Gleam

Project Backlog September 6th, 2019
Group 25
Derek Shu, Jie Li, Emily Ou, Shafer Hess

Problem Statement

Sharing news, ideas, opinions, information, etc. is a vital part of today's modern societal operations due to the connectivity between people all across the world. The capability to share information at such ease is something that should not be taken for granted and should in fact be actively cherished due to the fact that just a few decades ago, this concept would have been unimaginable.

Our goal is to expand upon the ideas of current social media sharing sites and craft our own refined platform that retains the idea of "sharing with a community", but fine tuned in a way to abstract out unnecessary bells and whistles to leave a simplistic application that at its core, focuses on sharing and discussion.

Background Information

Currently there are plenty of social media platforms that everyday users congregate on such as Twitter, Facebook, YouTube, etc., which all allow users to share opinions, ideas, and have an open conversation in theory. However, we believe that these mainstream platforms all suffer from similar issues of "herd mentality" which consequently leads to an extremely toxic one sided discussion where the minority/opposing side is always repressed.

Because the "herd mentality" phenomenon on social media is so prevalent on modern day platforms, we want to create a separate platform that strives for open discussion and prevents herd mentality from occurring by stripping away features such as liking/disliking which we believe naturally encourages such behavior.

Environment

Our application will be a full-stack web application, with the front-end and back-end living independently. Our front-end will be completely built and served with React (JavaScript), backed by Semantic UI for basic components to build the user interface of our application. Our back-end will be built with PostgreSQL and

Django (Python) and will simply act as an API to retrieve necessary information when the front-end requires it. For instance, when our front-end requires information about a user's post on our platform, it will fetch the required data from our API and display it accordingly for the user to see.

Both our VM and database will be hosted in AWS, that is, an Ubuntu EC2 instance in conjunction with a PostgreSQL database.

Our testing tools will include using Jest/Enzyme to test our front-end user interface and Django's built native testing tools to test and validate our API endpoints in the back-end. Additionally, if necessary, we will be using Postman for any manual testing of our API endpoints.

Functional Requirements

Backlog ID	Functional Req.	Hours	Status
1	As a user, I would like to create an account	4	Planned Sprint 1
2	As a user, I would like to login to my account	2	Planned Sprint 1
3	As a user, I would like to view "trending" post titles/post previews on the homepage	8	Planned Sprint 1
4	As a user, I would like the post previews to contain username of submitter and post title	4	Planned Sprint 1
5	As a user, I would like to be able to click an individual post on the page to see more details	4	Planned Sprint 1
6	As a user, when I click to view the details of a particular post, I would like to see the submission content (body text of the post)	4	Planned Sprint 1
7	As a user, when I click to view the details of a particular post, I would like to see other user's comments in the discussion section	8	Planned Sprint 1
8	As a user, I would like to be able to add comments to a post's discussion section	4	Planned Sprint 1

9	As a user, I would like to be able to view a list of my submitted posts	6	Planned Sprint 1
10	As a user, I would like to be able to search for certain posts by keywords in title	6	Planned Sprint 1
11	As a user, I would like to be able to filter discussion comments by “most recent”	6	Planned Sprint 2
12	As a user, I would like to be able to filter discussion comments by “most viewed”	8	Planned Sprint 2
13	As a user, I would like to be able to delete posts I have submitted	3	Planned Sprint 2
14	As a user, I would like to be able to delete comments I have posted	3	Planned Sprint 2
15	As a user, I would like to be able to edit post content for a post I have submitted	3	Planned Sprint 2
16	As a user, I would like to be able to edit my own comments in a discussion section	3	Planned Sprint 2
17	As a user, I would like to be able to bookmark/save posts	3	Planned Sprint 2
18	As a user, I would like to be able to view all my bookmarked/saved posts	6	Planned Sprint 2
19	As a user, I would like any list of posts to be paginated as to not clutter my webpage	8	Planned Sprint 1
20	As a user, I would like to be able to change my password	3	Planned Sprint 2
21	As a user, I would like to be able to view all my comments on all posts	5	Planned Sprint 2

Non-Functional Requirements

Architecture
Gleam will be developed to be a [PC/macOS] web application, meaning it should be operable on popular modern browsers such as Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge. Less popular browsers such as Internet Explorer will not be officially supported due to differences and varying incompatibilities with ES6 (especially if React polyfills cannot grant full backwards compatibility).

React will be the core part of the web application, directly serving the user by rendering the UI components on the page and allowing users to actually perform meaningful actions. Furthermore, React will handle all routing within the web application should a user want to navigate to a different page.

Our Django backend will be an API that allows our front-end to fetch/submit necessary data like a user's list of posts or submitting a comment and any other required actions/metadata to and from our PostgreSQL database.

Security
Security is an important consideration for all modern applications. As such, we will follow industry best practices when it comes to storing sensitive user information. In our case, this is simply just the user's password, as no other user data will be particularly sensitive. Our plan is to store the user's password in a secure manner where we salt and hash the plaintext form and store the final form in the database. Note that the salting, hashing, and storing will of course be done on the server-side. Furthermore, our PostgreSQL database will only be accessible from machines within our AWS Virtual Private Cloud (VPC).

Usability
The user interface should be easy to navigate such that the user experience is preserved. A standard user should easily be able to identify how to perform a desired action, and the visible possible actions (like buttons, dropdowns) should be minimal (not cluttered) to avoid confusion. Furthermore, we want the components in our interface to be "dynamic" so that browser windows of varying sizes and resolutions do not receive a less than optimal experience where components are out of place.

Performance
All components of this application will be entirely built and served from our Ubuntu VM, with the front-end (what the user interacts with) being done entirely in JavaScript (React). Our back-end API will be entirely done using Python (Django). Both the VM (x86 Ubuntu 18.04) and the PostgreSQL (10.6) database are running on t2.micro instances in AWS, meaning acceptable load will just be suitable for prototyping and demo application purposes. Both the VM and database are running with 1 virtual CPU and 1 GB of RAM with the database also limited to a maximum of 20 GB of storage capacity.

Due to React's component rendering algorithm which uses a virtual DOM to allow for efficient rendering (officially known as 'reconciliation'), we expect the users actions to feel responsive. Since JavaScript is asynchronous by nature to avoid a negative user experience, requests to our Django API from our front-end should be quick enough to not inhibit the user experience. Furthermore, while data is being fetched, loaders will be displayed appropriately.

Use Cases

Case 1:
Create a Gleam account

Action	System Response
1. Click "Sign Up" on navigation bar	2. Open modal with input fields for creating new account
3. Enter username, password	4. Validate inputs in real-time
5. Click submit	6. Send create account request to API
	7. If success, display "successfully created" message
	8. Store user information into DB appropriately (salt hash password)

Case 2:
Login to a Gleam account

Action	System Response
1. Click “Login” on navigation bar	2. Open modal with input fields for logging into an existing account
3. Enter username, password, submit	4. Send login credentials request to API
	5. If credentials match, set user session, close modal

Case 3:
Show trending posts on homepage

Action	System Response
1. Navigate to homepage	2. Display loader
	3. Send data request for “trending” posts to API
	4. Retrieves necessary posts and post metadata as JSON from API request
	5. Format and display posts and post metadata on page

Case 4:
Show post preview metadata

Action	System Response
1. Navigate to any page with posts (i.e. homepage, bookmarked, personal submissions)	2. Display loader
	3. Send data request for appropriate collection of posts to API

	4. Retrieves necessary posts and post metadata as JSON from API request
	5. Format and display posts and post metadata on appropriate page

Case 5:
Navigate (redirect) to full post from post preview on page by clicking a button

Action	System Response
1. Clicks on button with arrow on post preview	2. Redirect to empty page
	3. Display loader
	4. Send data request for full post data to API

Case 6:
Show body text/content of post when in “full post” view format (note: this is after redirecting from clicking the button on the post preview)

Action	System Response
1. Clicks on button with arrow on post preview	2. Redirect to empty page
	3. Display loader
	4. Send data request for full post data to API
	5. Display loader while fetching necessary data from API
	6. Display retrieved post content on page

Case 7:
Show comments in discussion section under the full post (note: this is after redirecting from clicking the button on the post preview)

Action	System Response
1. Clicks on button with arrow on post preview	2. Redirect to empty page
	3. Display loader
	4. Send data request for full post data to API
	5. Display retrieved comments and pertinent data in the comment section below the post content section

Case 8:
Adding a comment to a post's discussion section

Action	System Response
1. Clicks on text editing box	2. Allows user to type in the editing box
3. User types the comment	4. Validate user input in real-time
5. User clicks submission button	6. Send comment submission request to API
	7. If successful, display successfully commented message
	8. Update comment section with newly submitted comment

Case 9:
Allow user to see a collection of their submitted posts

Action	System Response
1. Clicks on “My Profile” dropdown	
2. Clicks on “My Posts” in dropdown	3. Redirect to page
	4. Display loader
	5. Send data request for collection of user-submitted posts to API
	6. Retrieves necessary posts and post metadata as JSON from API request
	7. Format and display posts and post metadata (post previews) on page

Case 10:
Search bar so a user can search for posts by keywords

Action	System Response
1. Clicks search bar at top of page	2. Allows user to type into search bar
3. User types their search query	4. Validate user input in real-time
5. User executes search query by clicking “Search” button OR by pressing enter on keyboard	6. Display loader
	7. Send data request for collection of posts filtered by search query to API
	8. Retrieves necessary posts and post metadata as JSON from API request
	9. Format and display posts and

	post metadata (post previews) on page

Case 11:
Filter comment section by “most recently posted”

Action	System Response
1. Clicks “Filter” dropdown in comment section	
2. Clicks “Recent” in dropdown	3. Display loader
	4. Send data request for collection of comments on this post filtered by most recent to API
	5. Retrieves appropriate comments and comment metadata as JSON from API request
	6. Format and display comments on page

Case 12:
Filter comment section by “most viewed”

Action	System Response
1. Clicks “Filter” dropdown in comment section	
2. Clicks “Views”	3. Display loader
	4. Send data request for collection of comments on this post filtered by most viewed to API
	5. Retrieves appropriate comments and comment metadata as JSON from API request
	6. Format and display comments

	on page

Case 13:
Allowing users to delete their post submissions

Action	System Response
1. Clicks on “My Profile” dropdown	
2. Clicks on “My Posts” in dropdown	3. Loads in the appropriate posts of the user (see Case 9 for details)
4. Clicks “Delete” on post preview	5. Prompts user “Are you sure?” in modal window
6. User clicks “Yes”	7. Send request to delete this post to API
	8. Reload posts on user’s post page

Case 14:
Allowing users to delete their comments on posts

Action	System Response
1. Clicks on “My Profile” dropdown	
2. Clicks on “My Comments” in dropdown	3. Loads in the appropriate collection of comments of the user (see Case 21 for details)
4. Clicks “Delete” on comment preview	5. Prompts user “Are you sure?” in modal window
6. User clicks “Yes”	7. Send request to delete this comment to API
	8. Reload comments on user’s comment page

Case 15:
Letting users edit the content of their post submissions

Action	System Response
1. Clicks on “My Profile” dropdown	
2. Clicks on “My Posts” in dropdown	3. Loads in the appropriate posts of the user (see Case 9 for details)
4. Clicks “Edit” on post preview	5. Opens modal with text box containing current content
6. User changes content	
7. User clicks “Save” in modal window	8. Send request to update post content to API
	9. Reload posts on user’s post page

Case 16:
Letting users edit their comments on posts

Action	System Response
1. Clicks on “My Profile” dropdown	
2. Clicks on “My Comments” in dropdown	3. Loads in the appropriate collection of comments of the user (see Case 21 for details)
4. Clicks “Edit” on comment preview	5. Opens modal with text box containing current comment
6. User changes comment	
7. User clicks “Save” in modal window	8. Send request to update comment to API
	9. Reload comments on user’s comment page

Case 17:
Letting users bookmark/save (and unbookmark/unsave) posts they like

Action	System Response
1. Clicks “Bookmark” on page of full post	2. Send request to add/remove post to user’s bookmarks to API
	3. Show that post has been bookmarked/unbookmarked (a toggle)

Case 18:
Allow user to see a collection of their bookmarked/saved posts

Action	System Response
1. Clicks on “My Profile” dropdown	
2. Clicks on “My Bookmarks” in dropdown	3. Display loader
	4. Send data request for bookmarked posts to API
	5. Retrieves bookmarked posts and post metadata as JSON from API request
	6. Format and display bookmarked posts and post metadata on appropriate page

Case 19:
Paginate posts previews that are displayed on the page

Action	System Response
1. Navigates to any page that displays a collection of items (posts, comments)	2. Send data request to API (posts, or comments in comment section)
	3. Returns collection of posts or

	comments in paginated manner
	4. Format and display posts or comments with pagination bar
5. User clicks pagination	6. Re-send data request to API but with different page number parameter
	7. Format and display results of the correct page

Case 20:
User changing password

Action	System Response
1. Clicks on “My Profile” dropdown	
2. Clicks on “Change Password”	3. Opens modal window to change password
4. User types in current password and new password	5. Validate inputs in real-time
6. User clicks “Submit”	6. Send password change request to API
	7. If credentials validated, make the password change in the database
	8. Display “Successfully changed” message

Case 21:
User viewing all of their comments

Action	System Response
1. Clicks on “My Profile” dropdown	
2. Clicks on “My Comments” in dropdown	3. Redirect page

	4. Display loader
	5. Send data request for all of user's comments to API
	6. Retrieve required comments data and metadata as JSON from API
	7. Format and display comment collection and relevant post info on page