Formal Requirements + AGILE

Project description:

CodeChuckle is introducing a new diff tool: SnickerSync—why merge in silence when you can sync with a snicker? The PMs have a solid understanding of what it means to "sync with a snicker" and now they want to run some user studies. Your team has already created a vanilla interface capable of syncing with the base GiggleGit packages.

Complete the following tasks:

- List one goal and one non-goal
- Create two non-functional requirements. Here are suggestions of things to think about:
    - Who has access to what
    - PMs need to be able to maintain the different snickering concepts
    - A user study needs to have random assignments of users between control groups and variants
- For each non-functional requirement, create two functional requirements (for a grand total of four functional requirements).

Project requirements

- Goal: Create an easy-to-use UI/UX that is familiar to users in terms of workflow and relevant memes.
- Non-Goal: Implement a real-time peer-to-peer editing interface allowing real-time updates to repositories when synced. (Similar to Google Docs)
- Non-functional requirement 1: Security + Authentication
    - Functional requirements:
        - Use OAuth authentication for user Github Accounts
        - Securely store any keys used in projects locally without leaking them. For example, if a user has a GoogleCloud key, we should warn them that their key may be public if a) the repo is public, or b) the project was deployed.
- Non-functional requirement 2: Accessibility
    - Functional requirements:
        - Implement keyboard navigation support so users don't need a mouse to sync.
        - Provide screen reader compatibility making features interpretable. This should include descriptions for images.

- Non-functional requirement 2: Sync-Consistency
  - Functional requirements:
    - The version control should log and display all Snicker syncs, users can review changes and Snickers applied during each sync.
    - Snicker updates to repositories should be the only thing changed to reflect the right updates and versions. This checks for consistency.

Agile

- Theme: Get the GiggleGit demo into a stable enough alpha to start onboarding some adventurous clients
- Epic: Onboarding experience

- User story 1: As a vanilla git power user who has never seen GiggleGit before, I want to import my existing repositories from GitHub with custom “meme-ified” workflows to see how it differs from GitHub.
  - Task:
    - Ticket 1: Store Keys for User Security
      - We need to ensure API keys aren't leaking on the web. Research and implement the current best practices appropriate for our stack.
    - Ticket 2: Implement OAuth
      - Implement OAuth for GitHub to securely import repositories. By authorizing GiggleGit on Github we could pull their account. We should make sure GiggleGit only requests necessary information to maintain a trusting relationship with clients and avoid any legal trouble.

- User story 2: As a team lead onboarding an experienced GiggleGit user, I want to streamline setup for a development team where we can collaborate on GiggleGit and add features and reactions.
  - Task:
    - Ticket 1: Team Permissions
      - Build a system that can manage permissions such that team leads can quickly modify the experienced user's roles/access to shared repos. Leads should also be able to view pull requests and react

with a supportive meme if it should become a feature.

        - Ticket 2: Create Shared Repos
            - Make teams able to set up permissions and roles when setting up a new project. Roles can be denoted by silly memes. For example, a senior developer could be DOGE. We would also need to make the system sync between user accounts.

- User story 3: As a CS professor I want to set up class repositories for collaboration among several different classes. I also want to be able to use GiggleGit for personal projects and don't want my students to see solutions or my projects.
    - Task:
        - Ticket 1: Implement Class/Organization repo creation
            - Build a feature that allows professors to create class repositories, giving the Prof., TAs, and CAs permission to add content. It should also allow students to join the class seamlessly and only require OAuth from GitHub (as they are likely vanilla users)

        - Ticket 2: Add personal project privacy
            - This should include visibility controls such that students cannot see future assignments or other students' work. From the previous story, this could be done using roles where TAs/CAs can see solutions/future work, but students cannot.