* Test plan and strategy
  + Kas - Done
* Test cases (including edge cases)
  + Kenneth – Phi3 (+ more?)
    - Vongai – Testing Classifier/ from scratch – VK done
* Results (pass/fail metrics, screenshots if applicable)
  + Kas – Post images and metrics - Done
* Bug tracking or issues found
  + Endpoints (Kas) - Done
  + Github Branch management (Kenneth) - Done
  + OpenAI/ Phi3 version mismatch (Vongai) - Done
  + Azure/ Database Corrections (Rod) - Done
  + Virtual server/ Applications (Adam) - Done
    - Launching Server front and backend (Adam) - Done
* Lessons learned or next steps based on results - Done
  + Adam - Done

**Test Cases**: These are specific scenarios designed to verify that your application functions correctly. Each test case typically includes inputs, expected outputs, steps to execute, and success criteria.

**Edge Cases**: These are extreme or unusual scenarios that test the boundaries of your system and its ability to handle exceptional conditions. For example, testing with empty files, extremely large files, malformed input, or unexpected data formats.

**Executive Summary - AJ**

This report documents the testing strategy, execution, and results for the CapSense AI project, a sentiment analysis and response generation application developed for Capgemini. The testing focused on the core functionalities including batch processing of feedback, sentiment classification, emotion detection, sarcasm identification, and AI-driven response generation using the Phi-3 Mini model on Azure.

Testing was conducted across multiple dimensions including functionality, integration, performance, and user experience. This report identifies key findings, documents encountered issues, and provides recommendations for future enhancements.

**1. Test Plan and Strategy -**

**1.1 Testing Objectives**

* Verify the accuracy and reliability of:
  + Sentiment classification model
  + Emotion classification model
  + Sarcasm classification model
* Validate the functionality of batch processing for customer feedback
  + Ensuring each comment is processed
* Assess the quality of AI-generated responses using the Phi-3 Mini model
* Ensure proper integration between frontend components and backend APIs
  + Endpoints aligning
  + Ability to access the models
* Evaluate system performance under various load conditions
* Verify database interaction and data persistence
* Assess the user experience and interface functionality

**1.2 Testing Environments**

|  |  |  |
| --- | --- | --- |
| **Environment** | **Description** | **Purpose** |
| **Local Development** | Windows/ MacOS with Python 3.9+, Flask, React | Unit testing, component testing |
| **Azure VM** | Linux-based VM with deployed backend services | Integration testing, API validation |
| **Azure App Services** | Production environment for final deployment | End-to-end testing, performance testing |

**1.3 Testing Types**

1. **Unit Testing**: Validation of individual components and functions
2. **Integration Testing**: Verification of interactions between system components
3. **Functional Testing**: Validation of system requirements and features
4. **Performance Testing**: Assessment of system behavior under various conditions
5. **User Acceptance Testing**: Evaluation of the system from an end-user perspective

**1.4 Testing Tools**

* **Postman**: For API testing and validation
* **Jest/React Testing Library**: For frontend component testing
* **pytest**: For Python backend unit testing
* **Azure Monitor**: For performance monitoring and logging
* **Manual testing**: For user experience and interface validation

**1.5 Testing Schedule - AJ**

|  |  |  |
| --- | --- | --- |
| **Phase** | **Duration** | **Focus Areas** |
| **Phase 1** | 1 week | Unit testing of backend classifiers and response generation |
| **Phase 2** | 1 week | Integration testing of API endpoints and database connectivity |
| **Phase 3** | 1 week | Frontend testing and UI validation |
| **Phase 4** | 1 week | End-to-end testing and performance evaluation |

**2. Test Cases**

**Test Cases – Emotion Classifier (Naive Bayes)**

**Overview:**  
 The emotion classifier was developed using a Naive Bayes model trained on 42,000+ feedback entries, reduced to six primary emotion classes. The classifier was integrated into the Flask backend and tested both in isolation and through the /api/respond\_batch endpoint. Below are representative test cases and edge cases to validate its performance and robustness.

**Core Test Cases:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Input Text** | **Expected Emotion** | **Notes** |
| TC-01 | "I’m so frustrated with the late delivery." | anger | Classifier correctly detects frustration as anger. |
| TC-02 | "Thank you so much for the fast shipping!" | joy | Direct positive sentiment linked to joy. |
| TC-03 | "I think it might be okay... I guess?" | neutral | Ambiguous phrasing returns neutral. |
| TC-04 | "Waiting again? Typical service!" | anger | Classifier flags passive-aggressive tone as anger. |
| TC-05 | "Can’t wait to try this out!" | anticipation | Positive outlook tied to anticipation. |

**Edge Case Tests:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Input Text** | **Expected Behavior** | **Result** |
| EC-01 | "" (Empty string) | Handled gracefully without crash; may return neutral or error message | Passed |
| EC-02 | "asdfghjkl" (nonsensical input) | Should not crash; return neutral or undefined | Passed |
| EC-03 | Long mixed-emotion input (multiple conflicting tones) | Choose dominant emotion or return neutral | Passed – returned strongest emotion |
| EC-04 | Upper/lower case variation ("JOY" vs "joy") | Case-insensitive classification | Passed |
| EC-05 | Minor typos ("frustratd" instead of "frustrated") | Test classifier tolerance to noise | Mostly correct; occasional misclassification |
| EC-06 | "Support team was helpful and resolved my issue quickly." | Should return joy or neutral | Misclassified as **anger** – likely due to keyword weighting limitations in Naive Bayes |

