# Culinary Command

DESIGN DOCUMENT

**Team:** 44

**Client:** Paul Durr

**Advisor:** Simantra Mitra

**Members:** Wyatt Hunter, Ryan Rockey, Kevin Tran, Anthony Phan, Matayas Durr

**Team Email:** sdmay26-44@iastate.edu

**Team Website:** https://sdmay26-44.sd.ece.iastate.edu

# Executive Summary

Our team has learned how to approach software design with a focus on both technical feasibility and user needs. We have gained experience in defining clear requirements, evaluating technologies for scalability, availability, and security. At the same time, aligning the design decisions with real-world use cases in the restaurant industry. Early work on researching Blazor and AWS services has provided insight into modern full-stack web application practices, whole discussions around multi-location and role-based access have emphasized the importance of maintainability, usability, and security in professional applications.

---

## Learning Summary

### Development Standards & Practices Used

List all standard circuit, hardware, and software practices used in this project. List all the engineering standards that apply to this project that were considered.

- Software Development Practices: Modular Design, testing, Git Management, Security, and Code reviews.
- Security Practices: Role-Based Access Control, secure authentication,
- Database Practices: Relational Database design, CRUD Operations

### Summary of Requirements

- Multi-location restaurant support
- Role-Based Authentication and Authorization
- User Accounts (manager, admin, worker, etc)
- Dashboard for tasks, prep-lists, and inventory
- Inventory Catalog with CRUD operations
- Secure database for storing user and restaurant data
- Scalability for future integration with API's
- Maintainability with structured design and documentation

### Applicable Courses from Iowa State University Curriculum

- COMS 309
- SE 319
- SE 339
- COMS 3110
- SE 421
- SE 317
- COMS 363

### New Skills/Knowledge Acquired That Were Not Taught in Courses

List all new skills/knowledge that your team acquired which were not part of your Iowa State curriculum in order to complete this project.

- Blazor/.NET Core Development: Building interactive web applications with a modern component-driven framework.
- AWS Cloud Deployment: Hosting applications in the cloud, using AWS IAM for secure access management.
- Multi-Tenant Architecture: Designing a system capable of supporting multiple restaurants under one application.
- API Research and Integration: Exploring API's for the development of data reports with orders and inventory.
- AI Research: Looking for potential places where the application can implement AI.

---

# Table of Contents

## List of figures/tables/symbols/definitions (This should be the similar to the project plan)

# 1 Introduction

### 1.1. Problem Statement

Running a restaurant is more than cooking food, it requires careful coordination behind the scenes to make sure the right ingredients are prepped, the right tasks are assigned, and nothing falls through the cracks. Many restaurants, including the client's, still rely on paper lists, spreadsheets, or memory to track work and inventory. These methods often lead to mistakes such as duplicated work, missing items, wasted food, and miscommunication between staff members. The problem becomes even more difficult when multiple restaurant locations are involved, since managers need to coordinate across teams while maintaining efficiency and consistency.

This challenge is not unique to one restaurant. Across the industry, small and mid-sized restaurants face the same issues of organization and communication, especially as they try to compete with larger chains that often have access to advanced management systems. When prep work and inventory are mismanaged, the impact is felt at every level. Owners lose money, staff waste time, and customers experience slower service or inconsistent quality.

Our project, Culinary Command, is designed to address this problem by creating a single, easy-to-use platform that brings all of these tasks together. Instead of scattered notes and spreadsheets, managers and staff will be able to use one secure web application to create prep lists and assign responsibilities to employees across multiple locations. By simplifying the process for the employees and making it more transparents, Culinary Command helps restaurants save time, reduce waste, and improve communication. Most importantly, it empowers smaller restaurants to operate at the same level of organization and efficiency as larger competitors.

### 1.2. Intended Users

The primary intended users of Culinary Command are restaurants, more specifically restaurant managers. Managers who are responsible for coordinating the daily operations of back of house and front of house operations. These daily tasks could span from prep work, managing inventory, and having smooth communication and interactions with the staff. With the usage of paper lists and spreadsheets, Culinary Command gives an easy and user friendly interface that saves time, reduces inventory waste, and duplicate tasks.

The second key user group is restaurant staff, that would include line cooks and servers. Having Culinary Command, cooks and servers can rely on instructions and tasks to complete accurately. While being under high pressure working conditions, staff and cooks don't have to rely on a paper system that can cause confusion and can be lost or misplaced. With Culinary Command, the staff can receive clear tasks, and have real-time updates from the managers. This helps with reducing stress, creating a more consistent workflow by minimizing errors in day to day operations.

The third group is the restaurant owners. Owners can focus on the profitability and efficiency of their restaurant. Owners often face financial losses due to food waste, inefficient processes, and inconsistent

customer experiences. With Culinary Command as a tool, owners can reduce the operational costs and track food waste. Culinary Command helps owners tackle the financial and operational challenges that are faced.

Beyond restaurant faculty, Culinary Command also directly benefits stakeholders such as business analysts, regional managers, and investors by giving them access to dashboards that provide restaurant metrics. By leveraging artificial intelligence, Culinary Command transforms raw data into actionable insights and provides stakeholders with a platform to make decisions regarding operations, resource allocation, and staffing. These stakeholders often face difficult decisions under challenging circumstances. Culinary Command bridges that gap by delivering timely and data-driven insights so that stakeholders can make well-informed decisions. In doing so, the platform creates value at every level of a restaurant's business: from the line cook on the kitchen floor, to day-to-day management, all the way up to ownership.

# 2 Requirements, Constraints, And Standards

## 2.1. Requirements & Constraints

**Functional Requirements:**

- Allows seamless usage of assigning tasks, viewing lists, and communication with staff
- Visualizing inventory, with visual indicators of low stock
- User account creation, login, logout, and password reset logic
- CRUD operations for prep list and inventory items
- Real-time task tracking with status updates such as: to do, in-progress, or completedIngredient consumption accurately reflects in the database
- Logging of user changes, role changes, and deployments for auditability

**Resource Requirements:**

- Resource/underlying infrastructure deployment environment on AWS
- Development tools such as Git, GitHub, and Figma for version control and visualization

**User Experience Requirements/Constraints:**

- System allows for quick task creations (quick task button, assign preset task to staff member)
- Easy navigation, especially training new users using the app
- Clear labeling of tabs and pages
- Common user actions (marking tasks complete, updating inventory catalog, progress)
- Mobile friendly (phones, tablets)
- Inaccurate touch input into phones or tablets **(Constraint)**

**User Interface Requirements/Constraints:**

- Centralized dashboard (priority list, alerts progression indicators)
- Minimal data entry (dropdowns, autofill, routines)
- Simple/clear navigation (dashboard, task list, inventory, assignments)

**Economic Requirements/Constraints:**

- Using low cost development tech stack (Blazor, AWS Free Tier)
- Paid APIs, relying on paid tools, if uses are beyond free tier limit **(Constraint)**
- Limit the ability to implement AI or machine learning **(Constraint)**

**Security & Compliance Requirements:**

- Passwords must be hashed using a modern algorithm and salted
- Least-privilege principles will be followed where it is applicable
- One sprint focused on security and compliance before release

**Performance Constraints:**

- API request limits, such as Margin Edge with a 1 request per second restriction, must be accounted for when designing and implementing the application
- Application should handle up 75 concurrent users without significantly degrading the application.

### 2.2. Engineering Standards

Engineering standards are important because they make sure everything works together the way it's supposed to. When different people or companies build products, following the same standards helps avoid conflicts and makes things more reliable and easier to connect. It also means that what we create can be used universally and stay adaptable instead of being limited to one setup or system.

ISO/IEC 27001, ISO/IEC 12207, and ISO/IEC 42001 are software standards that have been identified that are most relevant to Culinary Command. ISO/IEC 27001 is a standard that focuses on keeping information secure. It focuses on finding any security risks that Culinary Command might pose and steps to reduce them. Next is ISO/IEC 12207, which defines a roadmap for building software. This standard covers topics ranging from software planning to maintaining software and serves as a framework to organize processes, responsibilities, and deliverables. Culinary Command will be utilizing AI to make data-driven predictions. ISO/IEC 42001 will provide the team with best-practice guidelines to ensure AI is being used appropriately and responsibly.

All three standards are highly relevant to the project. ISO/ISE 27001 is important because the application will handle sensitive information, and following this standard helps protect data from security threats and vulnerabilities. ISO/IEC 12207 is relevant because it provides a structured development environment that keeps the team organized and helps other developers to understand more easily and contribute to the codebase. Finally, ISO/IEC 42001 is relevant since we are now working with AI, and none of the team members have much experience working with AI. Following the standards and guidelines will help the team to implement AI features while learning the best practices along the way.

After reviewing the standards as a team, most of the team members picked ISO/IEC 270001, 12207, and 42001, thinking these related to the project the best. Team 44 also mentioned other good standards that we could follow. Matayas mentioned ISO/IEC 25010, which defines software quality attributes like reliability and maintainability. This would fall under a similar category as 12207 and would be just as beneficial to follow for improving the overall quality of the software. Another standard that was mentioned was the ISO 9001, which focuses on quality management systems. This standard could help the team ensure the development process remains consistent, efficient, and focused on meeting the user needs. While the main focus is on the other three, these additional standards would be integrated as well to strengthen the quality and functionality of Culinary Command.

To incorporate ISO/IEC 27001, 12207, and 42001, the team plans to make many adjustments to the design and development process of Culinary Command.

First, ISO/IEC 27001 will influence how the system handles data and security across the whole application. Since the application will be working with Margin Edge API, which requires secure

authentication and transmits highly sensitive business data, following the ISO/IEC 27001 standards will help design a system that keeps user and restaurant data protected. It will be applied with API keys, possible encryption, and making sure user information is handled safely throughout Culinary Command.

For ISO/IEC 12207, the team will focus more on following a structured software engineering practice throughout the entire development process. This means enforcing consistent coding standards, using proper documentation, and organizing responsibilities within the team. In other words, no more vibe coding. Applying this structure early on will cause a domino effect to make the system easier to understand, debug, and expand on in future deployments. It'll also help keep the backend, frontend, and API integration more aligned.

Lastly, ISO/IEC 42001 will help guide the team through the application's AI features. Even though the AI feature is still in the early stages of planning, making sure it is built responsibly and effectively is a high priority. This standard will guide how the team will design and train AI functionality to ensure its transparency, fairness, and trustworthiness. Whatever the team uses the AI for, following the 42001 standard will help the team create the desired AI feature that is both useful and ethical.

Overall, incorporating these standards will help the team build a system that is structured, secure, and ethical. Setting a solid foundation for both short-term and long-term development goals.

# 3 Project Plan

## 3.1 Project Management/Tracking Procedures

Our team is adopting a Kanban project management style with continuous, iterative development cycles. Kanban is best suited for the project because Culinary Command involves continuous feedback, UI/UX refinement, and backend integration that will evolve throughout both semesters. This flexibility allows the team to adapt requirements, incorporate input, and deliver functionality early rather than waiting for a complete product at the end.

We are using GitHub for both version control and issue tracking. All branches and merge requests are tied to specific user stories or tasks, enabling clear traceability and progress. The Kanban board on Jira will track ongoing and completed work, with milestones representing major deliveries each semester. Team members will create and assign issues weekly and update status during weekly reviews.

Communication and coordination occur through text messaging and Discord for daily check-ins and quick questions/feedback, and the team has weekly team meetings for higher-level planning. Then once bi-weekly, the team meets the advisor to discuss what the team has done, improvements the team could add, or any other advice that the advisor might have. The team also maintains documentation in a shared Google Drive to record decisions, designs, and meeting notes. This combination of tools ensures transparency, accountability, and efficient tracking across all aspects of the project.

## 3.2 Task Decomposition

In order to solve the problem at hand, it helps to decompose it into multiple tasks and subtasks and to understand interdependence among tasks. This step might be useful even if you adopt agile methodology. If you are agile, you can also provide a linear progression of completed requirements aligned with your sprints for the entire project.

To manage development effectively, the project has been decomposed into several key components, each further divided into subtasks. This structure helps define dependencies and allows parallel work during sprints.

Major Tasks and Subtasks:

- Database and Backend
    - Design database schema (User, Task, and Other Future tables)
    - Implement entities, migrations, and data models.
    - Connect API endpoints to the database.
- Frontend / UI Development
    - Build Blazor components for login, dashboard, and task management
    - Implement data binding and form validation
    - Integrate backend APIs and UI actions
- Authentication and Authorization
    - Implement role-based access control (Admin, Manager, Employee)
    - Secure user sessions and data transactions (Cognito)
- Integration and Testing
    - Perform unit and integration tests on both frontend and backend
    - Conduct user acceptance testing before major deliverables (Second Semester)
    - Perform Playwirght testing for the UI
- Documentation
    - Maintain developer guides and API documentation
    - Deploy prototype on the team VM environment

Each sprint will focus on a subset of these tasks, beginning with backend infrastructure and proceeding toward full UI integration and testing. The linear progression will ensure that by the end of the first semester, the database and core logic are functional, and by the end of the second, all frontend features and integrations are complete.

