Nihal Rajan (Project Leader)

Dion Peng

Qusai Ghabrah

Shashank Kumar

TAs: Jakob Hain, Rylan Kasitz

CS 30700: Design Document - Team 6

Project Name: *Ollie*

1 Purpose

Without an effective management system, it can be difficult to track all the aspects of project stories such as assignee, priority, status, etc. Additionally, discussions about the stories may occur through email, Microsoft Teams, and other platforms which would make searching for conversations about a specific topic from weeks or even days ago difficult. One way to solve this is by having a designated section for comments regarding each ticket as JIRA does. Our project will include a lot of the core functionality of JIRA, but we plan to add other features such as different user roles, email notifications about tickets not recently updated, and scripting where when somebody pushes a commit, a ticket is automatically created for the code reviewer.

2 Design Outline

1. We are going to use the client-server architecture model. The three main components would be the client, the server, and the database. The client component’s purpose would be to provide the UI of the web app that would enable users to use CRUD functions for their stories. The server component’s purpose would be to host the web app and handle the back-end logic (e.g. communication with the database). The database component will store and verify credentials along with storing tickets/ticket information, and store the data created on the client side in order to view or manipulate them later on.
2. In the client-server model, there are four components. They are the client, the services the client interacts with, the database for storing project and user data, and the server component which responds to the services that the client has invoked.

The interactions between the client and the client services are the login and editing of the task board in order to create tickets and modify them based on attributes like priority of the task. The client interacts with the task board by using the services of the website and changes different attributes in the system based on their needs.

The services that the client invokes on the task board are stored using the server. Attributes of the task are updated by the client invoking the services of the website and these attributes are stored using the server so they can be reused. Along with this, username and password storage is also one key component that is required when talking about database storage. The user is stored along with a hashed password on the database side and checked when trying to log into the page, which is another interaction between client invoked components, the server, and the database. Along with this, the database will store ticket information and tickets.

(c)



