NormChat Software Design Document

Group 9 - ITCS 4155

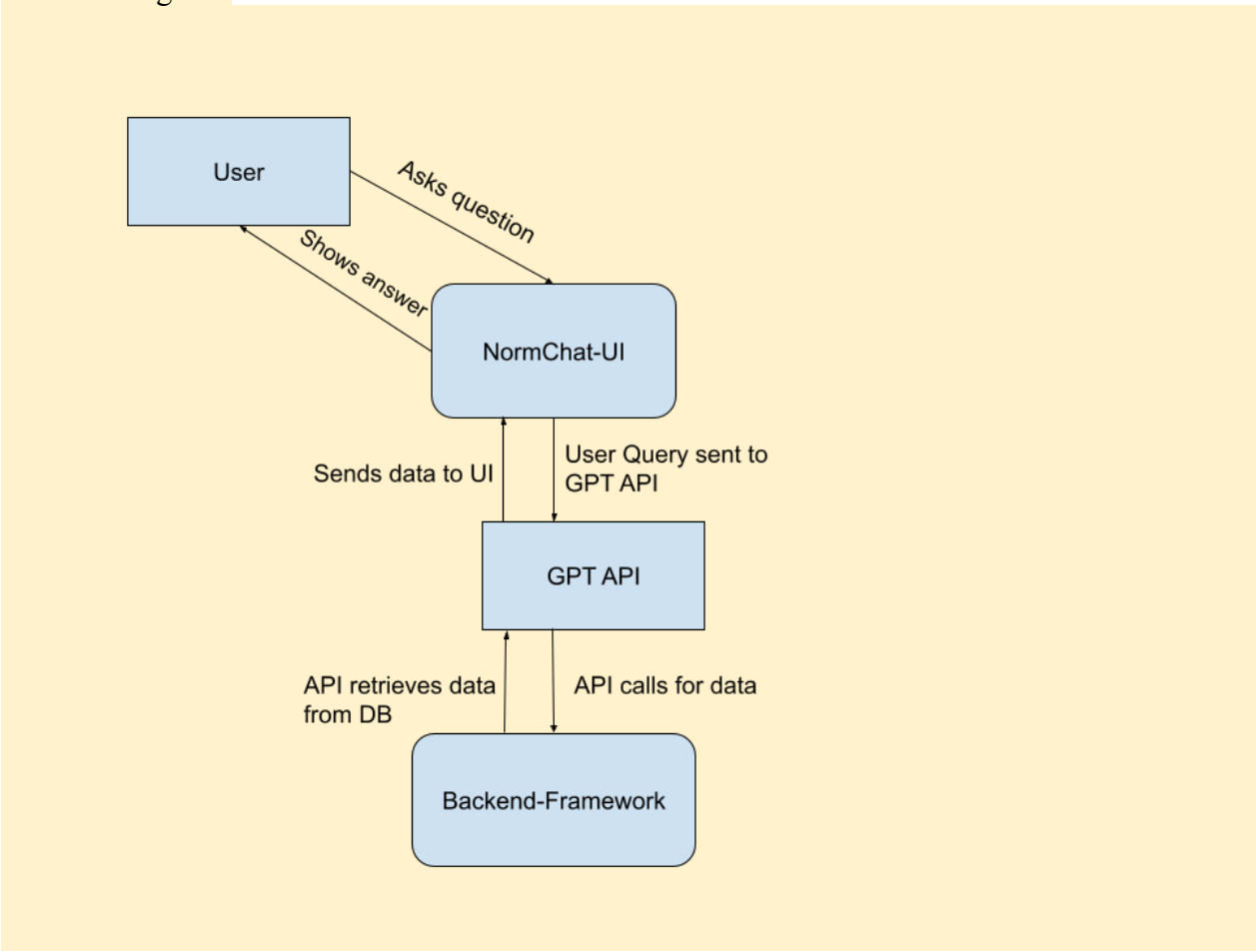
The purpose of the software design document (SD^2) is to provide an overview of the system and to help people understand the system. This document can be used by a programmer as a guideline for implementing the design without needing to make significant engineering design decisions, and will be provided 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 Navigating the Charlotte website can be frustrating, with the outdated user interface and important information covered under tabs among tabs.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 NinerChat to get their questions answered in seconds.