### 3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

**Phase 1 - Design and System Planning**
**Milestone:** Finalize tech stack and architecture using data flow diagrams.
**Metric:** 100% of the frontend and backend systems are fully defined and developed.
**Progress Tracking:** Using Gitlab Issues to track progress and setup of tasks.
**Phase 2 - UI/UX Mockups**
**Milestone:** Create UI mockups (key pages, dashboard, task list, inventory list).
**Metric:** 80% of main pages are completed to allow for implementation.
**Progress Tracking:** GitLab Issues and weekly meetings for feedback.
**Phase 3 - Functionality Development**
**Milestone:** Implementation of majority of backend and frontend modules (Task management, inventory tracking, Database CRUD operations).
**Metric:** 90% of tests pass and no bugs are within the modules.
**Progress Tracking:** CICD being monitored through GitLab pipelines.
**Phase 4 - Integration of Metrics**
**Milestone:** Develop a dashboard with real time metrics and visualizations.
**Metric:** Data accuracy is at least 99% accurate.
**Progress Tracking:** API testing and reviewing results using MarginEdge API calls
**Phase 5 - Testing**
**Milestone:** Start testing while gathering feedback from tests and users
**Metric:** Less than 10% of bugs are found user is satisfied
**Progress Tracking:** Using GitLab issues to test for bugs or errors and using weekly meetings to discuss and summarize progress
**Phase 6 - Deployment**
**Milestone:** Releasing the web application to live users

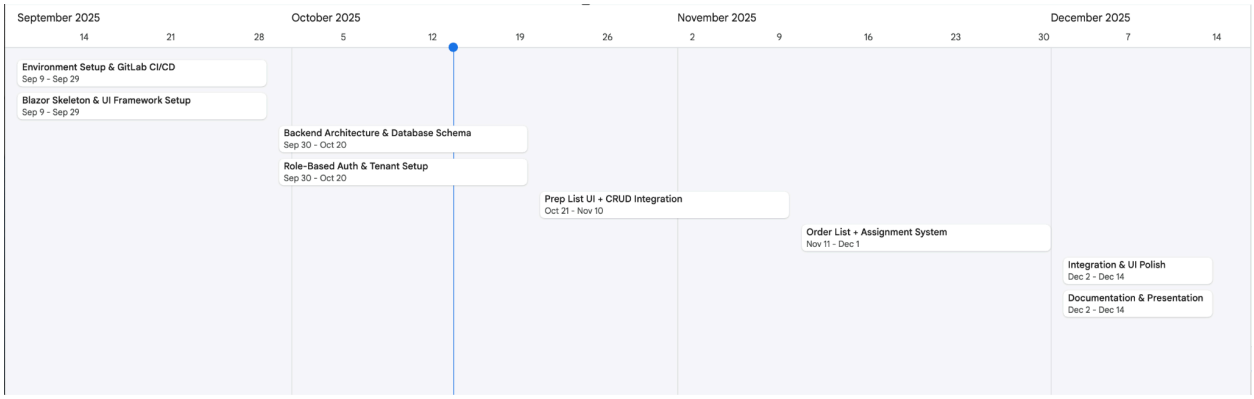
**Metric:** Keeping the system up 99% of the time
**Progress Tracking:** GitLab Issues and monitoring data accuracy,

Using an agile development approach with 2 week sprints, which allows the team sufficient time to complete work during each cycle. To evaluate how the team is progressing, the team will meet midway through the week, every week, for reviewing progress, addressing challenges, and discussing updates. The team will use GitLab to track the issues and commits from other teammates. Along with team meetings, there will be biweekly meetings with advisor to talk about progress, milestone progression, and feedback on progression

### 3.4 Project Timeline/Schedule

Our team is following an agile development model organized into 3 week sprints. This way we can ensure measurable progress and adaptability. This timeline is organized into focused iterations, balancing planning and flexibility while reducing scheduling risk. This Gantt chart maps the project's key tasks and subtasks highlighting sprint milestones and deliverable checkpoints throughout this semester.

| Sprint | Dates | Primary Technical Goals | Deliverable / Milestone |
|---|---|---|---|
| **1** | Sept 9 – Sept 29 | Environment setup, GitLab CI/CD, Blazor project scaffold | Initial architecture verified |
| **2** | Sept 30 – Oct 20 | Backend structure, database schema, role-based auth | Secure login prototype |
| **3** | Oct 21 – Nov 10 | Prep List UI + CRUD integration | Functional prep-list module |
| **4** | Nov 11 – Dec 1 | Order List + employee assignment system | Order-management feature` |
| **5** | Dec 2 – Dec 14 | Integration testing, UI polish, documentation | Final prototype & Deliverable 2 presentation |




Each sprint contains short, trackable subtasks (design, implementation, test, review) to reduce estimation errors and ensure steady incremental progress. Deliverables and sprint completion dates are annotated directly on the Gantt chart.

### 3.5 Risks and Risk Management/Mitigation

For Culinary Command, the main risks that the team faces include learning new technologies (0.8), handling new project features and requirements (0.6), and receiving timely feedback (0.7). These risks pose potential threats to the project timeline and overall project quality. To ensure the team stays on track, strategies were developed for each risk.

Technologies such as AWS and Blazor pose a significant risk to the quality of Culinary Command. These technologies have steep learning curves which can delay the teams' sprint velocity, resulting in the team missing deadlines as well as impact on feature quality. To mitigate this risk, each team member will be assigned to a technology. This will allow each team member to focus on one technology to build expertise, ensuring that we can maximize the value of each technology in the core architecture.

Agile methodologies will help the team handle new features that arise during the working weeks. More specifically, staying on top of the backlog will mitigate the delay that new features can cause. Additionally, a "parking lot" will be incorporated to the teams' daily meetings to discuss blockers or new features that come up.

The final risk that the team will have to prepare for is receiving timely feedback from the client. Without consistent feedback, the team risks working on features that don't meet project expectations and can lead to significant delays. To mitigate this, the team plans to schedule monthly meetings to streamline the communication process. This will allow the team to receive feedback in a timely manner while being able to discuss any additional comments.

### 3.6 Personnel Effort Requirements

| Feature | Estimated Effort (Hours) | User Stories |
|---|---|---|
| AWS Infrastructure Setup | 75 | Configure IAM roles, web-app hosting, database provisioning, CI/CD pipeline |

| | | |
|---|---|---|
| Blazor Frontend Development | 100 | UI/UX designs, page routing, component development |
| Authentication & Authorization | 40 | Defining RBAC controls for admin/user, research authentication tools (autho), implement authentication/authorization logic |
| Spikes for Technology | 60 | Various AWS service deep dives, functional researching, API research, PoCs, adhering to industry standards |
| Core Feature Implementation | 200 | Implement dashboards, AI predictions, task list, manager views, ingredient tracker, order tracker |
| Continuous Improvement | 45 | Improve pipeline, refactor project directory structure, etc |
| Testing | 40 | Ensure business logic is tested thoroughly, end-to-end testing, smoke testing, integration testing, UI testing |

*Figure 3.1*

For Culinary Command, AWS infrastructure setup, Blazor frontend development, authentication & authorization, and spikes form the technical foundation. AWS setup provides secure IAM, hosting, storage, and CI/CD so deployments are reliable and streamlined. Blazor work delivers the user-facing UI, routing, and reusable components that define the experience. Spikes and auth work reduce unknowns and secure access by validating service choices and integrating a robust provider with RBAC.

Core feature implementation, continuous improvement, and testing deliver and protect product value. Core features build the application's functionality through dashboards, AI predictions, task lists, and ingredient and order tracking that users rely on. Continuous improvement increases velocity and maintainability by refining pipelines and project structure. Testing enforces correctness and reliability through unit, integration, end-to-end, and smoke tests so releases meet client expectations.

It is estimated there will be a total of 16 weeks that will be devoted to working on Culinary Command. From previous conversations, each team member agreed to spend roughly 7 hours each week on the project– this comes out to 560 working hours in total. Based on this number, each feature was estimated according to the number of hours that the team is able to work with and can be seen in Figure 3.1.

## 3.7 Other Resource Requirements

Aside from financial resources, the Culinary Command project requires several technical and operational resources to support development and deployment. These include:

- Computing Resources
    - Local development environments with .NET, Visual Studio Code, and MySQL installed.
    - Developer machines capable of running Blazor Server applications and Playwright tests.
- Software and Development Tools
    - GitHub for version control, CI/CD pipelines, and issue tracking
    - Postman for API testing and validation
    - Playwright for end-to-end and UI testing.
    - dbdiagram.io for visualizing database schemas and architecture diagrams.
    - Telerik Report Designer for creating reports and dashboards
    - Google Drive/DOCS for team documentation and file sharing.
- Cloud and Hosting Resources
    - AWS services for secure application hosting, data storage, and database management, and security (Cognito)
    - MarginEdge API access for data integration and testing real-time restaurant metrics.
- Collaboration and Communication Tools
    - Discord and text messaging for daily communication
    - In-person meetings for the advisor and weekly meetings within the group.
- Testing and Evaluation Resources
    - Access to test accounts
    - sample restaurant data
    - mock APIs for simulating real-world operations.

These resources collectively support the full development lifecycle, from design and implementation to testing, deployment, and presentation of the Culinary Command system.

# 4 Design

## 4.1 Design Context

### 4.1.1 Broader Context

A web application that is designed to solve task management and communication in commercial restaurant kitchens. Managers and kitchen staff can manage and organize tasks that need to be done within a timeframe, increasing efficiency at the back of the house. The application addresses the societal need of modernizing a workplace, reducing stress and poor communication, through providing clear task visibility, real-time tracking, and workflow management. The broad context is the web application aims to improve operational efficiency, reduce food waster, and a more productive and safe kitchen.

List relevant considerations related to your project in each of the following areas:

| Area | Description | Examples |
|---|---|---|
| Public health, safety, and welfare | Ensure that food safety tasks are properly track, promote a safe workplace through training and practice | Putting food into the right temperature, prepping it correctly. |
| Global, cultural, and social | Complying with kitchen regulation in different locations, having an inclusive | New employees will have less experience but should be treated |

| | | |
|---|---|---|
| | diversity, creating a respectable environment | the same as if they been there for years |
| Environmental | Having a technological impact over a lifecycle | Tasks and management will be done over tablets or smartphones. |
| Economic | Having an improved working condition, minimizing the health and social costs and contribution to more technology, growing with cheaper technology usage over time. | Improving the working conditions causing less injuries and food illnesses. |

### 4.1.2 Prior Work/Solutions

The restaurant/food industry is massive, and many companies have dipped their feet in, but CulinaryCommands emphasis on back-of-house operations as well as the compatibility with the MarginEdge API makes it stand out.

| Existing | Description | Pros | Cons |
|---|---|---|---|
| Toast POS | Platform offering point-of-sale and management tools | Ecosystem, reliability | High cost, limited customization for analytics |
| 7shifts | Platform focused on scheduling and team communication | Good team coordination features | Lacks inventory and task tracking features |
| Operandio | Platform focused on daily task tracking and staff communication | Good mobile interface, strong multi-location management | Lacks inventory tracking and analytics features |

Each of these, while being successful in their own right, still lack the back-of-house tools as well as integration with a proven, highly present API in MarginEdge.

References:

[1] Toast, Inc., *"About Us – Toast POS,"* 2025. [Online]. Available: https://pos.toasttab.com/about
[2] 7shifts Inc., *"All About 7shifts,"* 2025. [Online]. Available: https://www.7shifts.com/about/
[3] Operandio Pty Ltd, *"About Us – Operandio,"* 2025. [Online]. Available: https://operandio.com/why-operandio/about-us/

### 4.1.3 Technical Complexity

CulinaryCommand demonstrates technical complexity through its multi-layered architecture, integration of diverse technologies, and incorporation of advanced analytics principles. The system is composed of distinct, independent subsystems, each utilizing different engineering concepts. Looking at the

app through a multi-tiered architecture, the presentation layer is developed with Blazor Server (.NET 9), applies web development principles to deliver a responsive, role-based interface for staff, managers, and administrators. Below that is the application layer, which implements modular C# services such as the AuthService, which manages user authentication and login logic. The data and integration layer uses relational database theory and object oriented mapping to connect application logic with the underlying mySQL database. This layer also introduces additional complexity by interfacing with the external MarginEdge API. Future plans for an AI Analytics Subsystem further increase the project's complexity.

