RewardBot: A smart bot-based reward and recognition system for software engineering teams

Ramaraja Ramanujan
ramaraja@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Shaunak Juvekar
jshaunak@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Dhruveel Lalit Chouhan
dhruveel10@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Ramnath Raghu
rramnath@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Fasi Ullah Khan Mohammed
fasi@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Abstract

Existing rewards and recognition systems for software engineering teams suffer from multiple flaws, including a lack of transparency, difficulty in quantifying rewards, and poorly defined actionable objectives. As a result, employee discontent arises, leading to alarming trends such as "Quiet Quitting" and "The Great Resignation." To address these issues, we introduce "RewardBot", a Slack bot that utilizes gamification principles to recognize and reward employees effectively. RewardBot awards points based on individuals' contributions to projects and their involvement in the open-source community. These accumulated points are displayed on a leaderboard and can be redeemed for various rewards, boosting employee motivation and fostering a healthy competitive environment. In addition, RewardBot leads to improved project outcomes by encouraging employees to follow clearly defined goals that align with the organization's objectives. Furthermore, RewardBot enhances overall team productivity, ensures transparency in the rewards system, and heightens the visibility of individual impacts on projects. By addressing the critical shortcomings of conventional rewards and recognition systems, RewardBot offers a promising solution to curb employee dissatisfaction and retain talent in software engineering teams. The code is available at https://github.com/r-ramaraja/RewardBot.

CCS Concepts: • **Software and its engineering → Collaboration in software development**; *n-tier architectures*; • **Human-centered computing → Reputation systems**.

Keywords: Rewards, Recognition, Collaboration

ACM Reference Format:
Ramaraja Ramanujan, Shaunak Juvekar, Dhruveel Lalit Chouhan, Ramnath Raghu, and Fasi Ullah Khan Mohammed. 2023. RewardBot: A smart bot-based reward and recognition system for software engineering teams. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (ICSE '23)*. ACM, Melbourne, Australia, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The changing landscape of the workplace, exacerbated by the COVID-19 pandemic, has brought forth new challenges for organizations, particularly in retaining talented employees. The emergence of trends such as "Quiet Quitting" [3] and "The Great Resignation" [19] highlights the growing dissatisfaction among employees regarding the meaning and value derived from their contributions. In response, organizations are seeking to arrest attrition rates by employing innovative strategies to improve employee retention and enhance their rewards and recognition systems.

Traditional rewards and recognition systems in the workplace have proven inadequate in meeting software engineering teams' evolving needs. These systems often lack transparency, fail to quantitatively measure contributions, and lack clearly defined actionable objectives for employees to pursue recognition and rewards. Moreover, traditional methods may introduce biases and discrimination, as subjective evaluations by managers or panels can inadvertently favor certain individuals or discriminate against others. These shortcomings not only contribute to employee discontent but also result in reduced productivity for both the team and the organization as a whole. It is evident that a significant overhaul of existing rewards and recognition systems is necessary to address these issues effectively.

To tackle these challenges, this project proposes integrating gamification [10] principles into the rewards and recognition system in the workplace. Our solution, "RewardBot," is a Slack bot [7] designed specifically for software engineering teams. RewardBot introduces a novel approach where employees can be recognized for their contributions and rewarded through a point-based system. These points can be manually awarded by managers or peers, acknowledging achievements such as resolving critical production issues or providing assistance in debugging complex problems. Additionally, an automated system can assign points for objective scenarios, such as committing new features to a Git repository or successfully completing sprint tasks without rollover issues in Jira. The accumulated points within RewardBot can then be redeemed for various rewards, ranging from organizational merchandise to time off, coupons, or even a lunch with the CEO. A key feature of RewardBot is the inclusion of a weekly leaderboard, visible to all organization members, which provides greater recognition to top performers.

