Project
Final
Report

amazon

UTD

Intelligent Service Requests Resolver Framework with Amazon

Omer Reshef, Zach Jackson, Ethan Hays, Mustafa Khan, Austin Hucabee, Olagoke Adereti

Executive Summary

This document aims to cover all aspects of the project management plan, requirement phase, architecture, design, development and test plan. Included is essential information about software design and implementation of requirements. Using class diagrams and sequence diagrams the software architecture is made evident. This construction will lend itself to incremental development and testing. Use cases and user stories are provided to illustrate the application and flow of events when using the resolver framework. The functional requirements of each service utilized by the framework are detailed, followed by the non-functional requirements. The Pipe & Filter architecture fits the needs of this framework best, due to the nature of connecting individual microservices through a pipeline. Alternative architectures were considered for this project, but they do not fit the requirements of the framework as closely as a Pipe & Filter architecture. The framework makes extensive use of AWS software services, all hosted on AWS servers. The intelligent service request resolver framework aims to resolve requests from users, analyze their sentences, and generate a response based on AI techniques, as well as provide a feedback loop to increase AI precision, as described more below.

Table of Contents

- **Executive Summary**
- **Introduction**
- **Project Management Plan**
- **Requirements Specifications**
- **Architecture**
- **Design**
- **Testing Plan**
- **References**

1. Introduction

1.1. Purpose and Scope

This document illustrates the specifications, architecture, planning, development, and testing entailed in building this project. The purpose of our product is to leverage machine learning frameworks in order to generate the best possible

response to a request sent in the form of a question by the client/user. The scope of this product includes anyone using AWS services internally and externally. The product is intended for use by internal and external clients/users.

1.2. Product Overview (including capabilities, scenarios for using the product, etc.)

This product is designed to receive requests from the user using the AWS API Gateway in the form of query parameters and respond appropriately using microservices and a serverless backend. After receiving the request, the pipeline activates the appropriate microservices to generate a response. These autonomous microservices have different responsibilities that work as a unit to identify the question and owner, check the database for previous instances, and use AI frameworks to identify intent and sentiment. The pipeline, using all the response data from the microservices, then sends the response to the client/user.

1.3. Structure of the Document

This document structured in the following way -

- Intro
- Project Management Plan
- Architecture
- Design
- Testing Plan

1.4. Terms, Acronyms, and Abbreviations

AWS - Amazon Web Services.
API - Application Programming Interface.
Scrum Leader/Master - A person who leads the update meeting.
AI - Artificial Intelligence.
ML - Machine Learning.
UTD - University of Texas at Dallas.

2. Project Management Plan

2.1 Project Organization

- Project Organization:
    - External Stakeholders
        - Eric Wong
            - ewong@utdallas.edu
        - Dongcheng Li
            - dxl170030@utdallas.edu

- Product Owners
  - Gaurav Acharya
    - achagaur@amazon.com
  - Mihir Desai
    - desamihi@amazon.com
- SCRUM Master
  - Omer Reshef
    - Oxr17001@utdallas.edu
- Developers
  - Omer Reshef (Team Leader / Project Manager)
    - Oxr17001@utdallas.edu
  - Zach Jackson
    - ZJJ180000@utdallas.edu
  - Ethan Hays
    - ech180000@utdallas.edu
  - Mustafa Khan
    - mnk170030@utdallas.edu
  - Austin Hucabee
    - austin.hucabee@utdallas.edu
  - Olagoke Adereti
    - Osa180000@utdallas.edu

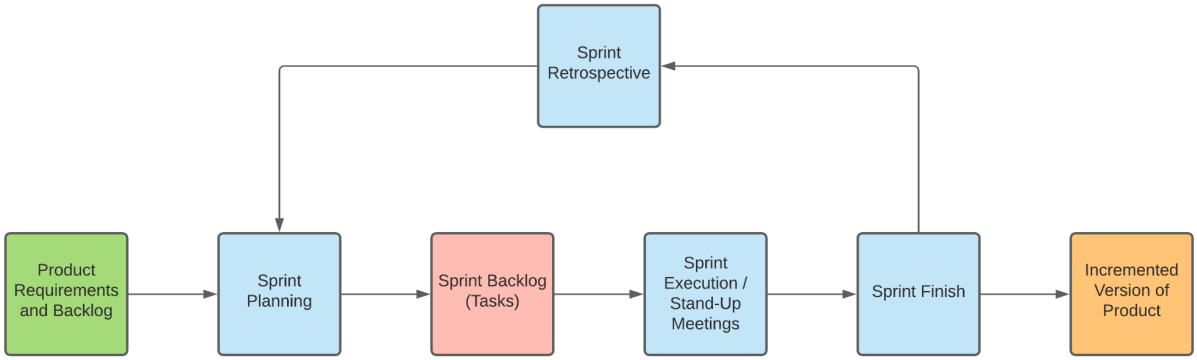
2.2 Lifecycle Model Used

The life-cycle model that we chose to implement is a SCRUM model. The project will be built iteratively, in the form of 2-week long sprint cycles. The requirements for each sprint cycle will be decided conjointly between the product owners and the developers. This will take place in a sprint planning meeting every 2 weeks. The development team will meet 2-3 days a week for a stand-up meeting. This is a meeting in which each developer shares their progress on the tasks in their backlogs since the last stand-up meeting, and what they intend to work on until the next meeting.

- SCRUM Life-cycle
  - 2-week sprint cycles
  - 2-3 stand-up meetings per week
  - 1 sprint retrospective every two weeks
  - first stand-up of a new sprint = sprint planning
  - Components:
    - Product Backlog
    - Sprint Backlog

        - Product versioning

Figure 4.1 - Lifecycle Model.



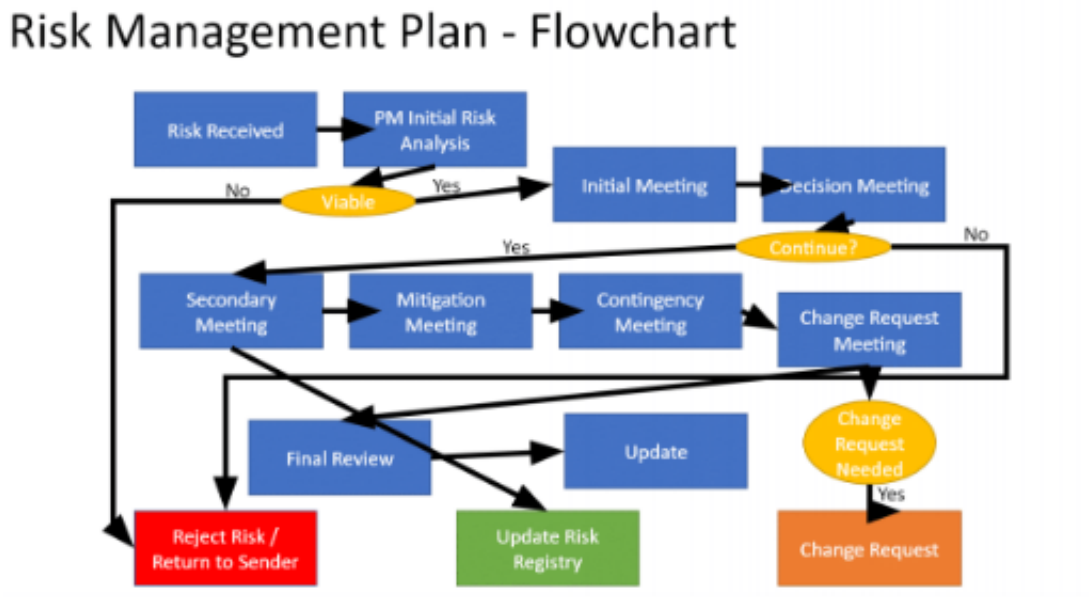
2.3 Risk Analysis

Possible Risks:

- AWS downages
    - Unlikely to occur
        - We are unable to prevent downages from occurring
- Excessive time to learn AWS
    - Unknown risk due to inexperience with the technology
        - This risk can be mitigated by using online resources to learn the technology faster
- Microservice downages (Github, Python)
    - Unlikely due to previous experience with these services
        - To mitigate downages from Github, we will backup code locally
- Errors in workload estimates
    - Unlikely due to experience on previous team projects
        - Can be mitigated by having additional help on portions of the project
- Software compatibility issues
    - Somewhat likely due to differing operating systems
        - This is mitigated by the M1 mac having Rosetta, allowing it to run Intel code

The process of handling said risks and others will be done as follows

Figure 5.1 - Risk Management Plan Flowchart.:



2.4 Hardware and Software Resource Requirements

- Hardware Requirements
    - AWS Servers
- Software Requirements
    - AWS Software
    - Boto3 Library
    - Python or Java
    - AI/ML Libraries
    - NLP Libraries
    - GitHub
    - Visual Studio Code
    - PyCharm
- New Hardware and Software Learned by the team
    - AWS Software
    - Boto3 Library
    - AI/ML Libraries

2.5 Deliverables, Schedule

- Deliverables:
    - Project Management Plan - Feb 4

        - Documentation of the plan for developing an Intelligent Service Requests Resolver Framework, including information on the organization of the project members, specifics on the life-cycle implemented for the project, a risk analysis of the project, resource requirements, monitoring and reporting mechanisms, professional standards, and the deliverables and schedule of the project. This plan is to be revisited every sprint cycle for analysis and amended as needed.
    - Requirements Documentation - Feb 18
        - Documentation of the requirements of the project.
    - Architecture Documentation - March 4
        - Documentation of the architecture design that is going to be implemented in the project.
    - Detailed Design Document - March 21
        - Detailed documentation of the design of the project.
    - Testing Plan - April 8
        - Documentation on the plan for how the project will be tested for quality and whether it satisfies the desired outcomes.
    - Final Project Report - April 29
        - A report on the process of working on this project.
    - Final Project Demonstration - April 29

Table 7.1

Identifier	**Deliverable**	**Deadline**	**Estimated Time**	**Developers**	**Dependencies**
A	Project Management Plan	Feb 4	10 business days	all	-
B	Requirements Documentation	Feb 18	10 business days	all	A
C	Architecture Documentation	March 4	10 business days	all	A, B
D	Detailed Design Document	March 21	14 business days	all	A, B, C
E	Testing Plan	April 8	10 business days	all	A, B, C, D
F	Final Project	April 29	8 business	all	A, B, C, D, E

	Report		days		
G	Final Project Demo	April 29	7 business days	all	A, B, C, D, E, F

2.6 Monitoring, Reporting, and Controlling Mechanisms

- Management Reports
  - Should contain whether tasks are being successfully completed or if there are bottlenecks in the backlog
  - Should provide comparisons to previous reports and analyze differences
  - Should provide an overview of risks to the project
- Project Monitoring and control mechanisms
  - Sprint Retrospective meetings
    - Find out what is working well, and what is not working well
  - Sprint planning meetings
    - Decide which tasks and features to bundle into the next version for the developers to work on.