3 Design Issues

**Functional Issues:**

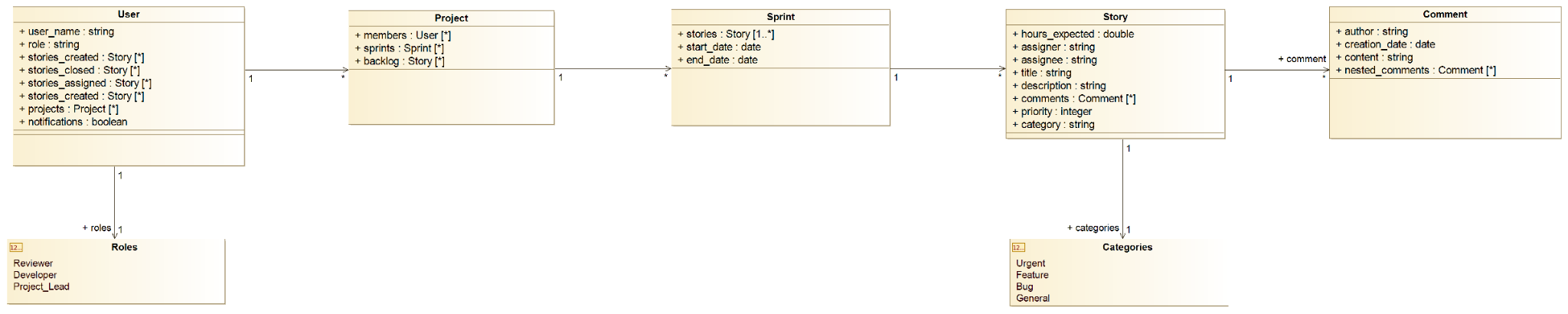
|  |  |  |
| --- | --- | --- |
| Descriptive Title | Solution options for the issue | Justification of choice |
| How should we let users create their accounts? | 1. Manual sign-up process (Fill name, email, password, etc…) 2. Allow users to sign-up using their Google account 3. **Allow users to sign-up using their Github account** | Using Github will make it easier for us when we work on the scripting part of our project. |
| How should we let users create tickets with different priorities? | 1. **Time estimate** 2. Story points | Story points are a subjective thing and an unnecessary abstraction for time/effort estimates. |
| What kind of modifications to tickets should be easily accessible? | 1. **Delete** 2. **Hours Estimate** 3. **Assigner** 4. **Assignee** 5. **Moving between statuses** 6. **Moving to/from different sprints and backlog** | These are things every ticket should have and we should make it as smooth as possible for users to fill in. |
| How should we let users search for different tickets? | 1. **Search within sprint** 2. **Search for just title** 3. **Search by ticket hash** | If possible, we would like to include all of the options for search functionality. This is because as the ticket number increases, different searches can lead to the right ticket faster. Along with this, different information known about the ticket with different searches can help find the right one. |
| How would we implement scripting? | 1. **REST Endpoints** 2. Shell script executable 3. JavaScript integration 4. GitHub Hooks | Scripting requires some form of git hooks to integrate version control with Ollie. REST is a clean approach since it allows for flexibility in the calls and since we’re already using the web. |
| UI component flow is succinct and displays required information without overcrowding  (user-friendliness) | 1. Include a lot of details about ticket but adjust font so that UI is not crowded 2. **Only include information that is required on each ticket such as number, priority, and ticket information** | User-friendly design speeds up the usage of the application and makes the overall workflow much more efficient. |
| How should we notify people if there’s an overdue story? | 1. **Email notification** 2. Browser notifications | Email notifications would be easier to implement and also more convenient. Browser notifications won’t be visible to users unless the browser is open. |
| How do we integrate with other teams? | 1. **Share boards through a link** 2. Give access to boards through invitations | Easy access for everyone to allow for cross-team collaboration |
| How do users organize their various sprints? | 1. **Most up-to-date sprint as the default view** 2. Ability to rearrange boards | Since most of the time users will be using the current sprint, having that as the default view makes the most sense. |
| How do users organize/keep track of stories? | 1. **Email notifications if they are the assignor/assignee** 2. **Ability to ‘watch’ a story to get notified whenever a change is made** 3. **Favorite stories to have quick access to a couple of them** | We want all of this functionality because it is important that people assigned to stories are aware of what changes are made to which stories and they are also aware of what stories they are a part of and the status for those. |
| How do we ensure users don’t accidentally delete their stories? | 1. Popup warning when deleting a story 2. **Temporary recycle bin for 30 days that they can restore from** | The recycle bin is useful because it helps keep track of what is deleted before it is permanently removed. Keeping track of what is removed also lets developers know what kind of stories are not feasible/are not required. |
| How do we provide an alternate option for creating stories for users that do not want to use the UI? | 1. **Pass parameters through URL to create new ticket (REST Endpoint)** 2. Allow users to fill out form in an email to create ticket | Passing parameters through the URL is a convenient way to create a new ticket. An email with a form could be sent out when the user first creates an account, but they would have to search for this email if it has been a long time since then. Sending emails periodically with the link to the form could crowd the inbox. |
| How should users view this from their mobile device? | 1. Mobile app 2. **Same Website** | Using the website will save time for developers, rather than having to worry about Android, iOS, etc and re-writing everything as a mobile app. |
| What colors should be associated with each priority? | 1. **High priority - red** 2. **Low priority - green** | Easier to know what tasks are important at a glance |
| Should we have dark mode? | 1. **Yes** 2. No | It’s a nice-to-have for certain users who prefer it, but will require some extra software engineer work. |
| How should users get help using the service? | 1. **Help tooltips when hovering over icons** 2. **Initial tutorial when a new user signs up** | Having users an initial tutorial gives them the option to learn more about the features of our app. Help tooltips give short tips without needing to go to the full tutorial. |
| How should we update the board when a change is made? | 1. **Hot reloading -- While looking at the board and user 1 makes a change, user 2 does need to refresh the page** 2. Require refresh | Better user experience, especially with many people looking at the board at the same time, during standup for example. |