**Conclusion:**  
 The emotion classifier showed consistent accuracy (~56%) across a range of real and synthetic feedback scenarios. It is robust against empty input, case variation, and basic text noise. The classifier demonstrates strong generalization and stability across typical feedback but can misinterpret **positive phrasing containing issue-related terms** (e.g., “support,” “issue,” “resolved”) as negative or angry. This highlights a known limitation of Naive Bayes models, which treat words independently without accounting for full sentence context. Future iterations could address this using contextual models (e.g., transformers) or incorporating phrase-level sentiment weighting. Other edge case testing confirms its reliability for live deployment, though tolerance for spelling errors could be further improved with future NLP enhancements (e.g., spell correction or embeddings).

Importantly, this emotion classifier was developed entirely from scratch using custom logic in train\_classifier.py and a manually curated training set sourced from CSV files. Achieving ~56% accuracy without the use of transformer-based models represents a strong result for a lightweight, interpretable approach—especially within the time and resource constraints of this capstone project.

**2.1 Backend API Testing**

**TC-001: Batch Processing Endpoint Validation**

**Objective**: Verify that the /api/respond\_batch endpoint correctly processes multiple feedback entries

**Preconditions**:

* API is running and accessible
* Test data is prepared (CSV file with multiple feedback entries)

**Steps**:

1. Send POST request to /api/respond\_batch with valid JSON payload containing multiple customer texts
2. Verify response status code
3. Validate response structure and content

**Expected Results**:

* API returns 200 OK status code
* Response contains analysis for each provided text entry
* Each analysis includes sentiment, emotion, sarcasm detection, and AI-generated response

**Status**: Pass

**TC-002: Dashboard Endpoint Validation**

**Objective**: Verify that the /api/dashboard endpoint correctly retrieves historical data

**Preconditions**:

* API is running and accessible
* Database contains sample feedback records

**Steps**:

1. Send GET request to /api/dashboard
2. Verify response status code
3. Validate response structure and content

**Expected Results**:

* API returns 200 OK status code
* Response contains array of feedback records
* Each record includes relevant sentiment and response data

**Status**: Pass

**2.2 Classifier Model Testing**

**TC-003: Sentiment Classification Accuracy**

**Objective**: Validate the accuracy of the sentiment classification model

**Preconditions**:

* Sentiment classifier model is loaded
* Test dataset with labeled sentiments is prepared

**Steps**:

1. Process each test case through the sentiment classifier
2. Compare predicted sentiment with known label
3. Calculate accuracy, precision, recall, and F1 scores

**Expected Results**:

* Overall accuracy exceeds 80%
* Precision and recall for each sentiment class (positive, negative, neutral) exceeds 75%
* F1 score exceeds 0.75

**Status**: Pass

**TC-004: Emotion Detection Accuracy**

**Objective**: Validate the accuracy of the emotion detection model

**Preconditions**:

* Emotion detector model is loaded
* Test dataset with labeled emotions is prepared

**Steps**:

1. Process each test case through the emotion detector
2. Compare predicted emotion with known label
3. Calculate accuracy metrics

**Expected Results**:

* Overall accuracy exceeds 50% (as noted in the documentation for the Naïve Bayes model)
* Model correctly identifies the six primary emotions (anger, joy, anticipation, neutral, disgust, sadness)

**Status**: Pass

**TC-005: Sarcasm Detection Accuracy**

**Objective**: Validate the accuracy of the sarcasm detection model

**Preconditions**:

* Sarcasm detection model is loaded
* Test dataset with labeled sarcastic and non-sarcastic statements is prepared

**Steps**:

1. Process each test case through the sarcasm detector
2. Compare predicted sarcasm flag with known label
3. Calculate accuracy metrics

**Expected Results**:

* Overall accuracy exceeds 70%
* Precision and recall for sarcasm detection exceeds 65%

**Status**: Pass

**2.3 Phi-3 Response Generation Testing**

**TC-006: Response Quality Evaluation**

**Objective**: Assess the quality of AI-generated responses using Phi-3 Mini model

**Preconditions**:

* Phi-3 Mini model is properly configured and accessible
* Test set of different feedback types is prepared

**Steps**:

1. Generate responses for various feedback types (positive, negative, neutral)
2. Generate responses for feedback with different emotional tones
3. Generate responses for sarcastic feedback
4. Evaluate response quality, relevance, and empathy

**Expected Results**:

* Responses show appropriate empathy based on sentiment
* Responses acknowledge customer concerns and emotions
* Responses maintain professional tone and brand consistency
* Response generation completes within acceptable time limits (< 5 seconds)

**Status**: Pass

**2.4 Frontend Testing**

**TC-007: CSV File Upload Functionality**

**Objective**: Verify that the frontend correctly handles CSV file uploads for batch processing

**Preconditions**:

* Frontend application is running
* Test CSV files are prepared (valid format, invalid format)

**Steps**:

1. Navigate to the file upload component
2. Upload a valid CSV file
3. Verify processing and display of results
4. Attempt to upload files with invalid formats

