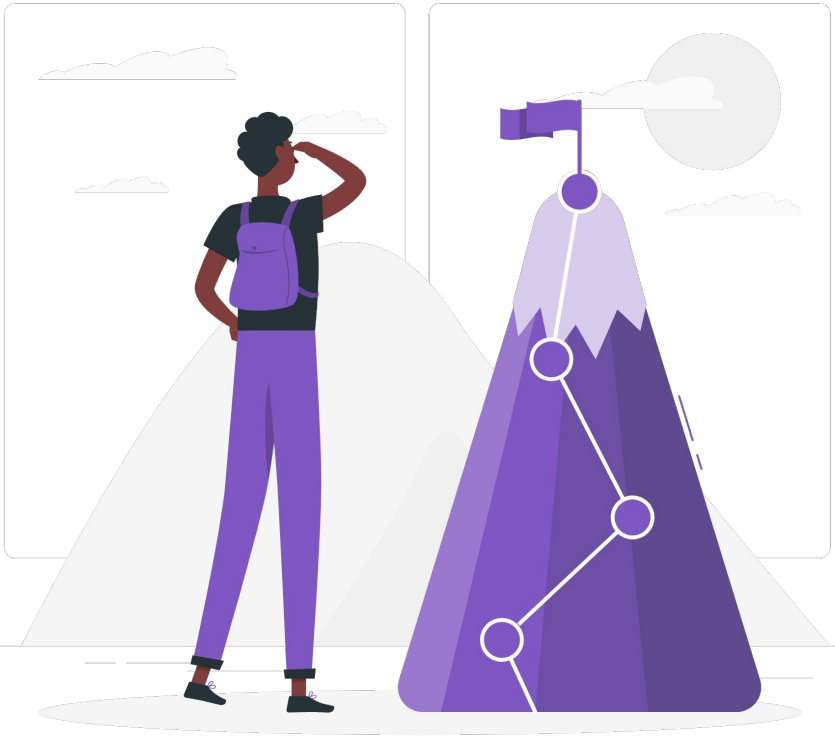
KNEX.JS

Intro a Knex

DEV.F::
DESARROLLAMOS(PERSONAS);

Elaborado por: César Guerra
www.cesarguerra.mx



Objetivos de la Sesión

- Aprender qué problemas resuelve Knex
- Aprender a instalar Knex en un proyecto
- Aprender a configurar Knex en un proyecto
- Conectar nuestro proyecto en NodeJS a una base de datos de PosgreSQL con Knex.
- Crear modelos con Knex y realizar migraciones.
- Crear nuestro servidor de express

DEV.F::

https://knexjs.org/

¿Qué es Knex?

Knex.js es un constructor de consultas SQL para JavaScript, una abstracción delgada que se encuentra en la parte superior del controlador de la base de datos para bases de datos relacionales como PostgreSQL, MySQL, SQLite2 y Oracle.

Usando Knex JS

DEV.F::
DESARROLLAMOS(PERSONAS);

1. Instalación

Realizamos la instalación de knex desde la línea de comandos, de manera global

DEV.F

Nota: Recordemos que -y nos sirve para qué nos pregunte todos los detalles de nuestro proyecto y cree rápidamente el archivo package.json de nuestro proyecto.

2. Creamos nuestro proyecto

Creamos la carpeta de nuestro proyecto (mk nombreproyecto)

Entramos a la carpeta con ayuda del comando cd (cd nombreproyecto)

Creamos un nuevo proyecto de node.js con ayuda de npm init.

