# CSC207 Final Project

# Character Chatbot

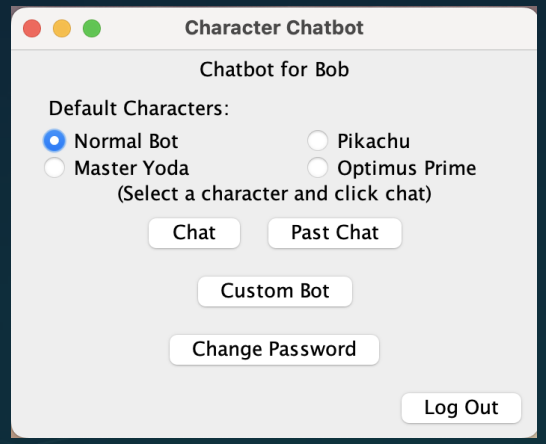
Group #199
Thomas Swanick, Yukun Wang, Yi Pan, John Deng

# PROJECT OVERVIEW

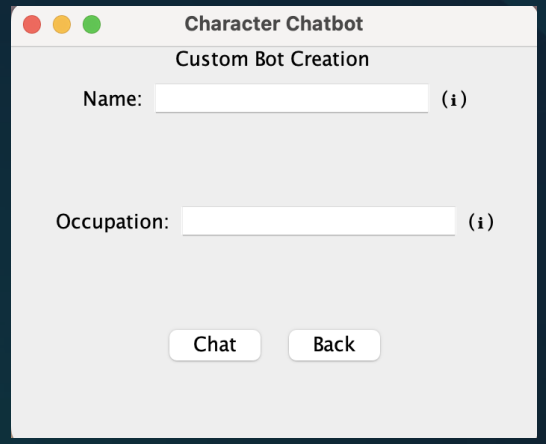
# Project Specification

**Our Goals:**

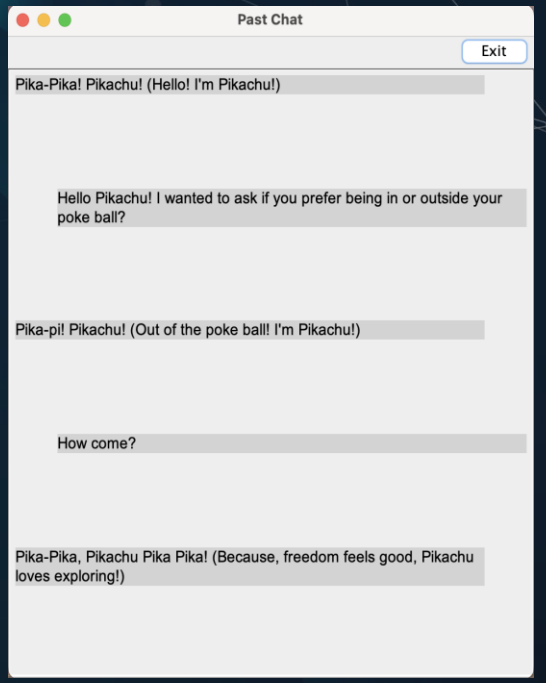
- Deliver natural, immersive, and context-aware dialogues through various chat bots.
- Engage users with iconic characters like Yoda, Pikachu, and Optimus Prime.
- Inspire creativity through unlimited character customization.




chat with **Default Characters**




create your own **Custom Characters**




view your **Past Chat**

# API Usage

## GPT API

```
private static final String API_URL = "https://api.openai.com/v1/chat/completions";
```

```
public String getChatbotResponse() throws Exception {
    final String jsonInputString = "{\"model\": \"gpt-4\", \"messages\": " + buildMessagesJson() + "}";
    final URL url = new URL(API_URL);
    final HttpURLConnection connection = (HttpURLConnection) url.openConnection();
    connection.setRequestMethod("POST");
    connection.setRequestProperty("Authorization", "Bearer " + API_KEY);
    connection.setRequestProperty("Content-Type", "application/json");
    connection.setDoOutput(true);
```

## MongoDB

```
public MongoDBDataAccessObject() {
    String connectionString = "mongodb+srv://jda1234112:dcJSlP8XfESt9FED@cluster0."
            + "2mrsi.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0";
    mongoClient = MongoClients.create(connectionString);
    databaseForHis = mongoClient.getDatabase("ChatHistory");
    database = mongoClient.getDatabase("chatbotDB");
    chatCollection = databaseForHis.getCollection("chathistory");
    userCollection = database.getCollection("userAccounts");
    chatbot = database.getCollection("character");
    chatStatus = Boolean.TRUE;
    this.currentUsername = null;
}
```

# USER STORIES

# Yi Pan

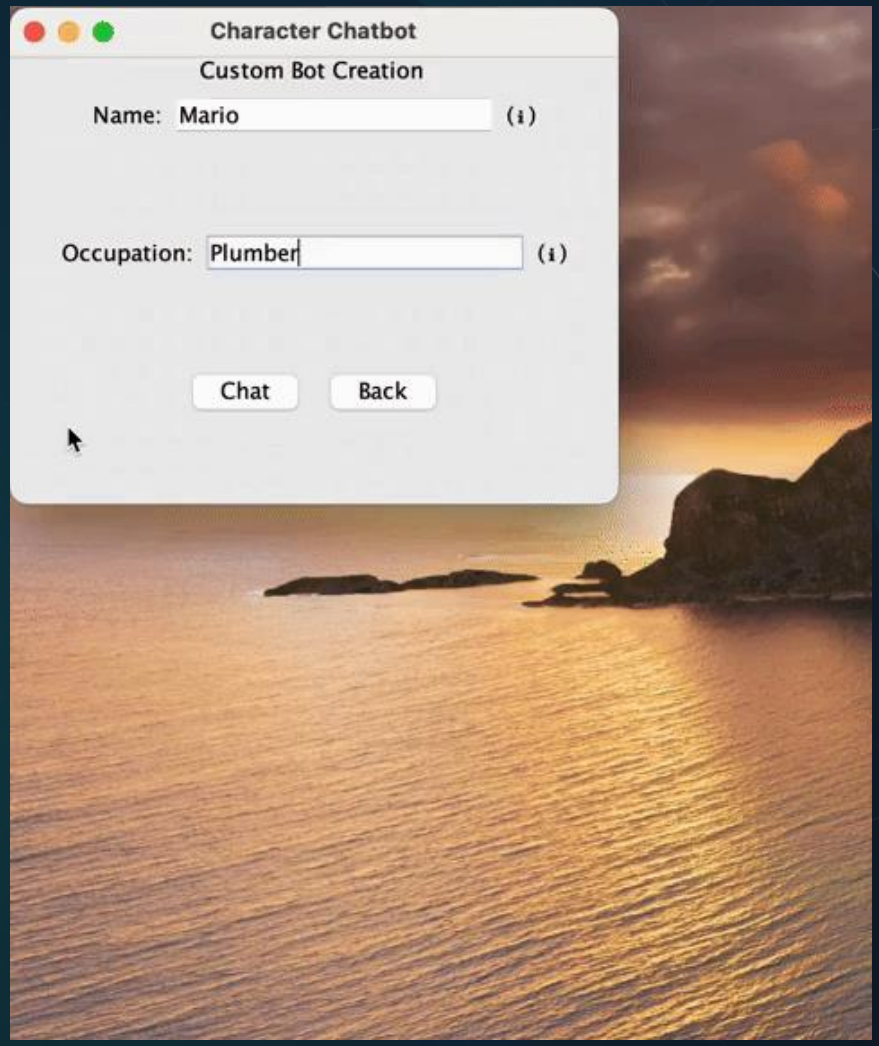
## User Story:

- A user wants to create a custom bot with a name and occupation of their choosing.

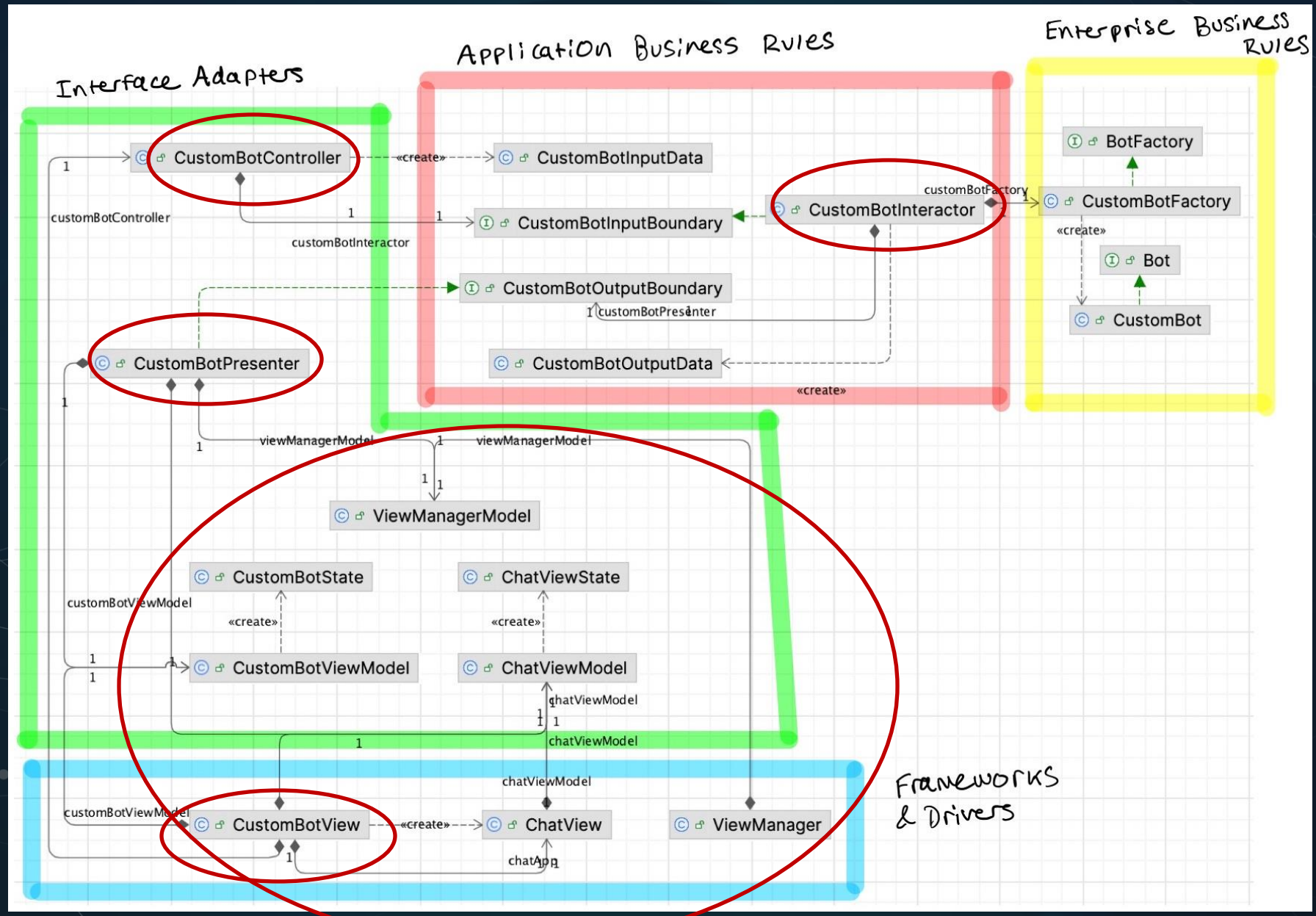
## Chosen Use Case:

