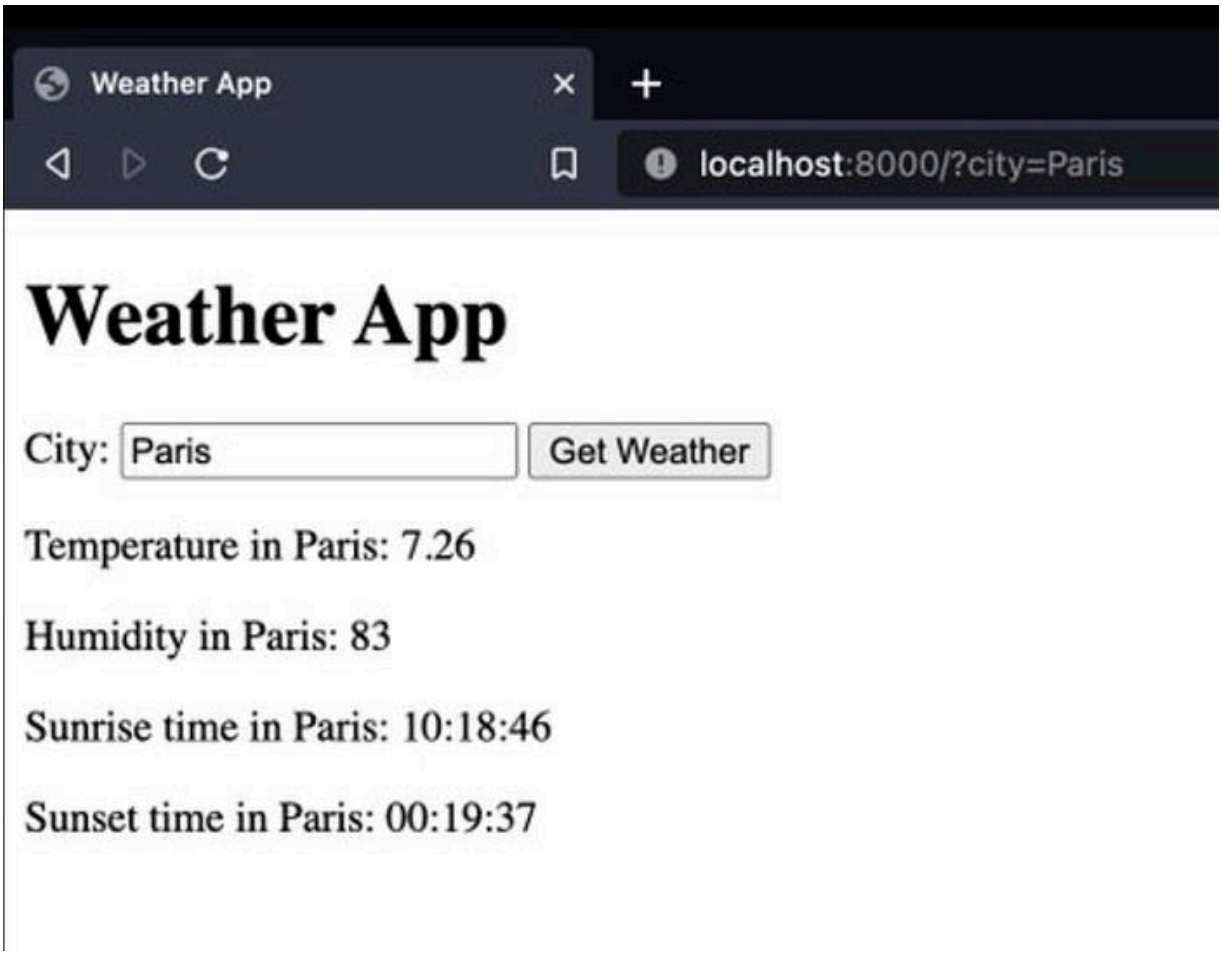
< Go to the original



Let us design this backend

Hussein Nasser

Follow

androidstudio · November 14, 2023 (Updated: November 14, 2023) · Free: Yes

This screenshot has become viral recently in Twitter, showing a very rudimentary frontend supposedly written by a backend engineer. But how would the backend actually look like for this weather app? I'll take a shot at the design.

Overview of the design

It will be very costly if the frontend calls the 3rd party API directly or even through the wrapped backend API.

We could have a database with a single table with few columns city, country, temperature, humidity, sunset, sunrise, date. indexed on the city. The table can be updated once daily using the 3rd party weather api. I can then expose an API for the frontend to query by city and that hits the table. From the UX, doesn't seem like history is needed, so current is fine.

Fleshing out the details

The textbox to write the city is error prone so I would hard code all cities in the frontend html and do a local search when user types to find the correct casing. This made me think that two cities may have the same name so probably better to use a unique id for each city and add that to the table, remove the index from city and add primary key on the id. This makes us avoid text search (which is a larger index) and not having to worry about casing.

The reason I'm going with clustered because we will almost always asking for all fields anyway, so a single btree seek to get the full row sounds better as opposed to scanning a secondary index then doing another IO to fetch the row from the heap (pg). So I think I'm going with MySQL for this one.

querying on page load. Especially when the app goes viral. we want to reduce load on the backend and db at the end.

The API will take the cityId, and return a JSON of the info by querying the table by the id. To make the API usable somewhere else other than this frontend (scope creep!) I can add a new route to return all cities ids and city name/country. Ill probably generate a JSON on the backend once knowing that will never change (and CDN it) as oppose to actually querying the db every time select id, city :

Updating Weather

Now we should have all what we need, to make the call and display it. But how about how the daily update will actually work? When we spin up the backend we need two threads, one thread can do the listening , connection acceptance , API request reading/response, and querying the db and writing the response. I think we can do almost all of that in one thread giving that most of this workload is async. The second thread sleeps for 24 hours and does one job, it fetches the 3rd party api all cities and starts a transaction and updates all rows in the table (makes it write back cache I guess?) let us call this thread the refresh thread

We need to be careful while scaling as I don't want multiple backends refresh threads racing to update the table, so I would acquire an application-level lock (known as advisory locks in pg) this way the second refresh thread fails when an existing refresh thread is working. Need to make sure this lock is very very early to avoid querying the 3rd party API.

Protocols

I don't see a need for using HTTP/2 or 3 as the frontend is very simple and doesn't send many requests, I rather save on backend CPU cycles/ memory by using vanilla HTTP/1.1 + TLS 1.3.

I think the Backend can be Node/Bun. I think a single instance can serve many requests with this design.

Connection Pooling

I expect queries to be fast, so we could setup a connection pool of min 5 max 10. Just so we don't overload the db if there is a rouge API calls. We can tweak as needed.

I would also set some timeouts to release resources early.

Sizing

I think the smallest MySQL and Node instance can do the trick as the design used as little cpu as possible. Most of the churn will happen during refreshing the table

#software-engineering