

AdvisorBot: A Discord ChatBot for Academic Advising

CSC 2310 – Spring 2023

Abstract

The university has many resources available that can be used to help students make informed decisions regarding their programs of study. This project describes the development of a conversational chatbot for Discord that focuses on providing automated support for academic advising. The system is to be developed using Python and integrated using various web services, Mongo, and Docker.

One of the challenges facing higher education, and specifically Tennessee Technological University, is the increased workload of academic advisors, especially as more universities move towards a centralized advising model. Often, many questions posed by students are available using online resources, with advisors providing personalized information, as necessary. With the continued popularity of GPT-3 from OpenAI.org, the use of chatbots for conversational customer service is becoming more commonplace. Many communication platforms now provide support for creating applications and bots. For instance, Microsoft Teams, Slack, and Discord, all support creation of “bots”. In this project, you will develop a conversational chatbot using Discord. This will include making use of a trained sequential neural net, web services, an unstructured database implemented using Mongo, and Docker.

The primary interface for accessing and interacting with the AdvisorBot chatbot will be Discord, as shown in Figure 1. For the purposes of testing, you will be creating your own Discord server (i.e., guild), enabling services on the Discord developer portal, and deploying your bots as a python application. Your primary focus will be on the creation of intent-driven conversational handlers to address user requests.

As you begin the brainstorming process, recall that as a “developer” you have biases, and that you need to try adopt the mindset of different kinds of users. You may want to ask yourself questions such as:

- Who are the users?
- How will they use this system?
- What are the sources of information that are available?
- How should the system be trained?

In this iteration, you will be doing the following:

- Brainstorm the potential features of the system – for instance, some users might want course descriptions, schedules, or other similar information.
- Specify user stories for each of the user types based on your problem analysis using the user story format of “As a <user>, I want <feature>, so that <reason>”.

As you go through this activity, you should attempt, as much as possible, to avoid thinking about how any of this is going to be implemented. ***The most important aspect of this activity is identifying all of the possible features for this system (noting that identification of all of the possible features does not mean you will have to implement all of the features).***

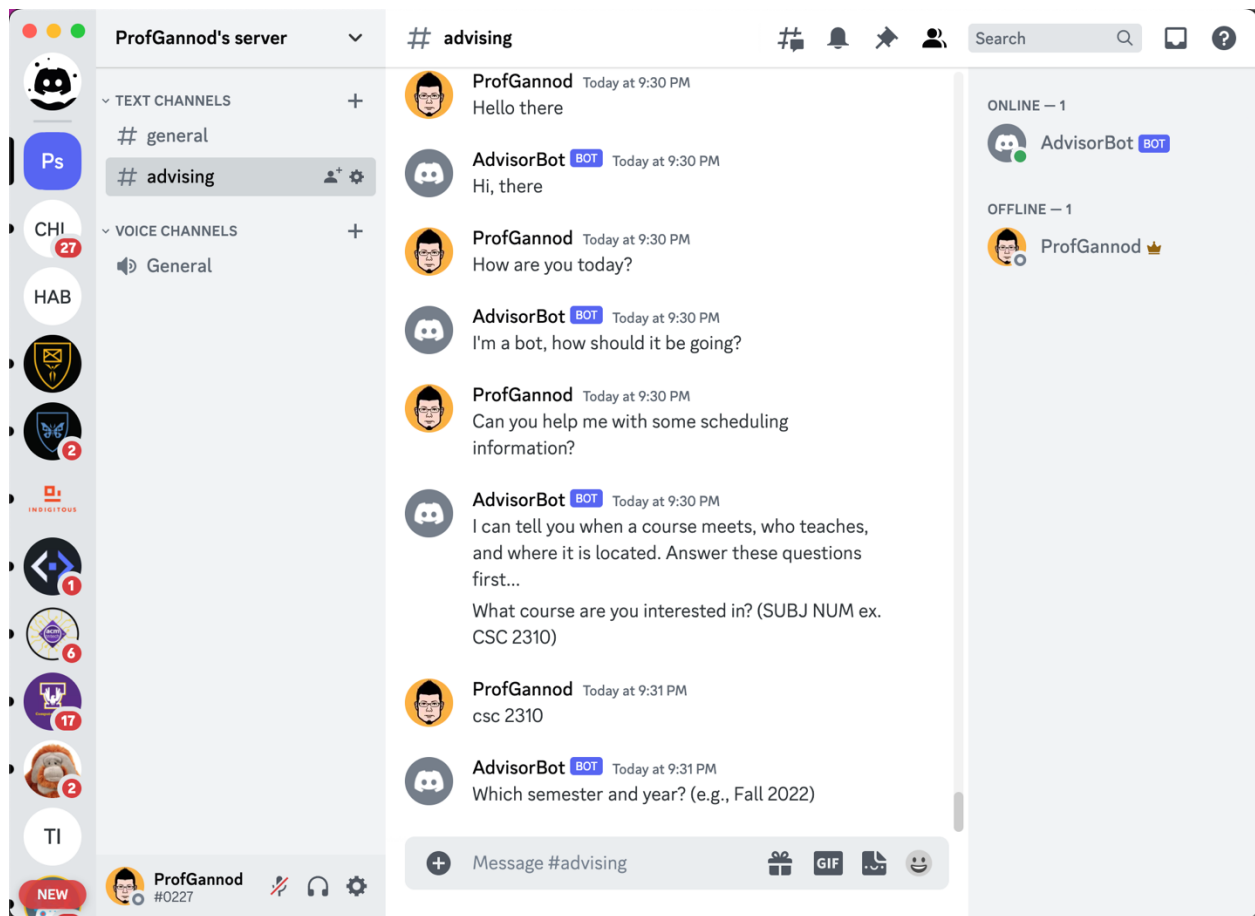


Figure 1 Example of Chatbot Conversation in Discord

Project Phases

The project will be conducted in two major phases: *concept-initiate* and *iteration-construction*. In the *concept-initiate* phase you will identify project requirements, create models, and prep your development environment to support your implementation activities. In the *iteration-construction* phase of the project you will iteratively develop software for the project by adding support according to the requirements developed during the *concept-initiate* phase. The schedule of the phases of the project and the individual iterations contained therein are defined below:

Date (Midnight)	Phase	Description
February 3	Concept Initiate 0	User stories
February 17	Concept Initiate 1	Use case diagrams
March 3	Concept Initiate 2	Initial Design
March 24	Iteration 0	Project environment setup and initial execution demonstration
April 7	Iteration 1	Intents
April 21 26	Iteration 2	Intents with services
April 28	Iteration 2.5	Delivery

Concept Initiate 0

User Stories

In the initial phase of the project, you will create user stories based on the description provided while also generating ideas of other potential features for such a system. The user stories that you create should be specified in the following form:

As a <user type>, I want <desired feature>, so that <outcome>

You must also include, with each user story, a description of how you would expect to be able to observe this behavior in the system. Your source of information for the user stories you will identify are given in the description provided above as well as any discussion sessions we conduct in class. You may also reach out to faculty or other students to generate ideas on potential stories. **However, the work you turn in must be your own. You may not copy other students' user stories, descriptions, or other similar work products.**

A template for your user stories is provided below.

User Story Template

Use the following user story template for *concept-initiate 0*. Replicate this as many times as is necessary.

User Story Name: <Short phrase for the user story>	
As a:	<user type>
I want:	<feature>
So that:	<outcome or reason>
Description	<description of how the user story / feature would be observed and tested in the system>

Submission

Submit your user stories as a PDF document only. *Microsoft Word, LibreOffice, or any other native word processing format will not be accepted.*

Rubric

Completeness: You will be graded on the completeness of your turn-in. In particular, you must analyze the description and attempt to identify as many features as you can that are relevant for the project. Note, there is no upper bound on the number of user stories you can define, but there is a lower bound **that will not be specified**.

Correctness: You will be graded based on your adherence to the format of the user story template and on the accuracy of the user stories with respect to relevance to the described project, including your description of how you might expect to observe the user story / feature in the system.

Points: The assignment is worth 20 project points.

Concept Initiate 1

Use Case Diagrams

In our initial step on this project, we did some work to try to identify user stories for the *AdvisorBot Chatbot* system. A baseline set of 32 user stories is found below and should be used as a starting point for completing the next activity in the project. You must add additional user stories from your own set produced from the set you created for Concept Initiate 0. For Concept Initiate 1, you will be creating use case diagrams for the *AdvisorBot Chatbot* system. The baseline user stories fall into a few categories:

- *Conversation-Type* – Stories that cover specific content that users may want to find
- *Direct-Conversation* – Stories that move a conversation from a bot to a person
- *Meta-Conversation* – Stories regarding the experience of a user
- *Training* – Stories related to training the *AdvisorBot* system

As you did in Laboratory 2, you will use GenMyModel to create your use case diagram(s) for the AdvisorBot system. The user stories that you will use for creating the use cases in the diagram are included as an attachment to the assignment in iLearn. Some things to note about these user stories:

- There are 32 base user stories specified in the reference set across 4 different user types in the baseline set. You must add your own additional user stories (at least 5-10). Set the color of your own use cases to distinguish them from the base set.
- User stories may differ by users with the same feature. These should be represented by different users in the diagram linked to a common use case.
- The use case diagram should be partitioned according to the feature categories specified above. Additional stories may result in additional categories that you should name, if appropriate.
- You may use multiple diagrams to reduce complexity.

Submission

For your turn-in, you should export the model to an image and copy and paste the image to word or some other word processing suite and generate a PDF. The due date is found in the table found in the Project Phases section found earlier in the document.

Rubric

Completeness: You will be graded on the completeness of your turn-in. In particular, your submission must at the very least include the user stories provided in the baseline set. However, it is also expected that you will include other stories in your diagram.

Correctness: You will be graded on the correctness of your use case diagrams. In particular, your submission must correctly represent use cases, actors (both interactive and external actors), and partitioning of the stories into appropriate subsystems by category.

Points: The assignment is worth 25 project points.

No.	1	Category	conversation-type
Title	calendar		
As a(n)	student		
I want to	get calendar information		
So that	I can plan my time and meet deadlines		

No.	2	Category	conversation-type
Title	career		
As a(n)	student		
I want to	get career development information		
So that	I can find part-time and full-time jobs		

No.	3	Category	conversation-type
Title	contact info		
As a(n)	user		
I want to	get contact info		
So that	I can contact people on campus		

No.	4	Category	conversation-type
Title	advisor info		
As a(n)	student		
I want to	find my advisor		
So that	I can schedule advising appointments		

No.	5	Category	conversation-type
Title	course info		
As a(n)	student		
I want to	get course information		
So that	I can find descriptions and pre-requisites		

No.	6	Category	conversation-type
Title	schedule info		
As a(n)	user		
I want to	get schedule information		
So that	I can find information about sections, locations, and professors related to a course		

No.	8	Category	conversation-type
Title	extra-curricular info		
As a(n)	student		
I want to	get extra-curricular information		
So that	I can learn about clubs, greek live, and other activities on and off-campus		

No.	9	Category	conversation-type
Title	building hours		
As a(n)	user		
I want to	get hours of operation		
So that	I know which buildings are open on campus		

No.	10	Category	conversation-type
Title	parking info		
As a(n)	student		
I want to	get parking information		
So that	I know where I can park, pay for tickets, and access other parking services		

No.	11	Category	conversation-type
Title	dining hours		
As a(n)	user		
I want to	get dining hours		
So that	I know when the caf is open and where I can eat		

No.	12	Category	conversation-type
Title	dining info		
As a(n)	user		
I want to	get dining info		
So that	I know how to buy a meal plan		

No.	13	Category	conversation-type
Title	majors		
As a(n)	student		
I want to	get major information		
So that	I can add/change majors or minors		

No.	15	Category	conversation-type
Title	maps		
As a(n)	user		
I want to	get maps		
So that	I know how to navigate around campus		

No.	16	Category	conversation-type
Title	transcripts		
As a(n)	student or alumnus		
I want to	order transcripts		
So that	I can send my transcript to potential employers		

No.	17	Category	conversation-type
Title	scholarships		
As a(n)	student		
I want to	get scholarship information		
So that	I can apply for a scholarship		

No.	18	Category	conversation-type
Title	substitutions		
As a(n)	student		
I want to	get substitution information		
So that	I know whether a course I am taking applies to my POS		

No.	19	Category	conversation-type
Title	transfer		
As a(n)	transfer student		
I want to	get transfer information		
So that	I know whether courses I have taken apply to my POS		

No.	20	Category	conversation-type
Title	tutoring		
As a(n)	student		
I want to	get tutoring information		
So that	I can get help on my course work		

No.	22	Category	direct conversation
Title	notifications		
As a(n)	academic advisor		
I want to	be notified when a student needs to talk to me		
So that	I can make an appointment with the student		

No.	23	Category	direct conversation
Title	transcripts		
As a(n)	academic advisor		
I want to	see a transcript of a bot-student conversation		
So that	I can get context behind a conversation that has been handed off to me		

No.	24	Category	meta-conversation
Title	ask questions		
As a(n)	user		
I want to	ask the bot questions		
So that	I do not have to talk to a person		

No.	25	Category	meta-conversation
Title	talk to bot		
As a(n)	user		
I want to	talk to a bot		
So that	I can get information available in the help system		

No.	26	Category	meta-conversation
Title	save transcripts		
As a(n)	user		
I want to	save a transcript of my chats		
So that	I can refer to them later		

No.	27	Category	meta-conversation
Title	cancel an inquiry		
As a(n)	user		
I want to	cancel an inquiry		
So that	backout when I make a mistake in a conversation		

No.	29	Category	training
Title	train system		
As a(n)	content expert		
I want to	add training information		
So that	the system can learn about potential questions		

No.	30	Category	training
Title	provide feedback		
As a(n)	user		
I want to	provide the system feedback		
So that	the system can better understand conversations		

No.	31	Category	training
Title	upload conversations		
As a(n)	academic advisor		
I want to	upload conversations from Teams		
So that	the system can use my conversations to learn new questions		

No.	32	Category	training
Title	add FAQ		
As a(n)	academic advisor		
I want to	add FAQ information		
So that	I do not have to repeatedly provide information to students		

Concept Initiate 2

Background

In the last iteration, we created use case diagrams for the AdvisorBot Project. A solution to the last iteration is available online on iLearn. Figure 2 below shows the use case diagram that captures the program conversation to be managed by Discord and our bot. In particular, the diagram shows that a user will typically want to talk to a bot to ask questions, or may cancel/reset an inquiry. In lecture, we discussed a model for how the basic execution for the AdvisorBot is structured by using a Sequence Diagram. That model is available in iLearn. The primary goal of the software we are creating is to manage a conversation using these ideas. According to Beebe et al., a conversation involves 5 distinct phases: *initiation*, *preview*, *business*, *feedback*, and *closing* (Beebe, 2002). Specifically, a basic *Conversation* can be modeled as a state model, with transitions to each state determined by recognizing conversational cues.

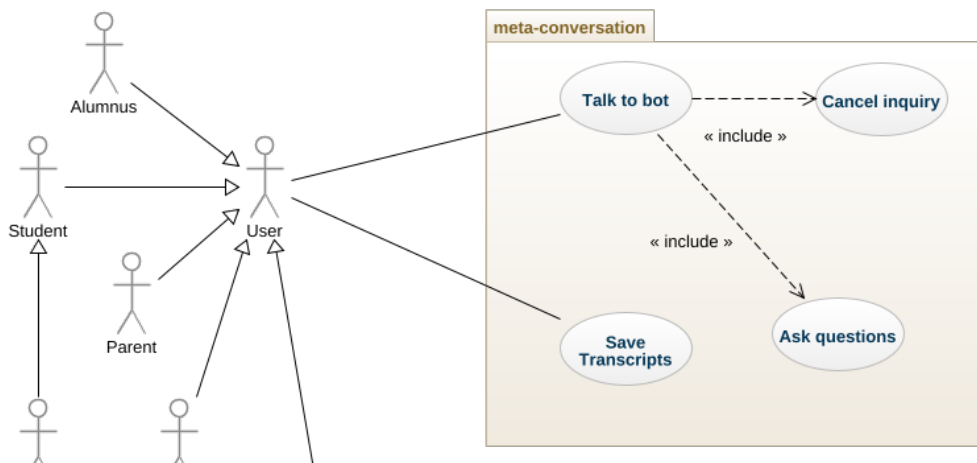


Figure 2 Conversation Stories

A design pattern is a reusable design for providing solutions to recurring problems. The *State Design Pattern* is a pattern that captures how to implement state models in an object-oriented programming environment. Figure 3 shows the basic structure of the pattern, with the *Context* class being the main object, and *ConcreteStates* representing all of the different states that the main object can transition through (Shvets, 2019). The *dashed arrow* between the interface named *State* and *ConcreteStates* indicates that all of the methods defined in *State* must be implemented by *ConcreteStates* classes.

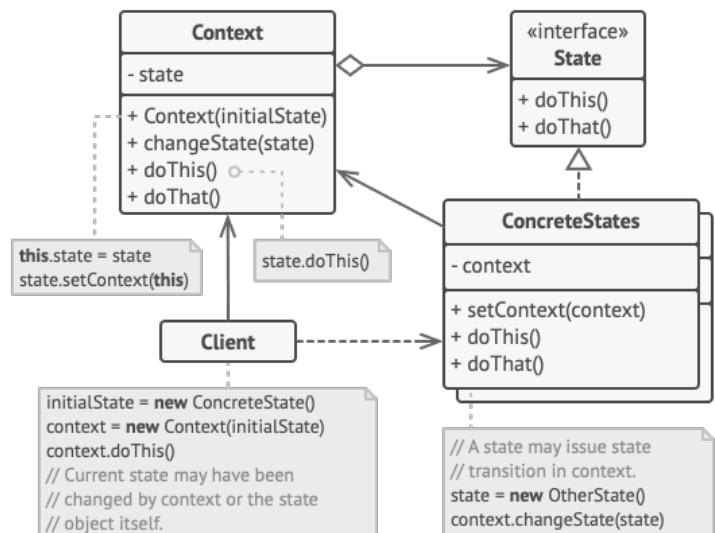


Figure 3 State Design Pattern

Conversation Design

Using the idea of the State design pattern and the phases of a *conversation*, **you are to create a class diagram that models the solution for a Conversation class that involves all the states for a conversation.** Figure 4 shows a state model for conversations, with each state representing the stages. Your task will be to take the information contained in this model, along with the *State* design pattern to create the class diagram for the *Conversation* model.

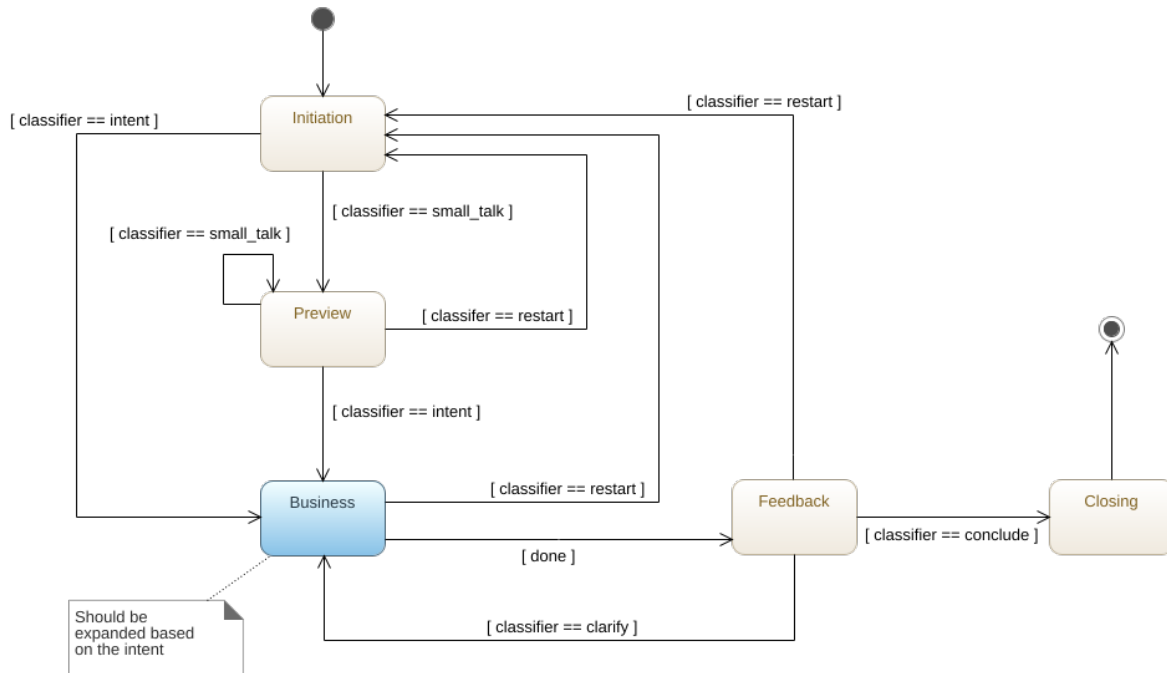


Figure 4 State Diagram of Conversations

Business Phase Design

Figure 5 shows an excerpt of the user stories related to the “Business” phase of a typical conversation. Once a conversation moves into this phase, there are specific questions that must be asked in order to complete the transaction between the user and the bot in order to complete a given task. For instance, take the scenario for retrieving schedule information for a given course. This requires that we query a user for the following:

- Subject (e.g., CSC)
- Course Number (2310)
- Semester (Fall)
- Year (2023)

This could also optionally include asking for other information such as instructor, section, etc. In much the same way that a general conversation can be modeled as a state diagram, the “Business” phase of a conversation can also be modeled as a state diagram, with different related questions being grouped together. For instance, it is natural to ask Subject and Number together, and Semester and Year together.

When engaging in such conversations, in addition to having a transaction of information, it is possible that users will do the following:

- Make mistakes, requiring repeating of a given query for information.
- Decide against proceeding with the line of inquiry, requiring canceling of the transaction and restarting the conversation to the initiation state.
- Leave the conversation, requiring saving or canceling the conversation.

To gain a better understanding of the “Business” phase of a conversation, *you will create a class diagram model based* on using the State Design Pattern of the following use cases:

- Contact Information
- Course Information
- Schedule Information

Each scenario should be modeled in its own diagram, with the “Context” class including an attribute that identifies all of the parameters that need to be gathered from a user in order to retrieve the desired information. In addition, your model should capture the potential special cases discussed above as possible states within the model.

Finally, for these three scenarios, you must create a *state diagram* similar to the one shown in Figure 4 to depict the possible transitions between states.

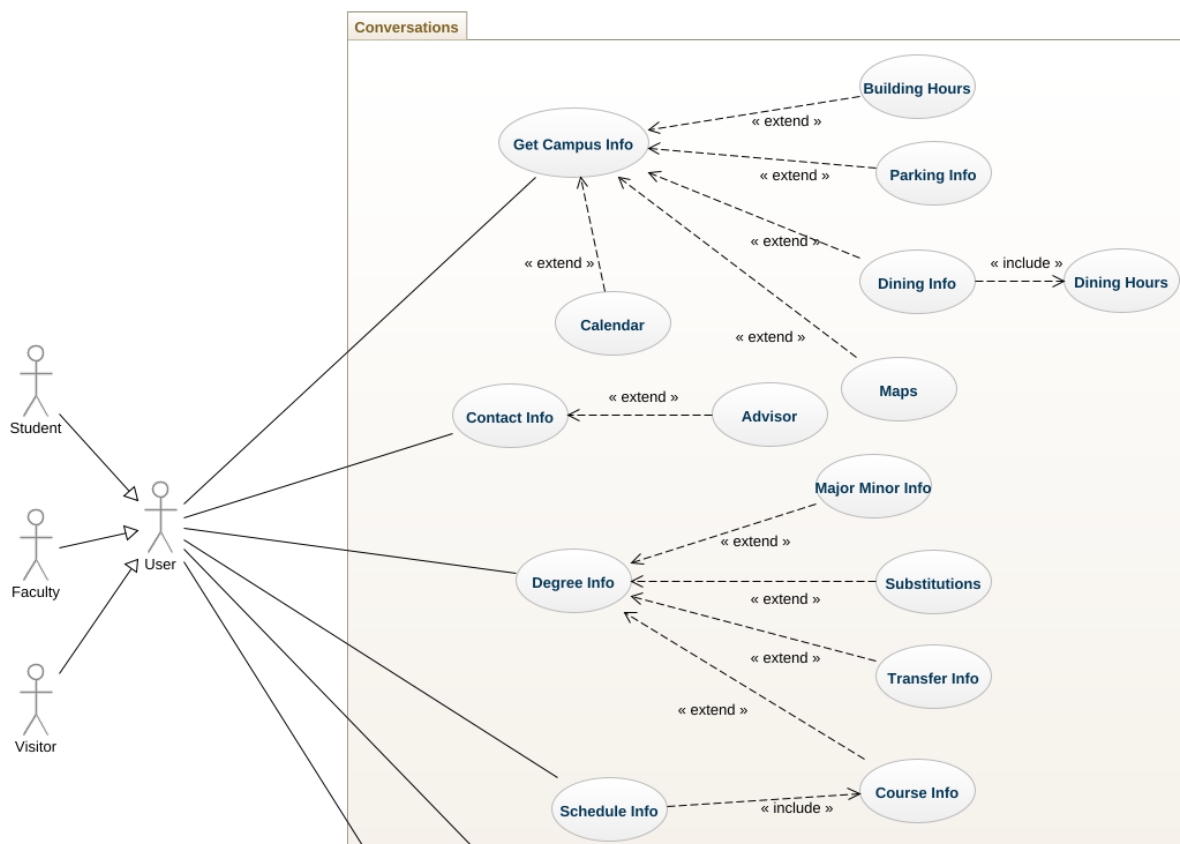


Figure 5 Excerpt of "Business" Conversations

Submission

Use the <https://app.genmymodel.com> system to create your models. Export your model to a PNG and submit as a PDF to iLearn. Videos are available online on the different modeling notations for UML.

Rubric

Completeness: You will be graded on the completeness of your submission, most notably the inclusion of classes and states described above as well as on the identification of implicit and derived classes that arise from a thorough analysis of the problem.

Correctness: You will be graded on the correctness of the class specifications you have created including correct capture of the relationships between the classes and the specification of attributes and methods that arise from a thorough analysis of the problem. State models must follow noted conventions for state diagrams.

Points: The assignment is worth 30 project points.

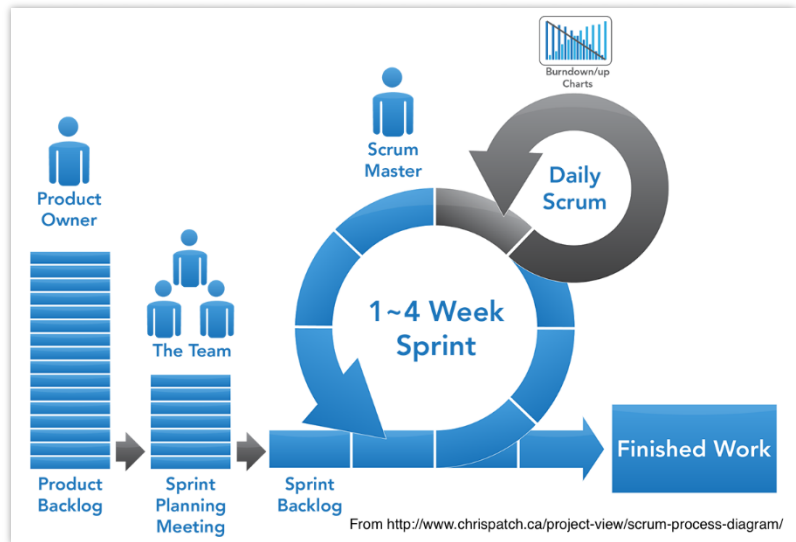
References

Beebe, S. A., Beebe, S. J., Redmond, M. V., Geerinck, T., & Salem-Wiseman, L. (2002). *Interpersonal communication: Relating to others* (p. 432). Boston: Allyn and Bacon.

Shvets, A. (2019). *Dive Into Design Patterns*. Refactoring.Guru

Iteration 0

Thus far, we have been following the Scrum process for software development. After having completed the creation of a backlog (i.e., a collection of user stories) and design, we are ready to begin development. For a majority of projects, Iteration 0 is the first step towards commencing software production. In this phase, development teams setup their development environments, initialize their git repositories, and configure the implementation platforms being used to create working software. For Iteration 0 of our project, you will:



- Create a Discord account (if you do not already have one or prefer to have a different development account for the project)
- Create a new discord server
- Provision a discord bot in the discord server
- Verify the setup

This iteration of the project is extremely critical for ensuring that you will be able to develop and test your code for the rest of the project. The activities that you will perform are relatively straightforward, and the requirements for what you must turn in are relatively trivial. Regardless, it is important that this step be completed and done so in a timely fashion.

Create Discord Account and Server

As communicated previously, the primary platform that we will be using for our application is *Discord*. You will need a *Discord* account and server, and you will need to provision a “bot” application on the *Discord* developer site. If you already have an account, you may use the credentials for that account for setting up your server and bot application. If not, create an account and login to Discord at <https://discord.com>. Once you have created your account, look for the “+” button in the list of servers to create a new Discord server, as shown in Figure 6. It is suggested that you do not use existing servers as the bot you will create will have access to all conversational traffic on the server.

Discord Application

After creating your server, you will provision a new application on the Discord Developer portal. To begin, go to the following url and login:

<http://discordapp.com/developers/applications>

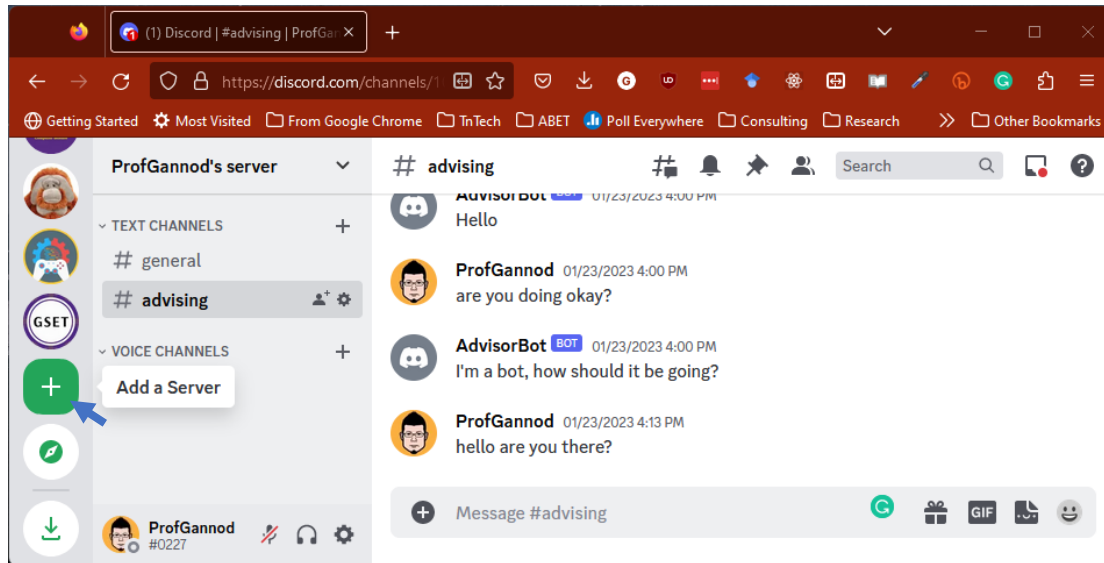
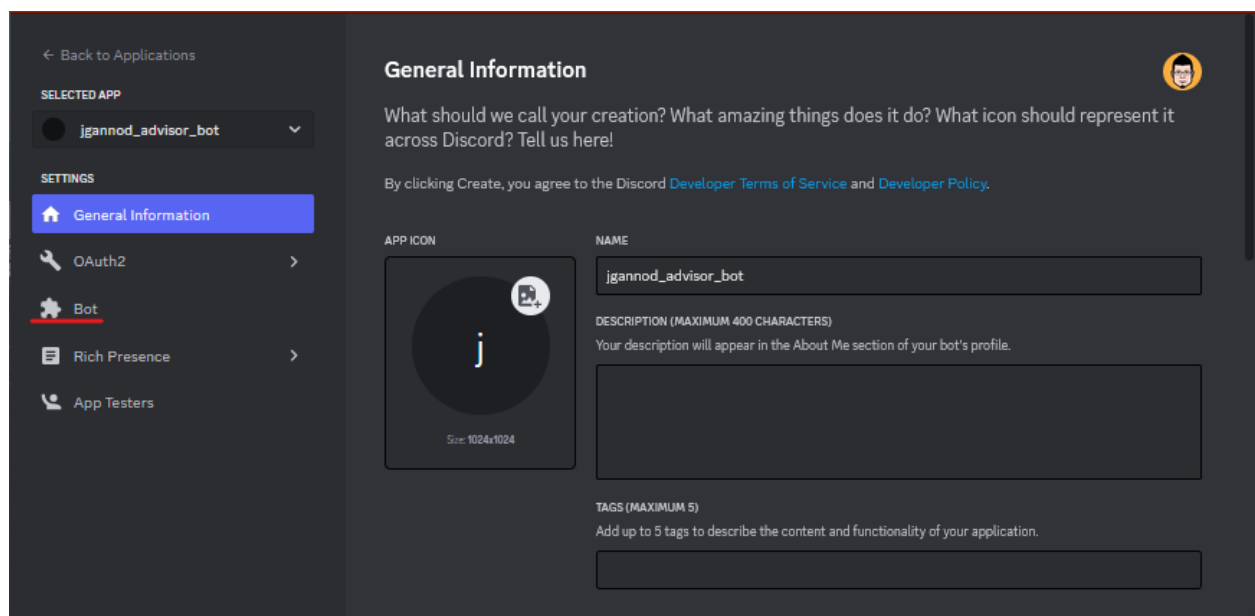
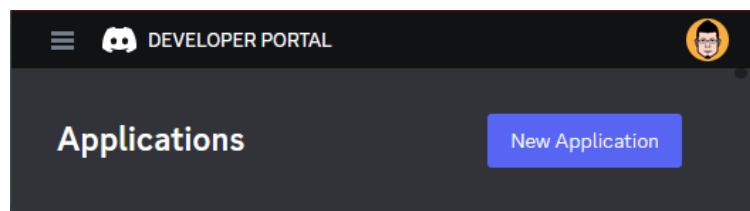


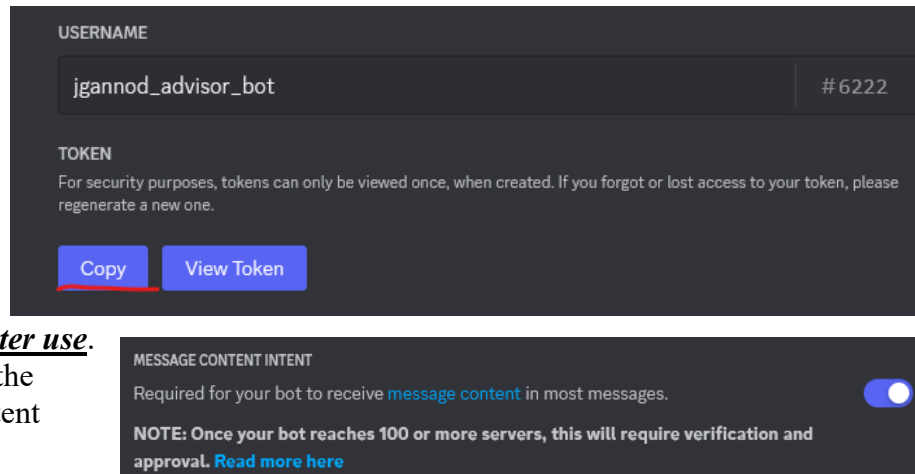
Figure 6 Discord

On the portal page, select “New Application” in the upper right hand corner.

When naming your application, use the following pattern: `userid_advisor_bot`. For example, `jgannod_advisor_bot`. Once you have created the application, you will be presented with a dashboard similar to the one shown below.



From this screen, select the “Bot” channel in the menu on the left-hand menu of the portal to create a new bot. Do so by selecting the “Add Bot” button to generate your bot. You will need to **copy the bot token and save it in a file for later use.** You will also need to set the option for “Message Content Intent” to “on”.



USERNAME

jgannod_advisor_bot # 6222

TOKEN

For security purposes, tokens can only be viewed once, when created. If you forgot or lost access to your token, please regenerate a new one.

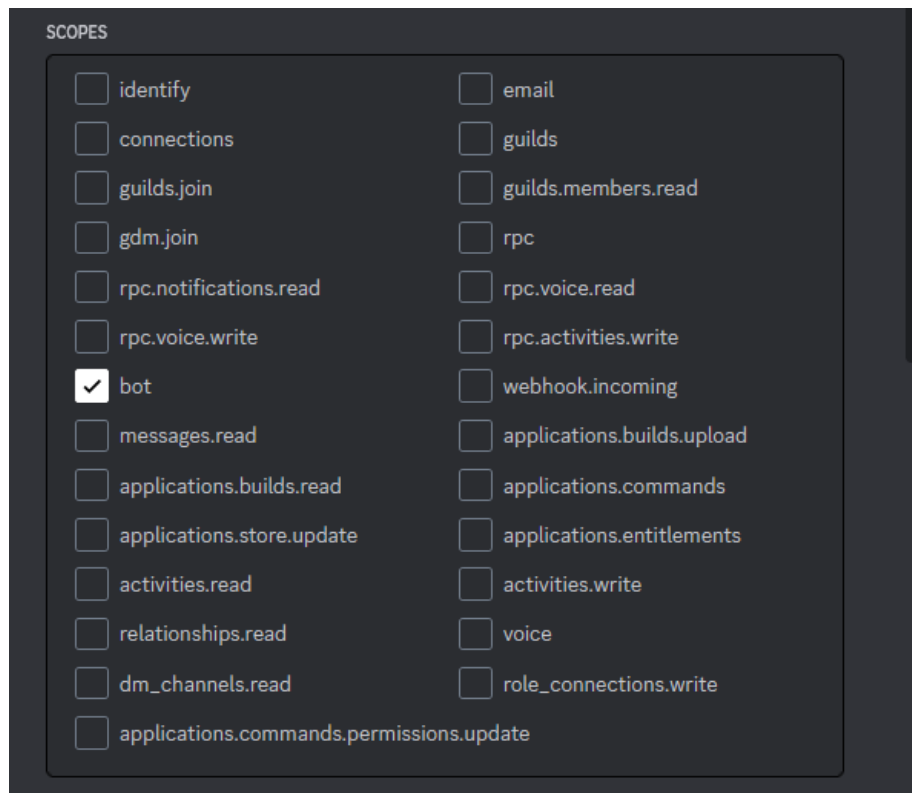
Copy View Token

MESSAGE CONTENT INTENT

Required for your bot to receive [message content](#) in most messages.

NOTE: Once your bot reaches 100 or more servers, this will require verification and approval. [Read more here](#)

Finally, select the OAuth2 tab and click “URL Generator”. In the “Scopes” section, you will see a number of click boxes. Select “bot”:



SCOPES

<input type="checkbox"/> identify	<input type="checkbox"/> email
<input type="checkbox"/> connections	<input type="checkbox"/> guilds
<input type="checkbox"/> guilds.join	<input type="checkbox"/> guilds.members.read
<input type="checkbox"/> gdm.join	<input type="checkbox"/> rpc
<input type="checkbox"/> rpc.notifications.read	<input type="checkbox"/> rpc.voice.read
<input type="checkbox"/> rpc.voice.write	<input type="checkbox"/> rpc.activities.write
<input checked="" type="checkbox"/> bot	<input type="checkbox"/> webhook.incoming
<input type="checkbox"/> messages.read	<input type="checkbox"/> applications.builds.upload
<input type="checkbox"/> applications.builds.read	<input type="checkbox"/> applications.commands
<input type="checkbox"/> applications.store.update	<input type="checkbox"/> applications.entitlements
<input type="checkbox"/> activities.read	<input type="checkbox"/> activities.write
<input type="checkbox"/> relationships.read	<input type="checkbox"/> voice
<input type="checkbox"/> dm_channels.read	<input type="checkbox"/> role_connections.write
<input type="checkbox"/> applications.commands.permissions.update	

After selection, another set of options will appear entitled “Bot Permissions”. As you are creating a development server and application, it is fine to choose “Administrator” level privileges. If you later decide to continue to use this bot, you may want to select a more restrictive set of permissions.

BOT PERMISSIONS

GENERAL PERMISSIONS	TEXT PERMISSIONS	VOICE PERMISSIONS
<input checked="" type="checkbox"/> Administrator	<input type="checkbox"/> Send Messages	<input type="checkbox"/> Connect
<input type="checkbox"/> View Audit Log	<input type="checkbox"/> Create Public Threads	<input type="checkbox"/> Speak
<input type="checkbox"/> Manage Server	<input type="checkbox"/> Create Private Threads	<input type="checkbox"/> Video
<input type="checkbox"/> Manage Roles	<input type="checkbox"/> Send Messages in Threads	<input type="checkbox"/> Mute Members
<input type="checkbox"/> Manage Channels	<input type="checkbox"/> Send TTS Messages	<input type="checkbox"/> Deafen Members
<input type="checkbox"/> Kick Members	<input type="checkbox"/> Manage Messages	<input type="checkbox"/> Move Members
<input type="checkbox"/> Ban Members	<input type="checkbox"/> Manage Threads	<input type="checkbox"/> Use Voice Activity
<input type="checkbox"/> Create Instant Invite	<input type="checkbox"/> Embed Links	<input type="checkbox"/> Priority Speaker
<input type="checkbox"/> Change Nickname	<input type="checkbox"/> Attach Files	<input type="checkbox"/> Request To Speak
<input type="checkbox"/> Manage Nicknames	<input type="checkbox"/> Read Message History	<input type="checkbox"/> Use Embedded Activities
<input type="checkbox"/> Manage Emojis and Stickers	<input type="checkbox"/> Mention Everyone	
<input type="checkbox"/> Manage Webhooks	<input type="checkbox"/> Use External Emojis	
<input type="checkbox"/> Read Messages/View Channels	<input type="checkbox"/> Use External Stickers	
<input type="checkbox"/> Manage Events	<input type="checkbox"/> Add Reactions	
<input type="checkbox"/> Moderate Members	<input type="checkbox"/> Use Slash Commands	
<input type="checkbox"/> View Server Insights		
<input type="checkbox"/> View Creator Monetization Insights		

GENERATED URL

https://discord.com/api/oauth2/authorize?client_id=1081788761022672977&permis... [Copy](#)

An external application **jgannod_advisor_bot** BOT wants to access your Discord account
Signed in as ProfGannod#0227 [Not you?](#)

THIS WILL ALLOW THE DEVELOPER OF JGANNOD_ADVISOR_BOT TO:

- ☒ Create commands in a server
- ☐ Record a new mixtape

ADD TO SERVER:

[ProfGannod's server](#) ▼

This requires you to have **Manage Server** permission in the server.

[Cancel](#) [Continue](#)

At the bottom of the page, you will find a URL that you will need to use to map your bot to your server. Copy and paste that URL to your web browser. You will be presented with a page similar to the following. Select your development Discord server and click “Continue”. At this point, you should receive a notification that a new user named “userid_advisor_bot” has joined your server.

Python Application Setup

The final step for setting up your application is to clone the repo that has been created for you using:

```
git clone url
```

where *url* is

```
https://gitlab.csc.tnitech.edu/csc2310-sp23-students/youruserid/youruserid-advisor_bot.git
```

Once you have cloned the repo, replace the “ENTER YOUR BOT TOKEN HERE” in “.env” with the token you saved earlier.

```
DISCORD_BOT_TOKEN="ENTER YOUR BOT TOKEN HERE"
```

To test your setup, you can run the program as follows:

```
python bot.py
```

In the terminal, you will see an output of the users:

Users:

```
jgannod_advisor_bot
```

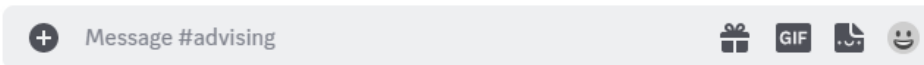
In the Discord app, if you send the message “Hello”, you will see the response “World!”.



ProfGannod Today at 11:35 PM
Hello



jgannod_advisor_bot BOT Today at 11:35 PM
World!



Submission

Your submission for this iteration will be to simply create a screen capture showing your bot responding to your “Hello” message, as shown above. The due date is March 24th at 11:59pm.

Rubric

The only validation for this lab is that you’ve completed creation of the app and that it is connected to your development server.

Points

The assignment is worth 20 project points.

Iteration 1

Architecture

Figure 7 AdvisorBot Design shows the design of the main conversational engine for the AdvisorBot Discord bot using the State Design Pattern as the primary architecture for the software. In this iteration you will be implementing the classes located in the `src.states.conversation` package, all of which represent the typical states of human conversation. In particular, you will be implementing the subclasses of the `State` class, focusing on the `parse_response` method in each class. A state diagram depicting the logic for how the software manages transitions between states is shown in Figure 8. Take special note that each state shows a transition to other states based on the value of `intent_class`.

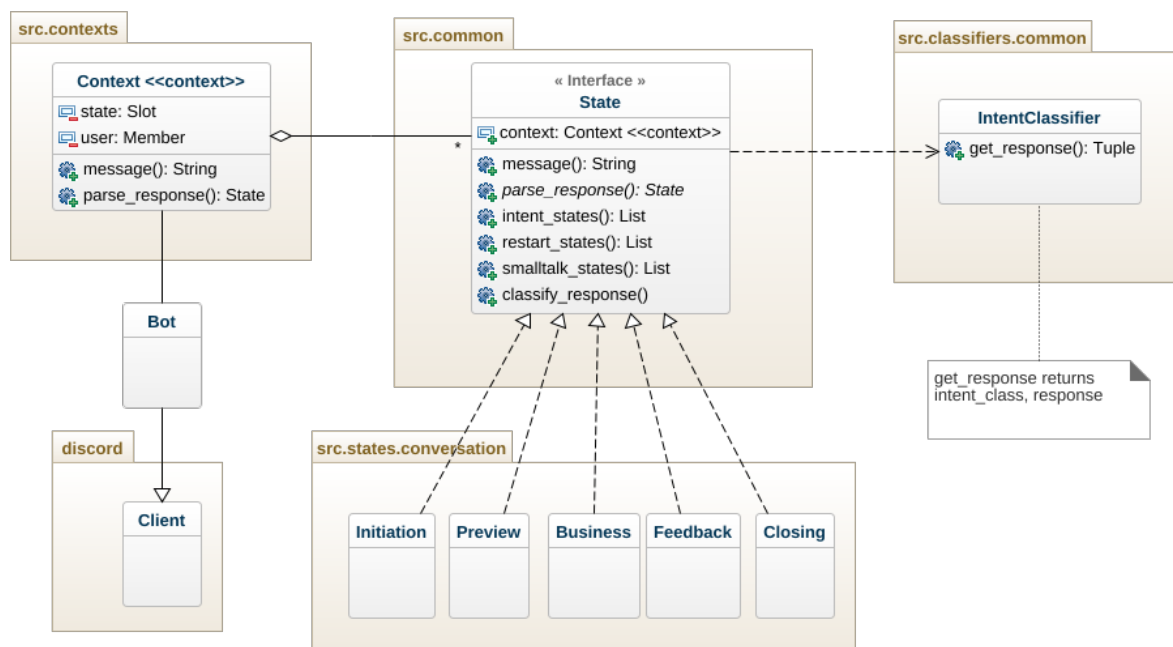


Figure 7 AdvisorBot Design

Repository

A repository with the starting point for the project has been pushed to your account in gitlab and can be retrieved using git as follows:

```
git clone url
```

where `url` is

https://gitlab.csc.tntech.edu/csc2310-sp23-students/youruserid/youruserid-advisor_bot_1.git

After you have cloned the files, edit the source files directly. DO NOT COPY THEM TO A NEW DIRECTORY for editing. This defeats the purpose of using git.

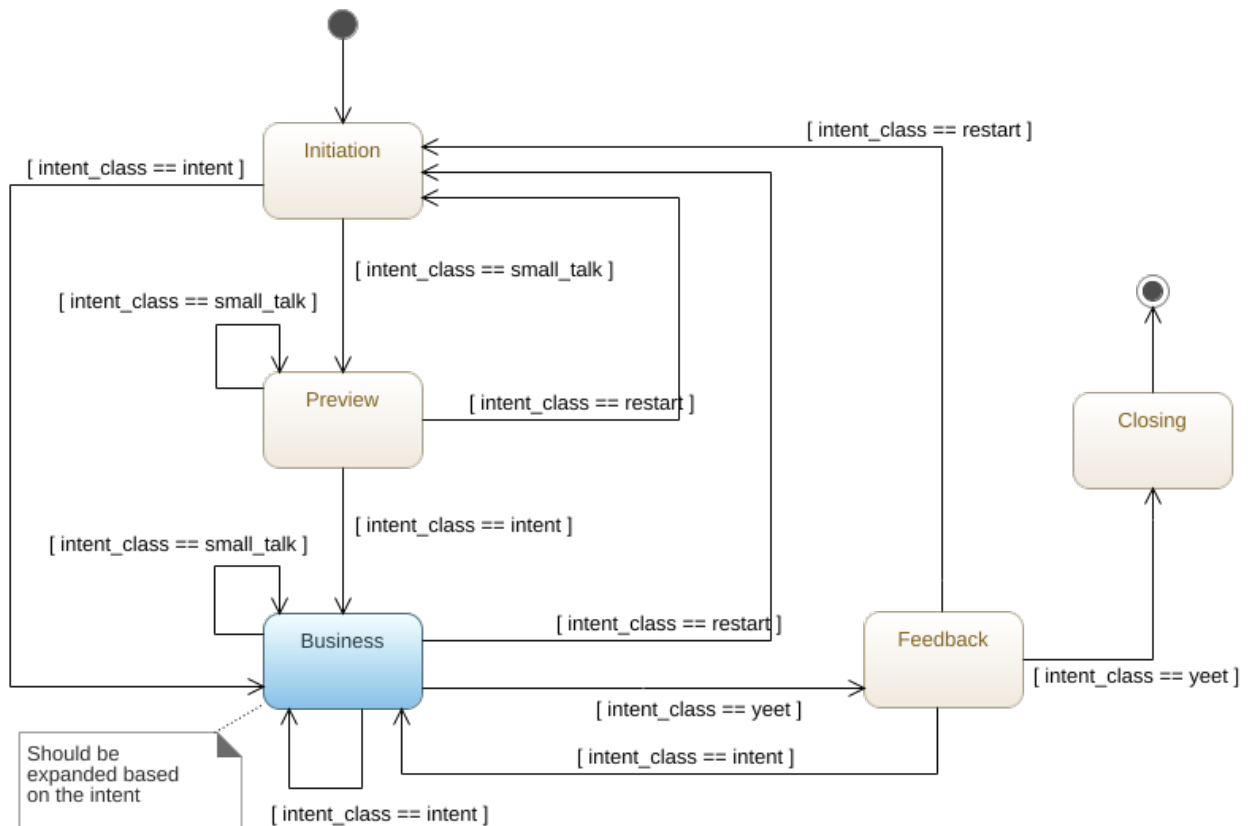


Figure 8 State Model for Conversations

Project Setup

The README.md file (also provided as README.pdf in iLearn) contains all the details needed to finish the setup of your development environment. READ THE DOCUMENT CAREFULLY. This includes making sure that docker is running, mongodb is installed, the training data is imported into the database, and the learner is configured to do classification of responses.

Implementation

You will be implementing each of the following classes in the corresponding files:

- Initiation (initiation.py)
- Preview (preview.py)
- Business (business.py)
- Feedback (feedback.py)

The Closing (closing.py) class should remain as is. Each file has a stub for each class with the appropriate signature for the *parse_response* method as follows:

```
def parse_response(self, resp: str) -> "State"
```

The input to the method is the text input provided by a user to the chatbot. The method is responsible for doing the following based on that input:

- Calling `self.classify_response` to identify the `intent_class` along with a text response to the user input
- Setting `self.message` to the value of the text response
- Instantiating a new State subclass (i.e., one of Initiation, Preview, Business, Feedback, or Closing) based on the value of the `intent_class`. This state should be returned by the method. In the case that the transition is back to the current state, the method should return `self`.

The State superclass has a number of attributes and methods that are necessary to support the creation of your solution. You may not make modifications to the `state.py` file. The details on this class are below:

`@property`

`def intent_states(self) -> List[str]:`

This read-only method returns a list of strings that make up the classifications related to different intents. The list can be found in the constructor for State.

`@property`

`def restart_states(self) -> List[str]:`

This read-only method returns a list of strings that make up the classifications related to transitioning back to the Initiation state.

`@property`

`def smalltalk_states(self) -> List[str]:`

This read-only method returns a list of strings that make up the classifications related to when a user engages in small talk.

`@property`

`def message(self) -> str:`

`@message.setter`

`def message(self, msg: str) -> None:`

The read/write message property allows you to set the message to be displayed to a user in the chatbot channel on discord.

`def classify_response(self, user_msg: str) -> Tuple[str, str]:`

The `classify_response` method is used to access the trained classifier. It takes a string as input and returns a pair of strings:

- `intent_class`: A string that identifies the type of message that was entered by the user. The classes include greeting, well-being-inquiry, well-being-response, inquiry-response, goodbye, thanks, help, schedule_info, switch, course_info, enrollment, and yeet.
- `response`: A string containing an appropriate response to be returned to the user in discord.

Testing

Unit tests have been provided for you in the tests directory. These tests represent a sampling of what will be used to grade your submission. You are required to create your own tests. Guidance on how to do this will be discussed in class. Specifically, for each of the test_X methods provided to you, you must create a new similar test method that tests the same transition condition but uses a different input string.

Submission

You must submit your solution via git. No submissions will be accepted in iLearn. To turn in your project, you should stage your changes, and commit as follows:

```
git commit -m "Completed solution"  
git push -u origin master
```

Rubric

Completeness: You will be evaluated on your completion of each of the four classes (Initiation, Preview, Business, and Feedback).

Correctness: You will be evaluated based on the correct transition between states in the given classes as determined by the parse_response method.

Testing: You will be evaluated based on the creation of your own tests for the project. For each of the states, you should provide alternative test methods for each of the tests provided in the unit test files.

Submission: You will be evaluated based on the correct submission of your solution back to the repository.

Points: The iteration is worth 50 project points.

Iteration 2

In the final iteration of the project, you will be implementing two Context classes that are responsible for retrieving data from web services. These classes are used to implement the following user stories:

- Course Information
- Contact Information

These classes are used by the State classes to pull data from services based on appropriate queries from a user.

Your solution for each class must handle two different categories of cases:

- “Happy Path” cases: these occur when a legitimate input is used and the system behaves as expected.
- Error cases: there are several scenarios to consider when accessing the given web services. These include the following:
 - Invalid or missing input is provided.
 - Network errors occur (such as the network is disabled or an incorrect url is used)

