**NormChat Software Design Document**

**Group 9 - ITCS 4155**

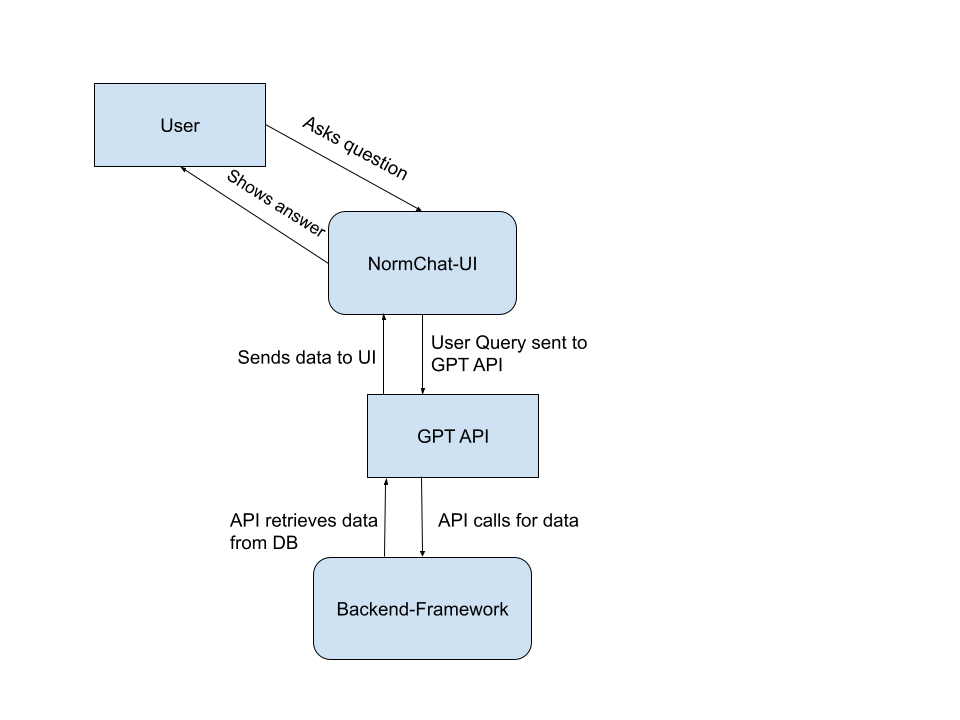
The purpose of the software design document (SD2) is to provide an overview of the system and to help people understand the system. This document can be used by a programmer as a guideline for implementing the design without needing to make significant engineering design decisions, and will be provided to the customer as a final deliverable.

1. Project Overview

NormChat is an AI Chatbot specifically created to streamline information from the UNC Charlotte website to students. Students can access the chatbot with their email address and ask UNC Charlotte related questions to the service. It will then get information from the site and create a simple response in seconds. Navigating the Charlotte website can be frustrating, with the outdated user interface and important information covered under tabs among tabs. Students who want their niche information answered can use NormChat to get their questions answered in seconds.

General Model -

Context Diagram:



User Stories:

1. Front-end: As a student I need to know specific prerequisites before registration so that I can register for classes easily, avoiding confusion. Back-end: User should be able to input specific class information into chatbox, output should display class information, filtering redundancy based on specific question
2. Front-end: As a freshman I want to know what's happening around campus so that I can get familiarized. Back-end: Users should be able to access dynamic links that showcase different events that are happening around campus.
3. Front-end: As a user I want a relevant link with search results. Back-end: When a user makes a query the system will return a relevant link when applicable.
4. Front-end: As a user I want to be able to log in with my email. Back-end: Users should be able to log in with email
5. Front-end: As a user I want to be able to go back and view past conversations. Back-end: User should be able to access previous conversations through the conversation list.

