Azure migration costs calculator with embedded AI
Final Report
CS 4850 - Section 02 – Fall 2024
Team Website: https://psenior4850.github.io/4850Azure/
C-Day Website: https://psenior4850.github.io
Number of lines of code: ~2900
Total man hours: ~440
Number of project components/tools: 15

Roles	Name	Major responsibilities	Contact
Project owner	Capstone		
Team leader	Kunal Shenoi	Organize meetings, monitor progress, and submit deliverables. Develop and implement testing protocols.	678-779-4770
Team members	Graham Allen	Developer - Developing back-end AI integration, implementing Azure/AWS API calls, and database connections. Assisting in front-end UI/UX development	770-841-6718
	Angel Hernandez	Developer - Designing and implementing a user-friendly interface, while assisting with backend and AI integration and development	770-318-5359
	Yvan Ngah	Documentation - Creating, maintaining, and ensuring the accuracy of all technical documentation.	171haden@gmail.com
Advisor / Instructor	Sharon Perry	Facilitate project progress; advise on project planning and management.	770-329-3895

Kunal Shenoi
Team Leader, Test

Graham Allen
Developer

Angel Hernandez
Developer

Yvan Ngah
Documentation

Table of Contents

Introduction

Project Overview

This document provides a comprehensive overview of the development, installation, configuration, and usage of the Azure Migration Costs Calculator with Embedded AI. The purpose of this project is to enable developers, engineers, and stakeholders to accurately assess the financial implications of migrating from on-premises infrastructure to Azure cloud services. Through an AI-augmented interface, this tool allows users to evaluate costs based on real-time Azure pricing data, aiding decision-making for efficient cloud adoption.

Background

As enterprises increasingly shift to cloud-based infrastructure, understanding the costs associated with cloud migration has become essential. The Azure Migration Costs Calculator with Embedded AI was chosen as a project to address the complexities and financial uncertainties associated with cloud migration, particularly for IT and engineering teams. By integrating Azure's pricing API and an AI language model, this tool provides more than just raw cost estimates, it offers insights based on user-specific requirements, thereby reducing manual workload and enabling more informed migration decisions. The motivation for this project lies in the need for accurate, streamlined cost estimation tools to aid in cloud strategy planning, making it highly relevant for today's cloud-centric technology landscape.

Scope of the Report

This report covers the entire development lifecycle and technical details of the Azure Migration Costs Calculator, offering insights into:

- Project Background and Objectives
- Requirements and System Architecture
- Development Process and Implementation
- Testing Methodologies and Results
- Challenges and Lessons Learned
- Summary and Future Improvements

Requirements

Functional Requirements

1. User Input Collection:
    - The system must allow users to input detailed information about their on-premises infrastructure, such as server configurations, storage needs, network requirements, and estimated data transfer volumes.
    - The AI-powered chatbot interface should guide users in specifying their migration parameters and requirements using natural language.
2. Cost Estimation:
    - The application must connect to the Azure Pricing API to retrieve up-to-date cost data for Azure services relevant to the migration process.
    - It should calculate estimated migration costs based on user inputs, such as infrastructure specifications, and recommended Azure services.
3. Real-Time Recommendations:
    - The system should provide recommendations on migration strategies (full cloud, hybrid, or on-premises) based on input data and user preferences.
    - The chatbot interface must allow users to dynamically adjust their inputs and receive refined recommendations in real-time.
4. Reporting and Visualization:
    - Generate a detailed migration cost report that can be downloaded or viewed in-app, including breakdowns by service and estimated monthly and annual costs.
    - Include a dashboard or visualization tool to visually display key metrics, cost components, and potential cost savings.
5. Data Storage and Management:
    - The system should save user inputs, historical estimates, and related data in a database.
    - It must ensure data is stored securely, with access restricted to authorized users.
6. User Authentication and Authorization:
    - Implement user authentication to ensure that only authorized personnel can access or manage sensitive data and reports.

Non-Functional Requirements

1. Performance:
    - The system should respond to user queries and deliver cost estimates within 2 seconds of submission.
    - Ensure database queries and API calls are optimized to handle multiple users simultaneously without significant delay.
2. Scalability:
    - The application should support scalable deployment to accommodate an increase in the number of users without compromising performance.
3. Security:

    - Implement encryption for all sensitive data, both in transit and at rest.
    - Use secure access tokens for API interactions and restrict database access only to authorized users.
4. Usability:
    - The chatbot interface should be intuitive and guide users step-by-step through the process of defining their migration needs.
    - The application should be accessible across common browsers and compatible with assistive technologies.
5. Maintainability and Reliability :
    - The codebase should be modular and thoroughly documented, facilitating easy updates and debugging.
    - Use CI/CD pipelines to automate testing and deployment.