2.7 Professional Standards

Meetings

The team shall meet twice a week as follows:

Wednesdays - 12:15 - 13:00 (Weekly Scrum)

Fridays - 13:00 - 15:45 (Document Preparation)

The team shall conduct a weekly meeting with the sponsor (Amazon) on:

Fridays - 12:00 - 12:45

Governance Plan

Meeting	Day	Time	Input	Structure	Output	Owner	Participants
Document Preparation Meeting	Friday	13:00 - 15:45	Progress on tasks, research.	Team members shall work and discuss the relevant documents.	Ready documents.	Team Leader	Team Leader Team Members

Weekly Scrum	Wednesday	12:15 - 13:00	Progress on tasks, Research. Issues and problems that need to be addressed.	Scrum leader goes through team members and inquires about current tasks and progress.	Progress about tasks, new tasks for next week. Code segments	Team Leader	Team Leader Team Members
Stakeholders Meeting	Wednesday	11:30 - 12:45	Progress with the project, stakeholder questions.Discussions.	Team members ask questions and stakeholders answer.	Better understanding of the project. Requirements. Revised deliverables.	Team Leader	Team Leader Team Member Amazon Stakeholders

Overall Expectations:

- Team members will not plagiarize materials
- Team members will give their best efforts to complete assignments according to schedule, within reason
- Team members shall notify in advance if they can't attend a meeting.

2.8 Evidence the Document has been Placed Under Configuration Management

- 2/1/2022 - Project Management Plan - Version 1.W3

2.9. Impact of the Project on Individuals and Organizations

This project aims to create a framework for customer service interactions through any channel, and for any domain dictionary. This kind of configurability allows for any company to take this solution and implement it to their liking, or how it best fits their customers. It can filter out easily-solvable customer service requests and alleviate some of the volume of incoming customer requests. The framework also benefits the customers by being available at all times, as well as receptive to feedback to improve future suggestions. This project may

impact society by influencing other organizations to adopt similarly intelligent service request solutions, and make useful information more readily accessible to members of the public.

3. Requirement Specifications

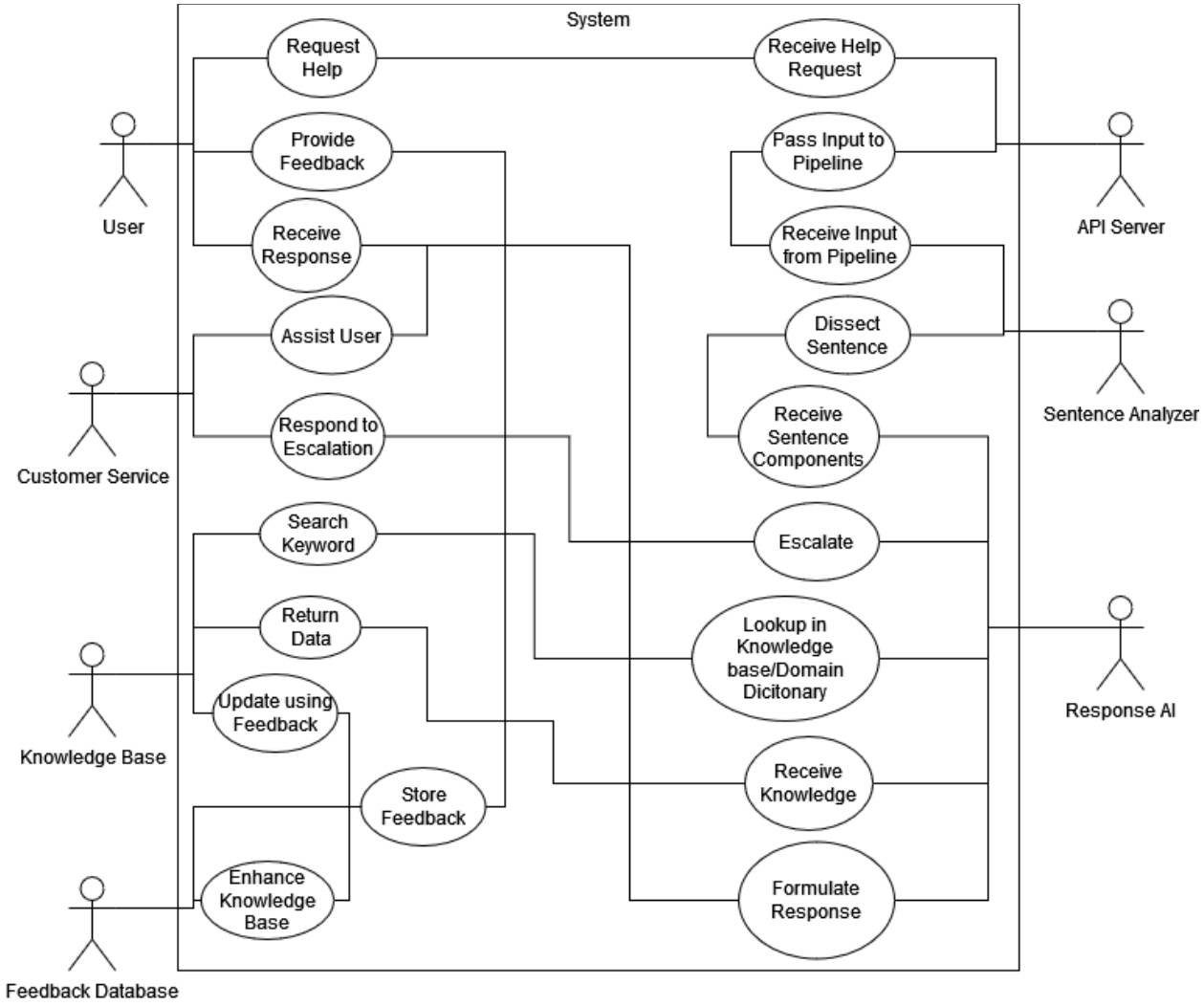
3.1. Stakeholders

- Eric Wong
  - ewong@utdallas.edu
- Dongcheng Li
  - dxl170030@utdallas.edu

- Product Owners
  - Gaurav Acharya
    - achagaur@amazon.com
  - Mihir Desai
    - desamihi@amazon.com
- SCRUM Master
  - Omer Reshef
    - Oxr17001@utdallas.edu
- Developers
  - Omer Reshef (Team Leader / Project Manager)
    - Oxr17001@utdallas.edu
  - Zach Jackson
    - ZJJ180000@utdallas.edu
  - Ethan Hays
    - ech180000@utdallas.edu
  - Mustafa Khan
    - mnk170030@utdallas.edu
  - Austin Hucabee
    - austin.hucabee@utdallas.edu
  - Olagoke Adereti
    - Osa180000@utdallas.edu
  -

3.2. Use case model

3.2.1. Graphic use case model



3.2.2. Textual Description of Use Case

Flow of Events: The User will create a request for help. The API server will receive the help request and pipe the request over to the Sentence Analyzer. The Sentence Analyzer will dissect the sentence (retrieving the sentiment, intent, context and keywords) These components are passed to the Response AI. The Response AI will either escalate the help request or will pass keywords and context to the knowledge base. The Knowledge Base will return the data to the Response AI which will then formulate a response for the user. After the user receives a response, they have the option to provide feedback.

Special Cases: Escalation: In the event of an escalation, the Response AI will forward the sentence received from the Sentence Analyzer to Customer Service who will respond. Customer Service will then Assist the User directly.

Feedback: In the event that a user wants to provide feedback, the Feedback database will store the feedback for later use. This later use is to enhance the Knowledge Base with additional information or by correcting outdated information.