By implementing this new reward and recognition system, software engineering teams can experience a positive shift in the meaning and value derived from their contributions. Team members gain immediate feedback on their achievements, and the system boosts productivity by promoting a healthy work culture where people don't become complacent by doing only the bare minimum. Moreover, the introduction of RewardBot allows organizations to align employees' goals with the organization's overall objectives. It also enables identifying and retaining top performers, thereby reducing costs associated with rehiring and retraining. In addition to the productivity and cultural benefits, this system allows organizations to monitor and nurture talent effectively. By addressing the critical shortcomings of conventional rewards and recognition systems, RewardBot offers a promising solution to curb employee dissatisfaction, improve retention rates, and ultimately contribute to the success of software engineering teams.

2 Motivating Example

Consider this motivating example. Suppose a team of software engineers has been working on a project for a few months. They have regularly pushed their code to GitHub and reviewed each other's work. Using RewardBot, team members can award points to each other for helpful code reviews, resolving a critical bug, or even staying late to finish a task. The leaderboard can showcase the top performers and encourage healthy competition within the team, while the reward catalog can give employees a tangible incentive to work towards it. RewardBot can help motivate team members and foster a positive work culture.

3 Related Work

Bravo [14] is an AI-based employee recognition software available on multiple platforms, including smartphones, PCs, and web extensions. It utilizes AI algorithms to collect and analyze data from various sources to determine the top candidates for recognition. Bravo's main objective is to give employees the recognition they deserve. The software's Data Analytics feature offers deeper insights into each employee's work, providing valuable analytics and visualizations through graphs. The spotlight feature implemented in Bravo has influenced our decision to incorporate a leaderboard into our bot. By introducing a leaderboard, we aim to promote healthy competition among employees, with the added incentive of rewards.

Bonusly [17] is an online service that rewards, recognizes, and celebrates employees. The system also automatically sends invitations or rewards to commemorate special events such as birthdays or work anniversaries. From Bonusly, we have drawn inspiration for features like integration with other apps. As Slack is widely used by many teams, incorporating our bot within the Slack platform proves advantageous. Integrating the bot into Slack allows employees to conveniently access the leaderboard, track their points, and view available rewards immediately after completing tasks or on an hourly basis, thereby providing them with constant motivation. Additionally, the concept of a reward catalog introduced in Bonusly has been instrumental in our decision to include a redemption system for accumulated points in our bot.

Stoeckli et al. [18] have conducted extensive research on the development and potential of chatbots, specifically in the context of connecting social interactions in instant messaging apps with external systems and corporate operations. Their work highlights the underexplored affordances of Slack apps and integrations in the workplace. The authors emphasize how chatbots can facilitate bottom-up automation by integrating social information and traditional corporate systems. They also highlight the emergence of novel shared affordances as individual affordances are actualized and how the actualization of one user's affordance can influence the perception of affordances by other users. This research sheds light on the novel phenomenon of chatbots within enterprise messengers. Their findings have motivated us to deploy our solution as a chatbot rather than a traditional website, leveraging chatbot technology's unique advantages and opportunities.

In another study, Leubef et al. [5] discuss the different types of bots, platforms for working with them, and guidelines for their creation and use. They emphasize the increasing dependence of businesses on bots, which have become integral to business operations. However, the authors also highlight the importance of addressing potential issues related to information overload and interruption overload caused by poor bot design or excessive use. Moreover, the ethical considerations outlined in Isaac Asimov's "three laws of Robotics" are also relevant to software bots, emphasizing the importance of avoiding harm, obeying commands, and self-protection. This paper provides valuable insights into the landscape of chatbots and offers guidance on designing bots ethically that enhance users' capabilities rather than impeding them.

While the aforementioned studies and implementations have significantly contributed to understanding chatbots and their implications in the workplace, our project introduces a novel application of chatbots specifically tailored to addressing the shortcomings of existing software engineering teams' rewards and recognition systems. By leveraging the advantages of chatbot technology and incorporating gamification principles, RewardBot aims to enhance employee motivation, productivity, and retention within software engineering teams. This unique approach offers a promising solution to the challenges faced by organizations in recognizing and rewarding the contributions of software engineers effectively.

