17. SQL from our Apps

Github repo

node-postgres

We are going to use node-postgres (`pg`) node package to interact with our database.

In order to connect with our database, we pass configuration options to the `pg` client:

```
const pg = require('pg');

const config = {
    user: '<user name>',
    password: '<password>',
    database: '<db>',
    host: '<host>'
};

const client = new pg.Client(config);
```

Then we tell our client to connect to the database and we execute queries using the client:

```
client.connect();
client
  .query('SELECT * FROM <table>')
  .then((result) => console.log(result));
```

NOTE: When using callbacks instead of promises: `pg` uses "error first" callbacks meaning that the first argument will always be the error (if any) or null and the second argument will be the return value from our query.

SQL Syntax Review

Browse

```
SELECT * FROM <table>;
```

Read

```
SELECT * FROM <table> WHERE id = <id>;
```

Edit

```
UPDATE <table> SET <column> = <value> WHERE id = <id>;
```

Add

```
INSERT INTO <table> (<column1>, <column2>) VALUES (<value1>, <value2>);
```

Delete

```
DELETE FROM <table> WHERE id = <id>;
```

Sanitization

We always want to sanitize any user-defined parameters in our SQL before running the query to prevent possible SQL injections.

In `pg`, we use prepared statements and pass an array of values as the second argument to `client.query()`:

```
client
  .query('SELECT * FROM <table> WHERE id = $1', [<id>])
  .then((result) => console.log(result));
```

In the above example, the `id` from the array will be interpolated into the SQL query wherever `$1` appears.

Protecting Secrets with Environment Variables

- We **NEVER** want to push keys/secrets to Github
- There are bots that crawl Github looking through repos for keys
- In order to protect our secrets, we want to inject them into our application at runtime (rather than storing them in variables inside our code)
- We use environment variables to accomplish this task
- Environment variables are specified when the application starts

```
# environment variables are specified before the application is started
PORT=3000 node server.js
# this PORT variable is accessible using process.env.PORT
```

- Or by using a package like `dotenv` to progammatically include them

```
npm i dotenv
```

```
# inside .env
PORT=3000
```

```
// inside server.js
require('dotenv').config();
```

Useful Links

- "SQL is demon spawn"
- node-postgres
- Postgres Numeric Data Types
- Little Bobby Tables
- SQL Injection

View on Compass | Leave Feedback