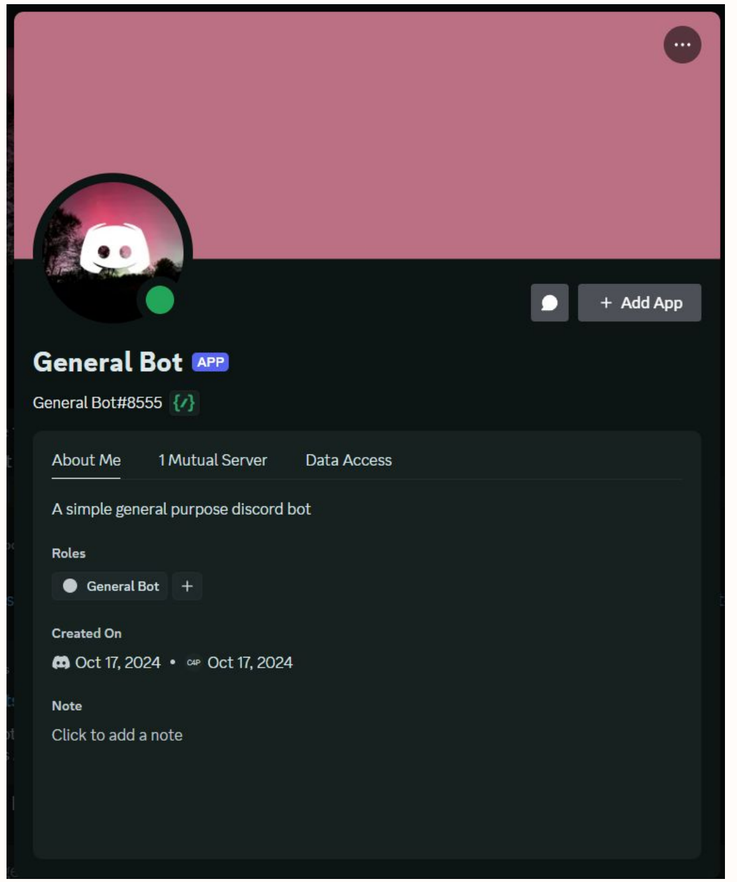
Discord Bot

Team name:Bots
Members: Ishan, Josh, Jared, Gani



Introduction

This Discord bot enhances server management with moderation tools, esports updates, and a user-friendly admin dashboard. Designed for efficiency and engagement, it eliminates the need for databases through a streamlined architecture. Developed by 'Bots,' a team of graduate and undergraduate students.

Elevator Pitch

Our bot, developed by the 'Bots' makes easy management on Discord. It provides a robust set of moderation commands to maintain order, delivers real-time esports updates through commands like **`/tournaments`** and **`/news`**, and offers a secure, user-friendly admin dashboard powered by Discord OAuth. The lightweight architecture, which eliminates the need for a database, ensures efficiency and reliability. Our solution empowers server admins to enhance engagement and streamline management seamlessly.

Overview

Moderation commands for efficient server management.
Admin dashboard with Discord OAuth for secure authentication.
Esports commands for tournaments and news updates.
No database for simplicity and lightweight performance.

Project Goal: "To create a feature-rich bot to enhance moderation and get esports updates."

Overview of Architecture

Diagram of architecture:

- Frontend (Admin Dashboard and OAuth): Jared and Gani.
- Backend (Command handling and Backend): Josh.
- Core Bot Functionality (Commands): Ishan.

Components:

- Discord.js for bot commands.
- OAuth for secure login via Discord.
- REST API for communication between the dashboard and the bot.

For Each Component of Complexity

Feature: Moderation Commands

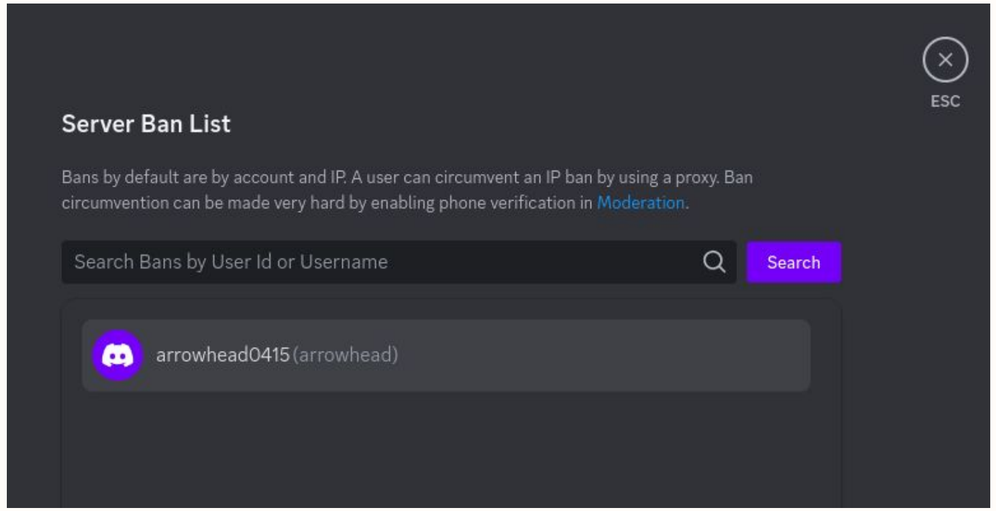
User Story:
"As a server admin, I want commands to timeout, ban, or mute users, so I can manage my community effectively."

