Stakeholders in WolfCafe:

A. Primary Stakeholders (direct actors in the system's use cases)
These are the roles the WolfCafe system was explicitly designed to serve.
Customer

- Role: End-user who browses the menu, selects items, places orders, pays with tax and tips, and later picks up the order.
- Needs: Easy-to-use ordering interface, transparency on tax/tip, real-time order status updates, accessible UI.

Why primary? Without customers, WolfCafe has no business value; almost every use case is triggered by them.

Staff (Baristas/Kitchen workers)

- Role: Operate the backend workflow by preparing orders, managing the order queue, fulfilling requests, and keeping inventory up to date.
- Needs: Efficient dashboard to view/filter orders, clear instructions on preparation, fast inventory tools to avoid bottlenecks.

Why primary? Staff turn customer requests into actual service; they ensure quality and timely delivery.

Administrator (Admin)

- Role: Manages roles and permissions, configures tax rate, and oversees the system's integrity.
- Needs: Secure tools to add/edit/delete users (both staff and customers), enforce compliance with NC sales tax, configure policies.

Why primary? Admins maintain the system and keep it aligned with business and legal requirements.

B. Secondary Stakeholders (supporting roles, not in use-case flows but directly influence requirements)
Teaching Assistants (TAs)

- Role: Act as project managers in your course context, reviewing PRs, ensuring features meet "done" criteria (backend API demo, frontend demo, test coverage).
- Needs: Clear documentation of requirements, working features to validate, traceability from requirements to code/tests.

Why secondary? They don't "use" WolfCafe as customers or staff, but their managerial oversight shapes requirements and acceptance.

Instructors (CTOs/Product Owners)

- Role: Define the problem statement, set priorities, act as product owners.
- Needs: System built according to scope (roles, orders, permissions, privacy), ability to measure progress against deliverables.

Why secondary? They represent the "business owners" of WolfCafe in this academic setting.

QA & Accessibility Reviewers

- Role: Verify that the system passes unit and acceptance tests, and ensure accessibility (color contrast, keyboard navigation, tooltips).
- Needs: Testable requirements, working accessibility features, coverage reports.

Why secondary? They ensure non-functional requirements are met, especially test coverage and accessibility.

C. Tertiary Stakeholders (external influences, indirect users)
Payment Notification Services (if integrated)

- Role: External systems for handling payments or sending customer notifications (e.g., credit card gateway, email/SMS service).
- Needs: Secure, reliable API interactions; error-handling when unavailable.

Why tertiary? They don't belong to WolfCafe but are essential for completing transactions and keeping customers informed.

Regulatory/Policy Bodies

- Role: Define compliance obligations such as NC's 2.0% food sales tax, privacy regulations for data collection/storage.
- Needs: System enforcement of tax, audit trails, privacy policy adherence.

Why tertiary? They don't log into the system but heavily constrain its design.

Future Developers / Maintainers

- Role: Extend or maintain WolfCafe after initial release, using the Developers' Guide.
- Needs: Clean code, documentation, clear branching practices.

Why tertiary? They aren't present-day users but their needs shape documentation, modularity, and maintainability requirements.

Stakeholder Bias Conflicts in WolfCafe

1. Customer Convenience vs. Staff Workflow Efficiency

   Customer bias: Wants unlimited order modifications, instant preparation, and the ability to change orders after submission

   Staff bias: Needs predictable preparation workflows, standardized recipes, and locked orders to maintain kitchen efficiency

   Clash: Customer demand for "extra hot, half-caf, oat milk latte with 2 pumps vanilla" creates complex preparation steps that slow down staff, especially during peak hours when the order queue is full

2. Administrator Control vs. Customer Privacy Expectations

   Admin bias: Wants detailed user data collection, order tracking, comprehensive audit logs for system monitoring and business analytics

   Customer bias: Expects minimal personal information requirements, anonymous ordering options, and data privacy protection

   Clash: Admin needs for "Create a privacy policy for WolfCafe" conflict with collecting extensive customer data - the more data collected, the more complex privacy compliance becomes

3. TA/Instructor Academic Requirements vs. Real-World User Experience

   TA/Instructor bias: Needs demonstrable features that meet specific grading criteria (70% test coverage, accessibility compliance, working demos)

   Customer/Staff bias: Wants intuitive, fast, reliable system that "just works" without academic constraints

   Clash: Academic requirement for "Backend demo through API calls" means development time spent on technical demonstrations rather than user experience refinement

4. Staff Operational Needs vs. System Security Requirements

   Staff bias: Wants quick login, shared terminals, easy access to order management without authentication barriers during rush periods

   Admin/Security bias: Requires individual user authentication, session timeouts, role-based access controls, and audit trails

   Clash: Security measures like frequent re-authentication slow down staff during busy periods when they need rapid access to fulfill orders

5. Customer Cost Sensitivity vs. Business Revenue Requirements

   Customer bias: Wants transparent pricing, minimal fees, optional tipping, and value for money.