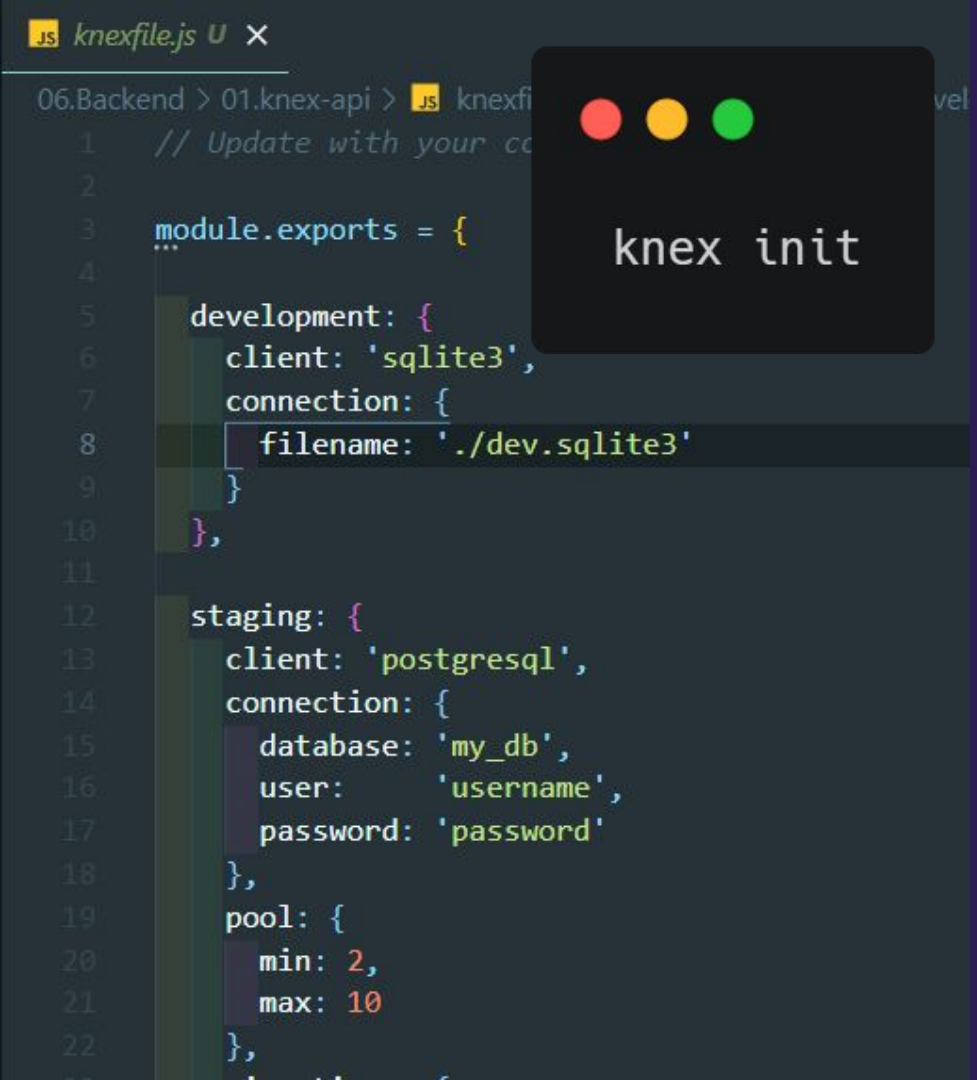
3. Instalar dependencias

Dentro de la carpeta del proyecto, instalaremos las dependencias necesarias.

En este caso:

- express
- knex
- pg

DEV.F::