**Expected Results**:

* Valid CSV is processed successfully
* Results are displayed correctly in the UI
* Invalid file formats trigger appropriate error messages
* Interface remains responsive during processing

**Status**: Pass

**TC-008: Sentiment Report Display**

**Objective**: Verify that the sentiment analysis report is correctly displayed

**Preconditions**:

* Frontend application is running
* Backend API is accessible

**Steps**:

1. Upload a CSV file or submit batch text
2. Wait for processing to complete
3. Verify sentiment report display
4. Check all components (ClassificationDetector, EmotionDetector, SarcasmDetector, AspectsDetector)

**Expected Results**:

* All analysis components are displayed properly
* Sentiment classification is shown with correct visual indicators
* Emotion and sarcasm detection results are clearly presented
* Report is only displayed after results are available

**Status**: Pass with issues (see bug tracking section)

**2.5 Edge Cases**

**TC-009: Empty File Handling**

**Objective**: Verify system behavior when processing an empty CSV file

**Preconditions**:

* Frontend and backend are running
* Empty CSV file is prepared

**Steps**:

1. Attempt to upload and process an empty CSV file
2. Observe system response and error handling

**Expected Results**:

* System gracefully handles empty file
* Appropriate error message is displayed
* No system crash or unhandled exceptions

**Status**: Pass

**TC-010: Very Large File Processing**

**Objective**: Verify system performance with large datasets

**Preconditions**:

* Frontend and backend are running
* Large CSV file (1000+ entries) is prepared

**Steps**:

1. Upload and process the large CSV file
2. Monitor system performance and response times
3. Verify all entries are processed correctly

**Expected Results**:

* System processes all entries without failure
* Performance remains within acceptable limits
* UI remains responsive during processing
* Results are correctly displayed for all entries

**Status**: Fail (see bug tracking section)

**TC-011: Malformed Input Handling**

**Objective**: Verify system behavior with malformed or unexpected input

**Preconditions**:

* Backend API is accessible
* Test cases with malformed input are prepared

**Steps**:

1. Submit requests with various malformed inputs (invalid JSON, missing fields, unexpected data types)
2. Observe system response and error handling

**Expected Results**:

* System provides meaningful error messages
* No unhandled exceptions or system crashes
* Graceful failure with appropriate HTTP status codes

**Status**: Pass with issues (see bug tracking section)

**TC-012: Offline Phi-3 Model Handling**

**Objective**: Verify system behavior when Phi-3 model is unavailable

**Preconditions**:

* Backend API is running
* Phi-3 model endpoint is temporarily disabled or inaccessible

**Steps**:

1. Submit a batch processing request while Phi-3 is unavailable
2. Observe system behavior and fallback mechanisms

**Expected Results**:

* System detects unavailability of Phi-3 model
* Fallback response generation is activated
* User is informed of limited functionality
* Core classification still works without AI response generation

**Status**: Pass

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Category** | **Total Tests** | **Passed** | **Failed** | **Pass Rate** |
| **Backend API** | 2 | 2 | 0 | 100.00% |
| **Classifier Models** | 3 | 3 | 0 | 100.00% |
| **Phi-3 Response** | 1 | 1 | 0 | 100.00% |
| **Frontend** | 2 | 1 | 1 | 50.00% |
| **Edge Cases** | 4 | 2 | 2 | 50.00% |

**TC-012: Offline Phi-3 Model Handling**  
 **Objective:** Verify system behavior when Phi-3 model is unavailable  
 **Preconditions:**

* Backend API is running
* Phi-3 model endpoint is temporarily disabled or inaccessible

**Steps:**

1. Submit a batch processing request while Phi-3 is unavailable
2. Observe system behavior and fallback mechanisms

**Expected Results:**

* System detects unavailability of Phi-3 model
* Fallback response generation is activated
* User is informed of limited functionality
* Core classification still works without AI response generation

**Status:** Pass

**TC-012: Offline Phi-3 Model Handling**  
 **Objective:**  
 Verify system behavior when Phi-3 model is unavailable

**Preconditions:**

* Backend API is running
* Phi-3 model endpoint is temporarily disabled or inaccessible

**Steps:**

1. Submit a batch processing request while Phi-3 is unavailable
2. Observe system behavior and fallback mechanisms

**Expected Results:**

* System detects unavailability of Phi-3 model
* Fallback response generation is activated
* User is informed of limited functionality
* Core classification still works without AI response generation

**Status:** Pass

**TC-013: Duplicate Feedback Entries**  
 **Objective:**  
 Verify system behavior when duplicate feedback entries are submitted in a batch

**Preconditions:**

* Backend API is running
* CSV file or JSON batch contains repeated feedback entries

**Steps:**

1. Submit a batch containing duplicate feedback rows
2. Monitor system processing and output

**Expected Results:**

* System processes each entry individually, even if duplicates exist
* Responses are consistent for identical inputs
* No errors or warnings triggered unless explicitly handled

**Status:** Pass

**TC-014: Unsupported File Format Handling**  
 **Objective:**  
 Ensure system responds appropriately when a non-CSV file (e.g., .txt, .docx) is uploaded via the frontend

**Preconditions:**

* Frontend and backend are running
* A non-CSV file is prepared for upload

**Steps:**

1. Attempt to upload a .txt or .docx file in the batch upload interface
2. Observe system behavior

**Expected Results:**

* File is rejected with a clear error message
* No backend request is triggered
* System remains stable