   Business/Admin bias: Needs revenue generation through required NC 2.0% tax, encouraged tipping (15%, 20%, 25% options), and potential service fees

   Clash: The "Human Flourishing" requirement to consider economic impact conflicts with maximizing revenue - higher prices may exclude lower-income students from using WolfCafe

Comment on prompt crafting:

When using zero-shot prompting we observe that all the AI models give their own version of answer and format stuff the way they want it. Since no previous instructions are given to it the output is faster as well. Although this leads to generic results which usually lack context and specificity. Zero shot prompting is more efficient for simple, well-defined tasks.

Careful prompting involves engineering the prompt to guide the model to a specific desired output. For example, when I told chatgpt to give me use cases for the wolf cafe "Write at least 10 use cases (=5 pages total): Each with: Preconditions, Main Flow, Subflows, Alternative Flows." It properly outputted the use cases in the given format forcing the model to think within a specific framework, aligning its output with user's specific needs whereas without this format for the use cases the gpt just output generic stuff in the output the way it wanted. Basically using careful prompting requires more effort from the user but yields significantly higher quality, more relevant, and structured results making the output actually useful.

This shows that careful prompting requires more upfront effort from the user, but the payoff is substantial. The outputs are typically more accurate, structured, and aligned with the user's goals. In practice, this reduces the time needed for post-processing or reformatting, since the model is already constrained to deliver content in a usable format.

Additionally, careful prompting can make the model behave more like a domain expert rather than a general-purpose assistant. By embedding context, constraints, and formatting requirements directly in the prompt, users can extract higher-quality, context-aware, and actionable outputs. This is especially useful for tasks like drafting reports, generating technical specifications, or simulating real-world scenarios.

Use Cases:

UC-1: Customer Places Order

1. Preconditions

Customer is logged in (or guest ordering allowed).

Menu items exist and are available in inventory.

2. Main Flow

Customer browses menu.

Customer selects one or more items.

Customer adds them to cart.

Customer proceeds to checkout.

System displays subtotal, tax, and tip options (15%, 20%, 25%, custom).

Customer selects tip and confirms payment.

System processes payment.
System creates order and adds it to queue.
Customer receives confirmation.

3. Subflows

[Select Tip]: Customer picks a suggested or custom tip.
[Tax Calculation]: System applies current tax rate from Admin settings.

4. Alternative Flows

[Payment Failed]: Order not created, customer notified, retry option.
[Out of Stock]: Item removed from cart, customer prompted to adjust.
[Cancel Before Payment]: Cart cleared, no order created.

UC-2: Staff Views and Fulfills Orders

1. Preconditions

Staff is logged in.
There are pending orders in the queue.

2. Main Flow

Staff navigates to Orders screen.
System displays pending orders.
Staff selects an order.
Staff prepares the order.
Staff marks order "Fulfilled."
System updates status.

3. Subflows

[Filter Orders]: Staff sorts/filter by time, size, or urgency.

4. Alternative Flows

[Item Unavailable]: Staff cancels order → customer notified.
[Staff Error]: Fulfillment status can be reverted.

UC-3: Admin Manages Users

1. Preconditions

Admin is logged in.

2. Main Flow

Admin navigates to User Management.
Admin creates, edits, or deletes a user.
Admin assigns or updates role (Staff/Customer).
System saves changes.

3. Subflows

[Create User]: Enter details + assign role.
[Edit User]: Update name, role, or details.
[Delete User]: Confirm deletion.

4. Alternative Flows

[Invalid Data]: System rejects and requests correction.
[Active Staff Deletion Attempt]: Block deletion if orders pending.

UC-4: Staff Manages Inventory

1. Preconditions
Staff is logged in.
2. Main Flow
Staff navigates to Inventory screen.
Staff adds new items or updates stock.
System validates input.
System updates inventory records.
3. Subflows
[Add Item]: Enter item details.
[Update Quantity]: Adjust count for existing item.
4. Alternative Flows
[Invalid Entry]: Reject negatives/non-numeric input.
[Duplicate Item]: Merge/update instead of duplicate creation.

