sdmay26-44

CULINARY COMMAND

FACULTY PRESENTATION

THE TEAM

Matayas Durr
Software Engineering

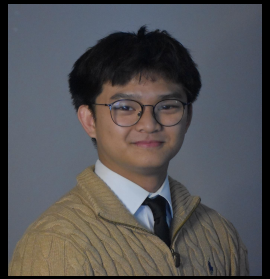
Frontend Developer & Client Liaison

Wyatt Hunter
Software Engineering

Backend & Architecture Lead

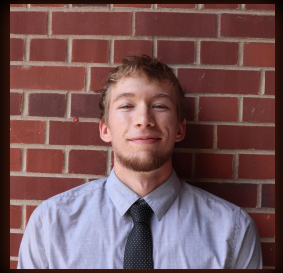
Kevin Tran
Software Engineering

Backend Lead / Security & Infrastructure



Anthony Phan
Software Engineering

Frontend & UI/UX Developer



Ryan Rockey
Software Engineering

Frontend & UI/UX Developer

The Core Problems Restaurant Teams Face

Restaurant owners often deal with chaotic paper lists and scattered spreadsheets to manage their restaurant.

Culinary Command serves as a platform that streamlines the restaurant management process to cut waste, boost efficiency, and compete head-to-head with larger chains by addressing:

Unreliable Prep Tracking

– Many restaurants still depend on paper notes, whiteboards, or memory.

– This leads to inconsistent prep amounts, last-minute scrambling, and wasted time.

Unclear Task Assignment

– Managers don't always have visibility into who owns each task.

– Work gets duplicated or missed, and communication has to happen in the middle of a rush.

Limited Operational Insight

– Prep work, inventory usage, and staff workload aren't tracked in one place.

– Owners can't easily see trends or make data-driven decisions about ordering and staffing.

Culinary Command is designed to replace these fragmented processes with a single, structured system.

OBJECTIVES

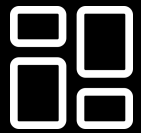
01	Ease of Use	The team wants to create a central location for restaurant owners to seamlessly manage their restaurant.
02	Prep List Management	Owners will be able to manage preparation lists/tasks for employees
03	Ingredient and Recipe Catalog	For each restaurant, their will be a catalog to view ingredients and recipes.
04	AI Integration	Utilize prehistoric data to make predictions about business conditions

REQUIREMENTS

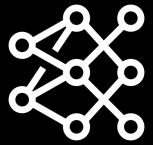
Custom recipes for individual restaurants

Inventory Management System tracking ingredients and stock

Task Management System for assigning tasks to employees



Independent dashboards for each employee



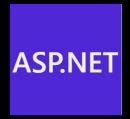
AI predictions on new stock orders and sales

Customization for individual owners/restaurants

TECH STACK

Culinary Command Tech Stack

FRONTEND & BACKEND



ASP.NET: web/API framework

Blazor: C# web UI framework

Playwright / bUnit:

