Juan Garrido Troche

grd dEV

PORTFOLIO

Contenido

[BASE DE DATOS 3](#_Toc140495441)

[ENTIDADES 3](#_Toc140495442)

[USUARIOS 3](#_Toc140495443)

[CURRICULUMS 4](#_Toc140495444)

[OFICIOS 4](#_Toc140495445)

[TRABAJOS 4](#_Toc140495446)

[HABILIDADES 5](#_Toc140495447)

[ESTUDIOS 5](#_Toc140495448)

[PROYECTOS 5](#_Toc140495449)

[TECNOLOGÍAS 6](#_Toc140495450)

[CONTACTOS 6](#_Toc140495451)

[RELACIONES 6](#_Toc140495452)

[TECNOLOGÍAS USADAS(PROJECTS – TECHNOLOGIES) 6](#_Toc140495453)

[CREACIÓN DE TABLAS EN MYSQL WORKBENCH 7](#_Toc140495454)

[BACKEND: NODEJS Y EXPRESS 11](#_Toc140495455)

[DEPENDENCIAS NECESARIAS PARA XPACCOUNT 12](#_Toc140495456)

[RUTAS / ENDPOINTS 14](#_Toc140495457)

PORTFOLIO DE LOS PROYECTOS REALIZADOS

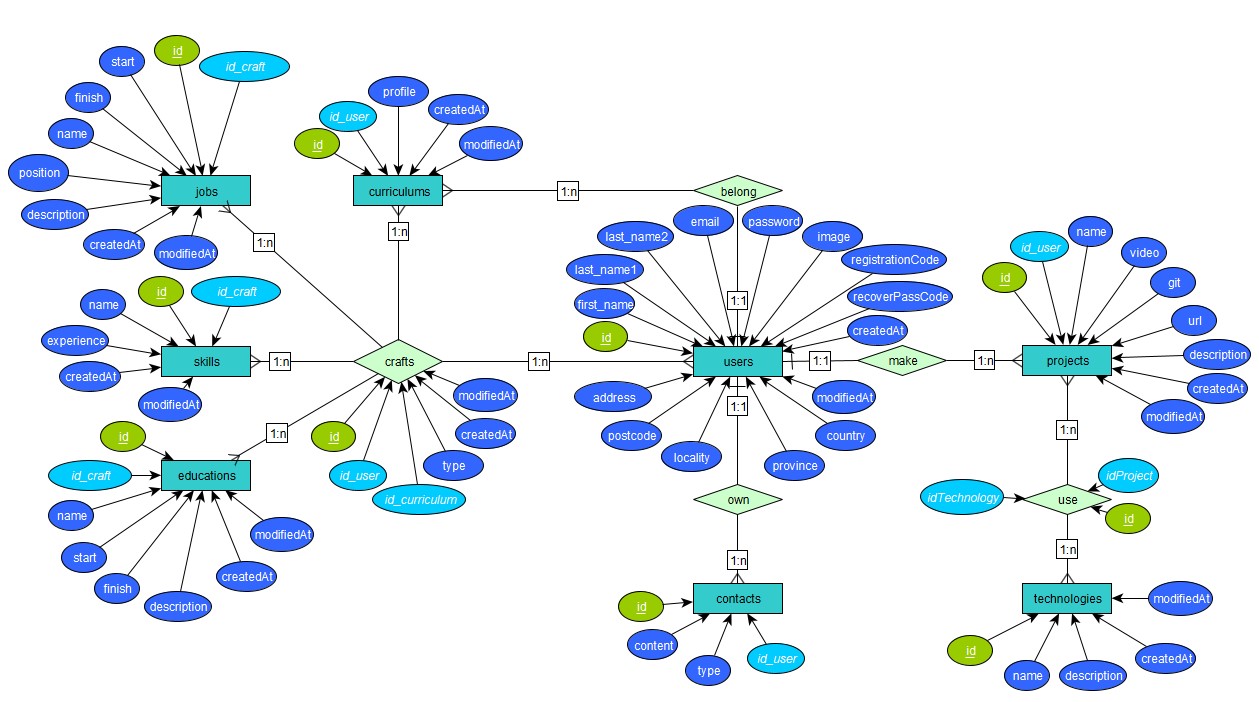
La aplicación se llamará portfolio, consiste en gestionar varios perfiles de curriculum vitae según la petición del reclutador y proyectos realizados a lo largo de mi vida laboral.

