Sequelize Quick Start Guide

Before you begin, make sure you've installed the mysql2 and
sequelize npm packages.

```json
{
    "name": "sequelize-example",
    "version": "1.0.0",
    "description": "",
    "main": "server.js",
    "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1",
        "start": "node server.js"
    },
    "keywords": [],
    "author": "",
    "license": "ISC",
    "dependencies": {
        "body-parser": "^1.15.2",
        "express": "^4.14.0",
        "mysql2": "^1.3.6",
        "sequelize": "^4.3.1"
    }
}
```

Also make sure you've globally installed the sequelize and sequelize-cli
npm package.

`npm install -g sequelize-cli sequelize`

Part One: Setting Up a Sequelize Project

While at the root of your project directory, run `sequelize init:config & sequelize init:models` (without the quotes) in your terminal. You should see similar output in the console.

```
[chriseckenrode:~/Desktop/temp/seqpract]$ sequelize init:config & sequelize init:models
[1] 69921

Sequelize CLI [Node: 6.11.0, CLI: 3.0.0, ORM: 4.11.3]

WARNING: This version of Sequelize CLI is not fully compatible with Sequelize v4. https://github.com/sequelize/cli#sequelize-support
Sequelize CLI [Node: 6.11.0, CLI: 3.0.0, ORM: 4.11.3]

WARNING: This version of Sequelize CLI is not fully compatible with Sequelize v4. https://github.com/sequelize/cli#sequelize-support

Successfully created models folder at "/Users/chriseckenrode/Desktop/temp/seqpract/models".
Created "config/config.json"
[1]  + 69921 done       sequelize init:config
[chriseckenrode:~/Desktop/temp/seqpract]$ ▎
```
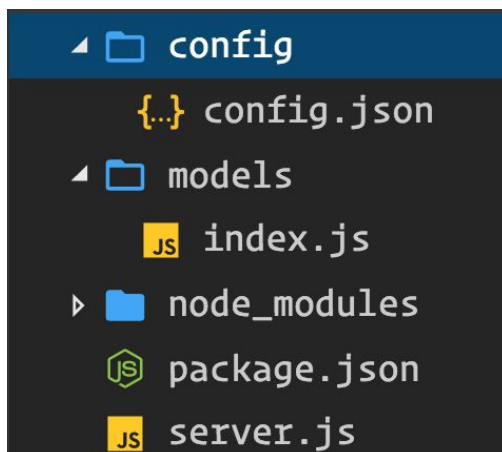
This should have created a few new files and folders for us.

- config/config.json
- models/index.js

Part Two: Connecting Our Database

Before we can set up sequelize to work with our database, we'll first need to first navigate to the config folder. Once inside, open the file `config.json`

```json
{
    "development": {
        "username": "root",
        "password": null,
        "database": "database_development",
        "host": "127.0.0.1",
        "dialect": "mysql"
    },
    "test": {
        "username": "root",
        "password": null,
        "database": "database_test",
        "host": "127.0.0.1",
        "dialect": "mysql"
    },
    "production": {
        "username": "root",
        "password": null,
        "database": "database_production",
        "host": "127.0.0.1",
        "dialect": "mysql"
    }
}
```

`config.json` stores JSON with details about the environment we'll be running our database in. To get started, alter the contents of the "development" object to match your local MYSQL database. If you already have a production database (a database you'll use when your app is deployed), go ahead and update those credentials in the "production" object. Otherwise go to the next step.

Part Three: index.js
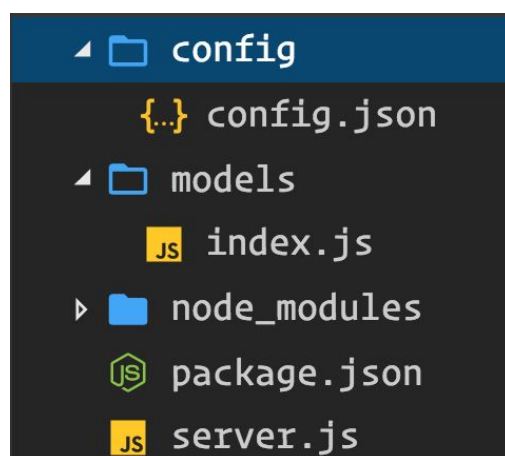
Next open up the models folder. Currently there should be a file inside named `index.js`. This directory is where we will define all of our sequelize models. Before we create our first model, however, here is an explanation of `index.js` and what it's doing for us. You may skip over this next part if you would like, it's not necessary to

