Report for Final Project template

Title of the Project: HN DB

Name of the students: Jacob Hummer

I. Introduction:

Motivation: why do you choose this problem and this domain? The importance of the application being implemented.
Describe the application.
Describe the overall organization of the report and task assignment for each team member

Making a Hacker News clone using my own DB store instead of using their DB store via API.
This application is a website that has a few views to read the data as well as some ways to create new submissions and comments. It's a very basic Hacker News clone. It supports logging in, creating submissions, upvoting submissions, creating comments, creating comments as replies to other comments, upvoting comments, viewing your profile, favoriting submissions, reporting submissions, sorting by top or by newest, degrading the scoring to penalize older submissions, paginating to see the next 30 submissions.

I don't really know what happened to the group, but I did this project on my own. I don't think that was the intent of the project, but it worked out OK. I learned about ORMs and how to use this new fancy Vercel tool that I wanted to learn about. This project served double duty that way!

II. Our Implementation

II.1 Description of the system architecture

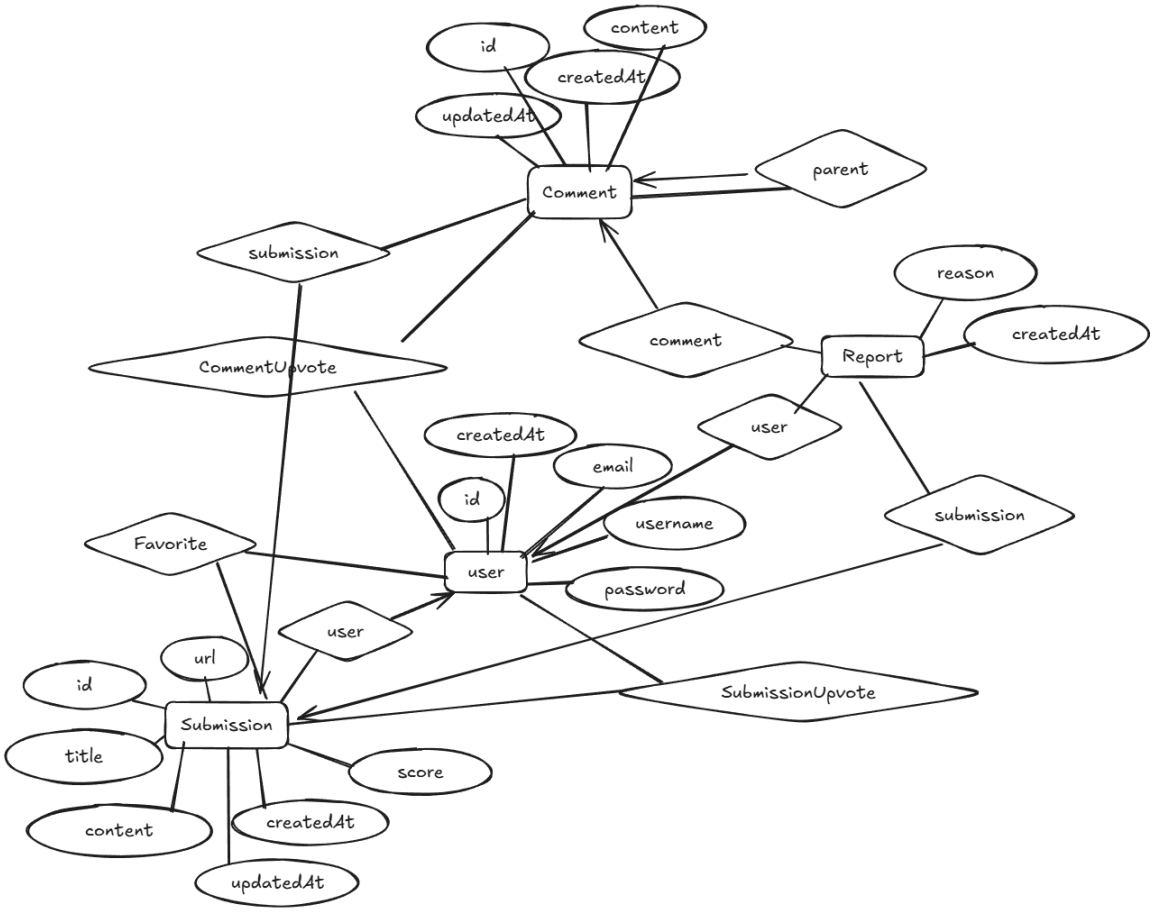
There's a database hosted on Vercel somewhere in US EAST that is running PostgreSQL. Then there's the application server which isn't really a server; it's serverless edge stuff. Those edge functions then render React components and fetch information from the PostgreSQL Vercel managed database in a way that's abstracted away from me, the developer. I just need to provide some Vercel PostgreSQL env vars and it connects automatically in dev (local machine) and prod (on Vercel's serverless). The project uses Next.js to render parts of the page on the server and do some other client-side interactions locally in the browser's DOM tree. It's all deliberately combined so that you can do things on the client and on the server with special "use server" or "use client" directives to direct the Next.js framework as to where your code wants to run. The database itself is not some kind of unmanaged raw SQL thing either; I'm using Prisma to define schema which is then applied to the database. Then there's also the seed data from hn_1.csv which is a large dataset of some Hacker News submissions. Prisma also lets me completely abstract the connection and SQL part of the database away. I just need to interact with a fluent JS API to do prisma.users.findFirst() or prisma.submission.findMany() and it runs the appropriate optimized SQL