End-to-end and unit testing

CLOUD INFRASTRUCTURE

Amazon Lightsail

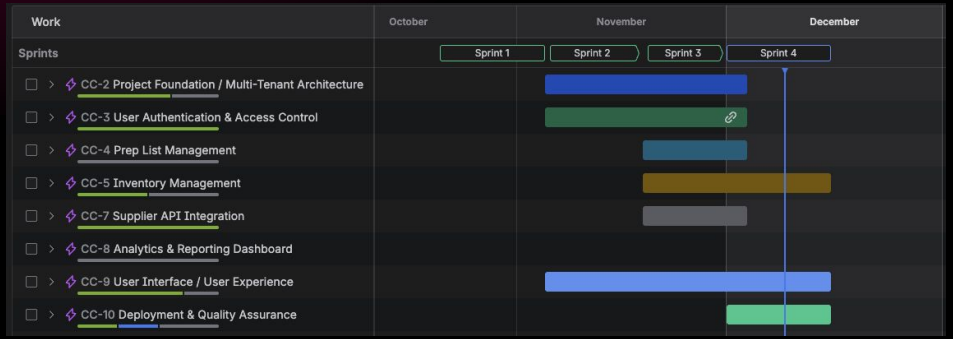
Amazon RDS MySQL

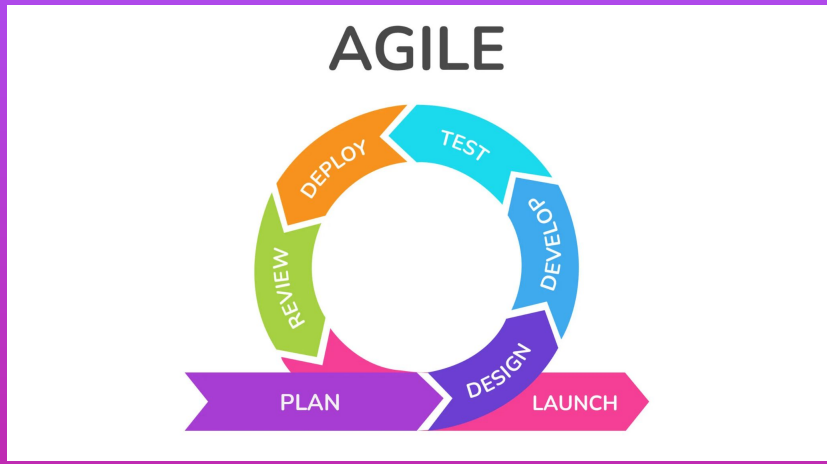
VERSION CONTROL & CI/CD

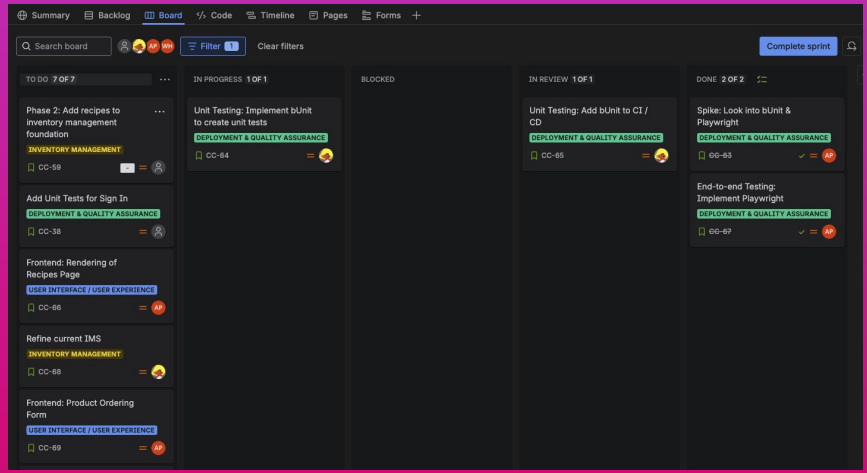
GitHub

AGILE FRAMEWORK