Overall, the project exceeds the technical expectations of a typical web application by combining secure authentication, cloud-hosting, real-time task synchronization, role-based access control, and AI-driven analytics tools.

## 4.2 Design Exploration

### 4.2.1 Design Decisions

The first tradeoff that had to be made was choosing Blazor Server over Blazor WebAssembly. This is because Blazor Server's server-rendered models enable faster feature implementation. The tradeoff is greater reliance on persistent connections and higher server computation under high traffic. To mitigate, the autoscaling logic will be configured for each app "tier".

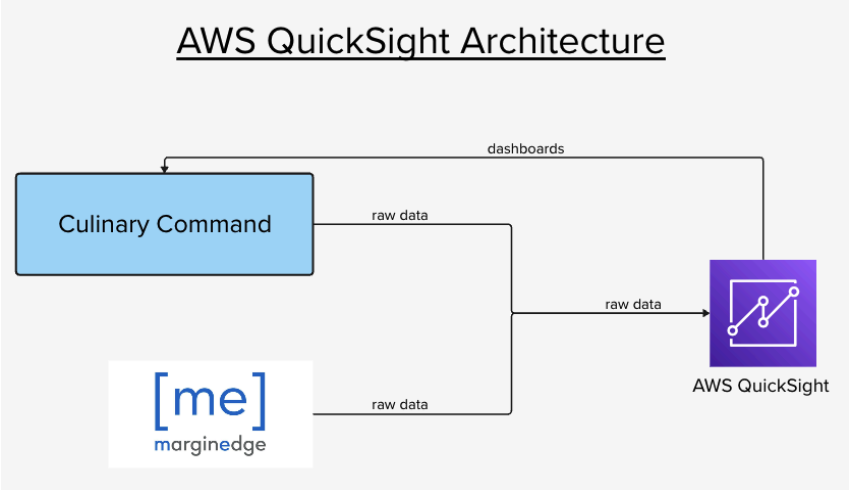
The next design decision that the team made was choosing the proper cloud provider. The team had two options: Azure or AWS. Azure was considered because of its native support for Blazor and ASP.NET, however; AWS was chosen to prioritize team familiarity. This will result in quicker feature delivery and reduced operational mistakes.

The last critical design decision focuses on how Margin Edge data will be analyzed using AI. More specifically, whether to adopt AWS QuickSight for managed business intelligence or build a custom ML pipeline. The former is much cheaper and seems to be the most appropriate service, possibly leading to quicker results.

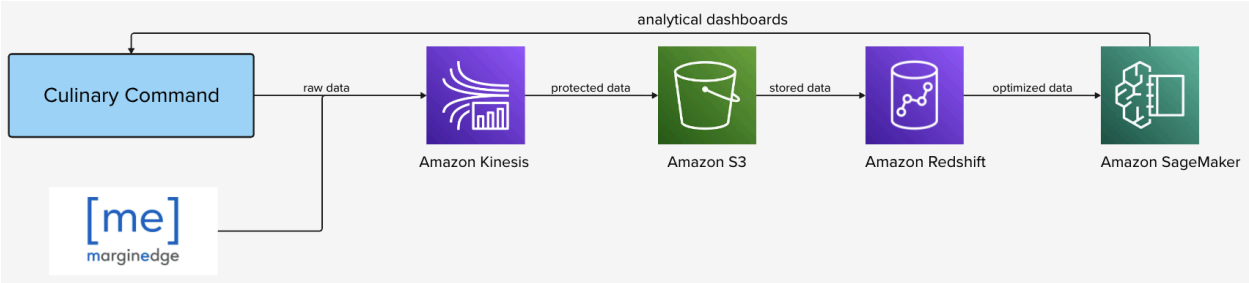
### 4.2.2 Ideation

The team brainstormed potential AI analysis pipeline options. Variations of these options were considered: AWS QuickSight, AWS QuickSight + custom ML pipeline, third-party analytic/ML SaaS.

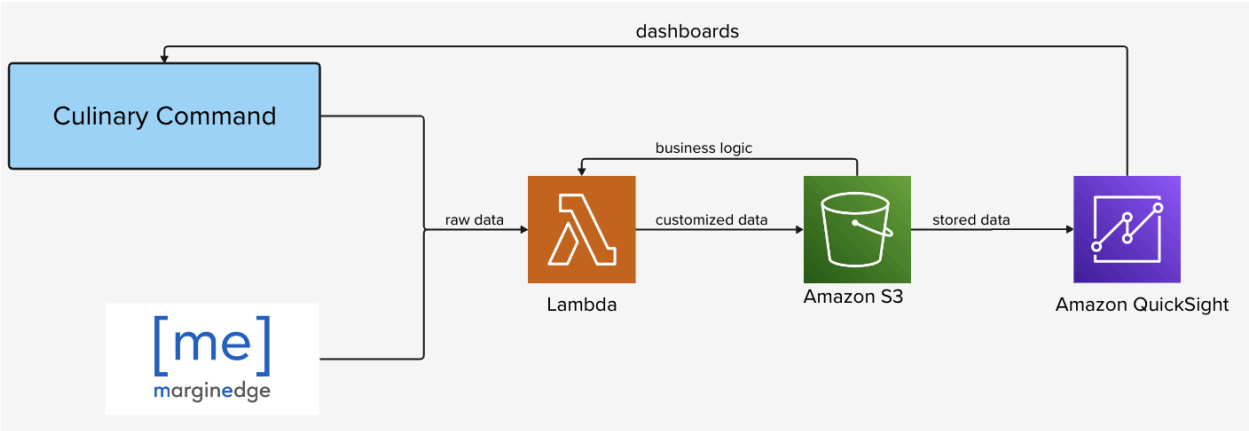
Using AWS QuickSight to directly ingest data for dashboards is the most basic, but results in the fastest time-to-insight and has low operational costs. However, it lacks customization capabilities which is a concern. Due to the limited budget of the team, this architecture is considered:




If the Culinary Command development team wanted to be aggressive and does not care about operational costs, the architecture would look a lot different. Instead, raw data will be redirected to Amazon Kinesis for real-time processing of large data and stored inside of Amazon S3. To ensure that the refined data can be accessed quickly, Amazon Redshift can be used to accelerate query performance. From there, Amazon SageMaker can ingest the data to produce powerful visualizations, see here:

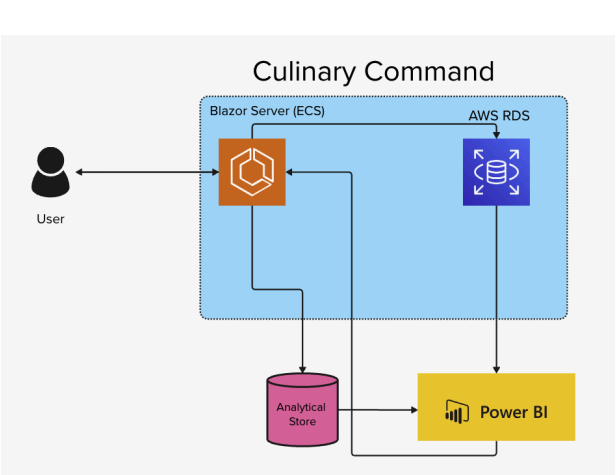


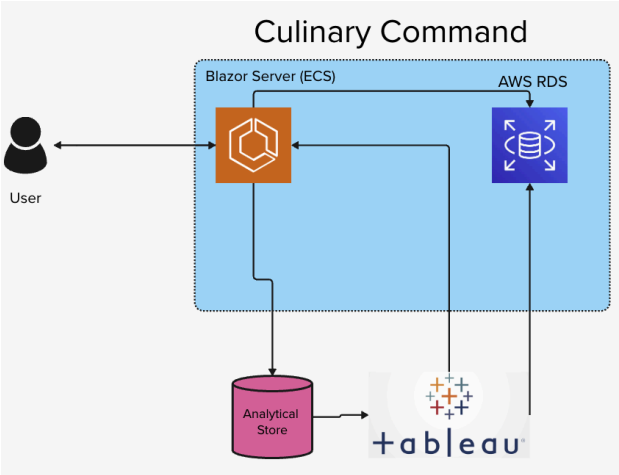

Unfortunately, this architecture is not in the team's budget. A more realistic architecture (where the team wants some customization) would involve a Lambda function, S3 bucket, and Amazon QuickSight. That would look something like:



Data will be processed efficiently by lambda functions and stored inside of S3 for data durability. The data inside of the S3 bucket will be used for business logic and also streamed into Amazon QuickSight to create visual dashboards for the end users.

Third-party services (outside of AWS) such as Power BI and Tableau are also being considered. Although it is unlikely that one of the listed pathways will be the one that the Culinary Command team ends up going down due to lack of familiarity with expertise, they are still viable options to be considered:




### 4.2.3 Decision-Making and Trade-Off

The team conducted an evaluation of the AI analytics pipeline to determine the most practical and cost-effective architecture. Each alternative was assessed on the basis of cost, scalability, ease of integration, and team familiarity. The AWS QuickSight–only solution scored highest in affordability and implementation speed, making it ideal for rapid deployment but limited in flexibility and long-term analytical depth. Other pipelines were more ideal for flexibility and customization, but introduced significant operational costs.

Ultimately, the AWS Lambda + S3 + QuickSight pipeline was chosen as the optimal solution. This architecture has a balanced tradeoff between cost efficiency and customization, allowing for light-weight processing and good visualization capabilities. This solution remains scalable and adaptable for future changes as the project evolves. This decision reflects the team's focus on delivering real-world insights quickly while staying budget friendly and within technical constraints.

## 4.3 Proposed Design

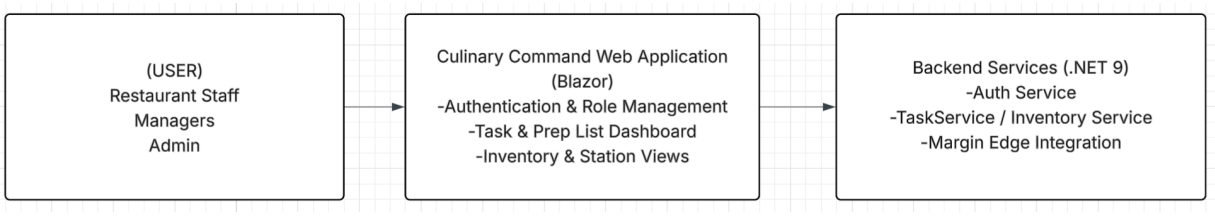
### 4.3.1 Overview

Culinary Command is a web-based restaurant management platform designed to simplify daily kitchen operations by centralizing task organization, inventory tracking, and team coordination into an accessible system. The application helps restaurant teams improve communication and efficiency by ensuring that everyone, from line cooks to managers, shares the same real-time view of responsibilities and preparation needs.

At its core, the system provides role-based user management, allowing managers to assign tasks, monitor progress, and oversee inventory while staff members focus on completing prep lists and reporting updates. The platform's task management subsystem organizes work items by kitchen station, making it easy to distribute and track assignments throughout the day. The inventory subsystem records ingredient quantities and stock levels, helping prevent shortages and streamline restocking.

