Create list of stakeholders (e.g., Admin, Staff, Customer, etc.)

Primary Stakeholders (Direct Users)

- **Customers (End Consumers):** Individuals, Families, Business Professionals, Event Planners.
- **Restaurant Partners:** Restaurant Owners/Managers, Restaurant Staff (Cooks/Chefs), Restaurant "Expeditor" Staff.
- **Delivery Fleet (Couriers/Delivery Drivers):** Employed Drivers, Independent Contractors (Gig Workers), Bicycle/Motorcycle Couriers.
- **System Administrators & Internal Operations:** Customer Support Agents, Operations Managers, IT & System Admins.

Secondary Stakeholders (Indirect Interest)

- **Investors & Shareholders:** Venture Capitalists/Angels, Shareholders.
- **Suppliers & Business Partners:** Payment Processors, Mapping & GPS Services, Cloud Hosting Providers, Packaging Suppliers.
- **Regulatory & Government Bodies:** Health & Safety Inspectors, Tax Authorities, Department of Labor, Local City Councils.
- **Competitors:** Other Food Delivery Apps, Restaurants with In-House Delivery.
- **Community & Society:** Neighborhoods/Residents, Environmental Groups, Advocacy Groups for Gig Workers.
- **Internal Company Departments:** Marketing & Sales, HR & Recruiting, Legal & Compliance, Finance & Accounting.

Identify stakeholder biases:

1. Conflict: Delivery Speed vs. Driver Safety & Pay

- **Stakeholders Involved:** Customer, Delivery Driver, Investors
- **Clash of Interests:** The customer's demand for fast delivery incentivizes unsafe driving practices for drivers, who are paid per delivery and do not directly benefit from the increased risk. Investors pushing for growth based on customer satisfaction metrics exacerbate this pressure.

2. Conflict: High Commission Fees vs. Restaurant Profitability

- **Stakeholders Involved:** Restaurant Partners, The Delivery Platform/Investors, Customer
- **Clash of Interests:** The platform's primary revenue stream (high commissions from restaurants) directly cuts into the thin profit margins of restaurants. This can force

restaurants to raise prices for delivery customers, making the service less sustainable for all parties.

3. Conflict: The "Gig Worker" Model vs. Employee Benefits

- **Stakeholders Involved:** Delivery Driver, The Delivery Platform/Investors, Regulatory & Government Bodies
- **Clash of Interests:** The platform's low-cost, scalable business model depends on classifying drivers as independent contractors, denying them benefits and stable income. Drivers seek financial security, and regulators may challenge this classification as a violation of labor laws.

4. Conflict: Convenience and Packaging vs. Environmental Impact

- **Stakeholders Involved:** Customer, Restaurant Partners, Community/Environmental Groups
- **Clash of Interests:** The operational requirement for safe, hygienic food delivery for customers and restaurants generates significant single-use packaging waste, creating a negative environmental externality borne by the community.

5. Conflict: Platform's Standardized Experience vs. Restaurant's Brand Identity

- **Stakeholders Involved:** Restaurant Partners, The Delivery Platform, Customer
- **Clash of Interests:** The platform's need for a consistent, simple user experience leads to standardized menus and branding that can dilute the unique brand identity and customer relationship that the restaurant has worked to build.

Comment on prompt crafting:

In analyzing our experiments with different prompting techniques and Large Language Models (LLMs), we've drawn some clear conclusions about how to best leverage these tools for a task like creating a requirements analysis for a food delivery system. The difference between a single, detailed "zero-shot" prompt and a "careful, iterative" conversational approach is not just a matter of style; it fundamentally changes the nature and quality of the output. Furthermore, the choice of model, whether Gemini 2.5 Pro or DeepSeek, introduces another critical variable that determines the final document's strengths. Here is our commentary comparing these approaches.

Zero-Shot vs. Careful Prompting

Our investigation reveals a clear trade-off: zero-shot prompting excels at efficiency and structural adherence, making it ideal for standard tasks with a known format. In contrast, careful, iterative prompting is vastly superior for achieving depth, nuance, and strategic insight, especially when tackling complex or undefined problems.

1. Depth of Analysis vs. Speed of Generation

The most immediate difference we noticed was in the richness of the stakeholder analysis.

