1 ==== blog ======

'use strict';  
module.exports = (sequelize, DataTypes) => {  
 const blogs = sequelize.define('blogs', {  
 title: DataTypes.STRING,  
 body: DataTypes.STRING  
 }, {});  
 blogs.associate = function (models) {  
 // associations can be defined here blogs.belongsTo(models.author, { foreignKey: authorID });  
 blogs.belongsTo(models.categories, { foreignKey: categoryID }); };  
 return blogs;  
};

2 ==== categories ======

'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: categoryID });  
 };  
 return categories;  
};

3 ==== seeders: authors=======

'use strict';module.exports = {  
 up: (queryInterface, Sequelize) => { return queryInterface.bulkInsert('authors', [{  
 name: 'Austin',  
 bio: '',  
 imageURL: '',  
 createdAt: new Date(),  
 updatedAt: new Date()  
 },  
 {  
 name: 'Alina',  
 bio: '',  
 imageURL: '',  
 createdAt: new Date(),  
 updatedAt: new Date()  
 },  
 {  
 name: 'Jaye',  
 bio: '',  
 imageURL: '',  
 createdAt: new Date(),  
 updatedAt: new Date()  
 },  
 {  
 name: 'Daniel',  
 bio: '',  
 imageURL: '',  
 createdAt: new Date(),  
 updatedAt: new Date()  
 },  
 {  
 name: 'John',  
 bio: '',  
 imageURL: '',  
 createdAt: new Date(),  
 updatedAt: new Date()  
 } ], {}); }, down: (queryInterface, Sequelize) => {  
 /\*  
 Add reverting commands here.  
 Return a promise to correctly handle asynchronicity. Example:  
 return queryInterface.bulkDelete('People', null, {});  
 \*/  
 }  
};

4 ==== seeders: categories=======

'use strict';module.exports = {  
 up: (queryInterface, Sequelize) => { return queryInterface.bulkInsert('categories', [{  
 name: 'Javascript',  
 createdAt: new Date(),  
 updatedAt: new Date()  
 },  
 {  
 name: 'Node',  
 createdAt: new Date(),  
 updatedAt: new Date()  
 },  
 {  
 name: 'CSS',  
 createdAt: new Date(),  
 updatedAt: new Date()  
 },  
 {  
 name: 'HTML',  
 createdAt: new Date(),  
 updatedAt: new Date()  
 },  
 {  
 name: 'Bootstrap4',  
 createdAt: new Date(),  
 updatedAt: new Date()  
 },  
 {  
 name: 'APIs',  
 createdAt: new Date(),  
 updatedAt: new Date()  
 },  
 {  
 name: 'React',  
 createdAt: new Date(),  
 updatedAt: new Date()  
 }, ], {}); }, down: (queryInterface, Sequelize) => {  
 /\*  
 Add reverting commands here.  
 Return a promise to correctly handle asynchronicity. Example:  
 return queryInterface.bulkDelete('People', null, {});  
 \*/  
 }  
};

5 ==== migration: author =======

'use strict';  
module.exports = {  
 up: (queryInterface, Sequelize) => {  
 return queryInterface.createTable('authors', {  
 id: {  
 allowNull: false,  
 autoIncrement: true,  
 primaryKey: true,  
 type: Sequelize.INTEGER  
 },  
 name: {  
 type: Sequelize.STRING  
 },  
 bio: {  
 type: Sequelize.STRING  
 },  
 imageURL: {  
 type: Sequelize.STRING  
 },  
 createdAt: {  
 allowNull: false,  
 type: Sequelize.DATE  
 },  
 updatedAt: {  
 allowNull: false,  
 type: Sequelize.DATE  
 }  
 });  
 },  
 down: (queryInterface, Sequelize) => {  
 return queryInterface.dropTable('authors');  
 }  
};