La aplicación podrá realizar las siguientes acciones o procesos:

* Se mostrarán los proyectos realizados.
* Se mostrarán los distintos perfiles de CV.
* Página con todos los contactos del candidato.
* Login de administrador.
* Como administrador se podrán realizar las siguientes acciones:
  + Crear, editar y eliminar proyectos.
  + Crear, editar y eliminar perfiles de CV.
  + Crear, editar y eliminar contactos.
  + Editar los datos personales y contraseña del administrador, que a su vez es el usuario del portfolio.

# BASE DE DATOS

Para realizar la base de datos hemos utilizado la herramienta yEd Graph Editor, donde hemos creado las entidades con sus atributos y las relaciones entre ellas:



## ENTIDADES

### USUARIOS

Users:

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* firstName: Tipo texto, VARCHAR con un máximo de 100 caracteres, requerido.
* lastName1: Tipo texto, VARCHAR con un máximo de 100 caracteres, requerido.
* lastName2: Tipo texto, VARCHAR con un máximo de 100 caracteres. No es requerido ya que no todo el mundo tiene 2 apellidos.
* email: Tipo texto, VARCHAR de un máximo de 100 caracteres, requerido y único.
* password: Tipo texto, VARCHAR de un máximo de 100 caracteres, requerido y obligatorio.
* image: Tipo texto, VARCHAR de un máximo de 50 caracteres.
* registrationCode: Código de registro de usuario: VARCHAR (20)
* recoverPassCode: Código de recuperación de contraseña: VARCHAR (20)
* address: VARCHAR de un máximo de 200 caracteres.
* postcode: VARCHAR de un máximo de 5 caracteres.
* locality: VARCHAR de un máximo de 100 caracteres.
* province: VARCHAR de un máximo de 50 caracteres.
* country: VARCHAR de un máximo de 50 caracteres.
* createdAt: Fecha de creación del usuario. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.

### CURRICULUMS

curriculums:

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* idUser: Clave foránea de la tabla users y número entero.
* profile: Nombre del tipo de curriculum (técnico, programador, gestor de proyectos, etc). VARCHAR (50), requerido.
* createdAt: Fecha de creación del usuario. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.

### OFICIOS

crafts:

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* idUser: Clave foránea de la tabla users y número entero.
* idCurriculum: Clave foránea de la tabla curriculums y número entero.
* type: Nombre del tipo de craft limitado a 3: job, skill, eduction. ENUM(‘trabajo, ‘habilidad’, ‘educación’). Obligatorio.
* createdAt: Fecha de creación del asiento. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.

### TRABAJOS

jobs:

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* idCraft: Clave foránea de la tabla crafts y número entero.
* start: Fecha de inicio del trabajo. Timestamp y requerido.
* finish: Fecha fin del trabajo. Timestamp.
* name: Nombre del trabajo VARCHAR(100) y requerido.
* position: Cargo que has ocupadoen el puesto de trabajo VARCHAR(100).
* description: pequeño resumen de tus funciones VARCHAR(250).
* createdAt: Fecha de creación del asiento. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.

### HABILIDADES

skills:

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* idCraft: Clave foránea de la tabla crafts y número entero.
* name: Nombre de la habilidad. VARCHAR (100). Obligatorio.
* experience: Experiencia en esta habilidad con un valor por defecto de 0 (no evaluado) y evaluado entre 1(sin experiencia) y 5 (experto). TINYINT UNSIGNED DEFAULT 0.
* createdAt: Fecha de creación del asiento. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.

### ESTUDIOS

educations:

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* idCraft: Clave foránea de la tabla crafts y número entero.
* name: Nombre del estudio. VARCHAR (100). Obligatorio.
* school: Escuela donde se impartió el conocimiento. VARCHAR (100).
* start: Fecha de inicio del trabajo. Timestamp y requerido.
* finish: Fecha fin del trabajo. Timestamp.
* description: breve resumen de los conocimientos adquiridos. VARCHAR(250).
* createdAt: Fecha de creación del asiento. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.

### PROYECTOS

projects:

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* idUser: Clave foránea de la tabla users y número entero.
* name: Nombre del proyecto. VARCHAR (100). Obligatorio.
* video: URL del video explicativo del proyecto. VARCHAR (100).
* repo: URL del repositorio donde tenemos el proyecto. VARCHAR (100).
* url: URL de la web donde está subido el proyecto. VARCHAR (100).
* description: breve resumen del proyecto. VARCHAR(250).
* createdAt: Fecha de creación del asiento. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.