- Starting a new chat with a custom bot.

## Before and After View:



# Yi Pan


Interface Adapters
Application Business Rules
Enterprise Business Rules
CustomBotController
«create»
CustomBotInputData
customBotController
customBotInteractor
CustomBotInputBoundary
CustomBotInteractor
customBotFactory
BotFactory
CustomBotFactory
«create»
Bot
CustomBot
CustomBotOutputBoundary
customBotPresenter
CustomBotOutputData
«create»
CustomBotPresenter
viewManagerModel
viewManagerModel
ViewManagerModel
CustomBotState
ChatViewState
customBotViewModel
«create»
«create»
CustomBotViewModel
ChatViewModel
chatViewModel
chatViewModel
chatViewModel
customBotViewModel
CustomBotView
«create»
ChatView
ViewManager
chatApp
Frameworks & Drivers

# Yi Pan

## Main Code from the CustomBotInteractor class:

```
    @Override
    public void execute(CustomBotInputData customBotInputData) {
        // retrieves the current user's username, custom bot name, and occupation from the input data
        final String username = customBotInputData.getUsername();
        final String botName = customBotInputData.getBotName();
        final String occupation = customBotInputData.getBotOccupation();

        // creates a custom bot object and the relevant backend prompt to start the chat
        // based on the input data.
        final String prompt = customBotFactory.create(botName, occupation).getPrompt();

        // Tell the presenter to signal the beginning of a new custom bot chat with
        // this information.
        final CustomBotOutputData customBotOutputData = new CustomBotOutputData(username, prompt);
        customBotPresenter.beginChat(customBotOutputData);
    }
}
```

# Thomas Swanick

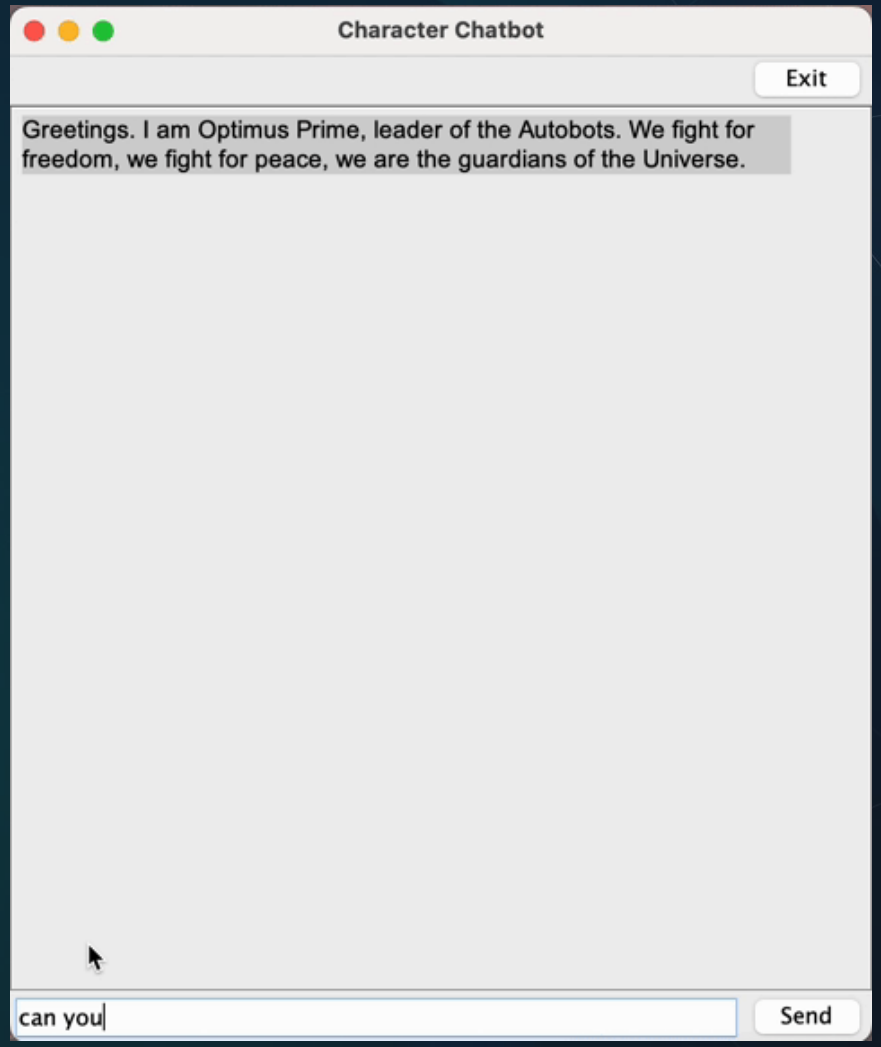
## User Stories:

- A user wants to be able to create an account, log in with this account and change their password if necessary.
- A user wants to be able to chat with the Optimus Prime default bot.

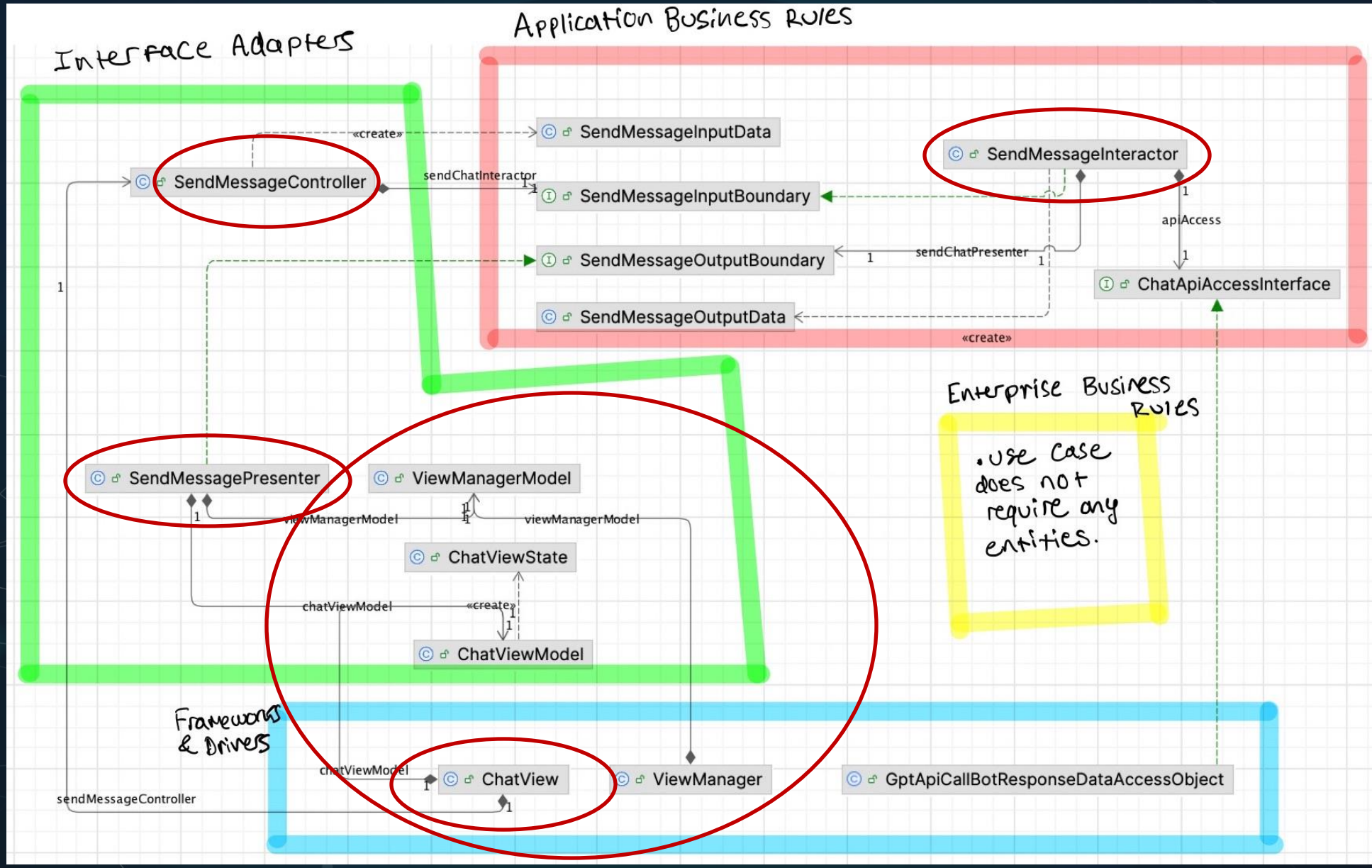
## Chosen Use Case:

- Send a message within a currently ongoing chat.