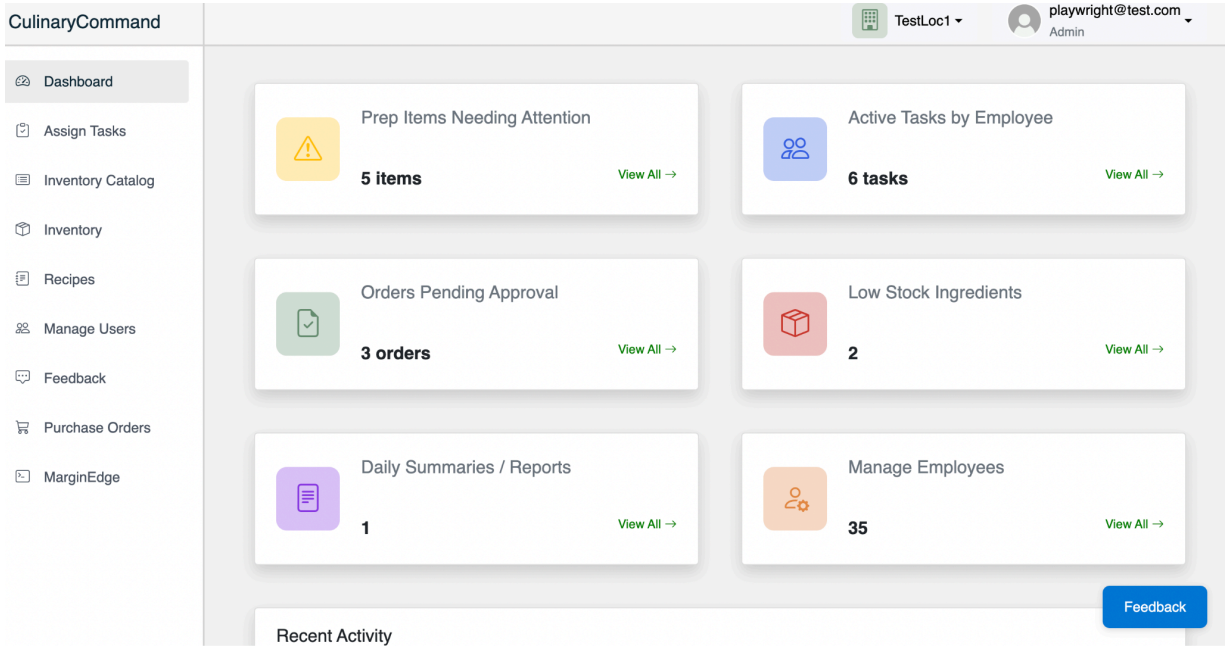
All user actions and data are securely managed through a central MySQL database, hosted locally during development and planned for deployment to Amazon RDS for scalability. Culinary Command's frontend is built using Blazor Server, offering a responsive and interactive web interface accessible from any browser. The system also included a custom authentication service, ensuring that each user's access level aligns with their role and responsibilities.

In the future, Culinary Command will integrate an AI-driven analytics engine that leverages data from Margin Edge to provide predictive insights, such as ingredient demand forecasting and labor optimization. This feature will allow managers to make more informed decisions based on historical and real-time data.




### 4.3.2 Detailed Design and Visual(s)

The detailed design of Culinary Command follows a structured and layered architecture to ensure maintainability, scalability, and clarity for future development. The system is organized into four major layers: the presentation layer, the application logic layers, the data access layer, and the persistence and integration layer. Together, these layers define how user actions in the web interface translate into secure, auditable operations within the database and connected API's.




**Firgure 4.1:** Visualization for an Admin's dashboard view. Sarting with assigning tasks and ending with the MarginEdge API feature.

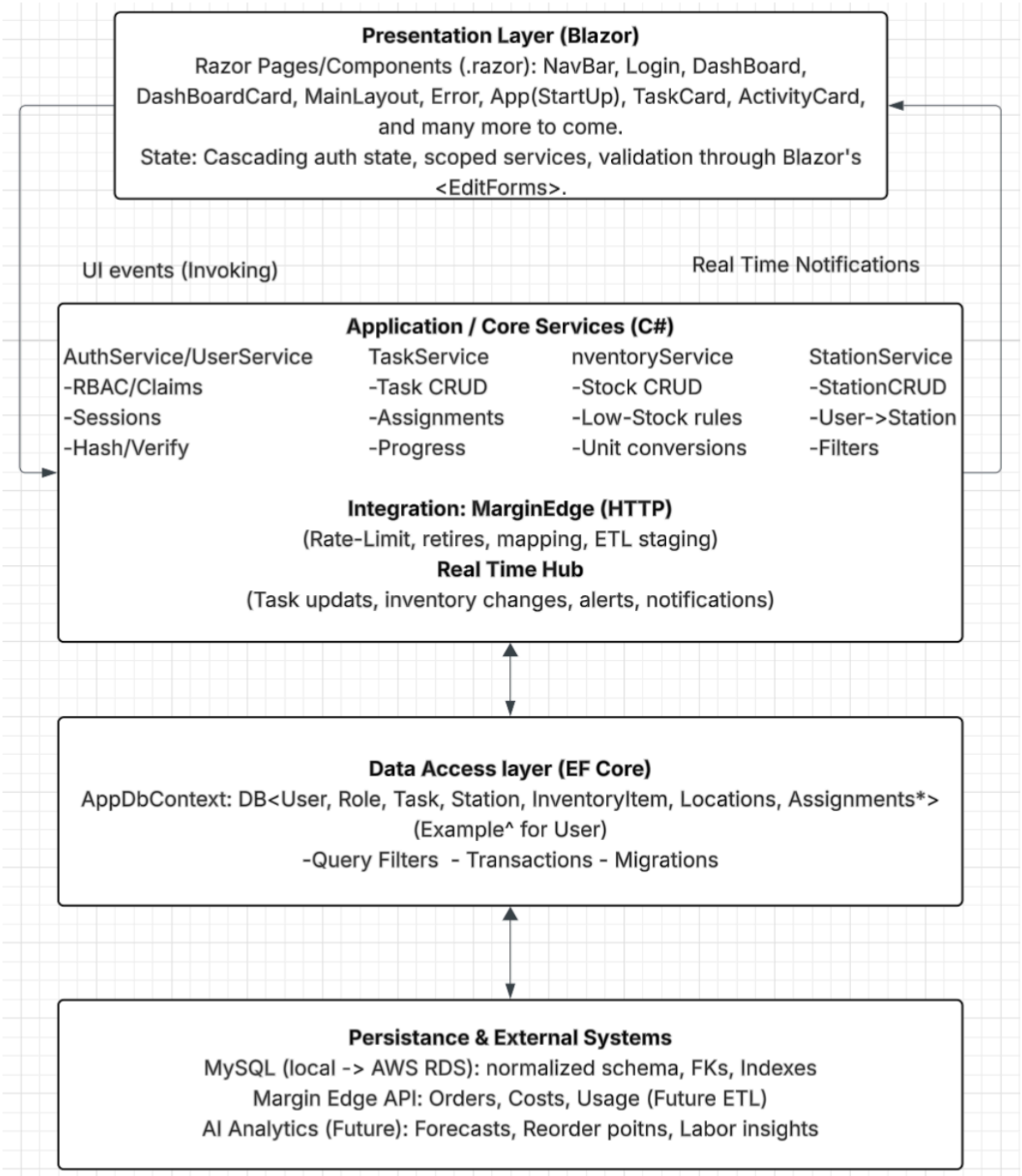
At the top of the stack is the presentation layer, implemented with Blazor Server (.NET9). This layer provides the user interface that the restaurant staff, managers, and owners interact with daily. It includes Razor components for login and autheentication, dashboards for prep lists and inventory management, and administrative views for assigning roles and managing restaurant locations. Each interface is responsive, role-based, and connected directly to the backend through scoped services that validate input, perform authorization checks, and synchronize state changes in real time.

Beneath the interface lies the application logic layer, which contains the core C# services that handle all business logic for the platform. Key components include the AuthService, Task Services, InventoryService, and StationService. The AuthService manages authentication, session validation, and role-based access control for all users. The TaskService is responsible for creating, updating, and assigning tasks, as well as tracking their progress and completion status. The InventoryService tracks ingredients and quantities within each location, automatically updating stock levels and issuing low-stock alerts. The StationService manages kitchen stations and links users and tasks to specific zones within a restaurant. Collectively, these services act as the intermediary between the frontend and the database, ensuring that data is validated, secured, and consistent across all operations.

The data access layer is implemented using Entity Framework Core, which provides object-relational mapping between the application services and the underlying MySQL database. The data context, AppDbContext, defines tables such as Users, Roles, Tasks, Stations, Inventory Items, and Locations/Tenants. Foreign keys enforce relationships to support multi-tenant functionality. The data access layer handles all CRUD operations, enforces transaction safety for multi-site updates, and supports migration as new features and tables are introduced.

At the bottom of the system is the persistence and integration layer, which includes the local MySQL database (currently running in development and planning for deployment on AWS RDS) and external APIs. This layer stores all persistent information, including user credentials, roles, tasks, inventory records, and audit logs. It is also responsible for interacting with Margin Edge API, which provides real-world restaurant data such as ingredient usage, vendor pricing, and purchasing history. This integration uses a custom MarginEdge Client service that handles HTTP requests with rate-limiting and error handling to ensure data consistency and security. In the future, a dedicated ETL pipeline will pull Margin Edge data into a staging area, map it to Culinary Command's schema, and update the main database with validated records.

Building on this foundation, the team plans to develop an AI Analytics Subsystem that leverages Margin Edge data combined with Culinary Command's own historical records to generate predictive insights. This AI module will analyze ingredient usage, order trends, and labor efficiency to forecast demand, suggest optimized reorder points, and identify potential waste. The system will be modular and cloud-hosted, communicating through a REST API endpoint that the Blazor frontend can query to display dashboards and recommendations to managers and owners. These features will follow the ISO/IEC 42001 standard for ethical and transparent AI design.




The figure above illustrates the system's architecture, showing how the frontend interfaces with the core services, database, and external API integrations. The presentation layer communicates with the application logic layer through service calls, which in turn use Entity Framework Core to perform data operations on the MySQL database. Margin Edge data and AI predictions/suggestions flow into the system through the

integration layer and are surfaced back to the user via the manager dashboard. To support scalability, all services are loosely coupled and registered with dependency injection, ensuring that future subsystems can be added without disrupting existing functionality.

By structuring the system into these modular components, Culinary Command ensures that each subsystem can be developed, tested, and expanded independently. The design not only meets the project's current base requirements, but also lays the groundwork for future integration with external APIs..

### 4.3.3 Functionality

Culinary Command is designed to operate as a centralized web application that streamlines daily restaurant management tasks through a role-based, interactive interface. The platform serves multiple user groups: staff, managers, and admins. Each with distinct permissions and functionalities tailored to their responsibilities.

When a manager logs in, they are presented with a dashboard summarizing the restaurant's active prep lists, staff assignments, and inventory status. From this dashboard, managers can create new tasks, assign them to specific stations or employees, and track their progress in real time. The manager can also access inventory data, view low-stock alerts, and adjust quantities directly through the system. All changes are securely logged in the database, ensuring accountability and enabling traceability across all operations.

Staff members access a simplified version of the interface designed for quick and efficient task completion. After signing in, staff can view assignments assigned to their station, mark them as complete, and report ingredient usage or shortages. These updates instantly reflect on the manager's dashboard, ensuring that progress and inventory data remain synchronized across all connected devices.

Admins, Owners, and other higher-level stakeholders have access an advanced reporting and analytics features. In the current systems, they can review data on task completion rates, inventory turnover, and location-level performance. As the project advances, these dashboards will integrate with Margin Edge data to present predictive insights powered by the planned AI analytics subsystem. For example, the system will be able to forecast ingredient demand, identify cost inefficiencies, and suggest optimal reorder points.

### 4.3.4 Areas of Concern and Development

How well does/will the current design satisfy requirements and meet user needs?

Based on your current design, what are your primary concerns for delivering a product/system that addresses requirements and meets user and client needs?

What are your immediate plans for developing the solution to address those concerns? What questions do you have for clients, TAs, and faculty advisers?

While the current design of Culinary Command effectively satisfies the foundational requirements of task management, role-based control, and inventory tracking, few areas remain under active development as the system evolves toward full production readiness.

One key concern is the integration of external data sources, specifically the Margin Edge API. Because the API enforces a strict rate limit of one request per second, careful scheduling and caching

mechanisms must be implemented to ensure reliable synchronization without violating the constraint. As of now, the team does not have real-world data to measure how many requests were to be made during certain hours of the day/week. The team plans to mitigate this by developing a background process that will batch requests during off-peak hours, store results in staging tables, and update the live system incrementally. (If it gets to that point of usage)

The upcoming AI Analytics module also presents challenges in implementation. Since the system will use sensitive business data for Margin Edge, data privacy and transparency will be crucial. The team will adhere to ISO/IEC 42001 guidelines to ensure that AI recommendations are accurate, explainable, and unbiased. In practical terms, this means building the AI system as a separate service with its own data isolation policies and secure API endpoints for communication with the main platform.

User experience remains another important development area. While the Blazor frontend currently provides an intuitive interface, additional usability testing is needed to validate its clarity across different device types, such as tablets and phones. These insights will guide refinements in layout, labeling, and visual hierarchy to ensure quick adaptation by the staff members.

Through continued iteration and collaboration with the client and advisor, Culinary Command aims to deliver a system that not only meets current operational needs but also evolves into a data-driven, intelligent management platform that helps restaurants operate more efficiently and profitably.

## 4.4 Technology Considerations

Culinary Command relies on a collection of modern, cloud based, and open source technologies chosen to balance cost, scalability and maintainability. These technologies are the Blazor server (.NET9), AWS cloud infrastructure, MySQL, Margin Edge API, and an AI analytics subsystem (like AWS quicksight w/ Lambda).

