Environment Variables and Configuration

Managing Environments

- In our node apps, we've been committing and hard coding certain things:
  - Node modules
  - Express server port
  - Database name / port / username / password
- However, our applications will often need to run both locally on our machines, but also on a remote server
- This requires slightly different setups, as the remote server might be running a different operating system, name things differently, use different ports etc.
- Not to mention, we don't want to put our passwords online!

Technique #1: .gitignore

- Some of our files aren't meant to get saved in our git repository:
  - Temporary files (.DS_Store, *.tmp)
  - Third party code (node_modules)
  - Log files (npm-debug.log, *.log)
  - Anything that holds passwords or login credentials
- We tell git which files by making a .gitignore file
- Each line is a regular expression that matches files for git to ignore
- This means that in each environment, you'll be responsible for generating or creating these files somehow if they're required to run (i.e. node_modules)

Example .gitignore

```
# NPM directories
node_modules/
.npm/

# Logs
*.log

# Runtime data
*.pid
*.seed
*.pid.lock

# Environment Variables
.env
```

Technique #2: Process Environment Variables

- In Node, we've seen the `process` object before when calling `.exit()`
- It can do many things in addition to that, but right now we're interested in the `process.env` object
- This holds **environment variables**, things that are specific to the current environment
- We can set these in a few ways:
  - In javascript, by setting them
  - In the command line, when we run node
  - From a .env file, that's been .gitignored

process.env (Javascript)

- Since `process.env` is just an object, we can assign to it if we want
- This can be useful if you want to force certain environments, or dynamically set environment variables in a sophisticated way

```
process.env.DEBUG = true;

// ...elsewhere in your code

if (process.env.DEBUG) {
    console.log("Debug: Printing debug statement here");
}
```

process.env (Command line)

- We can set environment variables on the command line by doing KEY=VALUE
- All environment variables are strings when done this way, so don't try to set to "false" or "0", these will be truthy

```
# In the terminal
node app.js PORT=6000
```

```
// app.js: Get the port from environment variable, default to 3000
const port = process.env.PORT || 3000;

app.listen(port, function() {
    console.log("Listening on port " + port);
});
```

process.env (.env file)

- The most sophisticated way is to use a .env file for setting variables
- This is a list of KEY=VALUE statements, one per line, in a .gitignored file
- It does however require us to use an npm module

```
npm install --save dotenv
```

```
# .env
DB_USER=myuser
DB_NAME=my_app
DB_PASSWORD=hunter12
DB_HOST=localhost
DB_PORT=5432
```

process.env (.env file)

```
// Loads the .env file, and sets all its values to process.env
require("dotenv").config();
const pg = require("pg");

const config = {
    database: process.env.DB_NAME,
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    host: process.env.DB_HOST || "localhost",
    port: process.env.DB_PORT || 5432,
};

if (!config.database || !config.user || !config.password) {
    console.error("Missing database configuration:", config);
    process.exit(1);
}

const pool = new pg.Pool(config);
```

Exercise: Convert Postgres Config

Let's take one of your existing Postgres / Express project and make it environment agnostic

- Add a .gitignore that removes node_modules and the .env file
- Make sure you have all of the dependencies in your package.json file
- Add a .env file with all the configuration you'll need
- Replace all the spots you hardcoded ports, passwords etc. with `process.env`
- Add defaults where you can (ports), or error messages where you need configuration (passwords)
- Run your app *without* a .env file and make sure you get error messages
- Run your app *with* it and make sure it works!