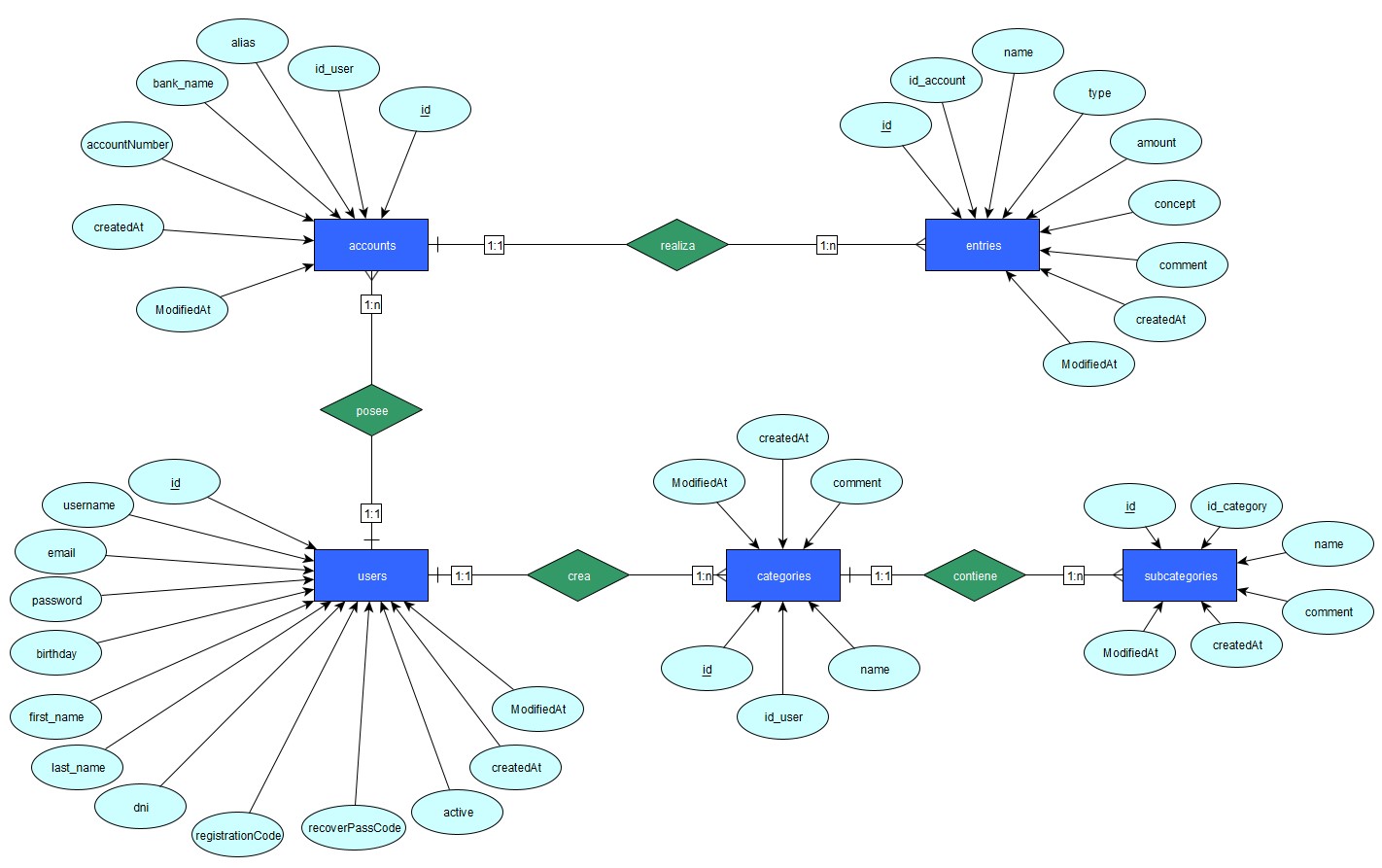
APLICACIÓN SOBRE MOVIMIENTOS DE TU CUENTA

La aplicación se llamará xpaccount, viene del inglés expenses account, gastos de mi cuenta bancaria. Consiste en gestionar los gastos e ingresos que se generan en mis cuentas bancarias.

* La aplicación permitirá añadir, editar y eliminar cuentas.
* Añadir, editar y eliminar movimientos/asientos bancarios de tu cuenta.
* Añadir asientos bancarios, tipos de movimientos según el asiento bancario.
* Indicar si es ingreso o gasto.
* Fecha del asiento.
* Conceptos, observaciones.
* Mostrar el Total de dinero en cuenta.
* Registro de usuario
* Login

# BASE DE DATOS

Para realizar la base de datos hemos utilizado la herramienta yEd Graph Editor, donde hemos creado las entidades con los siguientes atributos:



## ENTIDADES

Users:

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* username: Tipo texto, VARCHAR con un máximo de 50 caracteres, requerido.
* email: Tipo texto, VARCHAR de un máximo de 100 caracteres, requerido y obligatorio.
* password: Tipo texto, VARCHAR de un máximo de 100 caracteres, requerido y obligatorio.
* birthday: Fecha de nacimiento del usuario. Timestamp y requerido. Obligatorio ser mayor de edad.
* first\_name: Nombre/s del usuario. VARCHAR(100)
* last\_name: Apellido/s del usuario. VARCHAR(200)
* dni: DNI del usuario. VARCHAR(9)
* registrationCode: Código de registro de usuario: VARCHAR (20)
* recoverPassCode: Código de recuperación de contraseña: VARCHAR (20)
* active: El usuario puede darse de alta pero hasta que no introduce el código de registro no se activa su cuenta. Por defecto vendrá como false. BOOLEAN.
* createdAt: Fecha de creación del usuario. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.

