Discord Bot Integration for Replit Teams for Education

sddec23-15

Advisor Dr. Zambreno

Team Members Kristen Nathan
Sophie Waterman Hines
Kyle Rooney
Patrick Demers
Cole Mullenbach

Overview

Students in beginner programming classes at Iowa State University frequently have questions about their programming assignments. Since these questions are often repetitive and asked at odd hours of the day, it can take professors many hours to respond. This delay disrupts learning and frustrates young students. From the professor's perspective, taking the time to help each student consumes time from an already filled schedule. The Discord Bot project attempts to answer student questions immediately, without the need for a professor's interaction. When the Bot is unable to assist, it routes the student's question to the professor using a question management flow.

Design

Requirements

Functional Requirements

- The Discord bot shall be configurable for different classes and assignments.
- The Discord bot shall have an easy-to-use interface for configuring the bot.
- The Discord bot shall be able to download up-to-date student code from Replit.
- The Discord bot shall converse with multiple students at a time.
- The Discord bot shall attempt to answer student questions.
- The Discord bot shall provide helpful advice to students based on compiler error messages.
- The Discord bot shall answer pre-configured questions
- The Discord bot shall forward a student question to the teaching staff if the bot is unable to assist.

Non-Functional Requirements

- The Discord Bot shall respond to a student within five seconds.
- All information sent by the bot will be written with correct formatting and spacing to improve readability.
- All bot responses will be easy to understand and formatted in an appealing and organized manner.
- The messages sent by the Discord Bot will support using button reactions for positive/negative feedback.
- The Discord Bot will be capable of escalating the student's request to a professor if its first attempt receives negative feedback.

Engineering Constraints

- The Discord bot shall be written in Python.
- Replit does not offer an Application Programming Interface (API).
- Students ask questions with varying language.
- Questions the bot should answer will change depending on the semester.
- Bot credentials and API secrets need to be secure and accessible.

Relevant Standards

ISO/IEC/IEEE International Standard - Software and systems engineering – Software testing – Part 1: General concepts

Testing is something that is important to every software project. Therefore, general concepts of testing are going to be a standard that aligns very closely with our project. We implemented a variety of system tests, unit tests, integration tests and user-acceptance tests.

ISO/IEC/IEEE International Standard - Systems and software engineering – Design and development of information for users

This standard aligns well with our project because it establishes what information users need, how to determine how that information should be presented, and how to prepare the information and make it available. The main idea for creating our bot is to convey information to the students in Dr. Zambreno's class when asked. We have implemented standards for presenting the information effectively and making it available to the student when asked.

ISO/IEC/IEEE International Standard - Software and systems engineering – Software testing – Part 5: Keyword-Driven Testing

Our project involves extensive testing of keywords and phrases, which will be the majority of our input from real-life users. This standard will help guide us by providing a reference approach to implementing keyword-driven testing and defining requirements on frameworks for keyword-driven testing. Ultimately, this standard will help us create keyword-driven test specifications, create corresponding frameworks, and build test automation based on keywords within our discord bot testing.

Security

Security Concerns and Countermeasures

The discord bot requires a substantial amount of private information in order to operate at its full capacity. These include but are not limited to API secret keys, website and database credentials, and other sensitive information. In order to ensure confidentiality of this information, we've chosen to utilize .env files, a common industry standard for development.

Most commonly, .env files are used throughout software development due to their ability to neatly store confidential data in a local file. .env files are excluded from commits to online repositories, and makes an excellent easy solution to the problem of using so much information. The downside, however, is its ease of access for multiple members on the same project. Often times during development, it may not be possible to create a separate instance of different services cheaply or timely. Therefore, sharing confidential credentials becomes a necessity. Transferring this information (especially if done insecurely) may compromise the integrity of the information and make the privacy .env files provide pointless.

While .env files could be a permanent solution to the privacy issues, we have designed and coded a discord bot to assist in the future switch to a dedicated secrets manager. We elected not to implement this feature do to time constraints as well as our focus on finishing all the bot's main features, but ideally any system being run and accessed by multiple people should have access to privacy system with high availability. Secret managers allow for finer grain access controls as well as auditing. It also allows for multiple users to access the credentials without passing them through potentially unsecured channels. All calls made to the .env file in our code can easily be replaced with API queries to any cloud based secrets manager.

