Finzen

Your personal finance assistant

Descripción:

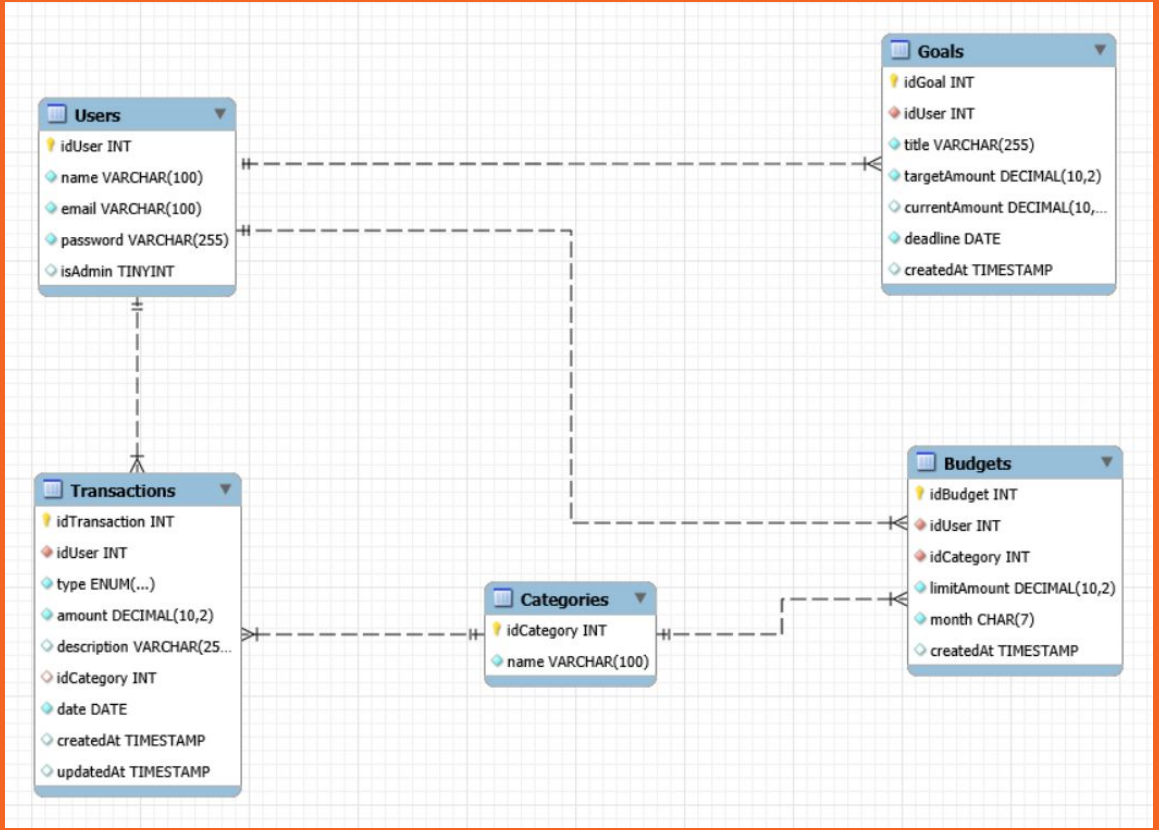
- Proyecto de API para contabilidad personal, presupuestos y objetivos de ahorro.
- Se conectará a una API bancaria que proporcionará los datos de transacciones de un usuario determinado, y las categorías de transacciones que usa su banco, y con todo ello operará la presente aplicación.
- Los usuarios se dan de alta por si mismos.
- Hay un usuario administrador único.

Base de datos

Transactions y Categories no tienen CRUDs. Sus datos se obtienen desde la API del banco que ya proporciona las categorías. Los movimientos bancarios del usuario ya vienen categorizados.

CRUDs:

- Goals:
  - report
- Budgets
  - report anual
- Users:
  - C. cualquiera
  - RUD solo admin



Detalles

```
const Budget = connection.define('Budget', {
  idBudget: { type: DataTypes.INTEGER.UNSIGNED, autoIncrement: true, primaryKey: true },
  limitAmount: { type: DataTypes.FLOAT, allowNull: false },
  month: { type: DataTypes.STRING(7), allowNull: false }, // YYYY-MM
}, {
  tableName: 'Budgets',        // nombre exacto de la tabla en la DB
  freezeTableName: true,       // evita que Sequelize pluralice "Budget"
  timestamps: true,            // Genera campo createdAt y updatedAt
  updatedAt: false             // No crear campo updatedAt en esta tabla
});
```

- Problema de pluralización del nombre de las tablas en Sequelize.
- Añadir timestamps a las entradas de las tablas.