4 Implementation

4.1 Requirements

To determine the requirements for RewardBot, our team took turns roleplaying as the customer/end-user. Through this approach, we were able to identify and prioritize the functional and non-functional requirements of the bot.

The following functional requirements were identified:

- Employees must be able to award points to each other for their contributions to projects and the open-source community.
- There will be three types of awards: Mastermind (100 points), Wizard (50 points)and Rising Star (25 points).
- GitHub contributions must be recognized and rewarded.
- Show a leaderboard with the top performers to promote healthy competition and provide recognition.
- An employee can be awarded only three times in a quarter

Non-functional requirements are as follows:

- The application must be responsive and able to respond to requests within 3 seconds.
- The application must have an uptime of 99.99% reliability to ensure that users can access the bot whenever they need to.
- Only authorized people within the organization to access the bot invokes it and allocated rewards
- The bot should be able to support at least 100 concurrent invocations at a given time
- The application must be easy to maintain and update, with clear documentation and well-structured code to facilitate future development.

4.2 Architecture

As illustrated in Figure 1, RewardBot is built on a 3-tiered architecture. We structured the application using a 3-tiered architecture [6] as the three layers in a 3-tiered architecture represent distinct concerns (presentation, business logic, and data storage), which makes it easier to design, implement, and maintain the application. It makes it easier to scale the application horizontally by adding more instances of each layer if required. Separating the layers makes it easier to apply security measures at each layer, which can help reduce the risk of security breaches. Also, the application can be optimized for performance at each layer, which can help to improve overall application performance.

The first tier consists of integrating the bot with Slack, where users interact with the bot using Slack's slash commands. The commands issued by the user are sent over an HTTP request to the middle-tier back-end service. In this tier, the command is parsed, and appropriate steps are taken to process the request and return a response to the user. The final tier is the data layer, which interfaces with the database to perform various actions such as storing and redeeming points, auditing commands, and persisting user details. By utilizing this architecture, RewardBot can effectively manage data storage and processing while maintaining a high degree of scalability and modularity.

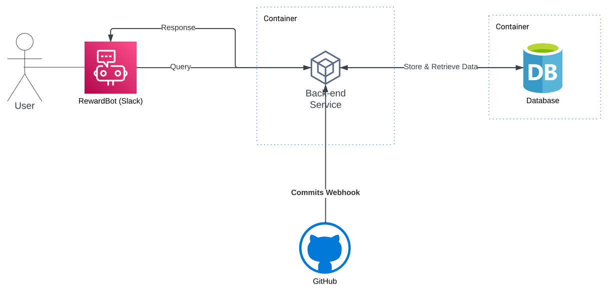


Figure 1. Architecture Diagram

4.3 Design Decisions

In developing RewardBot, we made several crucial design decisions to ensure its effectiveness and efficiency. Firstly, due to the data-centric nature of the application, we opted for a MySQL database. MySQL is open-source, reliable, and strongly adheres to the ACID (Atomicity, Consistency, Isolation, and Durability) properties. This was crucial in ensuring data consistency and avoiding any mismatch in the points allocated to users.

Python was our programming language of choice for developing RewardBot. We chose Python due to its widespread support, extensive documentation, and large community of developers. The `mysql-connector-python` [13] package was utilized for connecting to the MySQL database. Although pymysql [12] was faster, we opted for the officially supported mysql-connector-python package due to its well-documented examples.

After considering several messenger platforms such as Discord, Skype, WebEx, etc., we decided to use Slack as our preferred platform. Slack is widely used in the industry and is developer-friendly, making it an ideal platform for our bot. We utilized the Slack Bolt [16] framework as it was recommended by Slack and had a rich ecosystem that made development fast and easy.

Additionally, external integrations with tools and sites like GitHub were based on event-driven webhooks to avoid manual invocation or polling. To facilitate communication between the bot and the back-end services, we designed the REST APIs to be robust and conducive to scaling. The Singleton [9] design pattern was used to ensure the creation of a single database instance used for performing CRUD operations. Furthermore, the Observer [8] pattern was implemented for the automated points reward system. The service listens to new GitHub commit push events and responds by allocating points to the commit's author.