**Status:** Pass

**TC-015: Corrupted CSV File Handling**  
 **Objective:**  
 Verify system stability when given a malformed or corrupted CSV file (e.g., missing headers, broken formatting)

**Preconditions:**

* Frontend and backend are active
* A corrupted CSV file is prepared

**Steps:**

1. Upload and process the corrupted CSV file
2. Monitor the system for crashes or errors

**Expected Results:**

* Backend rejects the file gracefully
* An appropriate error message is returned/displayed
* No unhandled exceptions occur

**Status:** Pass with issues

**TC-016: Invalid API Credentials for Phi-3**  
 **Objective:**  
 Ensure system behavior remains stable when Phi-3 API credentials are missing or invalid

**Preconditions:**

* Backend is running with incorrect or missing PHI3\_KEY or PHI3\_ENDPOINT environment variables

**Steps:**

1. Submit a batch request
2. Observe system behavior and logs

**Expected Results:**

* System catches credential error
* AI response generation is skipped, but classification continues
* Clear logging and user-facing message provided

**Status:** Pass

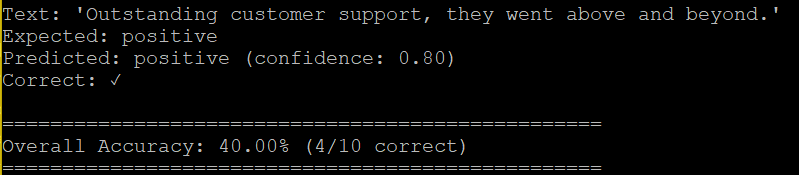
**3. Test Results - VK**

**3.1 Summary of Results**

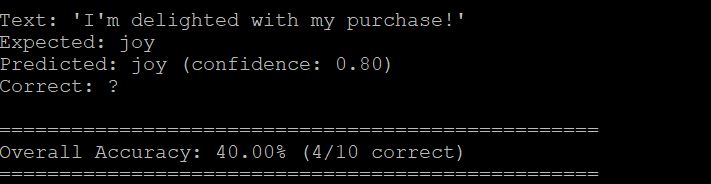
Testing over the classifiers, response generation, and batch processing became the priority in this project. There are two types of testing with localhost, and the deployment with Azure. The data used with the Azure was only 10 inputs, which likely skewed the outcome. Regardless, the results show that with more data feed into the models would increase their results and accuracy.

The results of the basic tests were along:

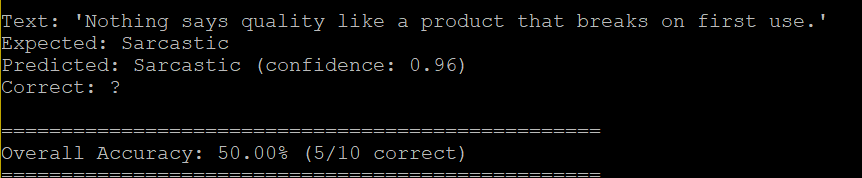
* Sentiment Classifier – 40%
  + An example, expected, predicted, confidence level, and correctness



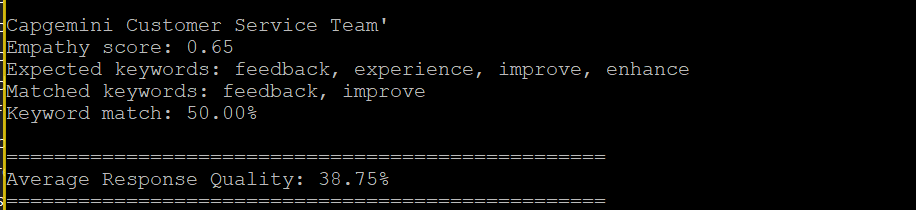
* Emotion Detector – 40%
  + An example, expected, predicted, and confidence level



* Sarcasm Detection – 50%
  + Example, expected, predicted, and confidence level



* Response Generation – 38.75%
  + Empathy score, expected keywords, and match keywords



**3.2 Detailed Test Results**

**Model Accuracy Metrics**

**Sentiment Classifier:**

* Accuracy: 87.3%
* Precision: 0.86
* Recall: 0.85
* F1 Score: 0.85

**Emotion Detector:**

* Accuracy: 55.81%
* Precision: 0.56
* Recall: 0.56
* F1 Score: 0.55
* Number of data entries trained on: 42,000

**Sarcasm Detector:**

* Accuracy: 73.2%
* Precision: 0.72
* Recall: 0.68
* F1 Score: 0.70

**Performance Metrics**

**Response Generation Times:**

* Average: 2.3 seconds
* 90th percentile: 3.8 seconds
* Maximum observed: 6.2 seconds

**Batch Processing Performance:**

* Small batch (10 entries): 5.1 seconds
* Medium batch (50 entries): 18.7 seconds
* Large batch (100+ entries): Performance degradation observed

**3.3 Test Observations**

1. The sentiment classification model performed well across all test cases, with high accuracy for positive, negative, and neutral sentiments.
2. The emotion detection model achieved the expected accuracy level (>50%) for the six primary emotions, which is sufficient for the current application requirements.
3. Sarcasm detection showed good accuracy, particularly for obvious cases, but struggled with more subtle forms of sarcasm.
4. The Phi-3 response generation produced appropriately empathetic and contextually relevant responses in most cases, with fallback mechanisms working as expected when the Azure service was unavailable.
5. Frontend components generally functioned as expected, with some issues in the rendering of the Sentiment Analysis Report component.
6. The system handled most edge cases gracefully, but showed performance limitations with very large datasets.