## Before and After View:



# Thomas Swanick



# Thomas Swanick

## Main Code from the SendMessageInteractor class:

```java
@Override
public void execute(SendMessageInputData sendMessageInputData) {
    // Get the user input from the input data and send it to the
    // gpt api to add it to the current conversation history.
    final String userInput = sendMessageInputData.getUserInput();
    apiAccess.addMessageToHistory("user", userInput);

    try {
        // retrieve the bot's response to the above user input from the api.
        final String response = apiAccess.getChatbotResponse().replace("\n", "");
        // add the bot's response to the current conversation history.
        apiAccess.addMessageToHistory("assistant", response);

        final boolean responseError = false;
        final String botResponse = response;

        // Pass the response to the presenter, through the output data.
        final SendMessageOutputData sendMessageOutputData = new SendMessageOutputData(botResponse, responseError);
        sendChatPresenter.returnBotResponse(sendMessageOutputData);

    } catch (Exception ex) {
        // If we cannot retrieve the bot's response, then "simulate" a response error message.
        final boolean responseError = true;
        final String botResponse = "Error: Unable to get response from GPT.";
        final SendMessageOutputData sendMessageOutputData = new SendMessageOutputData(botResponse, responseError);
        sendChatPresenter.returnBotResponse(sendMessageOutputData);
    }
}
```

# Yukun Wang

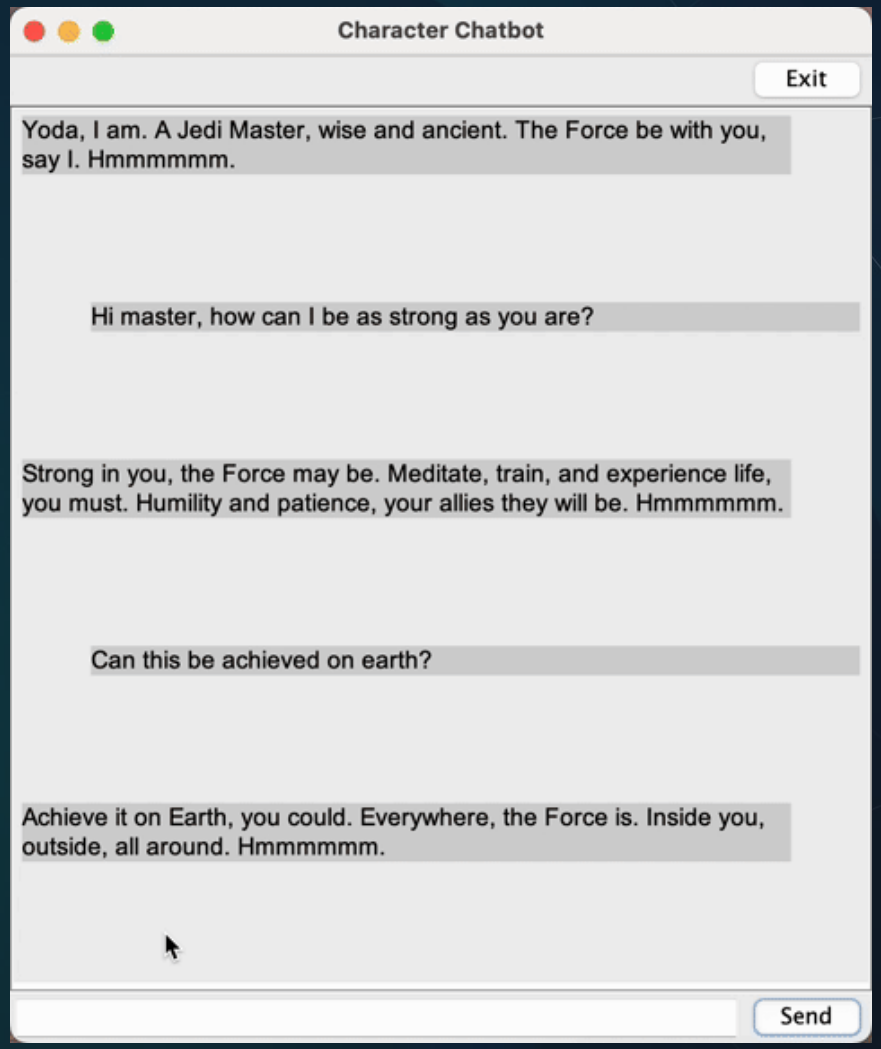
## User Stories:

- A user wants to be able to chat with the Master Yoda default bot.
- A user wants to be able to chat with the Pikachu default bot.

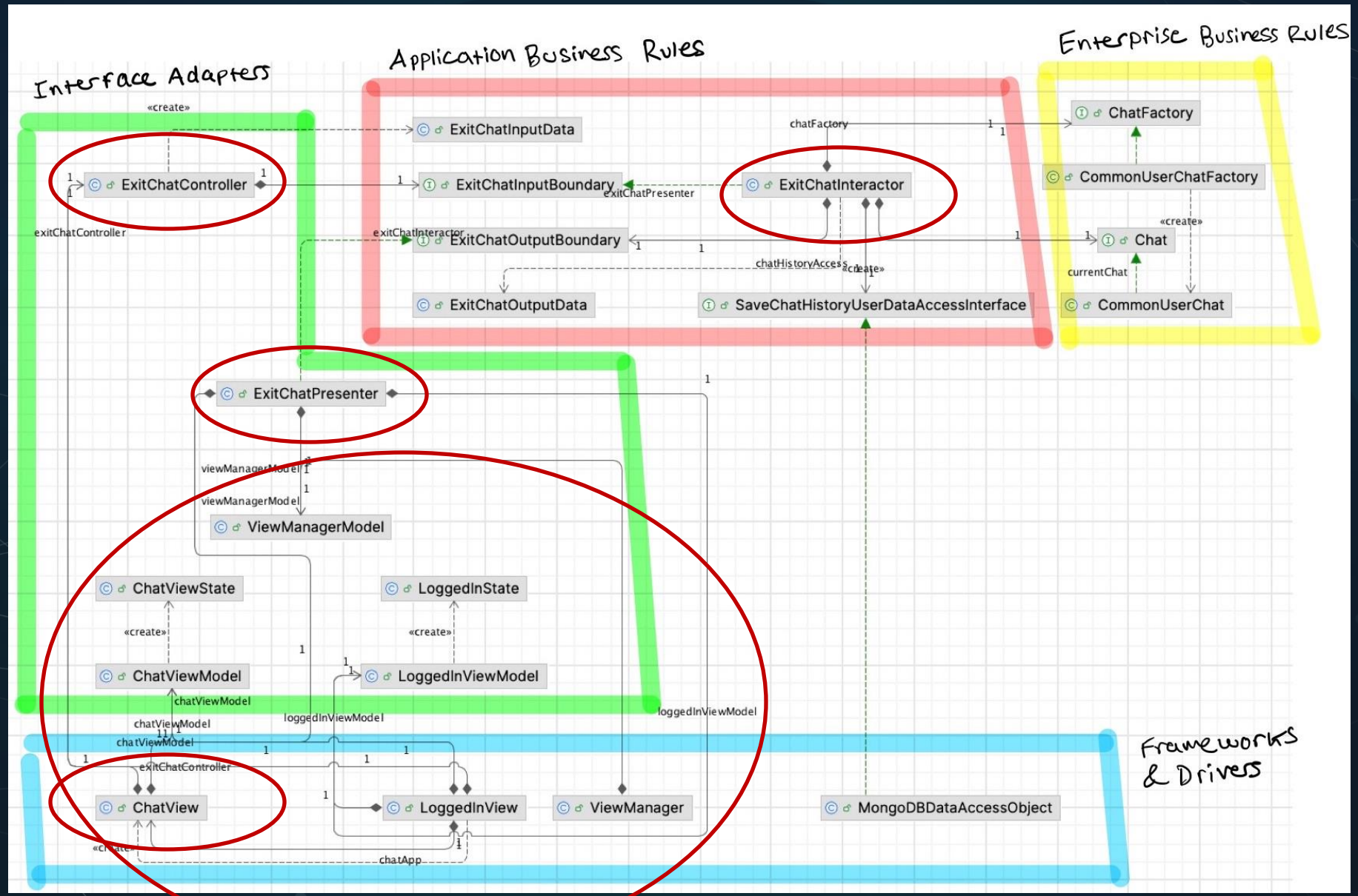
## Chosen Use Case:

- Exit and end a currently ongoing chat.