6 ==== migration: categories=======

'use strict';  
module.exports = {  
 up: (queryInterface, Sequelize) => {  
 return queryInterface.createTable('categories', {  
 id: {  
 allowNull: false,  
 autoIncrement: true,  
 primaryKey: true,  
 type: Sequelize.INTEGER  
 },  
 name: {  
 type: Sequelize.STRING  
 },  
 createdAt: {  
 allowNull: false,  
 type: Sequelize.DATE  
 },  
 updatedAt: {  
 allowNull: false,  
 type: Sequelize.DATE  
 }  
 });  
 },  
 down: (queryInterface, Sequelize) => {  
 return queryInterface.dropTable('categories');  
 }  
};

7 ==== migration: blogs =======

'use strict';  
module.exports = {  
 up: (queryInterface, Sequelize) => {  
 return queryInterface.createTable('blogs', {  
 id: {  
 allowNull: false,  
 autoIncrement: true,  
 primaryKey: true,  
 type: Sequelize.INTEGER  
 },  
 title: {  
 type: Sequelize.STRING  
 },  
 body: {  
 type: Sequelize.STRING  
 },  
 authorID: {  
 type: Sequelize.INTEGER,  
 references: {  
 model: 'authors',  
 key: 'id'  
 },  
 allowNull: false  
 },  
 categoryID: {  
 type: Sequelize.INTEGER,  
 references: {  
 model: 'categories',  
 key: 'id'  
 },  
 allowNull: false  
 },  
 createdAt: {  
 allowNull: false,  
 type: Sequelize.DATE  
 },  
 updatedAt: {  
 allowNull: false,  
 type: Sequelize.DATE  
 }  
 });  
 },  
 down: (queryInterface, Sequelize) => {  
 return queryInterface.dropTable('blogs');  
 }  
};

8 ===========

const db = require('./models');// db.categories.findAll()  
// .then(results => {// results.forEach(record => {  
// console.log(record.name);  
// })// })db.blogs.create({  
 title: 'Javascript functions',  
 body: 'executing javascript functions',  
 categoryID: 1,  
 authorID: 1  
})  
 .then(blog => { console.log(blog.id);  
 })  
db.blogs.create({  
 title: 'goolge apis',  
 body: 'maps with google apis',  
 categoryID: 1,  
 authorID: 1  
})  
 .then(blog => { console.log(blog.id);  
 })  
db.blogs.create({  
 title: 'styling with bootstrap',  
 body: 'styling with bootstrap',  
 categoryID: 1,  
 authorID: 1  
})  
 .then(blog => { console.log(blog.id);  
 })  
db.blogs.create({  
 title: 'flexbox vs bootstrap vs css grid',  
 body: 'which one do you love',  
 categoryID: 1,  
 authorID: 1  
})  
 .then(blog => { console.log(blog.id);  
 })  
db.blogs.create({  
 title: 'travelling while coding in a pandemic',  
 body: 'I now have nightmares about coding',  
 categoryID: 1,  
 authorID: 1  
})  
 .then(blog => { console.log(blog.id);  
 })  
db.blogs.create({  
 title: 'Node callbacks',  
 body: 'Callbacks are the workhorse of Node',  
 categoryID: 1,  
 authorID: 1  
})  
 .then(blog => { console.log(blog.id);  
 })

9 ===========

const pbkdf2 = require('pbkdf2');  
const crypto = require('crypto');let password = "some password";let salt = crypto.randomBytes(20).toString('hex');  
// console.log(salt);let key = pbkdf2.pbkdf2Sync(password, salt, 3600, 256, 'sha256');let hash = key.toString('hex');// console.log(hash);//this will be stored in databaselet stored\_password = `pbkdf2\_sha256$3600$${salt}$${hash}`;// console.log(stored\_password);//1. validating user password//checking a password   
////////////////////////////////////////////////////////////////let login = "some password";let password\_parts = stored\_password.split('$');// ['some password', '3600', 'salt', 'hash'];// console.log(password\_parts[2]);  
// console.log(password\_parts[1]);  
let keyNewLogin = pbkdf2.pbkdf2Sync(  
 login,  
 password\_parts[2],  
 parseInt(password\_parts[1]),  
 256,  
 'sha256'  
)// console.log(password\_parts[3]);  
let hashNewLogin = keyNewLogin.toString('hex');if (hashNewLogin == password\_parts[3]) {  
 console.log('passwords matched');  
}  
else {  
 console.log('you suck, try again.');  
}