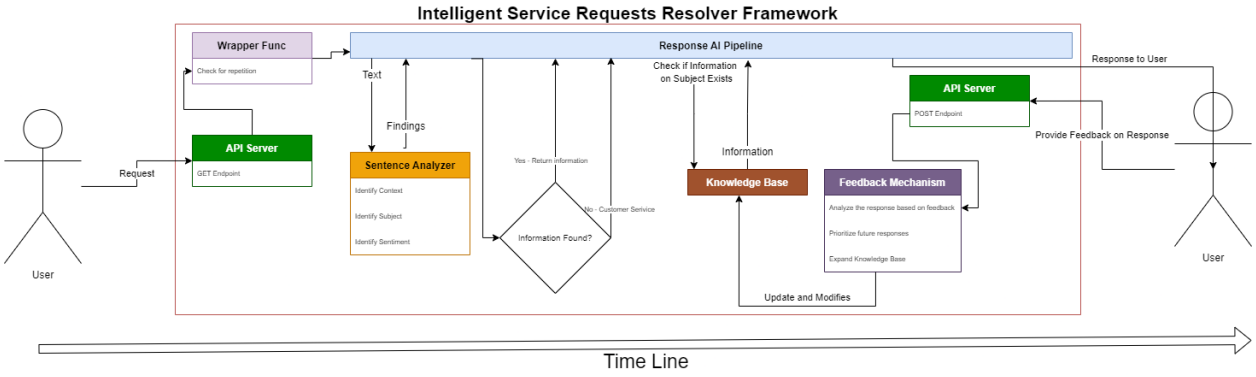
3.4. Non-functional requirements

- Knowledge Base:
    - The database will utilize Amazon S3 storage to sustain the large amount of information.
    - Retrieval, modification, insertion, and deletions from the database will be accomplished in milliseconds w/ S3 storage
    - The database shall elasticity, and automatically, scale to handle the data, based on S3 DynamoDB storage.
    - The database will be accessed only by the response AI to ensure security.
- Sentence Analyzer:
    - The analyzer shall return the extracted Sentiment, Context, Key Phrases, Key Words, Intent of the text using Amazon Comprehend.
    - The analyzer uses Amazon Comprehend API.
    - The analyzer shall not take more than 2 seconds to generate its findings.
- Generated Responses
    - The generated responses should make the user feel like they are having a conversation with a real person (response time should be less than 2 seconds).
    - The generated responses should reference multiple domain dictionaries
    - The generated responses will provide the user with context (background), possible actions, and results of suggested actions based on user inquiry.
    - The generated responses will be rated on a binary scale by the end-user to train the system in regard to helpful responses
- Pipeline:
    - The pipeline will be operational at all times (does not need uninterrupted runtime; reliable, repeatable instances can be utilized
    - An iteration through the pipeline shall be processed in no more than 5 seconds in SYNC mode.
    - An iteration through the pipeline shall be processed in no more than 5 minutes in A-SYNC mode.
    - The pipeline should not have direct access to the database, in order to ensure security
- API Server:
    - The API Server shall constantly be running
    - The API Server shall handle as many requests as the Amazon AWS API solver can.

4. Architecture

ARCHITECTURAL MODEL

The architecture model chosen for this project is a modified version of Pipe & Filter.



As we can see, the information starts flowing from the left towards the right, where the user is standing on both ends of the pipeline. The API endpoint receives the information from the user and pushes it to the pipeline's Lambda. The pipeline then converses back and forth with the microservices to understand and analyze the sentence, to retrieve information from the knowledgebase, and to better formulate an answer. Eventually the Response AI microservice gets all the information and sends back to the pipeline the final response, which ends up propagating to the user. The user is then asked to provide feedback on the quality of the response, that goes back to the system's feedback mechanism service, to better calibrate the system for future use.

Software Components

- AWS Services
    - This framework makes extensive use of the services hosted by AWS. In order to use these services, an Amazon AWS account is created, and subsequent IAM accounts are created to safely utilize these services.
- Amazon Comprehend
    - Comprehend is a natural-language processing service hosted on AWS servers that utilizes machine learning to process text and extract the sentiment, key phrases, and named entities from the text. Comprehend is used in the sentence analyzer service to identify and pass along the sentiment to the response AI. The key phrases and named entities are checked against the DynamoDB database to find any matches in the existing domain dictionaries. Any matches are also passed to the response AI. Comprehend was chosen because of its NLP capabilities, simplicity of use, and reliability.
- Amazon Lambda
    - Lambda is a serverless computing service hosted on AWS that operates on event triggers. Lambda is used as the main driver for this framework. Each microservice is implemented as a Lambda function, and the output from each triggers the next function in the pipeline. Lambda was chosen because it

eliminates the need to provision and manage servers, and can execute code at whatever capacity is needed.

- Amazon DynamoDB
    - DynamoDB is a serverless, key-value NoSQL database service hosted on AWS. DynamoDB will contain the data dictionaries used by the framework to identify and resolve the needs of the user. DynamoDB was chosen because AWS is reliable, DynamoDB supports scaling at whatever level is needed, and it supports NoSQL data structuring which allows for more flexibility.
- Python
    - Python is the main language of choice for this framework. It was chosen for its versatility, as well as the fact that it is supported by many of the AWS services.
- JavaScript
    - JavaScript is the language used for creating the API endpoints.
- Boto3 Library
    - The boto3 library is a python library for interacting with AWS services.
- GitHub
    - GitHub is the version control software of choice for hosting the source code of the framework. It was chosen because it is a free service, as well as a reliable one.

Hardware Components

- AWS Servers
    - The AWS services being used by the framework are hosted on AWS servers. AWS servers were chosen because the essential services for the framework utilize the AWS servers, as well as the fact that AWS servers are reliable.
- Github Servers
    - The source code is stored in a GitHub repository that uses GitHub servers.

Pipe & Filter strongly correlates to the following quality attributes.

Scalability
Description: the capability of the system to be changed in size and scale, as well as used/produced in a wide capacity.
Solution: The architecture supports scalability together with the AWS services. Each service, or filter, can be upscaled or downscaled based on need. The AWS services are built to scale.

Portability
Description: the usability of the software in different environments.
Solution: Pipe & Filter support portability, together with the AWS services. The microservices, with few steps of configuration can migrate to any other project. The microservices are tightly coupled, hence portability is high.

Security
Description: The protection of the system and users' information from theft and unauthorized use.
Solution: Pipe & FIlter puts only one point of failure for Security issues, which is the API endpoint. The rest of the services can only be called by a permissioned request, handled by AWS.
Availability & Reliability
Description: the probability that the system will produce correct outputs up to some given time.
Solution: Pipe & Filter, together with the AWS services, ensure reliability and availability of the microservices to a high degree of 9s. Basically the system is dependent upon the capabilities of Amazon's AWS. However, failure of a single component might not affect the resolution of the system as a whole, especially if backups can be used.

System Expectations

Volumes

- Estimated System Traffic: About 1500 requests per second. Arriving from users of various businesses.
- Registered Businesses: About 100
- Registered Tables per Business: About 15
- Projected 20 year Growth: About 30000 requests per second. 2500 registered businesses around the globe.

Performance

- Time to process a request: Around 1 second, end to end.
- Availability: The system shall be available based on the AWS standards.
- Concurrency: System can handle as many concurrent users as required, based on the AWS capabilities of creating and managing more Lambda functions.

ALTERNATIVE ARCHITECTURES NOT CHOSEN

Apart from Pipe & Filter, we have also considered using the following architectures:

Repository:

- Repository architecture is focused around a main repository surrounded by many different microservices that can be used at any given time. The repository handles the main traffic, and directs input to various microservices to receive their output. We chose not to use the Repository architecture due to

the required flow of information from one side to the other. In general, a repository that holds the pipeline, while communicating with various microservices is not a bad idea, but we chose to focus on the flow of information which strongly correlates to Pipe & Filter.

Layered:

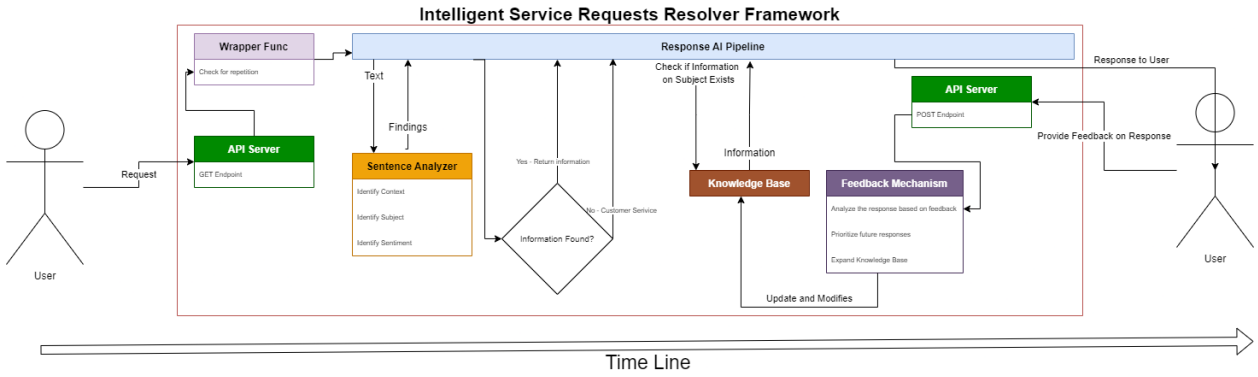
- Each layer in layered architecture offers a set of services to other layers. The communication between layers is done through procedure calls. However, each layer can usually only interact and utilize the services of the layer directly below it. This is highly inefficient, but can be improved if we allow layers to access all layers below it. Overall, the layered architecture is not a bad option, but due to the number of layers having a direct correlation to complexity, up-front cost, and performance penalties, we could not choose this option.

Peer-to-Peer:

- Peer-to-peer architecture is decentralized computing in which the flow of control and resources are distributed among peers. Peers are independent components with their own states and control thread. Communication between the peers is dictated by network protocols. This makes peer-to-peer not that great for time-critical data retrieval. Outside of that, it is highly robust in the face of failure for any given code, and it is also highly scalable with resources. Though highly useful for our needs, the issue related to our project with peer-to-peer is that we are using stateless services that do not work well with peer-to-peer, as well as the usage of a main pipeline to lead the flow of information from one side to the end doesn't fit the usage of peer-to-peer.