## Before and After View:



# Yukun Wang


Interface Adapters
Application Business Rules
Enterprise Business Rules
Frameworks & Drivers
«create»
ExitChatInputData
ExitChatController
ExitChatInputBoundary
ExitChatInteractor
chatFactory
ChatFactory
CommonUserChatFactory
exitChatPresenter
exitChatController
exitChatInteractor
ExitChatOutputBoundary
Chat
chatHistoryAccess
currentChat
ExitChatOutputData
SaveChatHistoryUserDataAccessInterface
CommonUserChat
ExitChatPresenter
viewManagerModel
ViewManagerModel
ChatViewState
LoggedInState
ChatViewModel
LoggedInViewModel
chatViewModel
loggedInViewModel
ChatView
LoggedInView
ViewManager
MongoDBDataAccessObject
chatApp

# Yukun Wang

## Main Code from the ExitChatInteractor class:

```java
@Override
public void execute(ExitChatInputData exitChatInputData) {
    // Get the current user from the input data.
    final String username = exitChatInputData.getUsername();

    // Get all the user inputs and bot responses from the current chat.
    final List<Message> lst = currentChat.getUserInputs();
    final List<Message> lst2 = currentChat.getBotResponses();

    // Save this chat (bot responses and user inputs) to the DB.
    for (int i = 0; i < lst.size(); i++) {
        chatHistoryAccess.saveHistory(username, lst.get(i).getContent());
        chatHistoryAccess.saveHistory("assistant", lst2.get(i).getContent());
        System.out.println(lst.get(i).getContent());
        System.out.println(lst2.get(i).getContent());
    }
    System.out.println(username);

    // Tell the presenter to update the views so that the chat "exits".
    final boolean endChat = true;
    final ExitChatOutputData exitChatOutputData = new ExitChatOutputData(endChat);
    exitChatPresenter.endChat(exitChatOutputData);
}
```

# John Deng

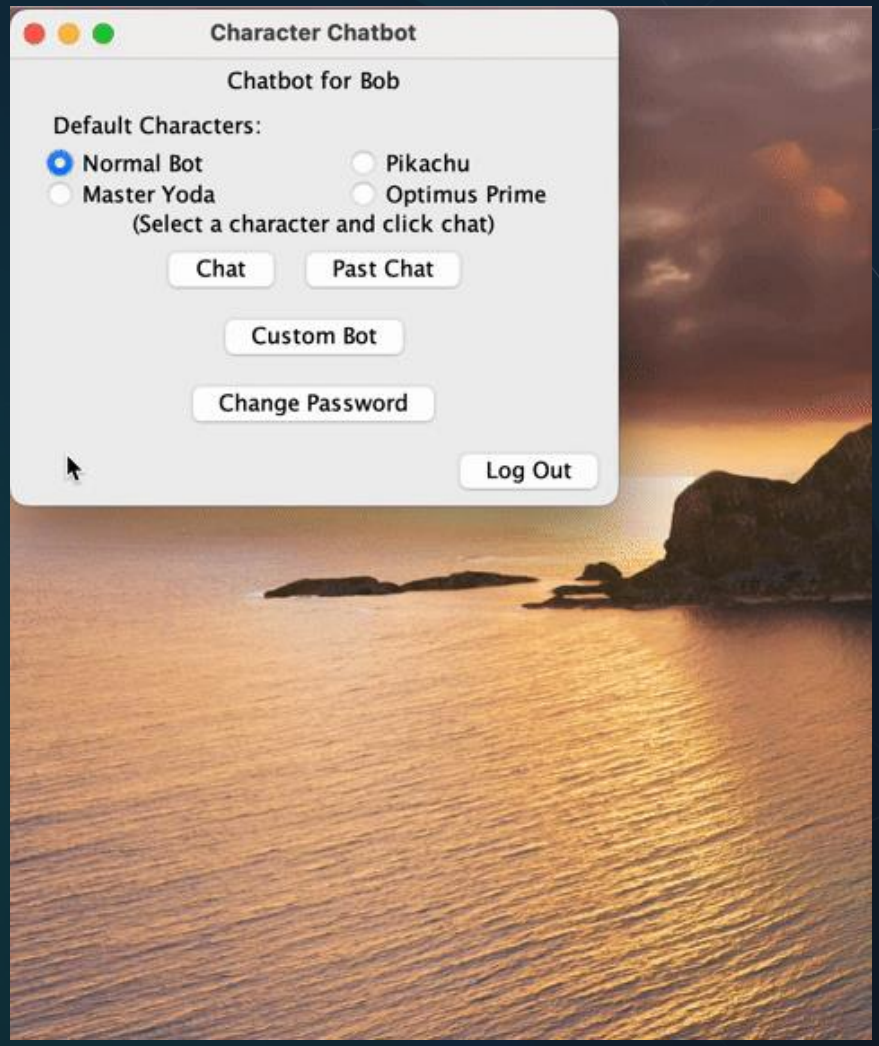
## User Story:

- A user wants to be able to see their past chat, no matter how much time has passed.

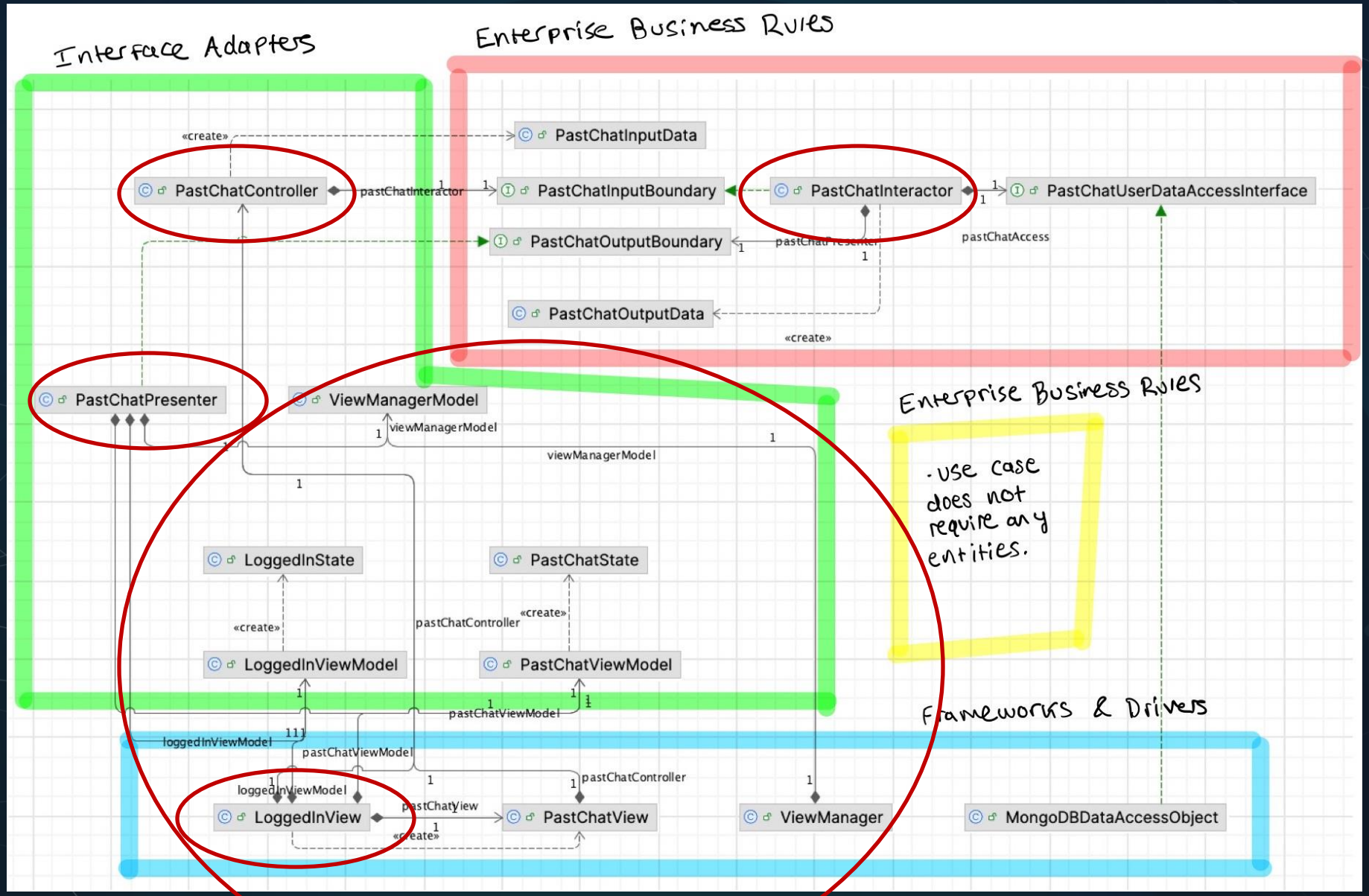
## Chosen Use Case:

- View your past chat.

## Before and After View:



# John Deng


Interface Adapters
Enterprise Business Rules
«create»
PastChatInputData
PastChatController
pastChatInteractor
PastChatInputBoundary
PastChatInteractor
PastChatUserDataAccessInterface
PastChatOutputBoundary
pastChatPresenter
pastChatAccess
PastChatOutputData
«create»
PastChatPresenter
ViewManagerModel
viewManagerModel
viewManagerModel
Enterprise Business Rules
- use case does not require any entities.
LoggedInState
PastChatState
«create»
pastChatController
«create»
LoggedInViewModel
PastChatViewModel
pastChatViewModel
loggedInViewModel
pastChatViewModel
Frameworks & Drivers
loggedInViewModel
pastChatController
pastChatView
LoggedInView
PastChatView
ViewManager
MongoDBDataAccessObject
«create»

