PAYMENTS API

DEMO PROJECT

Version of this document: **1.0**
Date of the last update: **24/09/2018**
Author: **Rafal Golarz <web@rafalgolarz.com>**

Table of Contents

1. PROJECT'S SCOPE

The purpose of this project is to present my approach to build APIs in Go.
This documentation describes my demo project of a sample payment API that can:
- fetch a payment resource
- create, update and delete a payment resource
- persists resource state (e.g. database)

2. SETUP / INTEGRATION

This project has been tested on Linux Mint 18.3 XFCE 64b.

Altough the code is built to be not tide to a storage type, it has a concrete implementation for MySQL to check the functionality in more real scenario.

If you don't have a local MySQL database setup, you can use docker (port 3306 should be open and not utililised).

go get github.com/rafalgolarz/payments-demo/cmd/paymentsd

cd $GOPATH:/github.com/rafalgolarz/payments-demo

docker pull mysql/mysql-server

docker run --name=mysql_payments -p3306:3306 -e MYSQL_ROOT_PASSWORD=password -e MYSQL_ALLOW_EMPTY_PASSWORD=yes -d mysql/mysql-server --default-authentication-plugin=mysql_native_password

it may take a couple of seconds, run this to check if the mysql server is up:

docker logs mysql_payments

Let’s create our database first:

docker exec -i mysql_payments mysql -uroot -ppassword -e "create database payments_demo"

then let’s import tables:

docker exec -i mysql_payments mysql -uroot -ppassword payments_demo < ./scripts/payments.sql

Let’s do the same for our test db:

docker exec -i mysql_payments mysql -uroot -ppassword -e "create database payments_demo_test"

then let’s import tables:

docker exec -i mysql_payments mysql -uroot -ppassword payments_demo_test < ./scripts/payments_testdb.sql

Now let’s jump to the console to check if databases exist:

docker exec -ti mysql_payments mysql -uroot -ppassword

Obviously it’s not recommended to pass password in the command line (may stay in the history) but for the sake of simplicity, we can do it that way.

Now time to run our API:

cd $GOPATH:/github.com/rafalgolarz/payments-demo/cmd/paymentsd/

go build .

./paymentsd

You should see something similar to:

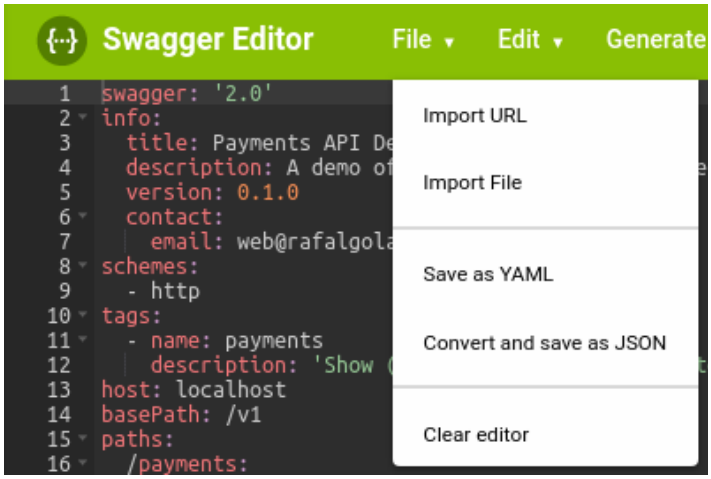
```
[GIN-debug] GET    /v1/payments              --> main.getPayments (3 handlers)
[GIN-debug] GET    /v1/payments/:id          --> main.getPaymentByID (3 handlers)
[GIN-debug] POST   /v1/payments              --> main.addPayment (3 handlers)
[GIN-debug] PUT    /v1/payments/:id          --> main.updatePaymentByID (3 handlers)
[GIN-debug] DELETE /v1/payments/:id          --> main.deletePaymentByID (3 handlers)
[GIN-debug] Listening and serving HTTP on :8080
```

Now open your browser at **http://localhost/v1/payments**

3. API ENDPOINTS

Documentation is stored OpenAPI format.
Please open **https://editor.swagger.io/**

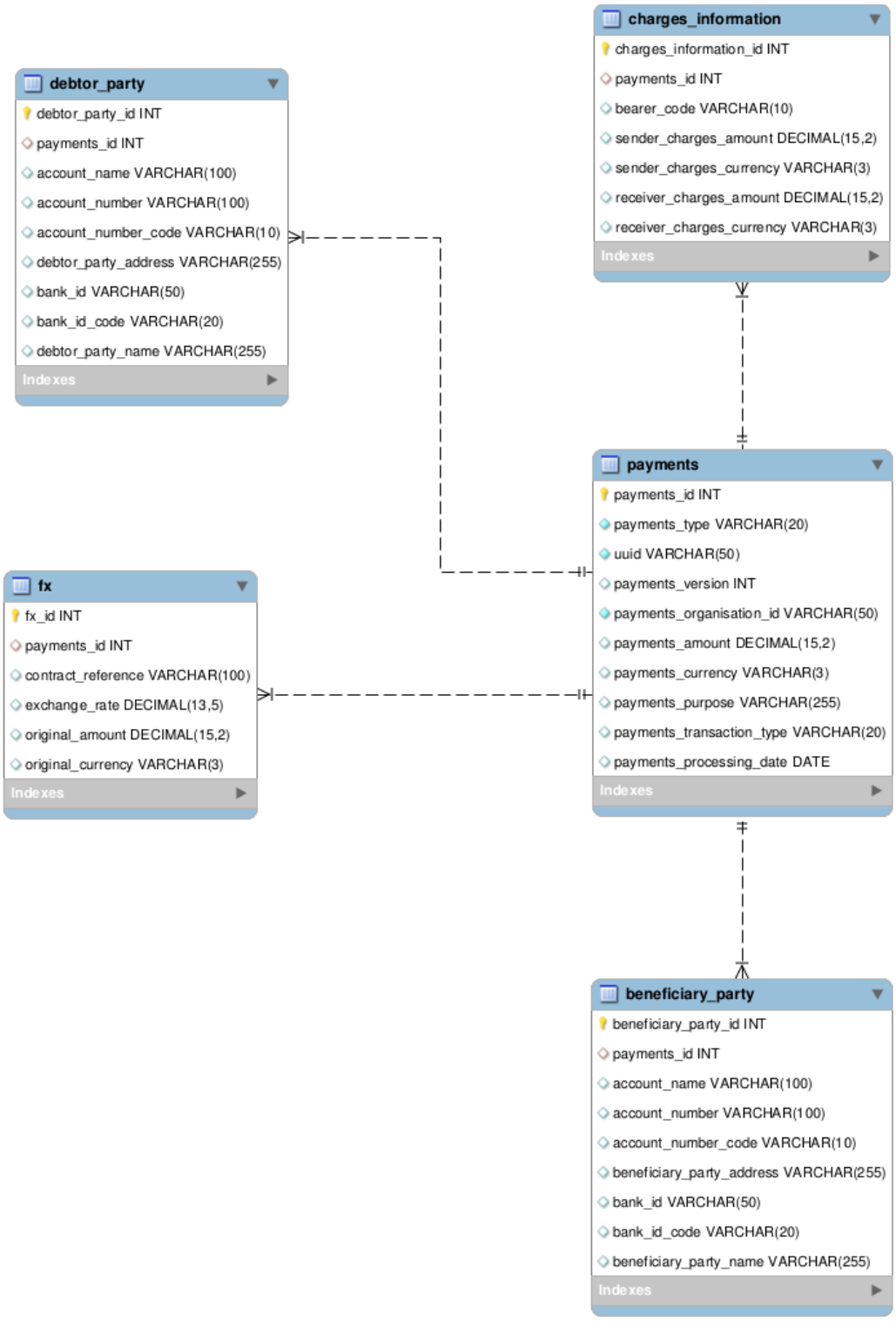
On the top menu select: File→ Import URL



Please upload $GOPATH/github.com/rafalgolarz/payments-demo/docs/**swagger.yml**

For the future integration tests I would use **dredd** to check if swagger documentation is covered by the API.

4. DATA STRUCTURE



5. TESTS

I build tests that check operations on the database (a new copy of the real one is used for such tests), and tests for API endpoints.

cd $GOPATH:/github.com/rafalgolarz/payments-demo
go test ./...

6. SECURITY

For the future development, authorisation should be added to the API. Also dummy DB passwords should not be used and the API should be served over HTTPS only.

7. CODING CONVENTION

7.1 API versioning

Versions of the API are tracked in the API url:

/v1/payments/...
/v2/payments/...

7.2 REST

Lower case. Plural form of nouns representing resources:

/v1/payments/{id}

7.3 JSON

Lower case. Use underscore ("_") to connect words in a single property name.