WEB102 | Intermediate Web Development

Intermediate Web Development Summer 2025 (@ Section 1 | Wednesdays 4PM - 6PM PDT)
Personal Member ID#: **107789**

Week 8: Lab 7 - Bet

Overview

In this lab, you will create a web application called Bet using React and Supabase. Bet is a forum built for Gen Z thrill-seekers where users can post challenges for other users. Users can take on a challenge by clicking the “Bet” button.

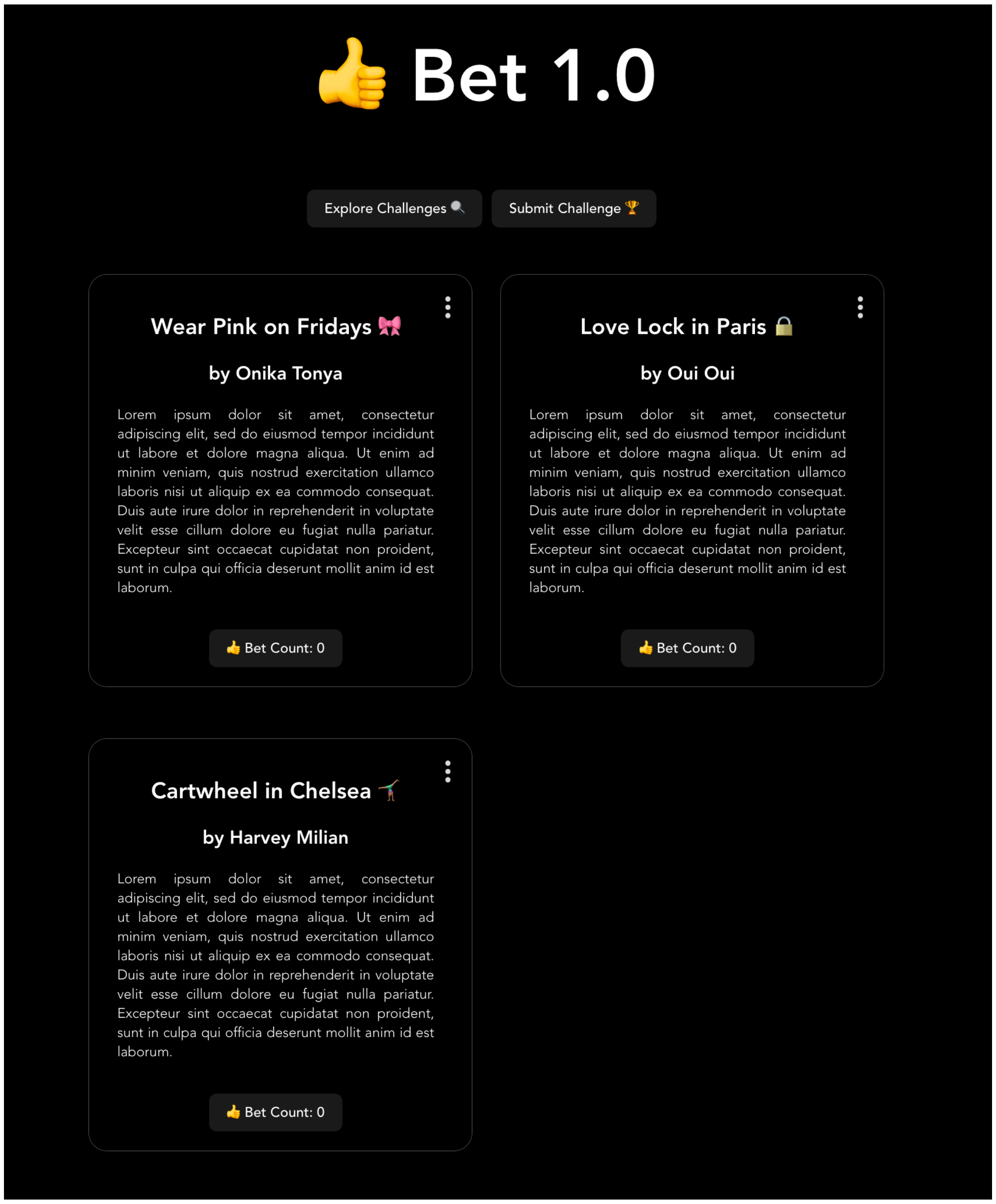
View an exemplar of what you'll be creating in this lab here!

Features

Required Features

The user is able to perform all four CRUD operations. The user can create a new challenge, read previous challenges, update a challenge, and delete a challenge.

- [ ] All submitted challenges can be read on the homepage
- [ ] A create form allows users to submit a new challenge
- [ ] A challenge can be updated once it has been submitted
- [ ] A challenge can be deleted once it has been submitted