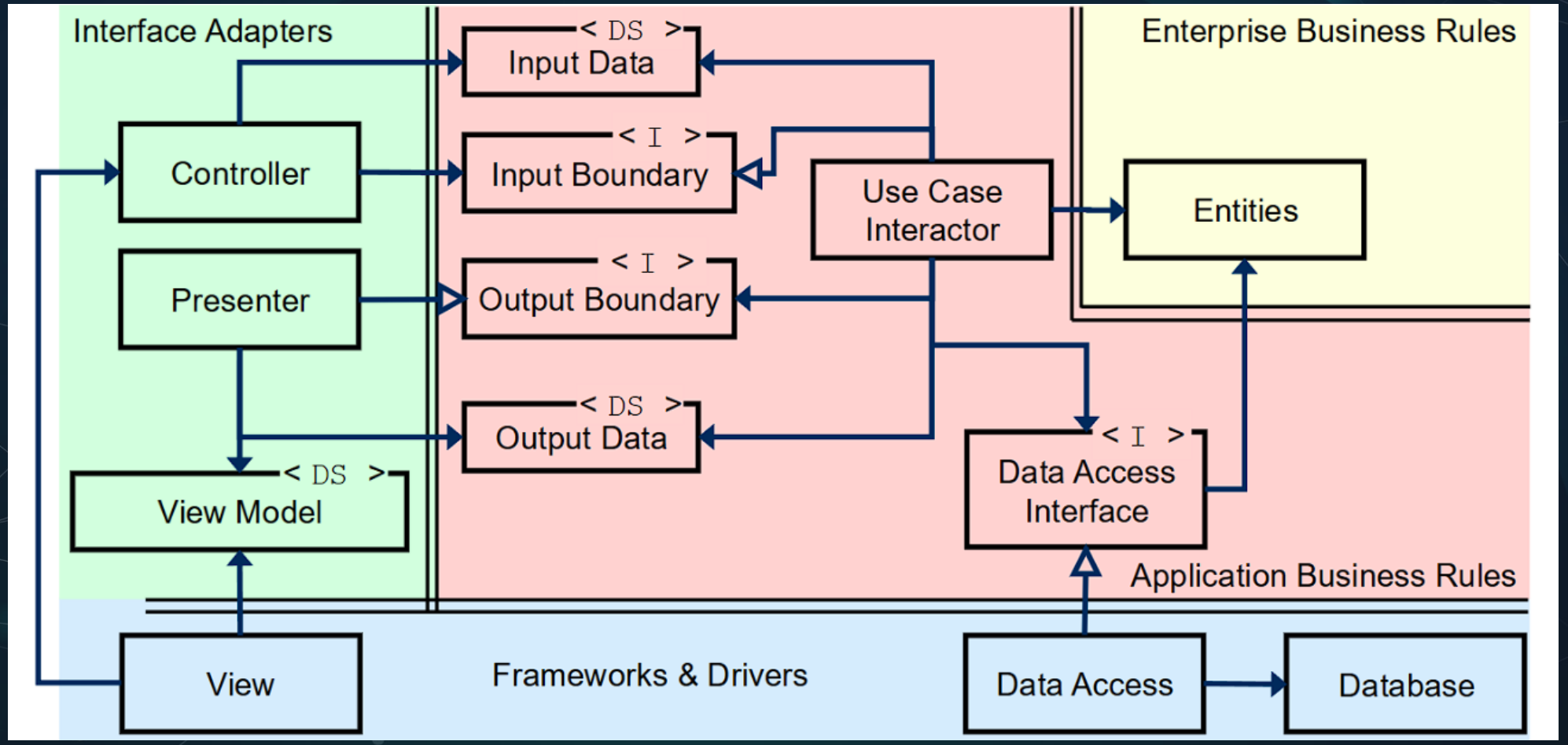
# John Deng

## Main Code from the PastChatInteractor class:

```
@Override
public void execute(PastChatInputData pastChatInputData) {
    // get the user's username from the input data.
    final String username = pastChatInputData.getUsername();

    // Get the past chat bot responses and user inputs corresponding to this user from the DB.
    // Store them as a list of strings for the output data.
    final List<Message> lst = pastChatAccess.mixedHistory(username);
    final List<String> pastChatMessagesInOrder = new ArrayList<String>();

    for (int i = 0; i < lst.size(); i++) {
        pastChatMessagesInOrder.add(lst.get(i).getContent());
    }

    final PastChatOutputData pastChatOutputData = new PastChatOutputData(pastChatMessagesInOrder);
    pastChatPresenter.presentPastChat(pastChatOutputData);
}
```

# Design, Organization, & Code Quality

# Clean Architecture




The CA structure followed for each use case

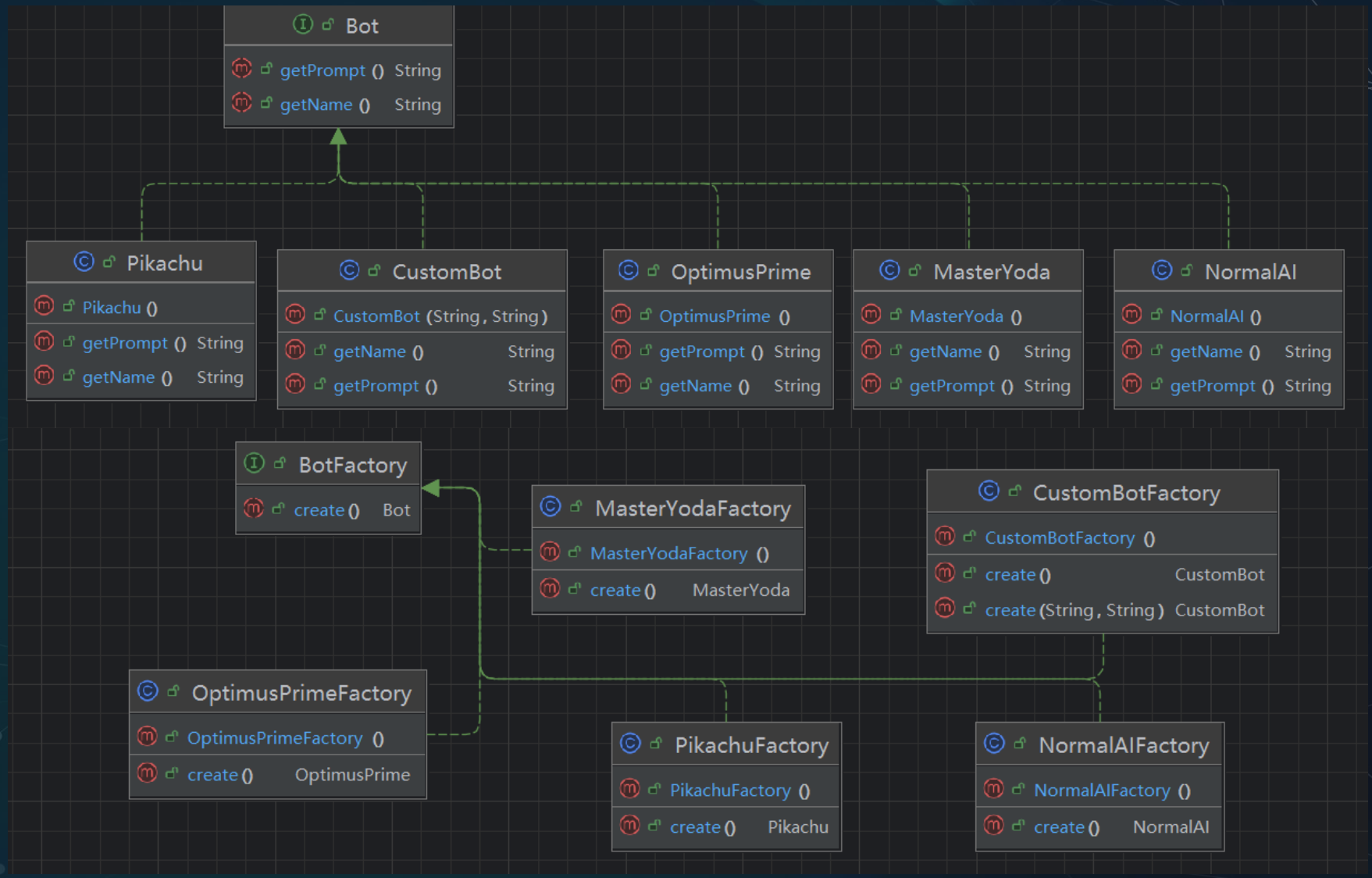
# SOLID Principles

• Open/Closed Principle:

We can add any number of new bots.
(open for extension)

No need to modify the Bot and BotFactory interfaces. Nor the classes that depend on them, Including the pre-exisiting Bot types.
(closed for modification)

Allows for the continuous development of new bots and thus, the future expansion of Character Chatbot.