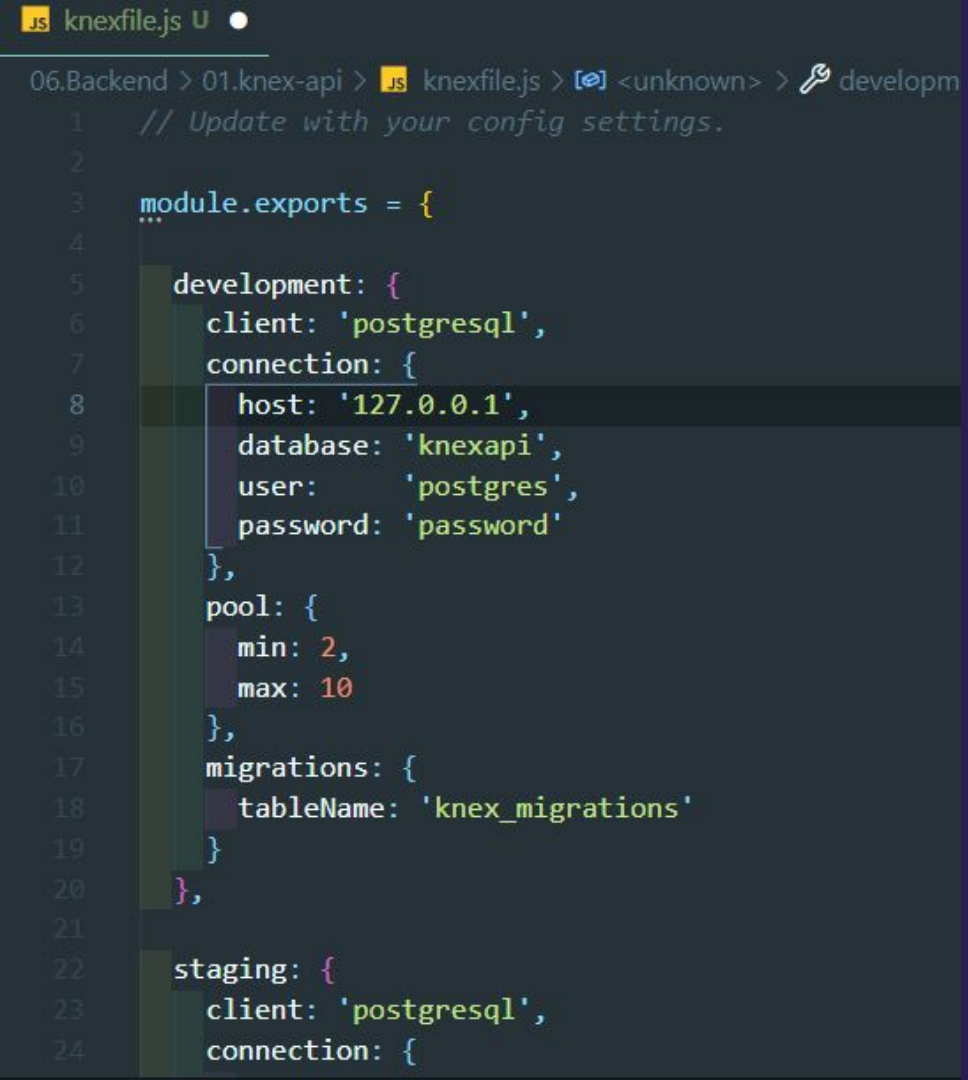


4. Inicializar knex

Ejecutar en la terminal el comando **knex init** nos ayudará a inicializar un archivo de configuración de knex para nuestro proyecto.

Este creará el archivo **knexfile.js** en la raíz de nuestro proyecto, donde tendremos que configurar manualmente la conexión y configuración de la base de datos para los diferentes entornos.

DEV.F::



5. Configurar los datos de conexión

Knex nos permite trabajar con bases de datos para diferentes entornos, nos aseguraremos de que el entorno de development tenga la configuración adecuada.

Nos aseguraremos de configurar la conexión de development apuntando a nuestra instancia local de postgreSQL.

DEV.F::

DEV.F:

ASEGURARNOS QUÉ LA BASE DE DATOS ESTE CREADA, EN CASO CONTRARIO ***CREARLA EN PGADMIN***

Migraciones

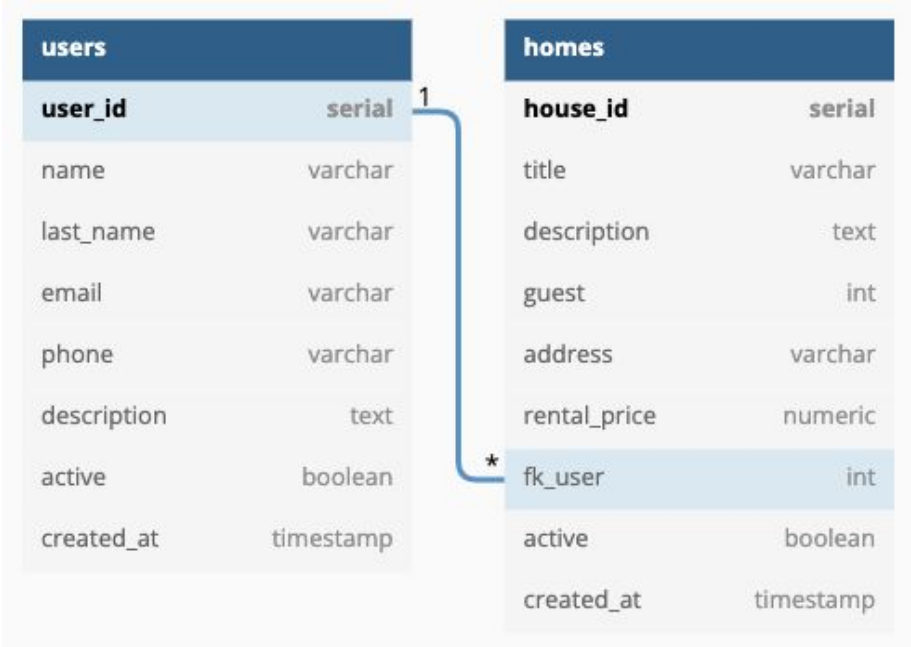
Las migraciones son un control de versiones de nuestra base de datos, pero en realidad son más que eso.