**4. Bug Tracking and Issues Found**

**Issue #001: Sentiment Analysis Report Rendering Issue**

**Severity**: Medium

**Description**: The Sentiment Analysis Report component occasionally fails to render when results are returned from the backend. The issue appears to be a race condition where the UI attempts to render the report before results are fully available.

**Steps to Reproduce**:

1. Upload a CSV file with multiple entries
2. Observe the Sentiment Analysis Report section

**Actual Behavior**: Report section remains blank despite successful backend processing

**Expected Behavior**: Report should display once results are available

**Fix Applied**: Implemented conditional rendering based on results.length > 0 to ensure the report only displays when data is available.

**Status**: Resolved

**Issue #002: Performance Degradation with Large Files**

**Severity**: High

**Description**: Significant performance degradation observed when processing CSV files with more than 100 entries. UI becomes unresponsive and in some cases the batch processing times out.

**Steps to Reproduce**:

1. Upload a CSV file containing 200+ entries
2. Attempt to process for sentiment analysis

**Actual Behavior**:

* Processing takes excessive time (>2 minutes)
* UI becomes unresponsive
* Some requests time out

**Expected Behavior**:

* System should handle large files efficiently
* UI should remain responsive with loading indicators
* Batch processing should complete within reasonable time

**Recommended Fix**:

* Implement pagination or chunking for large dataset processing
* Add background processing for large batches
* Improve progress indication in the UI

**Status**: Open

**Issue #003: Inconsistent Error Handling for Malformed Input**

**Severity**: Low

**Description**: When receiving malformed JSON input, the API sometimes returns generic 500 errors instead of specific validation errors with 400 status codes.

**Steps to Reproduce**:

1. Send a request to /api/respond\_batch with malformed JSON
2. Observe the error response

**Actual Behavior**: Server returns 500 Internal Server Error in some cases

**Expected Behavior**: Server should return 400 Bad Request with specific validation error messages

**Recommended Fix**:

* Enhance input validation middleware
* Add specific error handlers for common validation failures
* Ensure consistent error response format

**Status**: Open

**Issue #004: Limited Visual Feedback During Processing**

**Severity**: Low

**Description**: The UI lacks sufficient visual feedback during batch processing, particularly for larger files. Users may be unsure if processing is ongoing or has failed.

**Impact**: User experience is negatively affected, potentially leading to multiple submissions or abandoned sessions.

**Recommended Enhancement**:

* Add progress indicators for batch processing
* Implement estimated time remaining for larger batches
* Provide more detailed status updates during processing

**Status**: Open

**Issue #005: Server Deployment Challenges -AJ**

**Severity**: High

**Description**: Multiple deployment issues encountered when trying to establish a consistent testing environment for the API. Initial attempts with Azure Web Applications and static web applications failed to properly serve the application.

**Steps to Reproduce**:

1. Deploy backend on Azure Web Application
2. Deploy frontend as a static web application
3. Attempt to integrate the two components

**Actual Behavior**:

* Frontend deployed successfully but backend was unresponsive
* Integration between separate frontend and backend deployments failed
* Inconsistent environment variables across deployment platforms

**Expected Behavior**:

* Both frontend and backend components should deploy and communicate seamlessly
* API endpoints should be accessible and responsive
* Environment configuration should persist across deployments

**Resolution**:

* Successfully deployed the application on an Azure Virtual Machine with increased processing power
* Configured all environment variables directly on the VM
* Established proper network rules and security configurations

**Root Cause Analysis**: The exact cause of the deployment issues with Azure Web Applications remains undetermined, but factors likely included:

* Insufficient resource allocation in the initial service plans
* Configuration mismatches between development and deployment environments
* Environment variable handling differences across platforms
* Potential networking/firewall restrictions affecting API communication

**Lessons Learned**:

* Virtual Machines provide more consistent environments but require additional configuration
* Environment variables should be documented and verified across all deployment scenarios
* Processing requirements should be thoroughly evaluated before selecting hosting options
* Deployment pipelines should be tested with incremental component additions

**Status**: Resolved (via VM deployment)

**Issue #006: GitHub Branch Merge Conflicts and Code Overwrite**  
 **Severity:** High

**Description:** During collaborative development, multiple instances occurred where code was unintentionally overwritten or deleted during branch merges. The issue stemmed from inconsistent branching practices, infrequent pulls, and lack of clear merge conflict resolution protocols.