completely understand, but for some it may help.

```js
'use strict';

var fs         = require('fs');
var path       = require('path');
var Sequelize  = require('sequelize');
var basename   = path.basename(module.filename);
var env        = process.env.NODE_ENV || 'development';
var config     = require(__dirname + '/../config/config.json')[env];
var db         = {};

if (config.use_env_variable) {
  var sequelize = new Sequelize(process.env[config.use_env_variable]);
} else {
  var sequelize = new Sequelize(config.database, config.username, config.password, config);
}

fs
  .readdirSync(__dirname)
  .filter(function(file) {
    return (file.indexOf('.') !== 0) && (file !== basename) && (file.slice(-3) === '.js');
  })
  .forEach(function(file) {
    var model = sequelize['import'](path.join(__dirname, file));
    db[model.name] = model;
  });

Object.keys(db).forEach(function(modelName) {
  if (db[modelName].associate) {
    db[modelName].associate(db);
  }
});

db.sequelize = sequelize;
db.Sequelize = Sequelize;

module.exports = db;
```

This index.js does a few key things that make setting our project up with Sequelize much easier.

First it figures out which database it should use based on whether we're deployed to heroku ("production") or running locally ("development") and will use the appropriate configuration inside config.json. We can also optionally specify a database to be used for testing if we want.
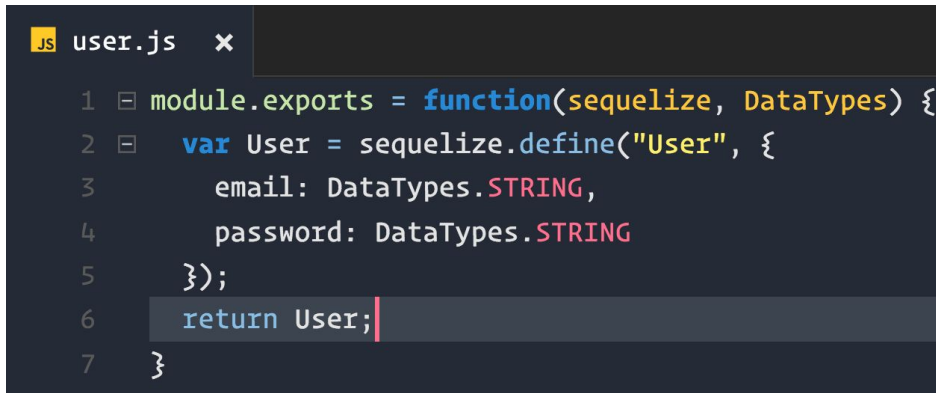
Then it goes through every other JavaScript file inside our models folder and runs them through Sequelize. It gives our models all of Sequelize's helper methods and makes sure that all of the associations between models are properly set up. It exports an object we will use to interface with Sequelize in our other files.

Important Note: On some versions of Windows, the slashes on line 8 may be backslashes ("\") instead of forward slashes ("/"). If you notice this, be sure to manually fix it before proceeding.

Part Four: Our First Model

Now let's create a new file inside of our models folder. As an example, we'll create a model for a User. Save it as user.js.

Inside this file add in the following code

```js
module.exports = function(sequelize, DataTypes) {
    var User = sequelize.define("User", {
        email: DataTypes.STRING,
        password: DataTypes.STRING
    });
    return User;
}
```

This is just about the minimum amount of code we need to write to create a Sequelize model. We export a function that takes in 2 variables. sequelize, and DataTypes. These are provided to us automatically by index.js.

"sequelize" in this case is actually our connection to our database.

DataTypes will be used to defining what type of data each property on our model should be.
http://docs.sequelizejs.com/en/latest/api/datatypes/#string

Inside of our function we run the "sequelize.define" method. We pass it two arguments. The name of our model as a string, and an object describing our model's schema. Each property will represent a column in the

database.

sequelize.define returns an object, which we store inside the variable "User". We return this variable at the end of the function on line 6.