Assumptions and Constraints

1. Assumptions:
    - It is assumed that users have an Azure account with the required permissions to access Azure Pricing APIs.
    - Users are expected to have some familiarity with cloud infrastructure and terminology, as the tool does not cover introductory education on cloud concepts.
    - The application assumes that the input data provided by users is accurate and complete, as estimates are based on the data given.
2. Constraints:
    - API Limitations: Azure Pricing API has rate limits and response time constraints, which could impact the responsiveness of the cost estimation feature.
    - Resource Allocation: Due to project time and resource limitations, the application may not support certain advanced Azure services or other cloud providers.
    - Platform Constraints: The application is designed primarily for web access; mobile compatibility is not prioritized in this release.

Analysis

The Azure Migration Costs Calculator with Embedded AI addresses a critical need for enterprises to understand the financial implications of migrating on-premises infrastructure to Azure cloud services. Current cloud migration tools are often either too simplistic or require advanced knowledge to produce reliable cost estimates, leading to potential overspending or underestimated budgets. This project aims to fill this gap by combining Azure pricing data with AI-driven natural language processing, allowing users to receive tailored recommendations and cost breakdowns based on their specific migration scenarios.

Technical Analysis

Frontend:

- Technology: React
- Reasoning: Flexible and component-based, perfect for building an interactive chatbot.

Backend:

- Technology: Python and Flask
- Reasoning: Python's libraries and Flask's lightweight framework are great for API development, handling requests, and data processing.

AI Language Model:

- API: Ollama API
- Reasoning: Excellent at understanding complex user inputs and extracting relevant information, making the chatbot user-friendly and natural.

Data Storage:

- Database: MySQL
- Reasoning: Reliable, scalable, and handles complex queries well.

Cloud Service Integration:

- API: Azure Pricing API
- Reasoning: Provides real-time Azure pricing data for accurate cost estimates.

