86. Simpler Data Load - FieldResolver voteStatus()

#typeorm #graphql #resolver #fieldresolver #sql #query #apolloserver #context #dataloader #backend

---

Implement voteStatus FieldResolver

- Let's implement a FieldResolver for voteStatus much like we did with creator

/resolvers/post.ts

```
@FieldResolver(() => Int, { nullable: true })
async voteStatus(
  @Root() post: Post, // get called for Post objects
  @Ctx() { req, updootLoader }: MyContext
) {
  if (!req.session.userId) {
    return null;
  }

  const updoot = await updootLoader.load({
    postId: post.id,
    userId: req.session.userId,
  });

  return updoot ? updoot.value : null;
}
```

Implement new utility function createUpdootLoader()

- This function will take an array of `{ postId, userId }` objects and return an array of Updoot objects that match those ids, or null if not object is found

/utils/createUpdootLoader.**ts**

```
import DataLoader from "dataloader";
import { Updoot } from "../entities/Updoot";

// [{postId: 5, userId: 10}] ==> [voteStatus for that postId and userId]
export const createUpdootLoader = () =>
  new DataLoader<{ postId: number; userId: number }, Updoot | null>(
```

```
    async (keys) => {
      const updoots = await Updoot.findByIds(keys as any);
      // we don't directly return this since it could be out of order, and order
matters here

      const updootIdsToUpdoot: Record<string, Updoot> = {};
      updoots.forEach((updoot) => {
        updootIdsToUpdoot[`${updoot.userId}|${updoot.postId}`] = updoot;
      });

      return keys.map(
        (key) => updootIdsToUpdoot[`${key.userId}|${key.postId}`]
      );
    }
  );
```

Create a updootLoader in the apolloServer context

- Note that context will be run on every request, so a new updootLoader will be created on every request
- This updootLoader batches and caches the loading of voteStatuses into a single DB query

/index.ts

```
const apolloServer = new ApolloServer({
  schema: await buildSchema({
    resolvers: [HelloResolver, PostResolver, UserResolver],
    validate: false,
  }),
  context: ({ req, res }: MyContext) => ({
    req,
    res,
    redis,
    userLoader: createUserLoader(),
    updootLoader: createUpdootLoader(),
  }), // context is shared with all resolvers
});
```

- Also update the MyContext type to include updootLoader

/types.ts

```ts
export type MyContext = {
  req: ExtendedRequest;
  res: Response;
  redis: Redis;
  userLoader: ReturnType<typeof createUserLoader>;
  updootLoader: ReturnType<typeof createUpdootLoader>;
};
```