PROJECT DESIGN

Cloud Cost Intelligence Platform

Michael, Ishan, Sean, Bryana, Tony

CMSC 495 Computer Science Capstone

Instructor: Lynda Metallo

January 27, 2026

Table of Contents

1. User Interface and Functionality 3
2. Application Structure 3
3. UML Diagrams 4
4. Project Scope 9
5. Requirements 10
6. Methodology 11
7. Schedule and Milestones 13
8. Tasks and Resource Assignments 14
9. Risk Assessment 16
10. Evaluation Plan 17
11. Testing Strategy 19
TEAM APPROVAL 20

1. User Interface and Functionality

The Code Collective has created an easy-to-use web dashboard for the Cloud Cost Intelligence Platform. With this tool, users can track, review, and manage their multi-cloud spending in one place. The dashboard works on Chrome, Firefox, and Edge browsers across Windows, macOS, and Linux.

Core functionality includes:

- FR-01: Unified cost dashboard displaying AWS, Google Cloud, and Azure spending
- FR-02: Spending trend visualization with daily/weekly/monthly charts
- FR-03: Provider and service filtering capabilities
- FR-04: Waste alerts for unused or underutilized resources
- FR-05: Rightsizing recommendations with potential savings
- FR-06: Export functionality for CSV and PDF reports
- FR-07: Budget threshold configuration with alert notifications
- FR-08: Resource usage metrics (global average or max by timespan)

2. Application Structure

The Cloud Cost Intelligence Platform is built with a three-tier architecture. It has separate layers for the presentation (frontend), business logic (backend), and data (database).

2.1 Presentation Layer (Frontend)

The user interface uses HTML, CSS, and JavaScript. The frontend displays the dashboard, charts, and tables, and handles user interactions. It connects to the backend using REST API calls.

2.2 Business Logic Layer (Backend)

The backend is built in Python and provides RESTful API endpoints for the frontend. It processes data, calculates costs, runs waste detection algorithms, and generates recommendations. The backend retrieves data from the database and prepares responses for the frontend.

2.3 Data Layer (Database)

Azure SQL Server will store simulated cloud cost data, resource details, budget settings, and alert thresholds. The database is set up to support queries for cost aggregation, trend analysis, and usage metrics.

3. UML Diagrams

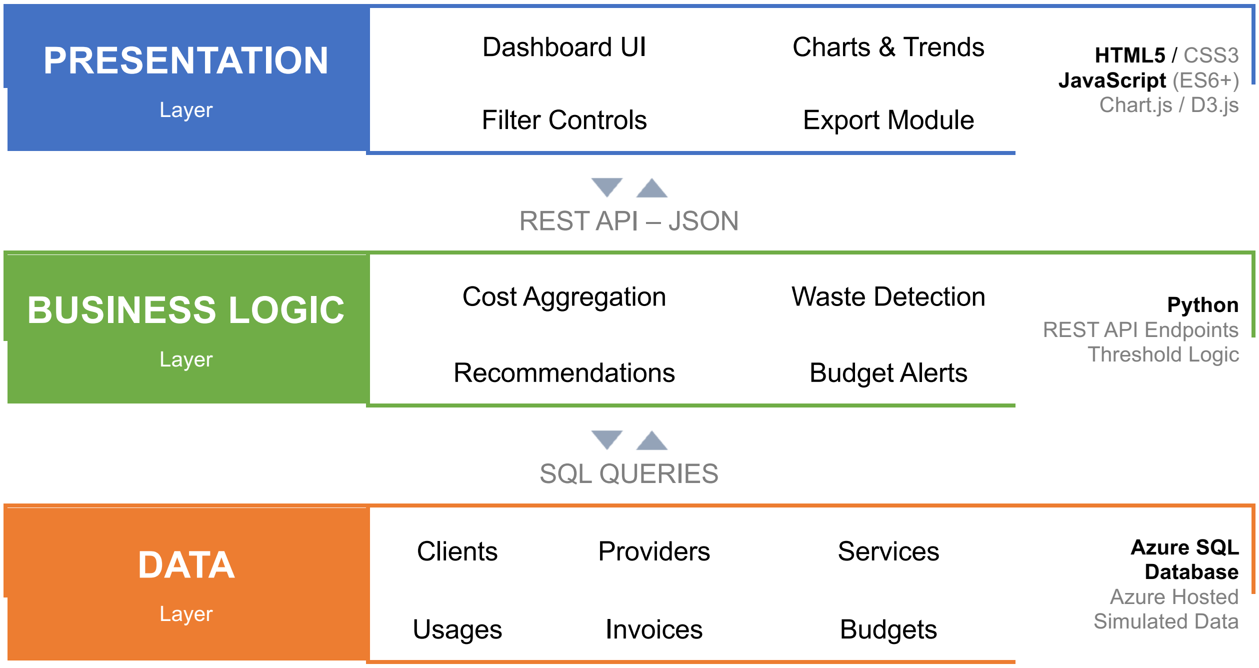
Below are UML diagrams representing different components of the application:

3.1 System Architecture

Three-tier architecture showing data flow between components:

System Architecture

Cloud Cost Intelligence Platform – Three-Tier Architecture



3.2 Database Objects

Entity tables and their relationships:

Cloud Cost Database
Entity Relationship Diagram

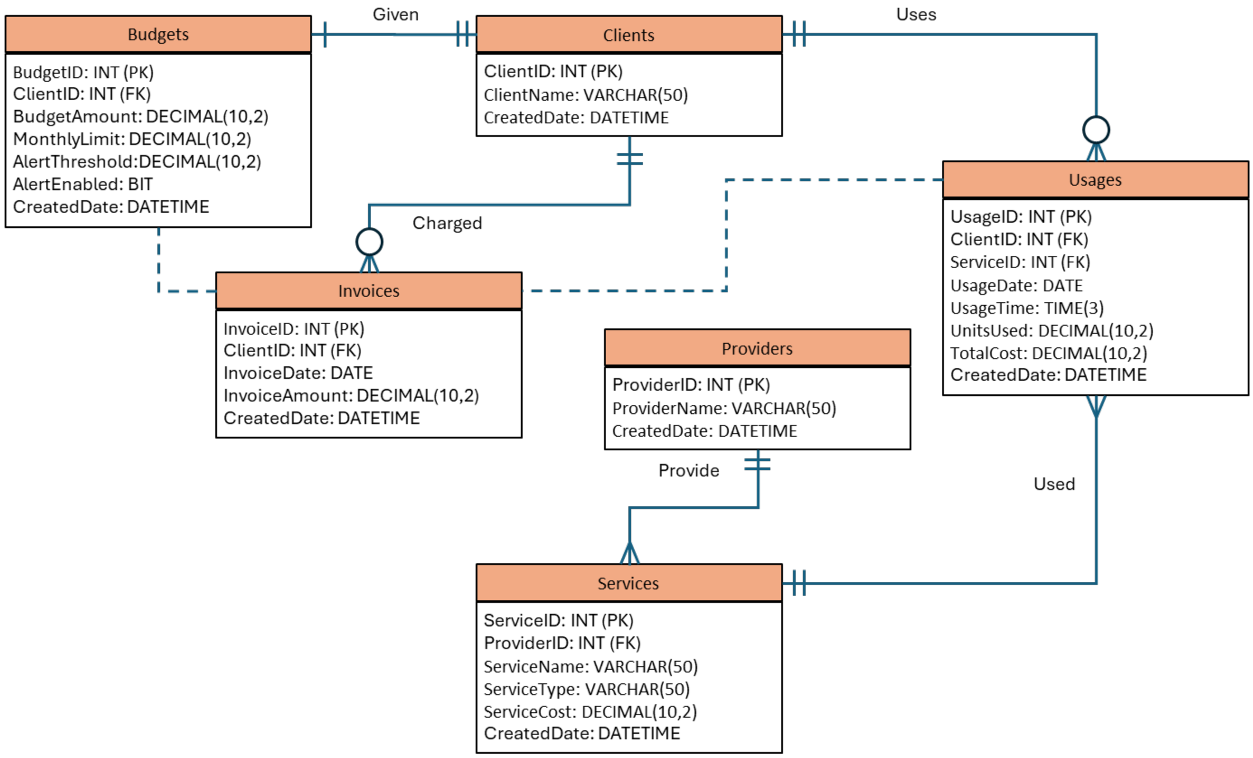
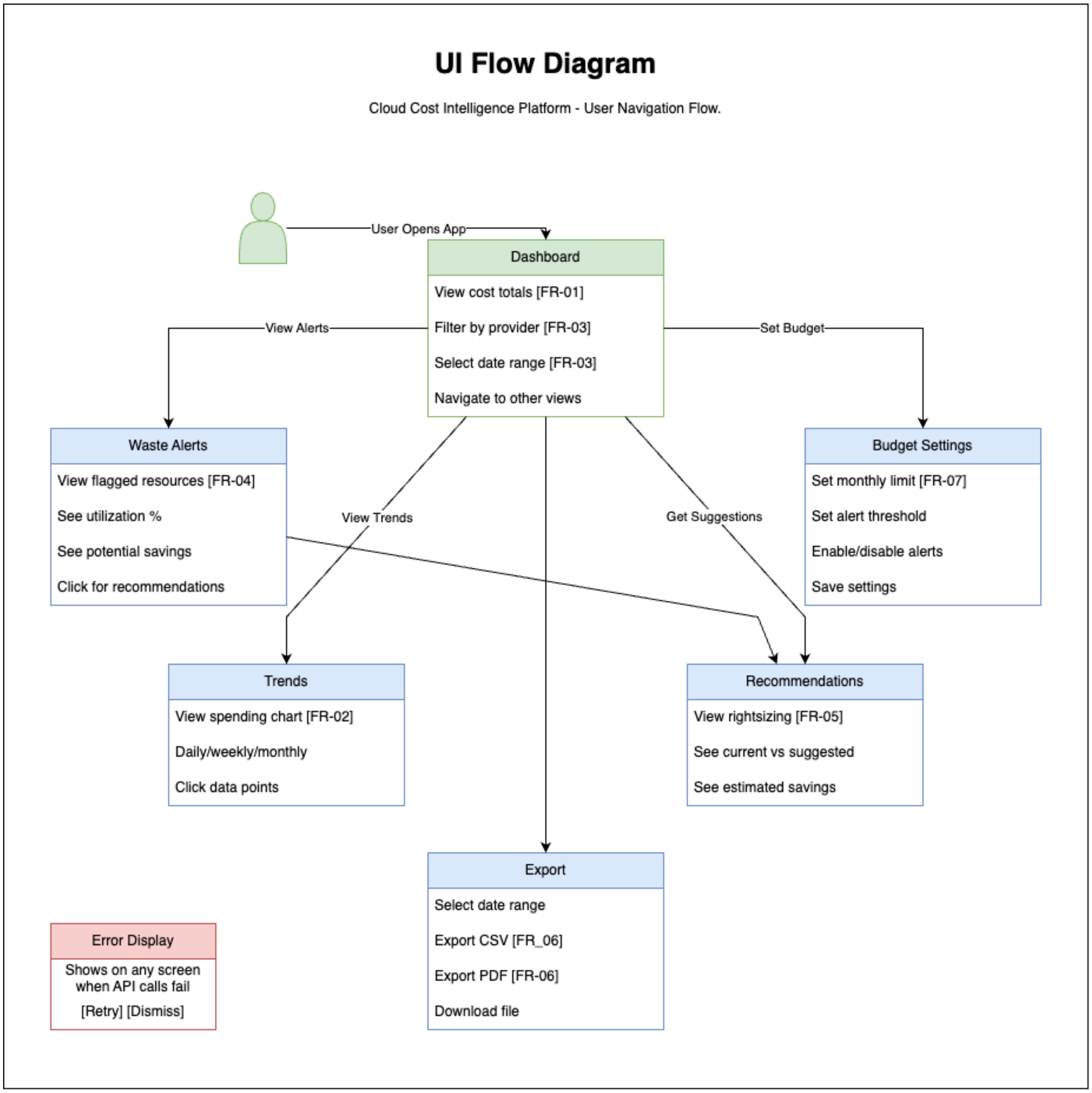


Table	Fields
Clients	ClientID (PK), ClientName, CreatedDate
Providers	ProviderID (PK), ProviderName, CreatedDate
Services	ServiceID (PK), ProviderID (FK), ServiceName, ServiceType, ServiceCost, CreatedDate
Usages	UsageID (PK), ClientID (FK), ServiceID (FK), UsageDate, UsageTime, UnitsUsed, TotalCost, CreatedDate
Invoices	InvoiceID (PK), ClientID (FK), InvoiceDate, InvoiceAmount, CreatedDate
Budgets	BudgetID (PK), ClientID (FK), BudgetAmount, MonthlyLimit, AlertThreshold, AlertEnabled, CreatedDate