General Model -

Context Diagram:



User Stories:

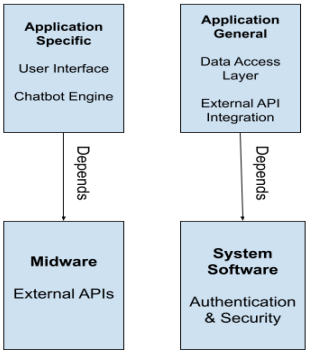
1. Front-end: As a student I need to know specific prerequisites before registration so that I can register for classes easily, avoiding confusion. Back-end: User should be able to input specific class information into chatbox, output should display class information, filtering redundancy based on specific 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

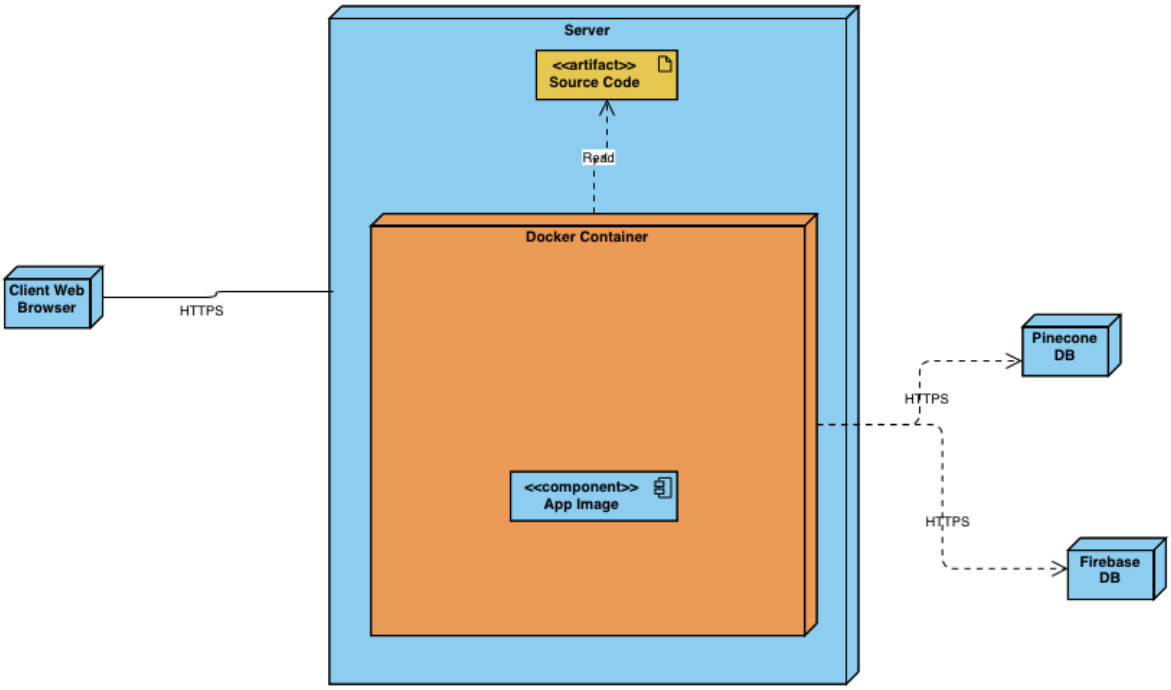
2. Architectural Overview

Initially, we began using an open source repository for Norm Chat's framework. We did this in order to expedite the processes by utilizing existing resources. However, several factors forced us to change our approach. Firstly, we found that some of the open source code was redundant and was leading to conflicts and inefficiencies without project requirements. Secondly, a lot of the code was too complex and many of us did not have certain skills that we would require to comprehend the code. BotSharp-UI https://github.com/SciSharp/BotSharp-UI
NormChat is designed for use on desktop computers, however it is also responsive for use on tablets or smartphones. Since NormChat is a chat-bot AI application, we thought it would be best to utilize a web-based architecture. This allows users to easily navigate and use all of the features that NormChat has to offer. For our website we used React as the framework.
Open Source Code we decided not to go with:

2.1 Subsystem Architecture



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.