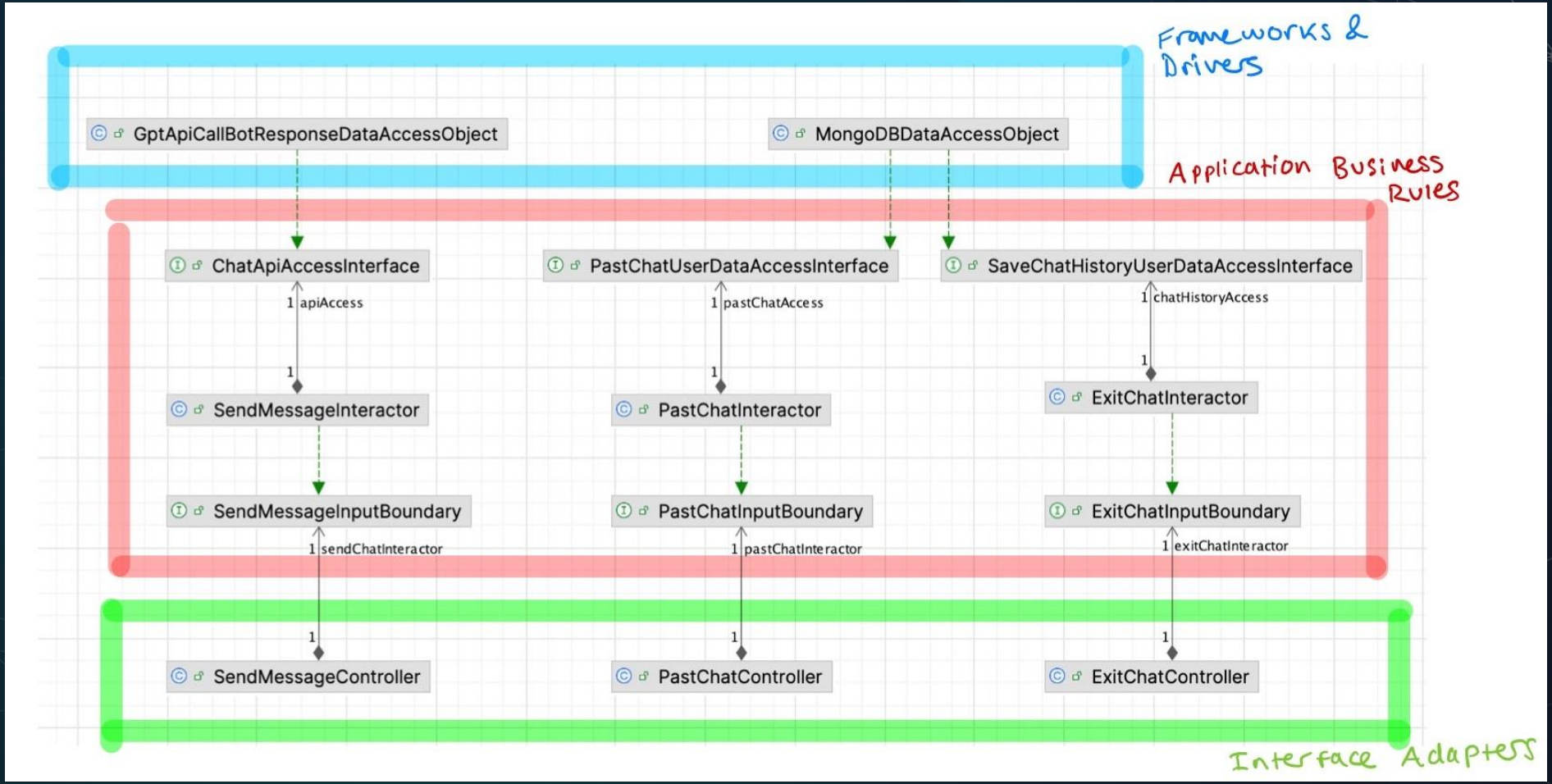


# SOLID Principles

- Dependency Inversion:

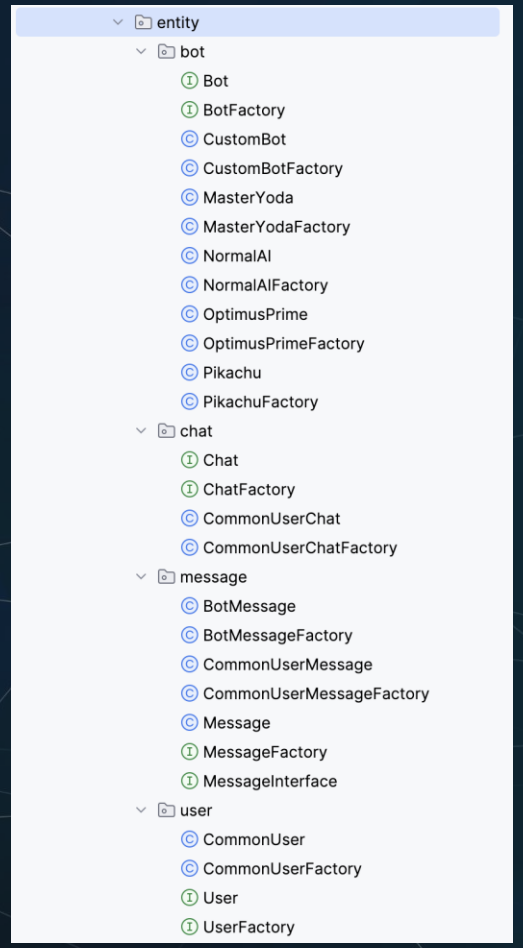
Higher level classes (interactors) do not depend on lower-level classes (controllers or DB access objects/API callers).

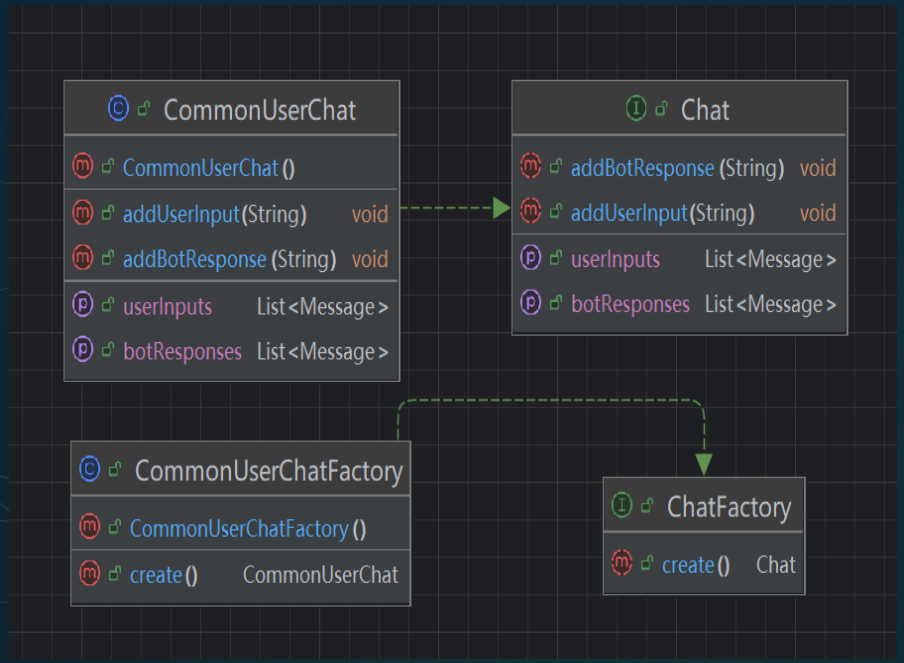
Both depend on abstractions (interfaces).



# Simple Factory Design Pattern

ALL Character Chatbot entities are created through factories and every interactor needing access to an entity takes a factory as an input argument and not the entity class itself.



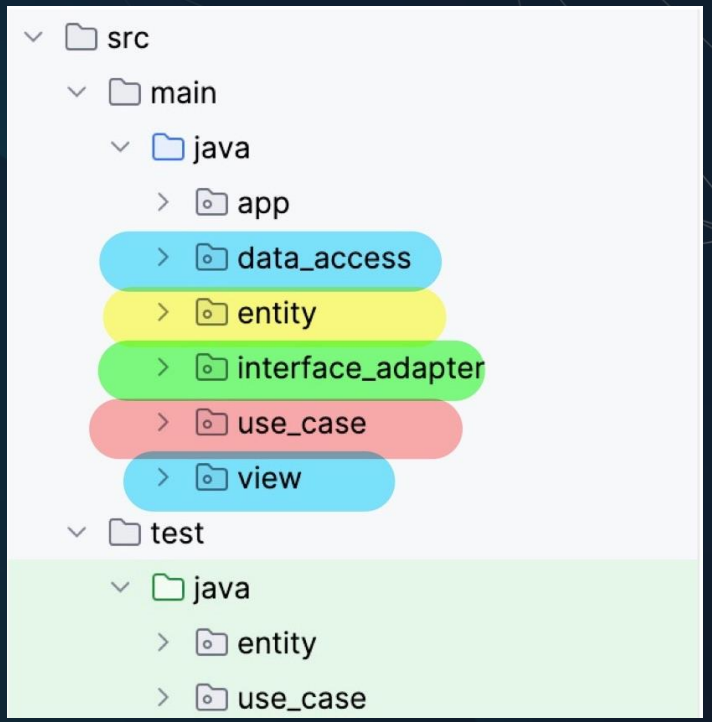



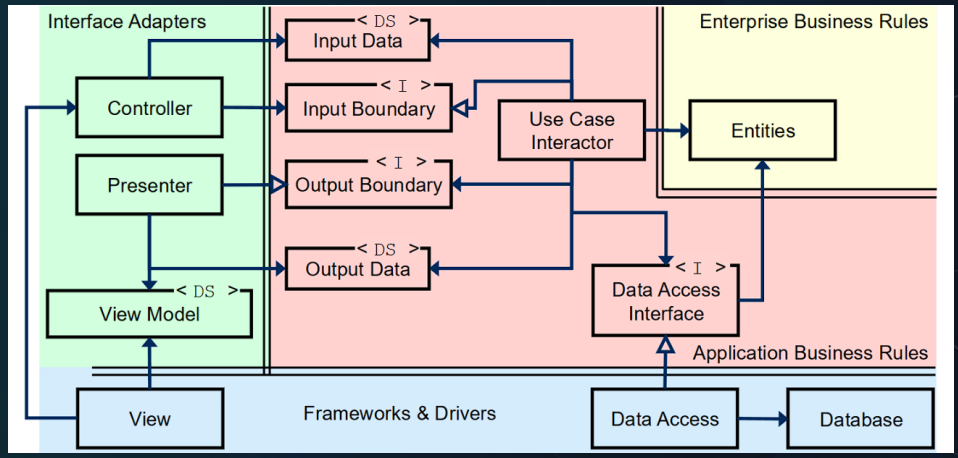

```java
public class ExitChatInteractor implements ExitChatInputBoundary {
    private final ExitChatOutputBoundary exitChatPresenter;
    private final SaveChatHistoryUserDataAccessInterface chatHistoryAccess;
    private final ChatFactory chatFactory;
    private Chat currentChat;

    public ExitChatInteractor(ExitChatOutputBoundary exitChatPresenter,
                              SaveChatHistoryUserDataAccessInterface chatHistoryAccess,
                              ChatFactory chatFactory) {
        this.exitChatPresenter = exitChatPresenter;
        this.chatHistoryAccess = chatHistoryAccess;
        this.chatFactory = chatFactory;
    }

    @Override
    public void execute(ExitChatInputData exitChatInputData) { ...
    }

    @Override
    public void newChat(String username) {
        chatHistoryAccess.setUp(username);
        currentChat = chatFactory.create();
    }
```