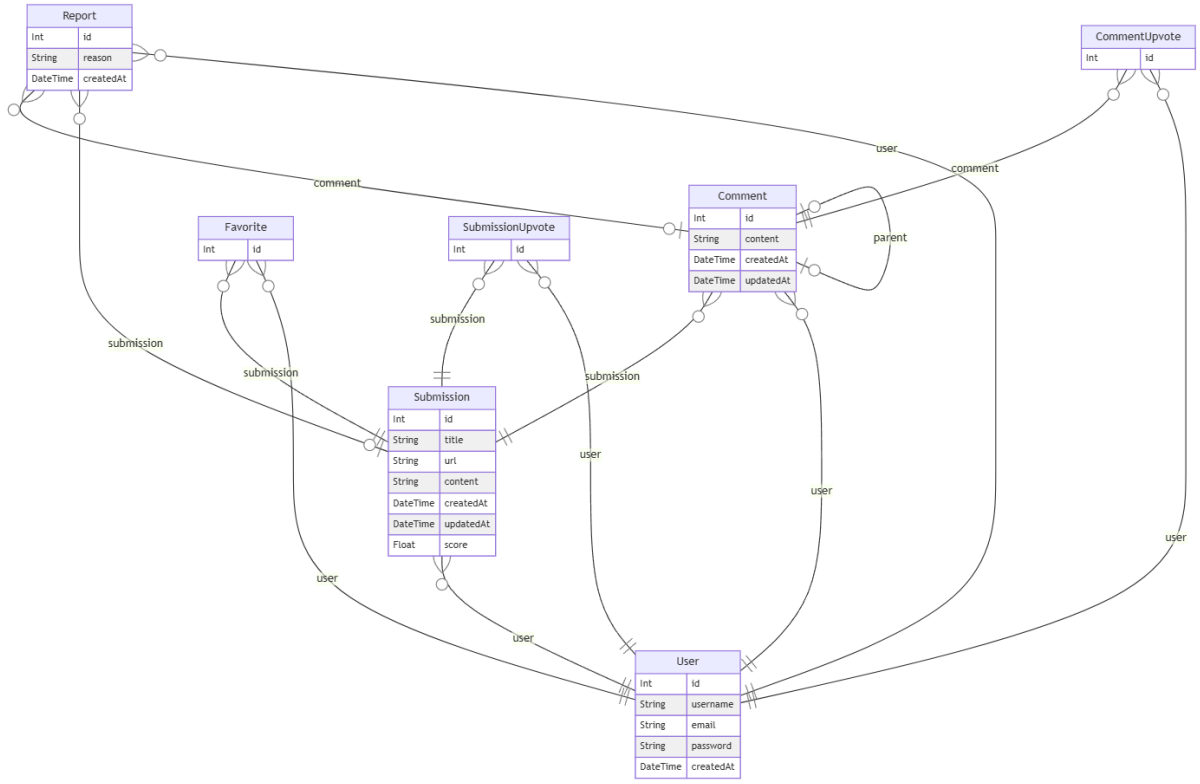
for me. There's also TailwindCSS to do some fancy styling so I don't have to write hardly any CSS; I just write all my CSS in React directly on the HTML components that I am defining.

II.2 Description of the dataset

The dataset that I used is very basic. It's just the author, a link, a number of comments that were there, and that's it. I seeded the database with a bunch of these. Then I created one comment under each submission so that the number of comments from the original dataset matches the number of comments in my HN clone. There are 13000+ comments total in the database.

