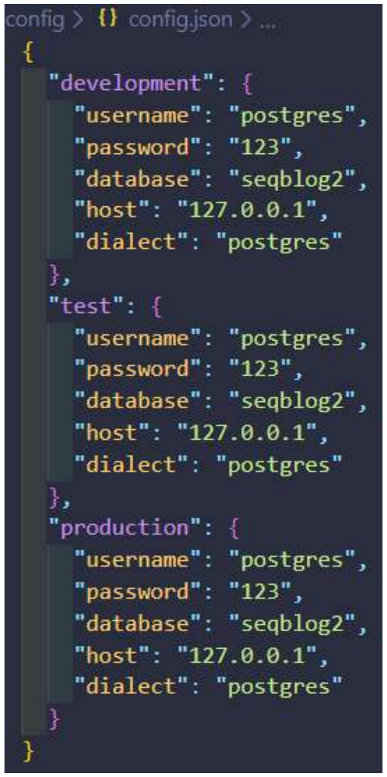
Authentication

Monday, March 16, 2020 3:58 PM

1. **Npm init**
2. **Npm install sequelize pg pg-hstore express ejs**
3. **Sequelize init**
4. **Create your .gitignore file in the root directory of the project**
   a. Run 'code .gitignore' in the terminal to open a text file
   b. Add 'node_modules/' without the quotes to your file and save
      i. This will prevent you from uploaded your nodemodules to Github

5. **Update config.json**
   a. Open config.json in the 'config' folder
   b. Update the file to match the following:

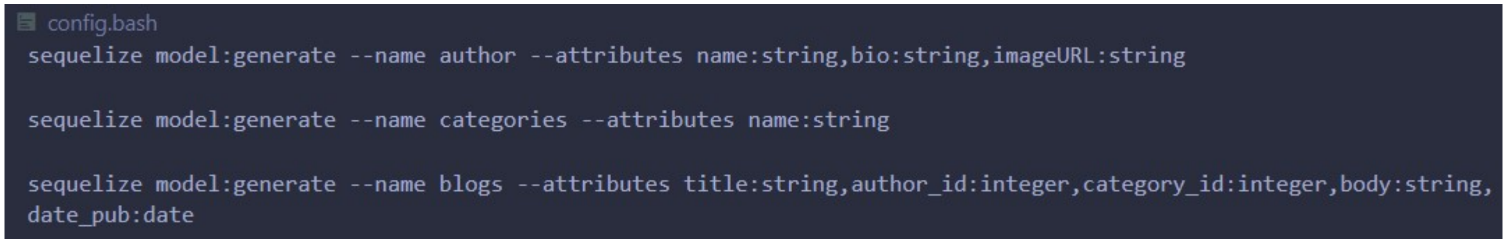


```json
{
  "development": {
    "username": "postgres",
    "password": "123",
    "database": "seqblog2",
    "host": "127.0.0.1",
    "dialect": "postgres"
  },
  "test": {
    "username": "postgres",
    "password": "123",
    "database": "seqblog2",
    "host": "127.0.0.1",
    "dialect": "postgres"
  },
  "production": {
    "username": "postgres",
    "password": "123",
    "database": "seqblog2",
    "host": "127.0.0.1",
    "dialect": "postgres"
  }
}
```

6. **Create your database in pgAdmin**
   a. Don't worry about create tables yet, just create the database as a whole

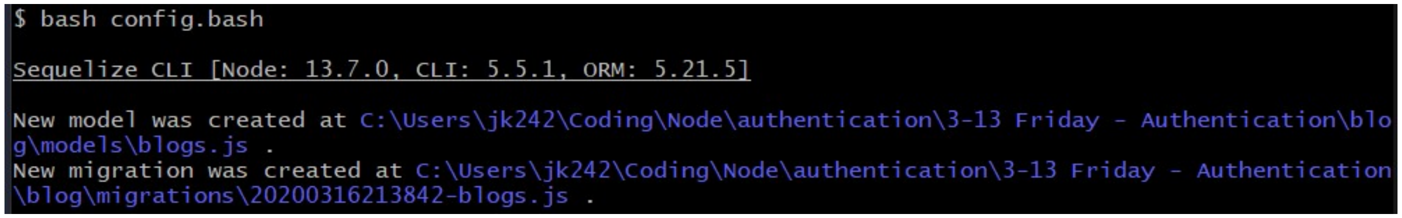
7. **Populate your tables - <u>Step 1</u>: create a bash file and use model:generate**
   a. Create a 'config.bash' file in your project root directory

i. Create your tables using the 'sequelize model:generate' commands in your bash file:



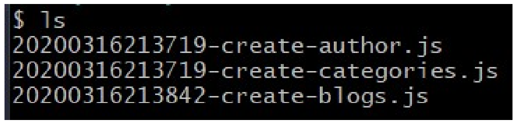
```
config.bash
sequelize model:generate --name author --attributes name:string,bio:string,imageURL:string

sequelize model:generate --name categories --attributes name:string

sequelize model:generate --name blogs --attributes title:string,author_id:integer,category_id:integer,body:string,
date_pub:date
```

b. Run your bash file
   i. In the terminal write: 'bash config.bash'
      1) You will get a notification that a new models and new migrations were created for each '--name' value from the config.bash file:



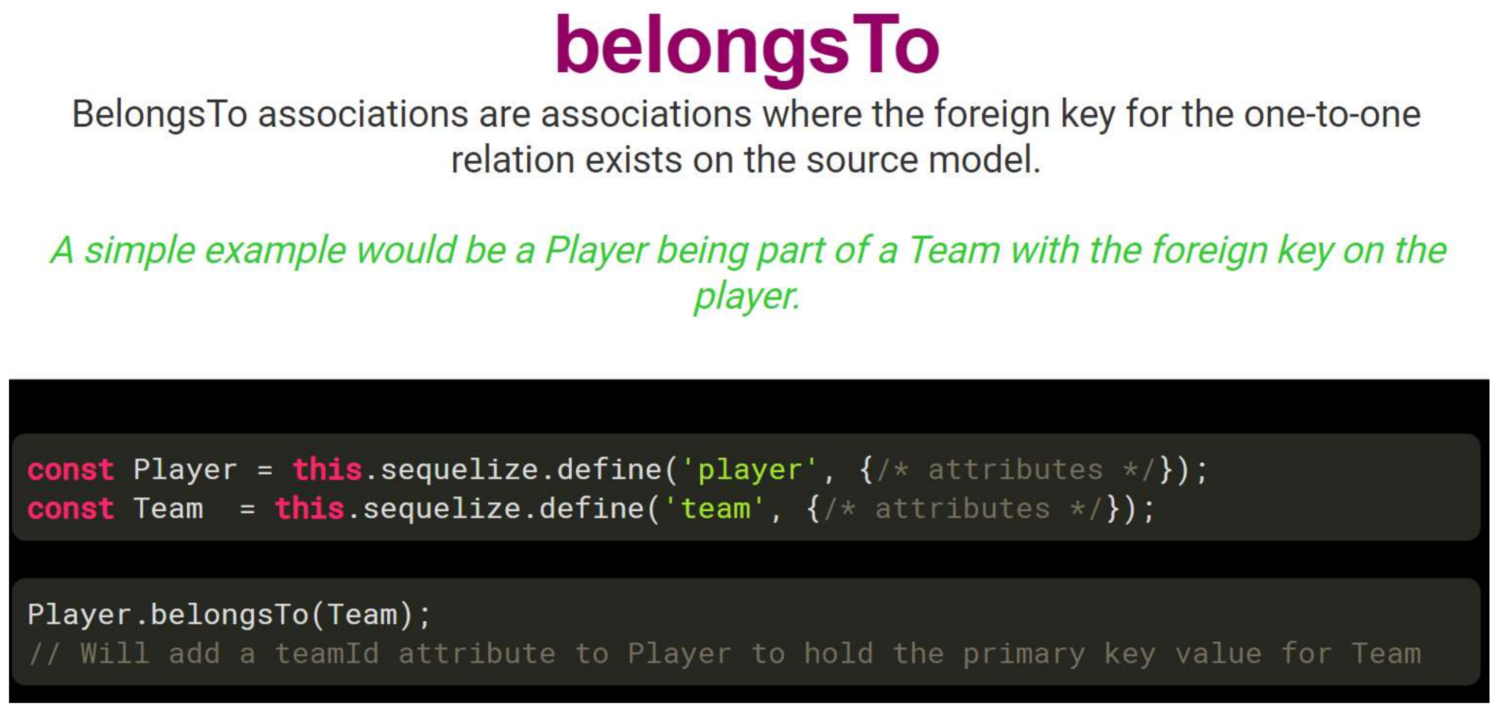
```
$ bash config.bash

Sequelize CLI [Node: 13.7.0, CLI: 5.5.1, ORM: 5.21.5]

New model was created at C:\Users\jk242\Coding\Node\authentication\3-13 Friday - Authentication\blo
g\models\blogs.js .
New migration was created at C:\Users\jk242\Coding\Node\authentication\3-13 Friday - Authentication
\blog\migrations\20200316213842-blogs.js .
```

         a) In the **'models'** folder, this will create a .js file for each '--name' we created in our bash file, along with an 'index.js' file by default
         b) In the **'migrations'** folder, this will create a corresponding .js file:



```
$ ls
20200316213719-create-author.js
20200316213719-create-categories.js
20200316213842-create-blogs.js
```

8. **Populate your tables - <u>Step 2</u>: set up any associations between your tables**
   a. Looks at the .js files in your 'models' folder
      i. Associations are defined in these .js files.
      ii. **belongsTo** are "associations where the foreign key for the one-to-one relation exists on the source model."
         1) An example is putting a foreign key on a "player" that points to his "team"



belongsTo

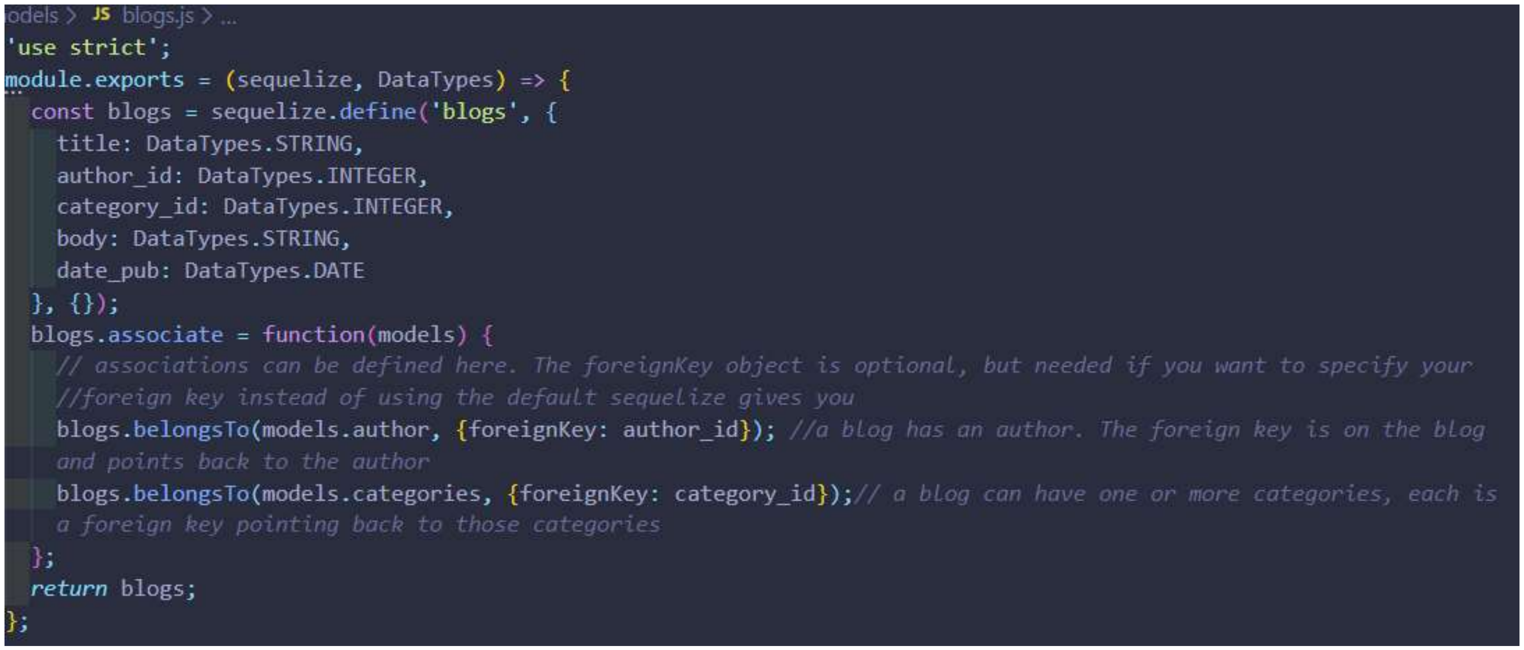
BelongsTo associations are associations where the foreign key for the one-to-one relation exists on the source model.

A simple example would be a Player being part of a Team with the foreign key on the player.

```
const Player = this.sequelize.define('player', {/* attributes */});
const Team  = this.sequelize.define('team', {/* attributes */});
```

```
Player.belongsTo(Team);
// Will add a teamId attribute to Player to hold the primary key value for Team
```

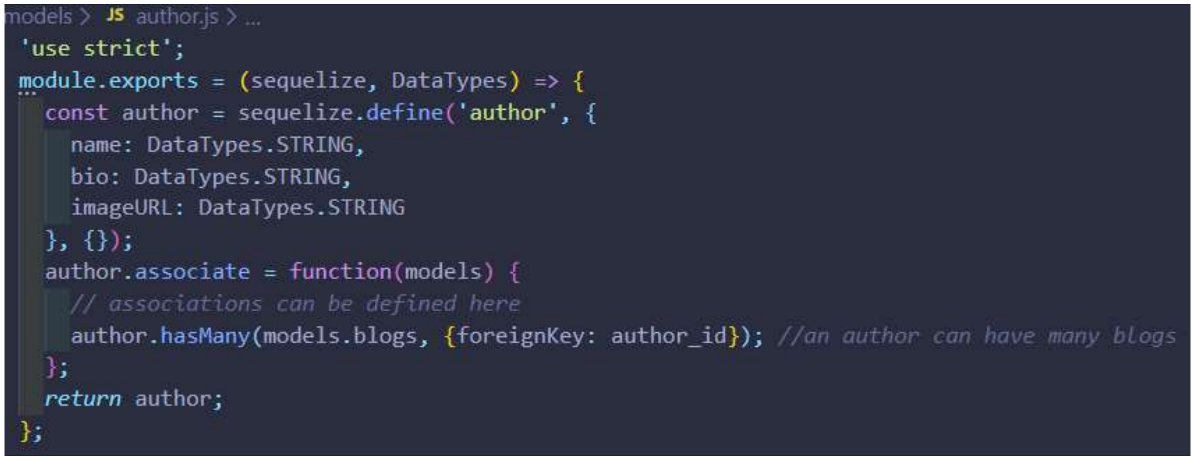
2) **Example** - we set our blogs to have a foreign key that points back to their author and what categories they belong to:



```
'use strict';
module.exports = (sequelize, DataTypes) => {
  const blogs = sequelize.define('blogs', {
    title: DataTypes.STRING,
    author_id: DataTypes.INTEGER,
    category_id: DataTypes.INTEGER,
    body: DataTypes.STRING,
    date_pub: DataTypes.DATE
  }, {});
  blogs.associate = function(models) {
    // associations can be defined here. The foreignKey object is optional, but needed if you want to specify your
    //foreign key instead of using the default sequelize gives you
    blogs.belongsTo(models.author, {foreignKey: author_id}); //a blog has an author. The foreign key is on the blog
    and points back to the author
    blogs.belongsTo(models.categories, {foreignKey: category_id});// a blog can have one or more categories, each is
    a foreign key pointing back to those categories
  };
  return blogs;
};
```

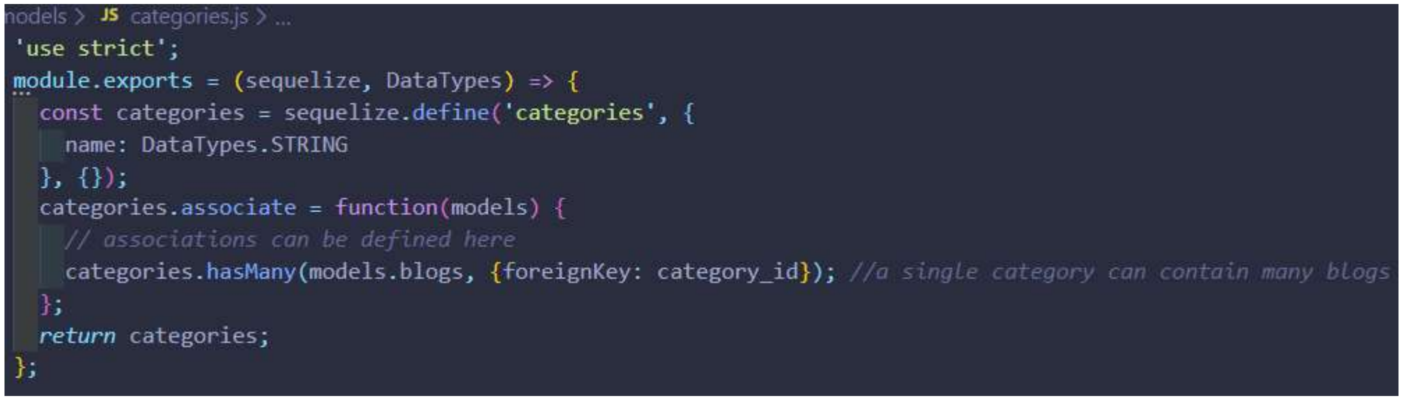
iii. **hasMany** are "One-To-Many associations are connecting one source with multiple targets. The targets however are again connected to exactly one specific source."

1) This is used to say something has many models that point back to it. For **example**, a single author might be the author of many different blogs. Each of the blogs has a foreign key that points back to her. **Notice:** we specified our own foreign keys, which is recommended:



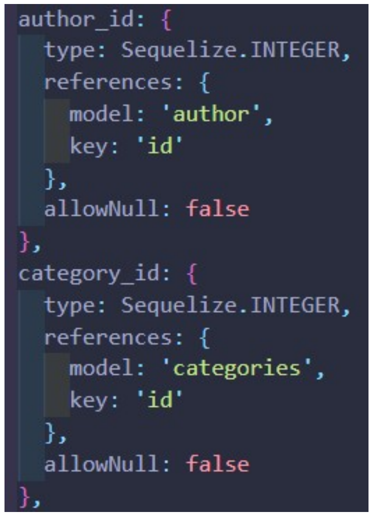
```
'use strict';
module.exports = (sequelize, DataTypes) => {
  const author = sequelize.define('author', {
    name: DataTypes.STRING,
    bio: DataTypes.STRING,
    imageURL: DataTypes.STRING
  }, {});
  author.associate = function(models) {
    // associations can be defined here
    author.hasMany(models.blogs, {foreignKey: author_id}); //an author can have many blogs
  };
  return author;
};
```

2) Another **example** - a category has many blogs in it:



```
nodels > JS categories.js > ...
'use strict';
module.exports = (sequelize, DataTypes) => {
  const categories = sequelize.define('categories', {
    name: DataTypes.STRING
  }, {});
  categories.associate = function(models) {
    // associations can be defined here
    categories.hasMany(models.blogs, {foreignKey: category_id}); //a single category can contain many blogs
  };
  return categories;
};
```

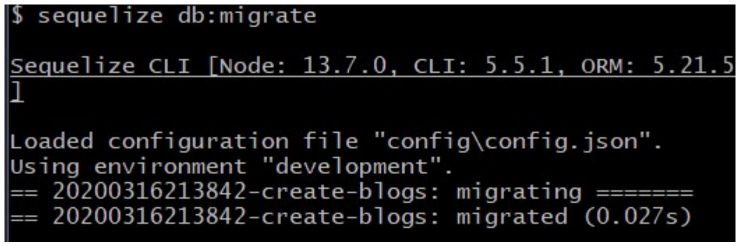
9. **Populate your tables - <u>Step 3</u>: set up your migration files in the 'models' folder**
   a. In the 'migrations' folder, look at the .js files that correspond to each table we want to create. These are tied to the .js files we have in the 'models' folder.
      i. In our models, wherever we declared a **.belongsTo**, we need to also make a change in its coressponding '-create-' .js file. **Below**, we changed author_id and category_id in the '#########-create-blogs.js' file because we set those up to be foreign keys in the 'blogs.js' model:



```
author_id: {
  type: Sequelize.INTEGER,
  references: {
    model: 'author',
    key: 'id'
  },
  allowNull: false
},
category_id: {
  type: Sequelize.INTEGER,
  references: {
    model: 'categories',
    key: 'id'
  },
  allowNull: false
},
```

NOTE: at this point we still have not created any tables, we have merely set everything up for them to *be* created.

10. **Create your table using 'sequelize db:migrate'**
    a. Navigate to the project root directory
    b. In the terminal, type 'sequelize db:migrate':



```
$ sequelize db:migrate

Sequelize CLI [Node: 13.7.0, CLI: 5.5.1, ORM: 5.21.5
]

Loaded configuration file "config\config.json".
Using environment "development".
== 20200316213842-create-blogs: migrating =======
== 20200316213842-create-blogs: migrated (0.027s)
```

c. **BOOM!** Your tables are now created!