4.4 Features

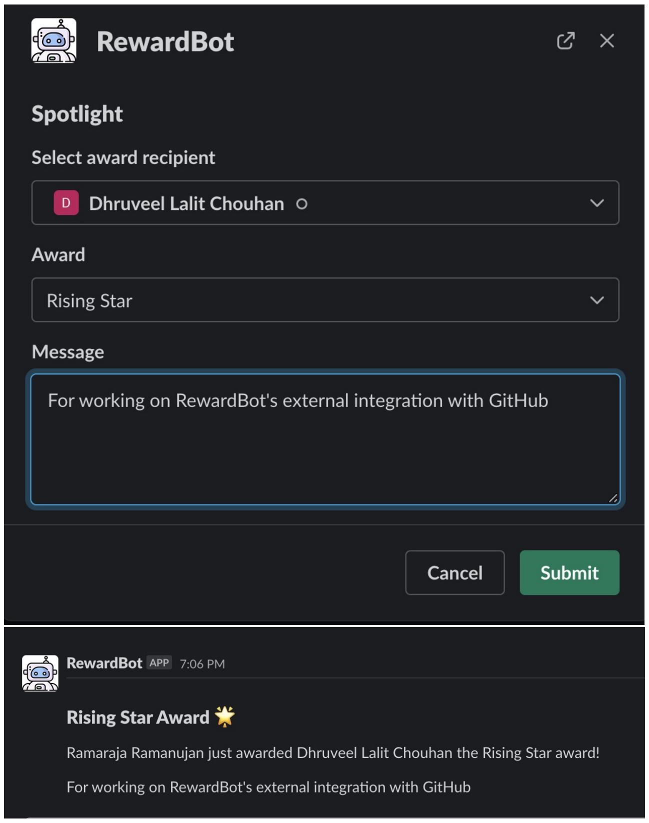
4.4.1 Award Points. The award points feature allows employees to recognize each other for their contribution. The points could be awarded by the manager to their direct report or it could also be peer to peer. The user issues a Slack slash command by typing `/reward`, which opens up a modal, as seen in Figure 2, in which you can type the details of the recipient, award type and a message detailing why they received the award. Once you click on submit, the dedicated public bot channel will receive a message, as seen in Figure 2, containing the details of the award. The user story for the feature is given as follows:

As an employee, I want to award points to my peers to reward good quality work in the particular project

Acceptance Criteria

- **Given** employee has the permission to interact with the bot
- **When** employee invokes the bot to award points for their peer's work
- **Then** bot accepts the information
- **And** stores the points
- **Then** displays a message to the team announcing the reward

4.4.2 External Integration. For the scope of this project, we limited external integration to just GitHub. The GitHub integration REST endpoint of RewardBot must be added to the GitHub repository's webhooks. And then, when a user merges a pull request (PR) in a GitHub repository it will trigger the webhook that will call RewardBot and RewardBot


RewardBot

Spotlight

Select award recipient: Dhruveel Lalit Chouhan

Award: Rising Star

Message: For working on RewardBot's external integration with GitHub

RewardBot APP 7:06 PM

Rising Star Award 🌟

Ramaraja Ramanujan just awarded Dhruveel Lalit Chouhan the Rising Star award!

For working on RewardBot's external integration with GitHub

Figure 2. Award Points

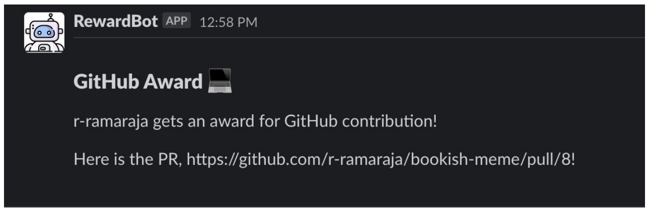
will automatically award 10 points to the author of the PR. The dedicated public bot channel will receive a message, as seen in Figure 3, announcing the details of the award. The user story for the feature is given as follows:

As an employee, I want the bot to recognise and reward my latest project contributions on GitHub

Acceptance Criteria

- **Given** the webhook has been configured in the repository
- **When** employee merges a PR
- **Then** bot received the webhook event and awards the points
- **And** stores the points
- **Then** displays a message to the team announcing the reward

4.4.3 Leaderboard. The leaderboard feature allows users to see the top ten performers.The user issues a Slack slash command by typing `/leaderboard`, which prompts the RewardBot to send the leaderboard as a direct message, as seen in Figure 4,. The user story for the feature is given as follows: As an Employee, I want to check the leaderboard so that I


RewardBot APP 12:58 PM

GitHub Award

r-ramaraja gets an award for GitHub contribution!

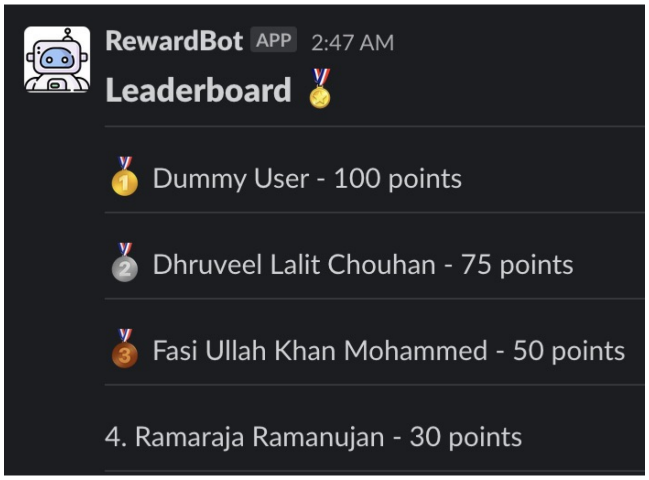
Here is the PR, https://github.com/r-ramaraja/bookish-meme/pull/8!

Figure 3. GitHub Award

can find out top 10 employees

Acceptance Criteria

- **Given** employee has the permission to interact with the bot
- **When** employee asks the bot show leaderboard
- **Then** bot fetches the leaderboard and returns a list of the top 10 employees.


RewardBot APP 2:47 AM

Leaderboard

Dummy User - 100 points

Dhruveel Lalit Chouhan - 75 points

Fasi Ullah Khan Mohammed - 50 points

4. Ramaraja Ramanujan - 30 points

Figure 4. Leaderboard

4.5 Process

For the development of RewardBot, we adopted an Agile [4] process. Agile development methodologies emphasize flexibility, collaboration, continuous improvement, and risk reduction, making them ideal for software development. Additionally, as we were working in a dynamic and rapidly changing environment where requirements and priorities were constantly evolving, the Agile approach was particularly effective.

We opted for a Kanban [1] methodology within the Agile process. We chose Kanban because we felt that our team could self-organize and manage our work well. Additionally, we wanted a simple, lean methodology that emphasizes continuous improvement, visualization of work, and limiting work in progress. We visualized the work on a project board on GitHub Projects and used metrics such as throughput and cycle time to optimize the flow of work through the system. The Kanban approach allowed us to work collaboratively and flexibly while ensuring we met our deadlines and delivered a high-quality product. By continuously improving and visualizing our work, we could track our progress, identify areas for improvement, and ensure that we were meeting our goals.

In addition to Kanban, our team utilized code reviews and pair programming during implementation. Code reviews were used as a way to provide constructive feedback and improve the overall quality of the codebase. Pair programming, on the other hand, was used to tackle particularly challenging or complex features, where two developers would collaborate in real-time to come up with the best possible solution. Both of these practices were instrumental in improving our development process.

