Node-Postgres

PostgreSQL client for node.js

♦ FULLSTACK 1 NODE-POSTGRES

postgres process



- The rDBMS itself; a daemon (background process)
- Waits for incoming SQL
- Knows how to read/write to disk in a performant way
- Sends back results

 ♦ FULLSTACK
 2
 NODE-POSTGRES

Where does the "incoming SQL" come from?

 ♦ FULLSTACK
 3
 NODE-POSTGRES

Query Sources ("Clients")

- psql CLI
 - human input as text
- GUI like Postico, Datazenit
 - human actions turned into SQL queries
- - "somehow" communicate with the postgres process

♦ FULLSTACK 4 NODE-POSTGRES

How to transmit SQL text to app?
How can postgres be "waiting for SQL"?
And how do the results get "sent back"?

 ♦ FULLSTACK
 5
 NODE-POSTGRES

Postgres is a TCP server!



- Listening on a TCP port (5432 by default) for requests
- Does disk access
- Sends back a TCP response to the client that made the requests

♦ FULLSTACK 6 NODE-POSTGRES

OK, Postgres is a TCP server. Is it... HTTP?

 ♦ FULLSTACK
 7
 NODE-POSTGRES

Postgres uses the postgres:// protocol

	Transport Protocol	Message Protocol	Content Type
Node + Express	TCP/IP	http://	Anything: HTML, JSON, XML, TXT, etc.
Postgres	TCP/IP	postgres://	SQL

♦ FULLSTACK 8 NODE-POSTGRES

For HTTP clients, the TCP/IP was handled for you by the browser or Node. How can our JS app communicate with the postgres server?

♦ FULLSTACK 9 NODE-POSTGRES

"Let's implement the postgres protocol in JavaScript ourselves!"

- AMBITIOUS MCOVERKILL

♦ FULLSTACK 10 NODE-POSTGRES



"On second thought... has anyone done this for us?"

- SANEY MCREASONABLE

♦ FULLSTACK 12 NODE-POSTGRES

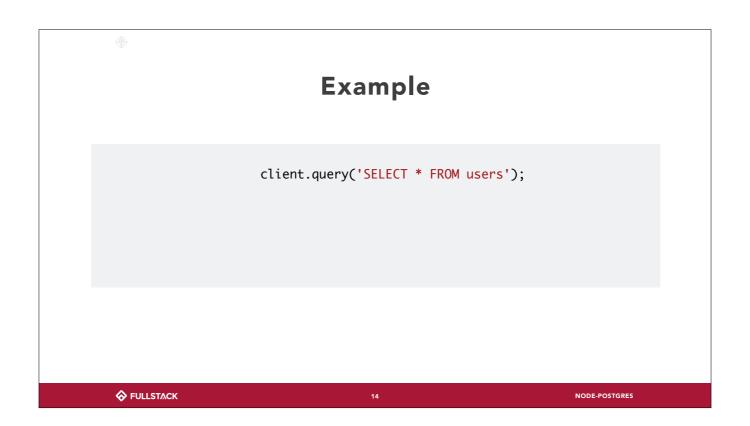
Node-postgres

- npm library: npm install pg --save
- database driver
- implements the postgres protocol in a Node module (JS!)
- Gives us a `client` object that we can pass SQL to
- Asynchronously talks via postgres protocol / TCP to postgres
- ø gives us a callback with `rows` array of resulting table

♦ FULLSTACK

13

NODE-POSTGRES



Here's the basic usage.

Client.query returns a promise, so...

Example const data = await client.query('SELECT * FROM users'); data.rows.forEach(function (rowObject) { console.log(rowObject); // { name: 'Claire' } }); ◊ FULLSTACK 15 NODE-POSTGRES

... We have to use the await keyword.

Then we can simply grab the data and loop over it.

try { const data = await client.query('SELECT * FROM users'); data.rows.forEach(function (rowObject) { console.log(rowObject); // { name: 'Claire' } }); } catch (err) { console.error(err); }

Remember that async operations can throw error - if the server is offline, for example, so wrap it in a try...catch block.