Product Backlog:

Sprint 1: <https://docs.google.com/spreadsheets/d/1OyyPfxTGlzIdfpAigmoGJQGeat_sFh_DdCqTJmh6uY4/edit?usp=sharing>

Sprint 2:

<https://docs.google.com/spreadsheets/d/1RWwvIk5Q6uQCBCjcRUcyR7igKs3G-2zORgU3jORX4cM/edit?usp=sharing>

Sprint 3:

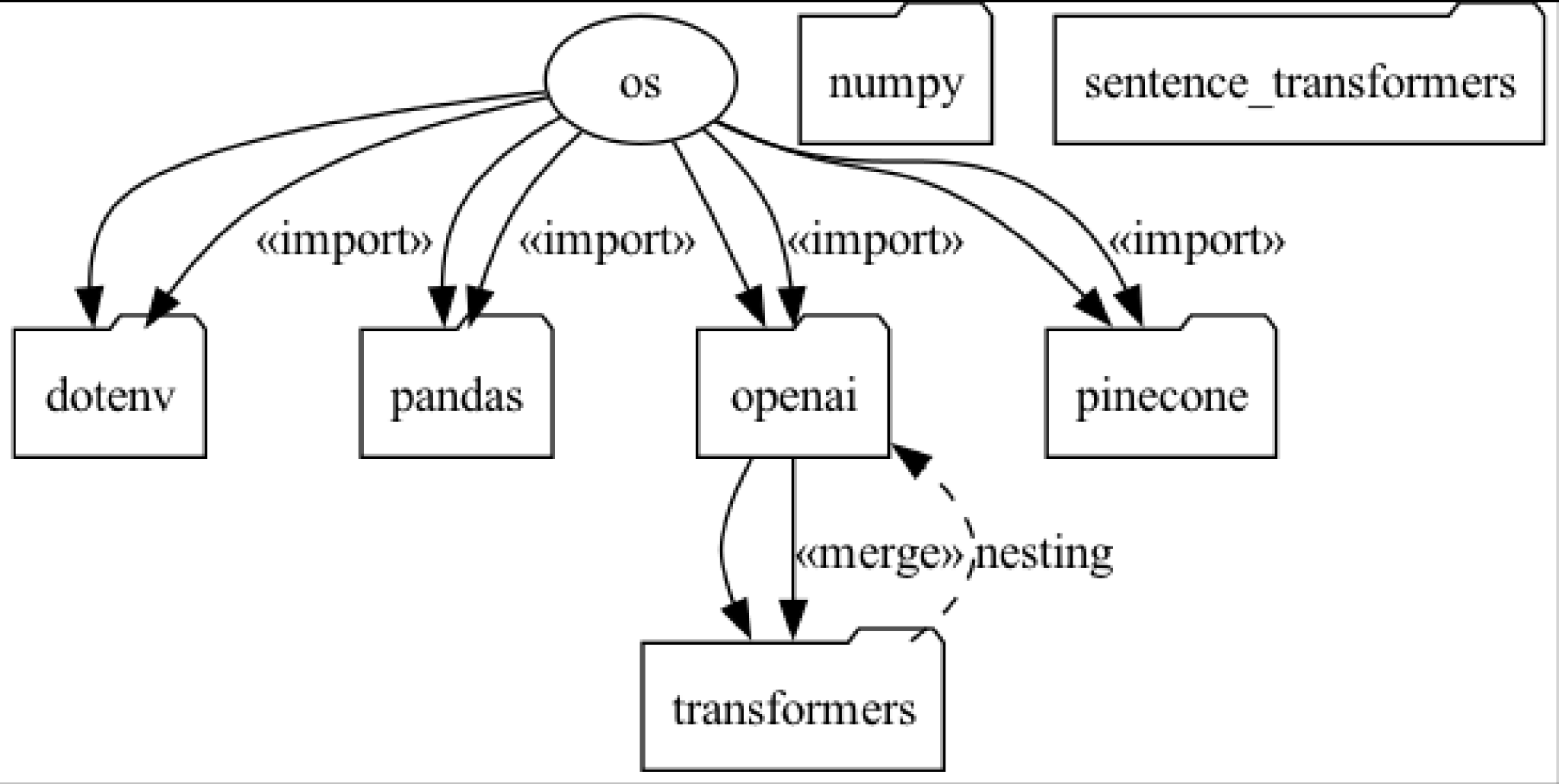
<https://docs.google.com/spreadsheets/d/1MQK4TSD3t9i4dGSKiv18oHZ9L12fl3jugYFZhDBfFcM/edit?usp=sharing>

2. Architectural Overview

Initially, we chose to utilize an open-source repository for Norm Chat's framework to expedite development and use existing resources. However, as we made more progress, we encountered challenges that required us to reevaluate our approach. Our decision to transition away from the initial repository was informed by considerations of established design patterns and architectural principles.