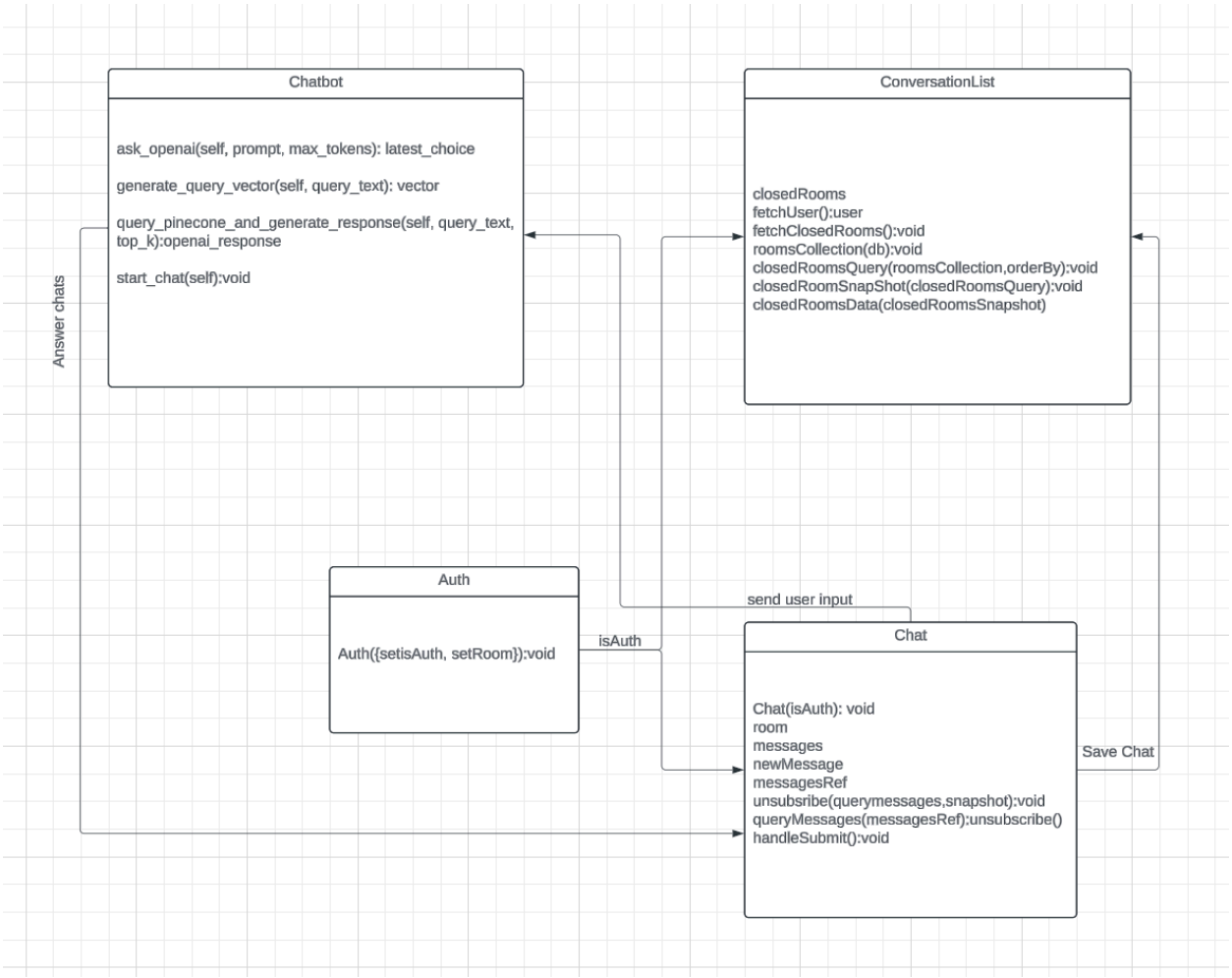
After this vector database is what the LLM references and uses a similarity search to find relevant information to give to the user. For user login information we are using a firebase database to hold that information in a secure way.

