68. Preventing Too Many Votes

#graphql #urql #resolver #query #sql #backend #graphql-codegen #frontend #ssr #nextjs #cookie

---

- We don't want the user to be able to upvote or downvote more than once. Currently, even though the backend does not allow more thatn 1 up or downvote, the frontend updates the points everytime user clicks,. We will prevent that.

Add voteStatus Field to Post Entity

- We add a new @Field in the Post entity that will indicate whether the current user upvoted or downvoted on that post before
- Note that we don't define this as a database column with the typeORM's @Column attribute, but instead will use the posts query in the Post resolver to resolve it by pulling it from the updoot table's value column

/entities/Post.ts

```
@Field()
@Column({ type: "int", default: 0 })
points!: number;

//---- Added here:
@Field(() => Int, { nullable: true })
voteStatus!: number | null; // 1 or -1 or null
//-----

@Field()
@Column()
creatorId!: number;
```

Update posts() Query in post.ts Resolver to Return this New Field

- Ben used the SQL query to do this so I'm also switching to using the raw SQL query instead of typeORM's query builder
- For the voteStatus field, if there's an entry in updoot table, we return that value and we return null if not

resolvers/post.ts

```
@Query(() => PaginatedPosts)
async posts(
  @Arg("limit", () => Int) limit: number,
  @Arg("cursor", () => String, { nullable: true }) cursor: string | null
  @Ctx() { req } : MyContext
): Promise<PaginatedPosts> {
  //await sleep(3000); // simulate delay to test csr vs ssr load times

  const realLimit = Math.min(50, limit);
  const realLimitPlusOne = Math.min(50, limit) + 1;

  // Use SQL query to get the data from DB:

  const replacements: any[] = [realLimitPlusOne];

  if (req.session.userId) {
    replacements.push(req.session.userId);
  }

  let cursorIdx = 3
  if (cursor) {
    replacements.push(new Date(parseInt(cursor)));
    cursorIdx = replacements.length
  }

  const posts = await getConnection().query(
    `
  select p.*,
  json_build_object(
    'id', u.id,
    'username', u.username,
    'email', u.email
    ) creator,
  ${
    req.session.userId
      ? '(select value from updoot where "userId" = $2 and "postId" = p.id)
"voteStatus"'
      : 'null as "voteStatus"'
  }
      from post p
```

```
      inner join public.user u on u.id = p."creatorId"
      ${cursor ? `where p."createdAt" < $${cursorIdx}` : ""}
      order by p."createdAt" DESC
      limit $1
      `,
    replacements
  );

  return {
    posts: posts.slice(0, realLimit),
    hasMore: posts.length === realLimitPlusOne,
  }; // see if there's more posts to retrieve
}
```

Update PostSnippet fragment to retrieve this new voteStatus field from GraphQL

/fragments/PostSnippet.graphql

```
fragment PostSnippet on Post {
  id
  createdAt
  updatedAt
  title
  textSnippet
  points
  voteStatus
  creator {
    id
    username
  }
}
```

- Run codegen to regenerate the TypeScript types

```
yarn gen
```

Update UpdootSection Component

- We want to implement two features:
  - Prevent voting twice in the same direction if a vote was cast
  - Change button color if a vote was cast
- We update the UpdootSection component as follows, to implement these feautres using the voteStatus information:

/components/UpdootSection.tsx

```
<Flex direction="column" justifyContent="center" alignItems="center" mr={4}>
  <IconButton
    onClick={async () => {
      if (post.voteStatus === 1) {
       return;
      }
      setLoadingState("updoot-loading");
      await vote({
        postId: post.id,
        value: 1,
      });
      setLoadingState("not-loading");
    }}
    isLoading={loadingState === "updoot-loading"}
    boxSize={6}
    backgroundColor={post.voteStatus === 1 ? "teal.100" : ""}
    icon={<ChevronUpIcon />}
    aria-label={"Upvote post"}
  />
  {post.points}
  <IconButton
    onClick={async () => {
      if (post.voteStatus === -1) {
        return;
      }
      setLoadingState("downdoot-loading");
      await vote({
        postId: post.id,
        value: -1,
      });
      setLoadingState("not-loading");
    }}
    isLoading={loadingState === "downdoot-loading"}
```

```
    boxSize={6}
    backgroundColor={post.voteStatus === -1 ? "red.100" : ""}
    icon={<ChevronDownIcon />}
    aria-label={"Downvote post"}
  />
</Flex>
```

Update the urqlClient to utilize voteStatus

- So that we don't update the frontend points value correctly, and prevent updating it when voting is not allowed
- There's a `#bug` here that causes wrong points to be displayed if post already has votes from other people and then the user changes his vote Perhaps I'll come back to it later. It looks like it's related to new user login after a logout does not refresh the homepage (cast votes and button colors stay for old user)

utils/createUrqlClient.ts

```
vote: (result, args, cache, info) => {
  const { postId, value } = args as VoteMutationVariables;

  const data = cache.readFragment(
    gql`
      fragment _ on Post {
        id
        points
        voteStatus
      }
    `,
    { id: postId } as any
  );

  console.log("data: ", data);

  if (data) {
    if (data.voteStatus === value) {
      return;
    }

    const newPoints =
```

```
      (data.points as number) + (!data.voteStatus ? 1 : 2) * value;

    cache.writeFragment(
      gql`
        fragment __ on Post {
          points
          voteStatus
        }
      `,
      { id: postId, points: newPoints, voteStatus: value } as any
    );
  }
},
```

Why is it not working?

⚠️ **This is still not working correctly**. When we refresh the page the voteStatus value is not retrieved from the updoot table so the frontend logic does not work, votes can be cast multiple times, and the button colors do not work.

When we do a client-side request it automatically sends the cookie to the server. In our posts query in the post resolver we have this code:

```
${
  req.session.userId
    ? '(select value from updoot where "userId" = $2 and "postId" = p.id) "voteStatus"'
    : 'null as "voteStatus"'
}
```

which needs the userId to pull the voteStatus from updoot table.

The reason is that when we have a server side rendering, the request is first sent to NextJS server which in turn sends it to graphql api:

SSR: browser → next.js → graphql ip

When we have client side rendering the request is directly sent to the graphql api

CSR: browser→ graphql api

The browser sends the cookie with the request. So when it is SSR, the cookie is sent to next.js server which doesn't know what to do with it so the cookie is lost and then when the request is passed to graphql there's no cookie, and no userId information that can be read

When we create a new post, however, the request (and the cookie) is directly sent to graphql and then the posts query is executed again. This time the cookie is available in graphql api so the userId can be read and the posts query works as intended