5. Design

Flow of Events



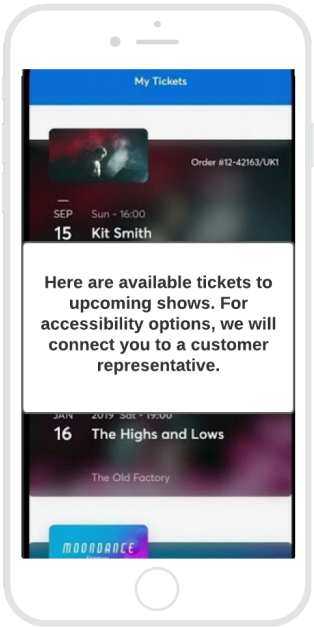
GUI Design

Screenshots from clients who are using our services throughout their businesses apps:

Ex. 1

1. A customer is using a virtual assistant in the TicketMaster app and asks for help finding tickets to an event or show they want to see.
2. The virtual assistant will search for available tickets to the requested show or event and display seats available with prices.
3. The user asks for special assistance or accommodations such as wheelchair accessible seats, and the virtual assistant will redirect to a representative.

Flow of events: The TicketMaster app sends the user's text to the framework's API. The text is analyzed by the framework and it is determined that they are looking for tickets to an event. After showing the customer available tickets and prices, the user asks for special assistance and since this cannot be handled by the framework, the user will be directed to speak with a TicketMaster representative.



Ex. 2

1. John, an expert chef for lucrative-cooking.com is in need of a new potato recipe.
2. John goes to his cooking website and types - "I need a new recipe for saturday, that includes potatoes as the main element. I also have tomatoes, garlic, and chicken breast, what can I make?"
3. Information shows up on John's cooking website - "You can make the following 3 recipes, here are the links: <potato pizza> <chicken palermo> <burmese fries>".

Flow of events: After John enters his request, the request makes its way to the framework's API. The sentence is analyzed, and found that John is interested in a potato dish, using the ingredients he already has. From the domain dictionary, the framework pulls several recipes, and returns it to John.

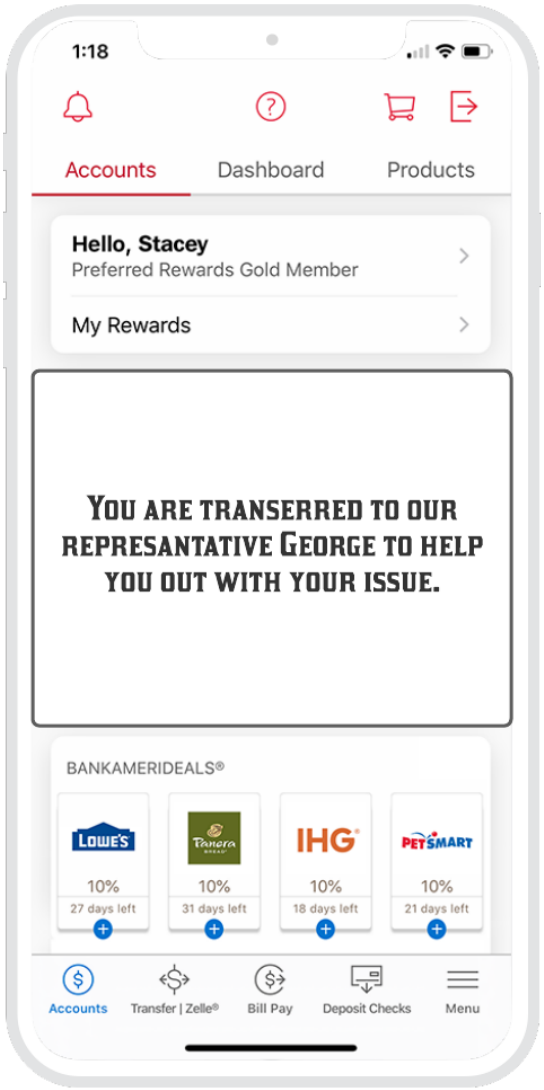
Special Conditions: The framework recognized that the customer is looking for recipes with several ingredients, where potato plays a big role in it. Hence it cross reference information inside the domain dictionary to retrieve the right information.

Ex. 3

1. A customer using the Bank of America's mobile app taps the virtual assistant and enters "Help me find out why the commissions are so high?"
2. The app will redirect the customer to the company's representative.

Flow of events: The app of Bank of America sent the user's text to the framework's API. The text was analyzed by the framework, and it found that the intent is negative, hence the request is escalated to a customer service representative. The framework also generated the required information regarding commissions, using the domain dictionaries, and sent the information to the company's representative.

Special Conditions: The framework recognized that the customer is unhappy, therefore a generic response would not be useful, hence a customer representative would be best to handle the situation, using the data the framework had found earlier.



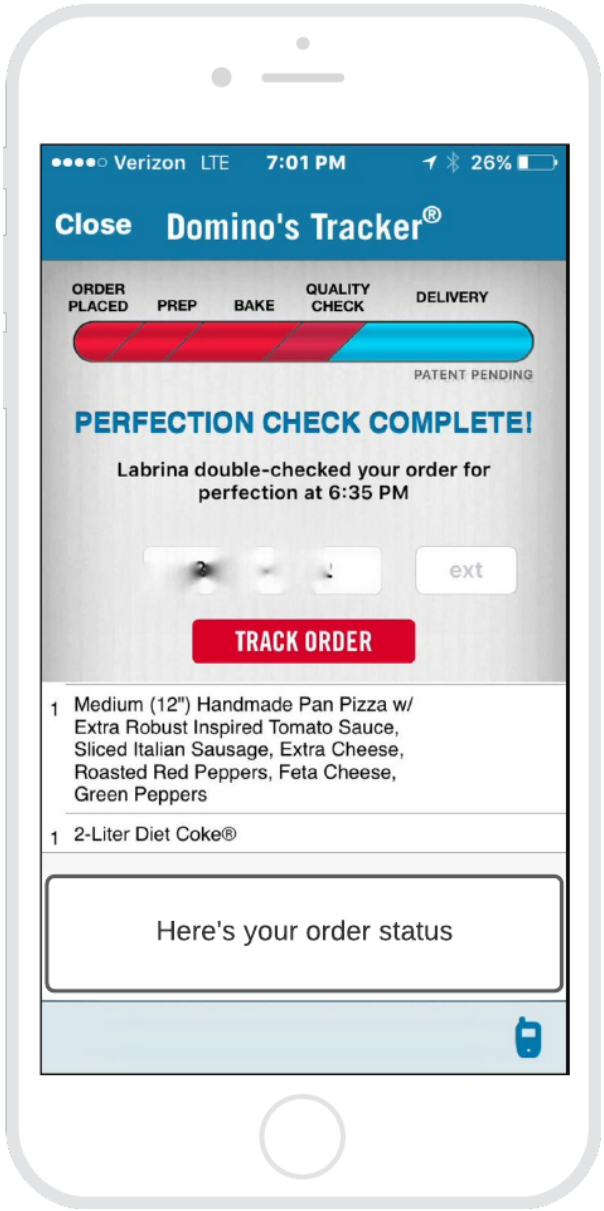
Ex. 4

1. A customer is ordering a pizza with the Domino's app and uses a voice enabled virtual assistant. The customer asks to have a meat lovers pizza delivered.
2. The virtual assistant asks to confirm the toppings, size of the pizza, number of pizzas, and offers additional sides or promotions.
3. The customer confirms their order and payment is applied using the credit card the customer has saved on file.
4. The customer has been waiting a while and asks the virtual assistant where the driver is/how much longer it will be.
5. The virtual assistant directs the customer to the order tracking section of the Domino's app.

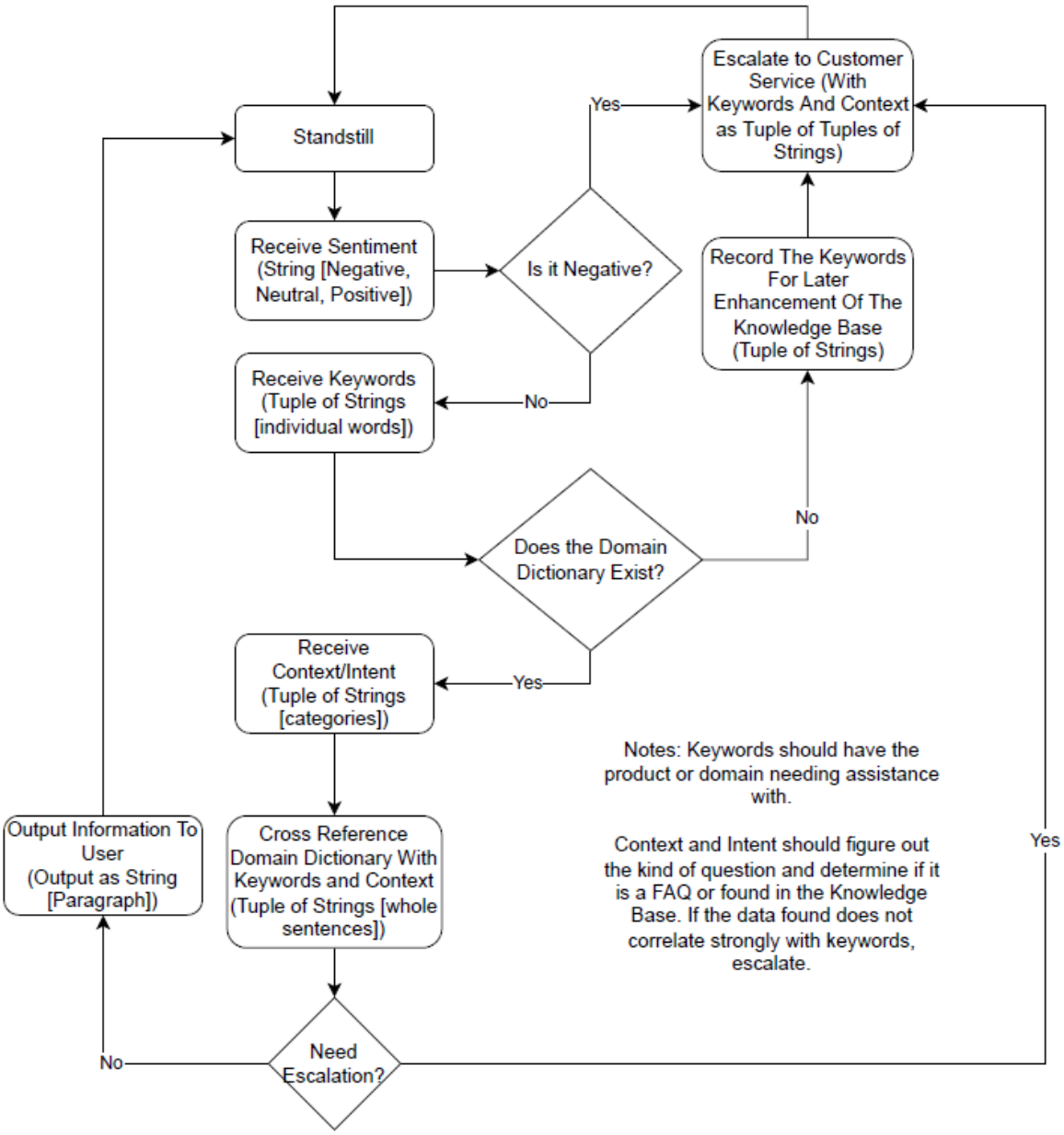
Flow of events: The Domino's app sends the voice data to the framework's API. The audio is converted into text and then the text is analyzed by the framework. The framework uses the

