Documentation

by Omri Kalman

This project was aimed at providing a system for a small coffee shop to manage data about its users, products, and orders. The store sells ground coffee beans, coffee machines, coffee capsules for machines and related products.

The system allows the public to make orders out of a list of products offered by the store, and leave reviews on products. In order to do that, they need to register using their personal details and then log in using the email and password they registered with. They can later keep track of their orders.

Members of the public who have not registered can also see the products that the store has to offer, and the reviews that registered customers have made on those products.

The system also allows for the store's management ('admins') to manage the products that are for sale: add and remove products, edit product details such as price and description, and restock products according to their physical availability.

To use this system, a user needs to make HTTP requests to `http://localhost:3000` + one of the paths derived from below, with the respective HTTP method mentioned,
to access their respective APIs, and interact with the data about the store's current state.

API access:
API HTTP methods:
Your path should look like: API router + API
API routers: Click to see each's APIs:
/api/auth
- /login
- /register
- /resetpw
- /unlock

/api/product

API access:

`public` – Everyone can use this API.

`user` – Logged-in users only (requests with valid tokens) can use this API.

`admin` – Admins only can use this API.

API HTTP methods:

`GET`

`POST`

`PATCH`

`DELETE`

`PUT`

Your `path` should look like: API router + API

API routers: Click to see each's APIs:

- /api/auth
- /api/product
- /api/order
- /api/review

Let's go router by router and explore the APIs.

/api/auth

- **/login** POST public

Lets users who already registered access their account by presenting the correct credentials (email + password)
A user is given 3 attempts to match a password with an email, after which if he gets it wrong again he becomes blocked.

req query: none

req body:

```
{    "email":"name@example.web",    "pw":"Pa$$word123"   }
```

res (success):

```
{
    "result": "success",
    "origin": "model",
    "body": {
        "msg": [
            "Logged in successfully."
        ],
        "token": "..."
    }
}
```

res (error):

401

```
{
    "result": "error",
    "origin": "model",
    "body": {
        "msg": [
            "Incorrect email or password."
                                or
            "You have exceeded the login attempt limit."
        ]
    }
}
```

For more possible errors see Error Guide

- **/register** POST public

Lets anyone on the web create an account.

req query: none

req body:

```
{
     "email":"name@example.web",
     "pw":"Pa$$word123",
     "phone":"0513334444",
     "fn":"John",
     "ln":"Doe",
     "city":"Townsville",
     "address":"123 Main St."
}
```

res (success):

```
{
    "result": "success",
    "origin": "model",
    "body": {
        "msg": [
            "Registered successfully."
        ]
    }
}
```

res (error):

409

```
{
    "result": "error",
    "origin": "model",
    "body": {
        "msg": [
            "Email already exists for another account."
        ]
    }
}
```

For more possible errors see Error Guide

- **/resetpw** PATCH user

Lets logged-in users set their password to a new value, without needing to remember their current password.

req query: none

req body:

```
{
     "pw": "New_pa$$word123"
}
```

res (success):

```
{
    "result": "success",
    "origin": "model",
    "body": {
        "msg": [
            "Password was changed successfully."
        ]
    }
}
```

res (error):

See Error Guide

- **/unlock** PATCH admin

Lets admins give users who got blocked another 3 chances to login, after which they get locked if they fail again.

req query: none

req body:

```
{
     "iduser": 2
}
```

res (success):

```
{
    "result": "success",
    "origin": "model",
    "body": {
        "msg": [
            "User 2 was unlocked successfully."
        ]
    }
}
```

res (error):

404

```
{
    "result": "error",
    "origin": "model",
    "body": {
        "msg": [
            "User 666 was not found."
        ]
    }
}
```

For more possible errors see Error Guide

/api/product

- / GET public

Lets anyone on the web view the products for sale.

req query:

- `min` and `max` pose filtration on price.
- `sort` defines whether cheapest or most expensive products will be listed.

all are optional

```
?name=קולומביאני
&category=ground
&min=50
&max=55
&sort=asc or desc
&page=1
&amount=5
```

req body: none

res (success):

```
{
    "result": "success",
    "origin": "model",
    "body": {
        "products": [
            {
                "idproduct": 1,
                "name": "קולומביאני - בהיר",
                "price": "49.90",
                "category": "ground",
                "stock": 185,
                "desc": "מיקס איכותי",
                "img": "defaultproduct.jpg",
                "hide": 0
            },
            { ... },
            ...
        ]
    }
}
```

res (error):