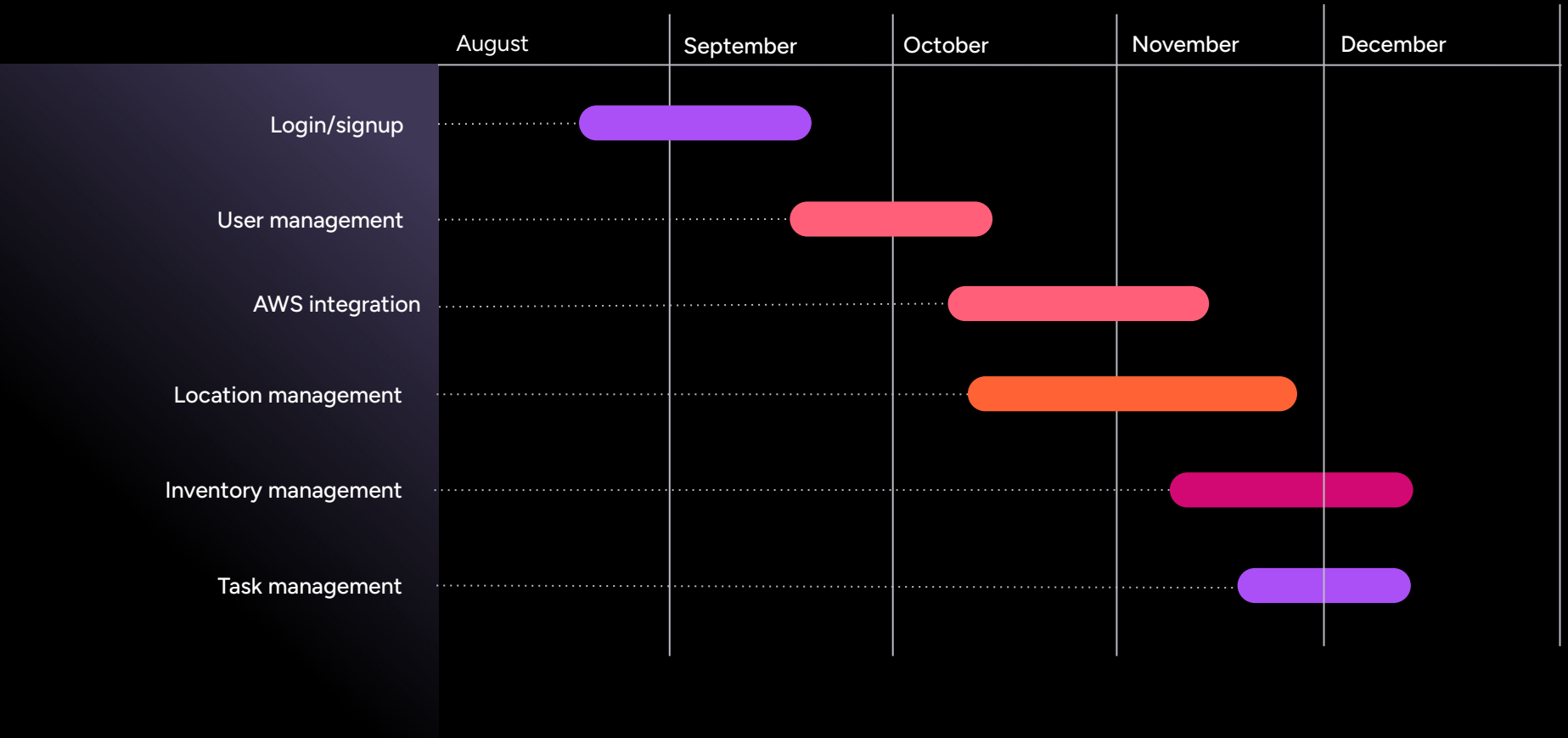
- Project management approach that emphasizes collaboration and continuous improvement to deliver value to customers.
- Weekly meetings to discuss progress, issues, and priorities.
- Two week sprints.
- Jira