**Blazor Server:**

**Strengths:**

- Allows us to create a very nice and interactive UI using C# instead of JavaScript, which helps with consistency between frontend and backend.
- Has real time UI updates through SignalR connections, which help with task management dashboards and live inventory data.
- Integration with ASP.NET Core helps authentication, dependency, injection and routing.

**Weaknesses/Tradeoffs:**

- Needs to always be connected to the server, which high concurrency can cause increase server load.

**Rationale:**

The team picked Blazor because it was something new that most of us haven't worked on before. It also allows for fast development with the ability to share code with the backend.

**AWS Cloud Infrastructure:**

**Strengths:**

- Very scalable, and is also secure hosting through EC2, RDS, and S3.
- Built in monitoring tools that help in debugging.

**Weaknesses/Tradeoffs:**

- Set up is complex, and there is a potential cost growth as services scale.
- Requires ongoing configuration management and security monitoring.

**Rationale:**

AWS was picked over Azure due to the team being more comfortable with AWS and there was a free trial. The environment also supports long term scalability for multi-tenant restaurants.

**MySQL:**

**Strengths:**

- Supports multi-tenant data schemas.
- Entity Framework Core helps simplify databases interactions with C# queries and migrations.
- Cross platform support can integrate with AWS.

**Weaknesses/Tradeoffs:**

- Performance tuning required for large scale queries.
- The multi tenant relationships may get complex.

**Rationale:**

MySQL was chosen because everybody on the team was familiar with it. It also allows the team to enforce relationships and constraints that ensure data consistency across restaurants and users.

**Margin Edge API:**

**Strengths:**

- Provides real-world restaurant data such as invoices, ingredient usage, and vendor pricing.
- Adds commercial value by enabling AI-based forecasting and insights.

**Weaknesses/Tradeoffs:**

- Enforces a strict rate limit (1 request per second).
- Requires secure key management and API throttling logic.

**Rationale:**

Despite some limitations, the API is crucial for realistic business data. We can mitigate the rate limit risk through caching and off-peak batching strategies.

**AI Analytics Subsystem(AWS QuickSight with Lambda):**

**Strengths:**

- Provides predictive analytics and visualization with minimal infrastructure management.
- Easily connects to S3 data for cost-efficient storage and analysis.

**Weaknesses/Tradeoffs:**

- Less customizable than a fully self-managed ML pipeline.
- Advanced AI features are limited without larger datasets.

**Rationale:**

AWS QuickSight with Lambda was selected as a middle ground solution which offers us analytic capability and affordability. It can also allow us in the future to migrate to SageMaker for custom modeling once we have enough data.

## 4.5 Design Analysis

The Culinary Command system has progressed from initial mockups into functional, full-stack application with core features implemented and actively tested. The frontend, built with Blazor Server, now supports key workflows such as user creation/authentication, role-based dashboards, and ingredient management with recipes and inventory. These components align closely with the original Figma designs, with adjustments made based on usability and development constraints. On the backend, services and database integration using Entity Framwork Core and AWS MySQL are operational, supporting reliable data persistence and multi-tenant behavior. Playwright testing has been introduced to validate end-to-end user flows., confirming the major interactions such as user creation and data updates function correctly across the system. MarginEdge API integration is currently in progress, with on going work focused on handling rate limits, and ensuring consistent data synchronication. AI-based analytics feature remain planned but not yet fully implemented due to limited real-world data. At this stage, the system is stable, meets core requirements, and provides a strong foundation for continued development, testing and feature expansion.

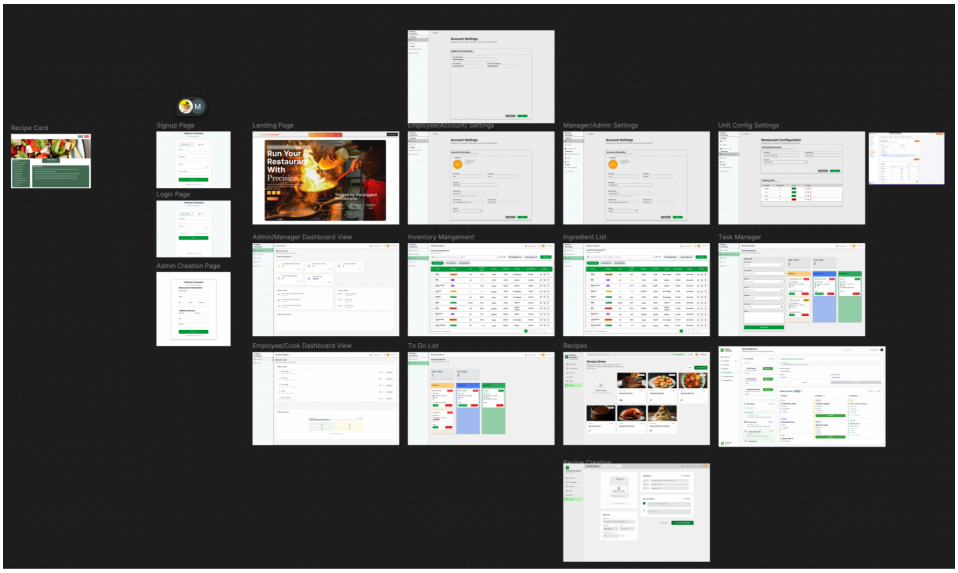



**Figure 4.2:** Figma designs implemented before actual code implementation.

# 5 Testing

Culinary Command uses a layered testing strategy to validate both correctness and usability across the full system. Because the application combines a Blazor Server frontend, backend service logic, database persistence, role-based authorization, and multi-tenant behavior, no single type of test is sufficient on its own. The team therefore uses unit testing for isolated behavior, integration testing for service and database interactions, interface testing for frontend-to-backend communication, and end-to-end testing for full user workflows. This strategy supports early defect detection and reduces the risk of breaking critical restaurant operations such as login, inventory updates, and task assignment. Testing is performed continuously as features are developed rather than only at the end of implementation.

## 5.1 Unit Testing

For unit testing in Culinary Command, the primary focus is on ensuring that the backend logic behaves correctly before it is ever combined with the frontend. Although the main testing tool for the full application is Playwright, Playwright is not intended for strict unit testing. Instead, the unit testing portion of the testing strategy will rely on standard .NET tools such as xUnit and Moq. These tests will target small, isolated pieces of logic such as authentication validation, task creation logic, company creation logic, data transformation, and other functions that prepare objects for the UI or external API's.

The smallest testable units are the backend service classes, repository logic, and utility functions. These include:

- AuthService (valid login, invalid login, token generation)
- Tenant Filtering Logic (ensures a user only sees pages for their selected role and current location)
- Task Management Logic (CRUD)
- Data transformers for MarginEdge product API formatting.

- And more

How they are tested

- Unit tests isolate each method using mocked dependencies so no external systems or databases are touched.
- EF Core InMemoryDatabase is used for repository tests, allowing testing CRUD behavior without hitting the real database instance.
- Mocking frameworks to simulate external API's to confirm correct handling of failure.
- All edge cases are tested. (empty input, bad input, unauthorized access)

Tools:

- xUnit - core unit test framework.
- Moq - mocking services and interfaces
- EF Core InMemory - quic, isolated DB tests
- PlayWrite - component tests and small UI flows

Unit tests allow the team to confirm that individual services perform consistently, even when the user interface or database is not involved. For example, the logic that ensures users only retrieve data for the location they selected, the multi-tenant isolation logic, can be tested entirely in memory using EF Core's InMemory database. By mocking dependencies, it can verify functionality like error handling, invalid inputs, or how services behave when external data is missing. While playwright is not used directly at this layer, the strength of these unit tests improves the reliability of the end-to-end tests that Playwright will eventually run.

### 5.2 Interface Testing

Interface testing focuses on how different parts of the system interact with each other, and this is where Playwright begins to play a major role. Since CulinaryCommand is a Blazor application, many core behaviors, such as submitting a task, switching locations, or loading data, happen through user actions that call backend APIs. Playwright allows us to test these interfaces by simulating real user clicks, inputs, and navigation inside a real browser environment.

With Playwright, Team 44 can observe that exact API calls triggered by the UI, inspect the request and response, and confirm that the system is correctly passing data between the front-end interface and the backend controllers. For example, when a user selects a restaurant location from the LocationSelector component, Playwright can verify that the correct tenant-specific APi endpoint is called and that the data shown on the page truly belongs to that company. This creates a natural, realistic interface test without needing additional tools. If necessary, Postman can serve an alternative by manually verifying the API endpoints, but Playwright will serve as the primary interface testing method because it validates behavior as real-world users would experience it.

### 5.3 Integration Testing

There are many integration paths involved in Culinary Command that need to be tested thoroughly including recipe-to-inventory synchronization, inventory-to-analytics pipeline and authentication enforcement. These paths are considered critical because they satisfy requirements for recipe management, business analysis, and RBAC.

To test each path, the team will be utilizing xUnit, EF Core InMemory, and Playwright as the foundation for integration testing. This provides the team with a structured and consistent approach to testing, ensuring overall high quality.

## 5.4 System Testing

The system level-strategy ensures that CulinaryCommand will satisfy all functional and non-functional requirements, including secure authentication, accurate prep task and inventory management, role-base access, and real-time task coordination across multiple locations. The system-level testing is achieved by combining unit tests, interface/component tests, integration tests, load testing, and full end-to-end tests. This forms a layered approach that verifies the entire system.

1. **Unit Testing**

   Purpose: Test isolated logic in services and entities to ensure correctness before integration.

   Key testing areas:

   a. Authentication:
      i. AuthService verifies password hashing and login validation
   b. Role-based access rules:
      i. UserService ensures that employees, managers, and admins have correct permissions.
   c. Recipe and ingredient logic:
      i. RecipeService and IngredientService correctly link recipes, ingredients, and prep tasks.
   d. Task assignment logic:
      i. Task calculations, including status changes and assignment rules.
   e. Validation and business rules:
      i. Ensuring data integrity in Ingredient, Task, Location, and User entities.

   Tools: xUnit - Run tests on service methods and entity logic

2. **Interface / Component Testing**

   Purpose: Verify individual UI components interact correctly with backend services and display accurate data

   Key testing areas:

   a. Forms and data binding:
      i. Sign in / sign up
      ii. Ingredient/recipe creation forms
      iii. Settings updates
   b. Dashboard displays:
      i. Correct prep tasks, inventory levels, and assignments for each user
   c. UI state changes:
      i. UI is responsive to new tasks, locations, and events
   d. Role-based access visibility:
      i. EmployeeView, ManagerView, AdminView show/hide content according to user's role

Tools: xUnit - Component logic, Playwright - verifying rendered info matches backend state

3. **Integration Testing**

   Purpose: Confirm that services, database, and entity relationships work together to support end-to-end workflows.

   Key testing areas

   a. Authentication -> Role-based access -> Page routing
      i. Ensures authenticated users cannot access restricted views
   b. Locations -> Users -> Task assignment
      i. Validates that managers can assign prep tasks correctly to employees of specific locations
   c. Recipes -> Ingredients -> Tasks
      i. Verifies that recipe creation and ingredient association generate correct tasks
   d. ManagerView -> Tasks assigned -> EmployeeView
      i. Verifies that managers can assign tasks and that those tasks show up in the employee view

   Tools: xUnit + EF Core InMemory - test service-to-database interactions,
   Playwright - verify integration of service results with actual UI presentation

4. **Load Testing**

   Purpose: Gauge the effectiveness of system infrastructure as traffic flows in and out of the application.

   Key testing areas

   a. Scalability to accommodate dynamic traffic
   b. Resilience to high-traffic

5. **End-to-End Testing**

   Purpose: Validate that the system meets functional requirements from the user's perspective, including cross-component workflows

   Key end-to-end scenarios

   a. Employee login -> redirected to correct dashboard
   b. Manager creates location, recipe, or ingredient -> immediately visible in relevant UI pages
   c. Employee marks task completed -> reflected in Manager dashboard and reporting
   d. Unauthorized user attempts restricted actions -> access denied

e. Prep list and inventory updates propagate correctly across pages

Tools: Playwright - Simulate real users, check workflows, and verify UI and backend interactions

### 5.5 Regression Testing

To ensure new additions to the project do not break old functionality, the team will rely heavily on automated tests integrated into the CI/CD pipeline. By running through the full set of tests on every commit, the team will ensure that any new additions do not compromise existing functionality.

More specifically, xUnit, EF Core InMemory, and Playwright will be integrated into the pipeline to validate backend logic, ensuring data consistency, and end-to-end UI testing. This approach allows for early detection as breakages are caught before deployment into production. Additionally, as features grow, the testing set will expand, but automation will keep them manageable.