Este nos permite crear tablas, establecer relaciones, modificarlas y por supuesto eliminarlas, y todo esto con comandos y de manera programacional en vez de directamente hacerlo en la base de datos.

exports.up: Es lo que la migración creará según lo que configuremos (tablas, campos, cambiar tipos de campos, etc.)

exports.down: Definimos como revertir la migración realizada por exports.up.

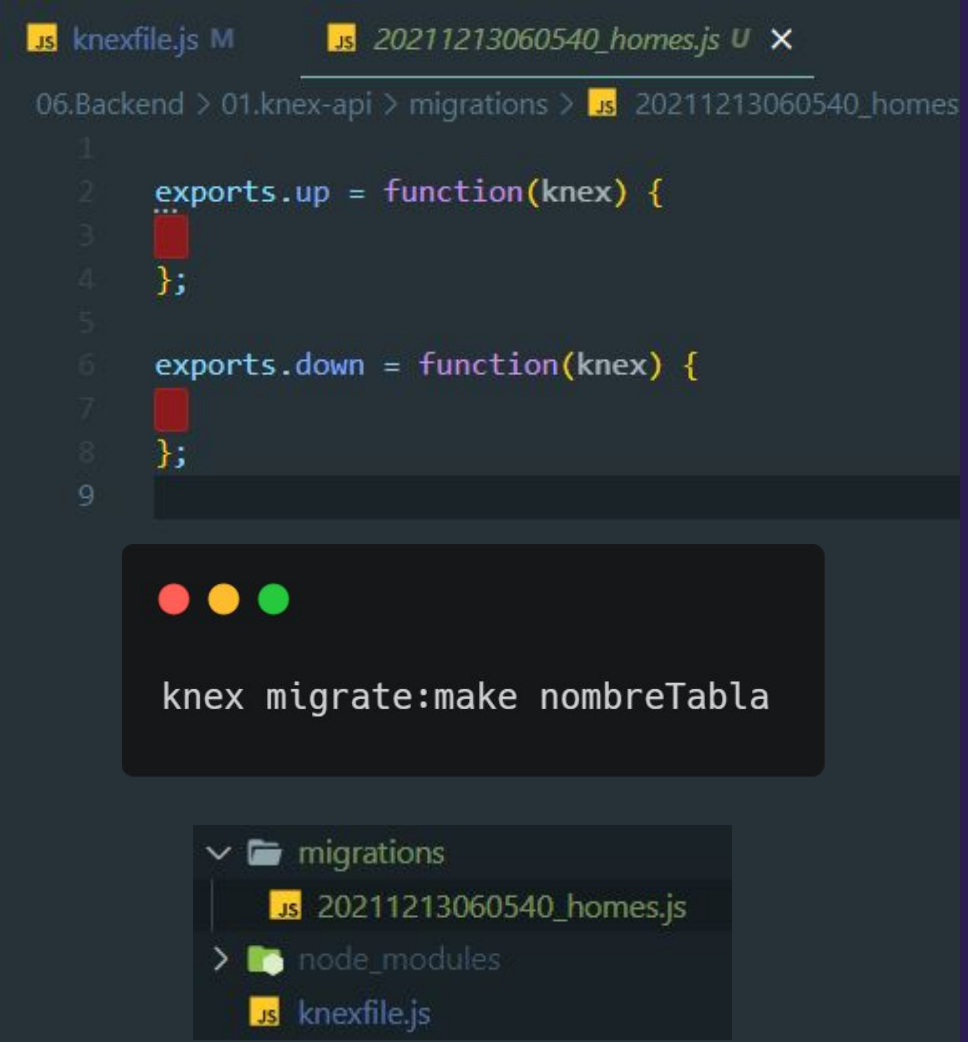
DEV.F::



Base de Datos a Realizar

Se utilizará como ejemplo una base de datos de un sistema parecido a AirBNB, donde un usuario podrá dar de alta una o más de sus casas

DEV.F::



6. Crear migración

Ejecutaremos el comando:

```
knex migrate:make homes
```

Esto creará una carpeta llamada **migrations** y dentro colocará una plantilla para que podamos colocar el cambio en nuestra base de datos con una tabla llamada **homes**.