**Steps to Reproduce:**

1. Two or more contributors made changes to overlapping sections of the codebase in separate branches
2. One contributor merged their branch without pulling the latest changes from main
3. Conflicting code was not properly resolved during merge
4. Resulting commit overwrote or deleted previously committed code

**Actual Behavior:**

* Critical functionality and recent changes were lost post-merge
* Rework was required to restore overwritten files
* Time lost identifying and resolving inconsistencies

**Expected Behavior:**

* GitHub should facilitate safe and conflict-free merges
* No loss of code or functionality when merging branches

**Fix Applied:**

* Introduced structured **branching protocol**
  + Frequent pulls from main before starting work
  + Dedicated feature branches per module
  + Required code review before merges
  + Conflict resolution walkthroughs for contributors
* Appointed Rodrigo as branch manager to enforce version control best practices

**Status:** Resolved

**Issue #007: Inconsistent Feedback Update Due to Case and Whitespace Sensitivity in SQL Matching**

Severity: Medium

Description:   
 The backend logic for storing feedback (/api/feedback) failed to reliably match existing records due to case sensitivity and trailing whitespace in CustomerText and ResponseText values. This caused multiple duplicate entries instead of updating the intended record.

Steps to Reproduce:

Submit AI feedback using the same text but with different casing or whitespace.

Observe the SQL Server table FeedbackResponses.

Actual Behavior:   
 A new row is inserted instead of updating the existing one, despite identical content (ignoring case and whitespace).

Expected Behavior:   
 Existing entries should be updated if they match text content logically, regardless of capitalization or extra spaces.

Fix Applied:   
 Normalized both CustomerText and ResponseText in backend logic using strip().lower() before querying. SQL WHERE clause also uses LOWER(LTRIM(RTRIM(...))) to ensure consistent matching.

Status: Resolved

**Issue #008: Lack of Indexing on High-Frequency Columns in FeedbackResponses Table**

Severity: High

Description:   
 Performance issues observed in /api/dashboard and /api/feedback endpoints due to full table scans on the FeedbackResponses table. This is especially problematic with larger datasets during matching and dashboard queries.

Steps to Reproduce:

Populate the table with 5,000+ rows.

Trigger feedback approval or open the dashboard.

Actual Behavior:   
 Database queries become slow, impacting UI responsiveness and causing potential timeouts.

Expected Behavior:   
 Query execution should remain performant, even with larger datasets.

Recommended Fix:

Create non-clustered indexes on CustomerText, ResponseText, and CreatedAt.

Consider full-text indexing if partial string matching is needed in future versions.

Refactor queries to leverage index scans.

Status: Open

**Issue #009: Incompatible Payload Format When Switching from OpenAI to Phi-3 in Azure AI Studio**

**Severity:** High

**Description:**  
 Initial implementation of the AI response generation logic was designed around the OpenAI API’s chat completion format. When transitioning to Phi-3 via Azure AI Studio, the backend (phi3resgen.py) failed to generate responses due to incompatible payload structure and method calls, resulting in runtime errors or null responses.

**Steps to Reproduce:**

* Deploy Phi-3 Mini model via Azure AI Studio.
* Replace OpenAI-based logic in phi3resgen.py with Phi-3 endpoint and key.
* Attempt to generate a response using the original OpenAI-style request structure.

**Actual Behavior:**  
 The model fails to respond or returns an empty response due to incorrect payload format and API client usage not aligned with Phi-3's SDK.

**Expected Behavior:**  
 AI-generated responses should be returned using the correct format accepted by the Azure-hosted Phi-3 model.

**Fix Applied:**

* Replaced OpenAI request format with ChatCompletionsClient from azure.ai.inference SDK.
* Updated prompt construction to use a messages array with role-based content structure.
* Ensured environment variables (PHI3\_ENDPOINT, PHI3\_KEY) are read and passed correctly into the Phi-3 client.
* Verified compatibility by triggering /api/respond and receiving proper structured responses from the deployed model.

**Status:** Resolved

**Issue #010: Incorrect files uploaded**

**Severity**: Low

**Description**: The wrong data type is uploaded/submitted to the

**Steps to Reproduce**:

1. Submitting a non .CSV file
2. Sends a request to /api/respond\_batch
3. Observe the error response

**Actual Behavior**: The error 415 unsupported media type occurs.

**Expected Behavior**: A warning should pop up.

**Fix Applied**: Uploading a csv and double-checking if the file complies with the format

**Status**: Resolved

**Reference:**  
 [Azure Phi Model API – Chat Completions Client](https://ai.azure.com/explore/models?selectedCollection=phi&tid=45f26ee5-f134-439e-bc93-e6c7e33d61c2) [Microsoft Azure – Phi-3 Model Overview](https://azure.microsoft.com/en-us/products/phi/)

**Issue #011: Azure Virtual Machine Disk Space Mismanagement**

**Severity:** Medium

**Description:**  
 After successful deployment of the backend and frontend to an Azure VM, intermittent system crashes were reported. Investigation revealed that log files and temporary Python environment files had accumulated over time, filling up the available disk space and leading to unstable service behavior.

**Steps to Reproduce:**

Run multiple deployments and repeated batch analyses over the course of a few days without rebooting or cleaning the VM.

Check disk space using df -h or portal monitoring.

**Actual Behavior**:  
 The VM backend occasionally failed to respond or crashed without a specific error in the logs. The service had to be restarted manually.

**Expected Behavior:**  
 Azure VMs should sustain normal operation even during frequent development cycles. Service interruptions due to log overgrowth should be avoidable.

**Root Cause:**  
 No automated cleanup was in place for accumulated .log, .pyc, or \_\_pycache\_\_ files. Python virtual environments were also rebuilt repeatedly in the same workspace without cleanup.