**Non-Functional Issues:**

|  |  |  |
| --- | --- | --- |
| Descriptive Title | Solution options for the issue | Justification of choice |
| What database would be the most appropriate for our data? | 1. **MySQL - Relational** 2. MongoDB - NoSQL | We chose a SQL database in this case for its uniformity along with its ease of usage - very easy to integrate into our application. |
| How are we hosting the service? | 1. **Web application** 2. Desktop application | We are choosing the web application because of the cross platform compatibility. Since we plan to use Github pages to host the web app, this allows for a more manageable application while delivering to the most users possible. |
| How are we storing user information? | 1. Username + Password stored as text 2. Username as text + hashed password 3. **Username as text + hashed password with salting** | We want to store the password as a hash with salting as it helps mitigate hash table attacks by creating unique passwords for each user even if the passwords match. Security increases as a result. |
| How often are we going to sync tickets to/from the server? | 1. Periodical synchronization (e.g. sync every 10 minutes) 2. **Synchronization after every change** | This will make our service more efficient and also ensures that everyone is up-to-date regarding the stories. |

4 Design Details

(a) Include class level design of the system (i.e. class diagrams) and be as detailed as you can.

**

(b)

The way these classes will interact will follow a general sequence:

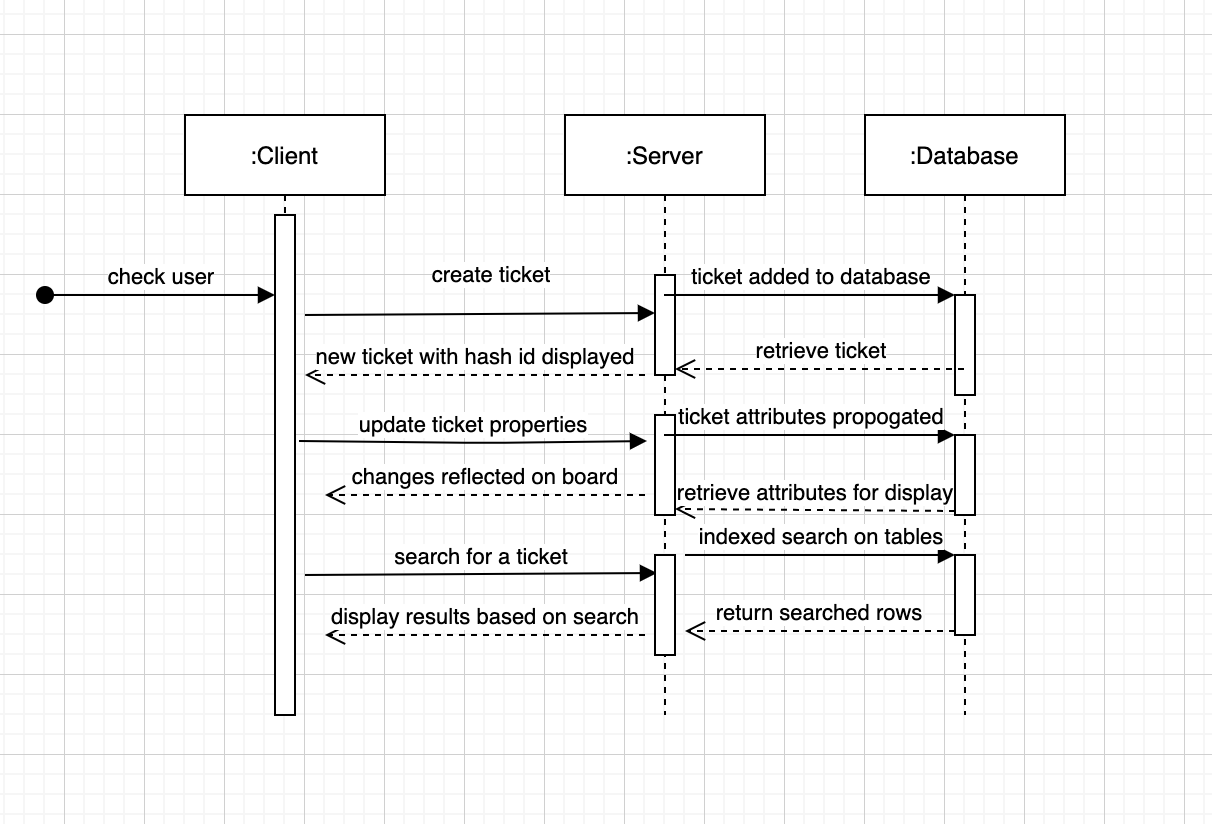
* After the user logs into the service and gets authenticated, we will get a User object from the database
* The user will choose which project to view and the service will get it from the Project array in the User object
* The user will choose which sprint to view which we will get from the Project object in the previous step
* The user will be shown a UI that will enable them to see current stories, their statuses, and all the other details provided in the Story objects.