UC-5. Admin Sets Tax Rate

1.Preconditions:
Admin is logged in.
System has orders that require tax application.
2.Main Flow:
Admin navigates to Settings → Tax.
Admin inputs new tax rate (e.g., 2.0%).
System validates input.
System updates tax rate configuration.
All future orders use the new rate.
3.Alternative Flows:
Invalid rate (negative, more than 100%) → System rejects input.
Admin cancels → No changes saved.

UC-6: View Order History (Customer)

1. Preconditions:
Customer logged in.
2. Main Flow:
Customer navigates to "My Orders."
System displays past orders, totals, tips, taxes.
3.Alternative Flows:
[No History] System shows empty state.

UC-7. Customer Picks Up Order

1. Preconditions:
Order is fulfilled by staff.
Customer is logged in and waiting for notification.
2. Main Flow:
System updates customer's UI with "Order Ready" notification.

Customer arrives at pickup counter.

Staff hands order to customer.

Customer confirms pickup (via app or staff system).

System marks order as "Completed."

3. Alternative Flows: Customer delays pickup → Order remains in fulfilled state; system can set a time limit.

Wrong customer attempts pickup → Staff verifies order ID/receipt before handoff.

UC-8: Admin Assigns Roles & Permissions

1. Preconditions

Admin is logged into the system.

User accounts already exist in the system.

Roles and permissions are defined in the configuration (e.g., Staff, Barista, Manager, Customer).

2. Main Flow

Admin navigates to the User Management panel.

System displays a list of existing users.

Admin selects a user to manage.

System displays the user's current role and assigned permissions.

Admin selects a new role from the available options (e.g., Staff, Barista, Manager).

System automatically assigns the default permissions for the chosen role.

Admin optionally adjusts specific permissions (grant/revoke).

Admin confirms and saves changes.

System updates the user's role and permissions.

System displays a confirmation message to the admin.

3. Subflows

[View Role Details]: Admin may request to view what permissions are included in a role before assigning.

[Custom Permission Assignment]: Admin can manually override or add permissions beyond the default role set.

[Audit Logging]: System records the change for accountability (user ID, role assigned, timestamp, admin ID).

4. Alternative Flows

[Invalid Role Selection]: If the admin selects a role that does not exist in the system, the system rejects the change and displays an error.

[Insufficient Admin Rights]: If the logged-in admin does not have sufficient authority (e.g., Staff Admin cannot assign "Manager" role), the system blocks the operation.

[System Error During Save]: If database update fails, system notifies admin and no changes are saved.

UC-9: System Sends Order Notifications

1. Preconditions:

User (customer/staff) has active session with notifications enabled.

2. Main Flow:

Event triggers notification (order ready, order delayed, low inventory).

System generates notification [Generate Notification].

Notification delivered via channel (in-app, SMS, email).

User views and acknowledges.

3. Subflows:

[Generate Notification] Use templates: Order Ready, Order Cancelled, Delay, etc.

4. Alternative Flows: [Notification Failed] Delivery channel offline → system retries with fallback (email if push fails).

UC-10: Admin Generates Sales Report

1. Preconditions:

Admin is logged in.

2. Main Flow:

Admin opens "Reports."

Admin selects date range and metrics [Select Range].

System queries order history.

System compiles totals (revenue, orders, tips, taxes).

System displays results with option to export [Export].

3. Subflows:

[Select Range] Admin chooses daily/weekly/monthly period.

[Export] Export as CSV/PDF.

4. Alternative Flows:

[No Data] Empty report generated with message.

[Export Failure] Retry or provide backup format.

UC-11: Customer Cancels Order

1. Preconditions

Customer has placed an order.

Order is in "Pending" or "In Queue" state (not yet fulfilled).

2. Main Flow:

Customer navigates to "My Orders."

Customer selects the active order.

Customer clicks "Cancel Order" [Request Cancel].

System verifies order is not fulfilled.

System processes refund [Process Refund].

Order status updated to "Cancelled."

Customer receives confirmation.

3. Subflows:

[Request Cancel] Customer taps cancel button → confirmation modal.

[Process Refund] System reverses charge with payment gateway, updates balance.

4. Alternative Flows:

[Fulfillment Started] Order already being prepared → cancellation not allowed. Show message.

[Refund Failure] Payment gateway error → flag issue, notify support.