3.3 GUI Components



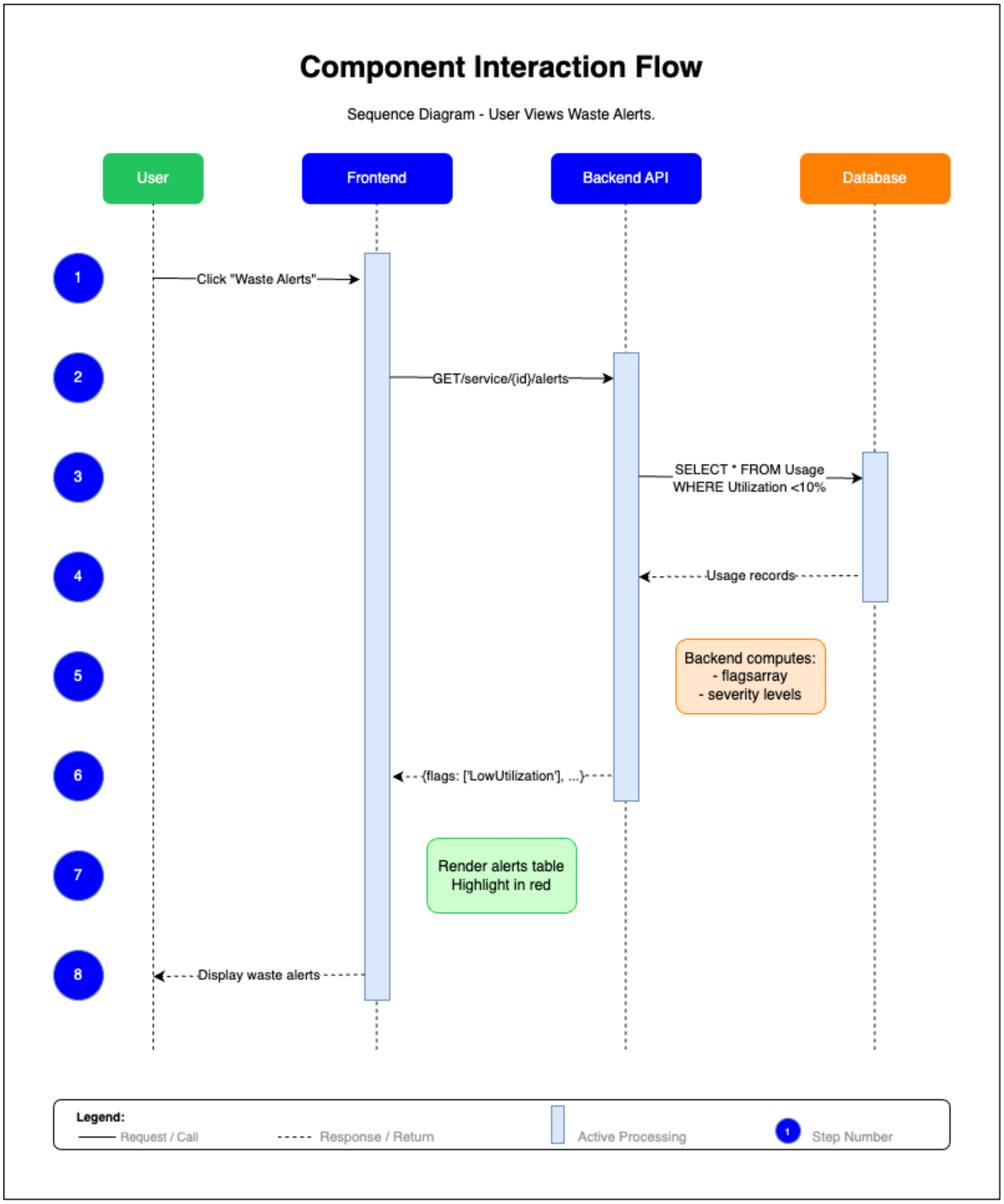
3.4 Component Interaction Flow

Data flow between GUI, Backend API, and Database:

GUI Component	API Endpoint	Returns	Tables Queried
DashboardView	GET/api/costs/summary	Aggregated Totals	Usage, Providers, Services
TrendChart	GET/api/costs/trends	Date-grouped costs	Usage
FilterPanel	GET/api/costs?Provider=&date=	Filtered results	Usage, Providers, Services
WasteAlertsList	GET/service/{service_id}/alerts	Computed alerts (flags array)	Usage (Utilization <10%)
RecommendationsPanel	GET/api/recommendations	Computed recommendation	Usage, Services
BudgetSettings	POST/api/budgets	Insert/Update	Budgets
ExportModule	GET/api/export?format=	CSV/PDF	Usage, Services