II.3 ER diagram (final version from previous checkpoint copied here)


id
content
createdAt
updatedAt
Comment
parent
submission
reason
createdAt
comment
Report
CommentUpvote
user
createdAt
email
id
username
submission
Favorite
user
password
user
url
id
SubmissionUpvote
Submission
title
score
content
createdAt
updatedAt



II.4 Relational model (final version from previous checkpoint copied here)

User(id: Int)
Submission(id: Int, userId: Int)
Comment(id: Int, userId: Int, submissionId: Int)
Favorite(id: Int, userId: Int, submissionId: Int)
SubmissionUpvote(id: Int, userId: Int, submissionId: Int)
CommentUpvote(id: Int, userId: Int, commentId: Int)
Report(id: Int, userId: Int, submissionId: Int?, commentId: Int?)

All the “id” fields are the __ underlined keys. Everything else could in theory be duplicated but in practice isn’t at a SQL-schema level. Such constraints are enforced at other layers on top of the SQL like the Prisma layer or the JS code layer.

II.5 Implementation: description of the prototype

On Vercel. Using PostgreSQL. Using Prisma. Defined schema over time to accommodate each addition. The prototype supports creating submissions, creating comments, viewing top 30, pagination (manual still), viewing newest 30 with pagination, and more. See top where described list in detail.

II.6 Evaluation: describe how you test your application (e.g create testing scenarios or queries or something else, running your application through these scenarios/queries/etc.., checking the returned results and see how the results make sense or not).

III. Conclusion

What do you learn from this project (both interesting and uninteresting points)? Have you found any relevant database knowledge you have learned in this course helpful and have you encountered any database relevant issues that have been discussed in this course?

I learned a lot about Vercel and Next.js which is what I wanted to do. I also learned how to utilize their database hosted offering for practical projects. This website isn't some bespoke thing that only runs on a computer with some certain software on it; it uses standard industry tooling and will run on Vercel, a popular code hosting platform using Next.js and PostgreSQL, the #1 JavaScript full-stack framework and accompanying DB platform. It is easy to host your own. For free no less!

I also learned how helpful ORMs are. SQL doesn't have autocomplete and is very finicky. It's not easy to integrate raw SQL stuff into an application. There's no types. You have no idea what the return type of sqlResult.get("theField") is. There's no type safety. You have to remember it. If you mistake a string for a number you could run into a mismatched expected/actual type. Or errors. ORMs enforce a level of type-ness that really helps. They give autocomplete. It's really quite something. Prisma even lets you define your entire database in terms of a Prisma schema and then heps you manage that schema and push it to the database. It's sorta like Terraform syntax and does a similar thing, just specific to only databases, not AWS cloud stuff. It even helps you with an easy way to express array relations and even back-and-forth two-way relations. "prisma format" will auto-generate all the reverse properties to tie everything together once you do the one-way relations. It really is an all-in-one ORM and database setup tool. It even comes with a seed command where you seed your database in a script.

I did not find the knowledge about normal forms or relation syntax or composite keys any help *at all* which was kind of weird. Everything is just like defining classes. You write a "User" class. Prisma will then make sure all the relations and backreferences and tables and splits and such are defined. Then you write a Submission class and do the same "check it for me Prisma" again and it just works. You don't have to think about first normal form, 3rd normal form, whatever. Just write classes and it pretty much works.

I also did not find the SQL stuff that we learned much help. I could see how the Prisma queries mapped to SQL ideas like "JOIN" and "WHERE" but it was a new distinct API to do those operations via a JS API instead of raw SQL.

Basically what I'm saying is I learned a lot about the industry tooling and realized how cool it is that you don't even need to know *any theory at all* and you can still make a really cool application just using JS tooling and Prisma and other batteries-included tools.

- You don't need to know SQL; just use Prisma ORM

- You don’t need to know normal forms; just use Prisma schema
- You don’t need to know joins and stuff; just use the Prisma finder functions
- You don’t need to know how to create tables; just use Prisma db push
- You don’t need to know how to start a db server; just use Vercel PostgreSQL
- You don’t need to know how to pg SSH into a server; just use the Vercel dashboard

And most importantly: You don’t need to know how performance works; you can optimize your queries later. This is something I found true. I have 6000+ and 13000+ entries in two tables. Even some of my early totally inefficient query-then-use-result-to-query-again functions were very fast or even just *fast enough* that the network transmission time from the Vercel edge to the PG DB server was the time delay, not the query itself.

IV. References (if any)

https://nextjs.org/

https://www.prisma.io/

https://vercel.com/

gh repo https://github.com/jcbhmr/hndb

website https://hndb.vercel.app/

use login email “test1@example.com” with password “password”