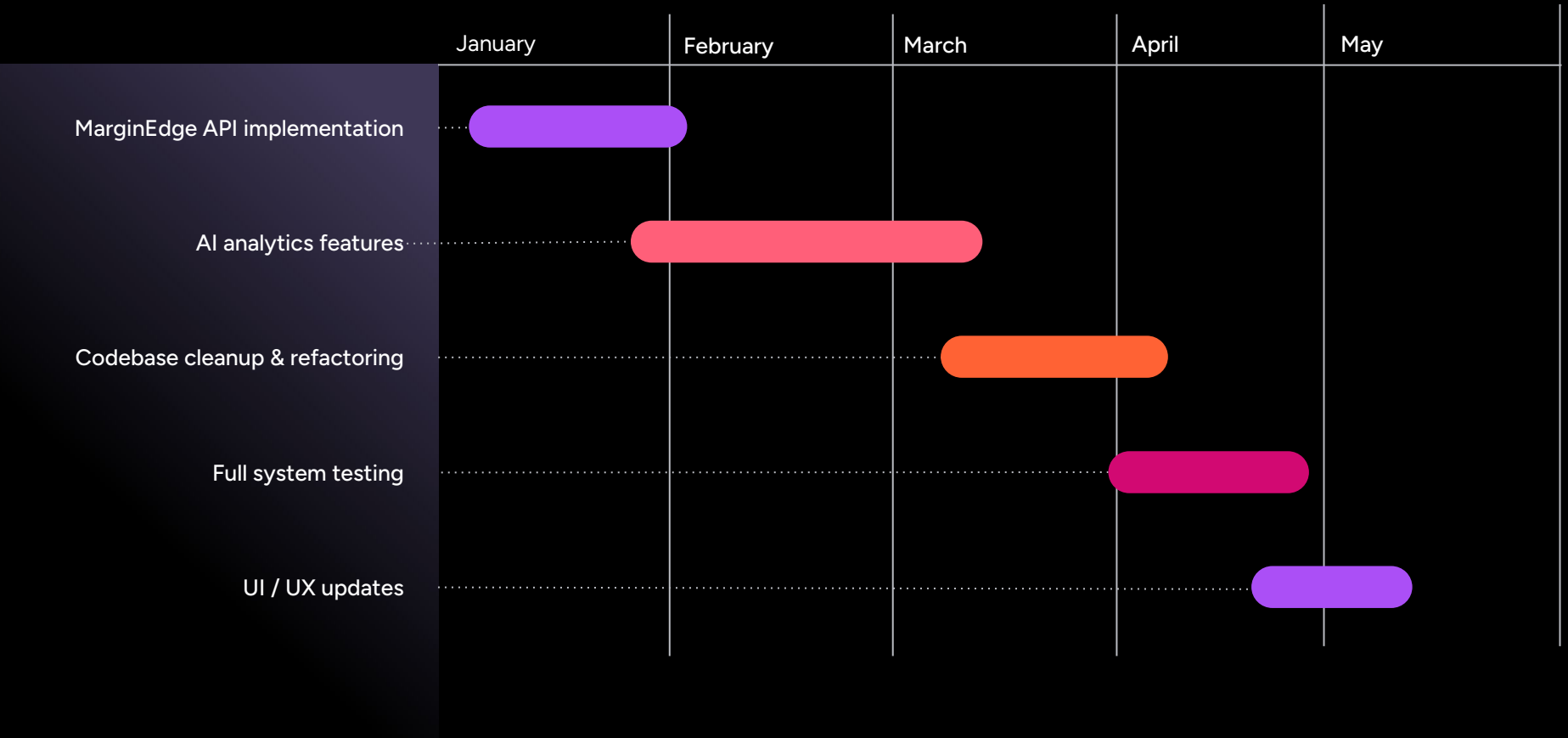




PROJECT TIMELINE (FALL SEMESTER)



PROJECT TIMELINE (SPRING SEMESTER)



PROJECT TIMELINE

We are here

01

Infrastructure
September - November

Set up underlying foundation for the project.

- GitHub Actions (CI /CD
- AWS Lightsail & RDS for hosting and database.

02

Inventory Management System
November - December

Designed mockups of UI and implemented business logic.

- Basic dashboard views
- Ingredient creation feature
- Inventory Management view

03

Additional Features
January - February

Continue implementing features, ensuring clean data.

- PurchaseOrder and Vendor logic.
- MarginEdge API Integration
- Research AI pipeline implementation

04

Data-driven AI Insights
March - April

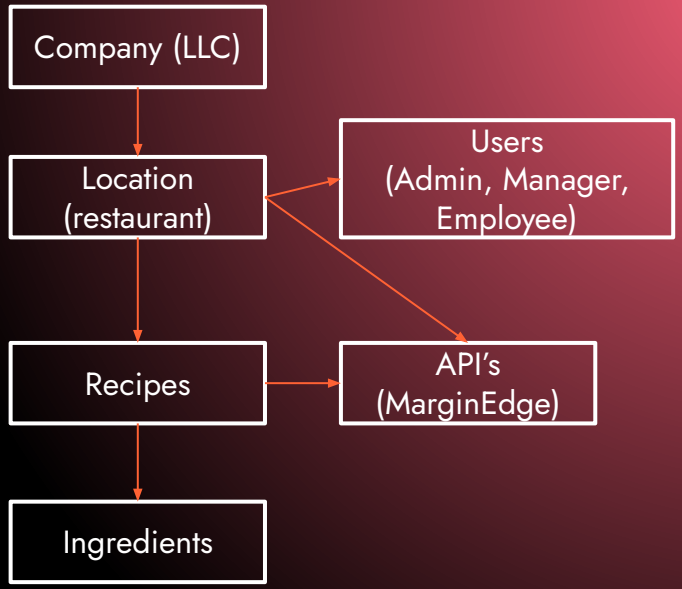
Build and maintain a machine learning engine that generates insights:

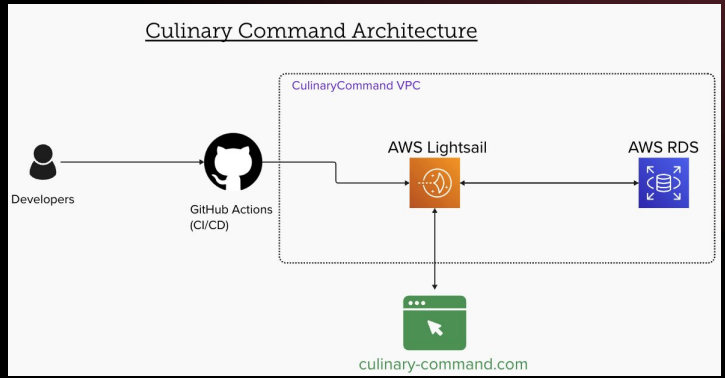
- Leverage AWS QuickSight to generate business insights
- Interactive dashboards

HIGH LEVEL ARCHITECTURE

Three Layered Structure:

- Front End - Blazor Server
  - Inter
  - Role-Aware
  - Real-Time rende
  - Uses dependency inj
    state services (Auth, Loc
- Application Back End - ASP.NET Core
  - Business Logic
    - Authentication & Auth
    - Location/Company Scoping
    - Inventory (recipe/ingredient) v
  - Service Layer (UserService, RecipeS
  - EF Core domain models mapping to th
- Data & Infrastructure - AWS
  - AWS RDS - Stores all data
  - AWS LightSail: Host Blazor Server & API
  - EF Core Migration
  - Designed for Multi-Tenant separation at C





CI/CD via GitHub Actions

GitHub Actions

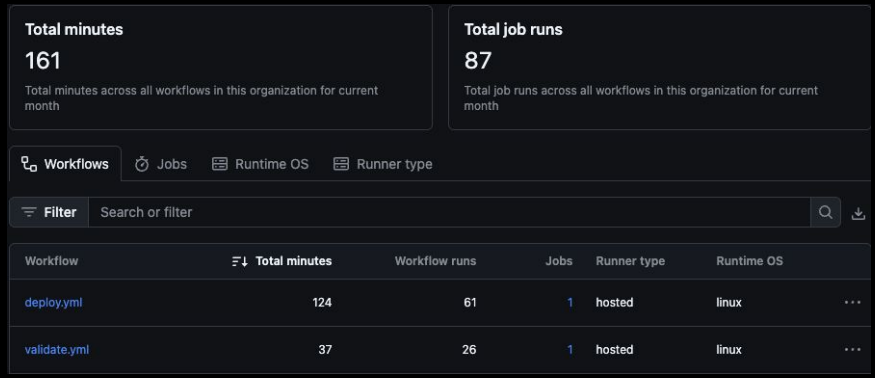
Pipeline features:

- Automatic unit tests
- Database schema migrations
- Server deployment
- AWS resource deployments using Terraform

Benefits:

- Improved code quality
- Reduced debugging time
- Increased development speed
- Consistency

Usage Metrics:



Performance Metrics:

DESIGN – FRONTEND

What we wanted:

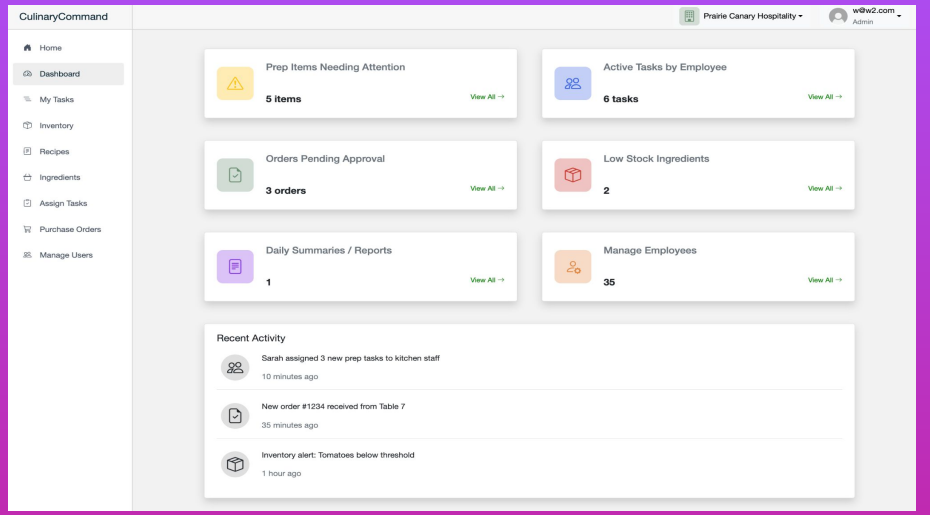
- A UI that doesn't make restaurant staff want to throw their device into the fryer.

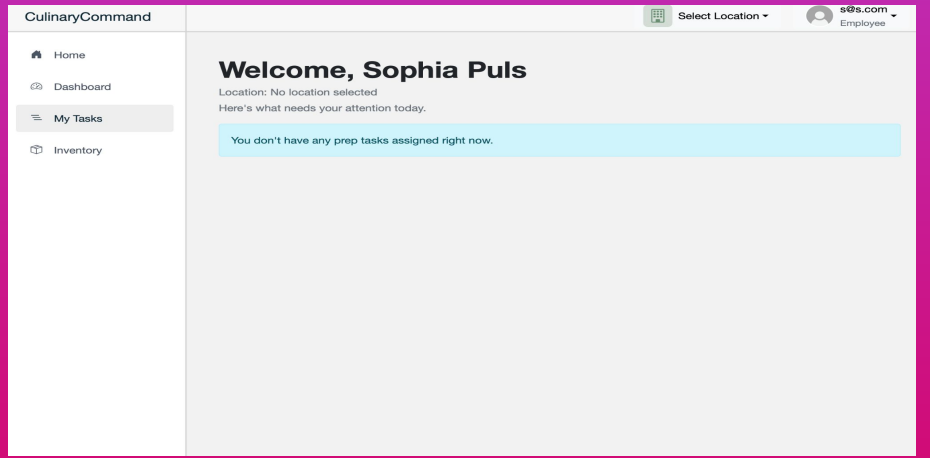
So we built it to be:

- Super Clean: nothing extra on the screen
- Touch Friendly: big buttons, big spacing and works on tablets.
- Fast to Use: every action should take about 1-2 touches on average (Admin is little different when setting things up).
- Consistent: same layout everywhere so no getting lost (unlike aws)
- Chef-Approved: easy enough for a busy line cook to use

How the UI is Organized:

- Left Menu = kitchen tool belt
- Top Bar = your restaurant/company and account settings
- Main Page = the action
- Simple Colors: Green means good, red mean check it out.





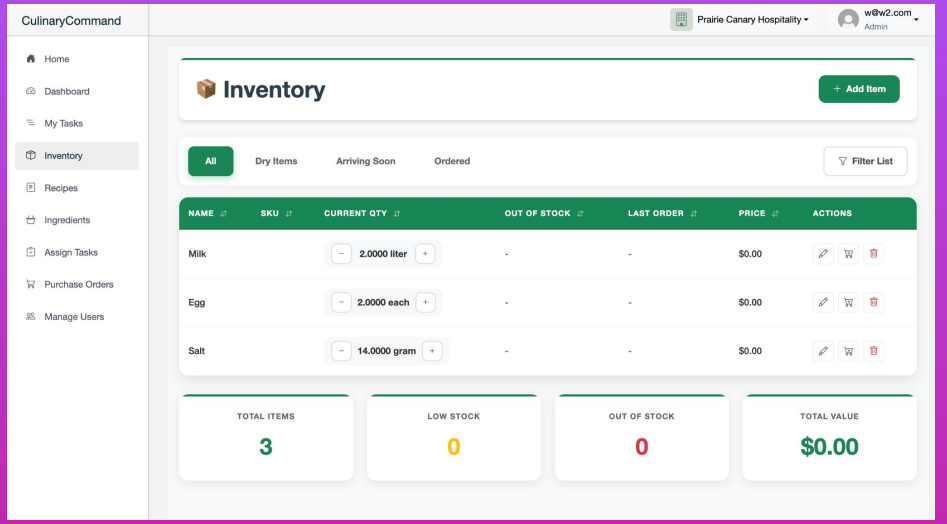
DESIGN – FRONTEND CONT.

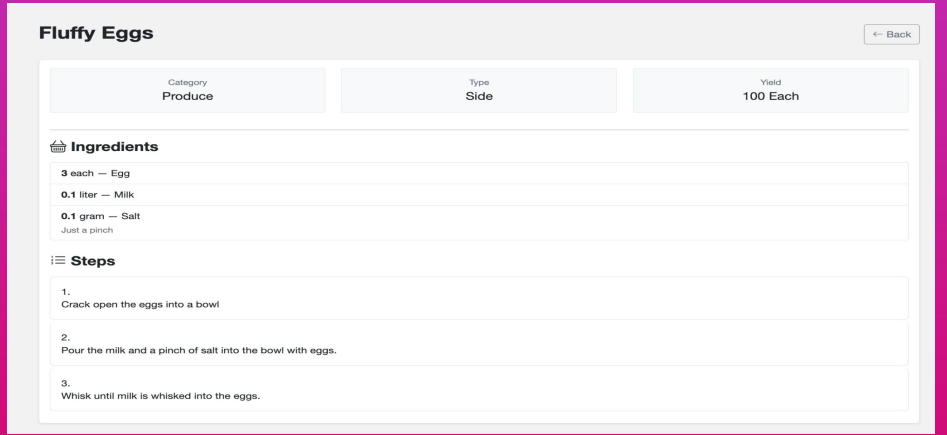
Inventory Page

- Feels like a grocery list but smart
- Tap +/- to adjust quantities
- See low, high or out of stock items, no surprises
- Mapping to recipes, ingredients and other inventory items.

Recipe Page

- Pretty, simple recipe display
- Ingredients exactly how cooks want them
- Clear automated steps so no confusion
- Great for training new staff or standardizing dishes.