### 5.6 Acceptance Testing

Acceptance testing for Culinary Command focuses on validating that the system satisfies all functional and non-functional requirements defined in section 2 and meets the expectations of the client, restaurant managers and staff. The goal of acceptance testing is to show that the application performs reliably in a real restaurant workflow.

Our approach to acceptance testing will be performed near the end of each major sprint, once the features implemented during that sprint are acceptable. The team will create an acceptance test checklist derived directly from the functional requirements, including:

- Creating and assigning tasks
- Viewing and completing prep-list items
- Updating inventory levels
- Viewing role-based dashboards
- Logging in/out using RBAC
- Switching between restaurants (multi-tenant support)
- Receiving real-time updates (SignalR functionality)

Non-functional requirements—such as usability, load expectations, navigation clarity, and mobile performance—will also be validated through client walkthroughs and observational testing.

To ensure the system meets real operational needs, the acceptance testing will involve a real world client (Paul Durr) and, if available, staff members from Prairie Canary or Hometown Heroes. Client involvement includes:

- Hands-on sessions where the client performs common workflows
- Feedback collection meetings to approve or reject specific features

- Verification that the platform aligns with real restaurant processes (prep flow, order tracking, staff roles)

The client's approval serves as the final gate for promoting features to "accepted" status. Any rejected tests will return to the development backlog with updated acceptance criteria.

### 5.7 Security Testing

The team will perform security testing using tools such as OWASP's Zed Attack Proxy and dotnet-security-guard. Specifically, the team will conduct web application vulnerability testing to identify and mitigate potential vulnerabilities. Throughout this process, the team will use the tools to automatically scan for SQL injections, XSS, and CSRF, and dotnet-security-guard to scan for insecure coding patterns.

### 5.8 User Testing

User testing focuses on evaluating how real restaurant staff interact with Culinary Command, ensuring the system is intuitive, efficient and supportive of fast-paced kitchen environments. Since the platform will be used daily by cooks, managers, and owners, validating usability is a must.

## Plan for User Testing

The team will introduce user testing during the second semester once core frontend pages are functional. The plan includes:

## 1. Observation-Based Testing

Restaurant staff (or student testers simulating staff roles) will be asked to perform real tasks such as:

- Checking assigned prep tasks
- Marking tasks complete
- Updating ingredient counts
- Navigating between stations
- Switching restaurants
- Reporting missing ingredients

During these sessions, the team will observe:

- Time to complete common actions
- Confusion points or misclicks
- Whether labels/buttons are clear
- How the UI performs on tablets/phones

## 2. Task Scenarios

Each participant receives a short scenario-based list such as:

- "You are a line cook—find your tasks for the Grill Station"
- "Update the inventory for Tomatoes to 15 lbs"
- "As a manager, assign a new task to Prep Station"

Performance will be measured by completion time and errors.

## 3. Feedback Surveys

At the end of each session, users complete a short survey evaluating:

- Ease of navigation
- Clarity of information
- Appearance and layout
- Responsiveness on their device
- Suggestions for improvement

The Likert-scale questions and free-response feedback help the team identify recurring UI issues.

## 4. Iterative Refinement

All user testing findings will be reviewed during weekly team meetings. Issues will be categorized as:

- High priority (impacts workflow or clarity)
- Medium priority (causes inconvenience)
- Low priority (visual/design polish)

### 5.9 Results

The team has completed initial testing on core functionality using testing frameworks (xUnit, Moq, and Playwright). Using these testing frameworks, there will be results stored in testing folders and files. The team has successfully tested core functionality through manually testing and moving into testing with the frameworks mentioned above.

The team was able to create and verify new users that are signing up. The user creation and user database verification flow follows a process that verifies if a user was created successfully. The database is verified through the integration of MySQL. The user creation includes the user's email to avoid duplicate registrations. The testing has allowed the team to see and confirm that a user was created successfully and confirm that the entity framework core implementation correctly handles user data persistence, and password hashing though the ASP.NET core identity and through Cognito.

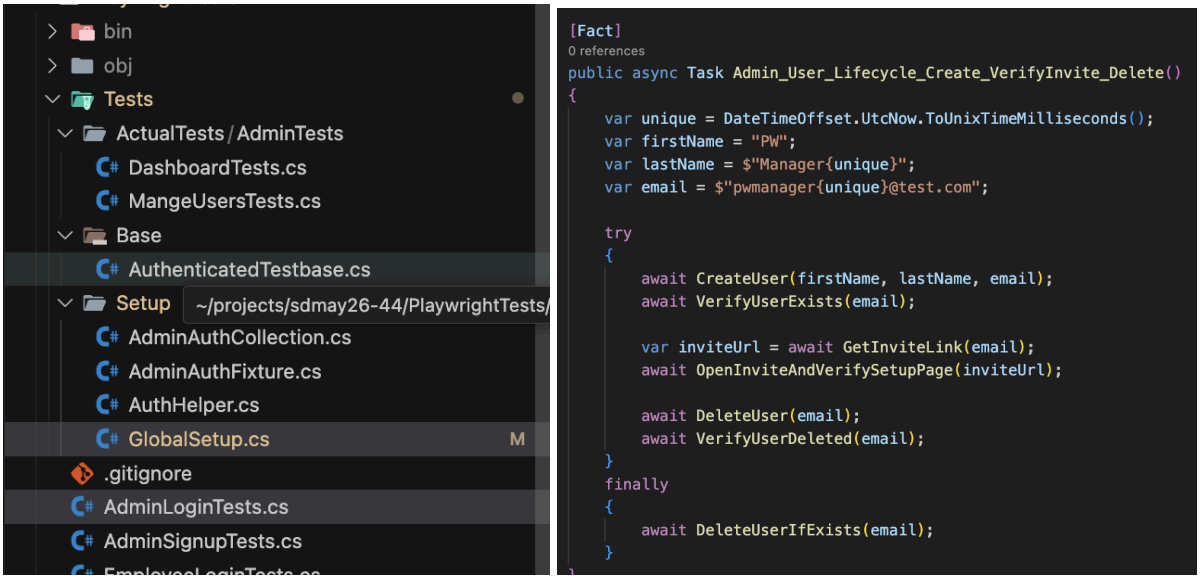



**Figure 5.1**: (left) Playwright tests over core functionality through frontend, backend and service/cognito testing. (right) An example of a Playwright test going through the User lifecycle in Culinary Command.

### Load Testing

It is critical to perform load testing on the Culinary Command application to ensure minimal downtime for the end users. Locust, an open-source load testing tool that simulates HTTP traffic was used to conduct testing. By utilizing Locust and various scripts, the team is able to gauge how effective the system is able to handle incoming traffic at various levels.

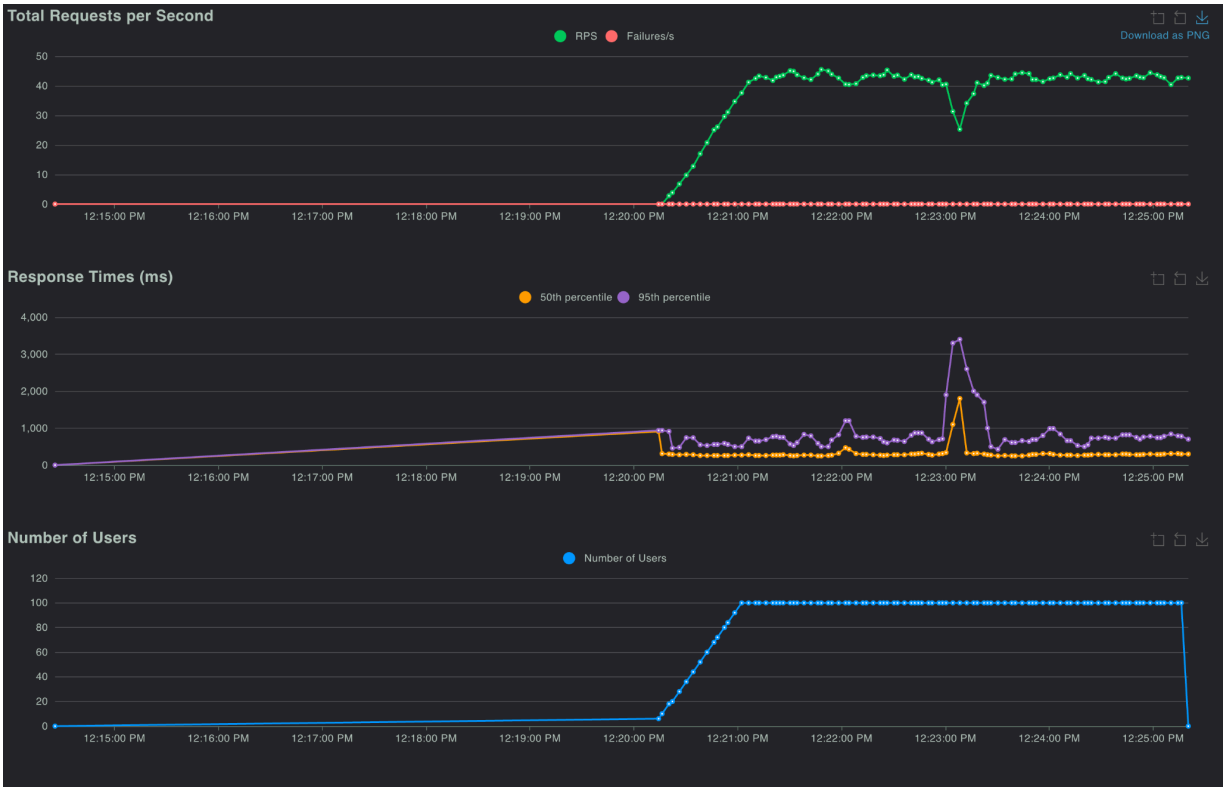



*Figure 5.2*

In the figure above (*Figure A*), there are three key metrics that Locust provides. The graph on the top visualizes how many requests are being sent to the Culinary Command web-app over time. In the middle, the average (50th percentile) and the worst case (95th percentile) response times can be seen over time. Then, the graph on the bottom visualizes the number of users interacting with the app.

From the results the team received, the team was able to extract valuable insights. More specifically, the team was able to set a baseline of the infrastructure needed to accommodate their end users. The team deducted that the Culinary Command application will be able to handle traffic with ease. As more features gets implemented and more data gets added, the team will have to revisit the load testing process to ensure that the end users will not be affected.

Moving forward the team will continue to test and follow testing strategies through the frameworks of xUnit, for unit and integration testing, Moq, for mocking a service and Playwright, for end to end user interface testing. Integration testing will test for end-to-end functionality and implementation of establishing CI/CD testing pipelines to ensure scalability to the application as new features and code are added. The team expects to see passing results and achieve improvements in the application and development.

## 6 Implementation

The Culinary Command team has made significant progress in establishing the underlying infrastructure for the application. One major milestone has been the setup of AWS resources to reliably host the team's Blazor Server app, ensuring scalability and maintainability. A high level diagram of the architecture can be seen in *Figure 6.1*. This foundation serves as the backbone for all subsequent development.

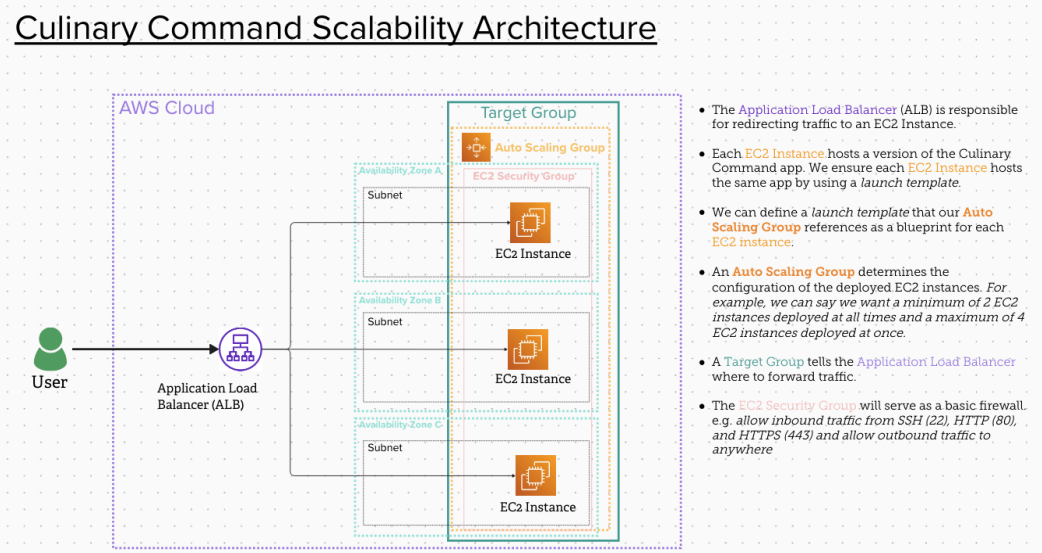



*Figure 6.1*

There has also been major progress on the inventory management system foundation. At this stage, the system supports full CRUD operations for ingredients. This allows users to add new items, view existing inventory, and remove entries as needed. Refer to *Figure 6.2* below for an example. This functionality represents the first operational layer of the team's broader design.