Domino's knowledge base to match the requested meat lovers pizza to that item on the menu, as well as to offer additional sides or promotions. Once the customer confirms their selections, the framework API will send the order information to Domino's internal system and payment is processed on Domino's end using the credit card the customer has saved to their account. After the order has been on its way for a little while and the customer asks where the driver is, the framework's API recognizes that the customer is wanting to know where their pizza is and that they are unhappy since they are asking where it has been after it has been 30 minutes.

Special Conditions: Since the customer is unhappy with their pizza not being delivered on time, the customer will receive an email apologizing for the delivery taking longer than expected and offering a coupon for a free pizza with their next order



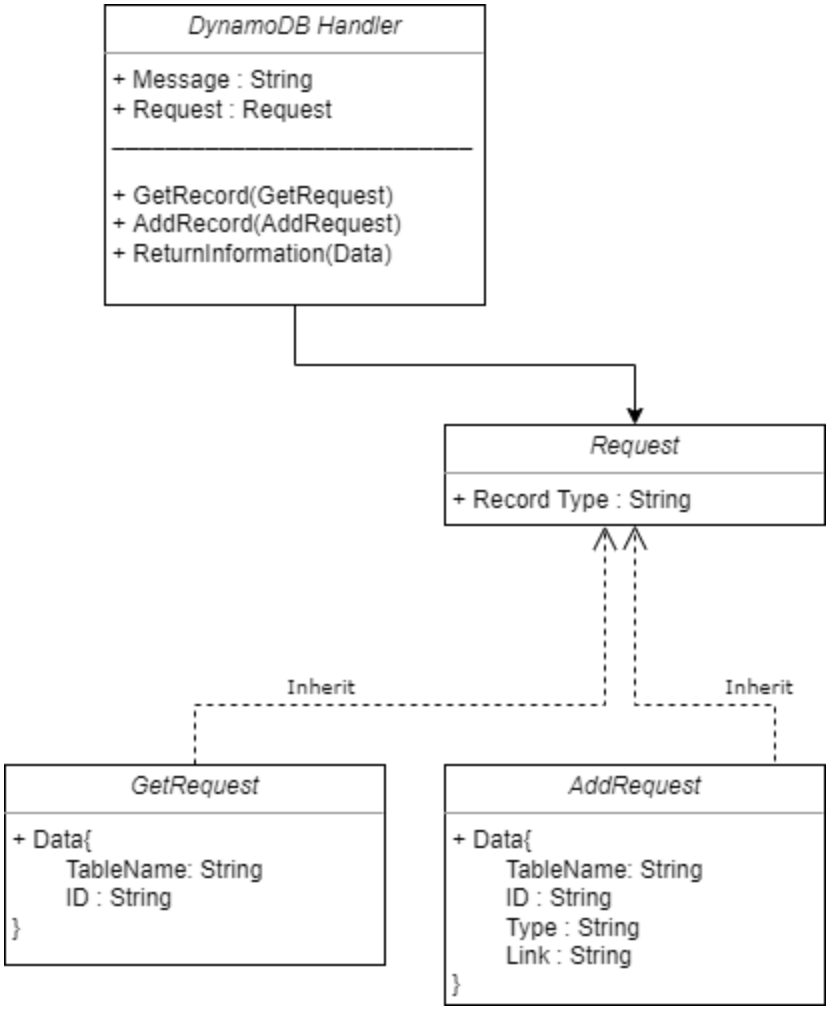
Decision Making AI Logic



CLASS DIAGRAMS

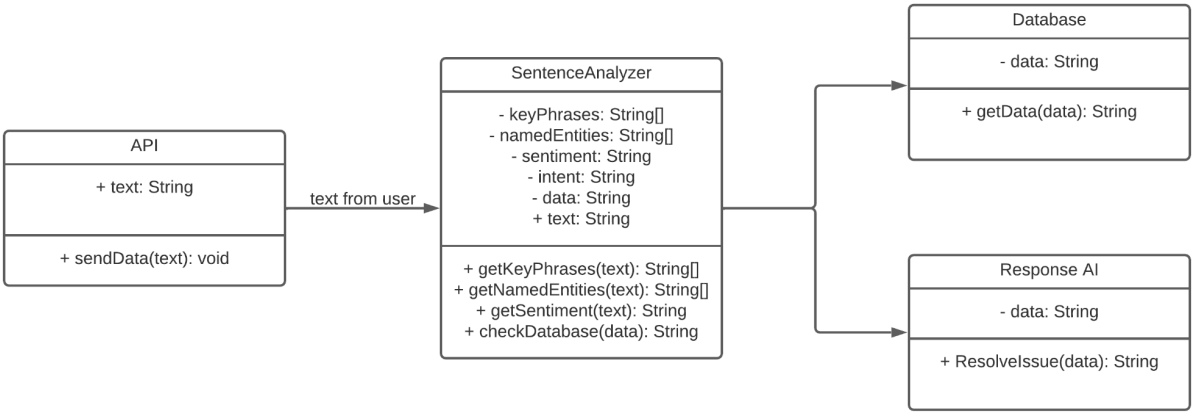
DynamoDB Handler

The class describes the functions and the objects inside the DynamoDB Handler. We have a request type that is used to communicate between the requester and the DB.



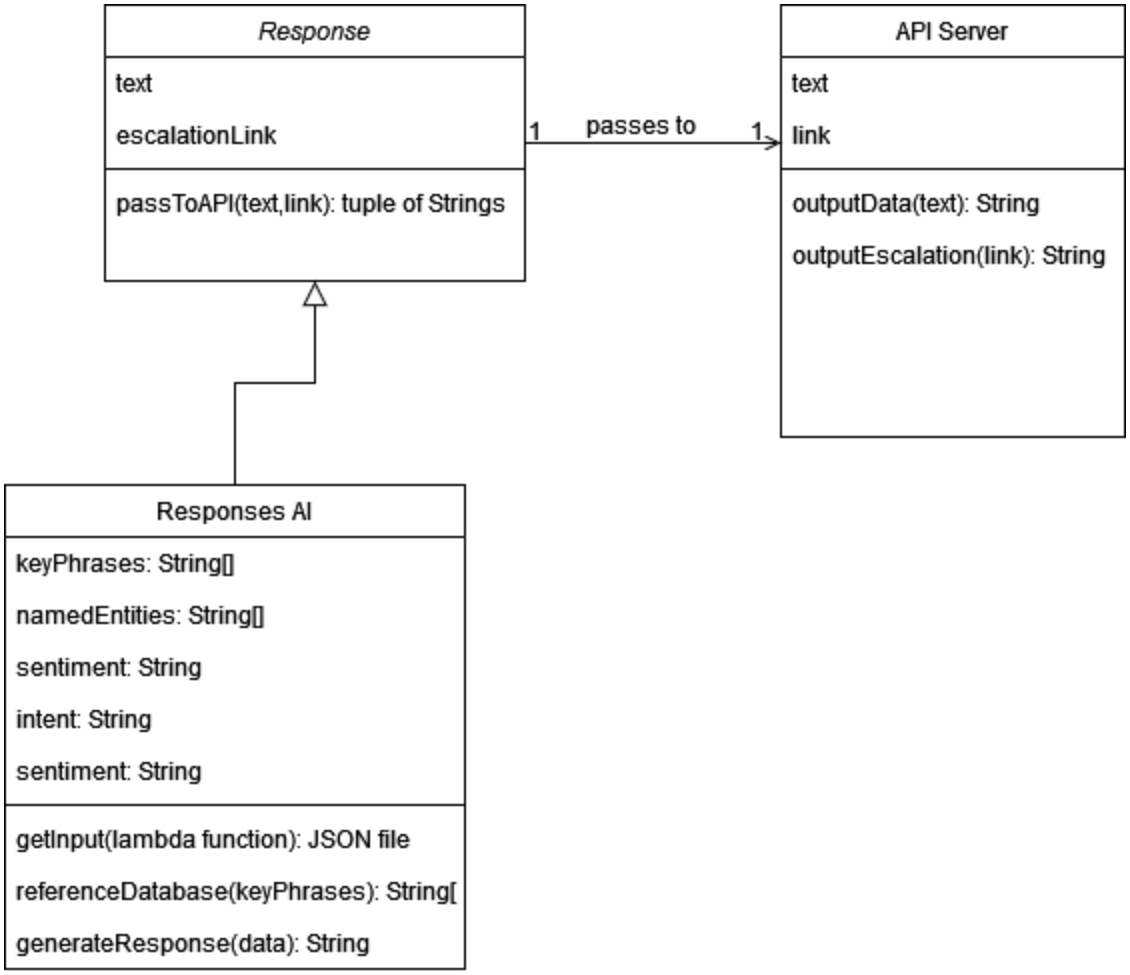
Sentence Analyzer

The Sentence Analyzer class functions to process input text that comes from the API class and store the insights extracted. This data can be checked against the data stored in the Database class, and any matches can be passed along to the Response AI class.