An important part of our decision-making was adopting a web-based framework to accommodate Norm Chat's functionality across various devices. This is why we selected React as the framework for our website, to ensure responsiveness and a better user experience on desktops, tablets, and smartphones. While we explored the initial open-source code, we found inconsistencies. Specifically, we found instances of redundant code conflicting with project requirements, leading to inefficiencies, as well as the complexity of certain parts of the code that was beyond the expertise of some of our team members. By switching to React and separating from the initial repository, we were able to develop Norm Chat in a way that ensures scalability and also sets the stage for a more sustainable architecture for future progress with the project.

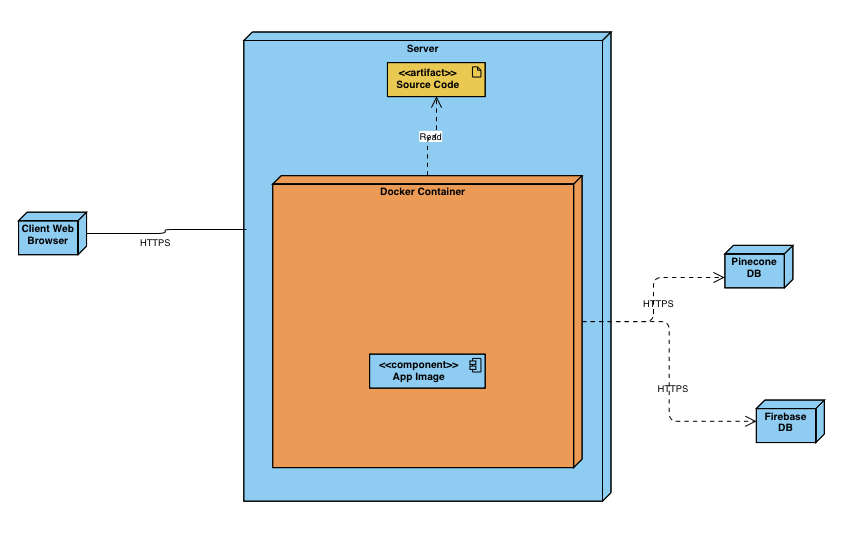
2.1 Subsystem Architecture



* dotenv Package:
  + Responsibility: Manages environment variables for the application by loading them from the .env file.
  + Architectural Style: Configuration Management (separating configuration from code for easy changes).
* pandas Package:
  + Responsibility: Handles data analysis and manipulation, especially for structured data.
  + Architectural Style: focuses on managing and processing data efficiently
* Openai Package
  + Responsibility: Interacts with the openai API for natural language processing tasks.
  + Architectural Style:provides functionality as services accessible over a network
* numpy Package:
  + Responsibility: Supports mathematical functions.
  + Architectural Style: focuses on efficient numerical computation.

pinecone Package:

* + Responsibility: Stores and queries high-dimensional vectors efficiently.
  + Architectural Style: provides optimized storage for vector-based data
* sentence\_transformers Package:
  + Responsibility: Generates vector representations of sentences using pre-trained models.
  + Architectural Style: focuses on specific features like sentence embeddings for easy integration).
* transformers Package:
  + Responsibility: Accesses pre-trained models for various natural language processing tasks.
  + Architectural Style: provides modular components that can be combined and customized for NLP tasks

2.2 Deployment Architecture

HTTPS is being used to connect the client web browser to the server because this is a web application and HTTPS is the secure way of accessing our service for the user.

2.3 Data Model

For our data model we are using a vector database to hold relevant information about UNCC from the auxiliary website and other information sources for UNCC. We compile all our data into a .txt file where we vectorize the information then store it in a vector database in pinecone. After this vector database is what the LLM references and uses a similarity search to find relevant information to give to the user. For user login information we are using a firebase database to hold that information in a secure way.

