CS3704 - PM2
Team Octo

Process Deliverable I

To summarize our retrospective tasks and priorities, we started with determining the goal of our application and requirements elicitation for this application. Regarding our goal of the application, we want to better organize task assignment and prioritization for larger design teams. With an application like this, we wanted to hear from our target audience about what features they would want in an application like ours; as well as hear more about what problems people have with current design practices and Kanban tools.

To elicit these requirements, we created a survey for people in our target audience, software developers, to fill out to get a better understanding of just what they would want in an application like ours. The questions ranged from focusing on the current knowledge of Kanban to how people felt about their group projects and their level of organization and effectiveness. With this survey, we had 11 responses and the overall message was that Kanban was relatively underutilized and that with an application like ours, they would want to see features such as checklists per card, the ability to assign cards to certain team members, linking related cards together, and being able to archive completed cards.

Following this, we had a heavy focus on analyzing our requirements in order to get a better understanding of both functional and non-functional requirements. We also wanted to explore our requirements via specification using use cases and user stories. The results of these analyses are listed below.

To touch on future priorities and goals, we will begin creating implementations and mockups of our design. With these mockups, we will do user testing where we get someone who hasn't used our application before to use it and tell us what they liked, what they didn't like, and what could have been improved. We'll analyze this feedback to redesign our mockup, where we'll then repeat the process until we feel that we've reached a satisfactory iteration of the Kanban tool application.

Requirements Analysis

- 5 nonfunctional requirements:
    1. Board loading time must take, at most, 2 seconds. The system must respond to local changes instantaneously, with subsequent server handshaking taking at most 0.25 seconds per change (performance).

2. The system must encrypt all board data and keep at least one backup per board (reliability).
3. The app must be accessible to users on both a website and a mobile application, with the former being accessible on both desktop and mobile (supportability).
4. Must be implemented using a relational database, such as SQL (implementation/constraints).
5. Must provide ample and easy-to-understand documentation on each feature (usability).

- 5 functional requirements:
    1. Must be able to load Kanban board(s) and display their saved progress.
    2. Must be able to take client-side user changes, apply them to the server model, and display them to other users.
    3. Must have the ability to assign attributes to tasks, such as type, priority, and designation.
    4. Must be able to change view mode between standard Kanban, list, and timetable.
    5. Must be able to handle instant messaging between members of a board.
- 5 formal use cases:
    1. Board creation
        - Primary actor: development team leader
        - Goal: creating a Kanban board for a team
        - Pre-conditions: user has an account
        - Post-condition: user owns a Kanban board and has the power to invite other users
        - Basic flow:
            - User accesses the application and enters their credentials
            - User clicks the *create board* button from their dashboard
            - User assigns board properties through a pop-up, such as name and description
            - System processes the request and creates a board
            - System directs the user to the board and returns a link for sharing the board
    2. Board switching
        - Primary actor: developer with access to two boards
        - Goal: switching between Kanban boards without losing progress
        - Pre-conditions: user is logged in, has been granted access to two boards, and is currently viewing a board
        - Post-conditions: local user changes are stored to affected boards

    - Basic flow:
        - User makes local changes on a given Kanban board.
        - User clicks a button in the top left corner to return to the dashboard.
        - The system makes sure to apply local changes before allowing the user to exit the current webpage.
        - Once completed, the user is directed to their dashboard.
        - User clicks on a different board and is directed to it.
3. Creating cards
    - Primary actor: developer with access to a board
    - Goal: creating new cards and assigning task data to them
    - Pre-conditions: user is logged in, has access to a board, and is currently viewing said board. Board contains at least one list.
    - Post-conditions: new cards are created and registered in the board both locally and in the server
    - Basic flow:
        - User select a given list within the board they're viewing
        - User clicks the *add card* button on the given list, creating a new card at the bottom of the stack
        - User clicks the card to expand it, opening a pop-up menu
        - User enters card title, description, type, and tags
        - Card is saved to the cloud once the user's client hits an autosave
4. Changing task completion level
    - Primary actor: developer with access to a board
    - Goal: changing the completion level of a card within a board
    - Pre-conditions: user is logged in, has access to a board, and is viewing said board. Board contains at least two lists and one card.
    - Post-conditions: lists are updated with correct card information both on the client and server
    - Basic flow:
        - User places their pointer (cursor on desktop, touch on mobile) over a given card and presses down on it
        - User drags card to an adjacent list and releases it
        - Card is moved from the first list's stack to the second list's stack
        - Changes are saved to the cloud once the user's client hits an autosave
5. Creating and modifying checklists

