TagTeam Dashboard

Authors:
Ali Manzoor, Jacob Morroni, Rishi Singh, Nebiyu Abreha

Process Deliverable II:

We chose to implement an agile SE process for developing TagTeam Dashboard. For the process deliverable, we are submitting a retrospective analysis and list of prioritized tasks for the next sprint planning milestone.

Retrospective Summary:

Achievements:

- Expanded Functionality: The tagging feature was successfully prototyped, allowing developers to mark problematic sections of code and notify their teams effectively.
- GitHub Integration Progress: Initial implementation of GitHub integration is completed, allowing developers to sync changes in code in real-time.
- Team Collaboration Improvements: Mockups and wireframes for the dashboard's UI were created and refined based on feedback, addressing usability concerns.

Challenges:

- Tagging Usability: Ensuring an intuitive process of tagging without sacrificing simplicity was indeed a design challenge.
- Integration Hurdles: Synchronizing GitHub updates with dashboard notifications required resolving API limitations.

Prioritized Tasks for the Next Sprint Planning Milestone:

1. Finalize Tagging Feature:
    - Link the tagging process seamlessly with GitHub to improve collaboration.
    - Priority: High
2. Enhance GitHub Integration:
    - Enable notification features for every commit associated with tagged code.
    - Priority: High
3. Develop Reward System:
    - Ensure rewards are distributed in a fair way using predetermined metrics.

    - Priority: Medium
4. Conduct Usability Testing:
    - Test wireframes and implemented features with target users for feedback.
    - Iterate designs and functionality to improve onboarding and accessibility.
    - Priority: Medium

High-Level Design

Our team will use the MVC (Model-View-Controller) architectural pattern to structure our system. The reason for this is because this pattern separates our concept into 3 main sections whose behavior can be seen here:
Model: Manages task data, user roles, and statuses.
View: Provides interactive UI for the dashboard.
Controller: Handles interactions, task updates, and team member assignments.
This choice facilitates scalability and maintainability, key for evolving collaborative tools. It allows for pinpointing of errors or bugs, and not needed to refactor the entire product if major issues arise. We believe this architecture would be the strongest option for our design.

Low-Level Design:

For users of the TagTeam dashboard, a design pattern that can be used is the Singleton pattern. This is a creation pattern that allows only a single instance to exist. This is good for users because only one instance of each user needs to be created.

Another design pattern that can be used in conjunction with the Singleton pattern is the Proxy pattern, which allows objects to represent other objects. This can be used when creating users with different levels of permissions, such as administrators and users. An administrator object can represent a user object but with additional permissions.

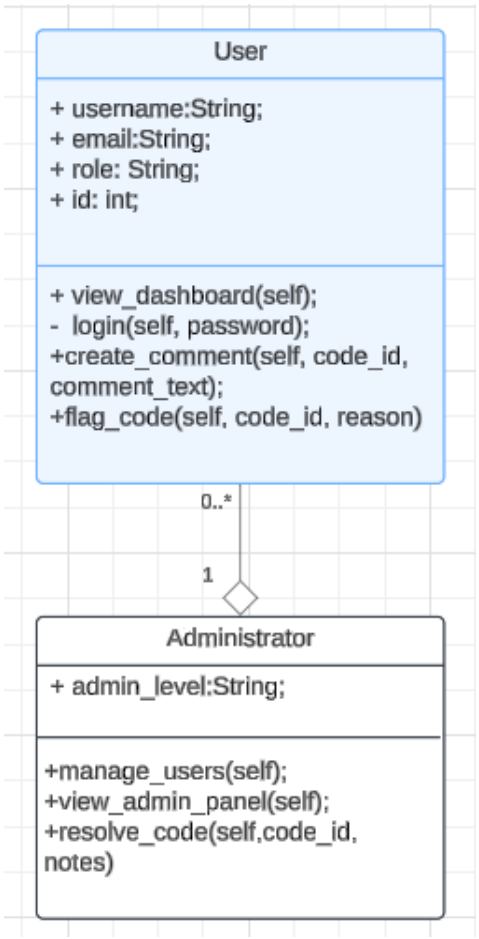
Finally, the last design pattern that can be used for users is the Observer design pattern. This would allow the developers and users to track how many people have access to the dashboard, allowing them to know who is contributing and how effective it really is.

Pseudocode:

```
class User:
        def __init__(self, username, email, role, id):
            self.username = username
            self.email = email
```

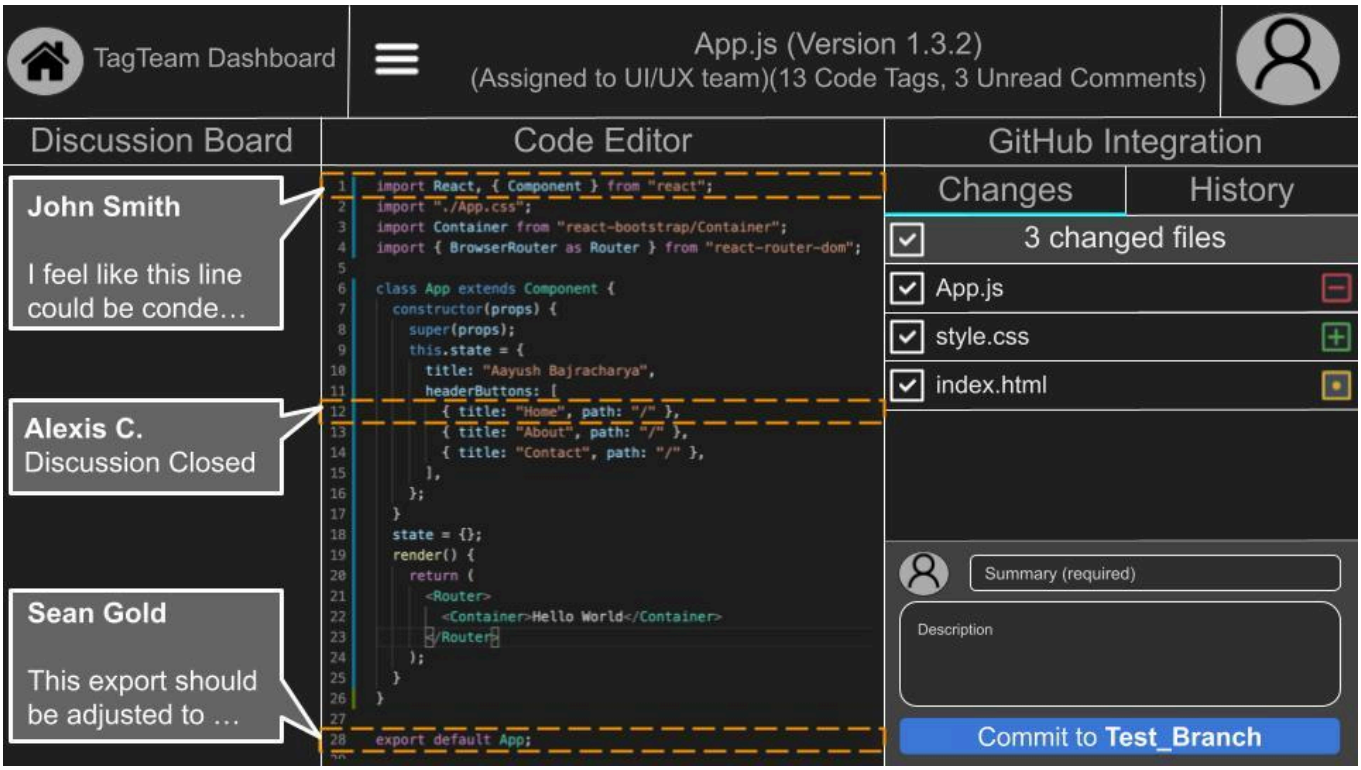
```
        self.role = role
        self.id = id

    def login(self, password):
        pass

    def view_dashboard(self):
        print(f"{self.username} is viewing the dashboard.")

    def create_comment(self, code_id, comment_text):
        print(f"{self.username} commented on code {code_id}: {comment_text}")

    def flag_code(self, code_id, reason):
        print(f"{self.username} flagged code {code_id} for {reason}.")

class Administrator(User):
    def __init__(self, username, email, admin_level):
        super().__init__(username, email, role, id)
        self.admin_level = admin_level

    def manage_users(self):
        print(f"Administrator {self.username} is managing users.")

    def view_admin_panel(self):
        print(f"Administrator {self.username} is accessing the admin panel.")

    def resolve_code(self, code_id, resolution_notes):
        print(f"Administrator {self.username} resolved code {code_id} with notes:
{resolution_notes}")
```

UML Diagram:


User
+ username:String;
+ email:String;
+ role: String;
+ id: int;
+ view_dashboard(self);
- login(self, password);
+create_comment(self, code_id, comment_text);
+flag_code(self, code_id, reason)
0..*
1
Administrator
+ admin_level:String;
+manage_users(self);
+view_admin_panel(self);
+resolve_code(self,code_id, notes)

Design Sketch

Wire Frame:



Design Rationale: Our wireframe mockup was designed based on inspiration from the GitHub Desktop app for the integration, any generic code editor, and the Google suite for comments in the discussion board. This means that the dashboard inherently uses consistency and standards by using industry leading styles for the design. Additionally, the header is formatted into a home button, navigation, and user access, allowing for easy user control and freedom, along with simple recognition by the user. Help and documentation can be found in the menu, and shortcuts and minimalism can be adjusted in settings. Lastly, system status and error messages will be displayed when needed. Therefore, our design hits on the majority of the UI/UX heuristics, which creates a strong product for users.