Part Five: Syncing Our Models

In order to create tables from our models, we need to sync them with our database. To do this, we'll utilize the sequelize.sync() method.

Navigate to your server.js file and add the following code.

```
1   var express = require("express");
2   var db = require("./models");
3
4   var PORT = process.env.PORT || 3000;
5   var app = express();
6
7   db.sequelize.sync().then(function() {
8     app.listen(PORT, function() {
9       console.log("Listening on port %s", PORT);
10    });
11  });
```

On line 2 we require our entire models folder. This just requires the index.js file by default. We could have also done 'require('./models/index');' if we wanted.

On line 8 we sync our database. The sequelize property on the db object is actually our connection to our database. 'sync' is a built in sequelize method that creates tables using the models we describe. After our database is sync'ed (this may take a certain amount of time) we start our express server. Using this method, we guarantee that our server won't start before our database is ready. We also guarantee that our server won't start

if there's an error connecting to our database. To check if this worked, open your database in MYSQL Workbench or the SQL GUI of your choice to see if the tables were added
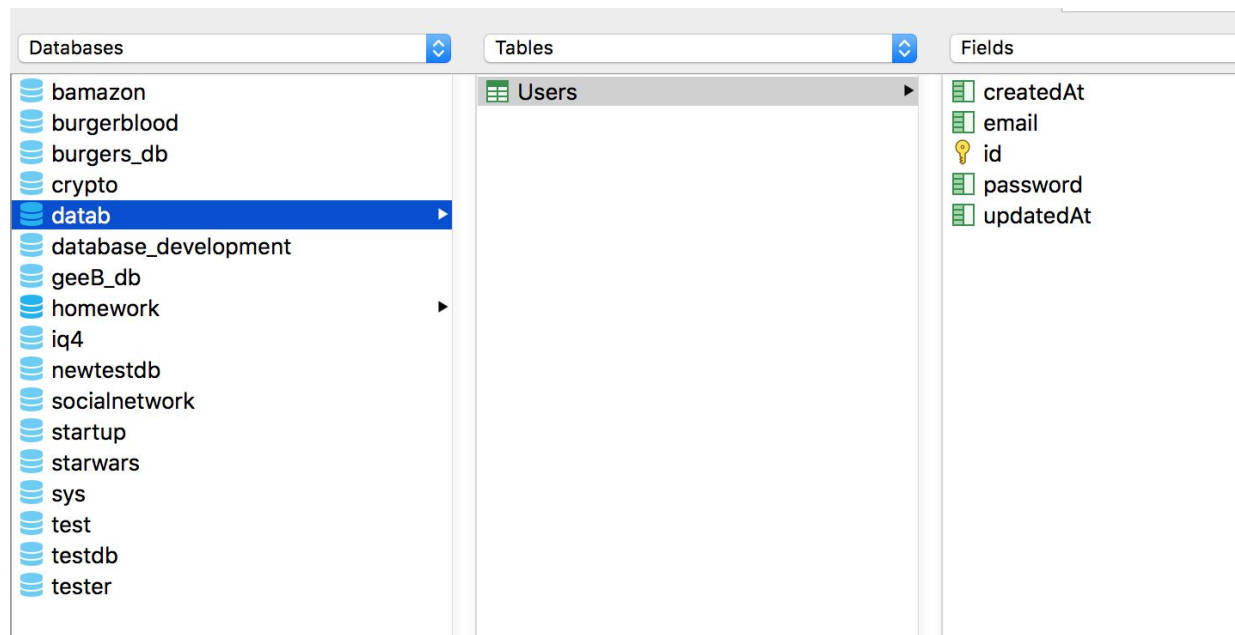
The information is displayed as follows:
Database > Tables > Columns

Sequelize will pluralize our table names by default, so always name your models in the singular.

Sequelize will also by default give us an auto-incrementing primary-key id, an updatedAt and a createdAt column. These are helpful if you need to do any kind of sorting by date when retrieving entries.

And that's all that's needed to connect set up a project with Sequelize.