2.4 Global Control Flow

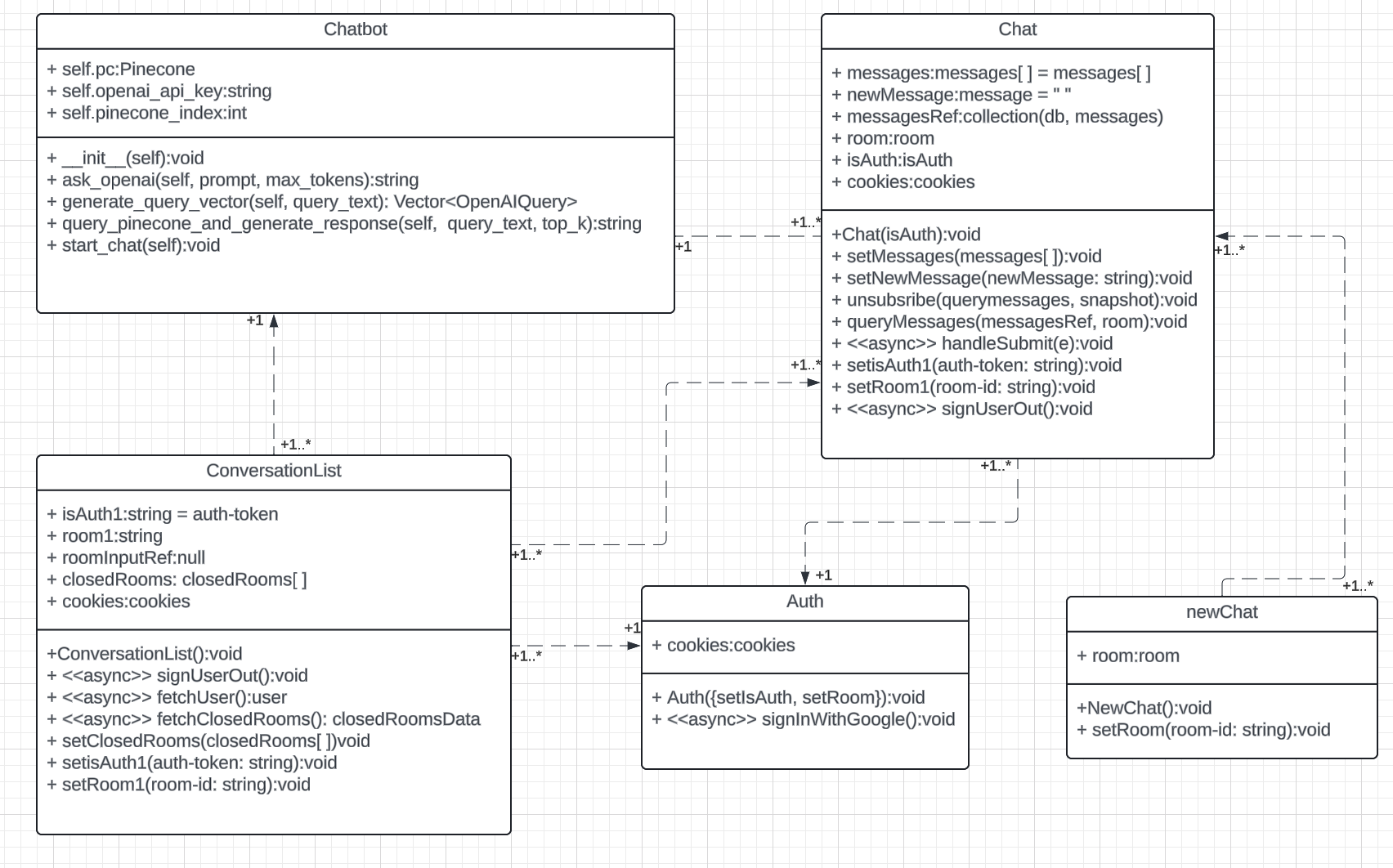
Event-driven: Normchat is an event-driven program because it responds to user inputs/queries rather than following a predefined linear sequence of steps.

Time Dependency: There are no time controlled actions, it responds to user inputs. For example if a user logs in, enters something in the chat, or exits out of the chatbots and wants to be redirected to the feedback page. All these actions are user controlled/event driven without any reliance on time controlled actions.