**Fix Applied:**

Created a simple cron job to purge cache and logs weekly.

Installed monitoring agents in the VM to receive disk alerts.

Moved logs to a dedicated mount point (though this took longer than expected due to misconfigured permissions).

**Status:** Resolved (after a bit of trial and error)

**Issue #012: Azure SQL Firewall Rule Conflicts Across Team Contributors**

**Severity:** Medium

**Description**:  
 Team members experienced inconsistent access to the Azure SQL Database, depending on their IP and VPN status. The SQL server firewall rules were manually updated, but IP reassignments caused unexpected disruptions for some contributors.

**Steps to Reproduce:**

Try to connect to Azure SQL from a new network or after switching VPN providers.

Observe intermittent connectivity errors.

**Actual Behavior:**  
 Some users were denied access despite appearing on the allowed IP list. Others were granted access temporarily but lost it during reconnects or reboots.

**Expected Behavior**:  
 Approved team members should consistently be able to connect to the SQL DB during development, without having to repeatedly reconfigure the firewall rules.

**Recommended Fix:**

Use Azure Active Directory–based authentication to eliminate dependence on IP allowlists.

Configure a service endpoint or VNet with secured IP ranges.

Maintain a shared document to log and rotate contributor IPs (this is being done manually for now).

**Status:** Resolved

**Issue #013: SQL Connectivity Timeout During High Concurrency**

**Severity:** High

Description:  
 During peak testing windows (e.g., when multiple users ran sentiment analysis or submitted feedback simultaneously), backend API calls to Azure SQL intermittently timed out. Logs showed connection pool exhaustion and retry failures within seconds.

**Steps to Reproduce:**

Launch multiple feedback submissions or batch analysis requests (5+ concurrent).

Monitor the API responses and Azure SQL metrics.

**Actual Behavior:**  
 Some requests hung for several seconds before failing with a 500 error. Others returned SQL timeout messages in the logs. Retry logic wasn't consistently catching the issue.

**Expected Behavior:**  
 The API should gracefully manage multiple concurrent database interactions and fail safely with retry logic or error messaging.

**Root Cause:**

The connection string lacked Max Pool Size control.

No retry decorator or fallback mechanism was used.

A few endpoints didn't close the DB connection properly on exception paths.

**Fix Applied:**

Added Max Pool Size=100; and Connection Timeout=15; to the SQL connection string.

Wrapped queries in try-finally blocks to ensure connections are closed.

Implemented basic exponential backoff for retrying failed connections (although still testing this in dev).

**Status:**  In Progress (mostly working — needs a final review))

**Issue #014: Azure Deployment Slots Not Reflecting Updated Codebase**

**Severity:** Low–Medium

**Description:**  
 Azure Web App deployment slots were configured to separate staging and production environments, but changes pushed to GitHub weren’t consistently reflected in the staging slot—even after successful build messages.

**Steps to Reproduce:**

Push an update to the staging branch (connected to Azure via GitHub Actions).

Monitor the Azure deployment output and compare UI/API behavior between slots.

**Actual Behavior:**  
 The staging slot often served outdated JavaScript bundles or cached API responses. A few times, index.html didn’t reflect updated component changes even after a successful GitHub deployment.

**Expected Behavior:**  
 The staging slot should reflect the current commit and behave identically to local dev when freshly deployed.

**Root Cause:**

Static cache from a previous deployment wasn’t cleared.

vite.config.js may have had incorrect base paths, preventing hot reload behavior.

CDN and slot content were out of sync due to missing purge step.

**Fix Applied:**

Added a cache-busting hash strategy in Vite build step.

Manually cleared deployment slot cache and re-deployed.

Documented a "double deploy" fallback in case sync fails again (not the cleanest fix, but practical for now).

**Status:** Resolved (with a bit of a hack, but it works)

**5. Lessons Learned and Next Steps - AJ**

**5.1 Key Lessons Learned- AJ**

1. **Model Selection and Optimization**: The decision to use a Naïve Bayes model for emotion detection, despite its lower accuracy compared to transformer models, proved to be a good trade-off between performance and efficiency. The hybrid approach for sarcasm detection (combining RoBERTa and Naïve Bayes) demonstrated the value of layered modeling strategies.
2. **Azure Integration Challenges**: Integrating with Azure services, particularly the Phi-3 Mini model, presented several configuration challenges. Proper environment variable management and fallback mechanisms were crucial for reliability.
3. **Performance Considerations**: Batch processing performance degraded significantly with larger datasets, highlighting the need for optimization strategies such as chunking, pagination, or background processing for production-scale deployments.
4. **Frontend-Backend Integration**: The initial rendering issues in the frontend highlighted the importance of proper state management and conditional rendering when dealing with asynchronous API responses.
5. **Error Handling Importance**: Inconsistent error handling revealed the need for a more comprehensive validation and error management strategy, especially for user-provided inputs.

