**CST-323 Design Report Template**

|  |  |  |
| --- | --- | --- |
| **Topic:** | Topic 5: Fully Functional Website Milestone 2 | |
| **Date:** | 11/22/2020 | |
| **Revision:** | 2.0 | |
| **Team:** | 1. Tim James | |
| 1. Daniel Cender | |
| 1. Jean Noel | |
|  | |
|  |  | |
|  |  | |
| **Weekly Team Status Summary:** | |  |  |  |  | | --- | --- | --- | --- | | **User Story** | **Team**  **Member** | **Hours**  **Worked** | **Hours Remaining** | | *As a dev I would like to have Sprints, burndown chart, and repo setup for project management.* | *Tim* | *1* | *0* | | *As a dev, I would like to have general tooling that will support the building of a server and web application.* | *Daniel* | *3* | *0* | | *As a dev, I would like a database and server routes which perform data operations on the data.* | *Daniel* | *5* | *0* | | *As a dev, I would like up-to-date documentation on our service API and client application.* | *Daniel/Tim* | *1* | *0* | | *As a dev I would like my users to have a landing page that requests they login or register* | *Tim* | *3* | *0* | | *As a dev I need a page that allows users to login to their account* | *Tim* | *1* | *0* | | *As a dev I would like a high level video overview of the code, webdesign, and features for presenting current progress.* | *Tim* | *1* | *0* | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | | |
| **GIT URL:** | *https://github.com/tjames222/CST-323CLC* | |
| **Peer Review:** | *Y/N* | We acknowledge that our team has reviewed this report and we agree to the approach we are all taking. |

**Planning Documentation**

**Agile Scrum Product Backlog:**

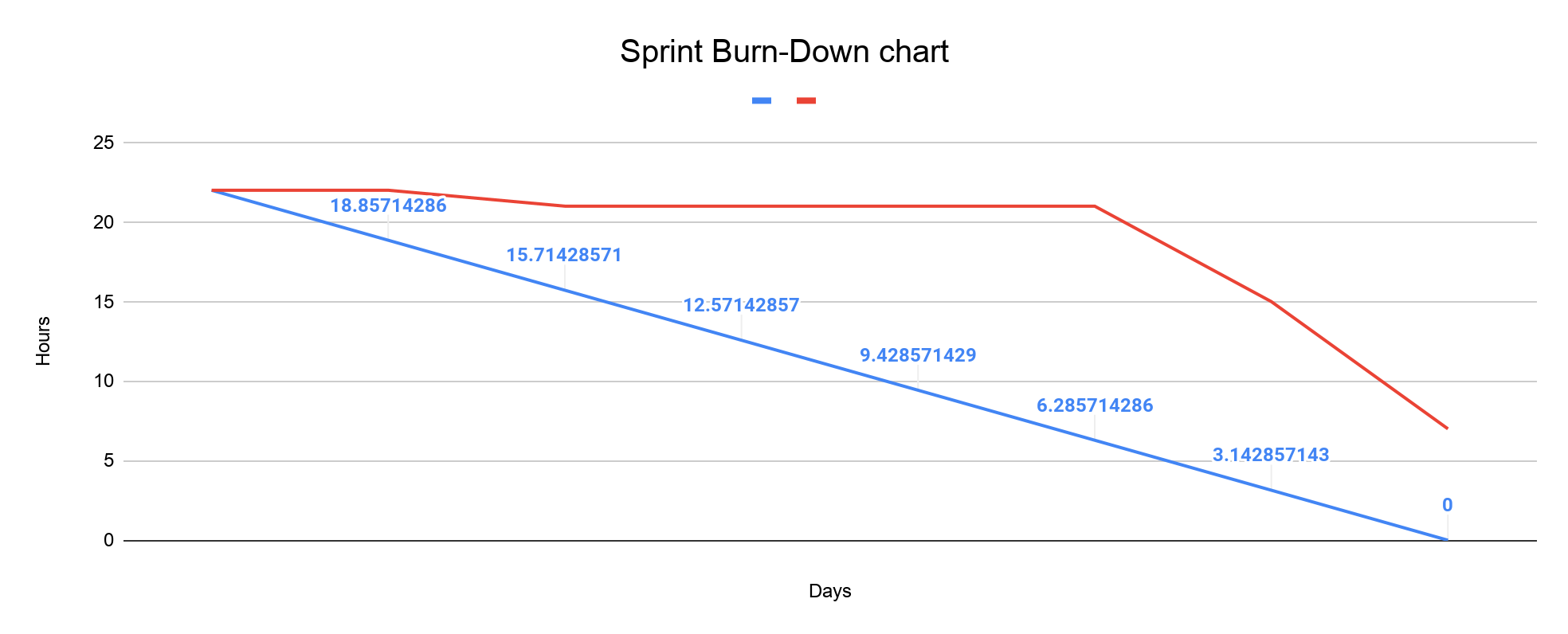
*Here is the link to our project backlog in our GitHub Project:* [*https://github.com/tjames222/CST-323CLC/projects/1#column-11492782*](https://github.com/tjames222/CST-323CLC/projects/1#column-11492782)

**Agile Scrum Sprint Backlog:**

The Sprint backlog (to do) list can be found in the GitHub Project: <https://github.com/tjames222/CST-323CLC/projects/1#column-11492788>

**Agile Scrum Burn Down Chart:**

*This needs to contain a URL to Bitbucket Scrum Burn Down Chart Artifact.*

****

**Agile Retrospective Results:**

*The following table should be completed after each Retrospective on Things That Went Well (Keep Doing). An alternative to the following table is to use a Mind Mapping tool such as Coggle. If you use a Mind Mapping tool, you must include a URL or Image File.*

|  |
| --- |
| **What Went Well** |
| Having a centralized method of communication helped communicate team standards to the group on how to complete and submit work. |
| The initial setup with react/ node/ npm/ and base project file worked flawlessly |
|  |

*The following table should be completed after each Retrospective on Things That Didn’t Go Well (Stop Doing) and What Would Be Done Differently Next Time with an Action Plan to Improve (Try Doing and Continuous Improvement). An alternative to the following table is to use a Mind Mapping tool such as Coggle. If you use a Mind Mapping tool, you must include a URL or Image File.*

|  |  |  |
| --- | --- | --- |
| **What Did Not Go Well** | **Action Plan** | **Due Date** |
|  |  |  |
|  |  |  |
|  |  |  |

**Design Documentation**

**Install Instructions:**

*Include step-by-step instructions for setting up your database, configuring, and deploying/installing your application. This section should also include detailed instructions for what configuration files are required by your application, what configuration settings need to be adjusted for various runtime (development or production) environments, and where the files need to be deployed to. This section should also contain detailed instructions for how to clone your application source code from BitBucket and deploy the application to an externally hosted site.*

The first step is to clone the repository to someone’s machine.

Once cloned, change directories to the the /app folder. Open a Terminal and run `npm install`, then change to the /client directory and run `npm install` (or `npm i`) again.

At this point, all the dependencies should be installed and linked.

You can now run the development server by changing back to the app/ directory and running `npm run start`. This will run the development server in server.js and start the react-script process in the client/ directory.

In order to build the application for deployment, one needs to change to the /client directory. Run `npm run build` and wait for the process to finish.

Now, if the app is deployed as a Node.js application to Heroku or any other platform, that service’s build system can install dependencies at the root /app level and start the application server with `npm run start`.

**General Technical Approach:**

*In your own words describe your approach and design here. You should also summarize any meeting notes, brain storming sessions, and so forth that you want to retain thru the design of your project.*

Our team approach is to design a blogging platform that faces the open web. We will work together using a variety of tools to help collaboration efforts. We will use Discord to stay in touch throughout the week and collaborate on projects. We will use GitHub for source code management and version control. We will use Google docs for sprint and burndown charts as well as Design report updates. We will use Draw.io for any wireframing or UML designs. The ER diagrams will be generated through MySQL Workbench.

**Key Technical Design Decisions:**

*Any final technical design decisions, (e.g., framework decisions) should be documented here. List the technology/framework, its purpose in the design, and why it was chosen.*

**Non-code/Team Tools:**

Our team will design a basic blogging platform..

Discord will be used for communication.

Google docs will be used for documentation.

GitHub will be used for version control.

Visual Studio Code will be used as the base code editor of choice for use with our toolset, but any other editor could be used as a replacement.

Loom will be used for video presenting.

**Framework/Language:**

Our application will use a pure JavaScript stack, comprised of:

* React - A frontend development framework
* NodeJS - A backend, server-side development framework
* MySQL - A database engine designed for the web, used by Uber and many other high-profile firms

**Libraries/Packages:**

In order to get more “bang for our buck” with the React/NodeJS stack, we will use an assortment of packages and libraries to fill out the functionality of our application. All of these packages are accessible via CDN or NPM package.

* MySQL: <https://www.npmjs.com/package/mysql>
* EditorJS: <https://editorjs.io/>
* Axios: <https://www.npmjs.com/package/axios>
* UUID: <https://www.npmjs.com/package/uuid>
* React Bootstrap: <https://react-bootstrap.github.io/>

**Known Issues:**

*Any anomalies or known issues in the code or functionality should be documented here.*

The most notable of all current issues is the lack-lustre frontend UI. As of right now, the frontend pages are quite bare and do not connect with the backend service-level API quite yet. This is due to limited team buy-in and collaboration up until this milestone, as can be observed from the burn-down sheet and user story breakdowns.

It will be a priority of the next sprint plan to get the UI fully filled out and functional with the database before moving on to the next milestone requirements.

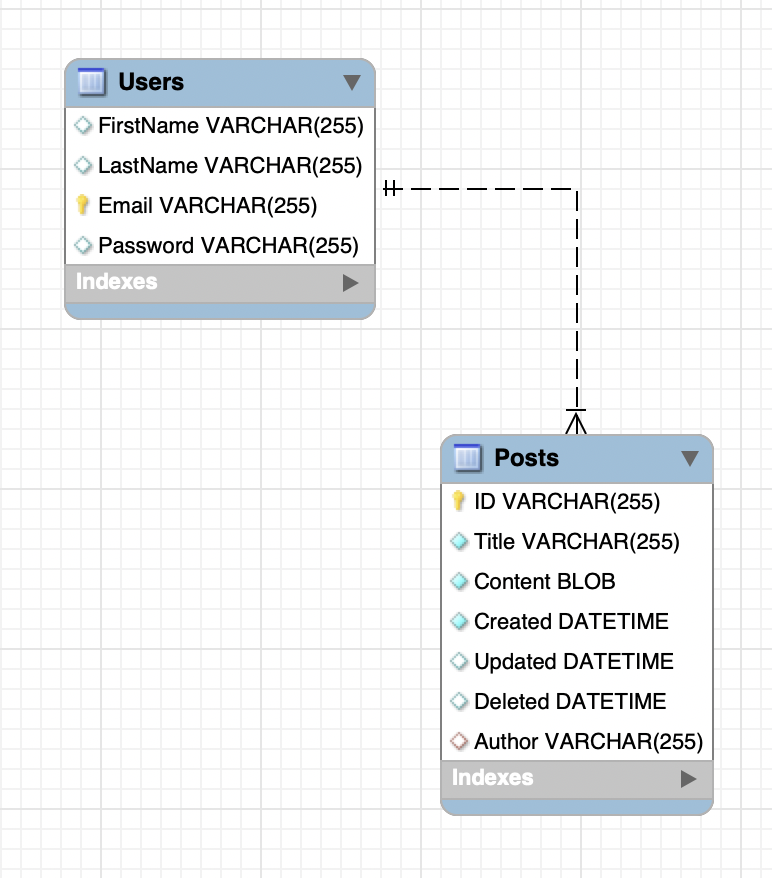
**Risks:**

*Any risks, unknowns, or general project elements that should be tracked for risk management should be documented here.*

No known Risks at this time.

**ER Diagram:**

Below is an image of the current database ER diagram.

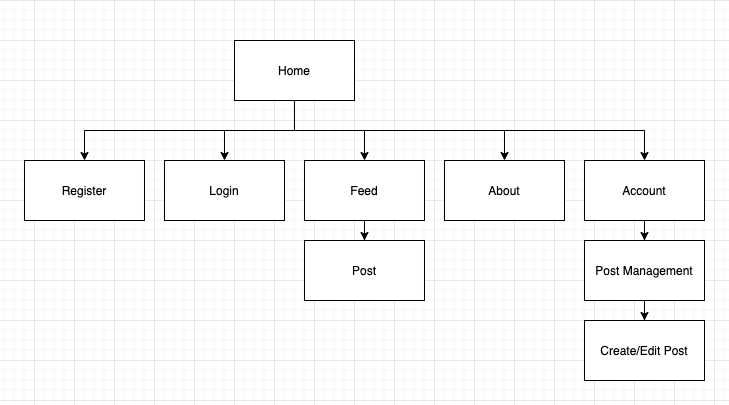
**

**DDL Scripts:**

The current DDL Create script can be found at the following link: <https://github.com/tjames222/CST-323CLC/tree/main/scripts>

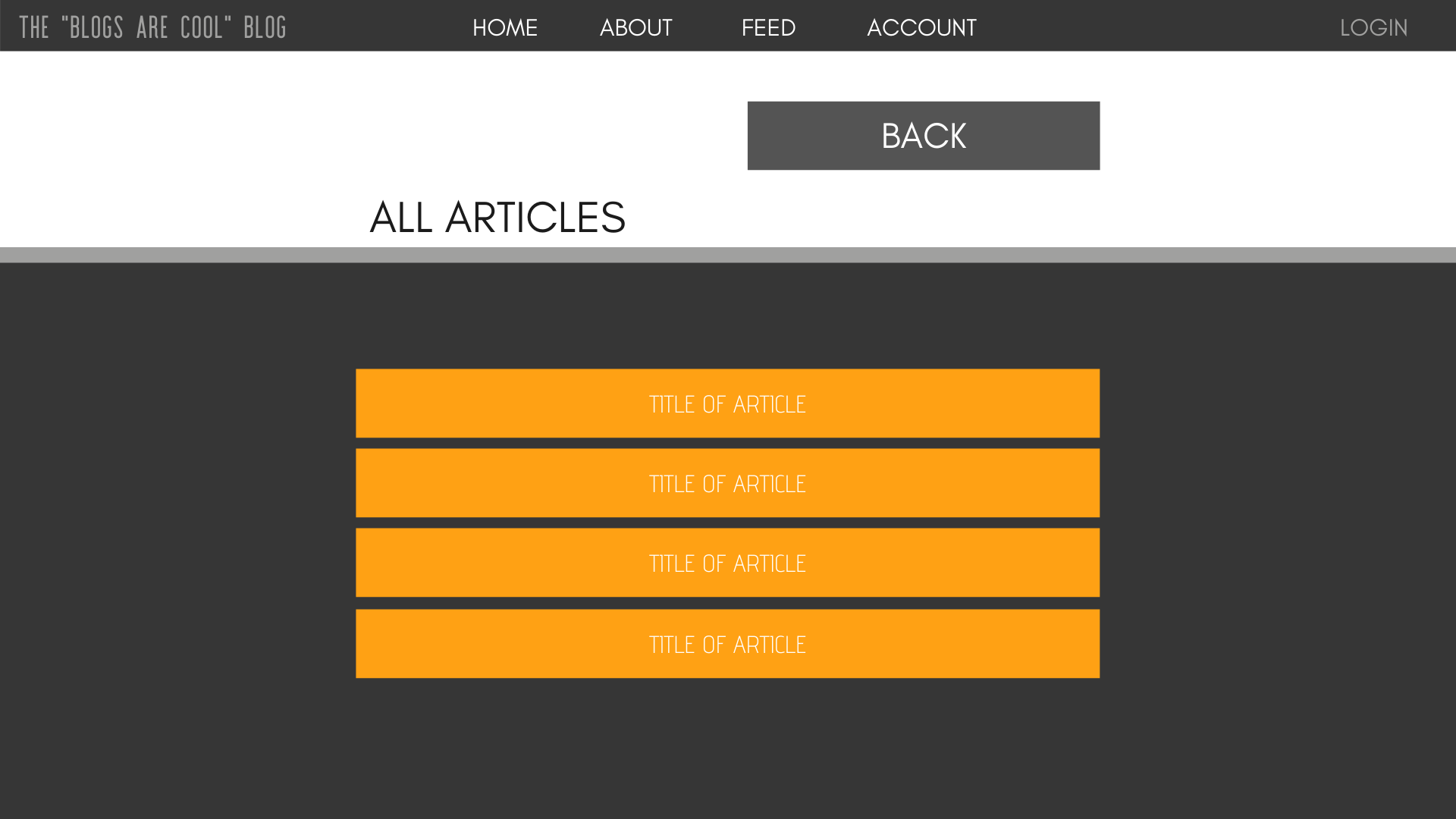
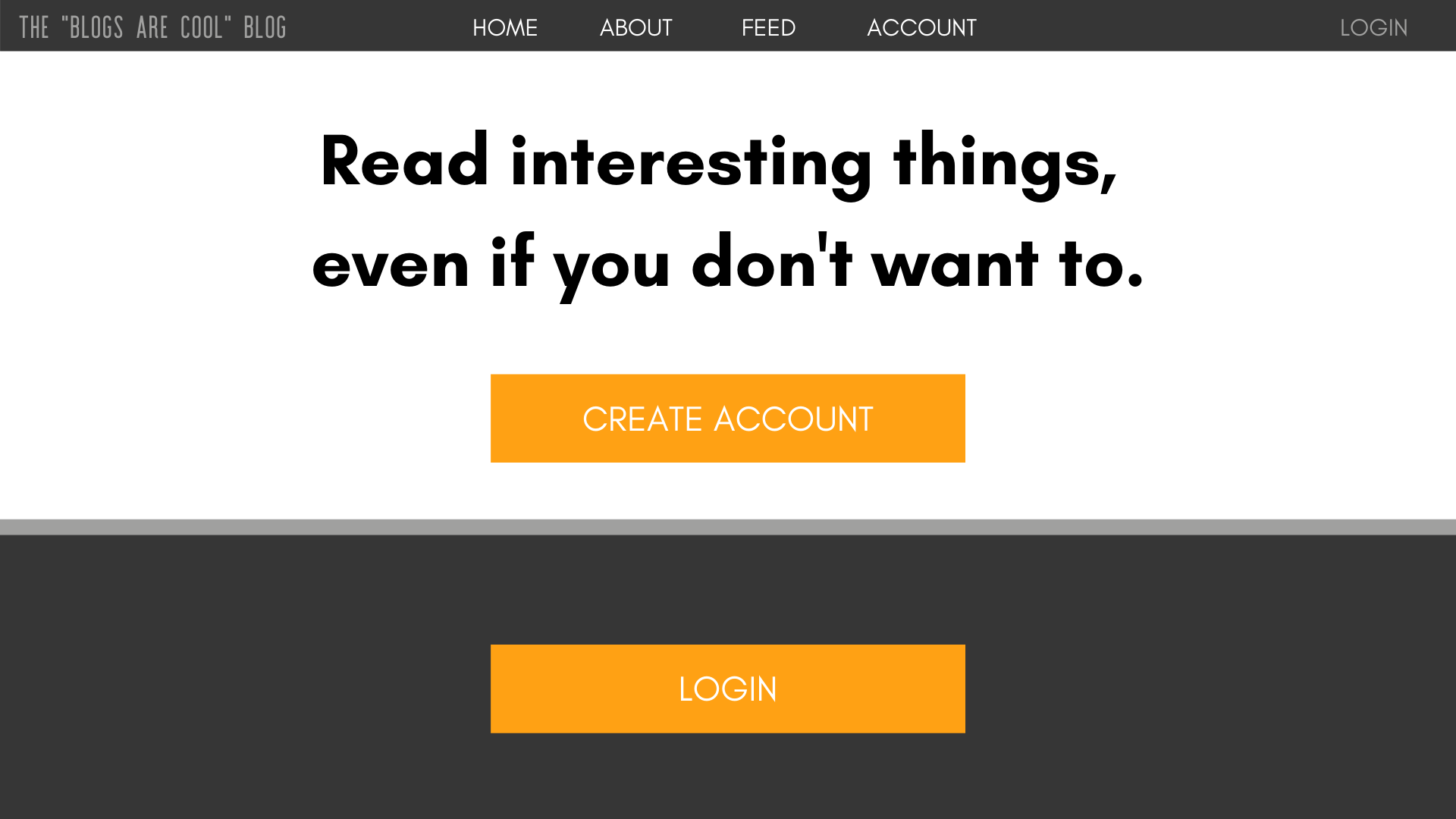
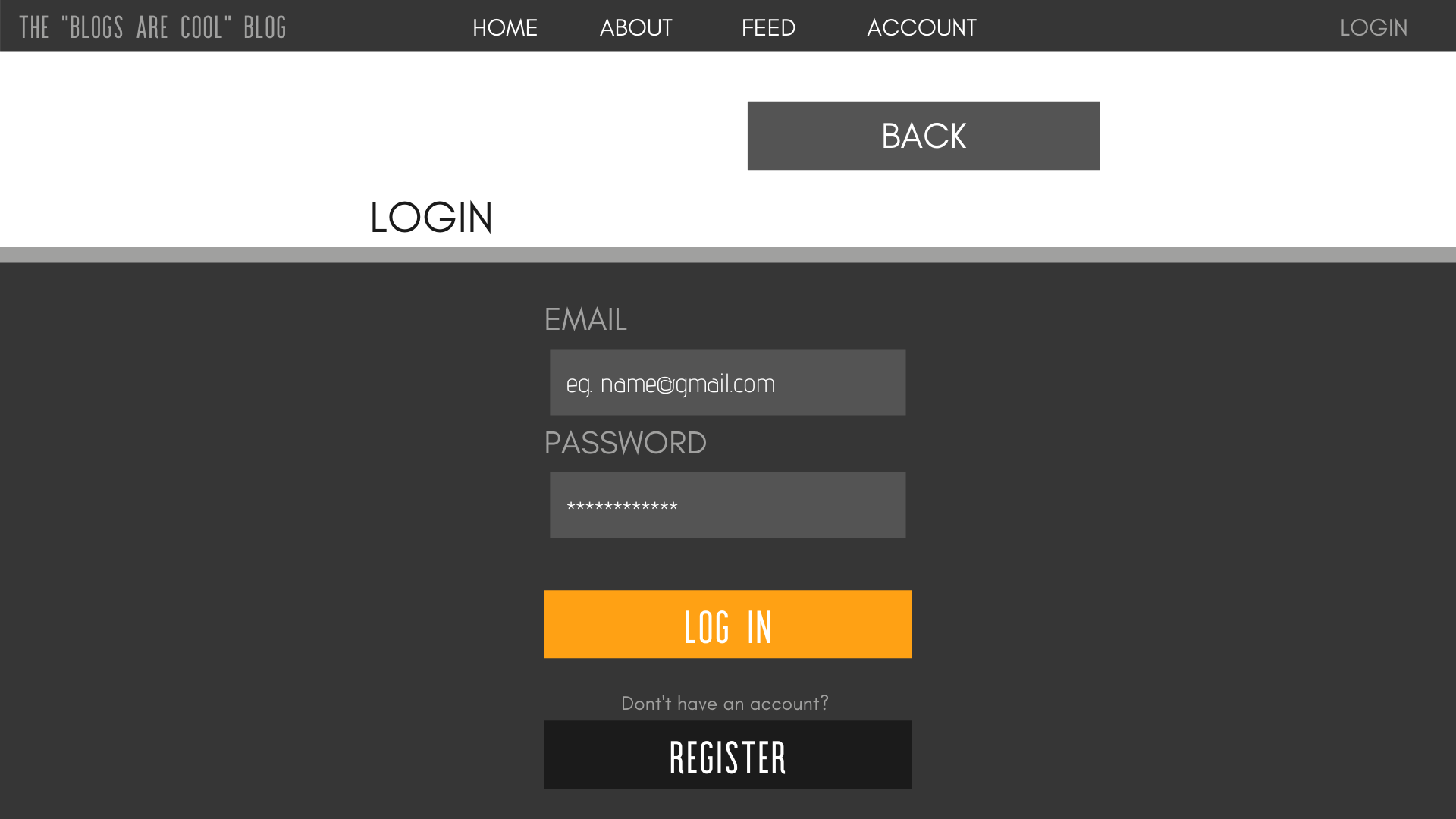
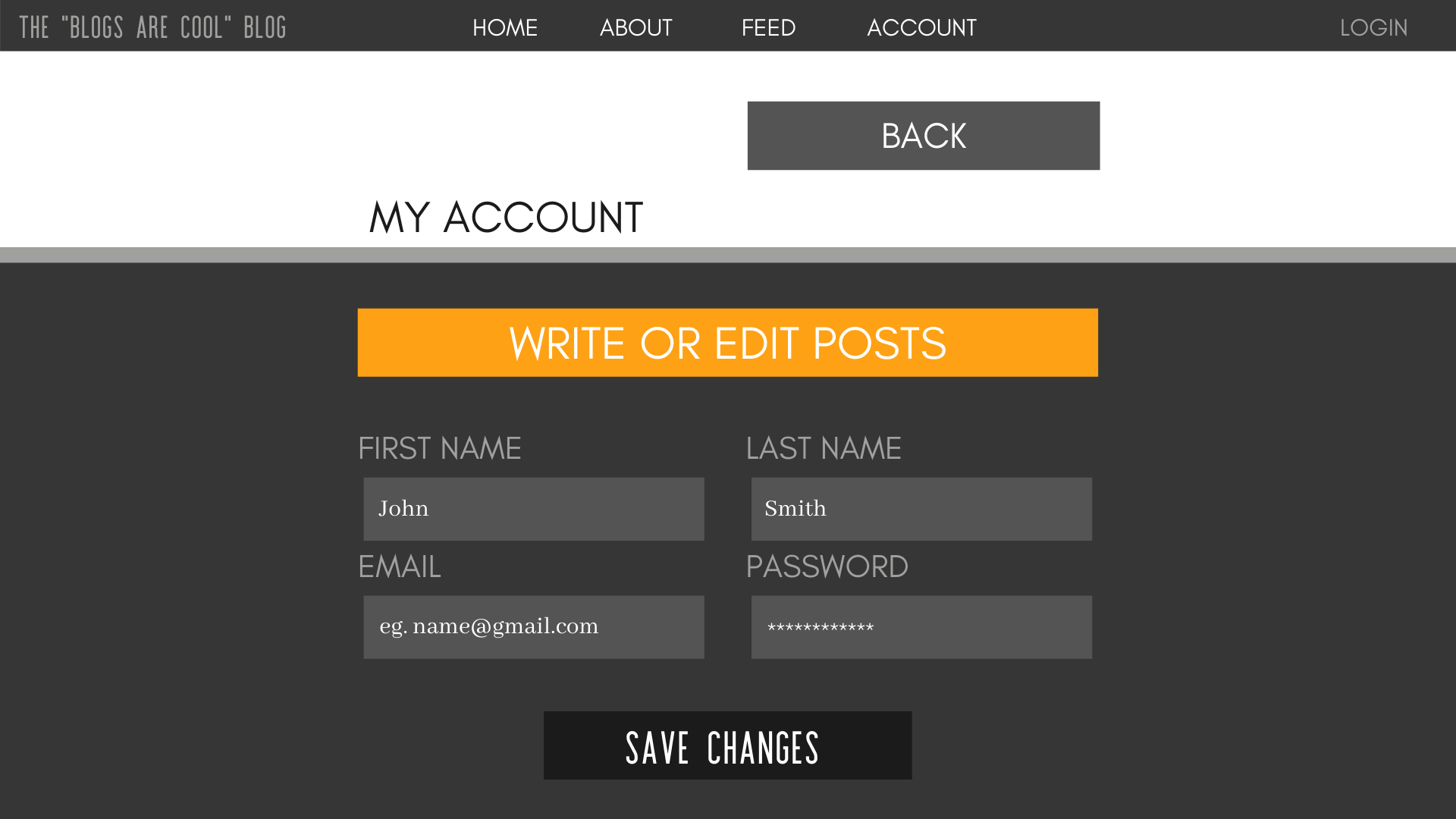
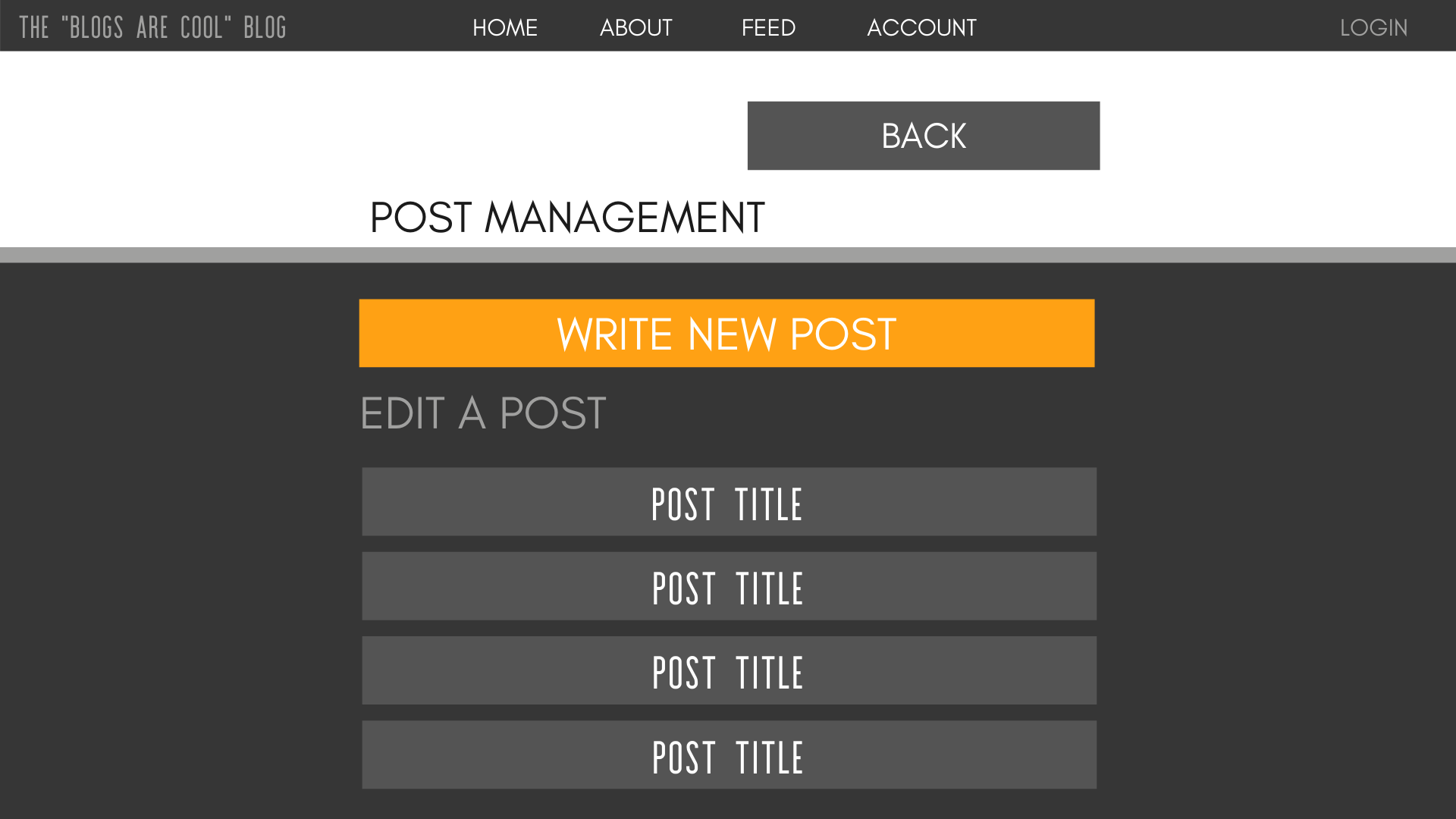
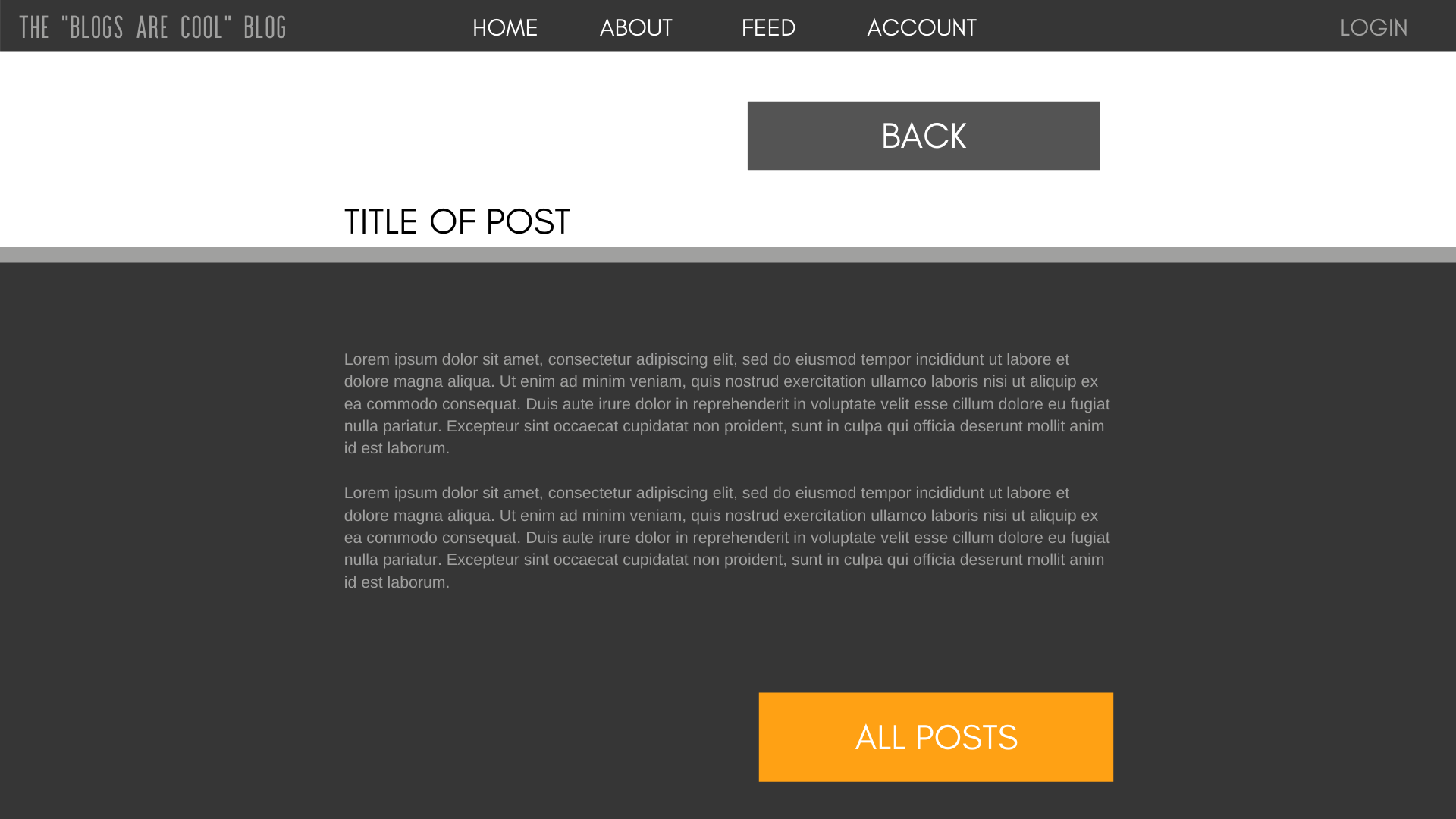
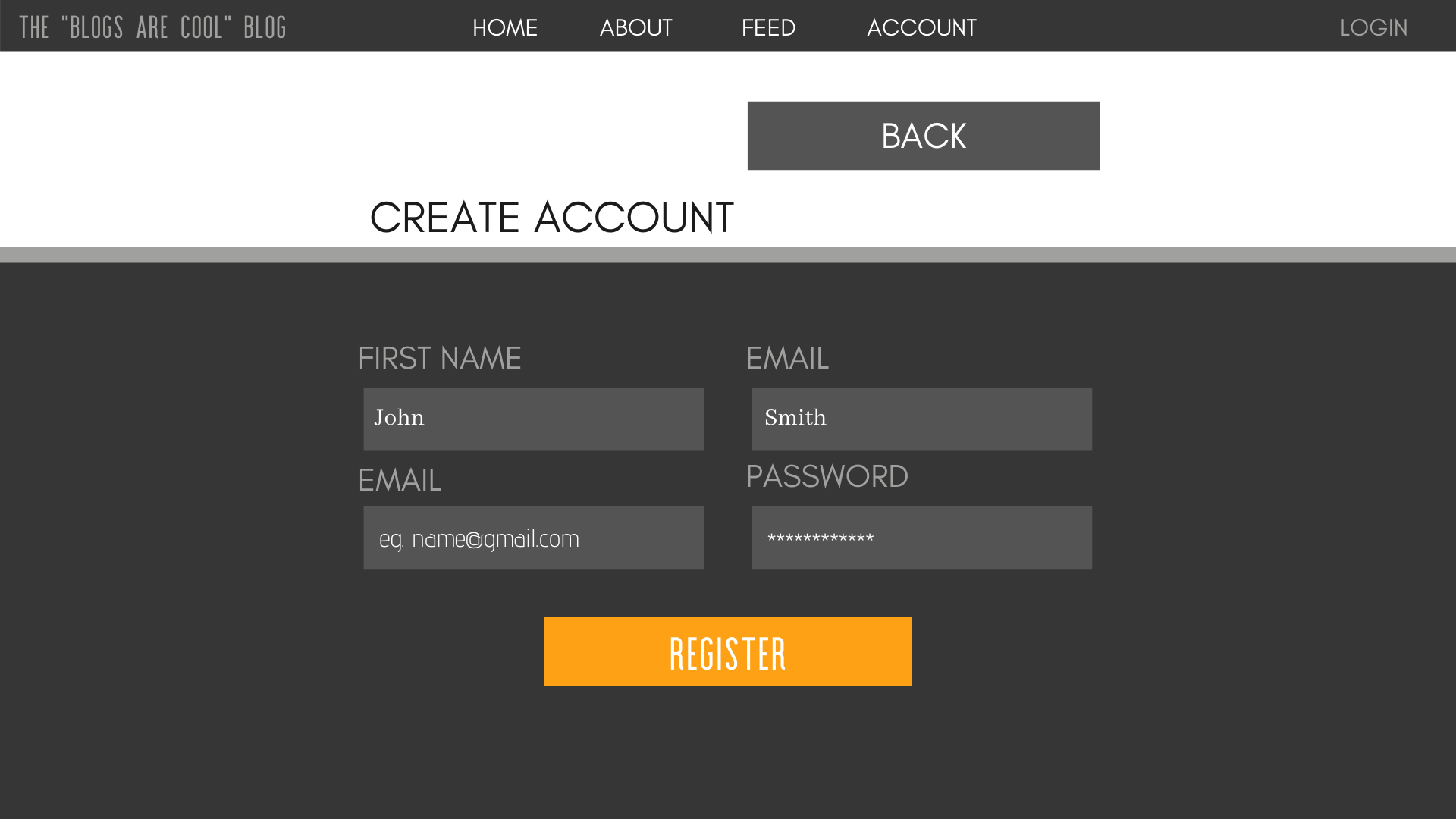
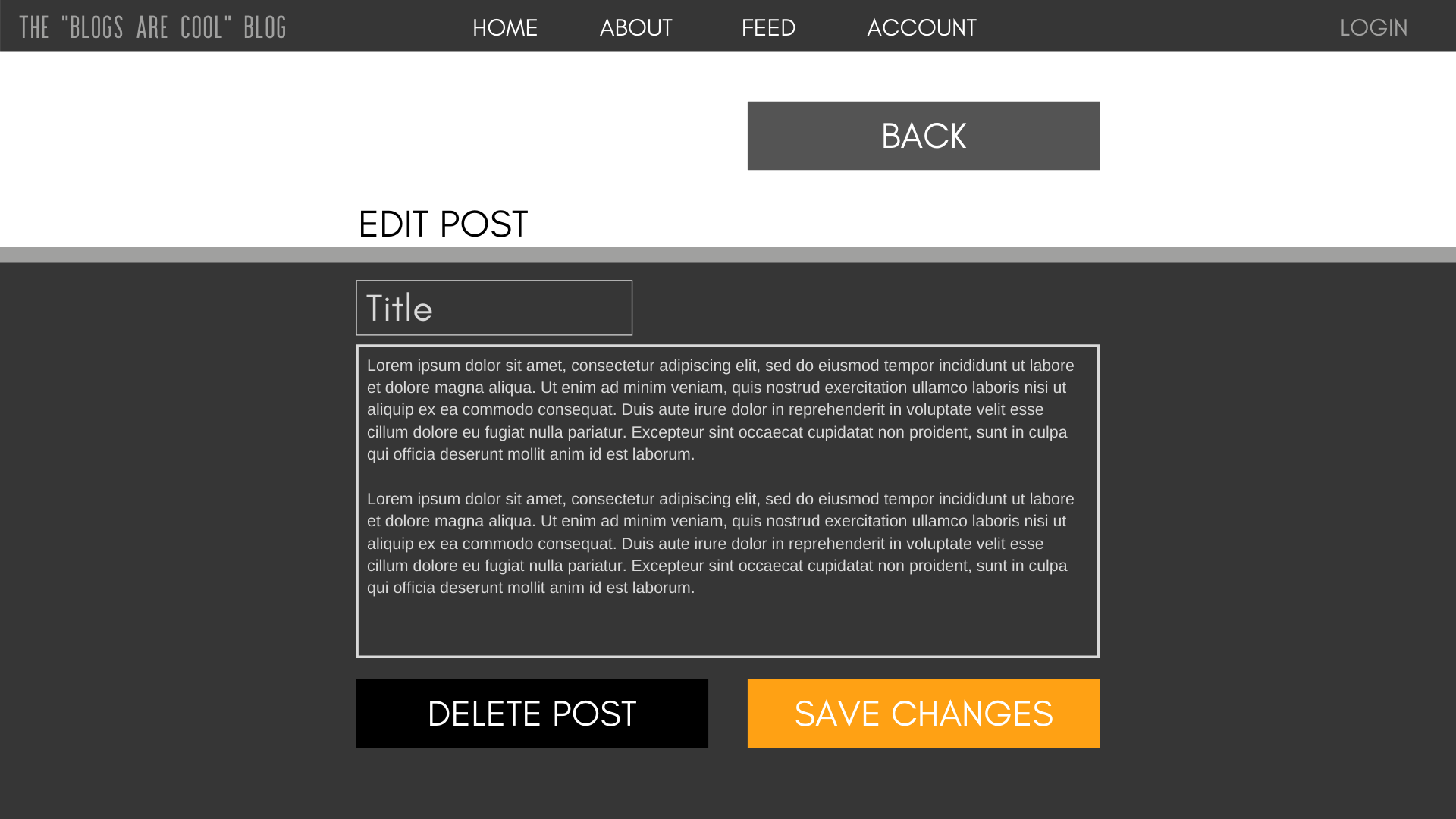
**Sitemap Diagram:**

*Include an image file of your Sitemap diagram.*

**

**User Interface Diagrams:**

*You should insert any wireframe drawings or white board concepts that were developed to support your application. If you have no supporting documentation, please explain the rationale for labeling this section N/A.*

**

**Class Diagrams:**

*You should insert any class diagrams here. Your class diagrams should be drawn correctly with the three appropriate class compartments, + and – minus to indicate accessibility, and the data types for the state/properties as well as method arguments and return types. If you have no supporting documentation, please explain the rationale for labeling this section N/A.*

**UML Diagram:**

No UML diagram at this time, while we sort out how we will organize and structure our code layers between the front/back ends.

**Service API Design:**

*This section should fully document any Third Party Service Interface API’s being consumed or application specific Service API’s being published, how to access the service, what parameters are required by the API, and the detailed JSON data format specification that could be used by athird party developer to integrate with the service and API.*

As of this milestone, we have no expectation that we will need to consume any 3rd-party APIs or services.

**Service API**

There will be an API layer between the web client and the database (available at route /api, port 3001). This layer supports REST routes for interacting with both Users and Posts records.

Users

* /user/:email (GET)
* /users (GET)
* /user/create (POST)
* /user/update (POST)

Posts

* /post/:id (GET)
* /posts (GET)
* /post/create (POST)
* /post/update (POST)

More documentation about the expected POST request parameters and the return objects of each route will be added soon to this documentation set.

**Security Design:**

*This section should outline the design for how authentication and authorization was supported. This section should also contain all of the roles and privileges that are supported by the design.*

Users will use their email and a unique password to log into their accounts. We will salt and hash the passwords before we store them. When the user makes a login attempt, we will salt and hash their inputted string, then compare it to the stored hash. This way, we won’t ever store their raw password in a table.

**Other Documentation:**

*You should insert any additional drawings, storyboards, white board pictures, project schedules, tasks lists, and so forth that support your approach, design, and project. If you have no supporting documentation, please explain the rationale for laveling this section N/A.*