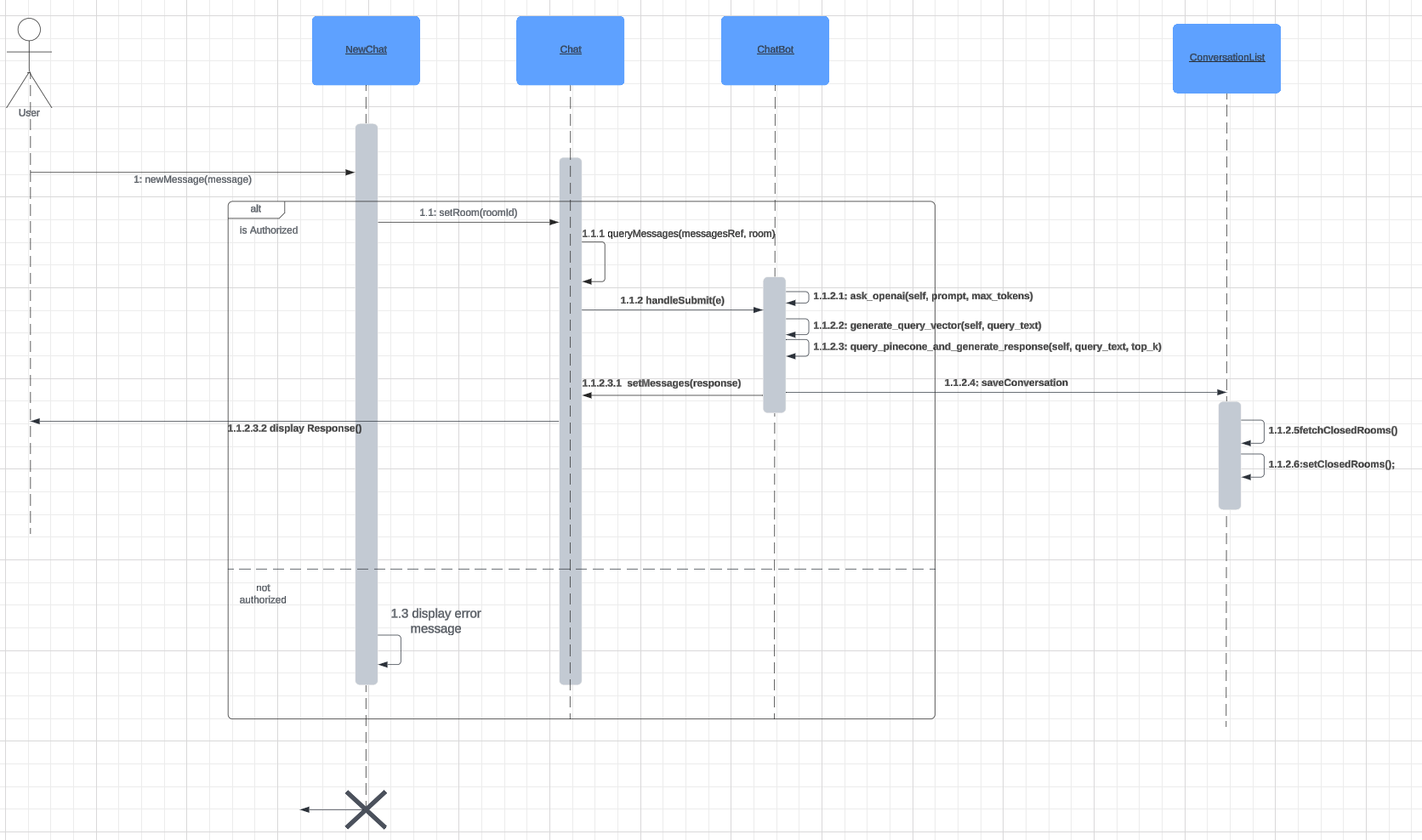
Concurrency: Normchat is designed for handling concurrent user interactions effectively.