With the **zero-shot** approach, both Gemini and DeepSeek produced solid, if generic, lists of stakeholders (Customers, Drivers, Restaurants, etc.). The documents were generated quickly and were immediately usable. The DeepSeek model, in particular, was incredibly efficient at creating a perfectly formatted requirements document from a single command, including ten well-structured use cases.

However, the **careful prompting** method yielded a far more sophisticated analysis. By starting with a simple brainstorm and then refining it, we were able to guide the models to think more deeply. Our conversation with the **DeepSeek** model led to a more granular breakdown of stakeholders, identifying subgroups like "Families" vs. "Business Professionals" and secondary stakeholders like "Environmental Groups" and "Gig Worker Advocacy Groups." This led to a more thoughtful consideration of stakeholder conflicts, such as the tension between the gig economy model and worker benefits.

Our conversation with **Gemini 2.5 Pro** took this a step further. When we introduced the abstract concept of "health," the model pivoted entirely. It identified a novel set of stakeholders, including "Nutritionists," "Public Health Organizations," and "Health Insurance Companies." This demonstrates that a conversational approach can uncover hidden dimensions of a problem that a single, upfront prompt would likely miss.

2. Adherence to Structure vs. Strategic Exploration

The two models revealed distinct "personalities" in how they handled iterative prompting, which has significant implications for what kind of task they are best suited for.

The **DeepSeek** model proved to be an exceptional tool for **structured refinement**. Our conversation with it was a logical, step-by-step process of building a high-quality requirements document. We asked for stakeholders, then conflicts, then use cases, and it delivered each component with precision. The final document was a more detailed and thoughtful version of what the zero-shot prompt aimed for. This makes it an ideal assistant for a user who knows what they need and wants to build it diligently and correctly.

In contrast, the **Gemini 2.5 Pro** model excelled at **strategic exploration**. Our conversation with it diverged from the initial goal of writing use cases. Instead of a requirements document, the final output was a high-level strategic analysis. By integrating external research we provided, it discussed concepts like "Choice Architecture" and business model realignment around health incentives. This shows a capacity for creative synthesis, making it a powerful partner for brainstorming and exploring the "why" behind a project before defining the "what." It didn't just build what we asked for; it helped us question what we should be building in the first place.

3. The Nature of the Final Deliverable

Ultimately, the prompting strategy determined the type of document we received.

- **Zero-Shot (Both Models):** Produced a functional, ready-to-use requirements document with a standard set of use cases. Fast, efficient, and good for getting a project started quickly.
- **Careful Prompting (DeepSeek):** Produced a superior, more comprehensive requirements document. It was the same *type* of document as the zero-shot version, but with greater detail, better-defined stakeholders, and more relevant conflicts.
- **Careful Prompting (Gemini 2.5 Pro):** Produced a strategic analysis and problem-space exploration. It did not generate any use cases, but instead provided the foundational strategic thinking that would be necessary to write truly innovative and impactful use cases later on.

In conclusion, our experiments show that there is no single "best" way to prompt an LLM. The optimal strategy is contingent on the goal. If our task is to quickly produce a well-defined, standard document, a detailed zero-shot prompt with a structure-oriented model like DeepSeek is highly effective. However, if our goal is to explore a complex problem, uncover novel insights, and develop a deep strategic understanding, a careful, iterative conversation with a model like Gemini 2.5 Pro, which excels at abstract reasoning, is unquestionably the superior approach.

Here are the documents containing the prompts given to Gemini 2.5 Pro and responses generated:

Zero Shot Prompting and Careful Prompting

Here are the documents containing the prompts given to Deepseek and responses generated:

Zero Shot Prompting and Careful Prompting

We have also used GPT5 and Claude 4 for the analysis, the results given by both the models can be found here.

Write at least 10 use cases (≈5 pages total):

UC-1: Customer Places an Order

Preconditions: The customer is registered and logged in. Their delivery location is set and validated to be within a serviceable zone.

Main Flow:

1. The system displays a list of available restaurants based on the customer's validated location.
2. The customer selects a restaurant.
3. The system displays the restaurant's active menu.

4. The customer adds items to their cart, potentially customizing them (See Subflow S1).
5. The customer proceeds to checkout.
6. The system displays an order summary (items, subtotal, delivery fee, tax, total).
7. The system displays available payment methods. The customer selects or adds a payment method (See Subflow S2).
8. The customer confirms the order.
9. The system processes the payment successfully.
10. The system confirms the order is placed, provides an estimated delivery time, and updates the order status to "Received by Restaurant."