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.

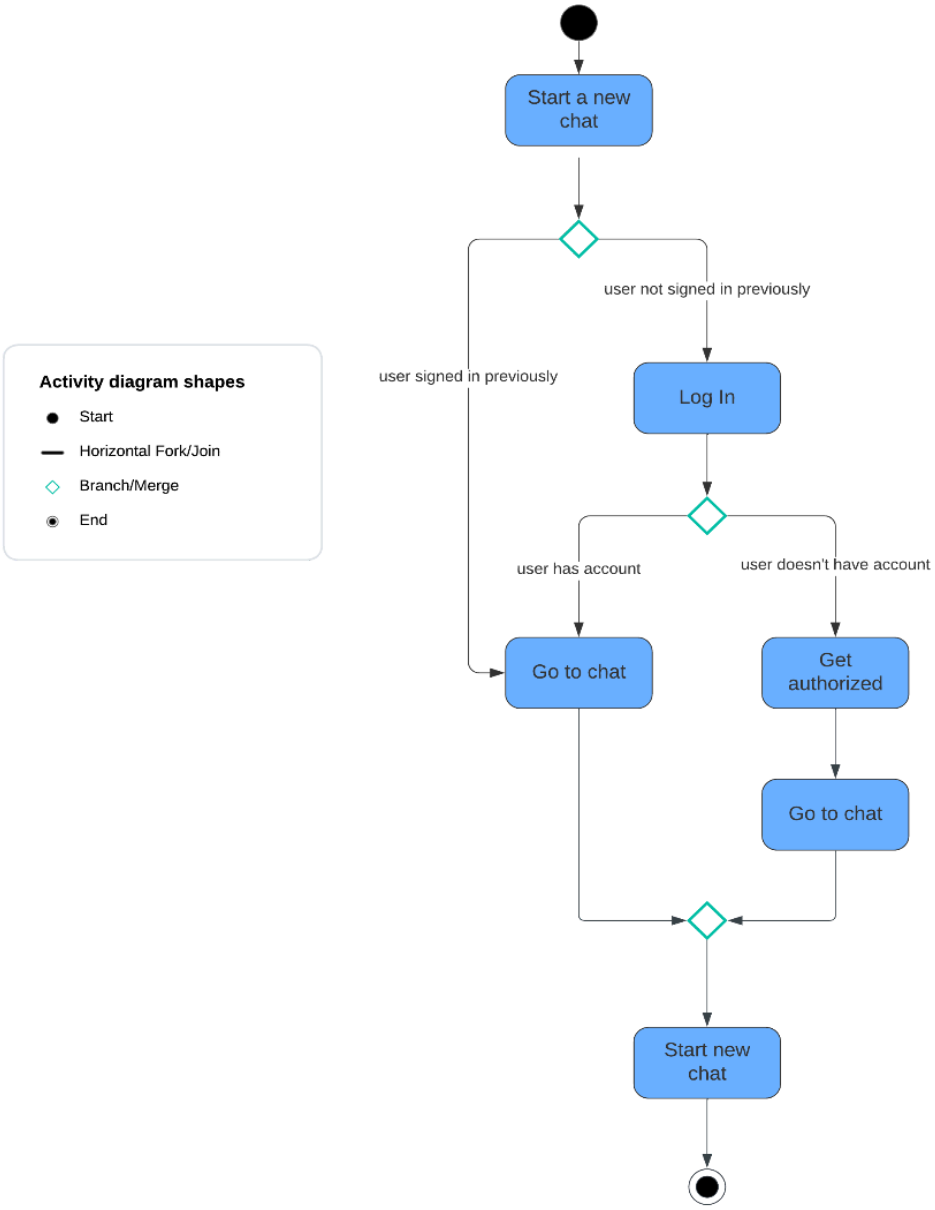
3 Detailed System Design

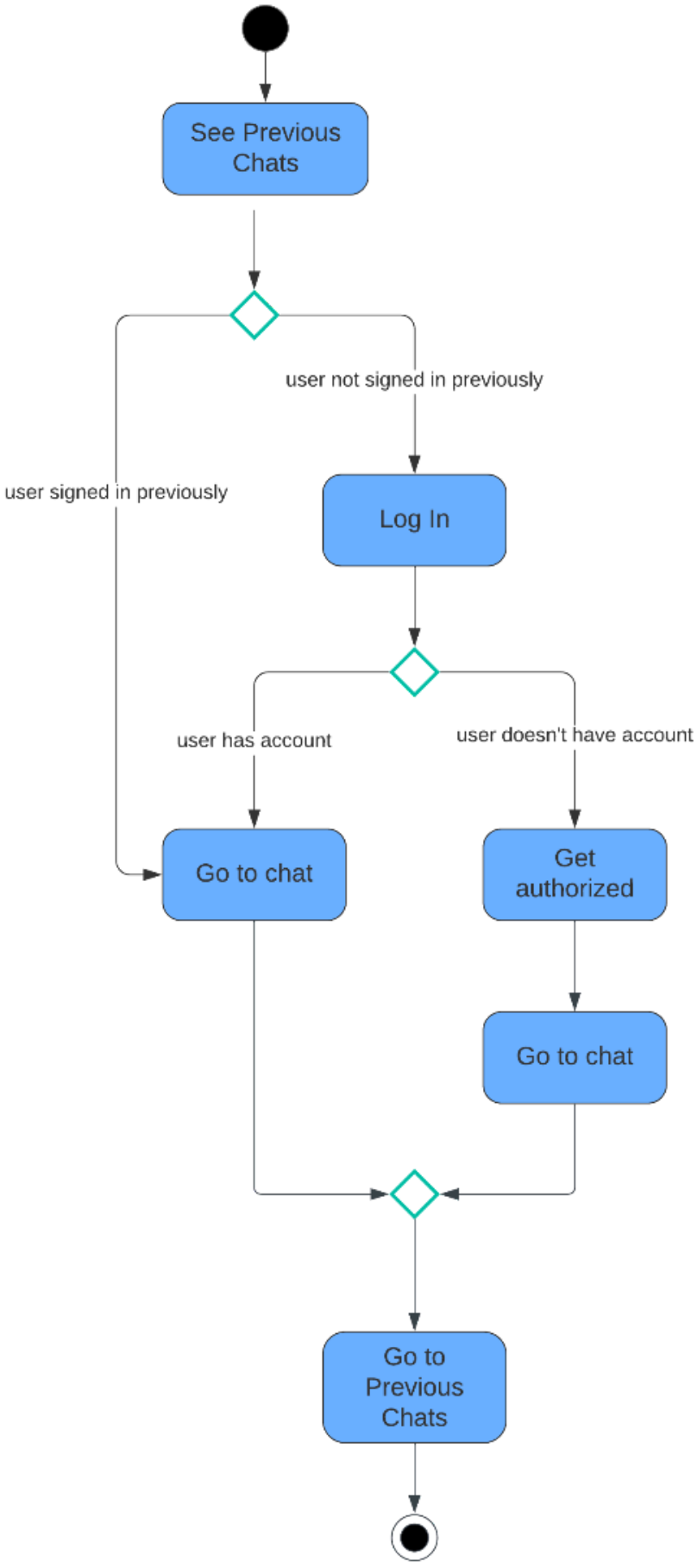
3.1 Static view

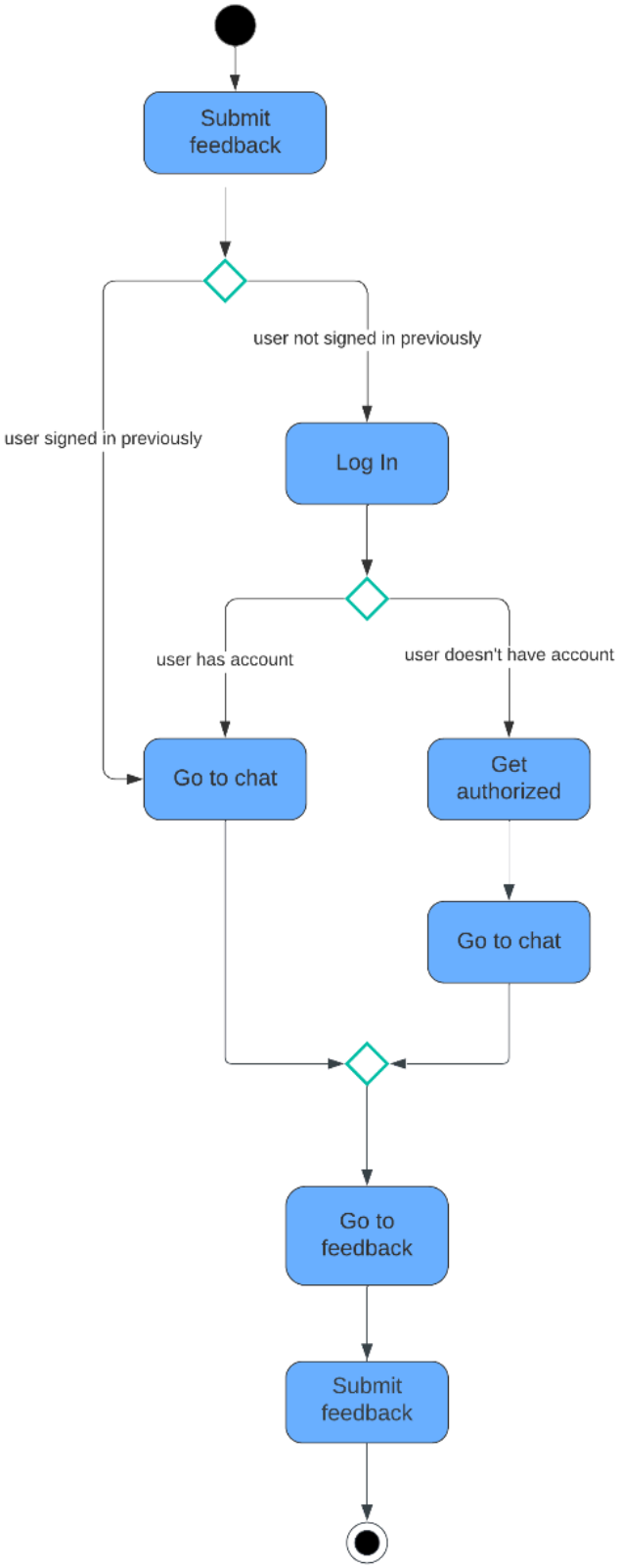


This decomposition of modules into classes was chosen for its simplicity and functionality. NormChat does not have that many specific classes, due to it being written in JS in React. However, there are many functions and implementations that act as classes and transfer data accordingly. For the most part, all of the modules depend on the Auth module, as it is required for most queries and functionalities. The Chatbot, Chat, and ConversationList all work together to provide a cohesive application. The user starts a new chat, which is then saved in the conversation list, and the chatbot receives the user's input from the chat.

3.2 Dynamic view




See Previous Chats
user not signed in previously
user signed in previously
Log In
user has account
user doesn't have account
Go to chat
Get authorized
Go to chat
Go to Previous Chats


Submit feedback
user not signed in previously
user signed in previously
Log In
user has account
user doesn't have account
Go to chat
Get authorized
Go to chat
Go to feedback
Submit feedback

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.