- Modelo Budget de la tabla de base de datos Budgets

```
import { DataTypes } from 'sequelize';
import connection from '../config/sequelize.js';

const Budget = connection.define('Budget', {
  idBudget: { type: DataTypes.INTEGER.UNSIGNED, autoIncrement: true, primaryKey: true },
  limitAmount: { type: DataTypes.FLOAT, allowNull: false },
  month: { type: DataTypes.STRING(7), allowNull: false }, // YYYY-MM
}, {
  tableName: 'Budgets',        // 👈 nombre exacto de la tabla en la DB
  freezeTableName: true,       // 👈 evita que Sequelize pluralice "Budget"
  timestamps: true,
  updatedAt: false
});

export default Budget;
```

Endpoints

RUTA	MIDDLEWARES	VALIDATION MIDDLEWARES	ERROR MIDDLEWARES	CONTROLLERS
// USERS				
router.get("/",	isLoggedInAPI, isAdmin,			userAPIController.getAll);
router.put("/:id/edit",	isLoggedInAPI, isAdmin,	...validateId(), ...validateUserData(),	Errors.throwErrors,	userAPIController.edit);
router.delete("/:id/remove",	isLoggedInAPI, isAdmin,	...validateId(),	Errors.throwErrors,	userAPIController.remove);
router.get("/:id",	isLoggedInAPI, isAdmin,	...validateId(),	Errors.throwErrors,	userAPIController.getByID);
// TRANSACTIONS				
router.get("/",	isLoggedInAPI,			transactionAPIController.getAll);
router.post("/getCatAndDate",	isLoggedInAPI,	...validateQueryFields(),	Errors.throwErrors,	tr..getByCategoryAndDate);
// GOALS				
router.get("/",	isLoggedInAPI,			goalAPIController.getAll);
router.post("/create",	isLoggedInAPI,	...validateGoalData(),	Errors.throwErrors,	goalAPIController.create);
router.put("/income",	isLoggedInAPI,	...validateIncomeData(),	Errors.throwErrors,	goalAPIController.income);
router.get("/report",	isLoggedInAPI,			goalAPIController.report);
router.put("/:id/edit",	isLoggedInAPI,	...validateId(), ...validateGoalData(),	Errors.throwErrors,	goalAPIController.edit);
router.delete("/:id/remove",	isLoggedInAPI,	...validateId(),	Errors.throwErrors,	goalAPIController.remove);
router.get("/:id",	isLoggedInAPI,	...validateId(),	Errors.throwErrors,	goalAPIController.getByID);
// CATEGORIES				
router.get("/",	isLoggedInAPI,			categoriesAPIController.getAll);
router.get("/:id",	isLoggedInAPI,	...validateId(),	Errors.throwErrors,	categoriesAPIController.getByID);
// BUDGETS				
router.get("/",	isLoggedInAPI,			budgetAPIController.getAll);
router.post("/create",	isLoggedInAPI,	...validateBudgetData(),	Errors.throwErrors,	budgetAPIController.create);
router.get("/report/:id/",	isLoggedInAPI,	...validateId(),	Errors.throwErrors,	budgetAPI..getAnualReport);
router.put("/:id/edit",	isLoggedInAPI,	...validateId(), ...validateBudgetData(),	Errors.throwErrors,	budgetAPIController.edit);
router.delete("/:id/remove",	isLoggedInAPI,	...validateId(),	Errors.throwErrors,	budgetAPIController.remove);
router.get("/:id",	isLoggedInAPI,	...validateId(),	Errors.throwErrors,	budgetAPIController.getByID);
// AUTHENTICATION				
router.post("/register",		...validateUserData(),	Errors.throwErrors,	authApiController.register);
router.post("/login",		...validateLoginData(),	Errors.throwErrors,	authApiController.login);