Nota: El nombre del archivo no se debe manipular, puesto que tiene un timestamp que usa knex para el versionamiento de la base de datos.

DEV.F::

```
exports.up = function (knex, promise) {
  return knex.schema.hasTable("homes").then(function (exists)
{
    if (!exists) {
      return knex.schema.createTable("homes", function (table)
{
        table.increments("house_id").primary();
        table.string("title").notNullable();
        table.text("description");
        table.integer("guests");
        table.text("address");
        table.decimal("rental_price",12, 2);
table.boolean("active").notNullable().defaultTo(true);
table.timestamp("created_at").defaultTo(knex.fn.now());
      });
    }
  });
};

exports.down = function (knex, Promise) {
  return knex.schema.hasTable("homes").then(function (exists)
{
    if (exists) {
      return knex.schema.dropTable("homes");
    }
  });
};
```

7. Crear tabla homes programáticamente

Dentro de exports.up colocaremos la lógica de creación de nuestra tabla.

Comprobaremos si esta existe o no en la base de datos, y procederemos a crearla en caso de que no exista.

DEV.F::

```
$ knex migrate:latest
Using environment: development
no existe la base de datos "knexapi"
error: no existe la base de datos "knexapi"
```

8. Correr la migración

Ejecutaremos el comando:

```
knex migrate:latest
```

En este punto, como se intenta crear la información en la base de datos, nos daremos cuenta si nuestra configuración es correcta.

DEV.F::

```
exports.up = function (knex) {
 return knex.schema.hasTable("users").then(function (exists) {
   if (!exists) {
     return knex.schema.createTable("users", function (table) {
       table.increments('user_id').primary(),
       table.string('name').notNullable(),
       table.string('last_name'),
       table.string('email').notNullable(),
       table.string('phone'),
       table.string('description'),
       table.boolean('active').notNullable().defaultTo(true),
       table.timestamp('created_at').defaultTo(knex.fn.now())
      });
   }
 });
};

exports.down = function (knex) {
 return knex.schema.hasTable("users").then(function (exists) {
   if (exists) {
     return knex.schema.dropTable("users");
   }
 });
};
```

9. Crear tabla users programáticamente

Ejecutaremos el comando:

`knex migrate:make users`

Dentro de exports.up colocaremos la lógica de creación de nuestra tabla.

Comprobaremos si esta existe o no en la base de datos, y procederemos a crearla en caso de que no exista.

DEV.F::

10. Correr la migración

Ejecutaremos el comando:

```
knex migrate:latest
```

DEV.F:

```
exports.up = function(knex) {
  return knex.schema.hasTable("homes").then(function (exists) {
    if (exists) {
      return knex.schema.table("homes", function (table) {
table.integer('fk_user')
.unsigned()
.references('users.user_id');
      });
    };
 });
};

exports.down = function(knex) {
 return knex.schema.hasTable("homes").then(function (exists) {
   if (exists) {
     return knex.schema.table("homes", function (table) {
       table.dropColumn('fk_user'); //borro la columna fk_user
     });
   };
 });
};
```

11. Crear relación entre tabla users y homes programáticamente

Ejecutaremos el comando:

```
knex migrate:make home_has_user
```

Es tiempo de crear la relación 1 a muchos entre las tablas users y homes.

Para ello agregaremos un nuevo atributo a la tabla de homes, donde crearemos la llave foránea que hará referencia al id de usuario qué le corresponda.

DEV.F::