Wear Pink on Fridays 🎀

by Onika Tonya

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

👍 Bet Count: 0

Love Lock in Paris 🔒

by Oui Oui

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

👍 Bet Count: 0

Cartwheel in Chelsea 🤸

by Harvey Milian

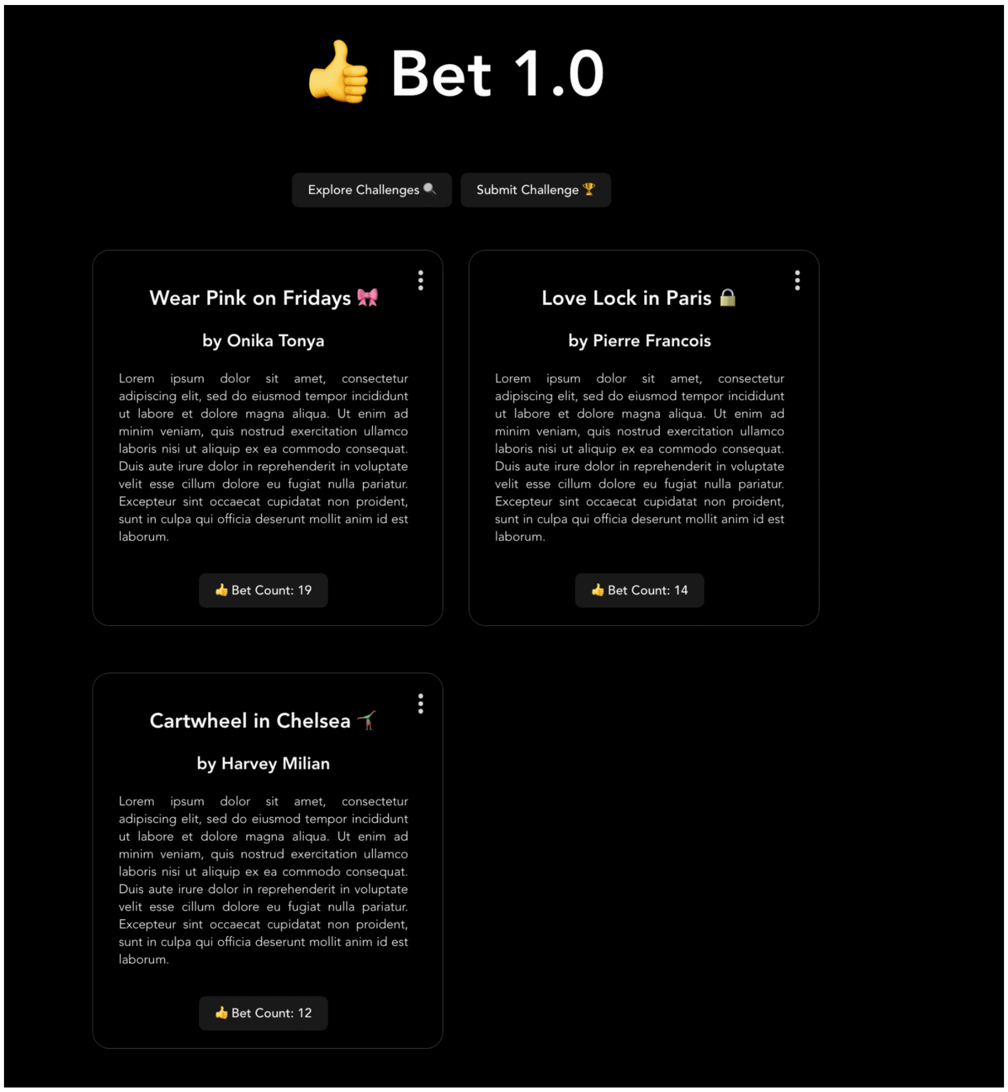
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

👍 Bet Count: 0

Stretch Features

The app is able to keep track of the bet count for each challenge. When a user clicks the bet button, the bet count is updated in and saved to the database.

- [ ] The site displays the total number of users who have indicated they have accepted each challenge



👍 Bet 1.0

Wear Pink on Fridays 🎀

by Onika Tonya

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

👍 Bet Count: 19

Love Lock in Paris 🔒

by Pierre Francois

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

👍 Bet Count: 14

Cartwheel in Chelsea 🤸

by Harvey Milian

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

👍 Bet Count: 12

🎯 Goals

By the end of this lab you will be able to...

- [ ] Setup a database for an application using Supabase
- [ ] Perform CRUD operations to manage data for a web application

Resources

- MDN Web Docs Glossary: CRUD
- Supabase Documentation
- Supabase JavaScript Library

Lab Instructions

Required Features

Step 0: Explore the Starter Code

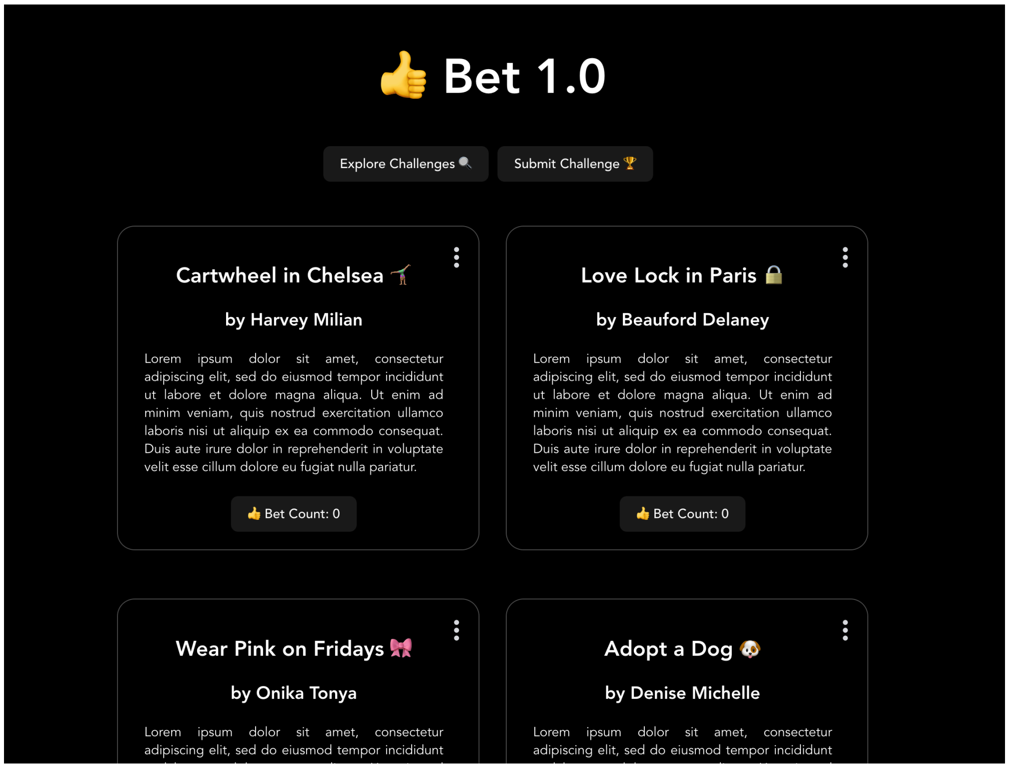
For this lab, we provide you with the front-end of this web application so that you can focus on setting up the database and performing the CRUD operations. In this step, download the starter code and explore the interface.

- [ ] Create your own copy of the starter code by forking the Unit 7 GitHub Repo
- [ ] Clone your forked repo. Your command should look similar to the following:

  ```
  git clone git@github.com:YOUR-USERNAME/web102_unit7lab.git
  ```

- [ ] Navigate into your cloned repository.
- [ ] Start the React app:
  - [ ] Using the Terminal, navigate into the root directory of your project.
  - [ ] To install the dependencies, enter the command `npm install` into the Terminal.
  - [ ] To start the app, enter the command `npm run dev` into the Terminal.
- [ ] Explore the application to familiarize yourself with the user interface.