Other ways that the classes will interact include:

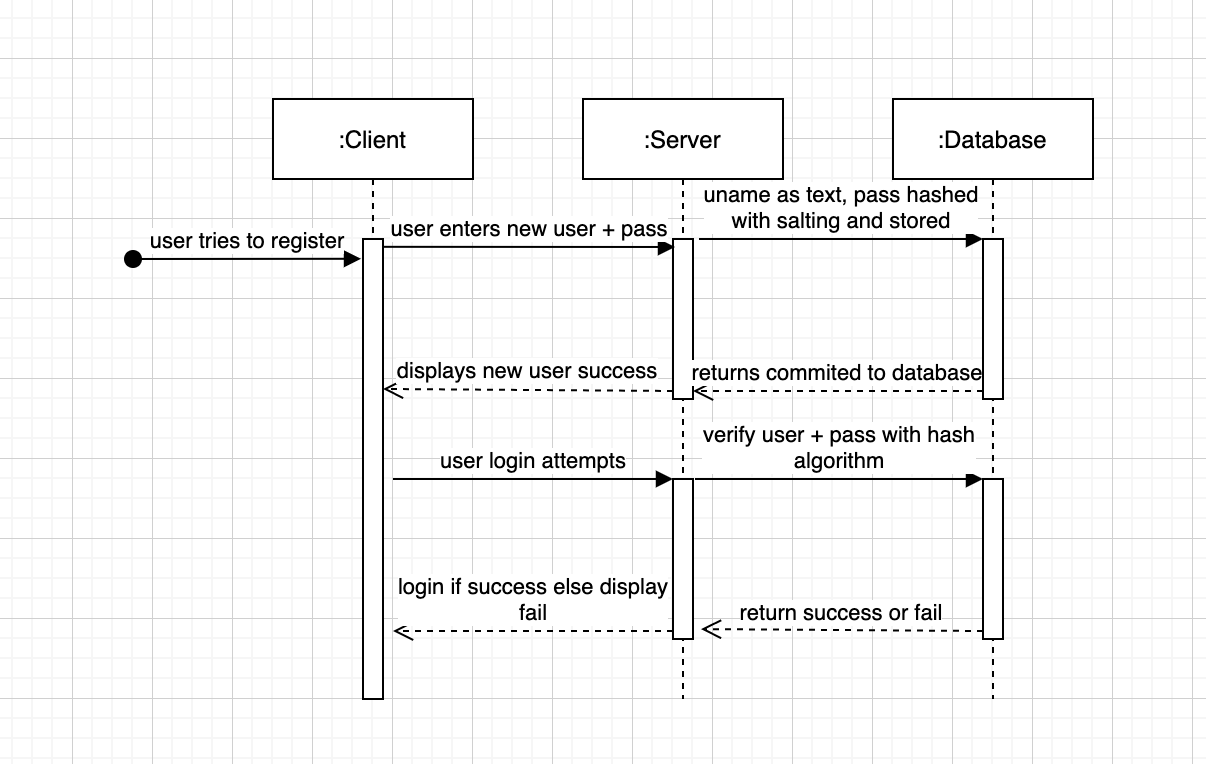
* Adding stories that are created/closed by a user in a Project object to their User object, and also updating the story array in the Project object

(c)

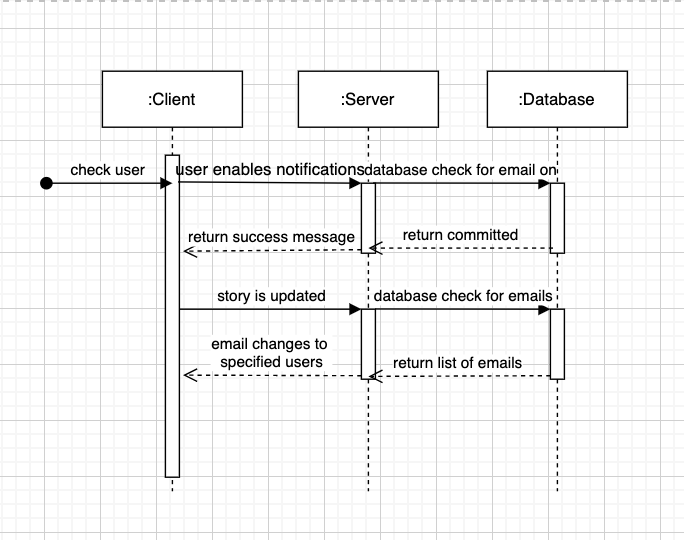
**Sequence Diagram 1: Ticket operations:**