# Code Organization

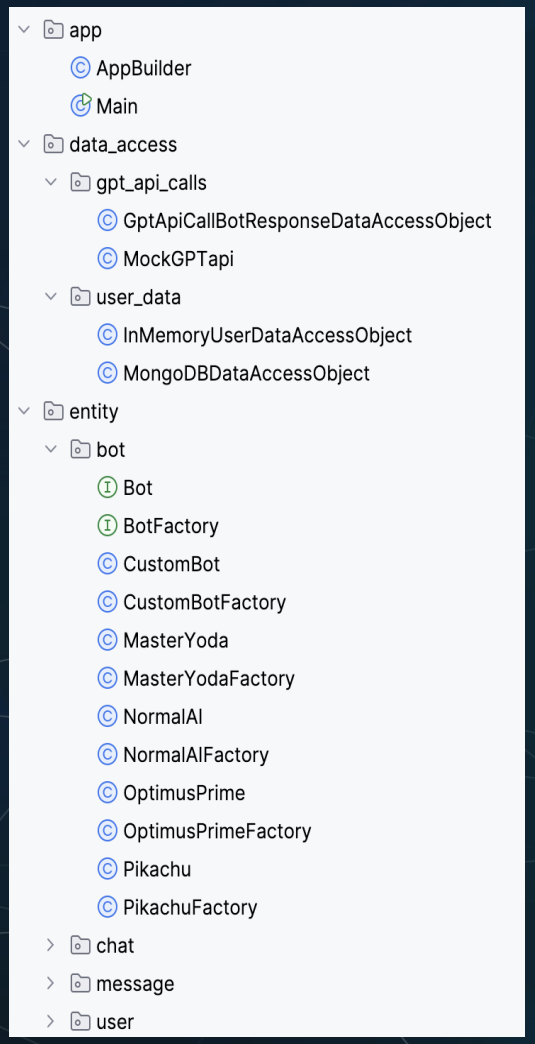
- **Package Structure:**

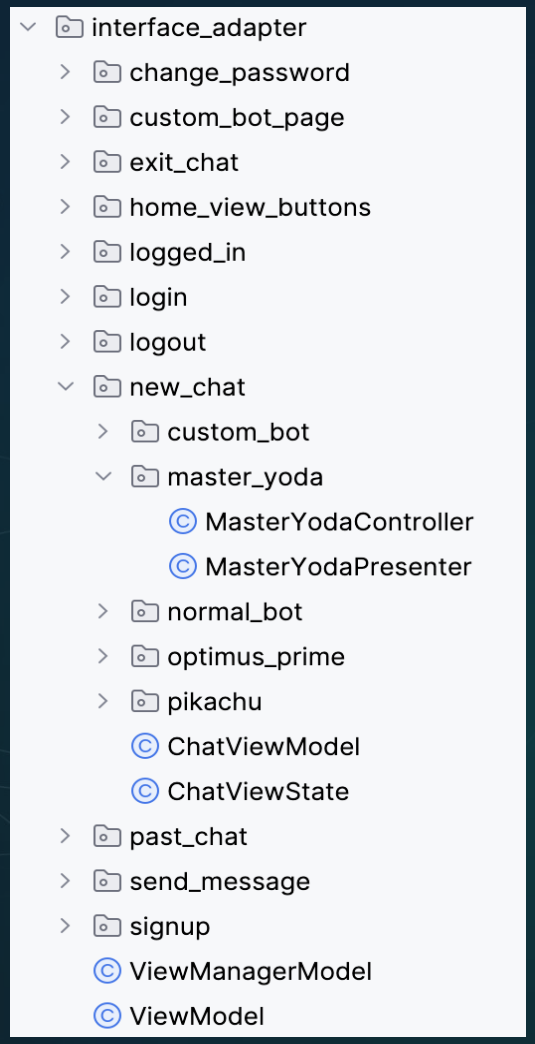
  - **app:** Classes AppBuilder and Main. Setup and entry for the application.
  - **entity:** All classes from the entities layer.
  - **use_case:** All classes from the application business rules layer.
  - **interface_adapters:** All classes from the interface adapters layer.
  - **data_access:** Classes in the Frameworks & Drivers layer related to data storage or API calls.
  - **view:** All views for the program.

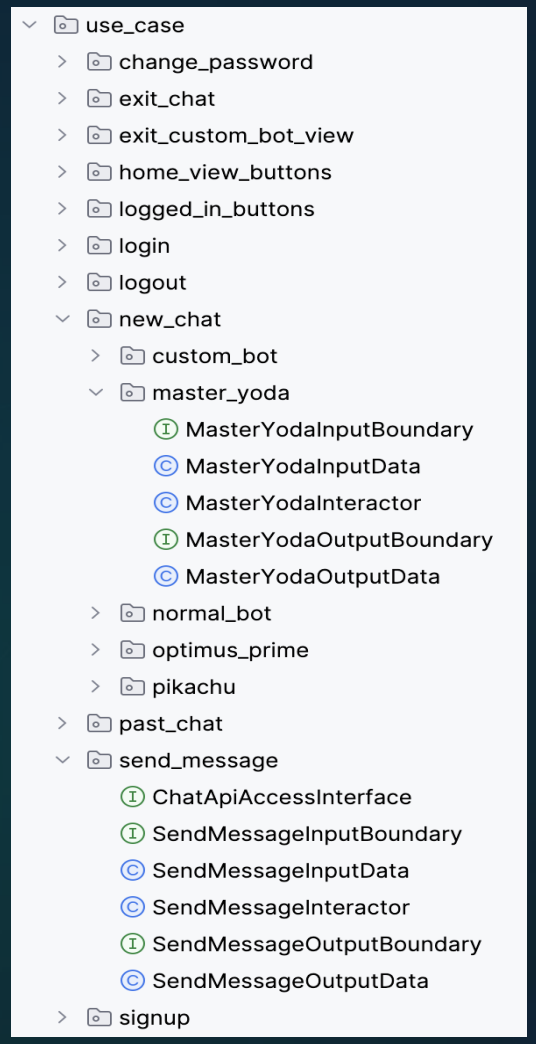


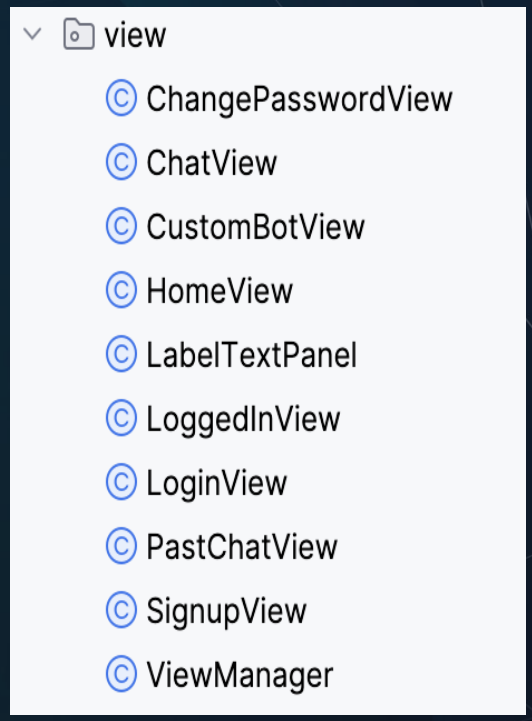



# Code Organization

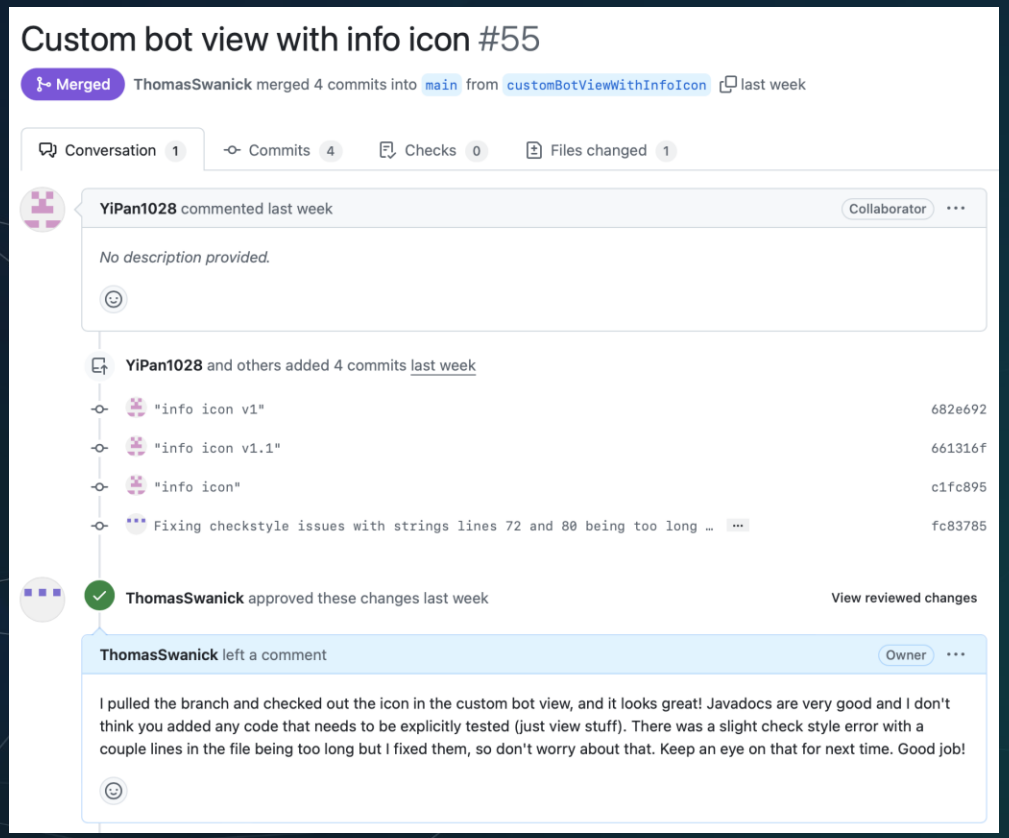


# Code Quality




## Code Review Procedures:

1 ) Pull the branch in question and test that the program itself still works (including any new features).