4.6 Tools

Throughout the course of the software engineering project, we utilized a range of tools to support our development efforts. These included:

- Zoom: We utilized Zoom to facilitate regular meetings and collaborate effectively on the project.
- Slack: We leveraged Slack as our preferred instant messaging (IM) platform, allowing for efficient communication and quick updates among team members.
- GitHub: We utilized GitHub as our Version Control System (VCS), allowing us to continuously integrate our code and manage changes effectively.
- GitHub Projects: We used GitHub Projects to visualize our work, manage tasks, and ensure that we stayed on track with our project goals.
- GitHub Issues: We used GitHub Issues to keep track of issues, especially bugs, code smells and technical debt.
- Render: We used Render (https://render.com) to continuously deploy the project to the cloud, ensuring that it was easily accessible and could be scaled as needed.
- VS Code: We opted for VS Code as our Integrated Development Environment (IDE) of choice, given its open-source nature and strong support within the development community.
- Overleaf: We utilized Overleaf to prepare our project proposal and report documents, allowing us to collaborate effectively and produce high-quality, professional-looking documents.
- SonarQube: We used SonarQube for static code analysis for metrics like cyclomatic complexity and cognitive complexity.

4.7 Testing

Following the completion of development, we conducted validation testing to ensure that the RewardBot met our desired specifications. We adopted a rigorous testing approach, in which we took turns playing the role of tester and manually tested each feature. This involved both individual testing of each team member's features and cross-testing of the features developed by other team members.

Our testing approach combined elements of black-box and white-box testing with the goal of identifying bugs and unexpected behavior. We also made sure to test for edge cases, intentionally pushing the application to its limits to identify any subtle issues that might have gone unnoticed.

Once we had completed most of the features, we conducted a round of system testing to verify that the application met all functional and non-functional requirements. To further enhance the robustness of our testing approach, we also added unit tests as needed, thereby automating a part of the testing process and ensuring that it was both efficient and effective.

5 Deployment

For development and demo purposes, we set up the environment locally. Our configuration in the local environment involved each member of the team creating their own Slack app and configuring it. Initially, we used Slack app's socket mode in Slack, negating the need to provide a REST URL. However, later on, we used Ngrok [11] to tunnel the localhost port to the internet, which allowed us to closely simulate the production environment.

Once we were satisfied with how the application was performing locally, we were ready to deploy it to a production environment. Before deploying to production, we ran a suite of unit tests and did a round of validation and system testing to ensure that there are no errors. Initially, for the production environment, we set up an AWS free tier virtual machine (VM). We manually deployed the code on the VM by cloning the latest changes from the GitHub repository and then running the service in the background using `nohup`. However, since the pace of development was rapid, we had to do multiple deployments in a day. Manual deployment became tedious after a while, so we figured out how to automate it.

5.1 Continuous Integration

For continuous integration, we followed the Git Feature Branch Workflow [2], which involves creating a separate branch for each new feature and merging to `main` once the feature is completed through a pull request. The workflow helps to isolate changes and minimize the risk of introducing bugs or conflicts in the main branch. It also allows for easy collaboration among developers, as each developer can work on their own feature branch without interfering with others. Moreover, submitting PRs gave me an opportunity to provide valuable feedback through code review. Before merging the PR to `main` branch, we ran unit tests to ensure that there were no regression errors.

5.2 Continuous Deployment

For continuous deployment, we used GitHub Actions, which allowed us to automate tasks such as running unit tests, SonarQube code quality checks, PyLint static code analysis, and finally deploying the application. With the help of GitHub Actions, we were able to deploy changes more rapidly and reliably without worrying about breaking changes. We also employed the use of git tags to correctly mark the last stable version so as to roll back in case the deployment goes wrong. We also switched from using an AWS VM to render.com [15] for our production environment, as the former required a lot of overhead regarding setup and maintenance, whereas the latter was self-managed as it was a Platform as a Service (PaaS). We automated the deployment such that when a new commit is pushed to `main` it will automatically trigger a deployment to a container on render.com. We chose to go with a basic deployment strategy as it was a small project that is intended only for the duration of the semester with limited users. Although we do acknowledge that this basic strategy carries a significant amount of risk. It lacks the ability to roll back in case of failure and is not outage-proof. Furthermore, any changes to the target environment can potentially cause the app to fail once deployed.

