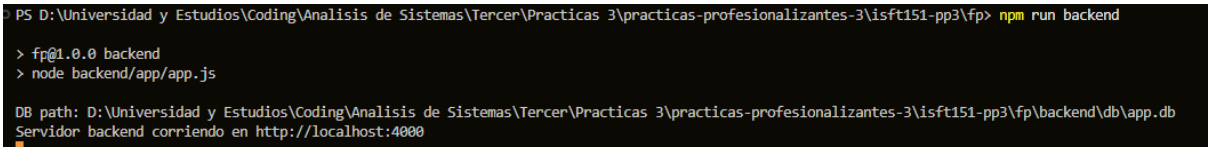
Tres 3 Dos
Documentación de la WebAPI

Base URL: http://localhost:4000
Autenticación: Token de sesión (bycript hash method).
Versión: 1.0.0



POST /users/register → Metodo que registra usuario
Request:
{
  "username": "Leo",
  "password": "Leo!", → Esto se hashea, por lo que en la db se guarda el hash
  "organizationId": 1
}
Validaciones:

- username debe tener mínimo 3 caracteres
- password debe tener una mayúscula y un carácter especial
- si el usuario ya existe, devuelve error

Response: 201
{
  "message": "ok",
  "user": {
        "id": 5,
        "username": "Leo",
        "organizationId": 1,
        "active": 1,
        "created_at": "2025-11-03T 18:00:00Z"
        }
}
Response: 400
{ "error": "Username already exists" }

POST /users/login → Inicia sesión y devuelve token de inicio
{
  "username": "Leo",
  "password": "hash de password"
}
Response: 200
{
  "message": "ok",
  "token": "session-<token>",

```
  "role": "usuario", → Al registrar, por defecto son usuarios
  "organizations": [
    { "id": 1, "name": "Tres 3 Dos" }
  ],
  "organizationId": 1
}
```

Response: 400
```
{ "error": "Invalid credentials" }
```

POST /users/logout → Cierra sesión
Request:
```
{ "token": "session-<token>" }
```

Response: 200
```
{ "message": "Logged out" }
```

Get /users → Devuelve lista de usuarios del sistema (solo permiso de administrador)
Response: 200
```
[
  {
    "id": 1,
    "username": "admin",
    "active": 1,
    "created_at": "<timestamp>",
    "role_id": 1,
    "role_name": "administrador_general",
    "organization_id": 1
  },
  {
    "id": 2,
    "username": "esorero",
    "active": 1,
    "created_at": "<timestamp>",
    "role_id": 3,
    "role_name": "tesorero",
    "organization_id": 1
  }
]
```

PUT /users/:id → actualiza datos de usuario
Request:
```
{
  "username": "Leo2",
  "password": "Leoo!"
}
```
Response 200:
```
{
```

```
  "id": 2,
  "username": "Leo2",
  "active": 1,
  "created_at": "<timestamp>"
}
```

DELETE /users/:id → Elimina usuario por ID
response: 200
```
{ "message": "User deleted" }
```

POST /users/:id/assignRole → Asigna rol de organización
Request:
```
{
  "roleId": 2,
  "organizationId": 1
}
```
Response: 200
```
{
  "message": "Rol asignado correctamente",
  "changes": 1
}
```
Response: 400
```
{ "error": "Debe indicar roleId" }
```

Organizaciones → Endpoint /organizations

GET /organizations → Obtiene todas las organizaciones registradas, solo para admin_general
Response:200
```
[
  { "id": 1, "name": "Tres 3 Dos" },
  { "id": 2, "name": "Banda musical de ciro" }
]
```

Roles disponibles en el sistema:
Get /roles → Lista los roles que hay en el sistema

Response 200:
```
[
  { "id": 1, "name": "administrador_general" },
  { "id": 2, "name": "administrador" },
  { "id": 3, "name": "tesorero" }
]
```

Oncomes: /api/incomes
POST / → incomeController.create → Registra un ingreso válido.
GET / → incomeController.findAll → Lista todos los ingresos.

GET /:id → incomeController.findOne → Obtiene un ingreso por ID.
DELETE /:id → incomeController.delete → Elimina un ingreso.

Payments /api/payments

POST / → paymentController.create → Registra un pago asociado a una deuda.
GET / → paymentController.findAll → Lista todos los pagos registrados.
GET /debt/:debtId → paymentController.findByDebtId → Lista pagos de una deuda específica.
GET /:id → paymentController.findOne → Obtiene un pago por ID.
DELETE /:id → paymentController.delete → Elimina un pago.

Products /api/products
GET / → productController.findAll → Lista todos los productos.
GET /:id → productController.findOne → Obtiene un producto por ID.
POST / → productController.create → Crea un nuevo producto.
PUT /:id → productController.update → Actualiza datos de un producto.
DELETE /:id → productController.delete → Desactiva o elimina un producto.
GET /:id/movements → productController.getStockMovements→ Obtiene movimientos de stock.

Debts /api/debts
POST / → debtController.create → Crea una nueva deuda.
GET / → debtController.findAll → Lista todas las deudas.
GET /:id → debtController.findOne → Obtiene una deuda por ID.
DELETE /:id → debtController.delete → Elimina una deuda.

Events /api/events
GET / → eventController.findAll → Lista todos los eventos.
GET /:id → eventController.findOne → Obtiene un evento por ID.
POST / → eventController.create → Crea un nuevo evento.
PUT /:id → eventController.update → Actualiza un evento existente.
DELETE /:id → eventController.delete → Desactiva un evento.
GET /:id/products → eventController.findProductsByEvent → Lista productos asignados a un evento.
POST /:id/products → eventController.addProductToEvent → Asigna un producto a un evento.
PUT /:eventId/products/:productId → eventController.updateProductEvent → Actualiza asignación.
DELETE /:eventId/products/:productId → eventController.removeProductFromEvent → Quita un producto.

Organizations /api/organizations
GET / → organizationController.findAll → Lista todas las organizaciones.
POST / → organizationController.create → Crea una nueva organización.
GET /:id → organizationController.findOne → Obtiene una organización por ID.
PUT /:id/budget → organizationController.updateBudget → Modifica el presupuesto.

Roles /api/roles
GET / → roleController.findAll → Lista todos los roles disponibles.