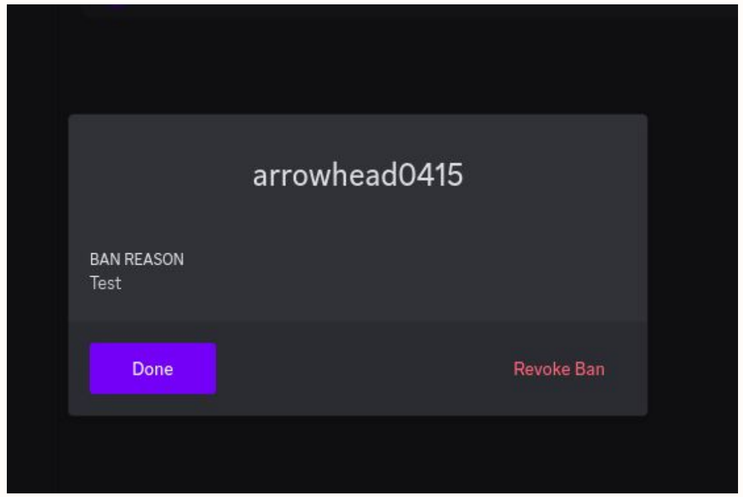
Requirement:
Commands: /timeout, /ban, /mute.
Permissions check for admin roles.

Design:
Sequence diagram of command flow (user → bot → server action).

Implementation:
Code snippets of moderation commands.
Use of PermissionFlagsBits for role-based checks.

Testing Plan Overview:
Correct admin usage.
Unauthorized user attempts.
Edge cases (e.g., banned user rejoining).





Feature: Admin Dashboard with Discord OAuth

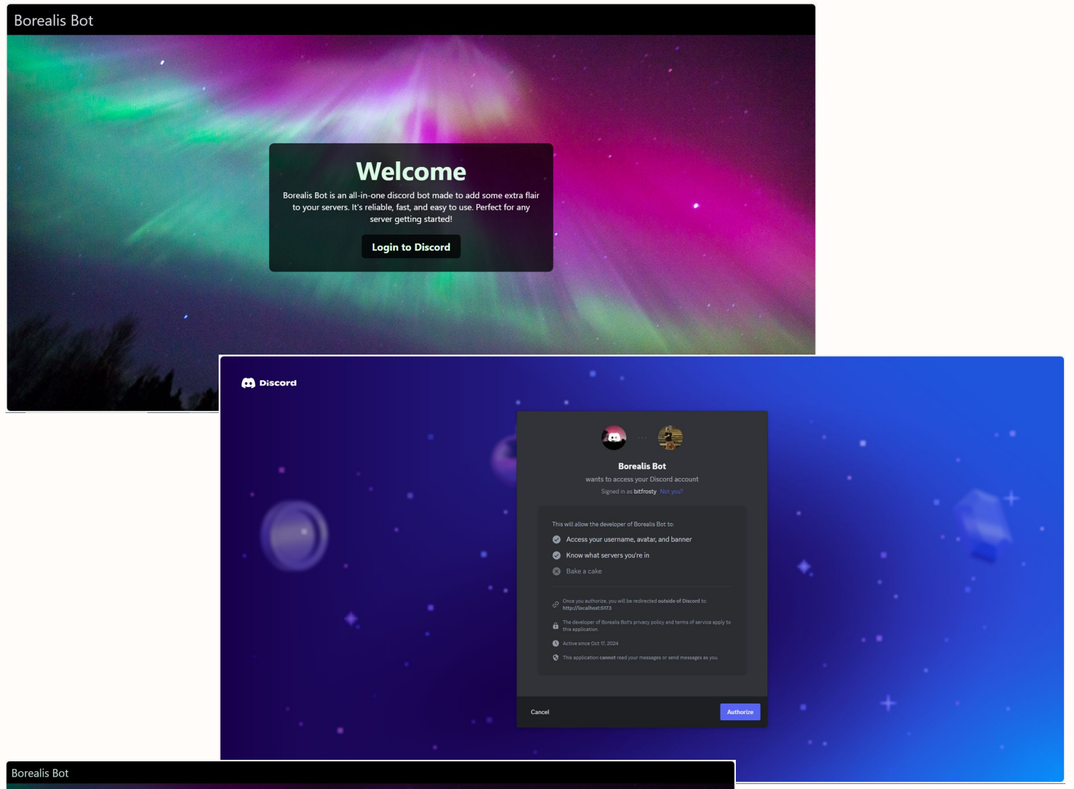
User Story:
"As a server moderator, I want authenticate using discord, and can choose which server to add my bot on."