5.3 Maintenance

After deployment, we did face some bugs that slipped through the cracks. Most of our post-deployment time went into corrective software maintenance, where we had to triage and fix bugs. The next most time spent area was towards perfective software maintenance, where we kept refining the codebase in terms of file organization, removing hardcoding, and adding new features. And lastly, we did quite a lot of adaptive software maintenance, like moving from an AWS VM to render.com and moving away from manual tasks to GitHub Actions.

To improve the maintainability of our code, we started implementing unit tests and created detailed documentation of the setup, development, and deployment processes using README.md files and the GitHub Wiki. Refactoring the code to improve readability and maintainability was another important aspect we focused on. We renamed variables to be more descriptive, added comments and reorganized methods to improve the code's structure. Regular code reviews also helped identify technical debt and code smells requiring refactoring. We implemented monitoring using GitHub badges and tracked issues using the GitHub issue tracker. We also used render.com to view logs and check the status of builds.

Incorporating these steps greatly reduced the incidence of bugs and also instances of team members calling each other to figure out how something worked. Documenting the application also helped to relook at some of our choices and redesign the application for better maintainability and quality. It increased our overall productivity and significantly reduced bug triage time. Taking these steps would be beneficial not only to us but also to future team members who may want to work on the code.

6 Limitations

Despite our best efforts, there were some limitations to our RewardBot that we could not overcome. Firstly, our bot is only available on the Slack platform, which limits its usability to teams that use Slack. As such, users of other platforms will not be able to take advantage of the benefits of our bot. Additionally, we were limited in the number of external integrations we could add to the bot. We chose to integrate with GitHub, but there are other platforms we could not integrate with due to time constraints.

Furthermore, while we tested our bot rigorously before deployment, there is always the possibility that some bugs may remain undetected. This is especially true when considering our bot's wide range of possible user interactions. In addition, since we deployed the bot to a containerized environment, there may be limitations to our application's scalability that we could not foresee.

Finally, our bot is only as effective as the users who interact with it. If users do not use the bot regularly or provide accurate feedback, the insights generated by our bot may not be useful or accurate. The success of our bot's gamification aspect depends on users' willingness to participate in the leaderboard and rewards system.

7 Future Work

Our RewardBot implementation is just the first step towards building a comprehensive employee reward and recognition system. Here are some ideas for future work:

- More integration with other external systems: We currently have integration with GitHub, but there are many other external systems, like JIRA, Salesforce, etc., that teams use to manage their work. Integrating these systems will give employees a more comprehensive view of work and contributions.
- Gamification: Adding game-like elements to the reward system could make it more engaging and motivate employees to participate more actively. For example, badges and rewards could be associated with levels, and employees could level up as they earn more points.
- Rules: Currently, the bot can be used indiscriminately to give rewards for the sake of it. We could improve upon this by having peer-to-peer rewards approved by the manager before being awarded. We could incorporate more rules to make the system foolproof and fair.
- Leaderboard 2.0: Improve the leaderboard by adding more metrics and visualizations. Currently, the leaderboard only shows the top 10 performers, but we plan to add more metrics, such as point distribution and frequency of recognition. Additionally, we plan to incorporate visualizations such as graphs and charts to help users better understand their performance.
- Customizability: Currently, the bot has a fixed set of commands and rewards, but in the future, we plan to allow administrators to customize the bot to better suit their company culture and needs.
- Points Redemption: Points redemption feature that would allow employees to exchange their accumulated points for tangible or intangible rewards, further incentivizing positive behaviors and increasing engagement with the platform.

In conclusion, our RewardBot implementation provides a solid foundation for building a comprehensive employee reward and recognition system. However, there are many avenues for future work, and we hope that our implementation will inspire further development and exploration in this area.

8 Conclusion

In conclusion, the current rewards and recognition systems used by organizations suffer from several drawbacks, that cause employee discontent and hence the system needs a complete overhaul. In this project, we presented RewardBot, an innovative chatbot solution that enables peer-to-peer recognition and rewards within the context of an organization. We explained the features of the bot, the architecture and design decisions, the process, the tools used, and the testing and deployment strategies employed. We also discussed the limitations of our implementation and presented possible directions for future work.