## CUENTAS

accounts:

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* idUser: Clave foránea de la tabla user y número entero.
* Alias: Nombre de la cuenta bancaria. VARCHAR (50).
* Bank\_name: Nombre del banco. VARCHAR (100).
* accountNumber: Número de cuenta compuesto por 24 dígitos. VARCHAR(24). Obligatorio. Los 24 dígitos son: código IBAN (International Bank Account Number code). Se compone de 2 letras y 2 números; Código de la entidad bancaria: Se compone de 4 dígitos; Código de la oficina bancaria: Se compone de 4 dígitos; dígitos de control: 2 dígitos; Número de cuenta: 10 dígitos.
* createdAt: Fecha de creación del usuario. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.

## CATEGORÍAS

categories:

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* idUser: Clave foránea de la tabla users y número entero.
* name: Nombre del tipo de categoría. VARCHAR (100). Obligatorio.
* comment: Comentario acerca de la categoría creada VARCHAR(200).
* createdAt: Fecha de creación del asiento. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.

## SUBCATEGORÍAS

subcategories:

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* idCat: Clave foránea de la tabla categories y número entero.
* name: Nombre del tipo de subcategoría. VARCHAR (100). Obligatorio.
* comment: Comentario acerca de la categoría creada VARCHAR(200).
* createdAt: Fecha de creación del asiento. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.

## ASIENTOS

entries:

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* idAccount: Clave foránea de la tabla account y número entero.
* createdAt: Fecha de creación del asiento. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.
* name: Nombre del asiento bancario. VARCHAR (50). Obligatorio.
* Amount: Cantidad asentada (ingreso o gasto). DECIMAL(9,2). Obligatorio.
* Concept: Concepto del asiento. VARCHAR(200). El banco nos entrega un concepto de hasta 128 caracteres.
* comment: Comentario acerca del asiento bancario creado VARCHAR(200).

## CREACIÓN DE TABLAS EN MYSQL WORKBENCH

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

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

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

* + - DROP TABLE IF EXISTS subcategories;
    - DROP TABLE IF EXISTS categories;
    - DROP TABLE IF EXISTS entries;
    - DROP TABLE IF EXISTS accounts;
    - DROP TABLE IF EXISTS users;
    - CREATE TABLE IF NOT EXISTS users (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

username VARCHAR(100) UNIQUE NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

password VARCHAR(100) NOT NULL,

birthday TIMESTAMP,

firstName VARCHAR(100),

lastName VARCHAR(200),

dni VARCHAR(9),

registrationCode VARCHAR(100),

recoverPassCode VARCHAR(20),

active BOOLEAN DEFAULT false,

createdAt TIMESTAMP NOT NULL,

modifiedAt TIMESTAMP;

* + - CREATE TABLE IF NOT EXISTS accounts (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

idUser INT UNSIGNED NOT NULL,

FOREIGN KEY (idUser) REFERENCES users(id),

alias VARCHAR(50),

bankName VARCHAR(100),

accountNumber VARCHAR(24) NOT NULL,

createdAt TIMESTAMP NOT NULL,

modifiedAt TIMESTAMP;

* + - CREATE TABLE IF NOT EXISTS entries (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

idAccount INT UNSIGNED NOT NULL,

FOREIGN KEY (idAccount) REFERENCES accounts(id),

category VARCHAR(100) NOT NULL,

subcategory VARCHAR(100) NOT NULL,

amount DECIMAL(9,2) NOT NULL,

concept VARCHAR(200),

comment VARCHAR(200),

createdAt TIMESTAMP NOT NULL,

modifiedAt TIMESTAMP;

* + - CREATE TABLE IF NOT EXISTS categories (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

idUser INT UNSIGNED NOT NULL,

FOREIGN KEY (idUser) REFERENCES users(id),

name VARCHAR(100) NOT NULL,

comment VARCHAR(200),

createdAt TIMESTAMP NOT NULL,

modifiedAt TIMESTAMP;

* + - CREATE TABLE IF NOT EXISTS subcategories (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

idCat INT UNSIGNED NOT NULL,

FOREIGN KEY (idCat) REFERENCES categories(id),

name VARCHAR(100) NOT NULL,

comment VARCHAR(200),

createdAt TIMESTAMP NOT NULL,

modifiedAt TIMESTAMP;

# INTRODUCCIÓN

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": "xpaccount",

  "version": "1.0.0",

  "description": "API que permite gestionar los asientos bacarios de tu cuenta corriente",

  "main": "server.js",

  "scripts": {

    "dev": "nodemon server.js"

  },

  "author": "Juan Garrido Troche",

  "license": "ISC"

}

# DEPENDENCIAS NECESARIAS PARA XPACCOUNT

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 |
|  |  |  |  |  |
| 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 |
|  |  |  |  |  |
| ENDPOINTS CATEGORIES | | | | |
|  |  |  |  |  |