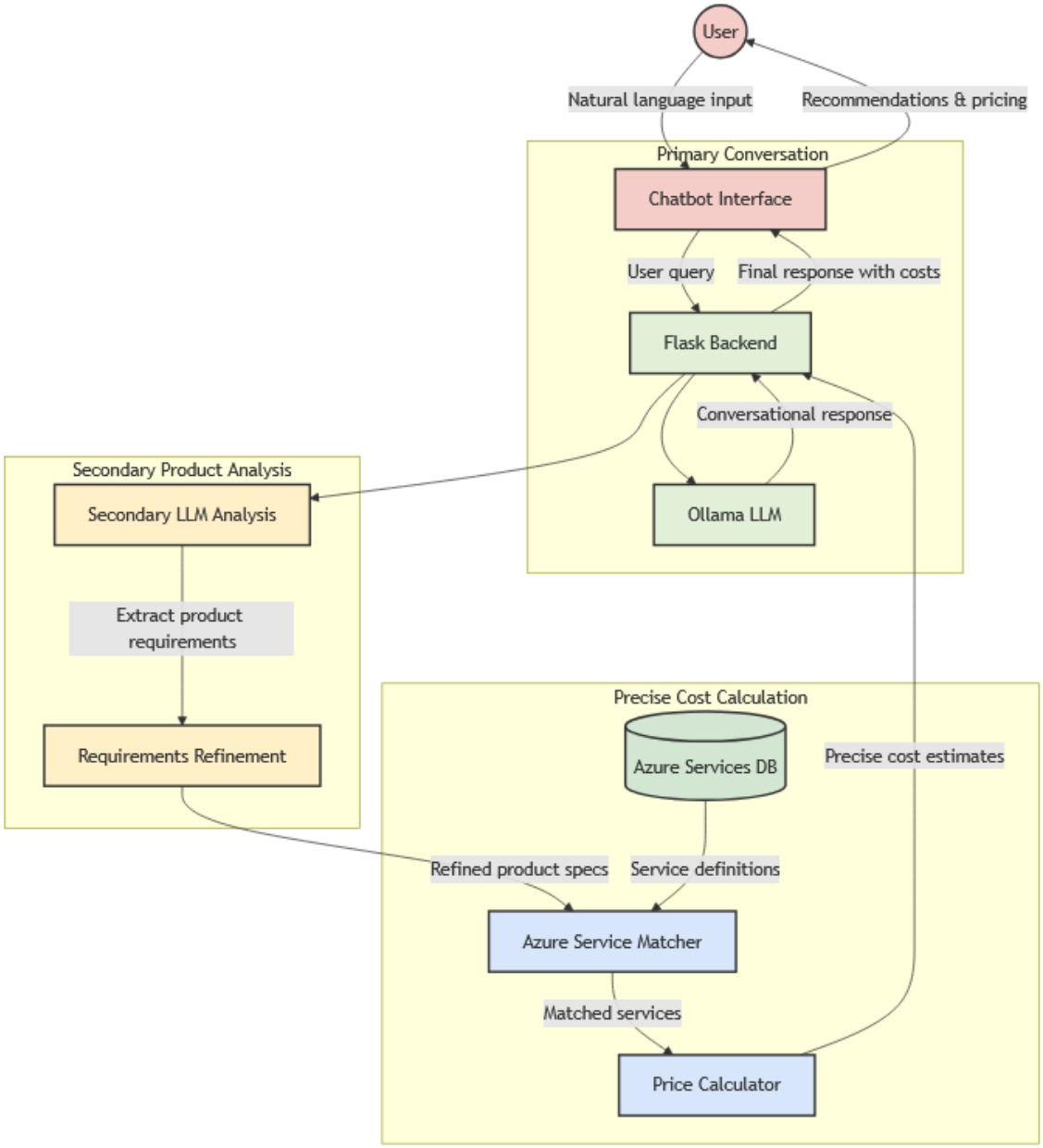
Feasibility Assessment

1. Technical Feasibility:
    - The chosen technologies are supported and are well-documented by large developer communities, making troubleshooting and support feasible. In addition, Azure and Ollama have stable SDKs and API's.
2. Financial Feasibility:

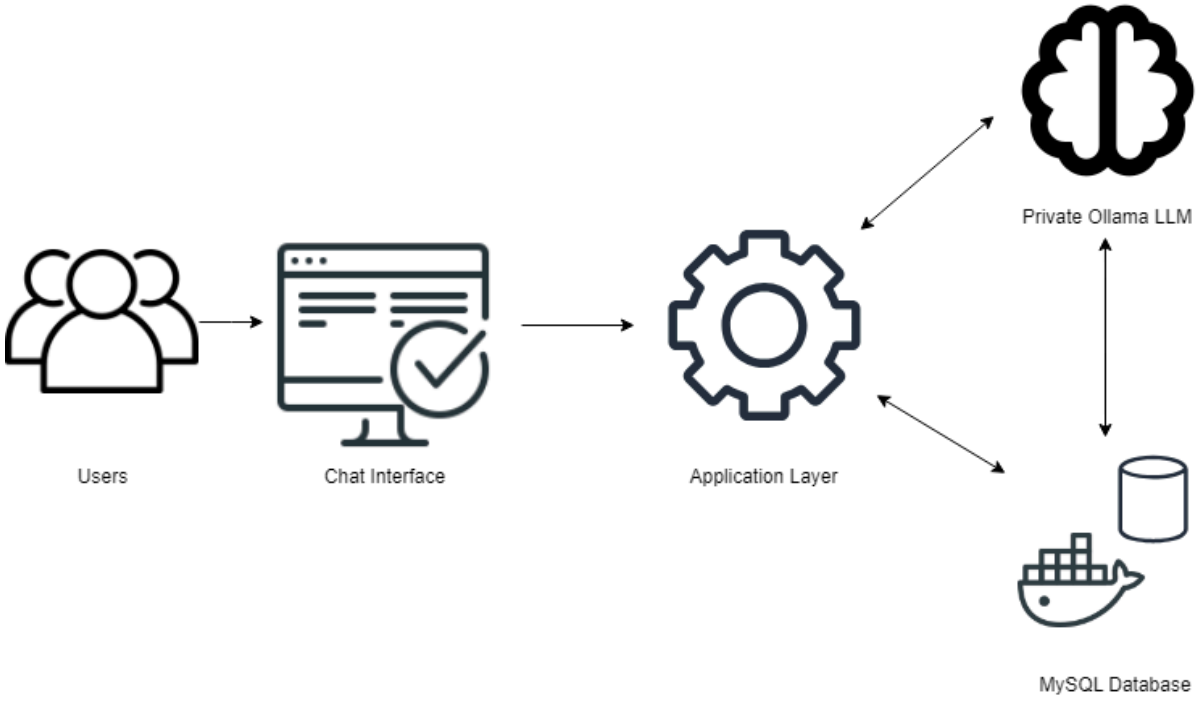
- The project primarily requires open-source software and free-tier cloud resources for initial development and testing. While API costs may arise with scaled usage, the entire project can be completed on self-hosted hardware.

Design

System Workflow Diagram



System Architecture Diagram



Database Entity Relationship Diagram

Development Process

Project Plan

1. Planning:
    a. Set project goals and define key features, focusing on providing Azure migration cost estimates.
    b. Develop a timeline and assign tasks across the team.
2. System Design:
    a. Frontend: React for an interactive chatbot interface.
    b. Backend: Python and Flask for API requests, data processing, and service connections.
    c. Database: MySQL for organized data storage.
3. Frontend Development:
    a. Build the chatbot interface using React to capture migration details.
    b. Connect the frontend to the backend for real-time data updates.
4. Backend Development:
    a. Set up a Flask API to manage user inputs, requests, and Azure API connections.
    b. Integrate the Azure Pricing API for real-time cost data retrieval.
5. Database Setup:
    a. Configure MySQL to store user and pricing data for secure and scalable data management.
6. Testing:
    a. Test each component to ensure a cohesive and secure application.
    b. Validate accurate data handling and secure access.
7. Deployment:
    a. Deploy to a staging environment for final testing.
    b. Configure production-ready settings for the application.