📍 **Checkpoint 0**: At this point in the lab, your app should look like this ⤵️



Cartwheel in Chelsea 🤸

by Harvey Milian

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Love Lock in Paris 🔒

by Beauford Delaney

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Wear Pink on Fridays 🎀

by Onika Tonya

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt

Adopt a Dog 🐶

by Denise Michelle

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt

Step 1: Sign Up for Supabase

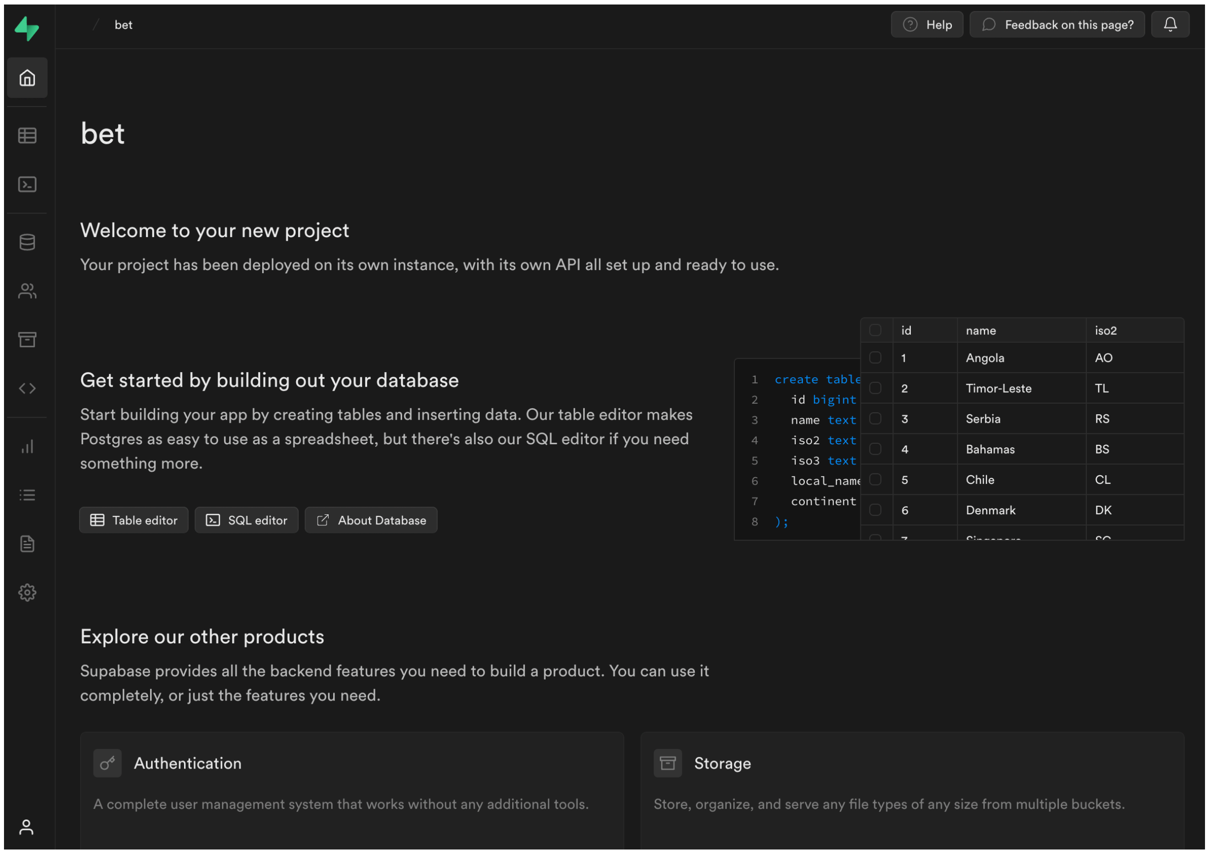
In this step, sign up for Supabase and begin setting up the database for the web application.

✨ AI Opportunity

▶ *Use AI to explain code concepts* → Supabase

- [ ] Go to the Supabase website
- [ ] Sign up for a Supabase account:
    - [ ] Click the green "Start your project" button.
    - [ ] At the bottom of the sign-in form, click the "Sign Up Now" link.
    - [ ] Create an account using your GitHub credentials by clicking the "Continue with GitHub" button.
- [ ] Create a new project:
    - [ ] Click the green "New Project" button to create a new project
    - [ ] Set the name of your project to `bet` and create a database password
    - [ ] Click the green "Create new project" button to submit to form
    - [ ] After submitting the form, give Supabase a few minutes to setup your project

📍 **Checkpoint 1**: At this point in the lab, the page in your browser should look like this ⤵️



bet

Welcome to your new project

Your project has been deployed on its own instance, with its own API all set up and ready to use.

Get started by building out your database

Start building your app by creating tables and inserting data. Our table editor makes Postgres as easy to use as a spreadsheet, but there's also our SQL editor if you need something more.

```
create table
  id bigint
  name text
  iso2 text
  iso3 text
  local_name
  continent
);
```

id	name	iso2
1	Angola	AO
2	Timor-Leste	TL
3	Serbia	RS
4	Bahamas	BS
5	Chile	CL
6	Denmark	DK

Explore our other products

Supabase provides all the backend features you need to build a product. You can use it completely, or just the features you need.

Authentication

A complete user management system that works without any additional tools.

Storage

Store, organize, and serve any file types of any size from multiple buckets.

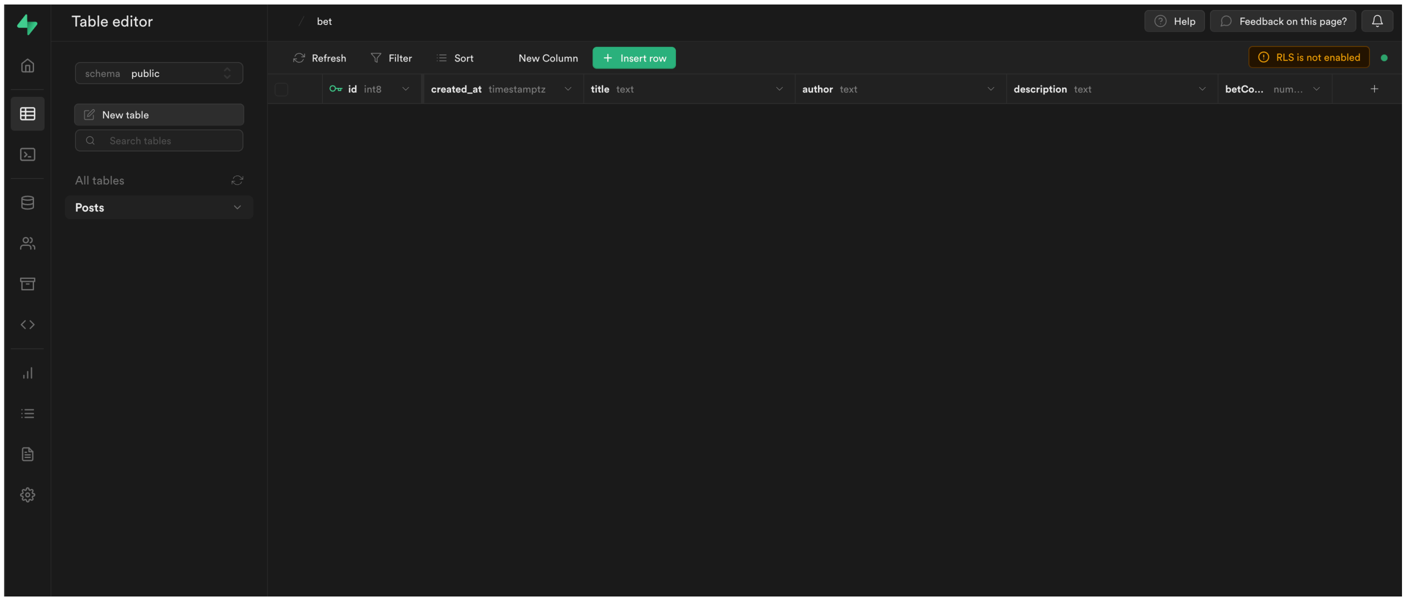
Step 2: Set Up the Database

In this step, you will create a new Supabase database for your project. For this project, we want to save data for each challenge that users post on Bet. Our database needs to store the title, author, description, and bet count for each post.

