High-Level Design

The Layered Architecture would be best as KarmaTracker's architectural pattern, influenced by the system's different functionalities. This architecture's modularity and separation of responsibilities are fit for KarmaTracker, as discrete levels regulate the user interface, point tracking, notifications, and external service communication. The Layered Architecture facilitates scalability by allowing for the addition of new features without disrupting the overall system. Its clear division of responsibilities promotes maintainability and enables for independent development on certain layers. KarmaTracker adheres to acknowledged software development best practices by employing this strategy. The user interface layer manages interactions, the business logic layer manages point tracking, and the integration layer manages external service communication, all of which create a systematic structure.

This layered design ensures that multiple components can interact seamlessly, accommodating the dynamic and growing nature of KarmaTracker. In summary, the Layered Architecture is the design choice for KarmaTracker because it provides a strong framework that combines flexibility, maintainability, and scalability.

Low-Level Design

The behavioral design pattern family would be best for implementing our project, specifically the “Chain of Responsibility” pattern. Due to our bot relying on commands originating from the users directly, following this pattern would be best to follow for writing clean and organized code. Starting with a command, this would be able to be processed and passed accordingly to the associated function that the user is trying to accomplish. For example, when a user is requesting ‘help’ to view to all available commands, the system should not interact with any classes that are used to award points. This design pattern will ensure that at each level through the process that the appropriate code is being executed and processed, while keeping all the classes organized.

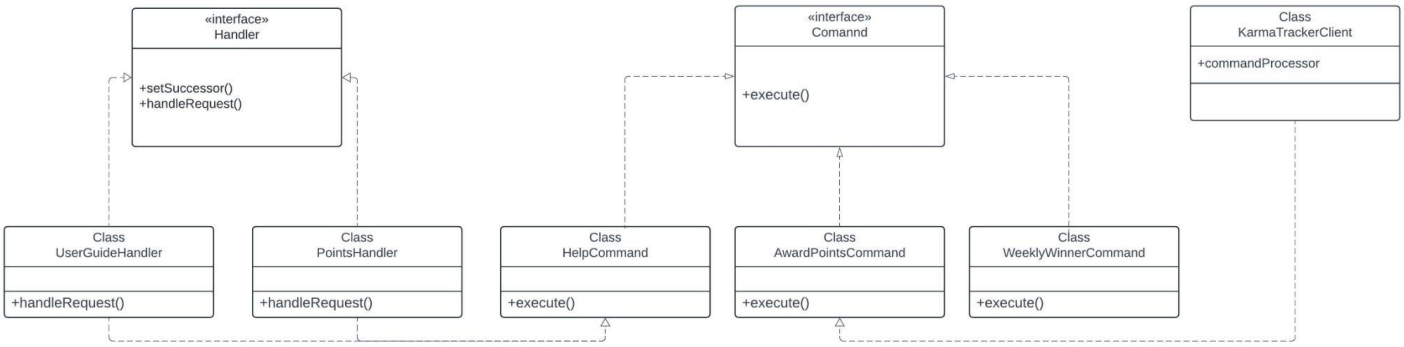
Pseudocode:

```
// Command Interface
interface Command:
    execute()

// Concrete Commands
class HelpCommand implements Command:
    execute(): // Display help information
        ...
class AwardPointsCommand implements Command:
```

```
    execute(): // Process and award points
        ...
class WeeklyWinnerCommand implements Command:
    execute(): // Identify and congratulate top point-jumper
        ...
// Add more classes for commands as needed

// Handler Interface
interface Handler:
    setSuccessor(Handler successor)
    handleRequest(Command command)

// Concrete Handlers
class UserGuideHandler implements Handler:
    handleRequest(Command command):
        if command is HelpCommand: // Display user guide
            ...
        else: // Handles other commands of the bot
            successor.handleRequest(command)

class PointsHandler implements Handler:
    handleRequest(Command command):
        if command is AwardPointsCommand:
            // Process and award points
            ...
        elif command is WeeklyWinnerCommand:
            // Identify top point-jumper
            ...
        else: // Handles other commands of the bot
            successor.handleRequest(command)
// Add more classes for commands as needed

// Client
class KarmaTrackerClient:
    commandProcessor = PointsHandler()

    // Setting up the chain
    userGuideHandler = UserGuideHandler()
    pointsHandler = PointsHandler()
```