Key Milestones

- Milestone 1: Initial Research and Planning
    - Defined goals, researched cloud solutions, created a project plan and designed the web app
    - Evaluated: Goal clarity, research quality, project feasibility.
- Milestone 2: Development and Integration
    - Developed the front end, implemented the AI chatbot, integrated Azure APIs, and tested
    - Evaluated: AI functionality, integration quality, UI usability.
- Milestone 3: Final Testing and Presentation
    - Developed the report feature, tested, fixed bugs, and prepared final documentation.
    - Evaluated: Overall project completion and usability.

Development Challenges

- Optimizing Performance:
    - Optimized API calls and backend processes.
    - Implemented caching for frequently accessed data.
    - Limited redundant requests to reduce response time.
- Handling Incomplete User Input:
    - Egbert's LLM asks follow-up questions to gather necessary information.
    - Provides accurate recommendations even with incomplete input by providing suggestions.
- Implementing Flask:
    - Transitioned to Flask for backend structure, adapting to its unique framework requirements.
- Querying Ollama:
    - Refined prompts for Ollama API to ensure accurate query responses.
- Login Authentication:
    - Set up secure login and user authentication.

Test Plan

Scope of Testing

In Scope

- Security and Authentication: Testing the account creation, login, and password recovery functionalities.
- Cost Calculation Feature: Validating cost accuracy, response consistency, and Azure Pricing API calls.
- Session handling: Testing the
- LLM Analysis: Ensuring accurate interpretation of user-provided infrastructure details.
- Report Generation: Confirming reports are generated correctly and downloadable.
- Performance testing: Ensuring the system can handle unexpected and varying numbers concurrent of users.

Out of Scope

- Third-party API Testing: We assume Azure will be reliable, so we will not test the Pricing API reliability and performance as part of this project.
- Browser Compatibility: Limited to major browsers; less popular or outdated browsers are out of scope.
- Hardware Compatibility: We were provided with what we assumed to be deployment-ready hardware which will be the same hardware the end user will be using.

Testing Approaches

A mix of manual and automated testing approaches will be used, supported by CI/CD pipelines

and frameworks. All critical user stories will be tested manually in the initial phase, followed by automated test scripts for final testing.

Types of Testing

- Functional Testing: Verifying the core functionalities, including cost calculations, report generation, and authentication.
- Performance Testing: Evaluating application performance under varying loads and ensuring response times meet expectations.
- Security Testing: Checking for common vulnerabilities such as SQL injection, endpoint manipulation, and authentication security.
- Usability Testing: Assessing the UI design for accessibility, clarity, and ease of use.

Testing Schedule

Testing Phase	Planned Start Date	Planned End Date
Test Plan Creation	October 1, 2024	October 7, 2024
Functional Testing	October 8, 2024	October 21, 2024
Integration Testing	October 22, 2024	October 30, 2024
Performance Testing	October 31, 2024	November 7, 2024
Security Testing	November 8, 2024	November 14, 2024
Usability Testing	November 15, 2024	November 19, 2024
Final Testing and Review	November 20, 2024	November 23, 2024

Risks Associated with the Testing Process

- Dependency on Azure Pricing APIs: Unavailability or changes in Azure Pricing APIs could delay testing or affect the accuracy of cost calculations.
- Resource Constraints: Limited availability of testers, especially for specialized testing like security.
- Time Constraints: Adherence to a strict project timeline may impact the thoroughness of the testing.

Test Report

Test Summary

The purpose of testing is to ensure that the Migration Cost Calculator meets functional, performance, security, and usability requirements. Testing includes functional, integration, performance, and security testing to validate that all user requirements are met and that the application operates reliably under various conditions.

Tests Completed

Requirement	Pass/Fail	Severity
Create Account	Pass	High
Login	Pass	High
Logout	Pass	High
Session Timeout	Pass	High
Report Generation	Pass	High
Report Export	Pass	Medium
Azure Pricing Integration	Pass	High
API Error Handling	Pass	High
Chat History	Pass	Low
Cost Calculation Accuracy	Pass	High
Consistency Testing	Pass	High
Browser Compatibility	Pass	Medium
API Rate Limit Handling	Pass	High
Data Input Validation	Pass	Medium
Delete Old chats	Pass	Medium
Website routes	Pass	High
Security (SQL Injection)	Pass	High
Stress Testing	Pass	High
Concurrent users	Pass	High
Screen Resolution adjustments	Fail	Low
Error Messages	Pass	Low
Performance Testing	Pass	High
Deleting all chats	Pass	Medium
Start with at least one chat	Pass	Medium
New chat	Pass	High
Execute inputs with an enter	Pass	Low
All buttons	Pass	Medium
Sessions with login/logout	Fail	Low
Chatbot Accuracy	Fail	High

Overall Test Metrics

Test Metrics	Count
Test Cases Planned	29
Test Cases Executed	29
Test Cases Passed	26

Test Cases Failed	3

Total Bugs

- Total Defects Identified: 1

Status of Bugs

- Open: 0
- Closed: 2
- In Progress: 1

Severity Breakdown

- High Severity:
    - Count: 1
    - Description: High-severity defects directly impact application functionality and user experience.
- Medium Severity:
    - Count: 1
    - Description: Medium severity issues are impactful but do not hinder core functionality and may only happen under specific situations.
- Low Severity:
    - Count: 1
    - Description: Low-severity issues do not affect application functionality and will be addressed once medium and high-severity issues have been fixed.