In addition to confidentiality, we also want to ensure the integrity of our SQL queries routed to our database remains high. SQLinjection is a security concern we are focused on mitigating throughout our bot's functions. Since we are connected to a database containing student replit and discord information, we want to ensure that this data cannot be access or changed by any unauthorized user. Unsanitized SQL queries have the potential to take unquestioned input from any malicious user to possible manage, view, or delete database entries. Due to this, input sanitization is a necessity.

We have made all of our SQL query statements resistant to injection attacks by never explicitly giving our parameters to the string containing the remainder of the query. Instead, the values we accept from the user our held separate until the time of execution comes, at which point our python library psycopg checks the input by converting it from a python object to an SQL literal. This prevents code issued by a malicious user from being executed along side the remainder of the query, and instead causes it to be used as the actual values for storage themselves.

Evolution From CPRE 491

Since CprE 491 our bot has stayed relatively the same. Each of our main functionalities and requirements have been a steady force in this project. Things that have changed are the ways we chose to implement them. In the first semester of this class, we hoped to be able to give students helpful hints or

links to resolve their questions as well as using a database of questions. The basis of this implementation stayed true but we chose to utilize AI to query our database of questions to find the right response for students.

We were also able to come up with a very convenient way to store student questions for reuse. Instead of using a database for an asked question, we store the question within the header of the discord message. Doing it this way, there is no database management issues, and it makes the question very accessible.

Implementation

Testing

Test Plan

We had three main systems to test the bot and verify requirements. These plans included end-to-end testing, unit testing, and user testing. For the majority of our work, we used manual testing to check requirements and satisfaction. End-to-end testing was used for a large portion of the bot's functionality: users asking questions, thread management, and providing feedback on compiled Replit code. We decided to utilize automated unit tests for all the logic portions that we could control and were not involved with the Discord API. This included providing answers to student questions, using Algolia to search assignment channels, organizing message threads, and compiling student code from Replit. We also did extensive manual user testing of the user interface and user experience.

End to End Tests:

Scenario: User links to their Replit Account
Actions:

1. Use the /link_replit command and pass Replit username "discordbotcpre1-test" to the bot.
2. Open thread "pp-01".
3. Use the /compile command.

Expected Result: The bot returns an explanation that not enough arguments have been passed.

Scenario: User unlinks their Replit Account
Actions:

1. Use the /link_replit command and pass Replit username "discordbotcpre1-test" to the bot.
2. Use the /unlink_replit command.
3. Use the /compile command.

Expected Result: The bot returns an error message.

Scenario: User asks a known question.
Actions:

1. Use the /question command and ask "why do I need to use return 0?".

Expected Result: The bot explains why return 0 must be used.

Scenario: User uses an answer link.
Actions:

1. Use the /question command and ask "why do I need to use return 0?".
2. Click the "Learn more".

Expected Result: A webpage opens with further explanation.

Scenario: User clicks Problem Solved
Actions:

1. Use the /question command and ask "why do I need to use return 0?".
2. Click "Problem solved".

Expected Result: Problem solved is highlighted green. The other option is disabled.

Scenario: User clicks I need help from the professor
Actions:

3. Use the /question command and ask "why do I need to use return 0?".
4. Click "I need help from the professor".

Expected Result: I need help from the professor is highlighted green. The other option is disabled. A new thread is created.

Scenario: Share thread with Instructor role
Actions:

1. (have instructor role in Discord).
2. Use the /question command and ask "why do I need to use return 0?".
3. Click "I need help from the professor".
4. Send a message saying "testing".
5. Run /share_thread command.

Expected Result: The messages sent in the thread are visible in the parent thread.

Scenario: Share thread with Student role
Actions:

6. (have student role in Discord).
7. Use the /question command and ask "why do I need to use return 0?".
8. Click "I need help from the professor".
9. Run /share_thread command.

Expected Result: An error about missing permissions is displayed.