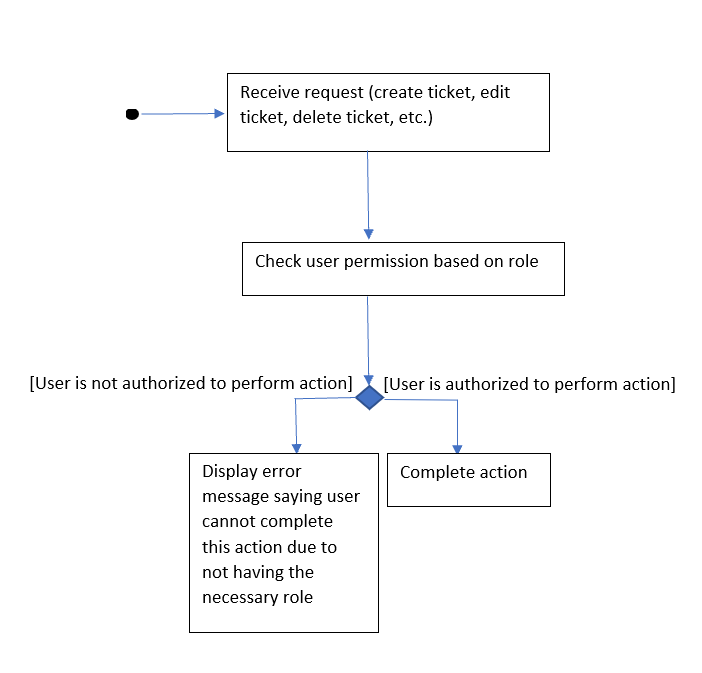
**Sequence Diagram 2: User Creation and Storage:**

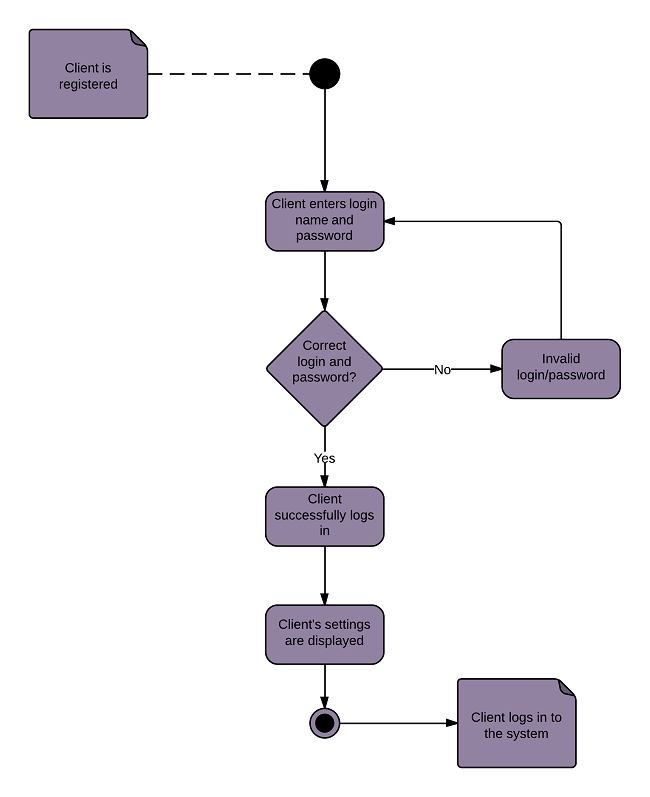


**Sequence Diagram 3: Email Notification System**

****

(d)





UI Mockups (What our product could potentially look like):