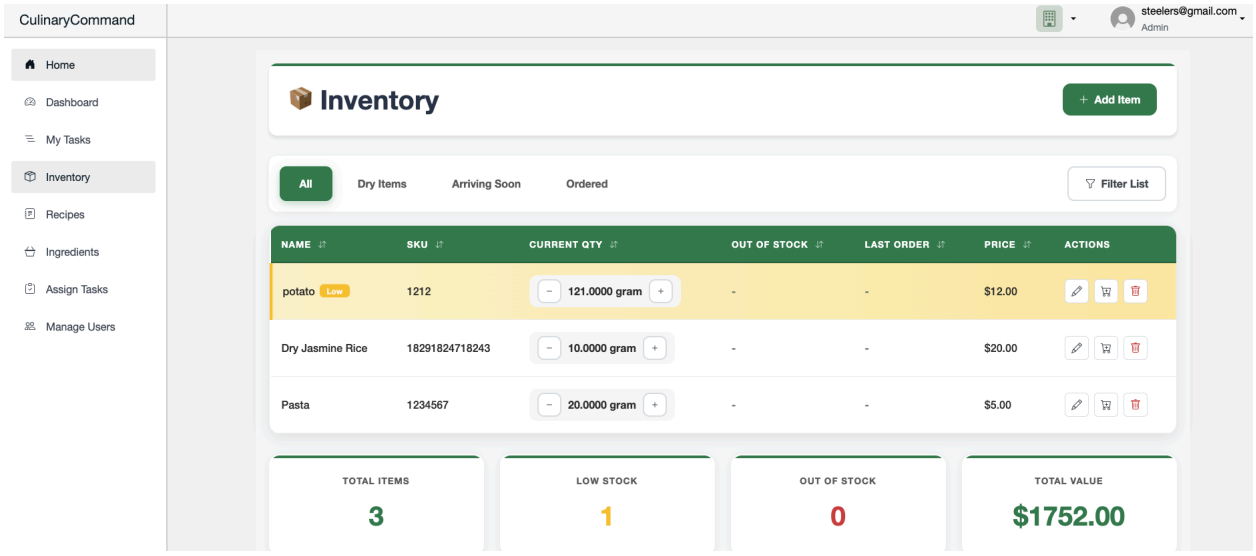



*Figure 6.2*

In parallel, the team has made user interface and experience development. On the team's deployed Blazor Server app, a user can create an admin user and sign in. Once signed in as an admin user, a respective owner can customize their restaurant that is catered to their needs. The frontend team spent much effort on a consistent visual theme to improve navigability. These milestones complement the backend infrastructure and inventory foundation, allowing for future expansion.

The work that has been completed up to this point has laid a strong foundation for upcoming features. Users can create an admin account sign in and if they wanted, they could create their own restaurant and manage ingredients. This functionality is a good starting point, but it still needs greater practicality to support real-world use cases. With the infrastructure and CRUD capabilities in place, the team is in a great position to implement practical features such as a fully operational inventory management system and a machine-learning pipeline to extract key insights from data.

## 7 Ethics and Professional Responsibility

Engineering ethics in the context of Culinary Command refers to designing and implementing a software system that is safe, secure, fair, and aligned with professional standards. Because the system handles sensitive business data, supports fast-paced work environments, and integrates AI-driven analytics, ethical responsibility is essential to ensuring trust, reliability and long-term success.

The Team follows the IEEE Code of Ethics because it aligns most closely with engineering practices involving safety, security and professional conduct. The table below incorporates the seven areas of professional responsibility defined by McCormack and maps them to relevant IEEE Code Items, along with how the Culinary Command team has acted in accordance with each responsibility.

| Area of Responsibility | Our Definition | Relevant IEEE Code Items | How the Team Has Addressed This Area |
|---|---|---|---|
| Work Competence | Performing work with highest quality, integrity, and professional skill. | IEEE 1: Accept responsibility in engineering decisions. IEEE: Improve technical competence. | We assign technologies to team members specializing in those areas and conduct research to gain competence before implementation. |
| Financial Responsibility | Delivering value, responsibly and using resources efficiently. | IEEE 4: Avoid injuring others, their property, reputation, or employment. IEEE 10: Assist colleagues and support ethical practice. | We design with AWS cost awareness, avoid unnecessary API calls due to MarginEdge rate limits, and make architectural choices that reduce long-term costs for the client. |
| Communication | Reporting work | IEEE 3: Be honest and | We provide accurate |

| | | | |
|---|---|---|---|
| Honestly | truthfully and describing designs accurately. | realistic in stating claims or estimates.<br>IEEE 7: Seek, accept, and give constructive criticism. | progress updates to the client, clearly state limitations, and avoid exaggerating system capabilities. |
| Health, Safety, and Well-Being | Minimizing risks to users, data, and restaurant operations. | IEEE 9: Avoid injuring others through negligent practices. | We use hashed passwords, encrypted AWS RDS connections, secure handling of API keys, and strictly limit who can access certain data. |
| Property Ownership | Respecting intellectual property and tools. | IEEE 8: Treat all people and respect privacy.<br>IEEE: Accept responsibility for decisions. | We use objective decision making when evaluating tools or features. |
| Sustainability | Designing a long term system that is maintainable, resourceful, and efficient. | Contribute to society and know and respect existing systems. | We selected scalable services, maintained clear documentation and used modular design for long term sustainability. |
| Social Responsibility | Producing a system that benefits society, users, and communities. | IEEE 1: Accept responsibility to make decisions consistent with public safety and welfare.<br>IEEE 5: Improve understanding of technology. | Our team contract emphasizes inclusive communication, constructive feedback, and supporting each member's contributions. |

Team Reflection:
Performing Well:

We consistently prioritize secure design choices, such as enforcing role-based access, encrypting data in transit, hashing passwords, and structuring the database to isolate tenants. This aligns with ACM principles related to avoiding harm and respecting user privacy. The early focus on AWS IAM, authentication, and proper database handling demonstrates strong professional responsibility.

Needs improvement:

While the design is modular, we still need clearer long-term documentation, more automated tests, and mature CI/CD practices. To improve, we will expand documentation, ensure architectural decisions are well recorded, and build out more robust automated testing.

## 7.2 Four Principles

| Context Area | Beneficence | Nonmaleficence | Autonomy | Justice |
|---|---|---|---|---|
| Public Health, Safety, Welfare | System helps prevent food waste and improves kitchen coordination | Avoids harm by reducing miscommunication that could lead to unsafe food practices. | Users control their tasks and workflow visibility. | The system treats all staff roles fairly in access and responsibilities. |
| Global/Cultural/Social | Promotes inclusive workflows and accessible UI. | Avoids discriminatory design patterns | Allows customization for diverse staff needs. | Ensures features work equally across locations with different cultures. |
| Environmental | Reduces waste through digital task lists. | Minimizes energy usage by efficient cloud resources. | Allows restaurants to choose how much history/data to store. | Access to analytics is equal regardless of restaurant size. |
| Economic | Increases operational efficiency and reduces costs. | Avoid causing financial harm via data breaches or errors. | Users choose when to automate workflows or adjust processes. | Provides fair access to insights, not favoring large restaurants. |

### Reflection on Context-Priciple Pairs

**Strong Pair:** *Public Health* and *Beneficence*

This project helps kitchens work safely by ensuring tasks are completed correctly, ingredients are tracked, and food handling issues are minimized. Culinary Command ensures it through clear UI, real-time updates, and a structured workflow system.

**Week Pair:** *Environmental* and *Nonmaleficence*

Although the app reduces paper waste, cloud services inherently use energy and creates a carbon impact. To mitigate this, we rely on AWS automatic scaling, efficient queries, and minimize resources usage. Future improvements might include offset planning or sustainability metrics.

## 7.3 Virtues

Three virtues important to the CulinaryCommand team:

1. **Communication**
    a. Great teams talk early and often, and with honesty

    b. The CulinaryCommand team has had great communication, with multiple avenues for discussion.
        i. The Discord server is used for more long-winded announcements and discussions about the development process
            1. The server contains multiple channels for development updates, planning proposals, and meeting minutes.
        ii. The iMessage group chat has been used for less formal updates or quick questions to the team.

2. **Ownership**
    a. Great teams treat the whole project as “their project”
    b. All members of the team have exemplified ownership in their contribution to the project.
        i. Team members consistently follow through on deadlines, stuck to clean coding principles, and fixed and notified the team of bugs.

3. **Curiosity**
    a. Great teams don’t stagnate, they learn and improve
    b. The team has demonstrated great curiosity, with several team members learning Blazor and .NET from scratch this semester. The integration with AWS is also a domain several team members had not touched previously.

Each team member also has one key virtue they exemplified this year, as well as one virtue they struggled to demonstrate. Those are listed below.

**Anthony Phan:**

- Virtue demonstrated: **Creativity**
    - Creativity matters to me because it lets me bring something new to the project instead of taking templates and applying them to the app. I like thinking from the client’s perspective and figuring out how our application can look and feel cleaner, simpler, and more intuitive. Throughout the semester, I have sketched and mocked up different UI ideas and elements, asked for feedback from my teammates, and adjusted those changes based on what the feedback to make sense for both the team and the client
- Virtue not demonstrated: **Cleanliness**
    - Clean, well-organized code is important because it saves everyone time and confusion. Right now, I know I lack this virtue sometimes. There have been times where my commits weren’t as clear as they should be, or my code wasn't commented well enough for my teammates to immediately understand what I have changed. I have slowly gotten into the habit of adding comments while coding to not just my team but myself also.

**Kevin Tran**

- Virtue demonstrated: **Curiosity**
    - Curiosity was a driving force behind my growth throughout this project. It has pushed me to learn .NET and C#, despite having no prior experience. I used my existing AWS knowledge to host infrastructure for a Blazor server, which allowed me to connect familiar cloud concepts with new backend delivery methods. That helped me bridge the gap between my prior knowledge and the unfamiliar aspects of .NET. This is important to me

because it reminds me that growth comes from curiosity. By continuing to ask questions and exploring new challenges, it will help me develop my skills.

- Virtue not demonstrated: **Communication**
    - While I believe I did demonstrate communication during this project, I recognize that it is a virtue I could have improved upon. For example, when implementing new functionality to the project, I didn't clearly convey the changes I was making. This left room for confusion and slowed down development at times. Next semester, I hope to improve by creating documentation with my implementations so that my teammates can easily understand the impact of my contributions. This would strengthen my communication and speed up team development.

**Matayas Durr:**

- Virtue demonstrated: **Assertiveness**
    - Assertiveness has been an important virtue to me throughout this project because sharing ideas confidently helps move the design forward and makes sure that important considerations don't get overlooked. I feel like I demonstrated assertiveness by contributing strong ideas during meetings, proposing UI features, and helping shape how the team approached design decisions. Speaking up and offering clear direction allowed the team to evaluate more options which helped refine the project's vision.
- Virtue not demonstrated: **Diligence**
    - Diligence is important because consistent effort and timely competition of tasks help the entire team maintain momentum and avoid delays. I noticed that I did not always demonstrate this virtue as strongly as I wanted to and there were times I struggled to complete tasks as quickly as I should have. To improve this, I plan to break tasks into smaller steps, and communicate earlier if I need help. Improving my diligence will help me stay accountable next semester.

**Wyatt Hunter:**

- Virtue demonstrated: **Industry**
    - This virtue is important because consistent effort, determination, and finishing are what move complex systems from design to implementation.
    - I consistently put substantial effort on both the backend and frontend architecture. Taking on difficult tasks, helping others get unblocked when needed, and working ahead of deadlines.
- Virtue not demonstrated: **Order**
    - Order ensures that the team remains aligned, documentation stays consistent, and development avoids chaos as the codebase grows.
    - I can improve by maintaining more structured documentation, organizing backend folders, and establishing clearer coding standards for code organization.