Test Results

The end-to-end user testing successfully demonstrated and validated the key functionalities of the application from a student/instructor's perspective. A breakdown of results per scenario:

Scenario: User links to their Replit Account
Result: PASS

Scenario: User unlinks their Replit Account
Result: PASS

Scenario: User asks a known question
Result: PASS

Scenario: User uses an answer link
Result: PASS

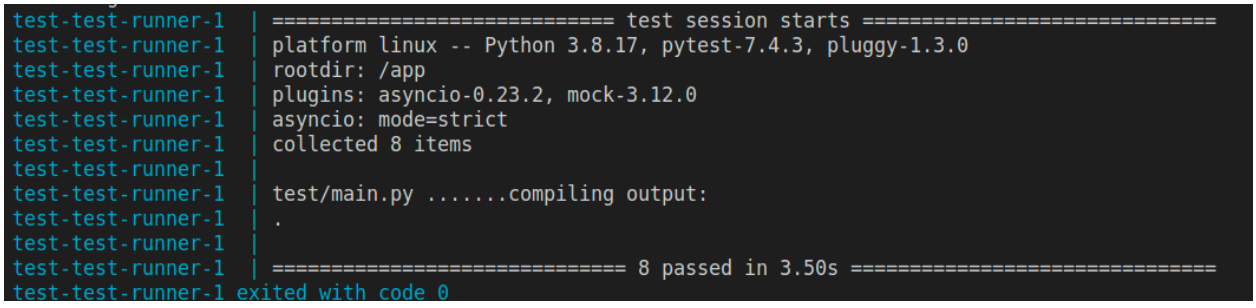
Scenario: User clicks Problem Solved
Result: PASS

Scenario: User clicks I need help from the professor
Result: PASS

Scenario: Share thread with Instructor role
Result: PASS

Scenario: Share thread with Student role
Result: PASS

The unit tests successfully execute and test the "actions" of the application. This provides assurances that the code works and protects against future regressions. Undirected user testing proved users were able to successfully use the bot's capabilities with limited training.



Appendix

Appendix I - Operation Manual

Running in Development

To run using docker, follow these steps:

1. Rename ‘.env.template’ to '.env'
2. Replace the values in the env file with actual values for your personal development.
3. Have docker installed on your local machine.
4. Run *make*. This will spin up your bot.
    - For running test cases, run *make test*.

Creating the Bot in Discord

1. Create a Discord Application
2. Copy the token from the Bot tab.
3. Load token into TOKEN environment variable.
4. In the Bot tab, activate server members and message content intents.
5. In the Discord OAuth tab, go to URL Generator. Select the following:
    - Scopes:
        - Bot
    - Permissions
        - Read Messages/View Channels
        - Send Messages
        - Create Private Threads
        - Send Messages in Threads
        - Manage Messages
        - Manage Threads
        - Embed Links
        - Read Message History
6. Copy the link and paste it into your browser.
7. Complete OAuth to add the bot to the class server.

Roles

The bot is dependent on 3 roles: "TA", "Instructor", and "Admin"

- The TA, Instructor, and Admin roles have the ability to call the share_thread command.
- The Instructor role will be added to a private thread when a student reacts to the bot saying "I need help from the teacher"
    - Note: This can be changed in bot.py by uncommenting out the all-caps commented code within help_button_callback().

The following link shows how to manage roles as well as other useful discord utilities:
https://www.minitool.com/news/add-assign-edit-delete-roles-in-discord.html

Interacting with the Bot

Here are 6 commands that are used to interact with the bot.

- **/close** - Marks a help thread as resolved.
- **/compile** - Compiles student's code given that their have linked their account.
- **/link_replit** - Link user's Replit ID to their discord ID in the database.
- **/question [question]** - This is how a student will ask a typical question to the bot.
- **/reopen** - Mark a resolved help thread as not resolved.
- **/share_thread** - Shares the private thread to the parent channel.
- **/unlink_replit** - Unlinks your Replit account's username from your discord account.

These can all be seen in discord when you type "/" and click on the bot.