2 ) Make sure any new classes inside the entities or use cases (application business rules) layers have accompanying tests with 100% code coverage.

3 ) Run our check style file (mystyle.xml in our github repo) and make sure there are no style issues (includes having JavaDocs).

4 ) Comment if any of this is missing, otherwise approve the the request! (we required at least one approval per pull request to merge).

# Code Quality

| | | | | |
|---|---|---|---|---|
| entity | 100% (19/19) | 100% (43/43) | 100% (68/68) | 100% (0/0) |
| user | 100% (2/2) | 100% (5/5) | 100% (8/8) | 100% (0/0) |
| message | 100% (5/5) | 100% (11/11) | 100% (17/17) | 100% (0/0) |
| chat | 100% (2/2) | 100% (6/6) | 100% (12/12) | 100% (0/0) |
| bot | 100% (10/10) | 100% (21/21) | 100% (31/31) | 100% (0/0) |

| | | | | |
|---|---|---|---|---|
| use_case | 100% (41/41) | 100% (108/... | 100% (263/... | 100% (14/14) |
| signup | 100% (3/3) | 100% (10/10) | 100% (26/26) | 100% (4/4) |
| send_message | 100% (3/3) | 100% (8/8) | 100% (27/27) | 100% (0/0) |
| past_chat | 100% (3/3) | 100% (6/6) | 100% (17/17) | 100% (2/2) |
| new_chat | 100% (15/15) | 100% (37/37) | 100% (81/81) | 100% (0/0) |
| logout | 100% (3/3) | 100% (6/6) | 100% (14/14) | 100% (0/0) |
| login | 100% (3/3) | 100% (8/8) | 100% (28/28) | 100% (4/4) |
| logged_in_buttons | 100% (3/3) | 100% (8/8) | 100% (12/12) | 100% (0/0) |
| home_view_buttons | 100% (1/1) | 100% (3/3) | 100% (4/4) | 100% (0/0) |
| exit_custom_bot_view | 100% (1/1) | 100% (2/2) | 100% (3/3) | 100% (0/0) |
| exit_chat | 100% (3/3) | 100% (11/11) | 100% (28/28) | 100% (2/2) |
| change_password | 100% (3/3) | 100% (9/9) | 100% (23/23) | 100% (2/2) |

100%  test coverage for the entity and use case layers

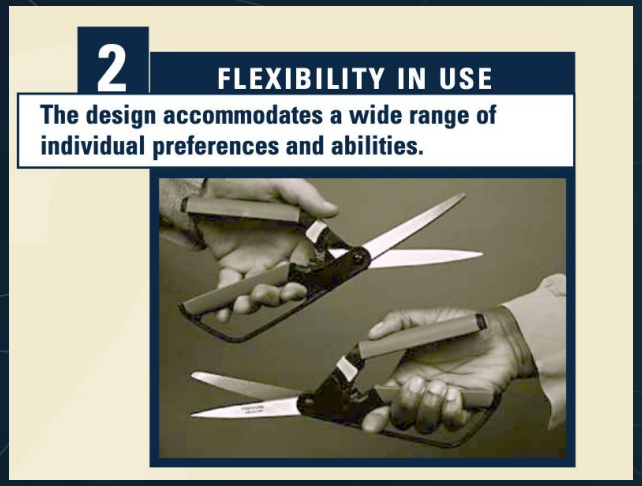
# Accessibility Report

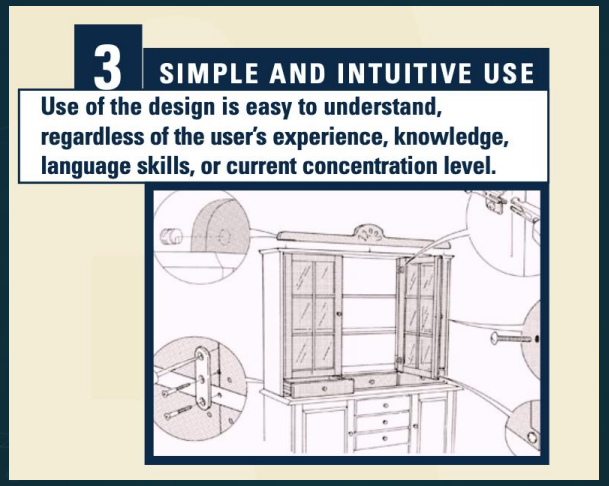
## Principles of Universal Design:

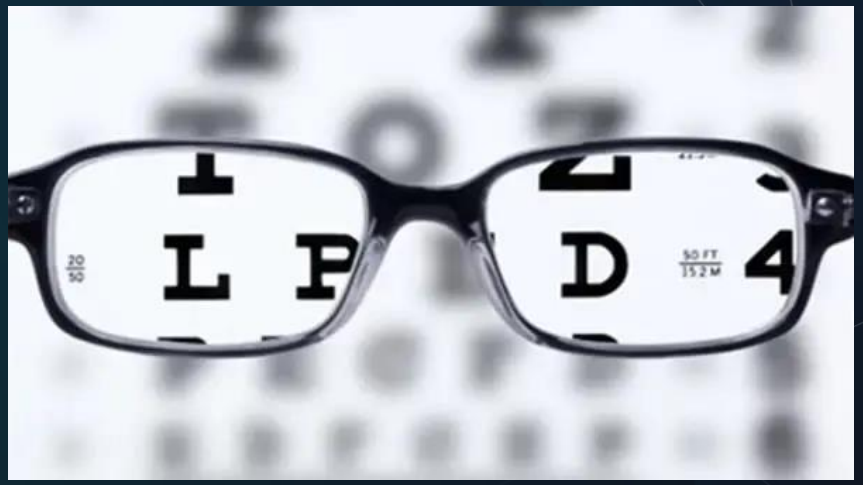
- Flexibility in Use
- Simple and Intuitive Use

## Target Audience and Demographic Considerations:

- Anyone with an interest in some sort of fandom involving characters
- Some limitations for the visually impaired

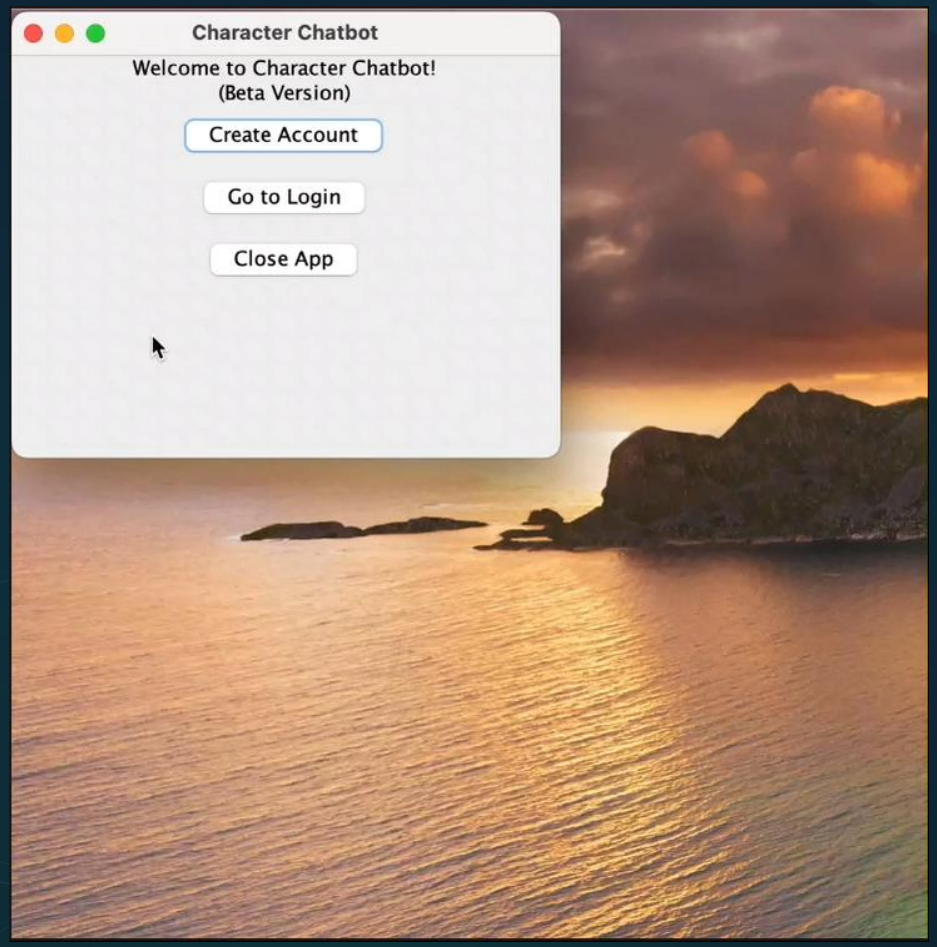


# Character Chatbot Demo

# Application Demo




https://youtu.be/ORRLnTfXpTQ?si=Qv3HVv4k017e8mWP

# Thank You!