FoodFast: Food Delivery Platform

Feature 1: Customer Account Management

- **Chosen pattern:** Request-Response model
- **Reasoning:**
    - Business requirements perspective: not costly
    - Technical considerations: Easy to implement
    - User experience impact: it provides immediate response and updates to the user
    - Scalability factors: Scales well under average conditions
- **Trade-offs accepted:**
    - Too many requests at peek hours may slow down the service, but this can be tolerable

Feature 2: Order Tracking for Customers

- **Chosen Pattern:** PubSub (RabbitMQ)
- **Reasoning:**
    - **Business requirements perspective:** Reduces load on client side, and updates feel like real time
    - **Technical considerations:** Decouples the server from the end user by using the message queue that takes the responsibility to deliver the message to the client asynchronously
    - **User experience impact:** Mobile battery is conserved because the user don't need to check for the status very frequently
    - **Scalability factors:** High scalability because it off-loads the server, and the message queue processes the messages asynchronously
- **Alternatives considered:**
    - **Short polling:** Rejected because we need to conserve the user's Mobile battery
    - **SSE:** Rejected because it adds load on the server to send too many notifications to all users
- **Trade-offs accepted:** Latency

Feature 4: Restaurant Order Notifications

- **Chosen Pattern:** Server-sent events (SSE)
- **Reasoning:**
    - **Business requirements analysis:** Orders must be displayed on the dashboard instantly without refreshing
    - **Technical considerations:** Multiple users can view the dashboard at the same time. It is also event-driven, so the restaurant will be notified when orders are placed. No need for the restaurant to check for new orders
    - **User experience impact:** The restaurant never misses an order, and customers are fulfilled directly
    - **Scalability factors:** It scales very well, because in peak hours, many orders (events) are received over the same connection
- **Alternatives considered:**
    - **Websocket:** Rejected because we need to have connections between the server and many connected staff members
    - **Polling:** Rejected because it isn't efficient because this may result in latency

Feature 5: Customer Support Chat

- **Chosen Patterns:** Websocket
- **Reasoning:**
    - **Business requirements analysis:** Chat should be 2-way communication between customer and support team. And messages should be received instantly
    - **Technical considerations:** Websockets implement a bi-directional communication between the client and the server. It also provides very low latency because it's built on top of TCP.
    - **User experience impact:** Users receive message immediately
    - **Scalability factors:** It scales well and can handle multiple connections at the same time
- **Alternatives considered:**
    - **Server-sent events (SSE):** Rejected because it is a one-way communication
- **Trade-offs accepted:** Complexity

Feature 6: System-Wide Announcements

- **Chosen Patterns:** Pubsub (Redis)
- **Reasoning:**
    - **Business requirements analysis:** Announcement notifications need to be sent to many users at the same time
    - **Technical considerations:** 1) Redis is a message broker that can forward messages to all subscribers that subscribed to certain topic(s). 2) Latency is accepted so no need to be real-time 3) The server initiates the communication
    - **User experience impact:** Users receive announcements about topics they are interested in.
    - **Scalability factors:** it scales very well because the server sends messages to Redis broker, then it takes the responsibility to forward messages to the users, so this doesn't over load the server with thousands of messages
- **Alternatives considered:**
    - **SSE:** Rejected because announcements don't need to be in real-time

Feature 7: Image-Upload for Menu Items

- **Chosen Patterns:** Short Polling
- **Reasoning:**
    - **Business requirements analysis:** Users should be able to check for the upload status and get progress percentage
    - **Technical considerations:** 1) The upload time can be estimated 2) if the upload fails the users will be able to know and check the status
    - **User experience impact:** Users are aware of the upload progress and are acknowledged when failures happen
    - **Scalability factors:** it scales well, because the upload process in this case takes 3 mins at maximum which may not result in a very huge number of status checks and requests. It scales also because it processes the images asynchronously so the user will not be blocked until the upload is done
- **Alternatives considered:**
    - **Long Polling:** Rejected because in case of failures, users will not be aware of it

    - **PubSub:** Rejected because it is more complicated
- **Trade-offs accepted:** Wasted requests and resources