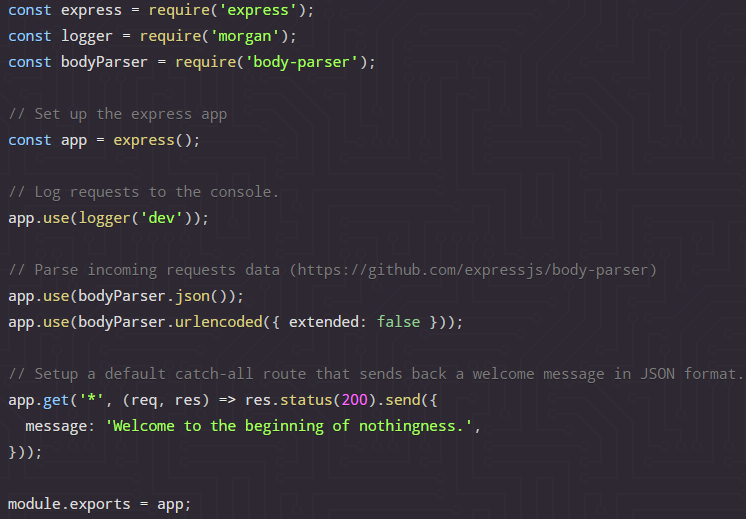
Setting Up an Express App  
https://scotch.io/tutorials/getting-started-with-node-express-and-postgres-using-sequelize

1. npm init -y
2. npm install --save express body-parser nodemon sequelize pg pg-hstore

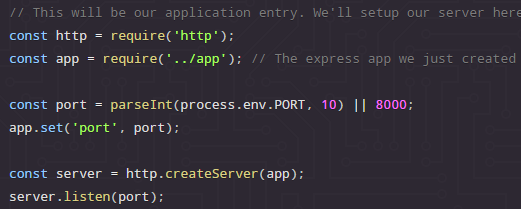
*pg will be responsible for creating the databse, while pg-hstore is a module for serializing/deserializing JSON data into the postgres hstore format*

npm install --save sequelize-cli -g

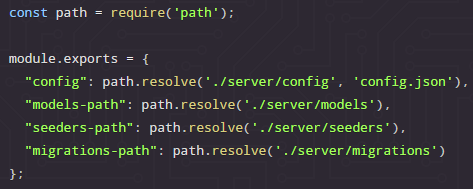
1. **Create the Express app** in a file called app.js in the root:



1. Create the application entry to set up the server in a separate file



1. npm start
2. **Configure Sequelize** by creating a file called .sequelizerc in the root. Here we specify the paths to the files that Sequelize will create when we run sequelize init in the next step.



**config.json** contains our application configuration settings, such as database authentication configuration. The **migrations** **folder** will hold our application's migrations, while the **models folder** will hold the application models. Seed data is initial data provided with a system for testing, training or templating purposes. The **seeders folder** typically holds seed data.

1. **sequelize init** to create the folders we just specified and generate the boilerplate code
2. Refactor boilerplate code from **server/models/index.js** from ES5 to ES6 (optional) and change back slashes in config to forward slashes
3. Update username, password, database and dialect in config.json



1. Create the database in the command line

"C:\Program Files\PostgreSQL\9.6\bin\createdb" -U postgres todos\_dev

*With our database and configuration in place, we’re now ready to generate models. In this example, We are going to have two models, Todo and TodoItem. The relationship between a Todo and it's TodoItems is going to be one-to-many, such that a Todo can have many TodoItems while a TodoItem can only belong to one Todo.*

1. In the command line, run:

sequelize model:create --name Todo --attributes title:string

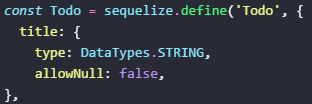
This generates two files with boilerplate code:

* models/todo.js In this file, we are defining our Todo model.   
   It’s going to have a single attribute, title, which is a string.
* migrations/<date>-create-todo.js

1. Now generate the TodoItem model:

sequelize model:create --name TodoItem --attributes content:string

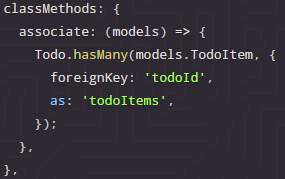
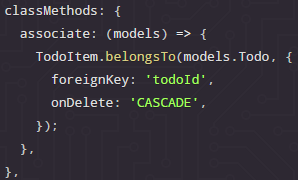
1. Refactor the generated files from ES5 to ES6 (optional) and add a not-null constraint. This means that the database will not allow us to write to it if we don't provide a value for the title field.



1. In the classMethods section of the generated model code, define the relationships between the models.

The**as: 'todoItems'** means that every time we query for a todo and include its todo items, they'll be included under the key todoItems instead of TodoItems (Sequelize defaults to using the model name). This change is just optional

Notice that we've edited both the content and complete fields. We've added a not-null constraint in the content field and a default value for the complete field. In general, having a default value means that if we don't provide a value for that field when creating it, the database is going to use the provided default value for that field. In addition to that, we've also defined the relationship between the TodoItems and the Todo objects. The onDelete: CASCADE tells Postgres that if we delete a todo, its associated todo items should be deleted as well (cascade the delete action).

1. Refactor migration files to ES6 (optional)

*Look in the migrations files. When we run these migrations, the up function will be executed. It will take care of creating the table and it's associated columns for us. If, for whatever reason, we needed to rollback (undo) the migration, the down function would be executed and it would undo whatever the up function did, thus returning the our database to the same state it was in before we performed the migration.*

*These migrations are a representation of how we want our models to look like in the database. Notice we define the relationship between our models in the create-todo-item.js migration file as well. The todoId field was not automatically generated and we've had to manually define it. Sequelize automatically generates the id, createdAt and updatedAt fields for you. In addition to that, any time a model is saved, the updatedAt field is automatically updated to reflect the new update time.*

1. With the models and migrations in place, we're now ready to persist the models to the database by running the migrations. To do this, we run the following command:

sequelize db:migrate

This will discover the migrations in our migrations folder and execute them. If you try running the same command again, it would not execute any migrations since it's clever enough to know that all of the current migrations have been executed.