**Ryan Rockey:**

- Virtue demonstrated: **Communication**
    - This virtue is important because good communication keeps everyone in the loop, avoids pointless work, and keeps the team aligned.
    - I've demonstrated communication by discussing openly with the team about progress, issues, and decisions.
- Virtue not demonstrated: **Consistency**

- Consistency is important because it leads to steadier progress, reduces stress, and makes it easier for teammates to plan around my development.
- I've found myself working in "sprints" throughout the week, where I contribute a lot of code once every few days instead of contributing small amounts daily. While the work gets done, it leads to more headaches and uneven progress. I would like to work harder next semester on making development a daily habit, even when classes or other projects get in the way.

# 8 Closing Material

## 8.1 Conclusion

Over the course of the semester, the Culinary Command team has made significant progress building towards a fully functional version of the application. The team has implemented core foundations, including role-based access control, location and user management, and inventory tracking. We established cloud infrastructure through AWS and followed structured engineering practices that will strengthen long-term maintainability and security. These achievements put the team in a strong position to expand and improve the application.

Our goal remains the same: to deliver a scalable and user-friendly platform that streamlines back of the house restaurant operations, improves communication among staff, reduces waste, and provides meaningful insights to owners and managers. Based on the progress so far, the best plan moving forward is to continue developing in iterative phases, finishing core backend logic, completing frontend integration, expanding automated testing, and refining UI/UX based on user feedback. This approach will ensure that the system remains stable while new features are introduced.

While the team made significant progress, there were several constraints that affected our ability to reach all of our goals this semester. The learning curve for some unfamiliar technologies like Blazor Server, AWS, and Entity Framework Core required extra development time. Integrating external systems such as the MarginEdge API presented additional challenges related to rate limiting and data synchronization. Coordinating backend–frontend alignment across a newly built architecture also required more planning and debugging than originally planned. These constraints slowed development but ultimately strengthened the strength of the final architecture.

In a future iteration, the team could benefit from spending more time on early architectural planning, and using a more granular task breakdown to reduce bottlenecks. Despite any challenges, the progress we made this semester shows that the project is on track and is in a great position for a successful completion following term.

## 8.2 References

[1] Toast, Inc., "About Us – Toast POS," 2025. [Online]. Available: https://pos.toasttab.com/about

[2] 7shifts Inc., “All About 7shifts,” 2025. [Online]. Available: https://www.7shifts.com/about/

[3] Operandio Pty Ltd., “About Us – Operandio,” 2025. [Online]. Available: https://operandio.com/why-operandio/about-us/

[4] International Organization for Standardization, “ISO/IEC 27001: Information security management systems,” ISO, 2022.

[5] International Organization for Standardization, “ISO/IEC 12207: Software life cycle processes,” ISO, 2017.

[6] International Organization for Standardization, “ISO/IEC 42001: Artificial intelligence management system,” ISO, 2023.

[7] Microsoft, “Blazor documentation,” Microsoft Docs, 2024. [Online]. Available: https://learn.microsoft.com/aspnet/core/blazor

[8] Amazon Web Services, “AWS Architecture and Security Best Practices,” AWS Documentation, 2024. [Online]. Available: https://docs.aws.amazon.com

# 9 Team

## 9.1 Team Members

1) Matayas Durr
2) Wyatt Hunter
3) Ryan Rockey
4) Kevin Tran
5) Anthony Phan

## 9.2 Required Skill Sets for Your Project

FRONTEND DEVELOPMENT – Building user interface for users, and pages for access with different permissions.
BACKEND DEVELOPMENT – Build logic, architecture, and API calls and integration for restaurant data.
AL/ML DEVELOPMENT – Train models, quick analysis, and logic for recommendations through outputs.
PROJECT MANAGEMENT – Planning milestones, ensure positive progress, maintaining documentation, and managing communications with the client.
SYSTEM INTEGRATION AND ARCHITECTURE – Ensuring frontend, backend, AI/ML models, and API work together.

## 9.3 Skill Sets covered by the Team

BACKEND DEVELOPMENT

1. Wyatt Hunter

2. Matayas Durr
3. Kevin Tran

FRONTEND DEVELOPMENT

1. Matayas Durr
2. Anthony Phan
3. Ryan Rockey

AL/ML DEVELOPMENT

1. All members of the project

PROJECT MANAGEMENT

1. Ryan Rockey
2. Anthony Phan

SYSTEM INTEGRATION AND ARCHITECTURE

1. Kevin Tran
2. Matayas Durr

### 9.4 Project Management Style Adopted by the team

Team will adopt the agile methodology/management style

### 9.5 Initial Project Management Roles

Matayas Durr: Client Interaction & Full Stack
Wyatt Hunter: Team Organization & Full Stack
Ryan Rockey: Individual Component Design & Frontend
Kevin Tran: Backend Management & Security
Anthony Phan: Full Stack

### 9.6 Team Contract

Team Members:
1) Anthony Phan2) Kevin Tran
3) Matayas Durr
4) Ryan Rockey
5) Wyatt Hunter

**Team Procedures**

**1. Day, time, and location (face-to-face or virtual) for regular team meetings:**

During first semester, the team met every Wednesday virtually on Discord, along with times allocated on Sundays when necessary. Times for in-person meetings are communicated if necessary.

**2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):**

The group communicates through a Discord server and an iMessage group chat.

**3. Decision-making policy (e.g., consensus, majority vote):**

The group makes decisions through majority votes.

**4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be**

**shared/archived):**

Meeting minutes can be kept in a shared document between the 5 of us. Wyatt Hunter can be responsible for keeping track of minutes, but anyone can also.

**Participation Expectations**

**1. Expected individual attendance, punctuality, and participation at all team meetings:**

Everyone should be attending the meetings and classes unless specified differently or communicating with the team members. We don't want to miscommunicate. Overcommunicating would be the best thing we could do.

**2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:**

We expect each team member to be accountable for what they are responsible for and when in trouble, reach out before digging the hole deeper. The biggest priority is that when someone is stuck or in trouble, they must speak out for help. The faster they are helped, the faster they learn, and then the faster the application will move forward.

**3. Expected level of communication with other team members:**

Our expected level of communication is very high. Overcommunication is far better than undercommunicating. Communication will be the one thing that makes or breaks the team.

**4. Expected level of commitment to team decisions and tasks:**

Our expected level is medium to high. We want everyone to join in on team decisions so we can have as many ideas or concerns as possible. Everyone should at least say their opinion on a decision or task.

**Leadership**

**1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):**

- Matayas Durr: Client Interaction
- Wyatt Hunter: Team Organization
- Ryan Rockey: Individual Component Design
- Anthony Phan: Frontend
- Kevin Tran: Backend Management
- Each member contributes to testing

**2. Strategies for supporting and guiding the work of all team members:**

- Task sheets (Azure, Git or any other platform)
- Using Jira for tracking issues
- Keeping track of deadlines and work using Discord channels
- Making sure you get all of your own tasks done and being available to help any other team members.

**3. Strategies for recognizing the contributions of all team members:**

We will keep track of who's doing what feature on a document and how everyone is putting the code/document together.

**Collaboration and Inclusion**

**1. Describe the skills, expertise, and unique perspectives each team member brings to the Team.**

Matayas Durr: I bring strong skills in web development and database design, which will help with the technical side of the project. I also have experience in UI design and customer facing restaurant systems, which will allow me to make a web app that will be both practical and user friendly. My first hand industry knowledge with restaurant operations gives me insight into real world needs that can help with design decisions. I also bring good communication skills which will help me share ideas clearly and coordinate tasks with my teammates.

Wyatt Hunter: I bring a strong technical expertise in Blazor and the .NET Core framework, which will serve as the foundation for the team's application implementation. I have completed multiple internships working directly with real clients, having valuable experience in understanding requirements and delivering solutions that align with client expectations. My background includes working with complex, large-scale codebases containing hundreds of business rules across diverse entities and clients. As a full-stack developer for two consecutive summers, I have gained extensive experience in building Blazor applications, implementing advanced reporting features, and designing robust validation processes. This combination of technical depth, real-world client exposure, and hands-on development experience allows me to contribute both practical insights and scalable solutions to the team.

Kevin Tran: I have a strong background in Cloud Security, specifically Identity and Access Management. I have been a PTS on the Cloud Security team at John Deere since 2023. On the Cloud Security team, I work with other engineers to secure our cloud environments across the entire organization. We use a lot of automation in our processes. During my time at Deere, I have worked with large codebases and applied Agile methodologies to meet deadlines. I am familiar with full-stack development. I wouldn't say it's my strongest trait, but I can hold my own. My strongest languages are Python, Java, and JavaScript.

Anthony Phan: I am strong with frontend development using React and frontend languages from HTML, CSS and JavaScript. I have experience with frontend projects and programming which will help in designing the UI to be user friendly. I also have hands-on experience as a back of house cook. Though I am more focused on frontend development, I have knowledge on the backend and its functionalities.

Ryan Rockey: I bring strong attention to detail and skills when it comes to UI development, primarily in React and Android development using Kotlin. I hope to gain more experience in backend development, gaining a more well-rounded understanding of the full-stack development process.

**2. Strategies for encouraging and supporting contributions and ideas from all team members:**

Matayas Durr: To encourage my teammates, I will make sure everybody has a chance to share their ideas, at the same time also encourage them to be actively listening without interrupting. I will also support my teammates by asking them for input on decisions, giving constructive feedback and recognizing the work they put in. Another idea to encourage my teammates would be creating an open environment where everyone can share their ideas without them being dismissed too quickly. Overall, I think it is important to make sure everyone feels included and that their voices are heard.

Wyatt Hunter: We will encourage contributions by fostering an open environment where all ideas are welcomed and respected. Team members will be given equal opportunities to share input during discussions, and we will actively listen and build on each other's ideas. Constructive feedback will be emphasized to support growth and collaboration.

Kevin Tran: To foster a healthy working culture, I will provide honest feedback and try to ask difficult questions. I will also try to emphasize communication. As long as we are all on the same page, it will be easy to tackle any obstacles that we run into. I am always willing to answer questions and will be open to helping anyone in need.

Anthony Phan: My strategies for encouraging and supporting contributions will be inclusive with the thoughts and ideas of all my team members. I will respect my teammates' opinions and listen actively. I will contribute to sharing my ideas with others and talk about my opinions. I want to have my teammates feel included and make sure if they have something to say that it is heard.

Ryan Rockey: I will encourage and support contributions by actively listening and participating in group discussions. All team members should be encouraged to share their ideas as well as fully hear out the ideas of the other team members. An ability to receive and give constructive feedback is also important; sometimes group members may miss the mark and it's important for them to be able to take criticism to heart and apply it going forward.

**3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)**

If the team environment is keeping team members from contributing, we will bring it up respectfully either during team meetings or in a private message. Issues should be addressed directly and early so they don't turn into bigger problems. Team members will explain clearly what is making things difficult for them to get things done and suggest possible solutions. If the team cannot resolve the issue on their own, we will involve a faculty advisor or the instructor to help mediate.

**Goal-Setting, Planning, and Execution**

**1. Team goals for this semester:**

- Complete a baseline rendering or mock up sketches of the application.
- Implement logic of simple actions for users, such as login or signing up for a new account.
- Create clear and consistent documentation for team members and users
- Complete sprints and team milestones on time

**2. Strategies for planning and assigning individual and team work:**

- Use Jira for task management
- Work or features will be broken down into 2 week sprints
- Work be assigned based on team members strengths, and reassign work if need to balance the workload

**3. Strategies for keeping on task:**

- Communicate daily on tasks or problems through Discord and/or text messaging
- Weekly meetings on Discord to discuss progress, problems, and feedback

**Consequences for Not Adhering to Team Contract**

**1. How will you handle infractions of any of the obligations of this team contract?**

Talk about why it happened, and what could be done if the infraction(s) were to happen again. If a team member fails to meet the expectations in this contract, the team will first have an open discussion to understand why the infraction occurred. This discussion will focus on identifying the problems that led to the infraction and trying to find a solution for it. The team will also agree on what steps should be taken to make sure this doesn't happen again

**2. What will your team do if the infractions continue?**

If, after talking with the involved team member(s), the infractions continue, the team will meet with the advisor to discuss possible consequences and develop a plan to correct the behavior.

**********

*a) I participated in formulating the standards, roles, and procedures as stated in this contract.*
*b) I understand that I am obligated to abide by these terms and conditions.*
*c) I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.*

| | |
|---|---|
| 1) Anthony Phan | DATE: 12/7/25 |
| 2) Kevin Tran | DATE 12/7/25 |
| 3) Ryan Rockey | DATE 12/7/25 |
| 4) Wyatt Hunter | DATE 12/7/25 |
| 5) Matayas Durr | DATE12/7/25 |