3 Detailed System Design

3.1 Static view



3.2 Dynamic view



The design rationale for decomposing modules into classes in the NormChat application, despite it being written in JavaScript using React, revolves around simplicity, functionality, and a cohesive application structure. Here's an expansion based on the provided information:

Module Decomposition and Design Rationale:

1. Simplicity and Functionality:

Using classes in the React environment allows for a structured and organized approach to managing the application's components and logic. While React primarily uses functional components, classes are still utilized for state management, lifecycle methods, and certain functionalities that benefit from object-oriented programming principles.

2. Dependency on Auth Module:

The Auth module plays a critical role in the application as it handles user authentication and authorization. Many functionalities and queries within other modules depend on the Auth module to ensure that users have the necessary permissions and access levels for their actions.

3. Collaboration Among Modules:

The Chatbot, Chat, and ConversationList modules collaborate closely to create a seamless user experience. The flow involves initiating a new chat, which is then stored and managed in the ConversationList. Meanwhile, the Chat module serves as the interface for users to communicate, and the Chatbot module handles the processing of user input, query generation, database querying, and response generation.

4. User Interaction Flow:

-New Chat Initiation:

Users initiate new chats through the Chat module, triggering the creation of a new conversation.

- Conversation Management:

The ConversationList module manages and displays ongoing conversations, providing users with an overview of their chat history.

- Chatbot Integration:

When users interact with the Chat module, their input is received by the Chatbot module. The Chatbot processes this input to generate a query vector.

- Database Querying:

The query vector generated by the Chatbot is used to query the Pinecone database, retrieving relevant information.

- Response Generation:

Based on the database query results, the Chatbot formulates a response, which is then sent back to the Chat module.

- Display for User:

The response generated by the Chatbot is displayed in the Chat module, allowing users to view and engage with the information provided.

This modular approach ensures that each component has a specific responsibility within the application architecture, promoting code reusability, maintainability, and a clear separation of concerns. By leveraging React's component-based structure and utilizing classes where appropriate, the NormChat application achieves a balance between simplicity, functionality, and user interaction flow.

**Submission:**

Submit your design documents inside your team Github repo. Moreover, **submit a PDF document in Canvas along with the link to repo; we will not grade any other document type**. Name your electronic submission as follows: **Team<number>\_D2.pdf.** Submit your team assignment via Canvas. Only one team member needs to submit the document.

**Tips:**

*Provide supplemental text to explain your diagrams through captions for them!* Make sure that you have described, somewhere in the document, the responsibilities of the elements (e.g., components/modules/classes) in your architecture/class/sequence diagrams. You should describe your design decisions that led you to this design, including a discussion of any alternative designs you considered but discarded. Providing these kinds of descriptions helps in understanding your design (as such, they can have a positive impact on your grade!). *Where to Stop/When to Stop Drawing Interaction Diagrams*? Generating a sequence diagram for the most important user stories is typically a good way to start. If you find that you have a bunch of sequence diagrams that essentially repeat the same interaction, then you are not creating useful models, just redundant ones. In this case, generalize the interaction and provide supplemental text that explains how and where the generalized interaction can be applied to capture multiple user stories/use cases. **Remember, you should model only what is needed and useful as discussed in the class.**

*Apply Design Principles!* You have learned about the importance of modules with high cohesion, a design with low coupling, and the benefits of relying on abstraction versus a concrete realization (e.g., application logic communicates with a hardware abstraction layer instead of directly with devices). Make sure that you apply these principles and clearly explain how your design achieves them.

*Proofread your documents!* Everyone on your team should proofread the document before it is submitted. It is important that we be able to understand your design to evaluate it, so you must take care in communicating your design. If you are not skilled in technical writing, plan in advance so that you can ask someone to proofread it for you. The university has a writing resource center that may be helpful to you.

*What UML tools should I use to draw the diagrams?* Any design tool may be used to draw your UML diagrams such as Draw.io and Lucidchart as we discussed in the class.

Be aware that while drawing programs may provide template tools for creating UML diagrams, those templates may not meet the diagramming guidelines we discussed in class. For instance, some versions of Microsoft Visio provided the wrong arrow for an “extends” relationship in the past for use cases. You should check to make sure your diagram meets the standards discussed in class and make manual edits in the drawing program if necessary to adhere to those standards.