The implementation of RewardBot demonstrated the potential for chatbots to be used in organizations for employee recognition and rewards. Using Slack as a platform and integrating it with other external tools like GitHub enabled seamless and efficient communication and recognition among team members.

However, the current implementation has some limitations, such as the lack of a reward redemption system and limited scope for personalization. Future work could improve the system's personalization capabilities by adding features allowing users to customize their profiles and award categories. Implementing an automated reward redemption system using blockchain technology could also make the system more secure and trustworthy.

In conclusion, RewardBot presents a promising solution for enhancing employee recognition and rewards within organizations. With the growing adoption of chatbots in the workplace, we believe that RewardBot's implementation will inspire further research in this area, leading to the development of more robust and personalized chatbots that can be used to improve employee engagement and motivation.

References

[1] Muhammad Ovais Ahmad, Jouni Markkula, and Markku Oivo. 2013. Kanban in software development: A systematic literature review. In *2013 39th Euromicro conference on software engineering and advanced applications*. IEEE, 9–16.

[2] Atlassian. 2015. Git Feature Branch Workflow. Retrieved May 5, 2023 from https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow

[3] Sandro Formica and Fabiola Sfodera. 2022. The Great Resignation and Quiet Quitting paradigm shifts: An overview of current situation and future research directions. *Journal of Hospitality Marketing & Management* 31, 8 (2022), 899–907.

[4] Martin Fowler, Jim Highsmith, et al. 2001. The agile manifesto. *Software development* 9, 8 (2001), 28–35.

[5] Carlene Lebeuf, Margaret-Anne Storey, and Alexey Zagalsky. 2017. Software bots. *IEEE Software* 35, 1 (2017), 18–23.

[6] X. Liu, J. Heo, and L. Sha. 2005. Modeling 3-tiered Web applications. In *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. 307–310. https://doi.org/10.1109/MASCOTS.2005.40

[7] Soumi Majumder and Atreyee Mondal. 2021. Are chatbots really useful for human resource management? *International Journal of Speech Technology* (2021), 1–9.

[8] James E McDonough and James E McDonough. 2017. Observer Design Pattern. *Object-Oriented Design with ABAP: A Practical Approach* (2017), 155–171.

[9] James E McDonough and James E McDonough. 2017. Singleton Design Pattern. *Object-Oriented Design with ABAP: A Practical Approach* (2017), 137–145.

[10] Benedikt Morschheuser, Juho Hamari, and Jonna Koivisto. 2016. Gamification in crowdsourcing: a review. In *2016 49th Hawaii international conference on system sciences (HICSS)*. IEEE, 4375–4384.

[11] Ngrok. 2012. Ngrok. Retrieved May 5, 2023 from https://ngrok.com/

[12] open source. 2010. PyMySQL. Retrieved May 5, 2023 from https://pymysql.readthedocs.io/en/latest/

[13] Oracle. 2010. mysql-connector-python. Retrieved May 5, 2023 from https://dev.mysql.com/doc/connector-python/en/

[14] Pozitive. 2023. Bravo. Retrieved February 17, 2023 from https://www.motivosity.com

[15] Render. 2017. render.com. Retrieved May 5, 2023 from https://render.com

[16] Slack. 2018. Bolt. Retrieved May 5, 2023 from https://api.slack.com/tools/bolt

[17] Inc Smartly. 2023. Bonus.ly. Retrieved February 17, 2023 from https://bonus.ly/home

[18] Emanuel Stoeckli, Falk Uebernickel, and Walter Brenner. 2018. Exploring affordances of slack integrations and their actualization within enterprises-towards an understanding of how chatbots create value. (2018).

[19] Donald Sull, Charles Sull, and Ben Zweig. 2022. Toxic culture is driving the great resignation. *MIT Sloan Management Review* 63, 2 (2022), 1–9.

Received 5 May 2023