project 1

Feature 1: Customer Account Management

Pattern Chosen: **Request/Response**
Reasoning:

- **Business requirement analysis:**
  Customers expect secure, reliable, and immediate access to their accounts, work in poor internet connection.
- **Technical considerations:**

  Server load: Initial login requires verifying credentials and signing a JWT, without storing session data.

  latency: low latency for login/profile updates

  reliability needs: reliability even with intermittent network connections
- **User experience impact:**

  user signup ,login, update profile with immediate confirmation.
- **Scalability factors:**

  Each request is independent and stateless, so the system can handle many concurrent users.
- **Trade-offs:** each request must carry all necessary data

Feature 2: Order Tracking for Customers

Pattern Chosen: **SSE**

Reasoning:

- **Business requirement analysis:**
  track order status from the server need **server initiates push updates**
  status changes need **continuous stream of responses**
  check status frequently (every 30 seconds to 2 minutes) need **near real time**
  1000+ concurrent then need to use **async** to handle multiple concurrent jobs
  avoid High battery usage need to High efficiency pattern(**SSE**)
- **Technical considerations:**

  Server load: Each client connection consumes memory for status changed and long live connection, using asyng helps reduse overhead

  latency: near real-time update with minimal delay using event driven and avoid blocking the asynchronous loop.

  reliability needs: handle disconnects store status in memory
- **User experience impact:**

  user track his order and see the updates in near real time and responsive improving overall convenience with avoid High battery usage.
- **Scalability factors:**

Able to handle multiple concurrent orders without slowing down and without blocking the asynchronous loop.

- **Alternatives considered:**
    - **Short Polling:** Rejected because it generates frequent, chatty requests, wasting bandwidth and potentially creating a bottleneck over a 30-minute period, not real time, High battery usage.
    - **Long Polling:** Rejected because it doesn't support initiates push updates to the client, and continuous stream of responses.
    - **WebSockets:** Rejected because of high cost and resource overhead and unless bidirectional communication.
- **Trade-offs accepted:** more server resources per client, Server must handle reconnection logic

Feature 3: Driver Location Updates

Pattern Chosen: **Web-socket**
Reasoning:

- **Business requirement analysis:**

  continuous client update for **Smooth map movement,** continuous update for **location every 10–15 sec** : so it need bidirectional connection

  Only visible to the order's customer: **private stream.**

  Mobile networks with variable quality → Must handle dropped connections.
  Active for 30–45 min max → Temporary session-based connection.
- **Technical considerations:**
    - **Server load** can it handle many persistent connections
      latency : real-time responsiveness
      reliability needs: automatically tries to reconnect if the connection drops unexpectedly or server goes down unexpectedly.
- **User experience impact:** Customers can track the driver in real time, providing smooth, accurate updates, which builds trust and satisfaction with the service.
- **Scalability factors:**

  High numbers of concurrent connections or rapid message increase latency, and risk dropped connections
- **Alternatives considered:**
    - SSE : Rejected because its unidirectional
    - pop-sup: Rejected because of decoupled, many-to-many messaging, which adds unnecessary complexity for a short-lived, 1-to-1 connection between driver and costumer.
- **Trade-offs accepted:** increases server resource usage

Feature 4: Restaurant Order Notifications

Pattern Chosen: pub-sub
Reasoning:

- **Business requirement analysis:**
  - immediate notify need near realtime 5 sec
  - order appear automatically server initiate push
  - handle max 2 order per min
  - multiple restaurant need multiple channels to subscribe
  - no missed order high reliability need to radis
- **Technical considerations:**

  server load: Redis + SSE is lightweight, SSE long-lived but low-overhead, Redis broadcasting to multiple connected clients efficiently.

  latency: real time notifications there is no latency
  Reliability need : reliable it is store the orders that confirmed in db and use radis to update new one.
- **User experience impact:**

  help worker to get notifications smoothly without refresh dashboard it is reliable because store the orders that confirmed in db and use radis.
- **Scalability factors:**
  - SSE + Redis can handle thousands of connected dashboards with minimal latency by using async workers
- **Alternatives considered:**
  - short pulling: Rejected because chatty adds extra overhead
  - long pulling: Rejected because it delivers only one response per request, is hard to debug, adds server load.
  - web-socket : Rejected because no need bidirectional which is heavier and need complex.
- **Trade-offs accepted:** Each client keeps an open HTTP connection, which can limit scalability compared to stateless requests.

Feature 5 : Customer Support Chat

Pattern Chosen: **Websocket**
Reasoning:

- **Business requirement analysis:**
  - instant message for customer and agent with typing indicators need bidirectional connection and Real-time communication.
  - customer and agent need to store data on the server async with websocket.
- **Technical considerations:** [Server load, latency, reliability needs]

  Server load: consume memory and CPU per active connection .

  Latency: low-latency

  Reliability needs: error handling.

- **User experience impact:**

  real time, responsive and interactive, interaction between customer and agent and the data is save across multiple servers because of saving data in db.

- **Scalability factors:**

  Async saving reduces the blocking of WebSocket threads, but under heavy load, the DB still needs to handle high write throughput.

- **Alternatives considered:**
  - SSE : Rejected because its unidirectional
  - popsup: Rejected because of decoupled, which adds unnecessary complexity for a short-lived, 1-to-1 connection between age and costumer.
    Trade-offs accepted: increases server resource usage
- **Trade-offs accepted:** increases server resource usage

Feature 6: System-Wide Announcements

Pattern Chosen: SSE
Reasoning:

- **Business requirement analysis:**
  - announcement broadcast to all connected user.
  - receive announcement while using the app
    - online user Send via SSE event loop.
    - users offline when announcement is sent so we save announcements in memory.
  - without needing persistent all user connections .
- **Technical considerations:**
  - Server load: consume memory and CPU per active connection .
  - latency: low-latency
  - reliability needs: Ensure ordering and deduplication logic and check global announcement .
- **User experience impact:**
  - user can see near real-time updates on new features, discounts, and announcements, making the app feel alive and engaging.
- **Scalability factors:**
  - handle many concurrent connections without overwhelming the server using event-driven servers
- **Alternatives considered:**
  - **long pulling:** Rejected because it is requires the client to repeatedly request to get updates.
  - **pub-sub:** Rejected because it adding unnecessary complexity and extra infrastructure.
  - **web-socket:** Rejected because there is no need bidirectional connection.

- **Trade-offs accepted**: cost memory for global announcement.

Feature 7 : Image Upload for Menu Items

Pattern Chosen: **pup-sup**
Reasoning:

- **Business requirement analysis:**

  multiple services with large file (upload might fail due to network issues or file problems)so need to not cascade fail with one service failure.

- **Technical considerations:**
  - **Server load:** pup-sup reducing blocking operations.
  - **Latency**:instant update
  - **Reliability**: independent service reduce risk failure.
- **User experience impact:**

  real-time progress updates for large uploads, improving trust and engagement.

- **Scalability factors:**

  process services for each file asynchronously and decoupled many uploads do not block the server.

- **Alternatives considered:**
  - **long pulling:** Rejected because it is requires the client to repeatedly request to get updates.
  - **short pulling :** Rejected because cause bottle neck with large image
  - **SSE** : Rejected because SSE may cause cascade failure.
  - **Web-socket**: Rejected because may cause cascade failure.

Trade-offs accepted: complexity in implementing pub/sub and increase infrastructure dependency (radis)