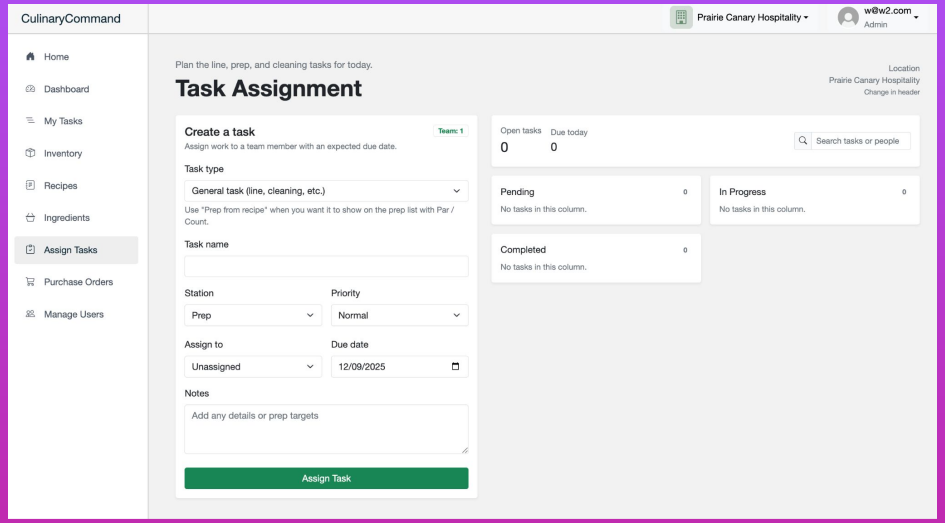
DESIGN – FRONTEND CONT.

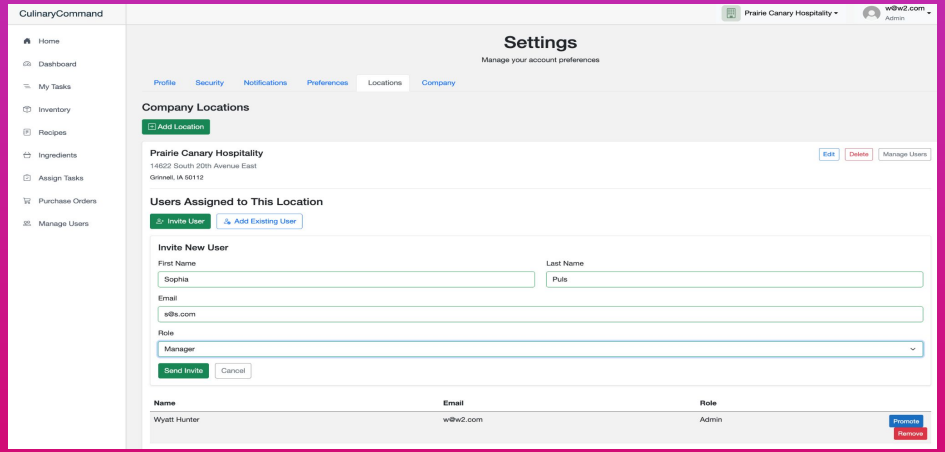
Task Assignment Page:

- Managers/Admins create tasks in seconds
- Maps to My Tasks page for assigned employee.
- Assigned to stations or specific workers
- Kanban style - “pending” , “In Progress”, and “Done”
- Real-time updates so no time is wasted

User Management:

- Add locations, employees/managers, editing
- Promote, demote, and remove with one click
- Make multi-restaurant management actually manageable and visually appealing





TESTING

Our goal is deliver a restaurant management system that is reliable, secure, and maintainable through comprehensive testing

Unit Testing

- Using bUnit, compatible with Blazor
- Test individual components in isolation
- Ensure the verification of logs and data
- Check edge cases and error handling

End-to-End Testing

- Using Playwright xUnit end-to-end testing
- Works seamlessly server side with Blazor
- Helps with role based testing, catches bugs early
- One command will run all tests and return with test results

Future of Testing

Expand testing coverage and leverage AI to predict and prevent issues before they occur

- Service Tests
- Authentication Tests
- Device Compatibility Testing

RISKS AND MITIGATIONS

User Feedback

Our pseudo-client, Paul Durr, has provided valuable insight into real restaurant workflows based on his experience operating multiple locations. However, our team has not yet conducted formal user testing or regular feedback sessions.

— Early feature discussions with Paul helped us understand pain points around prep lists, task assignment, and inventory management.

— Without consistent feedback during this phase, some assumptions about restaurant workflows may need adjustment once real users interact with the system.

— Next semester, structured user testing with Paul and his staff will help validate our designs and guide refinements to ensure the platform fits real operational needs.

