Product Design

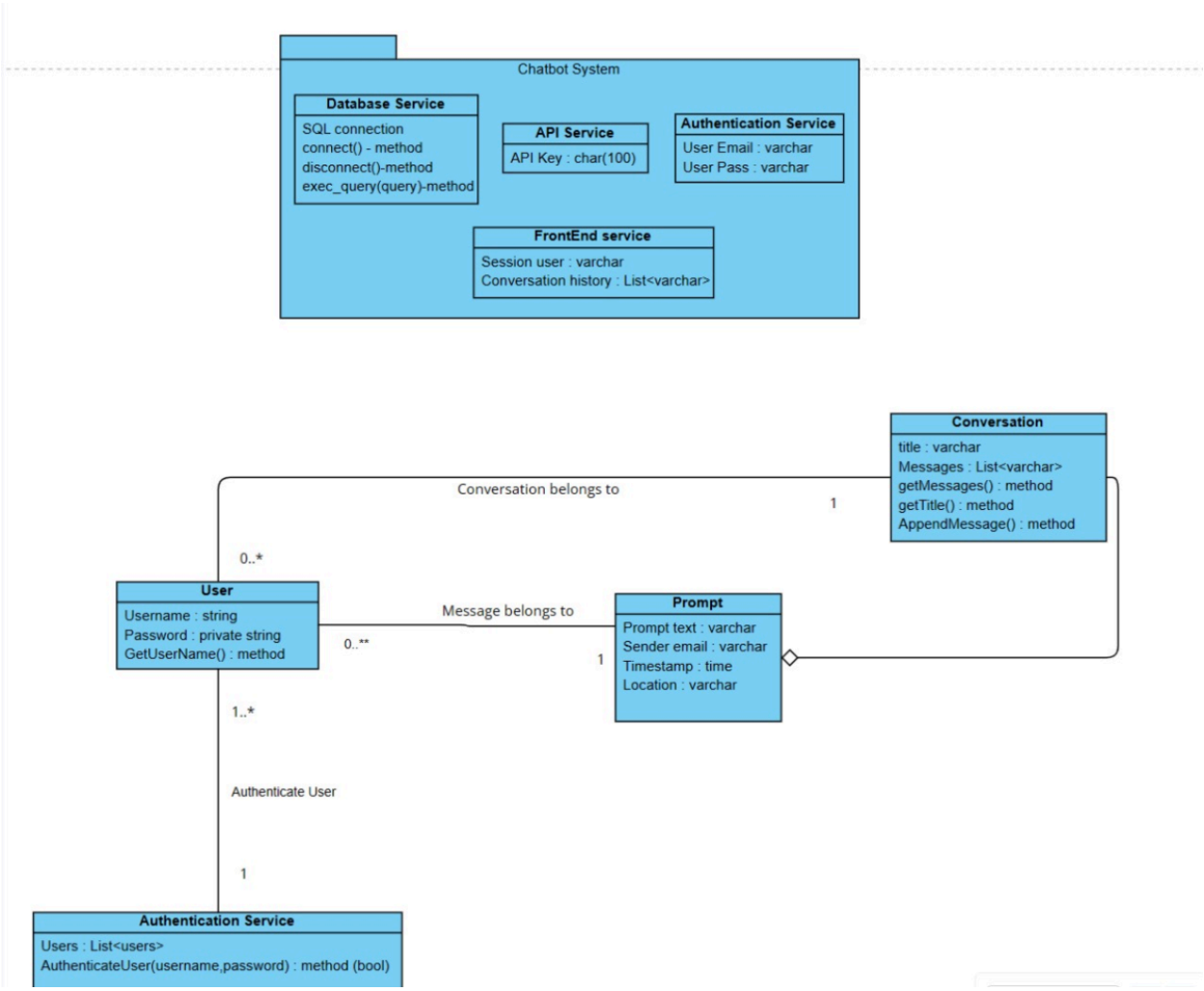
Team 14 (Ayush Sahu, Divyansh Pandey, Harshit Gupta, Rahul Singal)