Testing Environments

- Operating Systems: Windows, macOS, Linux.
- Browsers Tested: Chrome, Firefox, Safari, Edge.
- Tools Used: GitHub Issues for bug tracking, Google Docs for documentation, GitHub Actions for CI/CD.

Final Recommendations

- Prioritize resolving high-severity issues impacting the core functionality.
- Implement performance optimizations to address any potential latency issues.
- Review and address remaining low-severity UI issues post-deployment.

Version Control

Version Control Setup

We used Git for version control, organizing our project into multiple repositories within a centralized organization on GitHub. The primary branches were:

- Main Branch: Hosted the live build, integrating stable versions of both the front and back-end components.
- Backend and Frontend branches: Separate branches for testing backend and frontend changes independently before merging into the main branch.
- Working build branch: Combined the frontend and backend for integration testing.
- Prompt testing branch: Dedicated to testing LLM prompts, analyzing model responses, and improving prompt accuracy.

Commit Breakdown

Our commit strategy involved regular, detailed commits to track incremental changes and facilitate collaboration.

We followed these guidelines:

- Feature branches: For major features or changes, we created separate branches to isolate work.
- Frequent commits: To ensure traceability, we committed frequently with clear messages, especially for critical updates.
- Merging: After rigorous testing and meetings, changes were merged into the working build branch for integration testing and ultimately into the main branch for deployment.

Github Statistics

- Working build branch: Over 100 commits, making it our most active branch.
- Backend and Frontend branches: Approximately 60 commits each.
- Prompt testing branch: Around 10 commits as it was mainly for testing.

Challenges in Version Control

Managing separate repositories and branches for individual contributions presented occasional challenges in synchronizing code.

To address these:

- We implemented regular check-ins and code reviews to resolve conflicts.
- We held weekly updates to discuss branch progress and coordinate merges to avoid merge conflicts and ensure smooth integration.
- We promptly identified and resolved minor to severe bugs to keep the project on schedule.

Summary and Conclusion

Project Accomplishments

- Developed a fully functional Azure Migration Costs Calculator with a user-friendly chatbot interface.

- Achieved accurate cost estimates through Azure API integration and AI-based analysis of migration requirements.
- Implemented robust authentication and optimized API calls to improve responsiveness and user experience.
- Successfully tested and validated core functionalities, ensuring a reliable tool for enterprise migration cost assessment.

Lessons Learned

- The importance of modular version control setup, especially in a multi-contributor project, for efficient development and conflict resolution.
- How LLM prompt engineering requires iterative refinement to handle diverse user inputs and deliver precise results.
- How to document progress and deliver milestone updates to sponsors, ensuring transparency and alignment with project goals.
- Effective branch management can streamline testing and improve integration processes, particularly for frontend-backend projects.

