



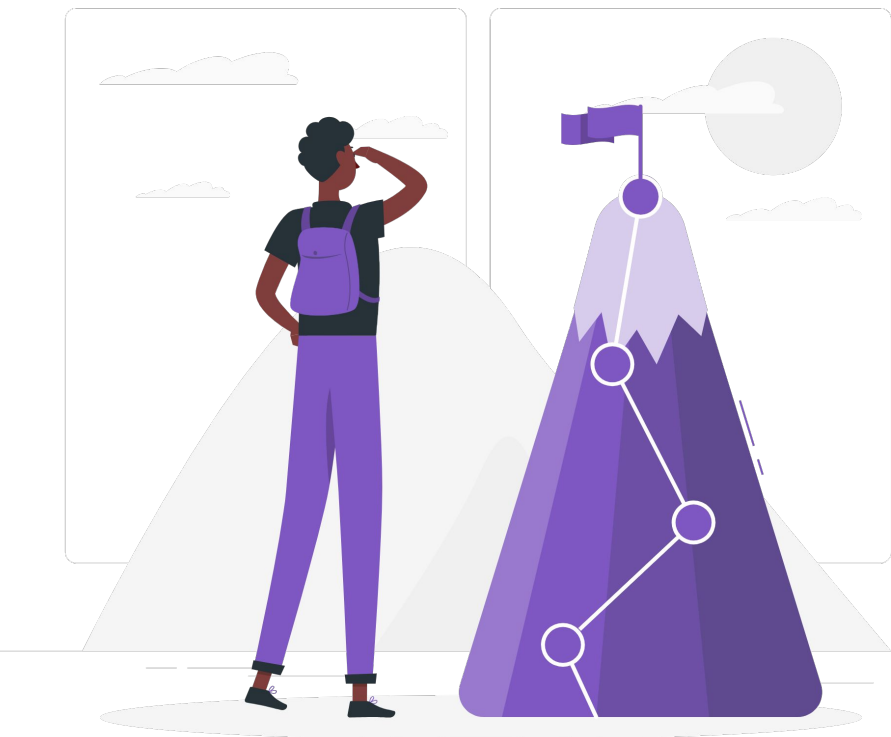
# KNEX.JS

## Intro a Knex

**DEV.F**  
DESARROLLAMOS(PERSONAS);

Elaborado por: César Guerra

dev



# 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 PostgreSQL con Knex.
- Crear modelos con Knex y realizar migraciones.
- Crear nuestro servidor de express



<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);

dev



```
npm install knex -g
```

# 1. Instalación

Realizamos la instalación de knex desde la línea de comandos.



*Nota: Recordemos que -y nos sirve para que 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.



```
npm i express knex pg
```

### 3. Instalar dependencias

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

En este caso:

- express
- knex
- pg

```
1 // Update with your config here
2
3 module.exports = {
4   ...
5   development: {
6     client: 'sqlite3',
7     connection: {
8       filename: './dev.sqlite3'
9     },
10   },
11
12   staging: {
13     client: 'postgresql',
14     connection: {
15       database: 'my_db',
16       user: 'username',
17       password: 'password'
18     },
19     pool: {
20       min: 2,
21       max: 10
22     },
23   },
24   ...
25 }
```

  
knex init

## 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.



```
06.Backend > 01.knex-api > JS knexfile.js > [?] <unknown> > 🔑 developm
1 // Update with your config settings.
2
3 module.exports = {
4   ...
5   development: {
6     client: 'postgresql',
7     connection: {
8       host: '127.0.0.1',
9       database: 'knexapi',
10      user: 'postgres',
11      password: 'password'
12    },
13    pool: {
14      min: 2,
15      max: 10
16    },
17    migrations: {
18      tableName: 'knex_migrations'
19    }
20  },
21
22  staging: {
23    client: 'postgresql',
24    connection: {
```

## 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.



# 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.

06.Backend &gt; 01.knex-api &gt; migrations &gt; JS 20211213060540\_homes

```
1
2 exports.up = function(knex) {
3   ...
4 };
5
6 exports.down = function(knex) {
7   ...
8 };
9
```



```
knex migrate:make nombreTabla
```

▼ migrations

JS 20211213060540\_homes.js

&gt; node\_modules

JS knexfile.js

## 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.

```
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").nullable();  
        table.text("description");  
        table.integer("guests");  
        table.text("address");  
        table.boolean("active").nullable().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("users");  
    }  
  });  
};
```

## 7. Crear tabla 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

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

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

# 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:5432/db'
})
```

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 # Generates a TypeScript migration
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 TypeScript seed file
knex seed:run # Runs all seed files
```

<https://devhints.io/knex>

# ¡ 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.