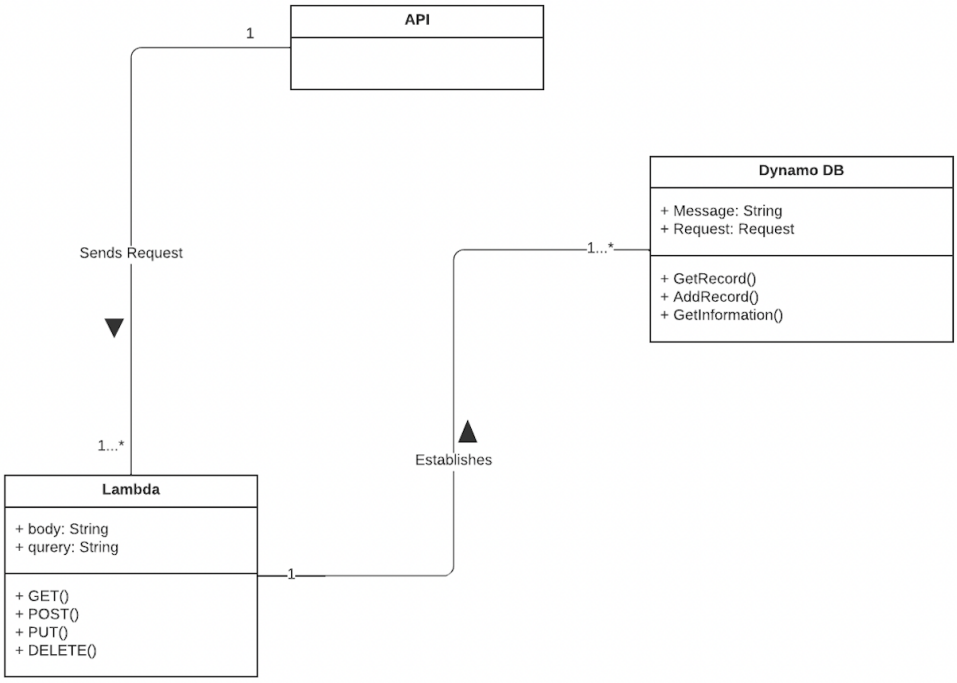
Response AI

The response AI class receives the data from the sentence analyzer in the form of a JSON file which is opened by the response AI. The pieces of data from the JSON are used to query the database to get back data that can be used for a response. If escalation is necessary then an escalation link will be output in the form of a response to the end user, otherwise, a normal text response will be output.



API

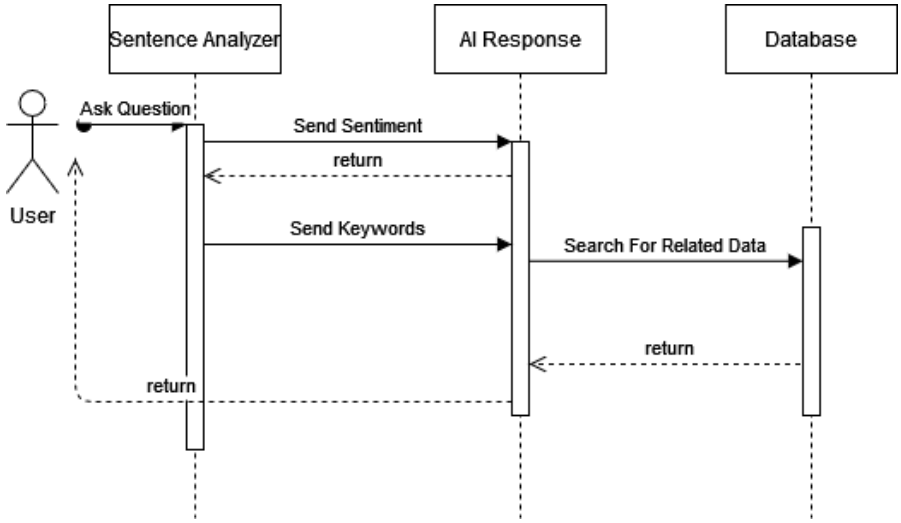
The API class sends a request to the Lambda class which establishes communication and receives responses from DynamoDB, a microservice. Lambda has some local variables for storing query or body texts as well as CRUD methods. DynamoDB has message and request variables as well as functions that handle the CRUD requests sent from Lambda.



SEQUENCE DIAGRAMS

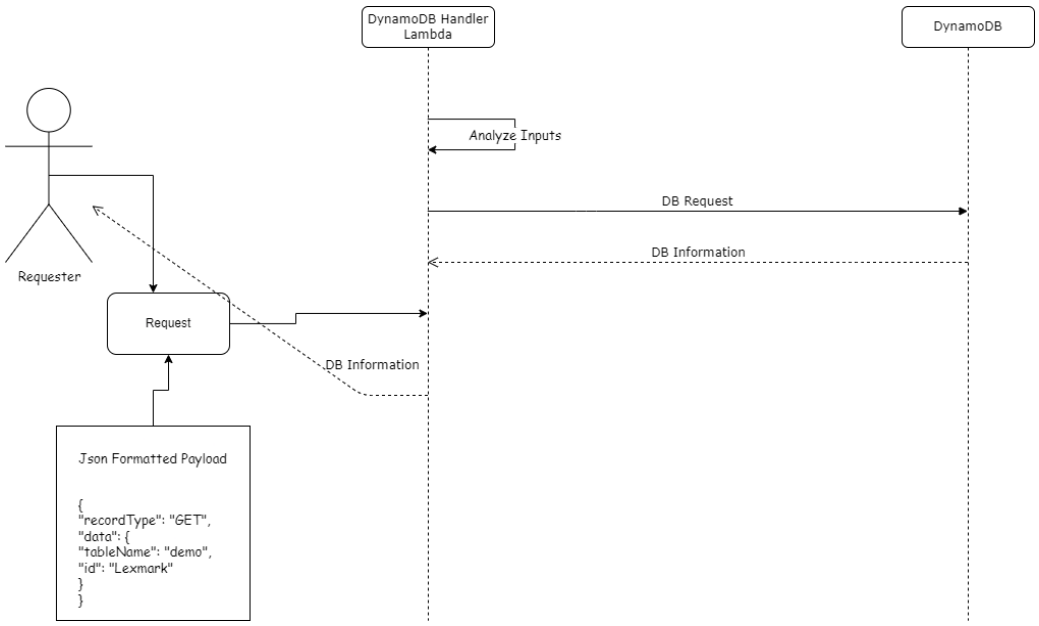
AI Response Function

The AI response function is the function responsible for being the intermediate function that handles the data output from the sentence analyzer and uses the database to find related data. This found data is then returned back to the user that requested it.



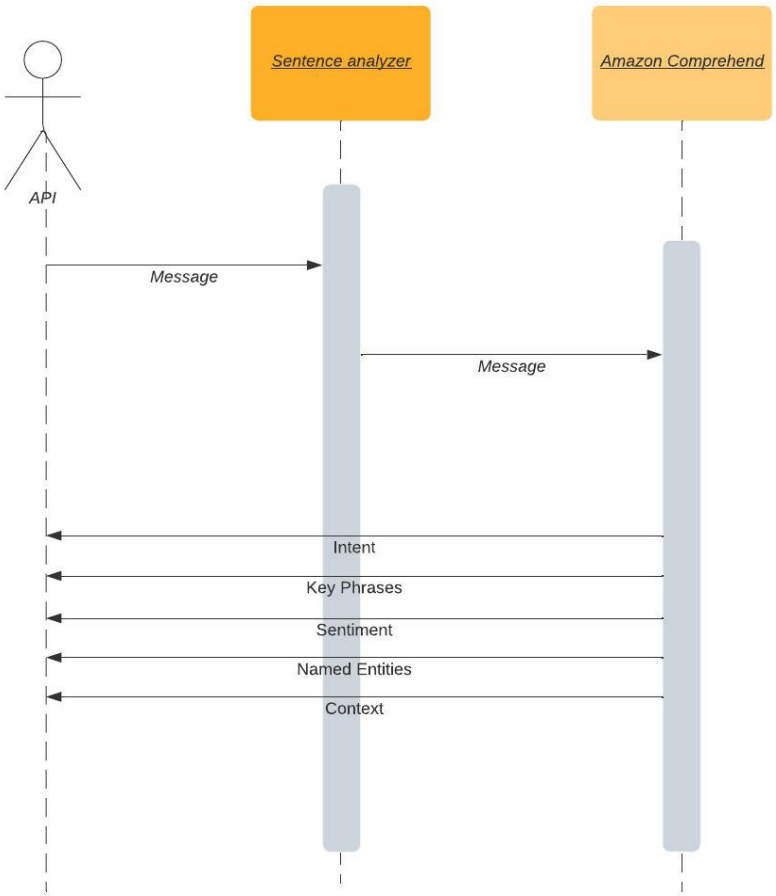
DynamoDB Handler

Input is sent to the Lambda function, in the form of a JSON object. The information is being analyzed by the Handler, and a request is sent to DynamoDB. The information then propagates back to the requester.