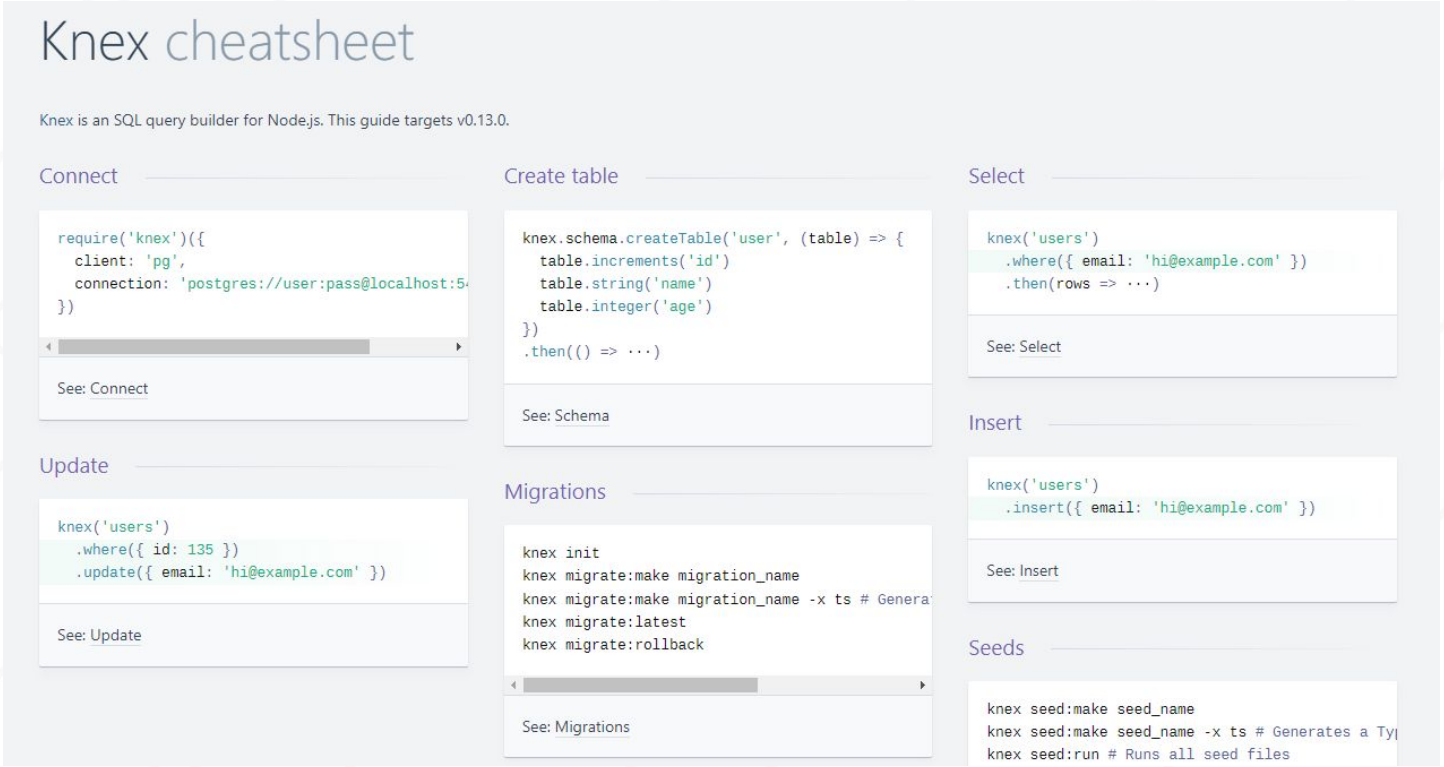
12. Correr la migración

Ejecutaremos el comando:

```
knex migrate:latest
```

DEV.F::

Knex Cheat Sheet



Knex cheatsheet

Knex is an SQL query builder for Node.js. This guide targets v0.13.0.

Connect

```
require('knex')({
  client: 'pg',
  connection: 'postgres://user:pass@localhost:5
})
```

See: Connect

Update

```
knex('users')
  .where({ id: 135 })
  .update({ email: 'hi@example.com' })
```

See: Update

Create table

```
knex.schema.createTable('user', (table) => {
  table.increments('id')
  table.string('name')
  table.integer('age')
})
.then(() => ···)
```

See: Schema

Migrations

```
knex init
knex migrate:make migration_name
knex migrate:make migration_name -x ts # Genera
knex migrate:latest
knex migrate:rollback
```

See: Migrations

Select

```
knex('users')
  .where({ email: 'hi@example.com' })
  .then(rows => ···)
```

See: Select

Insert

```
knex('users')
  .insert({ email: 'hi@example.com' })
```

See: Insert

Seeds

```
knex seed:make seed_name
knex seed:make seed_name -x ts # Generates a Ty
knex seed:run # Runs all seed files
```

https://devhints.io/knex

DEV.F::

¡ Listo !

Con esto hemos realizado de principio a fin una configuración inicial de knex en nuestro proyecto de node.js.

El siguiente paso es entonces, realizar con ayuda de Express un backend capaz de hacer uso de la base de datos a través de una API.


¡ GRACIAS !
CésarGuerra
www.cesarguerra.mx