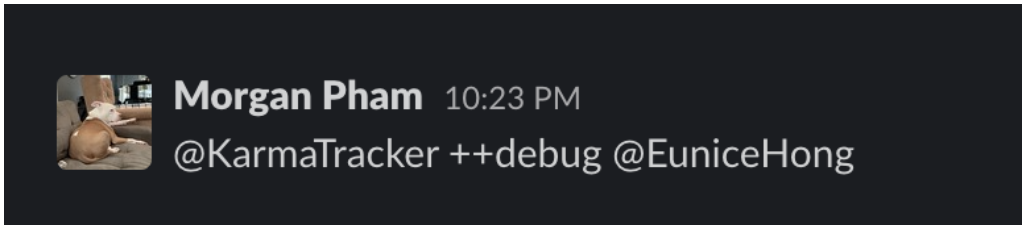
```
userGuideHandler.setSuccessor(pointsHandler)

// Command from user
userCommand = getUserCommand()

// Process the command through the chain
commandProcessor.handleRequest(userCommand)
```

Informal class diagram:

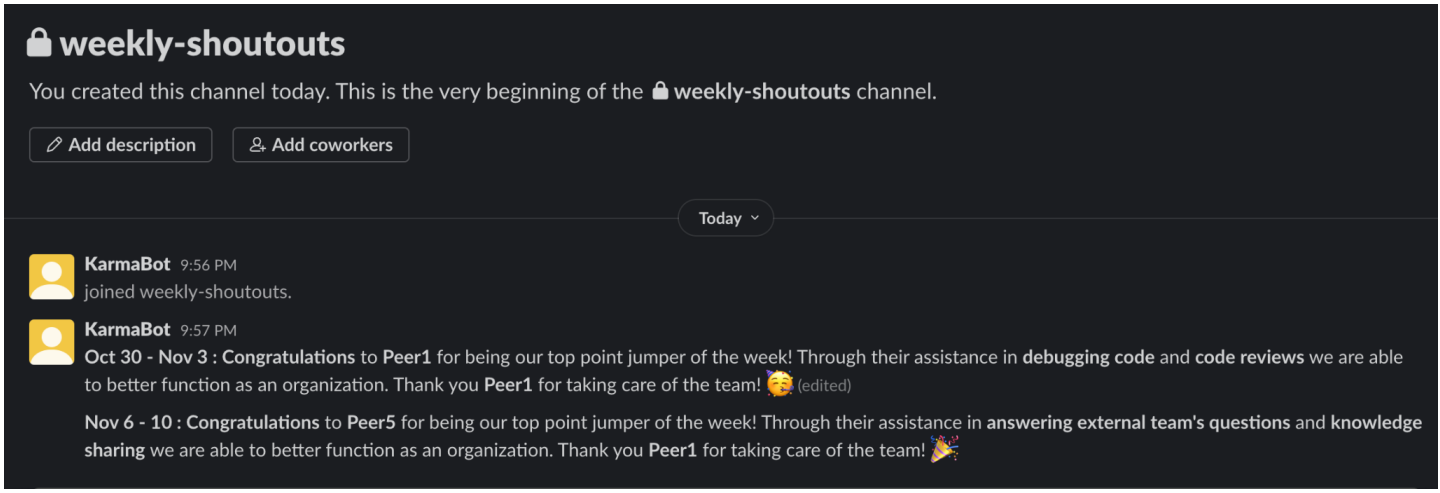


Design Sketch

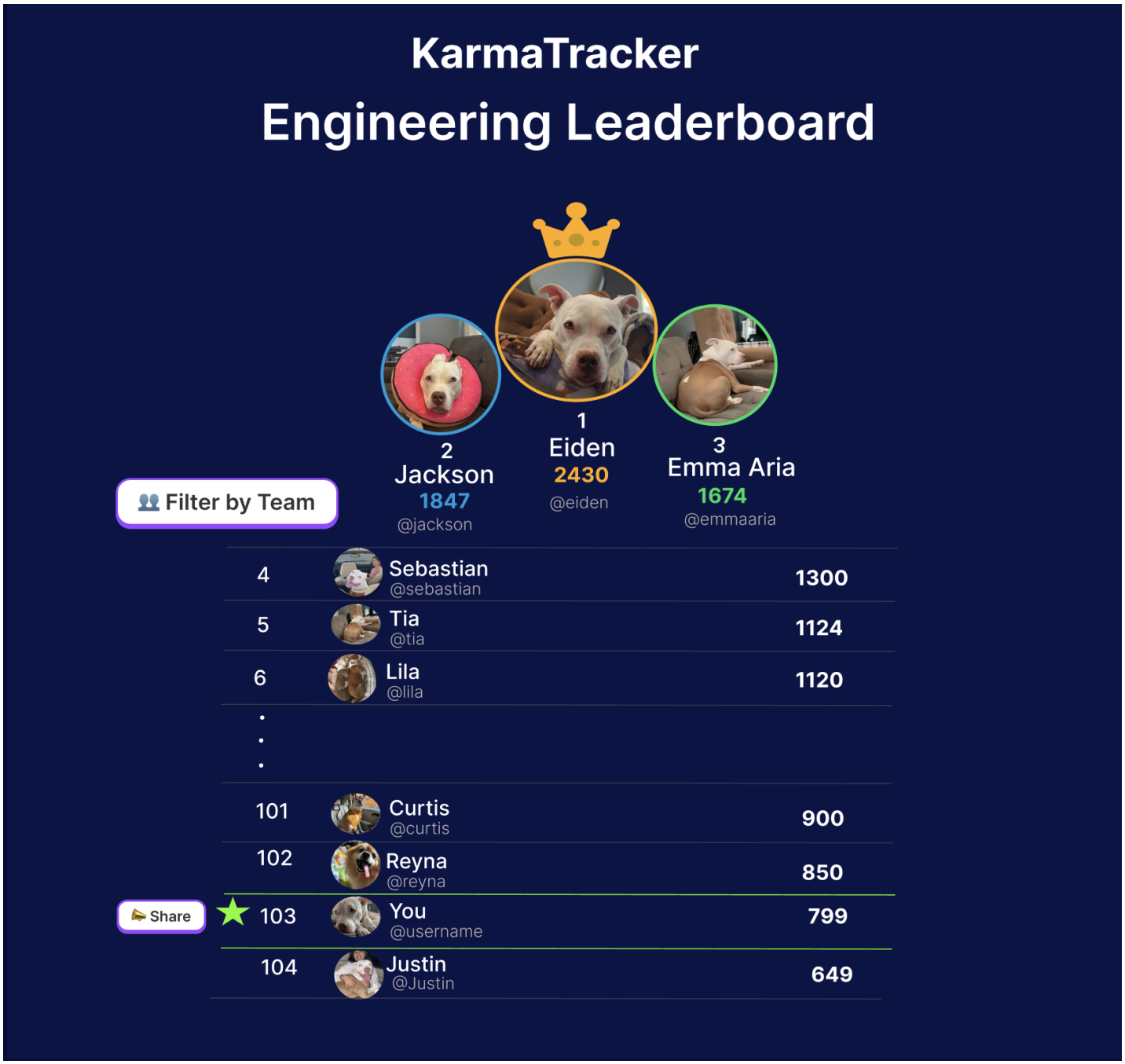
Sending command to KarmaTracker:



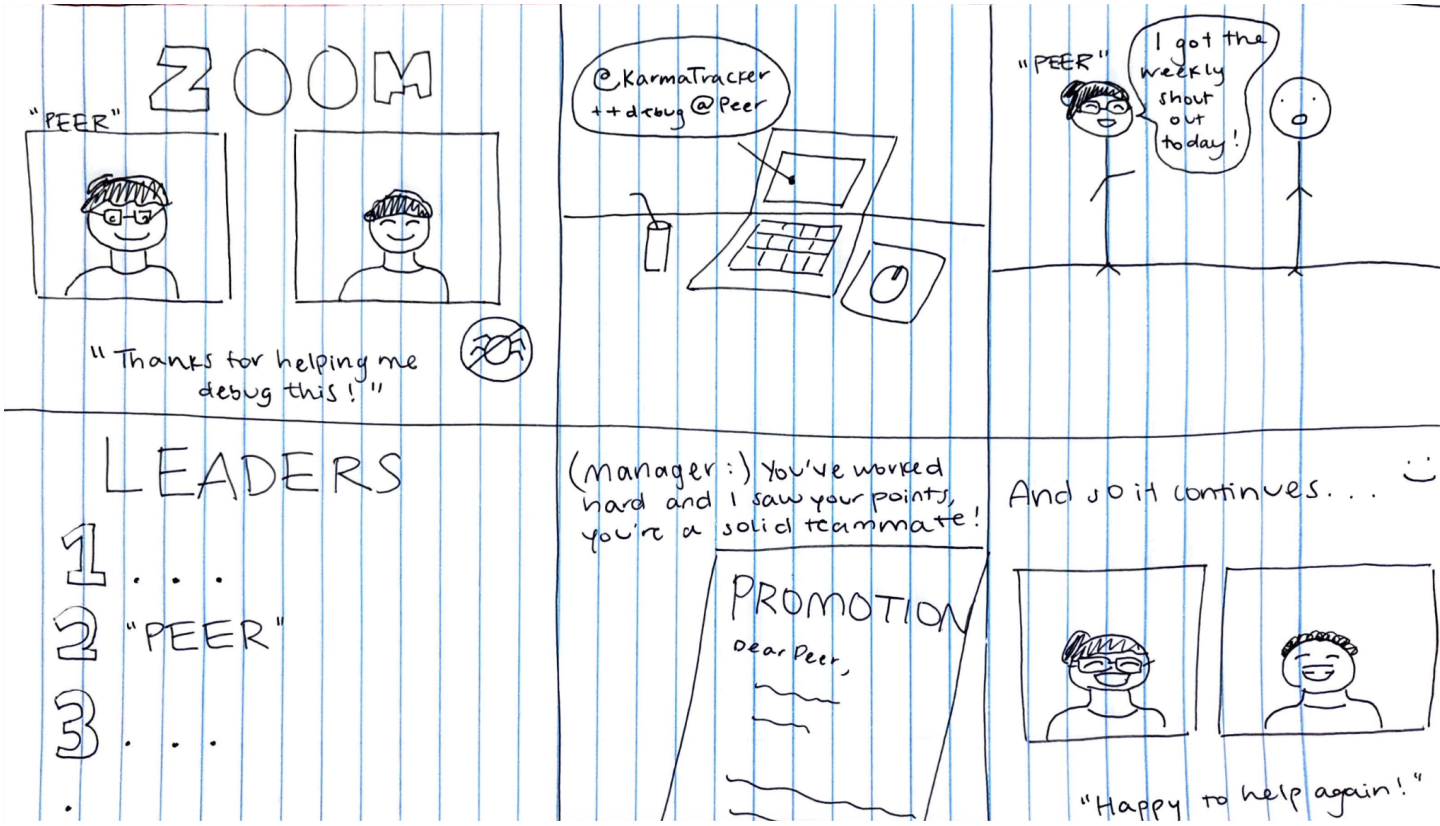
Weekly Shoutouts Slack channel:



KarmaTracker React Application Leaderboard:



Storyboard for primary task:
Primary task is awarding an engineer points for helping to debug an issue.



Brief Rationale:

For the SlackBot's usage, we wanted to keep the commands nice and simple to make it as easy as possible for our users to award other engineers points. We know that it already takes extra effort to thank those that help us, so we want to ensure as many engineers get recognized as possible for the hard work and care they put in to help the team and project overall. This is why the command simply addresses the bot, lets the bot know the specific action to award points for, and the user that the bot should be awarding. The message in the weekly shoutout channel is also meant to be kept simple as to not overwhelm the users in the channel - having constant notifications would likely cause users to mute the channel and therefore miss acknowledgements of hard and helping workers. The leaderboard itself had the most design choices made - we wanted to specifically include the photos of employees on the board as to add a more personal touch and recognize that real people are behind these kind actions. We allow for filtering by team to better narrow down the search results, and we wanted to allow users to share their points with others. The storyboard illustrates many of the design decisions discussed in a more broad scenario where an engineer gets awarded points and eventually their efforts are recognized via promotion.

Project Check-In:

Completed.

Process Deliverable

Notes from most recent Scrum meeting:

Date: 11/9, Topic: Final checks on PM3

Aiden

- Since last meeting, I have worked on our PM3 with the low level design pattern, and completed the survey
- Planning to add relevant documents to our git repository and help with the rest of PM3 if needed
- Nothing blocking me at the moment

Morgan

- I made more progress on PM3 since we last spoke, I managed to get all of the design images done (Leaderboard and storyboard)
- Thought it could be a good idea to also have a visual on the Slack aspect, so will add those in as well
- Will write up the brief rationale on design decisions before Friday
- No blockers at the moment

Vatsa

- Made more progress on our PM3 with the pseudocode and the informal class diagram, since our previous meeting.
- The pseudocode is very brief but if needed in the future can try to implement it entirely.
- No blockers currently.

Eunice

- Made some progress on my section High level design since the last meeting
- I chose an architectural pattern based on the inputs of my group
- Just wrote a quick draft since I was swamped with other work
- Plan to finish later