3.4 Component Interaction Flow *(Continued)*

Example data flow between GUI, Backend API, and Database for “Waste Alerts”:



4. Project Scope

4.1 Objectives

- Deliver a functional proof-of-concept dashboard for multi-cloud cost monitoring
- Demonstrate cost visualization, waste detection, and recommendation features
- Complete all course deliverables within the 8-week timeline

4.2 Deliverables

- Web-based dashboard application
- Python REST API backend
- MS SQL Server database with simulated data
- Project documentation (Plan, Design, Test Plan, User Guide, Final Report)

4.3 Boundaries and Limitations

Category	Key Functions	Time Permitting	Out of Scope
Cloud Data	Simulated	Live API integration	Real time streaming
Providers	AWS, Azure, Google Cloud	GCP	Beyond 3 providers
Recommendations	Hard code rules	Intelligent/ML	Automatic / Chat bot
Export	CSV + PDF	Direct email	Chat integration
Authentication	None (demo)	Basic login	Multi-user
Alerts	User-defined thresholds	Email notifications	Text notifications

5. Requirements

5.1 Functional Requirements

ID	Requirement	Description
FR-01	View Cost Dashboard	Display total costs by provider on single screen
FR-02	View Spending Trends	Show costs over time with daily/weekly/monthly charts
FR-03	Filter by Provider/Service	Drill down by AWS, Azure, GCP or by Service
FR-04	View Waste Alerts	Flag resources that are unused or underutilized
FR-05	View Recommendations	Show rightsizing suggestions with potential savings
FR-06	Export Reports	Download cost data as CSV or PDF
FR-07	Set Budget Thresholds	Configure spending limits with user-friendly alert thresholds. Users set custom warning levels.
FR-08	View Resource Metrics	Display global average or global max resource usage. User can select metric type and date range.

5.2 Non-Functional Requirements

Category	Requirement
Performance	Dashboard loads within 3 seconds
Compatibility	Supports Chrome, Firefox, Edge (latest versions)
Usability	Intuitive navigation; minimal training required
Maintainability	Modular code structure for future enhancements

6. Methodology

Our team will use an Agile-inspired approach that fits the 8-week academic schedule. We chose this method for several reasons:

1. **We have an 8-week schedule with weekly deliverables.** Waterfall uses a step-by-step process where each phase finishes before the next starts. In our course, we need to deliver something each week, which fits better with Agile's sprint-based approach.
2. **Our team is spread across three time zones:** EST (Maryland, Virginia), PST (Washington), and CET (Italy). Agile focuses on regular communication, daily meetings, and flexible check-ins, which helps us work together more easily than Waterfall's reliance on detailed documentation.
3. **Our main eight** functional requirements are set, but we figure out details like API contracts, data formats, and UI layout as we go. Agile is designed to handle changing requirements, while Waterfall is less flexible.
4. **Integration complexity:** Our three-tier architecture (frontend, backend, database) requires continuous integration. Agile's incremental delivery allows us to integrate and test components weekly rather than waiting until a final integration phase.
5. **Risk mitigation through iteration:** If a feature proves too complex (e.g., AI-powered recommendations), Agile allows us to descope to a simpler solution (rule-based recommendations) mid-project. Waterfall would require a formal change control process.

6.1 Development Phases

1. Planning (Weeks 1-2): Team formation, project selection, requirements gathering
2. Design (Week 3): Architecture, database schema, UI wireframes
3. Development Phase I (Week 4): Core infrastructure, database setup, basic API
4. Testing (Week 5): Test plan execution, bug fixes
5. Development Phase II (Week 6): Feature completion, polish
6. Documentation (Week 7): User guide, final testing
7. Delivery (Week 8): Final report, presentation preparation

6.2 Communication Cadence

Type	Frequency	Purpose
Saturday Sync	Weekly, 3:30 PM EST	Full team sync, section briefs, Q&A
Async Check-ins	Monday & Thursday	Status updates, flag blockers
Daily Standups	As needed via Teams	Quick updates, coordination

Saturday Sync Agenda (30 min):

- 5 min - Frontend (Ishan)
- 5 min - Backend (Sean)
- 5 min - Database (Tony)
- 5 min - Test & QA (Bryana)
- 5 min - Project & Documentation (Michael)
- 5 min - Decision Q&A

6.3 Phase I Support Assignments

Primary leads focus on their areas, Additional support assigned to balance workload:

- Test & QA Lead -> Frontend support
- Project Manager -> Backend support

6.4 Team Norms

- Respond to messages within 24 hours
- Attend scheduled syncs or notify in advance
- Push code to your branch at least every 2 days
- Flag blockers early — no surprises
- Stay engaged even when your specific tasks are light; help across areas
- Developer unit testing before formal QA

6.5 Key Architectural Decisions

Decision	Rationale
Alerts computed by backend	Dynamic threshold logic, not stores data
Recommendations computed by backend	Rule-based IF/ELSE, avoids stale data
Per-report analysis	Simpler than arbitrary timespan queries with gap handling
60-90 days simulated data	Supports trend analysis without overcomplicating DB
10 clients, hourly granularity	~40K records, manageable for proof-of-concept

6.6 Decision Log

Major decisions (scope of changes, tech pivots) documented in Teams or GitHub Issues with date, decision, and rationale.

7. Schedule and Milestones

Week	Phase	Deliverable	Points	Due Date
1	Planning	Team Formation	0	13 Jan 2026
2	Planning	Project Plan	100	20 Jan 2026
3	Design	Project Design	100	27 Jan 2026
4	Development	Phase I Source Code	100	3 Feb 2026
5	Testing	Test Plan	100	10 Feb 2026
6	Development	Phase II Source Code	100	17 Feb 2026
7	Documentation	User Guide	50	24 Feb 2026
8	Delivery	Final Report	300	3 Mar 2026

7.1 Key Milestones

Milestone	Target Date	Owner
Database ready for backend connection	Jan 24	Tony
Backend skeleton pushed to GitHub	Jan 24	Sean
Backend API endpoints Functional	Jan 31	Sean
Frontend components mapped	Jan 31	Ishan
Frontend connected to backend	Feb 1	Ishan
All features integrated	Feb 14	Team
Final testing complete	Feb 21	Bryana

7.2 Task Dependencies

Database Setup (Tony) -> Backend API (Sean) -> Frontend Integration (Ishan) -> Testing & QA (Bryana) -> Documentation (Michael)

8. Tasks and Resource Assignments

8.1 Phase I Task Assignments

Task (~35-40 hours team/ 8 per person)	Owner	Support	Est. Hours
Database			
Database schema + tables	Tony		Complete
Azure SQL Server setup	Tony		Complete
Simulated data generation (~40k rows)	Tony		4-6 Hours
Backend			
API documentation to GitHub	Sean		Complete
Flask API server + DB connection	Sean	Michael	4 hrs
Stubbed endpoints (mock data)	Sean	Michael	2 hrs
API endpoints (FR-01 through FR-08)	Sean	Michael	8-10 hrs
Frontend			
Dashboard skeleton (HTML/CSS, responsive)	Ishan	Bryana	4 hrs
Navigation bar + routing between views	Ishan	Bryana	2 hrs
Trend chart with Chart.js (mock data)	Ishan	Bryana	3 hrs
Frontend-backend integration	Ishan	Sean + Michael	4 hrs

8.2 Phase II Task Assignments

Task (~25-30 hours team/ 6 per person)	Owner	Support	Est. Hours
Dashboard/Analytics polish	Ishan	Bryana	6 hrs
Recommendations engine	Sean	Michael	6 hrs
Export functionality (CSV/PDF)	Sean	Ishan	4 hrs
Bug fixes and polish	Team		6 hrs

8.3 Technical Resources

Resource	Details
Development Machines	Personal computers (Windows/Mac/Linux)
Version Control	GitHub (repository: CMSC495-CloudCost)
Database Server	Azure SQL: cmsc495-cloud-cost.database.windows.net
Database	CloudCostDatabase
Frontend	HTML/CSS/JavaScript, Chart.js (React optional)
Backend	Python, Flask
Communication	MS Teams, GitHub Issues
IDE	VS Code, GitHub Codespaces
Budget	All services are free tier. Trial services cover project scope.

8.4 Data Generation Plan

Parameter	Value
Clients	10 simulated companies
Providers	AWS, Azure
Services	20-50 (EC2, S3, Azure VM, Azure Storage, etc.)
Usage records	~40,000 rows
Time range	60-90 days historical
Granularity	Hourly
Source inspiration	Azure billing data patterns

8.5 Summary

Estimated Total Project Effort: Phase I: ~35-40 hours (~7-8 hrs per person). Phase II: ~20-25 hours (~4-5 hrs per person). **Total: ~55-65 hours across 5 team members over 4 weeks**

9. Risk Assessment

Risk	Prob	Impact	Mitigation Strategy
Team Availability / Time Zones	High	High	Weekly Saturday syncs accommodate all. Async communication via Teams. Recorded meetings.
Team Member Drops Course	Low	High	Work on all components as a team. Document code thoroughly. No single point of failure.
Unfamiliarity with Tech Stack	Med	Med	Select the technologies team already knows. (Python, React, HTML/CSS/JS, MS SQL Server).
Scope Creep	Med	High	Firm scope boundaries in Section 3. PM gatekeeps new features.
Integration Issues	Med	Med	Define API contracts early. Integrate continuously. Use feature branches.
Schedule Conflict Issues	High	Low	Will complete action items before absence, monitor from phone, engage and ask for help early
Database Access Issue	Med	Med	Firewall exceptions handled by Database Lead; team sends IP for whitelisting
Schema Changes Late	Low	High	Schema locked by end of Week 3; changes require team discussion
QA Lead Travel (Feb26 – Mar2)	Known	Med	Plan testing handoff before travel; overlaps User Guide and Final Report crunch

10. Evaluation Plan

Success will be measured against the following criteria:

10.1 Functional Criteria

ID	Requirement	Success Metric	Verification Method	Pass Threshold
FR-01	Cost Dashboard	AWS and Azure cost displayed on dashboard load	Visual inspection + automated test	Both provider totals render within 3 seconds
FR-02	Spending Trends	Chart renders with selectable time periods	Manual test with toggle	Daily, weekly, AND monthly views all functional
FR-03	Filters	Provider/service/date filters update display	Manual test each filter	All 3 filter types return correct filtered data
FR-04	Waste Alerts	Resources with <10% utilization flagged	Query Validation against test data	≥90% of low-utilization resources correctly identified
FR-05	Recommendation	Rightsizing suggestions displayed	Count recommendations for test dataset	≥3 recommendation generates for test client
FR-06	Export	CSV and PDF download functionality	Download and open files	Both formats download, open without corruption
FR-07	Budgets	User-configurable limits with alerts	Set threshold, exceed it, verify alert	Alert triggers when spending exceeds threshold
FR-08	Metrics	Average and max usage metrics displayed	Compare displayed values to raw data	Calculated values match manual calculation ± 1%

10.2 Technical Success Criteria

Performance targets:

Criteria	Target
Test pass rate	≥80%
Browser support	Chrome, Firefox, Edge
Dashboard load	<3 seconds
API response	<2 seconds
DB queries	Accurate results

10.3 Project Success Criteria

How we know the project succeeded:

Criteria	Target
On-time delivery	100%
Peer review avg	≥7/10
User Guide	Reviewed by non-team member
Final demo	All 8 FRs working

10.4 Evaluation Methods

Who tests what and when:

Method	**Who**	**When**
Unit testing	Each dev	During coding
Integration testing	Bryana	Phase I & II
User acceptance	Team	Before final submit
Code review	Team	GitHub PRs
Peer eval	Everyone	Week 4 & 8

11. Testing Strategy

11.1 Testing Approach

Developer unit testing occurs before formal QA. Each developer tests their own code to validate core functionality and fix defects before integration.

11.2 Testing Phases

Phase	Owner	Focus
Database Testing	Tony	Tables, relationships, queries, sample data
Backend Testing	Sean + Michael	API endpoints, parameters, error handling
Frontend Testing	Ishan + Bryana	UI elements, dashboards, charts, filters, cross-browser
Integration Testing	Team	Frontend-backend-database communication
Final QA	Bryana	Usability, professional appearance, first-time user walkthrough

11.3 QA/QC Criteria

- Interface is clear, consistent, and easy to use
- All functional requirements (FR-01 through FR-08) verified
- Error handling tested for all user inputs
- Cross-browser testing (Chrome, Firefox, Edge)

TEAM APPROVAL

By submitting this document, all team members confirm agreement with this project plan.

Team Member	Role	Date
Ishan	Frontend Lead	1/24/2026
Michael	Project Manager	1/27/2026
Bryana	Test/ QA Lead	1/27/2026
Sean	Backend Lead	1/27/2026
Tony	Database Developer	1/26/2026