        - Primary actor: developer with access to a board
        - Goal: creating a checklist for a card and assigning initial values
        - Pre-conditions: user is logged in, has access to a board, and is viewing said board. Board contains at least one card.
        - Post-conditions: card contains a checklist with assigned entries, each entry having a completion status.
        - Basic flow:
            - User clicks a card to expand it, opening a pop-up menu.
            - User clicks the *add checklist*, prompting an additional pop-up menu.
            - User enters the title and priority of the checklist, then clicks *create*.
            - The system responds by appending a checklist to the card's data and reflecting it visually in the UI.
            - The user clicks the *Add item* button under the checklist at least twice, each click creating a new entry and prompting the user to enter text.
            - The clicks the checkbox next to a given entry, marking it as complete.

Requirements Specification

1. As a team manager, I want to be able to restrict access to certain functionality within a Kanban board. This is so that I can prevent certain cards and lists from being modified, as they are meant to be finalized and archived.
    - Acceptance criteria:
        - *Given* team manager owns the board or has admin permissions
        - *When* team member selects a given card or list within their board
        - *And* clicks the lock button and confirms the action through a pop-up
        - *Then* the system marks the given card/list as locked and prevents other non-admin users from modifying it
    - Function points: this action would take a low amount of effort, something like 2 function points. The user only has to find their desired item and click two buttons (the lock button and the confirm button), which is something that shouldn't take more than 30 seconds.
2. As a team manager, I want to be able to see the activity log of my employees within my Kanban board. This is so that I can see who's working on what, and at what rate.
    - Acceptance criteria:
        - *Given* team manager owns the board or has admin permissions

        - *When* team members is viewing the board
        - *And* clicks the *view activity log* button
        - *Then* the system shows a scrollable pop-up menu that shows a timestamped entry for each user's action on the board, and the team manager would peruse it.
    - Function points: this menu would be easy to open but require some effort in parsing, as it would contain a lot of dense information. For this reason, this action would require around 5 function points.
3. As a developer, I want to be able to assign myself to any number of tasks. This is so that I can be efficient at informing the rest of the team on what I'm working on.
    - Acceptance criteria:
        - *Given* developer has access to the board
        - *When* cards exist within the board and the developer is viewing the board
        - *And* clicks on a particular card, opening a pop-up menu, then clicks on the *join* button
        - *Then* the system vinculates the user's account with the card, notifying board managers and whoever is also assigned to the card
    - Function points: this should be a very streamlined action since it's so common, so we made sure to make it simple. The action requires only clicking on a card and checking a box, it's as simple as an action can get, making it cost only 1 function point.
4. As a developer, I want to be able to message any given team member about the state of the board we share. This is so that we can answer any questions and solve conflicts.
    - Acceptance criteria
        - *Given* the developer and at least one other user has access to the board
        - *When* the user has a concern they need to address with their team
        - *And* clicks the *member list* on the board, scrolls to find their desired person, clicks on their profile, clicks the *send a message* button.
        - *Then* a small message box should appear at the bottom of the screen, allowing instant communication between the two (with desktop/mobile notifications).
    - Function points: this action requires quite a bit of conscious thought, as the user has to think of a question, find the person they're interested in, and actively engage in the conversation. For this reason, this action requires around 10 function points.