### TECNOLOGÍAS

technologies:

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* name: Nombre de la tecnología empleada en el proyecto. VARCHAR (100). Requerido.
* description: breve resumen de la tecnología utilizada. VARCHAR(250).
* createdAt: Fecha de creación del asiento. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.

### CONTACTOS

contacts:

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* idUser: Clave foránea de la tabla users y número entero.
* type: Nombre del tipo de contacto del usuario (teléfono, email, git, twitter, etc). VARCHAR (100). Requerido.
* content: El contenido del tipo de contacto, por ejemplo, si type es email, el content sería mi.nombre@midominio.com. VARCHAR(100). Requerido.
* createdAt: Fecha de creación del asiento. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.

## RELACIONES

### TECNOLOGÍAS USADAS(PROJECTS – TECHNOLOGIES)

used\_techs:

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* idProject: Clave foránea de la tabla projects y número entero.
* idTechnology: Clave foránea de la tabla technologies y número entero.

## CREACIÓN DE TABLAS EN MYSQL WORKBENCH

Las tablas se crearán dentro de la base de datos llamada portfolio que, a su vez, se incluirá dentro del proyecto dentro del archivo bbdd/initDB.js:

* + - CREATE DATABASE IF NOT EXISTS portfolio;
    - USE portfolio;

Se eliminan las tablas de forma inversa a como se han creado

* + - DROP TABLE IF EXISTS contacts;
    - DROP TABLE IF EXISTS used\_techs;
    - DROP TABLE IF EXISTS technologies;
    - DROP TABLE IF EXISTS projects;
    - DROP TABLE IF EXISTS educations;
    - DROP TABLE IF EXISTS skills;
    - DROP TABLE IF EXISTS jobs;
    - DROP TABLE IF EXISTS crafts;
    - DROP TABLE IF EXISTS curriculums;
    - DROP TABLE IF EXISTS users;
* CREATE TABLE IF NOT EXISTS users (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

firstName VARCHAR(100) NOT NULL,

lastName1 VARCHAR(100) NOT NULL,

lastName2 VARCHAR(100),

email VARCHAR(100) UNIQUE NOT NULL,

password VARCHAR(100) NOT NULL,

image VARCHAR(50),

registrationCode VARCHAR(100),

recoverPassCode VARCHAR(20),

address VARCHAR(200),

postcode VARCHAR(5),

locality VARCHAR(100),

province VARCHAR(50),

country VARCHAR(50),

createdAt TIMESTAMP NOT NULL,

modifiedAt TIMESTAMP;)

* + - CREATE TABLE IF NOT EXISTS curriculums (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

idUser INT UNSIGNED NOT NULL,

FOREIGN KEY (idUser) REFERENCES users(id),

profile VARCHAR(50) NOT NULL,

createdAt TIMESTAMP NOT NULL,

modifiedAt TIMESTAMP;

* + - CREATE TABLE IF NOT EXISTS crafts (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

idUser INT UNSIGNED NOT NULL,

FOREIGN KEY (idUSer) REFERENCES users(id),

idCurriculum INT UNSIGNED NOT NULL,

FOREIGN KEY (idCurriculum) REFERENCES curriculums(id),

type ENUM('Trabajo', 'Habilidad', 'Educación') NOT NULL,

createdAt TIMESTAMP NOT NULL,

modifiedAt TIMESTAMP;

* + - CREATE TABLE IF NOT EXISTS jobs (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

idCraft INT UNSIGNED NOT NULL,

FOREIGN KEY (idCraft) REFERENCES crafts(id),

start TIMESTAMP NOT NULL,

finish TIMESTAMP,

name VARCHAR(100) NOT NULL,

position VARCHAR(100),

description VARCHAR(250),

createdAt TIMESTAMP NOT NULL,

modifiedAt TIMESTAMP;

* + - CREATE TABLE IF NOT EXISTS skills (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

idCraft INT UNSIGNED NOT NULL,

FOREIGN KEY (idCraft) REFERENCES crafts(id),

name VARCHAR(100) NOT NULL,

experience TINYINT UNSIGNED DEFAULT 0,

createdAt TIMESTAMP NOT NULL,

modifiedAt TIMESTAMP;

* + - CREATE TABLE IF NOT EXISTS educations (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

idCraft INT UNSIGNED NOT NULL,

FOREIGN KEY (idCraft) REFERENCES crafts(id),

name VARCHAR(100) NOT NULL,

school VARCHAR(100),

start TIMESTAMP NOT NULL,

finish TIMESTAMP,

description VARCHAR(250),

createdAt TIMESTAMP NOT NULL,

modifiedAt TIMESTAMP;

* + - CREATE TABLE IF NOT EXISTS projects (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

idUser INT UNSIGNED NOT NULL,

FOREIGN KEY (idUser) REFERENCES users(id),

name VARCHAR(100) NOT NULL,

video VARCHAR(100),

repo VARCHAR(100),

url VARCHAR(100),

description VARCHAR(250),

createdAt TIMESTAMP NOT NULL,

modifiedAt TIMESTAMP;

* + - CREATE TABLE IF NOT EXISTS technologies (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

description VARCHAR(250),

createdAt TIMESTAMP NOT NULL,

modifiedAt TIMESTAMP

* + - CREATE TABLE IF NOT EXISTS used\_techs (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

idProject INT UNSIGNED NOT NULL,

FOREIGN KEY (idProject) REFERENCES projects(id),

idTechnology INT UNSIGNED NOT NULL,

FOREIGN KEY (idTechnology) REFERENCES technologies(id);

* + - CREATE TABLE IF NOT EXISTS contacts (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

idUser INT UNSIGNED NOT NULL,

FOREIGN KEY (idUser) REFERENCES users(id),

type VARCHAR(100) NOT NULL,

content VARCHAR(100) NOT NULL

# BACKEND: NODEJS, EXPRESS Y MYSQL

Antes de instalar alguna dependencia crearemos nuestro fichero package.json con la siguiente línea de comandos, situada en la carpeta de nuestro proyecto:

* + - npm init
    - npm init -y (si queremos crear package.json con valores por defecto)

En nuestro proyecto queda del siguiente modo:

{

  "name": "portfolio",

  "version": "1.0.0",

  "description": "API que permite gestionar el CV y proyectos",

  "main": "server.js",

  "scripts": {

    "dev": "nodemon server.js"

  },

  "author": "Juan Garrido Troche",

  "license": "ISC"

}

# DEPENDENCIAS NECESARIAS PARA PORTFOLIO

Las dependencias son aquellas aplicaciones o bibliotecas requeridas por otro programa para poder funcionar correctamente. Por ello, se dice que determinado programa “depende” de tal aplicación o biblioteca (library). Para instalarlos, emplearemos un sistema de gestión de paquetes de Node llamado “npm” (Node Package Management).

Existen 3 tipos de dependencias:

* Dependencias CORE: Aquellas que ya vienen instaladas y se pueden utilizar solo llamando al módulo, como, por ejemplo, path o fs.
* Dependencias: Aquellas necesarias que instalaremos para que nuestra API funcione de manera correcta. Ejemplo: mysql2, dotenv, bcrypt, etc.
* Dependencias para desarrolladores(-D): Aquellas dependencias que nos harán la programación más fácil como control de sintaxis (eslint), formateo del código para que se vea ordenado (prettier) o reinicio de nuestro servidor cada vez que guardamos un cambio (nodemon). Este tipo de dependencias se instalan añadiendo un -D en la línea de comandos. En cuanto finalice nuestro proyecto y lo pongamos en producción se podrían desinstalar (npm uninstall mi\_DevDependence).

A continuación, vamos a indicar que dependencias son las que vamos a utilizar en nuestro proyecto:

|  |  |  |  |
| --- | --- | --- | --- |
| NOMBRE | INSTALACIÓN | TIPO | USO |
| express | npm i express | Dependencia | Nos permite levantar un servidor. Además nos permite leer los datos enviados desde el body en formato raw -JSON. |
| dotenv | npm i dotenv | Dependencia | Nos permite leer los valores de las variables del archivo oculto .env |
| mysql2 | npm i mysql2 | Dependencia | Crea una conexión con una base de datos y una vez conectados, podemos hacer “crud” de tablas y/o registros pero nunca crear la base de datos. |
| bcrypt | npm i bcrypt | Dependencia | Permite encriptar y desencriptar datos. |
| nodemon | npm i nodemon -D | Dev | Monitoriza los cambios en el código fuente que se está desarrollando y automáticamente reinicia el servidor que está corriendo sin tener que reiniciarlo. |
| jsonwebtoken | npm i jsonwebtoken | Dependencia | Nos crea un token de seguridad estándar para transmitir de forma segura en internet, por medio del formato JSON. |
| fs | No hay que instalar | Core | Gestiona los ficheros. |
| path | No hay que instalar | Core | Gestiona las rutas de nuestros ficheros. |
| joi | Npm i @hapi/joi | Dependencia | Sirve para validar todo tipo de datos introducidos por el usuario. |
| randomstring | npm i randomstring | Dependencia | Genera cadenas de caracteres alfanuméricos. En nuestro caso lo utilizaremos para generar el código de recuperación. |
| nodemailer | npm i nodemailer | Dependencia | Nos permite enviar correos a través de una configuración sencilla. |
| cors | npm i cors | Dependencia | Nos permite compartir recursos en distintos dominios y orígenes, por lo que podemos conectar con fetch entre backend y frontend |
| morgan | npm i morgan | Dependencia | Muestra cualquier petición al servidor a través de nuestro terminal de node |
| uuid | Npm i uuid | Dependencia | Renombra los ficheros que subamos a nuestra bbdd o genera el código de validación para activar el usuario. |
| xlsx | npm i xlsx | Dependencia | Nos ayudará a leer los archivos excel |

# RUTAS / ENDPOINTS

Un endpoint es una pasarela que conecta los procesos del servidor de la aplicación con una interfaz (API, Application Programming Interface), es decir, la ruta a la que se envían peticiones que no tiene porqué coincidir con nuestra URL.

Antes de empezar a programar nuestra API debemos establecer los endpoints necesarios para que nuestra aplicación sea funcional:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MÉTODO | RUTA | FUNCIÓN | USO | PERMISOS |
| ENDPOINTS USUARIOS | | | | |
| POST | /user/login | loginUser.js | Login de usuario. | Todos los usuarios |
| POST | /user/register | registerUser.js | Registrar un nuevo usuario. | Todos los usuarios |
| PUT | /user/register/validate/:registrationCode | validateUser.js | Puedes activar el usuario de tu cuenta. | Todos los usuarios |
| GET | /user/loggedProfile | readLoggedProfile.js | Consultar los datos del usuario logueado. | isAuth |
| GET | /user/accounts | getUserAccounts.js | Consultar las cuentas de un usuario | isAuth |
|  |  |  |  |  |
| ENDPOINTS ACCOUNTS | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| POST | /account | createAccount.js | Crear una cuenta bancaria. | isAuth |
| POST | /account/:idAccount | updateAccount.js | Actualizar una cuenta bancaria. | isAuth |
| PUT | /account/:idAccount | deleteAccount.js | Eliminar una cuenta bancaria. | isAuth |
| GET | /account/:idAccount | readAccountEntries.js | Obtener todos los asientos bancarios de una cuenta | isAuth |
|  |  |  |  |  |
| ENDPOINTS CATEGORIES | | | | |
| POST | /account/:idAccount/category | createCategory.js | Crear una categoría de asiento bancario. | isAuth |
| PUT | /account/:idAccount/category/:idCategory | updateCategory.js | Actualizar una categoría de asiento bancario. | isAuth |
| DELETE | /account/:idAccount/category/:idCategory | deleteCategory.js | Eliminar una categoría. | isAuth |
|  |  |  |  |  |
| ENDPOINTS SUBCATEGORIES | | | | |
| POST | /category/:idCategory/sub | createSubcategory.js | Crea una subcategoría de asiento bancario. | isAuth |
| PUT | /category/:idCategory/sub/:idSub | updateSubcategory.js | Actualiza una subcategoría de asiento bancario. | isAuth |
| DELETE | /category/:idCategory/sub/:idSub | deleteSubcategory.js | Elimina una subcategoría. | isAuth |
|  |  |  |  |  |
| ENDPOINTS ENTRIES | | | | |
| POST | /account/:idAccount/entry | createEntry.js | Crear un nuevo asiento | isAuth |
| PUT | /account/:idAccount/entry/:idEntry | Actualizar un asiento | updateEntry.js | isAuth |
| DELETE | /account/:idAccount/entry/:idEntry | Eliminar un asiento | deleteEntry.js | isAuth |