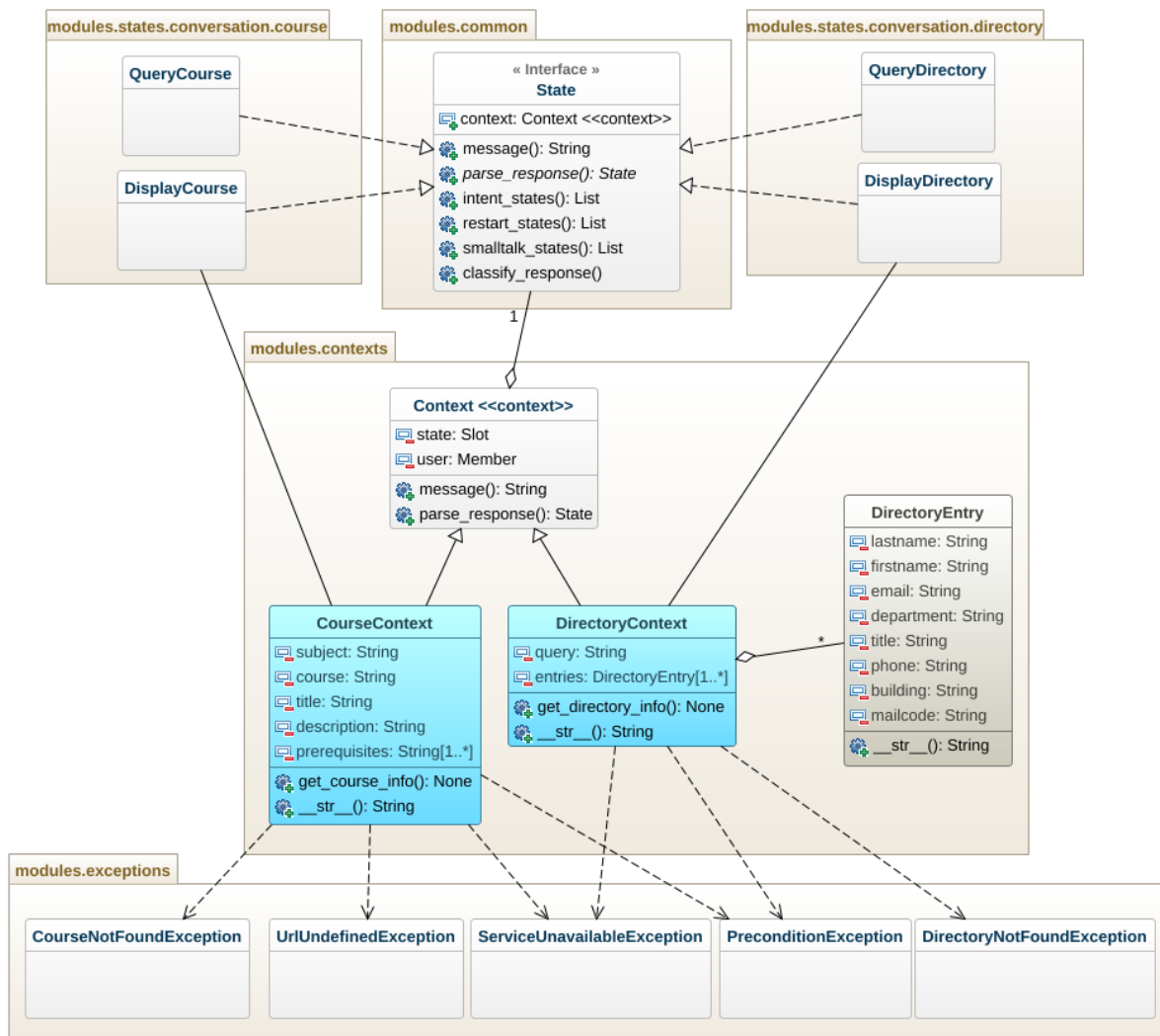


Figure 9 Context Design

Architecture

The modified design for the system is shown in Figure 9. The two classes that you should be most interested in are `CourseContext` and `DirectoryContext`. Each has attributes that are either used as input for the queries to the corresponding web services or are set by the class when the data is retrieved from the web service.

Repository

A repository with the starting point for the project has been pushed to your account in gitlab and can be retrieved using git as follows:

```
git clone url
```

where *url* is

```
https://gitlab.csc.tntech.edu/csc2310-sp23-students/youruserid/youruserid-advisor_bot_2.git
```

After you have cloned the files, edit the source files directly. DO NOT COPY THEM TO A NEW DIRECTORY for editing. This defeats the purpose of using git.

Project Setup

Project setup is identical to the last iteration with respect to docker and mongo. The `.env` file will need to be updated to include your `DISCORD_BOT_TOKEN` and `DIRECTORY_API_KEY` values either retrieved from the discord developers portal or via e-mail.

Implementation

You are responsible for implementing the following classes, both of which are subclasses of `Context`:

- `CourseContext`
- `DirectoryContext`

You are provided with the following exception classes that you will use in the implementation of the `Context` subclasses:

- `CourseNotFoundException`
- `URLUndefinedException`
- `PreconditionException`
- `ServiceUnavailableException`
- `DirectoryNotFoundException`

You will also be provided with a class called `DirectoryEntry`, which is used as a simple class for organizing contact information for a faculty or staff person. Figure 9 provides the details on which modules each of the above is contained in.

CourseContext

The `CourseContext` class is used to retrieve specific information about courses based on the *subject* and *course number*. The url to the service is the following:

`http://csclnx01.tntech.edu:8080/courses`

and it uses the following parameters:

- subject – a 3- or 4-character subject code (e.g., CSC)
- number – a 4-digit course code (e.g., 2310)
- term – a 6-digit year and semester code (e.g., 202280). For our purposes, you may assume the use of 202280

A well-formed url used to make a call to this service looks like the following:

`http://csclnx01.tntech.edu:8080/courses?subject=CSC&number=2310&term=202280`

Using the requests library, a call to the method is made by passing a well-formed url as a string to the get method as follows within a try-except block:

```
r = requests.get(url)
r.raise_for_status()
data = r.json()
```

Upon return the service will return a dictionary formatted like the following example:

```
{
  "id": 76,
  "attribute": {
    "subject": "CSC",
    "number": "2310",
    "title": "Object-Oriented Prgrming/Dsgn",
    "description": "Prerequisites: C or better in CSC 1310. Theory and practice ...",
    "credits": 4,
    "prerequisites": ["CSC 1310"]
  }
}
```

For a full example on how to make a call to this service, please refer to the code found at the following url:

`https://gitlab.csc.tntech.edu/jgannod/bot_webservices.git`

Your implementation for the CourseContext class must do the following:

- Implement getters and setters for each of the attributes shown in the model.
- Implement the following method:

```
def get_course_info() -> None:
```

This method retrieves data using the aforementioned service. In addition to handling *happy path* cases, it should handle the following error scenarios by raising exceptions as listed:

- Missing inputs: `PreconditionException`
- Course is not a valid course: `CourseNotFoundException`
- URL used is either bad or missing: `URLUndefinedException`
- Network is unavailable: `ServiceUnavailableException`. When handling this exception, you may need to handle the `ConnectionError` exception prior to raising the `ServiceUnavailableException`.
- Implement the `__str__` for handling the printing of course information. When printed, the output should look similar to the following, which is simply the subject, course, title, and description:

CSC 2310

Object-Oriented Prgrming/Dsgn

Prerequisites: C or better in CSC 1310. Theory and practice of object-oriented programming and design. Encapsulation, inheritance, dynamic binding, and polymorphism; and introduction to UML and design patterns. Students complete a series of weekly laboratory exercises for developing proficiency in object-oriented programming and design. 0 OR 4 Credit hours 0 OR 3 Lecture hours 0 OR 2 Lab hours Levels: Undergraduate Schedule Types: Laboratory, Lecture All Sections for this Course Computer Science Department

DirectoryContext

The `DirectoryContext` class is used to retrieve directory information about faculty and staff on campus using a query string such as a name, email address, or phone extension. The url to the service is the following:

`https://portapi.tntech.edu/express/api/unprotected/getDirectoryInfoByAPIKey.php?`

and it uses the following parameters:

- `apiKey` – a 35-character key has been provided to you via e-mail
- `searchCriteria` – one of three kinds of inputs are possible here: a name, e-mail address, or phone extension

A well-formed url used to make a call to this service looks like the following (assuming *apikey* is your key):

`https://portapi.tntech.edu/express/api/unprotected/getDirectoryInfoByAPIKey.php?apiKey=apikey&searchCriteria=gannod`

The python code needed to make the call is similar to what was shown above. The json returned will have a format like the following:

```
{ "FirstName": "Jerry",  
  "LastName": "Gannod",  
  "EmailAddress": "jgannod@tntech.edu",  
  "Dept": "Computer Science",
```

```
"Title": "Chairperson",  
"Phone": "931-372-3691",  
"Building": "Bruner Hall (BRUN) 242",  
"POBox": "5101" } ]
```

Note that this is an array of dictionaries that can potentially contain several entries. As before, there is an example in the `bot_webservices` repository showing how to use the service.

Your implementation of the `DirectoryContext` class must do the following:

- Implement getters and setters for the attributes shown in the model
- Implement the following method:

```
def get_directory_info() ->
```

which retrieves data using the aforementioned service. In addition to handling *happy path* cases, it should handle the following error scenarios by raising exceptions as listed:

- Missing input – `PreconditionException`
- Directory entry not found – `DirectoryNotFoundException`. This occurs when the contact person that the user enters is not found in the directory.
- URL is either bad or missing – Occurs when a url is malformed or missing
- Network connection bad – `ServiceUnavailableException`. This occurs when the service is not available. You may need an `except` clause to catch a `ConnectionError` exception.
- Implement the `__str__` for handling the printing of directory information. When printed, the output should look similar to the following, which is an instance of the `DirectoryEntry` class:

```
Jerry Gannod  
Title: Chairperson  
Email: jgannod@tntech.edu  
Department: Computer Science  
Phone: 931-372-3691  
Building: Bruner Hall (BRUN) 242
```

Testing

Unit tests have been provided for you in the `tests` directory. These tests represent a sampling of what will be used to grade your submission. You are required to create your own tests. Guidance on how to do this will be discussed in class. Specifically, for each of the `test_X` methods provided to you, you must create a new similar test method that tests the outputs retrieved from each of the web services.

Submission

You must submit your solution via `git`. No submissions will be accepted in iLearn. To turn in your project, you should stage your changes, and commit as follows:

```
git commit -m "Completed solution"  
git push -u origin master
```

Rubric

Completeness: You will be evaluated on the completion of implementing the two classes (CourseContext, DirectoryContext) and whether the classes attempt to make calls to the web services, implementation of exception handlers and error generators.

Correctness: You will be evaluated on the correctness of your solutions with respect to tests provided as well as tests created specifically for evaluation (that you will not have access to). These tests will exercise both the *happy path* and error paths of the system.

Testing: You will be evaluated on the quality of your own tests.

Submission: You will be evaluated based on the correct submission of your solution back to the repository.

Points: The iteration is worth 50 project points.