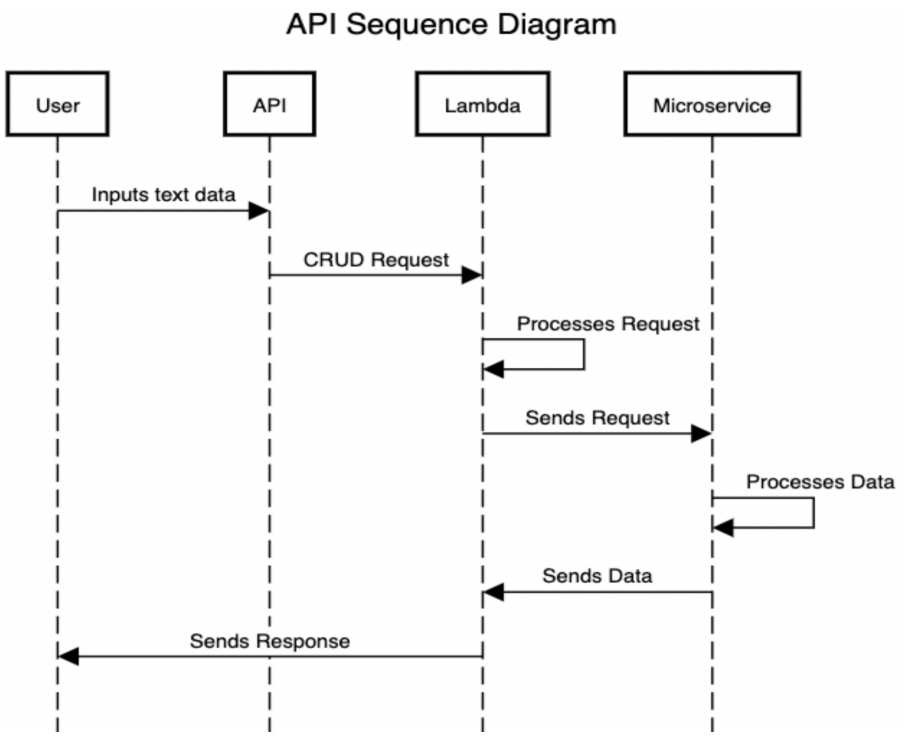
Sentence Analyzer

The Sentence Analyzer processes input text received from the API. The sentence analyzer uses Amazon Comprehend to extract intent, key phrases, sentiment, named entities, and context and this information is returned back to the API.



API

The user interacts with the chatbot which establishes communication with an API. The API sends a CRUD request to Lambda which Lambda processes and sends the request to the appropriate microservice. The microservice computes local data and sends a response back to Lambda. The user receives the response from Lambda.



TRACEABILITY FROM REQUIREMENTS TO DETAILED DESIGN MODEL

Requirements:

Functional:

Knowledge Base:

1. The knowledge base shall be a database.
2. The database shall hold information.
3. Information can be retrieved from the database.
4. Information can be added to the database.
5. Information can be removed from the database.
6. Information can be modified in the database.
7. The database shall be able to expand and shrink in size if needed.

Sentence Analyzer:

8. The sentence analyzer shall analyze text.
9. The analyzer shall detect sentiment from the input text.
10. The analyzer shall detect key phrases from the input text.
11. The analyzer shall detect context from the input text.
12. The analyzer shall detect intent from the input text.
13. The analyzer shall detect named entities from the input text.

Pipeline Worker:

14. The worker shall push the result from the sentence analyzer to the response AI
15. The worker shall return/push the first few pieces of information contained in the database about the subject to the user
16. The worker shall alert user and make a note if there is no information about the requested subject

API Server:

17. The API request shall only be triggered by the user
18. A GET endpoint shall be used to receive user's HTTP requests.
19. The API request will utilize end-user input as initial parameters

Generated Responses:

20. The generated response will return to the user.
21. The generated response will be within a domain dictionary.
22. The generated response will contain lines of information.
23. The generated response will be rated on its quality.

Non-Functional:

Knowledge Base:

24. The database will utilize Amazon S3 storage to sustain the large amount of information.
25. Retrieval, modification, insertion, and deletions from the database will be accomplished in milliseconds w/ S3 storage
26. The database shall have elasticity - it will automatically scale to handle the data, based on S3 DynamoDB storage.
27. The database will be accessed only by the response AI to ensure security.

Sentence Analyzer:

28. The analyzer shall return the extracted Sentiment, Context, Key Phrases, Key Words, Intent of the text using Amazon Comprehend.
29. The analyzer uses Amazon Comprehend API.
30. The analyzer shall not take more than 2 seconds to generate its findings.

Generated Responses

31. The generated responses should make the user feel like they are having a conversation with a real person (response time should be less than 2 seconds).
32. The generated responses should reference multiple domain dictionaries
33. The generated responses will provide the user with context (background), possible actions, and results of suggested actions based on user inquiry.
34. The generated responses will be rated on a binary scale by the end-user to train the system in regard to helpful responses

Pipeline:

35. The pipeline will be operational at all times (does not need uninterrupted runtime; reliable, repeatable instances can be utilized

36. An iteration through the pipeline shall be processed in no more than 5 seconds in SYNC mode.
37. An iteration through the pipeline shall be processed in no more than 5 minutes in A-SYNC mode.
38. The pipeline should not have direct access to the database, in order to ensure security

API Server:

39. The API Server shall constantly be running
40. The API Server shall handle as many requests as the Amazon AWS API solver can.

Business Use Cases:

1. Instructions to print using email
2. Help to save on credit
3. Find new adjustable diet routine on DIET.COM
4. Customer is unhappy with product they ordered
5. Customer receives damaged product in the mail
6. Customer needs assistance setting up new device
7. User needs assistance with a service that is not working
8. Utility Help Finding

Product Use Cases:

1. Getting intent of text
2. Adding a new domain dictionary to the knowledge base.
3. Update an existing domain dictionary
4. Getting key words and phrases of text
5. Update knowledge base with feedback from user
6. Getting named entities from text

Traceability Matrix

Req #	Buc #	Puc #	Design Link	Status
1-7 24-27	3	3-5	DyanmoDB Handler	Done (Green)
8-13 28-30	8	1,6	Sentence Analyzer	Work in Progress (Yellow)

14-16 39-40	1-8	5	API	Work in Progress (Yellow)
17-23 31-38	1-8	1-6	Response AI Logic	Work in Progress (Yellow)

Requirements Verification Template					
BUC/PUC/REQ #	Description	Rationale	Fit Criteria	Dependencies	Supporting Materials
1/1/12	G	G	G	N/A	G
2/4/10	G	N	N	N/A	G
3/6/13	G	N	G	N/A	N
4/5/6	G	N	G	G	N
5/1/12	G	G	G	N/A	N
6/1/20	G	G	G	N/A	G
7/4/10	G	G	G	N/A	N
8/1/8	G	N	G	N/A	N/A

Prioritization Matrix

The ranking of the requirements based upon the descending value of its total weight is the most ideal prioritization for implementation of the requirements/deliverables.

Req.	Factor - Score out of 10	%Weight applied	Fator - Score out of 10	%Weight applied	Factor - Score out of 10	%Weight applied	Factor - score out of 10	%Weight applied		Total Weight
	Value to Customer	40	Value to Business	20	Minimise Implementation Cost	10	Ease of Implementation	30	Priority Rating	100
Req. 1	4	1.6	3	0.6	2	0.2	10	3	5.4	54
Req. 2	4	1.6	6	1.2	2	0.2	10	3	6	60
Req. 3	6	2.4	6	1.2	3	0.3	9	2.7	6.6	66
Req. 4	7	2.8	5	1	2	0.2	9	2.7	6.7	67
Req. 5	6	2.4	5	1	3	0.3	8	2.4	6.1	61
Req. 6	8	3.2	5	1	3	0.3	8	2.4	6.9	69
Req. 7	8	3.2	7	1.4	1	0.1	7	2.1	6.8	68
Req. 8	5	2	6	1.2	2	0.2	7	2.1	5.5	55
Req. 9	7	2.8	7	1.4	2	0.2	4	1.2	5.6	56
Req. 10	6	2.4	5	1	1	0.1	5	1.5	5	50
Req. 11	7	2.8	5	1	1	0.1	3	0.9	4.8	48