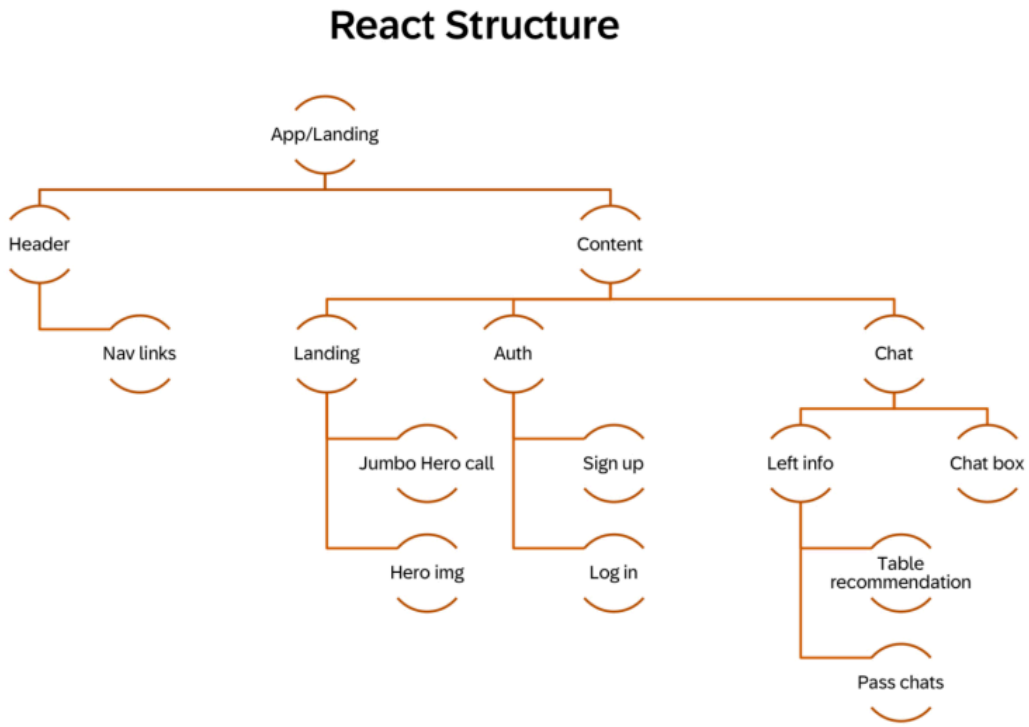
Future Work or Improvements

- Enhanced Reporting Features:
    - Offer more detailed migration analysis reports with cost breakdowns and trend insights.
- Multi-Cloud Integration:
    - Extend support to include additional cloud providers, making the tool applicable to multi-cloud environments.
- Migration Scenario Simulation and Comparison:
    - Enable users to simulate and compare various migration scenarios to make informed decisions.
- Automated Discovery Agent:
    - Implement a discovery tool to automatically gather on-premises infrastructure details.
- Benchmarking Tools:
    - Include benchmarking features to evaluate on-premises vs. cloud performance and cost efficiency.
- Integration with CI/CD Pipelines:
  Provide integration with CI/CD pipelines for smoother deployments and automated testing.
- Alerts and Notifications:
    - Set up notifications to alert users about changes in Azure pricing or cost estimates for continuous cost monitoring.