Subflows:

- **S1: Customize Item:** The customer selects an item. The system displays available customization options (e.g., remove onions, add extra cheese). The customer makes their selections and confirms, adding the customized item to the cart.
- **S2: Add Payment Method:** The customer selects "Add new card." The customer enters card details (number, expiry, CVC, billing postal code). The system validates the card details with a payment processor and tokenizes the card for storage. The system adds the new card to the customer's available payment methods.

Alternative Flows:

- **A1: Payment Failure:** In step 9, if the payment method is declined, the system notifies the customer and prompts them to use a different payment method, returning to step 7.
- **A2: Restaurant becomes unavailable:** After step 3, if the restaurant goes offline (e.g., too busy), the system notifies the customer and prevents them from ordering, returning them to the restaurant list.
- **A3: Restaurant Fails to Complete:** After the order is accepted, the restaurant cannot fulfill it. The restaurant contacts support. The system cancels the order, initiates a full refund, and notifies the customer.

UC-2: Restaurant Accepts and Manages an Order

Preconditions: The restaurant is registered, logged in, and has its status set to "Online." The restaurant's menu is published and active.

Main Flow:

1. The system sends a new order notification (via API or tablet app) to the restaurant.
2. The restaurant staff reviews the order details and customizations.
3. The staff accepts the order.
4. The system confirms acceptance to the customer and updates the order status to "Preparing."
5. The kitchen staff prepares the order.
6. Once the order is packaged and ready for pickup, the staff marks the order as "Ready" in the system. The system updates the order status to "Ready for Pickup."

Alternative Flows:

- **A1: Order Rejection:** In step 3, if the restaurant cannot fulfill the order, they reject it. The system notifies the customer, cancels the order, and initiates a refund.

UC-3: Driver Accepts a Delivery Offer

Preconditions: The driver is registered, logged in, has passed checks, and has set their availability status to "Available" (See UC-11).

Main Flow:

1. The system's matching algorithm sends a delivery offer to the most suitable available driver based on proximity, current load, and other factors. The offer contains pickup location, drop-off location, and estimated payout.
2. A driver accepts the offer.
3. The system assigns the order to the driver, notifies the restaurant and customer, and updates the order status to "Driver Assigned."
4. The system provides the driver with navigation to the restaurant.

Alternative Flows:

- **A1: No Driver Accepts:** If no driver accepts the offer within a timeout period, the system may increase the payout amount and re-ping drivers. If still unsuccessful, it notifies the customer of a delay or cancels the order.
- **A2: Driver Cancels:** After step 3, if the driver must cancel, the system returns the order to the pool of available offers.
- **A3: Offer Already Accepted:** The system receives an acceptance after the offer has already been assigned to another driver. The system sends a "Offer No Longer Available" notification to the subsequent driver.

UC-4: Driver Picks Up Order

Preconditions: The driver has been assigned to an order and has arrived at the restaurant. The order status is "Ready for Pickup" or "Preparing."

Main Flow:

1. The driver announces their arrival and order number to restaurant staff.
2. The staff hands the packaged order to the driver.
3. The driver confirms the order items on the receipt match the system order.
4. The driver selects "Picked Up" in their app.
5. The system updates the order status to "Picked Up," notifies the customer, and provides the driver with navigation to the customer.

Alternative Flows:

- **A1: Order Not Ready:** The driver arrives and the order is not ready. The driver waits and may notify the customer of a delay via the system.
- **A2: Items Missing/Incorrect:** The driver identifies a discrepancy. The driver requests the restaurant correct the order. If not possible, the driver contacts support via the system.

UC-5: Driver Delivers Order to Customer

Preconditions: The driver has picked up the order and has arrived at the customer's confirmed delivery location.

Main Flow:

1. The driver follows the delivery instructions provided by the customer (e.g., "leave at door," "call upon arrival").
2. For "leave at door" orders, the driver takes a photo of the order at the doorstep as proof.
3. The driver delivers the order to the customer.
4. The driver selects "Delivered" in their app.
5. The system updates the order status to "Delivered," completes the transaction, and prompts the customer to rate the experience.

