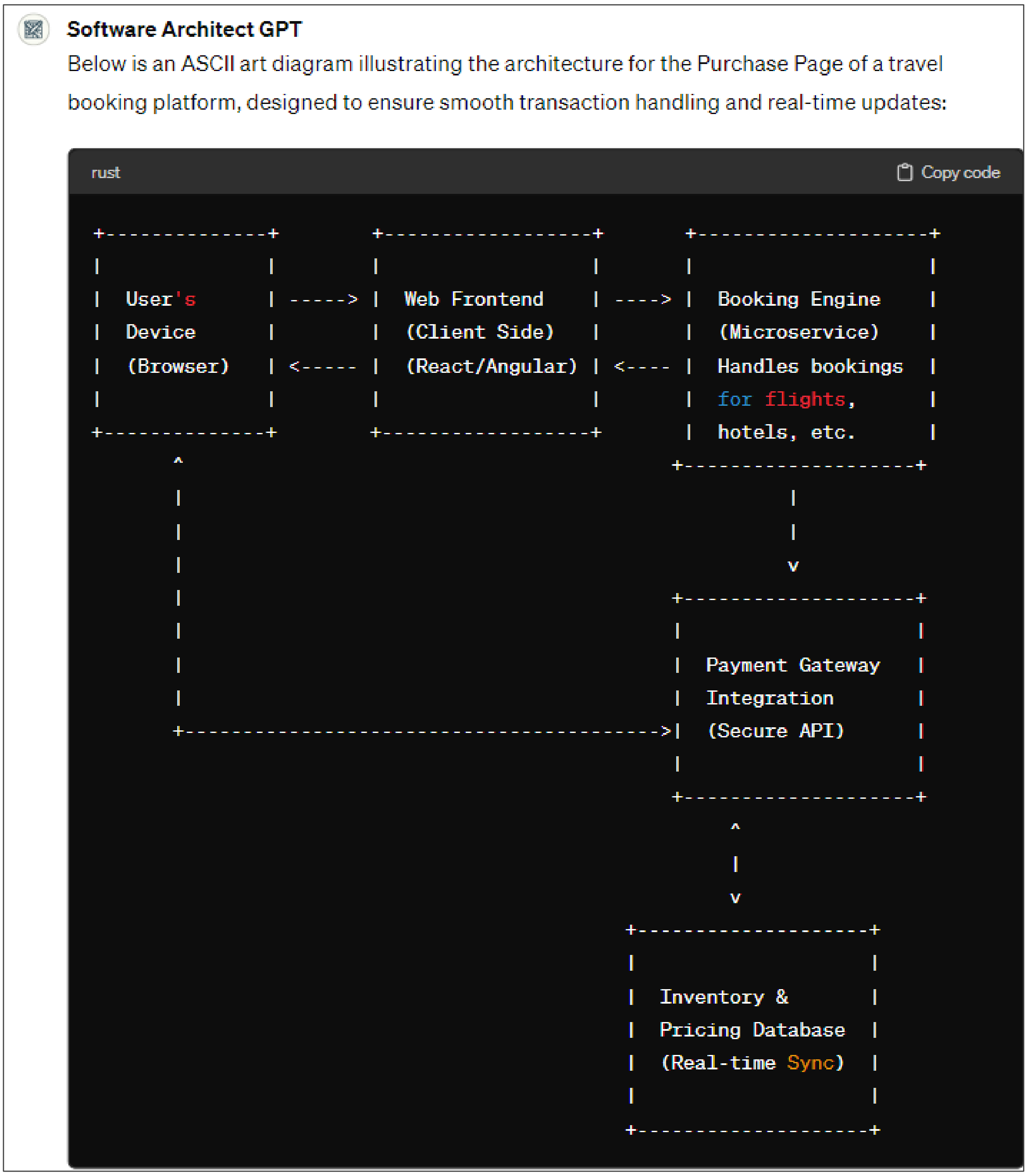
- Offline booking capabilities in the initial rollout.

1.11 Use the following prompt to generate an ASCII art diagram for the purchase page:
Create an ASCII art diagram illustrating the architecture of the Purchase Page.



You
Create an ASCII art diagram illustrating the architecture of the Purchase Page.