express-validator

Validación de id y de consulta de movimientos bancarios con express-validator

```
import { check } from "express-validator";
import errors from "../utils/errors.js";
```

```
function validateId() {
    return [

        // Validación de id
        check("id")
            .notEmpty().withMessage(errors.DATA_IS_EMPTY)
            .isInt().withMessage(errors.MUST_BE_NUMBER)
        ];
}

Windsurf: Refactor | Explain | Generate JSDoc | ×
function validateQueryFields() {
    return [
        check("dateInit")
            .notEmpty().withMessage(errors.DATA_IS_EMPTY)
            .isISO8601().withMessage(errors.INVALID_DATE),

        check("dateEnd")
            .notEmpty().withMessage(errors.DATA_IS_EMPTY)
            .isISO8601().withMessage(errors.INVALID_DATE),

        check("idCategory")
            .notEmpty().withMessage(errors.DATA_IS_EMPTY)
            .isInt().withMessage(errors.MUST_BE_NUMBER)
    ];
}
```

express-validator

definición de errores

```
import { validationResult } from "express-validator";


/**
 *  USER ERRORS
 */

const USER_EMAIL_NOT_PROVIDED=100;
Windsurf: Refactor | Explain
class UserEmailNotProvided extends Error {
    Windsurf: Refactor | Explain | Generate JSDoc | ×
    constructor() {
        super("User email not provided");
        this.idError=USER_EMAIL_NOT_PROVIDED;
        this.statusCode = 400;
    }
}

const USER_PASSWORD_NOT_PROVIDED=101;
Windsurf: Refactor | Explain
class UserPasswordNotProvided extends Error {
    Windsurf: Refactor | Explain | Generate JSDoc | ×
    constructor() {
        super("User password not provided");
        this.idError=USER_PASSWORD_NOT_PROVIDED;
        this.statusCode = 400;
    }
}

const USER_EMAIL_ALREADY_EXISTS=102;
Windsurf: Refactor | Explain
class UserEmailAlreadyExists extends Error {
    Windsurf: Refactor | Explain | Generate JSDoc | ×
    constructor() {
        super("User email already exists");
        this.idError=USER_EMAIL_ALREADY_EXISTS;
        this.statusCode = 400;
    }
}

const USER_INVALID_CREDENTIALS=103;
Windsurf: Refactor | Explain
class UserInvalidCredentials extends Error {
```

lanzador de errores

```
function throwErrors(req,res,next) {
    const errores=validationResult(req);

    if (!errores.isEmpty()) {
        // Si hay errores de validación, lanza el primero usando idError
        switch (errores[0].msg) {
            case NAME_NOT_PROVIDED: throw new NameNotProvided();
            case NOT_FOUND: throw new NotFound();
            case DATA_IS_EMPTY: throw new DataIsEmpty();
            case MUST_BE_NUMBER: throw new MustBeNumber();
            case INVALID_FORMAT: throw new InvalidFormat();
            case ALREADY_ACHIEVED: throw new AlreadyAchieved();
            case ALREADY_EXPIRED: throw new AlreadyExpired();
            case USER_EMAIL_NOT_PROVIDED: throw new UserEmailNotProvided();
            case USER_PASSWORD_NOT_PROVIDED: throw new UserPasswordNotProvided();
            case USER_EMAIL_ALREADY_EXISTS: throw new UserEmailAlreadyExists();
            case USER_INVALID_CREDENTIALS: throw new UserInvalidCredentials();
            case USER_PASSWORD_SHORT: throw new UserPasswordShort();
            case USER_PASSWORD_NEEDS_NUMBER: throw new UserPasswordNeedsNumber();
            case USER_PASSWORD_WITHOUT_LETTER: throw new UserPasswordWithoutLetter();
            case USER_IS_NOT_ADMIN: throw new UserIsNotAdmin();
            case USER_NOT_LOGGED_IN: throw new UserNotLoggedIn();
            case INVALID_OR_EXPIRED_TOKEN: throw new InvalidOrExpiredToken();
            case NO_TOKEN_PROVIDED: throw new NoTokenProvided();
            case INVALID_API_KEY: throw new InvalidApiKey();
            case INVALID_DATE: throw new InvalidDate();
            default:
                throw new UnhandledError();
        }
    }
    next();
}
```

Middleware global de gestión de errores

En index.js:

```
//Middleware manejador global de errores
app.use(errorHandler);
```

Middleware:

```
// Middleware global para manejar errores
Windsurf: Refactor | Explain | ×
function errorHandler(err, req, res, next) {
    console.error(err); // Para ver el error en consola
    if (err instanceof Error) {
        return res.status(err.statusCode || 500).json({
            message: err.message,
            statusCode: err.statusCode || 500,
            idError: err.idError
        });
    }

    // Si no es un error personalizado, manejamos de otra forma
    return res.status(500).json({
        message: 'Unhandled error / Server error',
        statusCode: 500
    });
}

export default errorHandler;
```

That's Folks!