- [ ] Click the "Database" icon in the left side bar menu

- [ ] Click the green "New table" button to create a new table in the database
- [ ] Create a `Posts` table:
    - [ ] Name the table `Posts`
    - [ ] Uncheck the option "Enable Row Level Security"
    - [ ] Check the option "Enable Realtime"
    - [ ] By default, your table should have a column called `id` and `created_at`. The `id` attribute is a unique identifier for each entry in your database. When a new entry is inserted into the database, Supabase will assign it a new ID number. The `created_at` attribute is a timestamp of when an entry was inserted into the database. Supabase will automatically assign it a timestamp when it is inserted.
        - [ ] Create a column called `title`. For Type, select `text`. Set the default value to `Null`.
        - [ ] Create a column called `author`. For Type, select `text`. Set the default value to `Null`.
        - [ ] Create a column called `description`. For Type, select `text`. Set the default value to `Null`.
        - [ ] Create a column called `betCount`. For Type, select `numeric`. Set the default value to `0`.
        - [ ] Click the green "Save" button to save the table.
- [ ] To view your table, click the "Table Editor" icon in the left side bar menu. Then, under all tables and click "Posts". Your table should be empty, as we did not add any entries yet into the database.

📍 **Checkpoint 2**: At this point in the lab, the page in your browser should look like this ⤵️



ep 3: Connect to the Database

In this step, you will add the Supabase client to the React application so that we can connect the front-end to the database.

- [ ] Terminate the React server in the terminal.
- [ ] Install the Supabase library using the following command:

```
npm install @supabase/supabase-js
```

- [ ] To start the application again, enter the command `npm run dev` into the Terminal.
- [ ] In the `src` directory, create a JavaScript file called `client.js`. In this file, we will create our Supabase client that will connect our React app to the Supabase database.

In `client.js`:

- [ ] Import `createClient` from Supabase using the following import statement

  ```
  import { createClient } from '@supabase/supabase-js'
  ```

✨ AI Opportunity

▶ *Use AI to explain code concepts* → createClient

- [ ] Create a variable called `URL`. Assign it to a string containing your Supabase Project URL. To find the Project URL on the Supabase website, click the "Project overview" house icon in the left side bar menu and then scroll down to where it says "Project API". Click the "Copy" button next to "Project URL".

  ```
  const URL = 'insert your Project URL here'
  ```

- [ ] Create a variable called `API_KEY`. Assign it your a string containing your Supabase Project API Key. To find the Project API Key on the Supabase website, click the "Project overview" house icon in the left side bar menu and then scroll down to where it says "Project API". Click the "Copy" button next to "API Key".

  ```
  const API_KEY = 'insert your Project API key here'
  ```

  - [ ] Create a variable called `supabase`. Assign it a function call to `createClient()`. Pass `URL` and `API_KEY` variables to the `createClient()` function. This line creates a new Supabase client instance that will allow your app to connect to your Supabase project.

  ```
  const supabase = createClient(URL, API_KEY)
  ```

  - [ ] Export the `supabase` variable by adding the `export` keyword before `const supabase`.

  ```
  export const supabase = createClient(URL, API_KEY)
  ```

📍 **Checkpoint 3**: At this point in the lab, your `client.js` file should look like this ⤵️

```
import { createClient } from '@supabase/supabase-js'

const URL = '<insert your URL here'
const API_KEY = '<insert your API key here>'

export const supabase = createClient(URL, API_KEY)
```

Step 4: Create Database Entry

Currently, when a user tries submitting a new challenge, the data does not save to a database.

In this step, we will update the event handler `createPost()` so that it adds a new challenge post to the database when the "Submit New Challenge" form is submitted.

- [ ] Go to `CreatePost.jsx` in the `pages` directory.
- [ ] First, import `supabase` from `client.js`

```
import { supabase } from '../client'
```

- [ ] Then, define a `createPost()` function in the starter code. We will use this function to send data to the database before redirecting to the homepage.
- [ ] Make `createPost()` an asynchronous function by adding the `async` keyword in front of the parameter list.

```
const createPost = async () => {...}
```

- [ ] Add `event` as a parameter to `createPost()` because we will use this function to handle the the form submission event when the user submits the Create Post form.
- [ ] By default, the page will reload whenever the form is submitted. Add `event.preventDefault()` to your function to prevent this behavior.
- [ ] Add the following code snippet to after `event.preventDefault()` in `createPost()`

```
await supabase
  .from('Posts')
  .insert({title: post.title, author: post.author, description: post.description})
  .select()
```

Let's break down what this code snippet is doing:

- First, making calls to the Supabase client is an asynchronous operation so we include the `await` keyword.
- Next, we use the `.from()` method to specify which table in the project we want to use. In this case, we want to access data "from" the `Posts` table.

- Next, we use the `.insert()` method to indicate that we want to perform an insertion operation (the "create" of CRUD). To the `.insert()` method, we pass an object with the `title`, `post`, and `description` each populated with data from the form.
- Lastly, we call the `.select()` method, which returns the database entry once it has been inserted into the database.

✨ AI Opportunity

▶ *Use AI to understand provided code* → "Explain this" with Copilot

- [ ] Lastly, add `window.location = "/"` after the `await ...` code. This will redirect the browser to the root URL aka the homepage of our site.

Your `createPost()` function should now look like this:

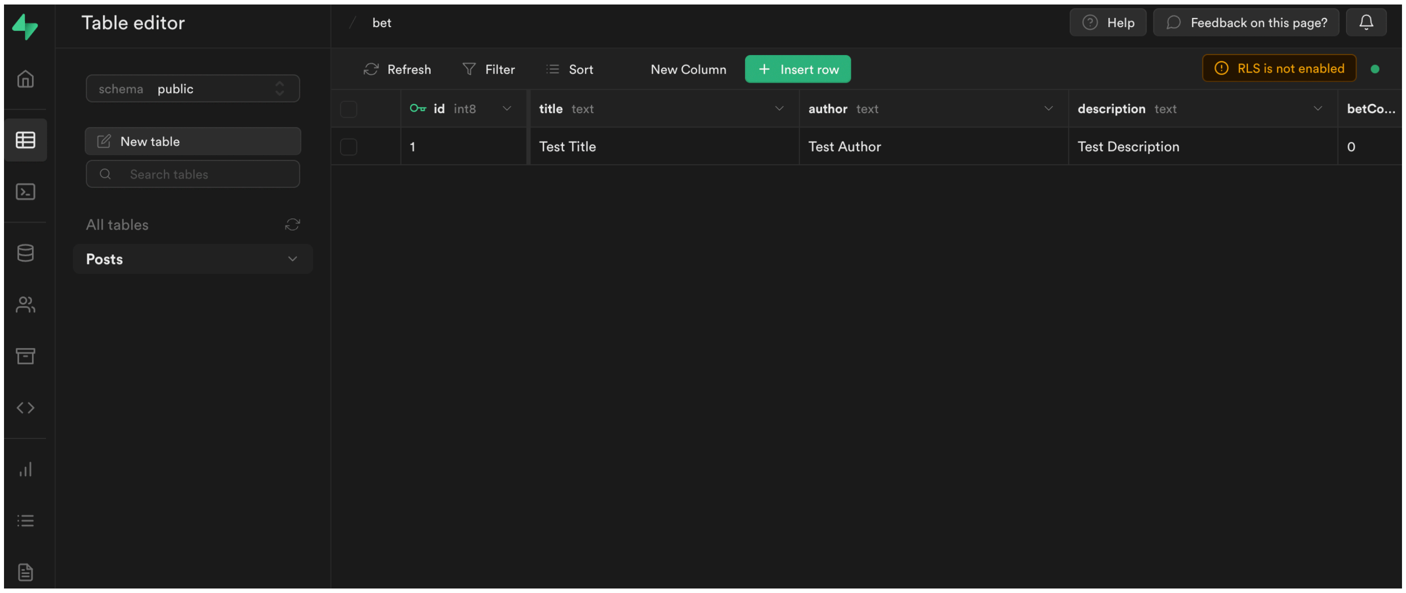
```
const createPost = async (event) => {
  event.preventDefault();

  await supabase
    .from('Posts')
    .insert({title: post.title, author: post.author, description: post.description})
    .select();

  window.location = "/";
}
```

- [ ] Create an `onClick` event handler for the submit button to call `createPost()`.

```
<input type="submit" value="Submit" onClick={createPost} />
```

- [ ] Let's test it! In the web app, fill out the form and click submit. Then, go to the "Table Editor" view on Supabase to see if it worked. To view your table, click the "Table Editor" icon in the left side bar menu. Then, under all tables and click "Posts".

📍 **Checkpoint 4**: At this point in the lab, your Supabase table should look like this ⤵️



id int8	title text	author text	description text	betCo...
1	Test Title	Test Author	Test Description	0

Step 5: Read Database Entries

Currently, when a user goes to the home page, there are four static entries. This is because the `posts` props being passed to the `ReadPosts` view is the hardcoded array `posts` located in `App.jsx`. In this step, we will make a request to the database to instead read all of the entries in the database and display them on the homepage.

- [ ] Go to `ReadPosts.jsx` in the `pages` directory.
- [ ] Import the `supabase` client from `client.js`

```
import { supabase } from '../client'
```

- [ ] Locate the `useEffect()` function in the starter code.
- [ ] Inside the `useEffect()` function, replace the existing code with a function definition for an asynchronous function called `fetchPost()`.
- [ ] Inside the `fetchPost()` function, add the following code:

```
const {data} = await supabase
  .from('Posts')
  .select()
  .order('created_at', { ascending: true })

// set state of posts
setPosts(data)
```

Let's break down what this code snippet is doing:

- First, making calls to the Supabase client is an asynchronous operation so we need to include the `await` keyword.
- Next, we use the `.from()` method to specify which table in the project we want to perform an act
on. In this case, we want to perform an operation "from" the `Posts` table.

- Next, we call the `.select()` method which returns all of the database entries once they have been inserted into the database. We assign this return value to the variable `data`
- Next, we order the database entries. The `.order()` method orders the fetched database entries by a given column(s). The first parameter specifies which column we want to sort by. In this case, we want to sort the data based on the `'created_by'` column. The second parameter takes an object with option parameters. The `ascending` parameter is a boolean. If `true`, the result will be in ascending order. If `false`, the result will be in descending order.
- Last, we update the state of the state variable `posts` by passing all of the returned database entries stored in `data` to `setPosts()`.

✨ AI Opportunity

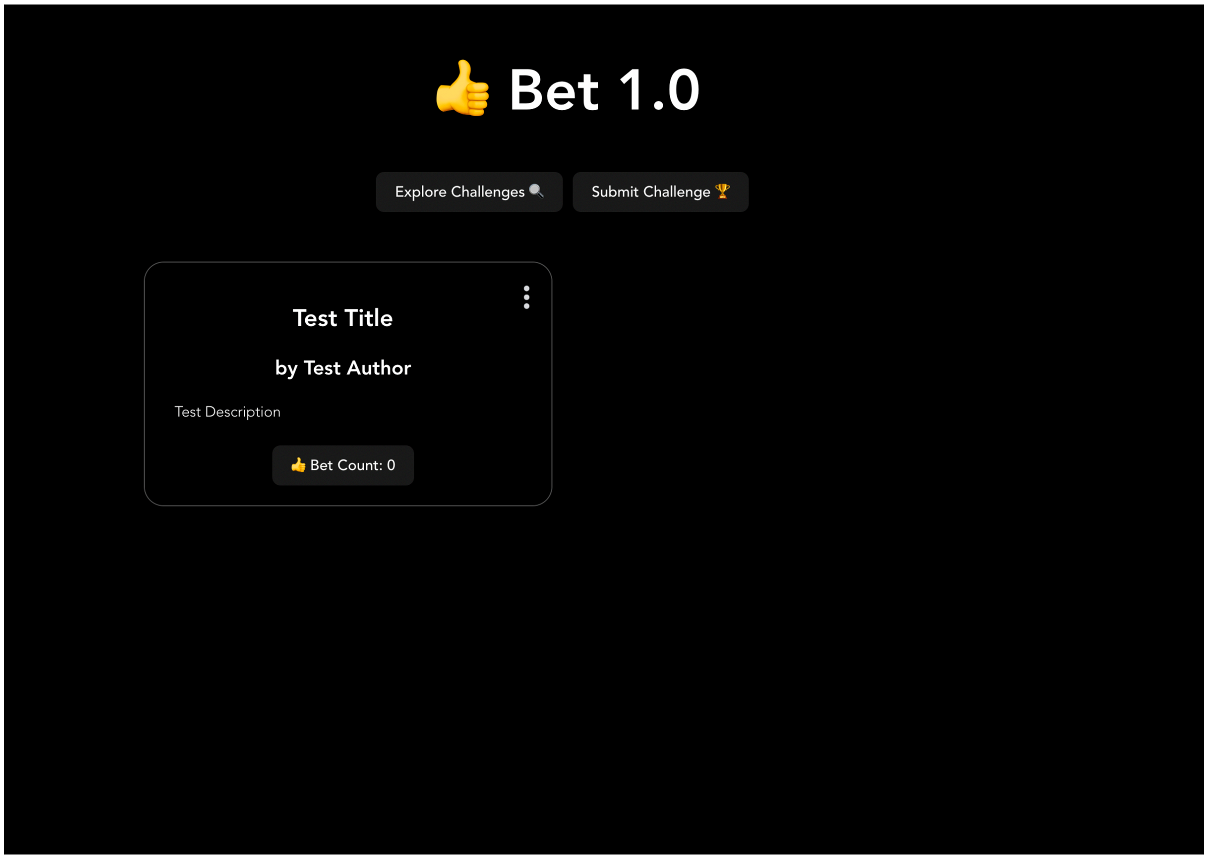
▶ *Use AI to understand provided code* → "Explain this" with Copilot

Your `fetchPost()` function should look like this:

```
// READ all post from table
const fetchPosts = async () => {
  const {data} = await supabase
    .from('Posts')
    .select();

  // set state of posts
  setPosts(data)
}
```

- [ ] At the end of the `useEffect()` function, call `fetchPosts()`.
- [ ] Let's test it! Refresh the page and you should see your test entry on the home page.

📍 **Checkpoint 5**: At this point in the lab, the your web application should look like this ⤵️



Step 6: Update Database Entries

Currently, when a user clicks the vertical three dot icon on any card in the homepage it redirects them to a form where they can update their submission. However, their changes are not updated in the database when they submit their changes. In this step, we will make a request to the database to update a given post and display the changes in the web application and database.

- [ ] Go to `EditPost.jsx` in the `pages` directory.

We will write this code to send the data to the database to update the entry and then redirect to the homepage.

- [ ] First, import `supabase` from `client.js`

```
import { supabase } from '../client'
```

- [ ] Define an asynchronous `updatePost()` function in the starter code. Add `event` as a parameter, because we will use this function to handle the form submission event when the user submits the Update Post form.

✨ AI Opportunity

▶ *Use AI as a pair programming partner* → Remembering syntax

▶ **Want to double-check your code? Compare what you have to this ⤵️**

- [ ] By default, the page will reload whenever the form is submitted. Add `event.preventDefault()` to your function to prevent this behavior.
- [ ] Add the following code snippet after `event.preventDefault()` in `updatePost()`

```
await supabase
  .from('Posts')
  .update({ title: post.title, author: post.author,  description: post.description})
  .eq('id', id)
```

Let's break down what this code snippet is doing:

- First, making calls to the Supabase client is an asynchronous operation so we need to include the `await` keyword.
- Next, we use the `.from()` method to specify which table in the project we want to perform an act on. In this case, we want to make a perform an operation "from" the `Posts` table.
- Next, we use the `.update()` method to indicate that we want to perform an update operation. To the `.update()` method, we pass a JSON with the `title`, `post`, and `description` from the post data from the form.
- Last, we apply a equivalent `.eq()` filter which will match only rows where column is equal to value. In this case, this will ensure that the row with the matching `id` of the current post is updated in the database.

✨ AI Opportunity

▶ *Use AI to understand provided code* → "Explain this" with Copilot

- [ ] Lastly, add `window.location = "/"` after the `await ...` code. This will redirect the browser to the root URL aka the homepage of our site.

Your `updatePost()` function should look like this:

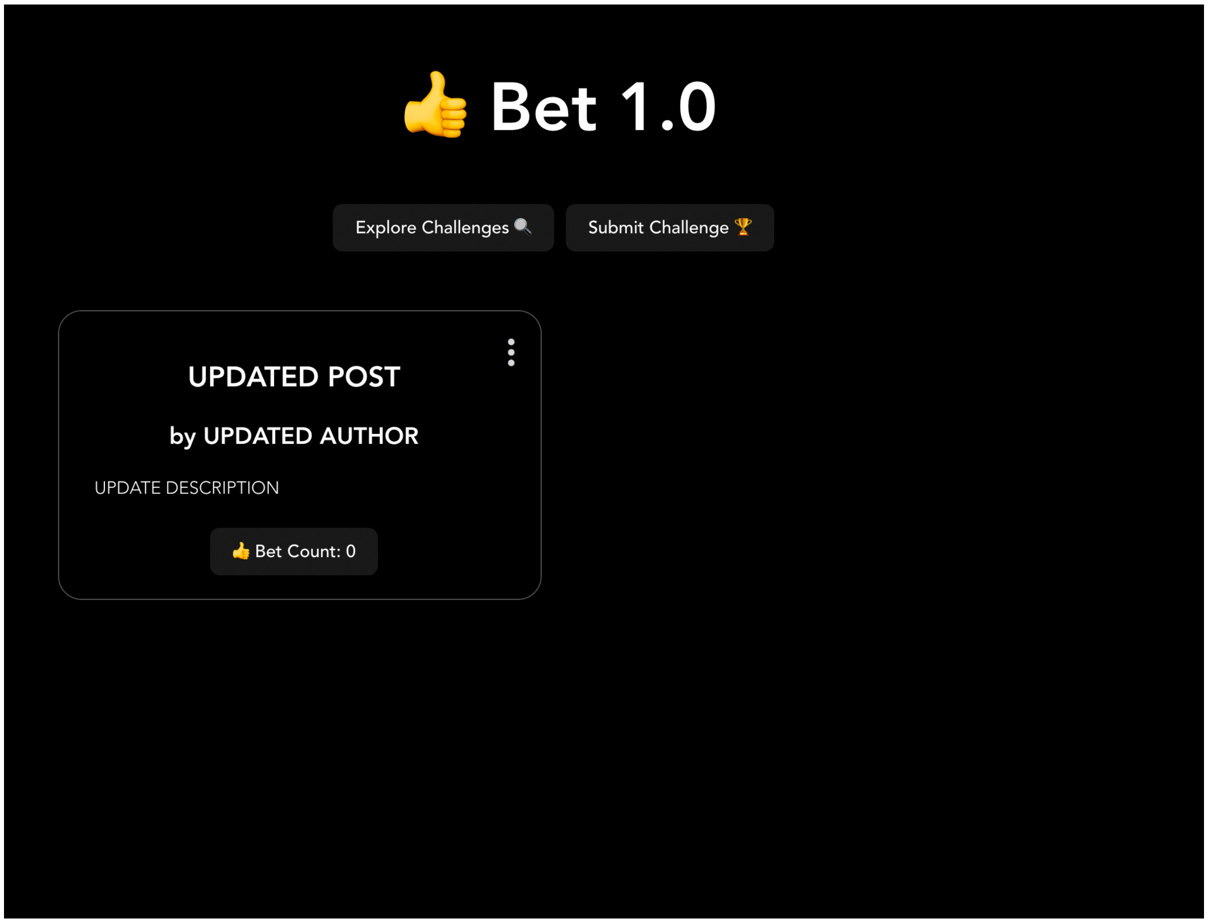
```
const updatePost = async (event) => {
  event.preventDefault();

  await supabase
    .from('Posts')
    .update({ title: post.title, author: post.author,  description: post.description})
    .eq('id', id);

  window.location = "/";
}
```

- [ ] Add an `onClick` event handler to the submit button to call `updatePost()`.

  ```
  <input type="submit" value="Submit" onClick={updatePost}/>
  ```

- [ ] Let's test it! Click the vertical three dot menu on one of the posts on the homepage. Use the form to change some or all of the post's data and click submit. On the Supabase website, check if the data entry was updated. On the homepage, the post should also reflect the updated changes.

📍 **Checkpoint 6**: At this point in the lab, the your Supabase table should look like this ⤵️



Step 7: Delete Database Entries

Currently, when a user clicks the delete button below the Update Post form, the Update Post page reloads, but nothing is deleted. In this step, we will make a request to the database to delete a given data entry from database and homepage.

- [ ] Go to `EditPost.jsx` in the `pages` directory.
- [ ] Define an asynchronous `deletePost()` function in the starter code. Add `event` as a parameter, because we will use this function to handle the event when the user clicks the delete button.
- [ ] By default, the page will reload whenever the delete button is clicked. Add `event.preventDefault()` to your function to prevent this behavior.

- [ ] Add the following code snippet after `event.preventDefault()` in the `deletePost()` function:

  ```
  await supabase
    .from('Posts')
    .delete()
    .eq('id', id)
  ```

Let's break down what this code snippet is doing:

- Just like with our previous three functions, making calls to the Supabase client is an asynchronous operation so we need to include the `await` keyword.
- Again, similar to our previous three functions, we use the `.from()` method to specify which table in the project we want to perform an act on. In this case, we want to make a perform an operation "from" the `Posts` table.
- Next, we use the `.delete()` method to indicate that we want to perform an delete operation.
- Last, we apply an equivalent filter, `.eq()` , which will match only rows where the column is equal to a specified value. In this case, we specify that only the row with an `id` that matches the `id` of the current post is deleted from the database.

✨ AI Opportunity

▶ *Use AI to understand provided code* → "Explain this" with Copilot

- [ ] Lastly, add `window.location = "/"` after the `await ...` code. This will redirect the browser to the root URL aka the homepage of our site.

Your `deletePost()` function should look like this:

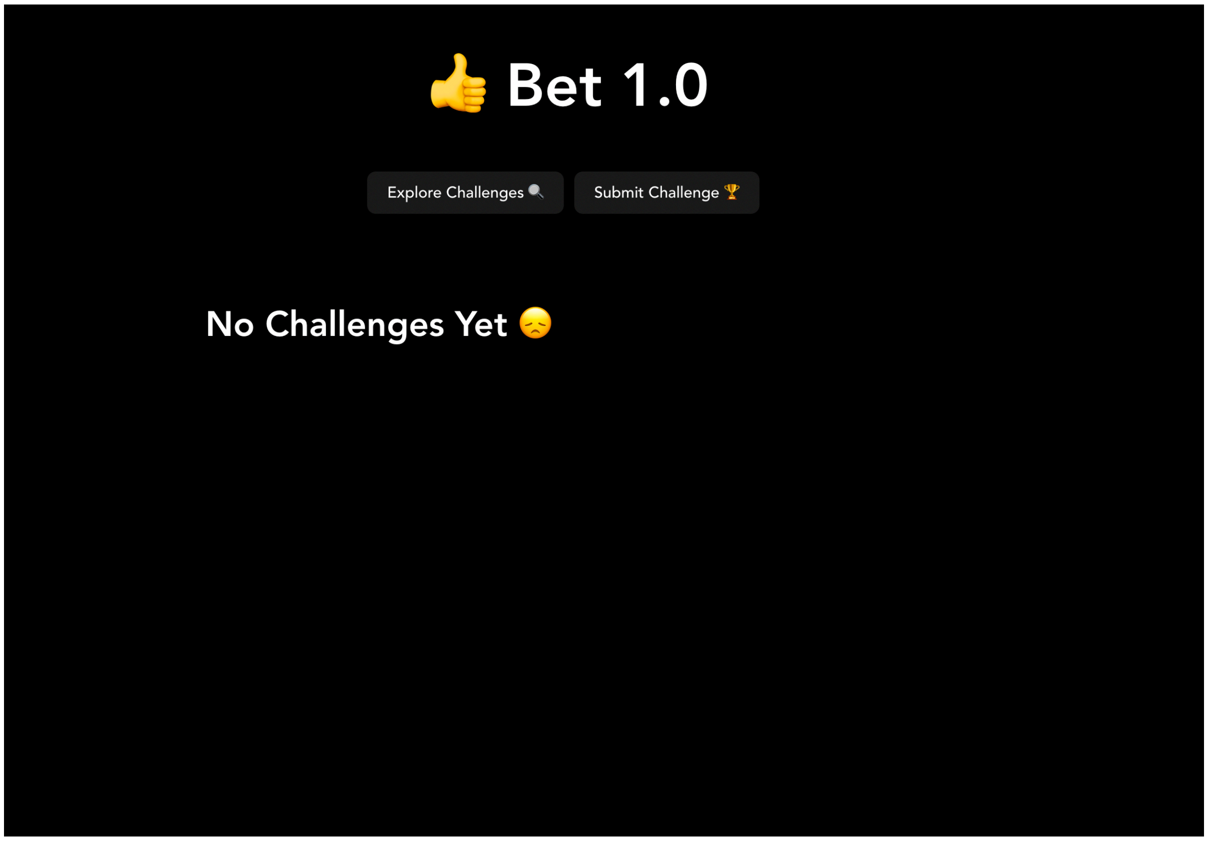
```
// UPDATE post
const deletePost = async (event) => {
  event.preventDefault();

  await supabase
    .from('Posts')
    .delete()
    .eq('id', id);

  window.location = "/";
}
```

- [ ] Add an `onClick` event handler to the delete button that calls `deletePost()`.

  ```
  <button className="deleteButton" onClick={deletePost}>Delete</button>
  ```

- [ ] Let's test it! Click the vertical three dot menu on one of the posts on the homepage. Click the delete button. On the Supabase website, check if the data entry was deleted. On the homepage, the post should no longer be displayed.

📍 **Checkpoint 7**: At this point in the lab, the your Supabase table should look like this ⤵️



🎉 Congratulations, you've successfully created a full-stack web application! 🎉

If you have time left over, continue on to the stretch features to customize and improve your app!

Stretch Features

Step 8: Save Bet Count to Database

In this step, we will update the `updateCount()` function so that when a user clicks the bet button the bet count is updated in and saved to the database.

- [ ] Add some sample posts to your database for testing.

- [ ] Go to `Card.jsx` in the `components` directory.
    - [ ] Import the `supabase` client from `client.js`
    - [ ] Locate the `updateCount()` function in the starter code.
    - [ ] Make `updateCount()` an asynchronous function by adding the `async` keyword and add `event` as a parameter.
    - [ ] Use `event.preventDefault()` to prevent the page from refreshing.
    - [ ] Inside the `updateCount()` function:
        - [ ] Make a call to the supabase client.
        - [ ] Use `from()` to perform the operations on the `"Posts"` table.
        - [ ] Use the `update()` function to update the `betCount` to `count + 1`
        - [ ] Apply an `eq()` filter to retrieve the row with the matching id of the current post.
        - [ ] Increment the `count` variable using `setCount()`.
- [ ] Go to the parent component `ReadPosts.jsx` in the `pages` directory where we render our Card component.
    - [ ] Pass the betCount value from each post in your Supabase `Posts` database as a prop to the Card component by adding `betCount={post.betCount}` when rendering `<Card />`.
- [ ] Go back to `Card.jsx`
    - [ ] Update your state initialization to `const [count, setCount] = useState(props.betCount)` so so that each card displays the actual bet count from the database rather than always starting at zero.

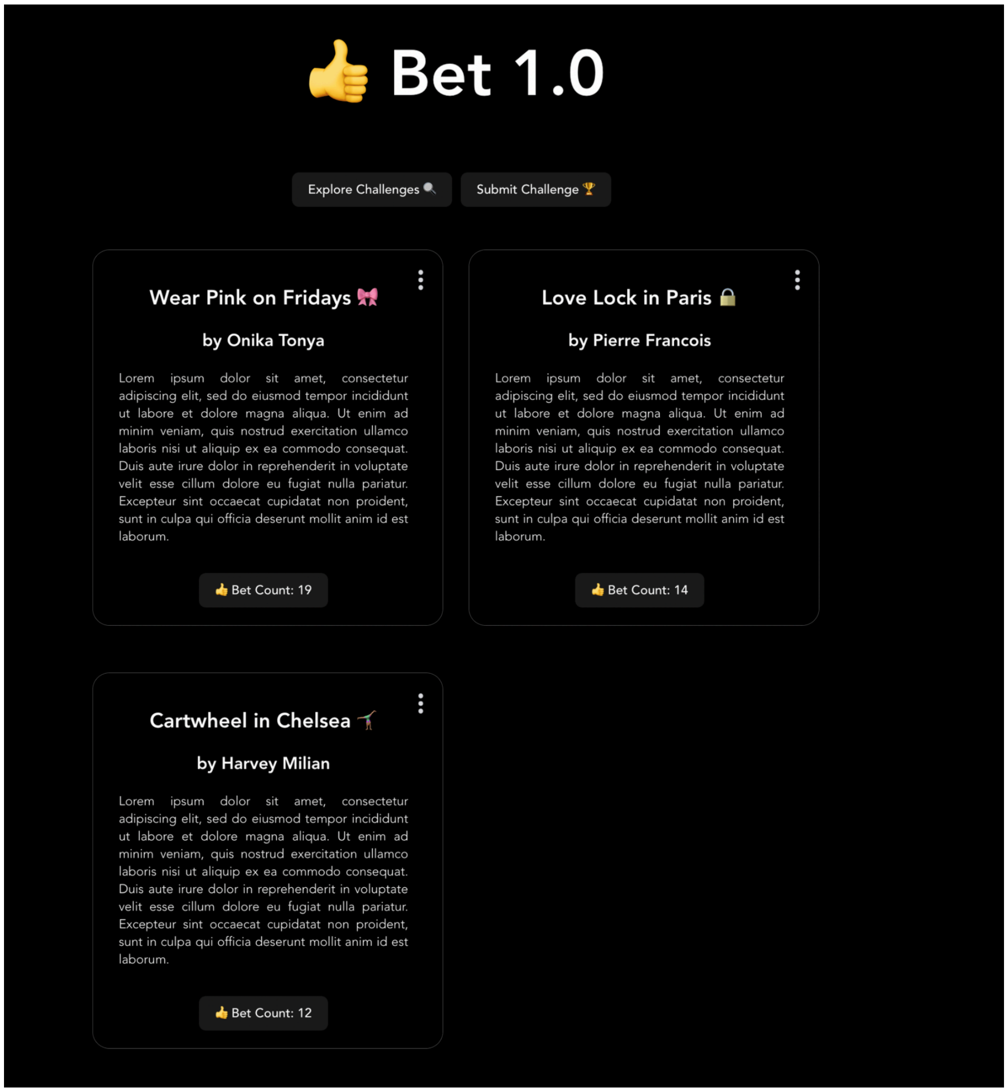
✨ AI Opportunity

▶ *Use AI to understand why your code isn't working →* `updateCount()`

▶ **Want to double-check your code? Compare what you have to this ⤵️**

- [ ] Let's test it! Click the Bet button on one of the cards. Refresh the page and you should see that the bet count remains the same on the web app. Also, the bet count should have updated in the database.

📍 **Checkpoint 8**: At this point in the lab, the your web application should look like this ⤵️



Wear Pink on Fridays 🎀

by Onika Tonya

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

👍 Bet Count: 19

Love Lock in Paris 🔒

by Pierre Francois

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

👍 Bet Count: 14

Cartwheel in Chelsea 🤸

by Harvey Milian

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

👍 Bet Count: 12

🎉 Congratulations 🎉

You've completed your lab AND stretch goals! 🚀

💡 **Tip:** Remember to come back and reference this lab when you need to do similar things in your project!