Req. 12	8	3.2	6	1.2	3	0.3	5	1.5	6.2	62
Req. 13	6	2.4	4	0.8	4	0.4	4	1.2	4.8	48
Req. 14	6	2.4	4	0.8	1	0.1	8	2.4	5.7	57
Req. 15	4	1.6	4	0.8	6	0.6	7	2.1	5.1	51
Req. 16	5	2	5	1	8	0.8	8	2.4	6.2	62
Req. 17	2	0.8	3	0.6	6	0.6	8	2.4	4.4	44
Req. 18	3	1.2	2	0.4	3	0.3	8	2.4	4.3	43
Req. 19	7	2.8	5	1	7	0.7	10	3	7.5	75
Req. 20	6	2.4	4	0.8	3	0.3	9	2.7	6.2	62
Req. 21	9	3.6	2	0.4	3	0.3	5	1.5	5.8	58
Req. 22	7	2.8	3	0.6	4	0.4	9	2.7	6.5	65
Req. 23	6	2.4	8	1.6	5	0.5	9	2.7	7.2	72
Req. 24	3	1.2	7	1.4	7	0.7	8	2.4	5.7	57
Req. 25	6	2.4	6	1.2	3	0.3	6	1.8	5.7	57
Req. 26	7	2.8	8	1.6	8	0.8	7	2.1	7.3	73
Req. 27	9	3.6	10	2	3	0.3	8	2.4	8.3	83
Req. 28	6	2.4	8	1.6	4	0.4	6	1.8	6.2	62
Req. 29	6	2.4	8	1.6	8	0.8	7	2.1	6.9	69
Req. 30	8	3.2	5	1	2	0.2	5	1.5	5.9	59
Req. 31	7	2.8	7	1.4	3	0.3	7	2.1	6.6	66
Req. 32	5	2	4	0.8	1	0.1	5	1.5	4.4	44
Req. 33	6	2.4	6	1.2	3	0.3	6	1.8	5.7	57
Req. 34	8	3.2	10	2	5	0.5	9	2.7	8.4	84
Req. 35	7	2.8	9	1.8	2	0.2	6	1.8	6.6	66
Req. 36	8	3.2	4	0.8	3	0.3	5	1.5	5.8	58
Req. 37	7	2.8	3	0.6	3	0.3	5	1.5	5.2	52
Req. 38	9	3.6	5	1	2	0.2	8	2.4	7.2	72
Req. 39	8	3.2	9	1.8	3	0.3	6	1.8	7.1	71
Req. 40	7	2.8	9	1.8	6	0.6	9	2.7	7.9	79

6. Test Plan

6.1. Requirements/specifications-based system level test cases

Integration Testing

- Branch Coverage Testing
    - Case 1- Moved to customer service

        - Input: "text : I want to eat pizza"
        - Expected Output: "You are transferred to customer service"
    - Case 2- Received response as required
        - Input: "text : I need to reset my password"
        - Expected Output: "ResetPassword.com"
    - Case 3- Received bad response
        - Input: "text : I need to reset my password"
        - Expected Output: "OrderPizza.com"

Unit Testing

- API
    - Case 1 - GET endpoint
        - Based on the query parameters, the correct response is generated.
        - Input: "password"
        - Output: "ResetPassword.com"
    - Case 2 - POST endpoint
        - Recognizes request body
        - Creates row(s) in the correct specified database
    - Case 3 - PUT endpoint
        - Recognized request body
        - Replaces values in the correct row(s) in the database table
    - Case 4 - DELETE endpoint
        - Deletes correct row(s) in the database table
- ResponseAI
    - Case 1 - Bad Sentiment
        - Input: "text : I can't find my password, i'm lost, i'm stuck, what do I do"
        - Expected Output: return to API - "(Negative Sentiment) You are transferred to customer service"
    - Case 2 - Intent was not found
        - Input: "text : I want to eat pizza"
        - Expected Output: return to API - "(Intent was not found) You are transferred to customer service"
    - Case 3 - Working Sentence
        - Input: "text : I need to reset my password"
        - Expected Output: return to API - "ResetPassword.com"
    - Case 4 - Database does not include item
        - Input: "text : I can't find my password, i'm lost, i'm stuck, what do I do"
        - Expected Output: return to API - You are transferred to customer service"
- SentenceAnalyzer
    - Case 1 - Check forgotten password response
        - Input: "I forgot my password, please help me change it"

        - Expected Output: Intent class = Password with a confidence score of >.7
    - Case 2 - Check network error response
        - Input: "I can't connect to the WiFi"
        - Expected Output: Intent class = Network with a confidence score of >.7
    - Case 3 - Check equipment malfunction response
        - Input: "My keyboard stopped working"
        - Expected Output: Intent class = EquipmentFault with a confidence score of >.7
    - Case 4 - Check unrelated response
        - Input: "I need help setting up Outlook"
        - Expected Output: Intent class = FALSE
    - Case 5 - Check false password response
        - Input: "I changed my password but now I forgot how to delete my history"
        - Expected Output: Intent class = FALSE
- DynamoDB
    - Case 1 - An item that exists in the DB
        - Input: "ID : password", "tableName : demo"
        - Expected Output: Link: "ResetPassword.com"
    - Case 2 - An item that doesn't exists in the DB
        - Input: "ID : blue cheese", "tableName : demo"
        - Expected Output: "key was not found"
    - Case 3 - A table that does not exist
        - Input: "ID : blue cheese", "tableName : green table"
        - Expected Output: "table was not found"
    - Case 4 - Can't connect to DynamoDB
        - Input: "ID : password", "tableName : demo"
        - Expected Output: "Could not connect to DynamoDB"

6.2. Traceability of test cases to use cases

Requirements Verification Matrix

Requirements Verification Template					
BUC/PUC/REQ #	Description	Rationale	Fit Criteria	Dependencies	Supporting Materials
1/1/12	G	G	G	N/A	G
2/4/10	G	G	N	N/A	G
3/6/13	G	G	G	N/A	G
4/5/6	G	G	N	G	G
5/1/12	G	G	G	N/A	G
6/1/20	G	G	G	N/A	G
7/4/10	G	G	G	N/A	G
8/1/8	G	G	N	N/A	N/A

Requirements Traceability Matrix

Req #	Buc #	Puc #	Design Link	Status
1-7 24-27	3	3-5	DyanmoDB Handler	Done (Green)
8-13 28-30	8	1,6	Sentence Analyzer	Work in Progress (Yellow)
14-16 39-40	1-8	5	API	Work in Progress (Yellow)
17-23 31-38	1-8	1-6	Response AI Logic	Work in Progress (Yellow)

6.3. Techniques used for test generation

Integration Testing

We chose we following two techniques in order to test the system:

- Branch Coverage Testing - In order to verify that we cover all the various branches of the system, and to make sure that each one provides the response we aim to get.
- End to End Testing - In order to test the system as a whole, and to make sure all the many services are doing their part, and no error or mistake happens in any parts of the system.

Unit Testing

API - Postman SaaS

- White box testing - white box testing applies to the API - SaaS microservice by allowing us to refer to the various endpoints, and test them using specific inputs. These various test cases will allow us to make sure that the endpoints are working as required. The specific endpoints are answering various requirements of the project, and by making sure each of them works using white box testing, we are covering that requirements.

ResponseAI

- Branch Coverage - each step of making sure that the output is coming from the right key/table/DB means that a branch is formed with one path going on to the next step as a success and the other branch leading to a fail state. Each of the tests above either lead to a total success state or one of the possible fail states, which leads to 100% branch coverage.

SentenceAnalyzer

- White box testing - white box testing can be applied to the sentence analyzer service to help provide structural analysis of the code, which helps with ensuring adequate code and branch coverage. This means to test all logical pathways of the code, and to verify the logic of the code. This includes testing many different intent types, sentiment values, named entities, and key phrases. This also includes trying to find cases where this module falsely identifies these parameters.

DynamoDB

- White box testing - white box testing applies to the DynamoDB microservice by allowing us to refer to the actual content of the database with respect to the tests we want to implement. Such cases would include both checking for data that is actually in the database and making sure that the database isn't returning messages containing data not maintained by the DynamoDB
- Branch Coverage - each step of making sure that the output is coming from the right key/table/DB means that a branch is formed with one path going on to the next step as a success and the other branch leading to a fail state. Each of the tests above either lead to a total success state or one of the possible fail states, which leads to 100% branch coverage.

6.4. Assessment of the goodness of your test suite

In order to test the project, and to make sure that the tests are indeed up to our standards, we have used several suits of testing. We have used Branch tests, white tests, integration testing and unit testing in order to make sure that our software does what it is intended to do. The metrics that are used are that the tests will all pass with 100% pass rate. And the various test cases are covering all the branches of the code, as well as white box testing of the probable test cases, including sentences, text input, frontend usage.

References

1. M. Desai and G. Acharya, "SE4485_Spring2022_IntelligentRequestResolver_Framework," Jan-2022. .
2. "S3 Amazon," *Amazon*. [Online]. Available: https://aws.amazon.com/s3/features/?nc=sn&loc=2. [Accessed: 18-Feb-2022].
3. D. Rangel, "DynamoDB: Everything you need to know about Amazon Web Service's NoSQL database," *Amazon*, 2015. [Online]. Available: https://aws.amazon.com/dynamodb/?trk=ps_a134p000006papBAAQ&trkCampaign=acq_paid_search_brand&sc_channel=PS&sc_campaign=acquisition_IL&sc_publisher=G

oogle&sc_category=Database&sc_country=IL&sc_geo=EMEA&sc_outcome=acq&sc_detail=amazon+dynamodb&sc_content=DynamoDB_e&sc_matchtype=e&sc_segment=536393696638&sc_medium=ACQ-P%7CPS-GO%7CBrand%7CDesktop%7CSU%7CDatabase%7CDynamoDB%7CIL%7CEN%7CText&s_kwcid=AL%214422%213%2153639369663 8%21e%21%21g%21%21amazon+dynamodb&ef_id=Cj0KCQiApL2QBhC8ARIsAGMm-KFtIMZgBzPAsqxgacFe3P8ERzbhfjftLfAXFEPd9RW7yJaLjSuaefwaAvfWEALw_wcB%3AG%3As. [Accessed: 18-Feb-2022].

4. J. Mylet, "Amazon LEX," *Amazon*, 2012. [Online]. Available: https://aws.amazon.com/lex/. [Accessed: 18-Feb-2022].
5. D. A. V. I. D. MUSGRAVE, "Lambda," *Amazon*, 2022. [Online]. Available: https://aws.amazon.com/lambda/. [Accessed: 18-Feb-2022].
6. "Deploy to your site using the hosting REST API | firebase documentation," *Google*. [Online]. Available: https://firebase.google.com/docs/hosting/api-deploy. [Accessed: 18-Feb-2022].