Limited Technical Experience

- Our team is still learning new tools at the same time we're building the system — Blazor, PlayWright, .NET, AWS, AI, LightSail, and Margin Edge. With everyone picking up a unfamiliar tech, development can possibly slow down, mistakes can happen more easily, and certain features take longer than expected.

→ Experienced teammates actively mentor others on Blazor, AWS, and testing tools.

→ Use code reviews and automated testing via GitHub Actions to catch mistakes early.

→ Pair teammates up on features so no one learns new tools alone

→ Document pitfalls or problems in the shared communication channels

Integration Delays

When a system gets ahead of another (frontend, backend, or AWS), features can get stuck waiting. Small API change or late endpoint delivery can stall entire UI sections. These delays slow down development and increase work.

→ Lock down model structures so both teams know exactly what data shape to expect.

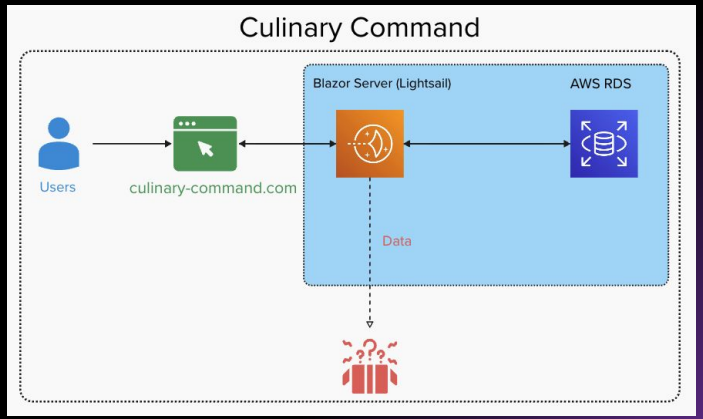
→ Mock data and a staging environments so the frontend can keep moving even if the backend isn't ready yet.

→ Weekly integration check-ins help catch breaking changes before they hit main.

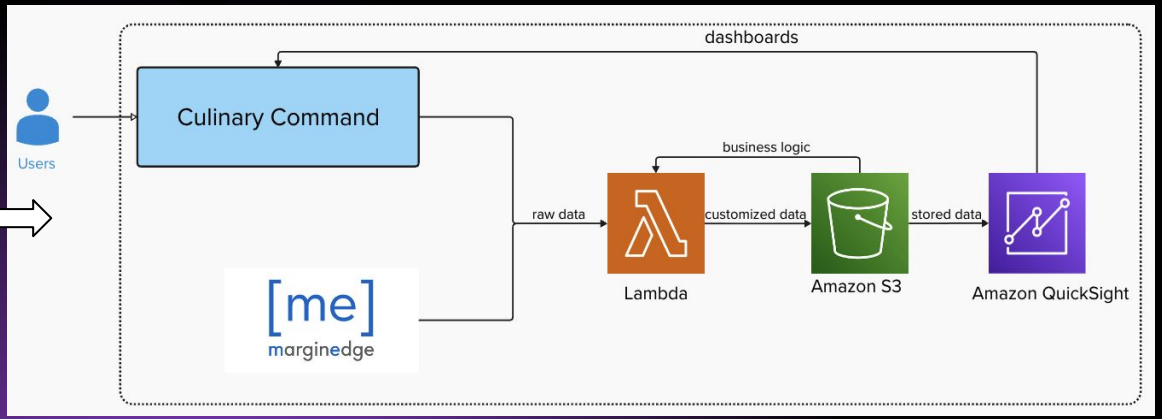
→ For example, a backend update to the Recipe model can instantly break multiple UI pages if it isn't communicated earlier.

The Journey Ahead: Current State and AI-Enabled Future

Current Culinary Command Architecture



Culinary Command + Machine Learning Engine Architecture



CONCLUSION

What We Accomplished

Foundational Progress

· Built core infrastructure on AWS (Lightsail + RDS)

· Implemented authentication and multi-location support

· Developed inventory management with full CRUD

· Established consistent UI/UX foundations across pages

Where We're Going Next

Next Semester Focus

· Complete task management and assignment workflows

· Integrate MarginEdge API for real restaurant data

· Expand automated testing (unit, integration, E2E)

· Refine UI for managers, staff, and multi-location workflows

Long-Term Vision

Final Deliverable

· Deliver a polished, production-ready platform

· Provide data-driven insights using AWS QuickSight

· Support real restaurant pilot testing

· Empower owners and staff to streamline operations

Q & A