Design Model

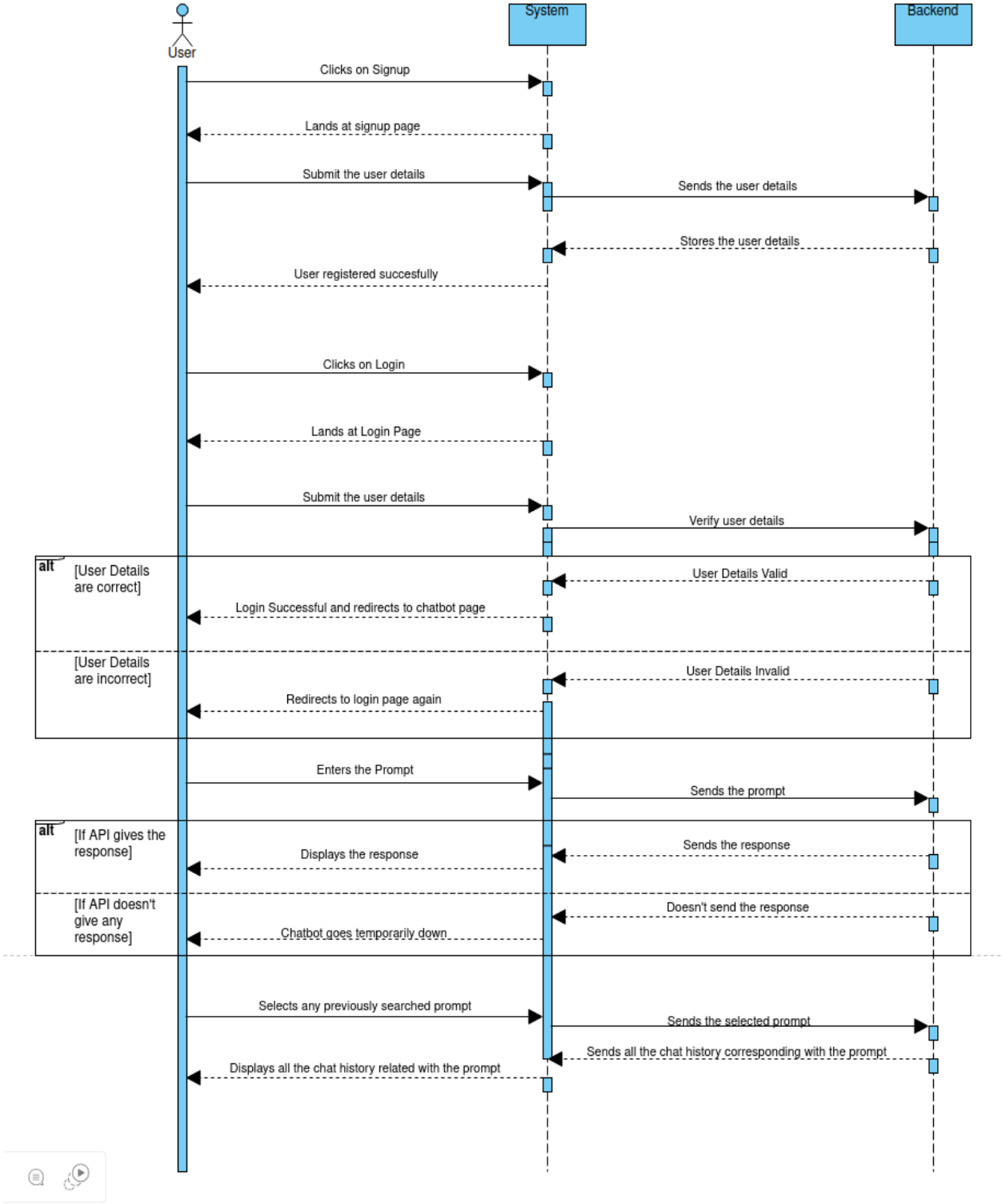
<Users>	Class state : · Details about the user including his credentials Class behavior: ● GetUserName() method is used to retrieve email address of the user to be displayed on the UI.
<Prompt>	Class state: Details about a single prompt instance corresponding to a user Class behavior: · This class has no public methods, it is used to track prompts
<Conversation>	Class state : Details about a conversation history for a particular user. Class behavior · Methods getMessage() , getTitle() , AppendMessage() are used to retrieve information about a conversation or add a new message(prompt) to the conversation history.
<Authentication Service>	Class state : Used to authenticate a user using his credentials by verifying the credentials through database Class behavior : AuthenticateService method() is used to access the service and authenticate user to give chatbot access.
<Database Service>	Class state : Used to store prompt history and user credentials. Class behavior :

	Accessed through a connection instance to the database and queries to fetch necessary records
<FrontEnd service>	Class state : FrontEnd module used to display the prompts and conversations of a user Class behavior : Encompasses rendering methods of each module.
<Chatbot Service>	Class state : Superclass encompassing all modules of the system as attributes which can be monitored in this class. Class behavior : Parent class consisting of subclasses.

UML Class Diagram



Sequence Diagram(s)



Design Rationale

The concept for the website involves creating a comprehensive chatbot integrated with authentication. Upon landing on the homepage, users are presented with the option to either log in or sign up. To enhance the visual appeal, a captivating typing effect on the left side of the screen showcases potential prompts users can enter.
Upon choosing to sign in, users input the required details and, upon submission, are redirected to the login page. If the user has an existing account, entering correct credentials leads them directly to the chatbot, while incorrect ones trigger an error message.
In the chatbot interface, a dynamic left side features a "New Chat" button that, when clicked, generates and stores the original chat. This history is conveniently accessible in the sidebar for users to revisit if needed. The send button remains disabled until the user inputs a prompt, providing a seamless and responsive experience.

To distinguish between user and chatbot responses, the design incorporates various color variations. While there was initially a proposal to exclude authentication, a thorough team analysis concluded that its inclusion is more beneficial. Additionally, the typing effect initially exclusive to the landing page has been extended to the login and sign-up pages, contributing to a cohesive and engaging user experience.

Recently we have added a section of user settings that allows the user to log out of the chatbit and goes directly to the landing page.
The framework used is Django, which has been implemented as per the client’s needs. Using Django we have created routes to the several pages of the website and integrate the Gemini API into our UI.We have also implemented storing the history in a text file with its help.

The latest updates include displaying the user's name alongside their initials, modifying the logout button to allow logging out by clicking on the user profile. Additionally, a loading animation now appears when the user sends a prompt and stops once the response is generated.
An "About Us" page has been added to the landing page, providing information on the chatbot's purpose.
Most significantly, the chatbot now filters for COVID-19 related context. It specifically addresses queries related to COVID-19. If a user attempts to input irrelevant prompts, an error message is displayed, indicating that the chatbot cannot answer the query as it is out of context.