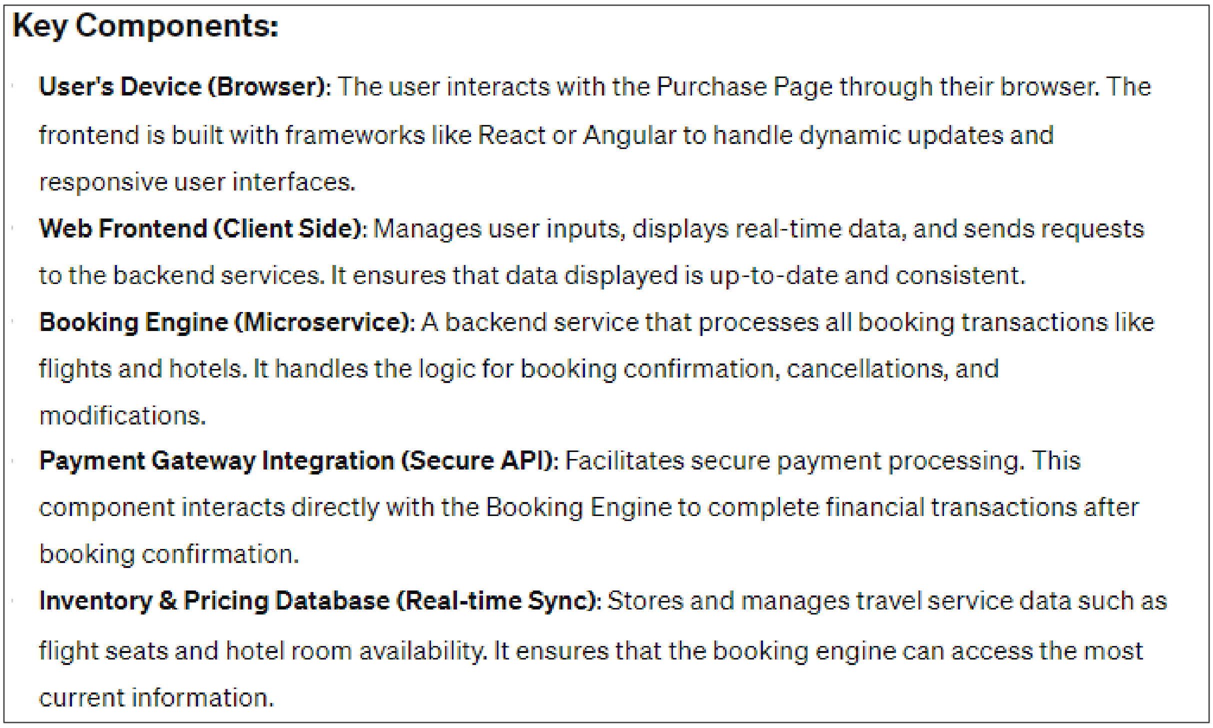
The ASCII art architecture for the purchase page is as follows:



Software Architect GPT
Below is an ASCII art diagram illustrating the architecture for the Purchase Page of a travel booking platform, designed to ensure smooth transaction handling and real-time updates:

```
+--------------+        +------------------+       +--------------------+
|              |        |                  |       |                    |
|  User's      | -----> |  Web Frontend    | ----> |  Booking Engine    |
|  Device      |        |  (Client Side)   |       |  (Microservice)    |
|  (Browser)   | <----- |  (React/Angular) | <---- |  Handles bookings  |
|              |        |                  |       |  for flights,      |
+--------------+        +------------------+       |  hotels, etc.      |
       ^                                          +--------------------+
       |                                                    |
       |                                                    |
       |                                                    v
       |                                          +--------------------+
       |                                          |                    |
       |                                          |  Payment Gateway   |
       |                                          |  Integration       |
       +----------------------------------------->|  (Secure API)      |
                                                  |                    |
                                                  +--------------------+
                                                       ^
                                                       |
                                                       v
                                              +--------------------+
                                              |                    |
                                              |  Inventory &       |
                                              |  Pricing Database  |
                                              |  (Real-time Sync)  |
                                              |                    |
                                              +--------------------+
```

The following screenshot shows the description of the ASCII art architectural components:



Key Components:

- **User's Device (Browser):** The user interacts with the Purchase Page through their browser. The frontend is built with frameworks like React or Angular to handle dynamic updates and responsive user interfaces.
- **Web Frontend (Client Side)**: Manages user inputs, displays real-time data, and sends requests to the backend services. It ensures that data displayed is up-to-date and consistent.
- **Booking Engine (Microservice):** A backend service that processes all booking transactions like flights and hotels. It handles the logic for booking confirmation, cancellations, and modifications.
- **Payment Gateway Integration (Secure API):** Facilitates secure payment processing. This component interacts directly with the Booking Engine to complete financial transactions after booking confirmation.
- **Inventory & Pricing Database (Real-time Sync):** Stores and manages travel service data such as flight seats and hotel room availability. It ensures that the booking engine can access the most current information.

By following these steps, you have successfully generated detailed architecture diagrams for key components of an e-commerce platform, enhancing functionality and user experience.