Requirement:
Secure OAuth-based login.
Responsive UI for desktop and mobile.

Design:
Wireframe of the dashboard interface.
Flowchart of OAuth login process.

Implementation:
Integration with Discord API for OAuth.
Frontend built using modern frameworks (e.g., React or Svelte).

Testing Plan Overview:
Successful login.
Logout and session expiration.
Handling invalid OAuth tokens.



Feature: Help Command

User Story:
“As an end user, I want to be able to use the commands with some form of hints with what is needed to run the command.”

Requirement:
Include /help in the bot to dynamically display available commands.
Provide a description and example usage for each command.

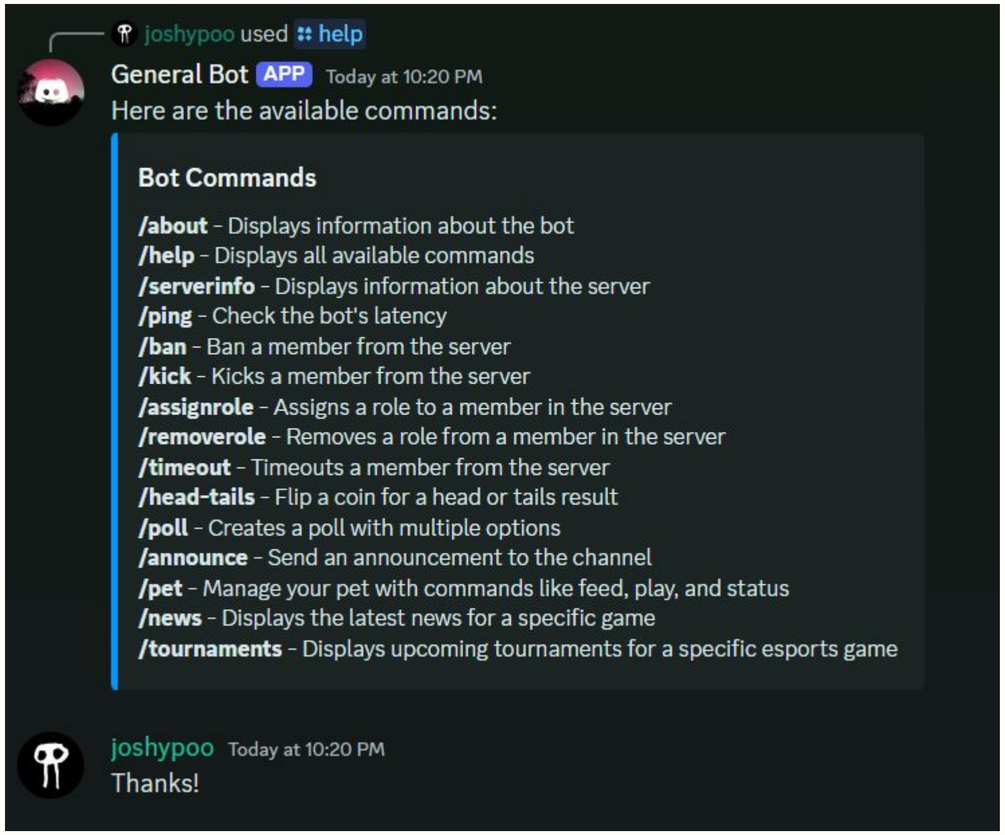
Design:
Command structure includes descriptions, permissions, and examples.
Embed message for a visually appealing and organized layout.
Implementation:

Implementation:

Code for /help to parse commands and return a formatted embed.

Testing Plan Overview:

Validate /help displays commands correctly.
Check formatting and clarity of hints and descriptions.



Conclusion

Recap:
The bot provides moderation, an admin dashboard, and esports information in one package.

Key Learnings:
Collaboration across modules, integrating APIs, and handling bot functionalities without a database.

Future Improvements:
Add a database for user analytics.
Expand esports commands with live event notifications.