Alternative Flows:

- **A1: Customer Not Available:** The driver cannot complete the delivery. The driver contacts support via the app. Support attempts to contact the customer. If unresolved, the driver may be instructed to leave the order in a safe place or dispose of it.
- **A2: Customer Disputes Delivery:** The customer claims non-receipt. The system triggers a dispute process. An administrator reviews GPS data, the delivery photo, and history to resolve the issue by denying the claim, issuing a refund, or providing a promo credit.

UC-6: Customer Registers an Account

Preconditions: The user has accessed the application or website.

Main Flow:

1. The user selects "Register."
2. The user provides an email address and password, or chooses to register with a social media account.
3. The system validates the email address is not already in use.
4. The system sends a verification email or SMS code.
5. The user enters the verification code.
6. The system creates the account and prompts the user to add a default delivery address and a payment method.

Alternative Flows:

- **A1: Email Already Registered:** In step 3, if the email exists, the system informs the user and suggests a password reset.

UC-7: Restaurant Manages Its Menu

Preconditions: The restaurant partner is registered and logged into the restaurant administration portal.

Main Flow:

1. The restaurant staff navigates to the "Menu Management" section.
2. The system displays the current menu structure.
3. The staff selects "Add New Item."
4. The staff enters item details (name, description, price, category, allergens, image).
5. The staff saves the new item.
6. The system adds the item to the live menu, making it available for customers to order.

Alternative Flows:

- **A1: Edit Item:** The staff selects an existing item, makes changes, and saves them. The changes are immediately reflected on the live menu.
- **A2: Deactivate Item:** The staff toggles an item's status to "Unavailable" to temporarily remove it from the live menu (e.g., out of stock).

UC-8: Process Payout to Restaurant & Driver

Preconditions: Delivered orders (Status: Delivered) exist from a completed time period.

Main Flow:

1. The system's scheduled payout job triggers (e.g., weekly).
2. The job compiles all delivered orders for each restaurant and driver for the past payout period.
3. For each restaurant, the system calculates the total order value minus the platform's commission.
4. For each driver, the system calculates the total delivery fees and tips earned.
5. The system initiates a bank transfer (e.g., via ACH) for the calculated amounts to the registered bank accounts of each restaurant and driver.
6. The system updates internal financial records and makes a detailed payout statement available to each payee.

Alternative Flows:

- **A1: Bank Transfer Fails:** The transfer is rejected. The system automatically retries after 24 hours. After X failed attempts, it suspends future payouts, flags the account, and notifies the payee and a finance administrator to resolve the issue.

UC-9: User Views Real-Time Order Status

Preconditions: An order has been placed and exists in the system.

Main Flow:

1. The customer navigates to their "Order History" or "Current Orders" screen.
2. The system displays a list of the customer's orders.
3. The customer selects the active order.
4. The system displays the current status of the order on a visual timeline (e.g., Ordered, Preparing, Driver Assigned, Picked Up, On The Way, Delivered).
5. For statuses involving a driver, the system displays a map with the driver's real-time location and estimated time of arrival.

Alternative Flows:

- **A1: Status Not Updating:** The customer perceives a lack of updates. The system provides a "Help" or "Contact Support" button within the order view.

UC-10: Administrator Manages User Accounts

Preconditions: The administrator is authenticated and has elevated privileges to access the admin dashboard.

Main Flow:

1. The administrator navigates to the "User Management" section.
2. The system displays a list of users (customers, drivers, restaurants) with filter and search options.
3. The administrator selects a user account to review.
4. The administrator reviews the user's details, documents, and activity history.
5. The administrator takes an action (e.g., verifies a driver's documents, deactivates a customer due to Terms of Service violations).
6. The system applies the change, logs the administrative action, and notifies the user if their account status has changed.

Alternative Flows:

- **A1: Reactivate Account:** The administrator searches for a deactivated account and changes its status back to "Active." The system notifies the user that their account has been reinstated.

Conclusion

This document provides a foundational specification for a food delivery system. The identified stakeholders outline all parties with a vested interest in the system's operation. The analysis of their conflicts reveals critical design challenges that must be addressed to ensure ethical and sustainable operation. Finally, the ten detailed use cases describe the core functionality required to facilitate the primary business transaction—getting food from a restaurant to a customer—while accounting for the necessary operational and administrative support. This document is now ready for review by development, design, and business stakeholders to guide the next phases of implementation.