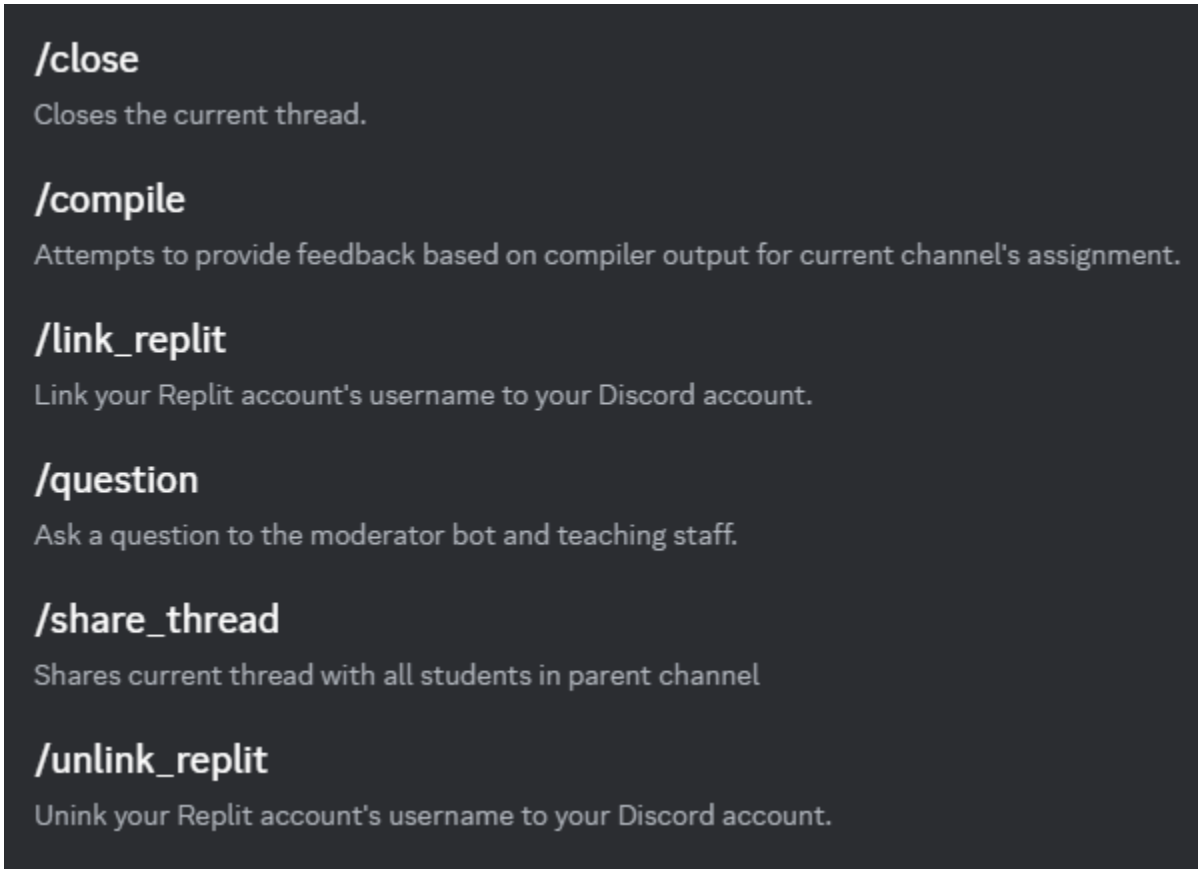


Figure 1: Snapshot of what the commands look like within Discord.

Linking Discord and Replit accounts

To link a student's Replit account with their Discord account, all they need to do it call the /link_replit command and they should be able to successfully link their accounts.

Compiling Code

Once a student's account is linked, they should be able to navigate to the assignment they want to compile in the channels link.

Figure 2: View of the Discord Server's channels.

Once inside the channel, the student can call the /compile command and the bot should compile the student's code and give them feedback based on Algolia's coverage.

Asking Questions

If a student has a question, all they need to do it call the /question command and provide their question as an argument and the bot will provide the student with an answer as best as it can based on Algolia's

coverage. From there, the student can respond by saying, "Problem Solved" or "I need help from the teacher"

Threads

The bot uses a threading system to manage student conversation. When a user clicks on the "I need help from the teacher" button, it will open a private thread with the student and all of the people within the "Instructor" role.

The question will get brought into the private thread where further discussion can be had. Once the problem has been resolved, anyone in the private thread can call the /close command that will mark the thread as resolved. Additionally, any of the following roles: "TA", "Instructor", and "Admin" can call the /share_thread command to share the current thread with all of the students in the parent channel.

Appendix II - Alternative Design Choices

Question Answering Method

When answering student questions, there were multiple approaches considered. The following options were weighed on maintainability, cost, and student experience.

Option 1: ChatGPT
ChatGPT would provide a great answer for the students but is expensive to use and difficult to control.

Option 2: Students choose from question list
In this option, students could be presented with a list of questions and select the one that best suits their issue. This is maintainable and cost effective but does not provide additional information beyond the readme, thus student experience is not significantly improved.

Option 3: Algolia
Algolia is a document indexing service which can match a free text input to the closest matching indexed document. It is cost effective and allows Dr. Zambreno to maintain a JSON file of questions and answers.

Appendix III - Other Considerations

There were no other significant considerations made when designing the Discord bot.