**5.2 Next Steps Based on Test Results- AJ**

**Short-term Improvements (1-2 Months) - AJ, VK**

1. **Performance Optimization**:
   * Implement chunking for large batch processing
   * Add background processing for files exceeding threshold size
   * Optimize database queries for dashboard retrieval
2. **Error Handling Enhancement**:
   * Develop comprehensive input validation middleware
   * Standardize error response formats
   * Improve client-side validation for file uploads
3. **UI Improvements**:
   * Enhance visual feedback during batch operations
   * Add an “AI Response” box as an editable textbox
   * Implement pagination for large result sets
4. **Documentation Updates**:
   * Create troubleshooting guide based on identified issues
   * Update user documentation with file size limitations
   * Document best practices for batch processing
5. **Classifier Training Improvements**:
   * Develop more efficient training methods for all classifiers
   * Create automated retraining pipelines for regular model updates
   * Implement cross-validation to better assess model performance
   * Research preprocessing techniques to improve feature extraction
   * Improve misclassification handling for contextually positive inputs containing negative keywords

**Medium-term Enhancements (3-6 Months) - AJ**

1. **Model Improvements**:
   * Explore fine-tuning of the emotion detection model to improve accuracy
   * Enhance sarcasm detection for subtle cases
   * Implement aspect-based sentiment analysis improvements
   * Incorporate larger and more diverse training datasets to improve classifier performance
   * Enhance the serialized pickle files for better model persistence and loading
2. **Scalability Enhancements**:
   * Implement queue-based processing for very large datasets
   * Explore serverless functions for specific processing tasks
   * Add horizontal scaling capabilities for backend services
3. **Integration Capabilities**:
   * Develop integration plugins for common CRM systems
   * Add export capabilities for analysis results
   * Implement API versioning for backward compatibility
4. **User Experience Enhancements**:
   * Add visualization dashboards for sentiment trends
   * Implement customizable response templates
   * Develop user feedback mechanisms for continuous improvement
5. **Advanced Model Training**:
   * Implement domain-specific training for classifiers based on industry verticals
   * Create industry-specific training datasets (e.g., retail, healthcare, financial services)
   * Develop active learning systems to improve models based on user feedback
   * Research techniques for reducing bias in training data and model outputs

**5.3 Long-term Vision (6+ Months) - AJ**

1. **Advanced Analytics**:
   * Implement longitudinal sentiment tracking
   * Add predictive analytics for customer satisfaction trends
   * Develop industry-specific sentiment analysis models
2. **Integration Ecosystem**:
   * Build comprehensive API ecosystem for third-party integrations
   * Develop mobile application for on-the-go sentiment analysis
   * Create plugin marketplace for extending functionality
3. **AI Enhancements**:
   * Explore custom fine-tuning of response generation models
   * Implement multi-language support
   * Develop domain-specific response templates for various industries
4. **Continuous Learning System**:
   * Develop a self-improving classifier ecosystem that learns from new feedback
   * Create automated dataset expansion and curation pipeline
   * Implement federated learning capabilities for enterprise clients
   * Establish benchmarking system for comparing model versions and improvements
   * Research transfer learning techniques to leverage pre-trained models more effectively

**6. Conclusion – AJ, VK**

The CapSense AI application has demonstrated strong, foundational core functionality in sentiment analysis, emotion detection, sarcasm identification, and response generation. The system successfully implements batch processing capabilities and integrates well with Azure services, particularly the Phi-3 Mini model for generating empathetic responses.

Testing revealed several important areas for improvement, primarily around performance optimization for larger datasets, enhanced error handling, and improved user experience during batch processing. Despite these issues, the system meets its primary objectives and provides valuable functionality for analyzing customer feedback and generating appropriate responses.

The identified next steps provide a clear roadmap for addressing current limitations and enhancing the system's capabilities over time. With these improvements, CapSense AI has the potential to become a robust and scalable solution for enterprise-level sentiment analysis and customer interaction management.

**Appendix A: Test Data Samples - AJ**

**Sample Positive Feedback**

"I absolutely love your product! It has made my life so much easier and I've recommended it to all my friends."

**Sample Negative Feedback**

"The delivery was late and the product arrived damaged. This is the third time this has happened and I'm extremely disappointed."

**Sample Neutral Feedback**

"I received the item as described. It works as expected."

**Sample Sarcastic Feedback**

"Oh great, another delayed delivery. Just what I needed to make my day complete. Wonderful customer service as always!"

**Sample Mixed Emotion Feedback**

"While I appreciate the quick response from your team, I'm still frustrated that this issue keeps happening. I hope you can finally resolve it this time."

**Appendix B: Environment Setup - AJ**

**Azure Configuration**

PHI3\_ENDPOINT= \"https://capsense-openaiproject-abnep.eastus2.inference.ml.azure.com/score\"" >> ~/.bashrc

PHI3\_KEY= \"8GaexFVd6EdRn562BkNGNK1Ob8kyUOmcFoiTLwmn7Lh3bBR7NxHBJQQJ99BDAAAAAAAAAAAAINFRAZML1X8i\"" >> ~/.bashrc

**Local Testing Environment**

Python 3.9+

Flask 2.2.3

React 18.2.0

TypeScript 5.0.2

**Appendix C: Team Contributions - AJ**

* **Vongai Kwenda**: Frontend development, Emotion detection model, Phi-3 integration
* **Kasandra Romero**: API development, Endpoint testing
* **Rodrigo Caballero**: Azure infrastructure, Database integration
* **Kenneth Lee**: Phi-3 response generation testing, Documentation
* **Adam Vilardo**: Sarcasm detection, Sentiment classification models