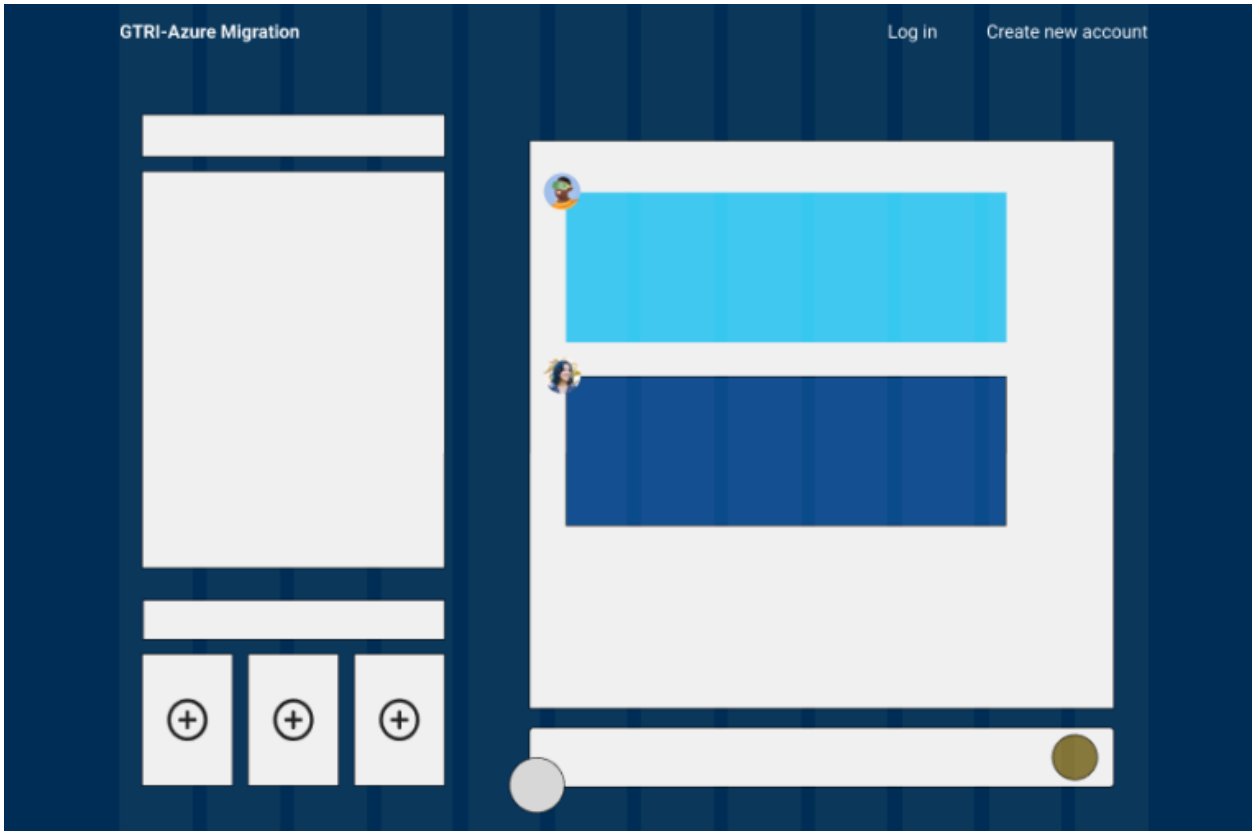
Appendix

Screen Mockups and UI Designs

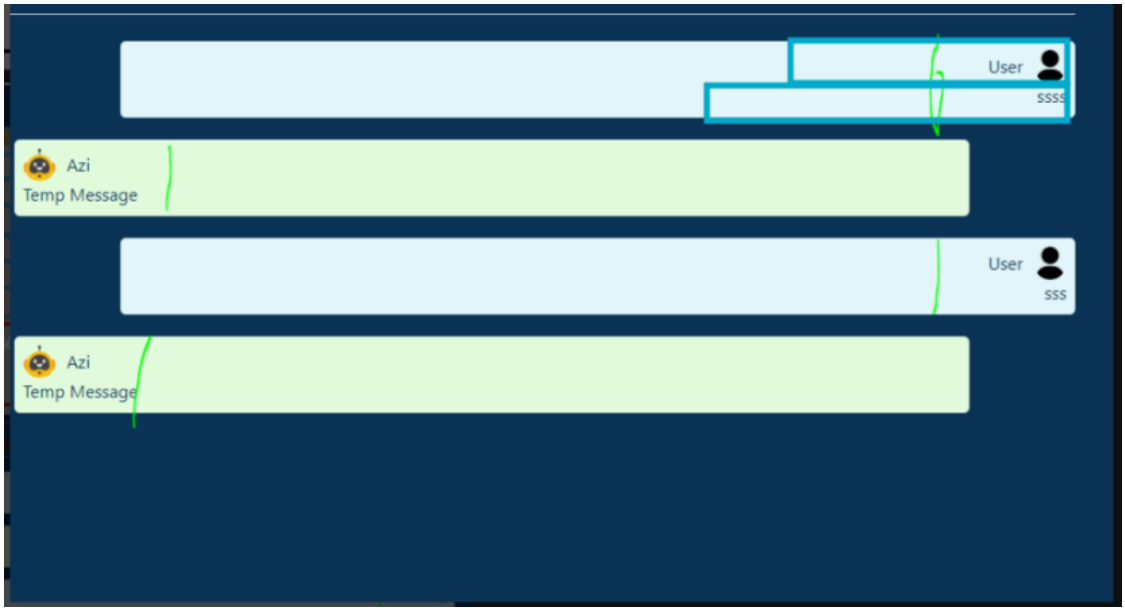
React Structure:



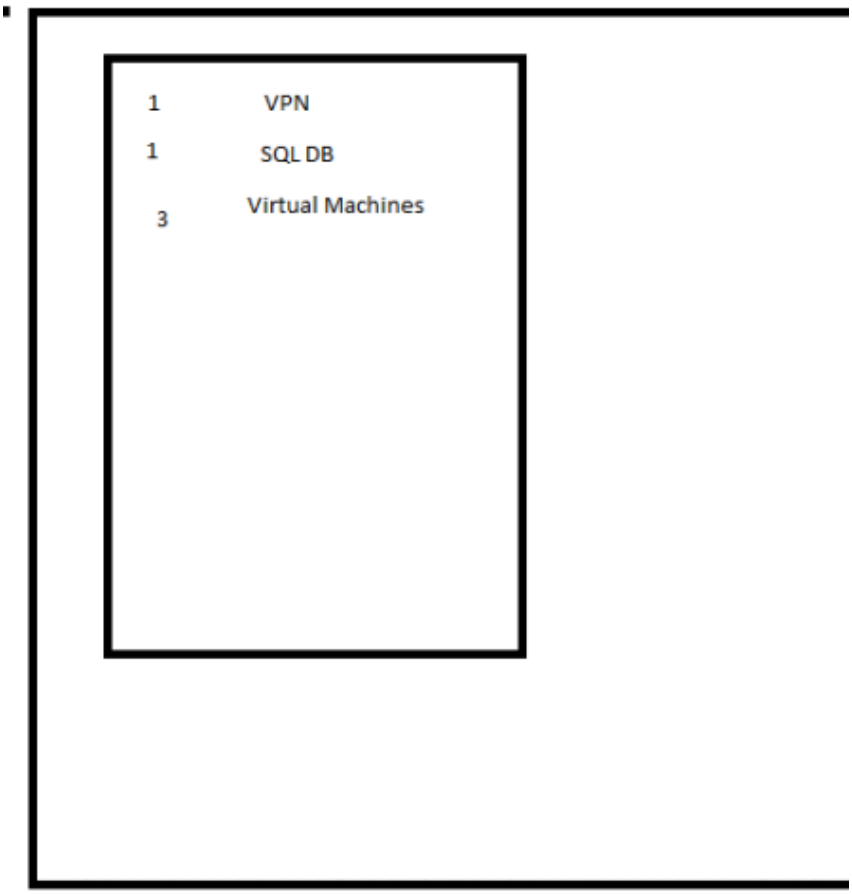
Initial Website Mockup:



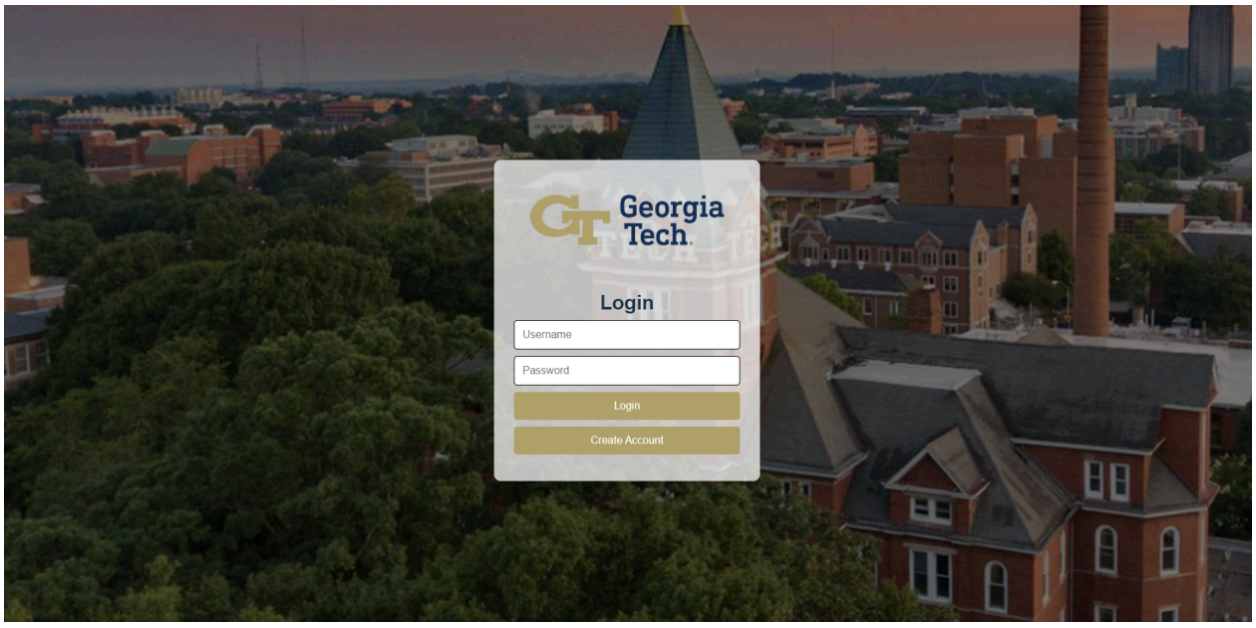
Improving Chat UI:



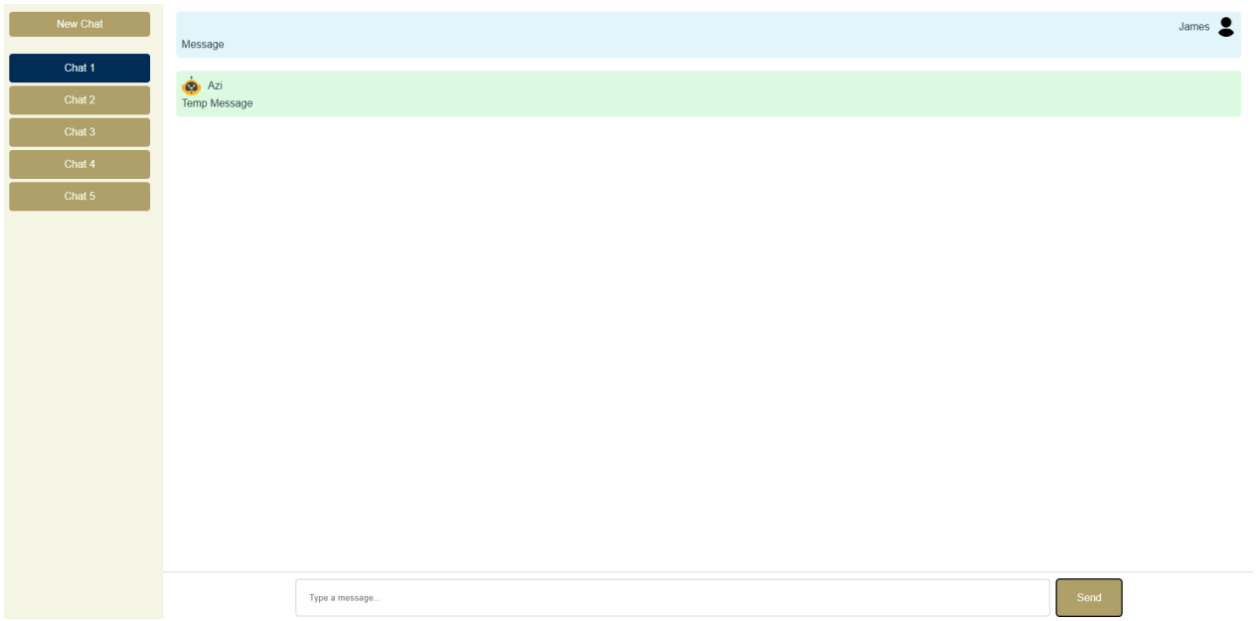
Migration Details Mockup:



Initial Login Page:



Initial Chatbot Interface:



Initial workflow diagram: