TagTeam Dashboard

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

Process Deliverable III:

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:

- Tagging Feature Completion: The tagging feature is now fully functional and integrated with GitHub, allowing developers to tag, comment, and collaborate on specific code sections.
- GitHub Integration Improvement: Task status is in real-time with GitHub commits thus developers will see seamless updates.
- Reward System: Reward system is up and running, tracking contributions and scores for the developers based on their contributions with tagged issues.
- Positive Feedback about UI/UX: The usability testing has provided overwhelmingly positive feedback, with new users completing their first tagging task within the target time of 5 minutes.

Challenges:

- User Adoption: Consistent use of the reward system and collaboration tools from all team members is still a challenge.

Prioritized Tasks for the Next Sprint Planning Milestone:

1. Optimize System Performance:
    - Conduct additional load testing to validate improvements.
    - Priority: High
2. Finalize Security Features:
    - Test security measures for vulnerabilities and compliance with industry standards.
    - Priority: High
3. Refine Reward System:

- Improve the reward tracking algorithm to better reflect individual contributions and collaboration efforts.
- Implement a feature to redeem points for tangible rewards or recognition badges.
- Priority: Medium

4. User Training and Engagement:
    - Create a training manual and other training materials to educate proper use amongst developers.
    - Priority: Medium

Black Box Test Plan:

Test ID	Description	Expected Results	Actual Results
Test 1: GitHub Display Test (Jacob Morroni) Test Type: UI/User Behavior	Preconditions: User has an existing GitHub and dashboard account, the repository is valid, and the User has access to the branch. Steps: 1. Open a file from the repository in the code editor 2. Make a simple edit in the Code Editor 3. Check the GitHub tab for a new "changed file" and ability to commit	On edits, GitHub tab will update showing changes to files on the UI and will allow commits to a chosen branch with comments and a summary.	Intentionally Blank
Test 2: GitHub Integration Test (Jacob Morroni) Test Type: Integration/ Communication	Preconditions: User has an existing GitHub and dashboard account, the repository is valid, and the User has access/edit rights to the branch. Steps: 1. Open a file from the repository in the code editor 2. Make a simple edit to the file 3. Write a summary and commit the to current branch 4. Open GitHub and check if the file was properly updated and the commit is present in logs.	On commits, the GitHub repository will update to mirror changes made to the files through the dashboard.	Intentionally Blank
Test 3: Comment Email Notification Test (Jacob Morroni) Test Type: Informational	Preconditions: User has a dashboard account, access to the repository, and commenting rights. Additionally a manager and separate user have at least the same permissions and access and have email notifications enabled. Steps: 1. Open a file from the repository in the code editor	On comments, the PM and users subscribed to the repository should get a notification alerting them of the user who commented,	Intentionally Blank

	2. In the Code Editor, select a line of code in the current file 3. In the pop-up menu, select “Add a comment to the Discussion Board”, and then send it to the discussion. 4. Determine if the manager and separate user both get an email notifying them of the newly created discussion.	highlighted code, and comment.	
Test 4: Comment Response/ Managing Test (Jacob Morroni) Test Type: Visibility/User Behavior	Preconditions: User has a dashboard account, access to the repository, and commenting rights. Additionally a manager and separate user have at least the same permissions and access. Steps: 1. Open a file from the repository in the code editor 2. In the Code Editor, select a line of code in the current file 3. In the pop-up menu, select “Add a comment to the Discussion Board”, and then send it to the discussion. 4. On the separate user’s dashboard, determine if they can see the newly created comment, who posted it, and have the option to respond to the post. 5. On the manager’s dashboard, determine if they can respond but also remove or flag the post, along with making it a “prioritized” post.	On comments, the PM and users should be able to respond to comments and see highlighted code, and the PM should have the mentioned additional actions for team management purposes.	Intentionally Blank
Test 5: High Volume Response Time Test (Jacob Morroni) Test Type: Performance/ Overload	Preconditions: Have 500 sample users editing 20 different repositories for 25 person teams. Out of the 25, 10 are making code edits, 10 are making commits, and 5 are commenting. Steps: 1. Record the system response times of each of these user’s actions and average them. 2. Compare to system response times when a single user is performing each of the behaviors described above. 3. Determine if the high volume response times are within a 1% of the low volume response times.	Up to at least 500 users using the system at a single time, the system does not exhibit a response time degradation of greater than 1% for the average user.	Intentionally Blank

Test 6: Flagging Code Block for Review (Rishi Singh) Test Type: Functional/UI	Preconditions: User has an existing GitHub and dashboard account, the repository is valid, and the User has access/edit rights to the branch. User has access to their current code in need of review Steps: 1. Have someone with permission flag a code block for review 2. Have another developer see whether they can access the block of code when viewing the dashboard	The code should be successfully flagged. The block of code should appear on the other developer's dashboard, ready to be reviewed.	Intentionally Blank
Test 7: Marking flagged code block as "Resolved" (Rishi Singh) Test Type: Informational	Preconditions: User has an existing GitHub and dashboard account, the repository is valid, and the User has access/edit rights to the branch. User has access to their current code in need of review. Administrator has access to code block and menu to mark code as "Resolved" Steps: 1. Developer will flag code for review 2. Another developer will fix the code and push to the Github repository 3. Administrator will review changes on the repository and flag as "Resolved" if the code is functional	The code should no longer be flagged, and a log of all comments and changes should be saved onto a database.	Intentionally Blank
Test 8: Move flag to top of priority when left unfinished/ urgent (Rishi Singh) Test Type: Visibility	Preconditions: User has an existing GitHub and dashboard account, the repository is valid, and the User has access/edit rights to the branch. User has access to their current code in need of review. Steps: 1. Developer will flag code for review and let code sit for a while without any fixes 2. Once a lengthy time period passes, the developer/administrator reflags the code, moving the block to the top of the priority list 3. Other developers see that the code has been unfinished for a long time, with the time of the code being highlighted	The code that was initially in the bottom of the queue due to no support gets moved to the top. Other developers will now see the flagged code, as it is prioritized over newer tasks to prevent falling behind.	Intentionally Blank
Test 9: Applying	Preconditions: User has an existing GitHub and dashboard account, the repository is valid, and	Other developers should be able	Intentionally Blank

different types of visual effects to blocks of code (Rishi Singh) Test Type: UI/Visual	the User has access/edit rights to the branch. User has access to their current code in need of review. Steps: 1. The developer flags a block of code 2. When flagging the code, the developer highlights the functional parts of code in green, the dysfunctional parts in red, and bolds important variables	to see the visual effects applied by the developer to the code to allow them to better understand and interpret the changes needed.	
Test 10: Test role/ permission functionality (Rishi Singh) Test Type: Security	Preconditions: One developer has a role applied to their profile which is for project 1. The other developer has a role for project 2. An administrator can view both. The developer in project 1 should only be able to view their project and vice versa unless given temporary authorization from the administrator Steps: 1. Developer 1 attempts to access Project 2 2. Developer 2 attempts to access Project 1 3. Administrator applies temporary authorization to both developers for the other projects	Developer 1 should be able to view and review the code from Project 2 for however much time the administrator specified and vice versa.	Intentionally Blank