404

```
{
    "result": "error",
    "origin": "model",
    "body": {
        "msg": [
            "No products were found."
        ]
    }
}
```

For more possible errors see Error Guide

- / POST admin

Lets admins add new products for sale.

req query: none.

req body:

```
{
    "name": "מארז חג",
    "price": 76.60,
    "category": "capsule",
    "stock": 23,
    optional "desc": "lorem ipsum",
    optional "hide": 0
}
```

res (success):

```
{
    "result": "success",
    "origin": "model",
    "body": {
        "msg": [
            "Product was added."
        ]
    }
}
```

res (error):

See Error Guide

- / PATCH admin

Lets admins edit the details of a product.

req query: none

req body:

```
{
     "idproduct": 11,
     "values": {
          all are optional
          "name": "New Name",
          "price": 100.00,
          "category": "capsule",
          "stock": 20,
          "desc": "ipsum lorem",
          "hide": 1
     }
}
```

res (success):

```
{
    "result": "success",
    "origin": "model",
    "body": {
        "msg": [
            "Product was updated."
        ]
    }
}
```

res (error):

404

```
{
    "result": "error",
    "origin": "model",
    "body": {
        "msg": [
            "Product 666 was not found."
        ]
```

```
    }
}
```

For more possible errors see Error Guide

- / DELETE admin

Lets admins completely and unreversably delete a product from the database.
(Notice there is also the option of hiding a product from the public using `PATCH /` and `{ hide: 1 }` as specified in the previous bullet point)

req query: none

req body:

```
{
     "idproduct": 11,
}
```

res (success):

```
{
    "result": "success",
    "origin": "model",
    "body": {
        "msg": [
            "Product was deleted."
        ]
    }
}
```

res (error):

404

```
{
    "result": "error",
    "origin": "model",
    "body": {
        "msg": [
            "Product 666 was not found."
        ]
    }
}
```

For more possible errors see Error Guide

- **/restock** PATCH admin

Lets admins *add to the stock* of specified products, with each getting its own specific quantity.
This is in contrast with `PATCH /` and `{ stock: 19 }` which doesn't *add* but rather *resets* it as the value specified.

req query: none

req body:

```
[
      {
            "idproduct": 3,
            "qty": 24
      },
      {
            "idproduct": 7,
            "qty": 2
      },
      ...
]
```

res (success):

```
{
    "result": "success",
    "origin": "model",
    "body": {
        "msg": [
            "2 products were restocked"
        ]
    }
}
```

res (error):

404

```
{
    "result": "error",
    "origin": "model",
    "body": {
        "msg": [
            "idproduct 666 was not found.",
            "None of the re-stocks were commited."
        ],
```

```
        "idproduct": 666
    }
}
```

For more possible errors see Error Guide

- **/fav** PUT user

Lets logged-in users add a product to their favorites.
Notice a user can't add the same product to his favorite twice and will be encountered with an error.

req query: none

req body:

```
{
     "idproduct": 3
}
```

res (success):

```
{
    "result": "success",
    "origin": "model",
    "body": {
        "msg": [
            "Added to favorites."
        ]
    }
}
```

res (error):

409

```
{
    "result": "error",
    "origin": "model",
    "body": {
        "msg": [
            "Product is already in favorites"
        ]
    }
}
```

For more possible errors see Error Guide

/api/order

- / GET user

Lets logged-in users view their previously-made orders.
Notice: this API doesn't bring the "lines" of the order (products and quantities ordered).
This is done by `GET /line`

req query: none.

req body: none

res (success):

```
{
    "result": "success",
    "origin": "model",
    "body": {
        "orders": [
            {
                "idorder": 19,
                "iduser": 5,
                "datetime": "2023-01-23T17:13:48.000Z",
                "recurring": null
            },
            {
                "idorder": 23,
                "iduser": 5,
                "datetime": "2023-02-24T17:11:08.000Z",
                "recurring": null
            },
            ...
        ]
    }
}
```

res (error):

404

```
{
    "result": "error",
    "origin": "model",
```

```
    "body": {
        "msg": [
            "No orders were found."
        ]
    }
}
```

For more possible errors see Error Guide

- **/line** GET user

Lets logged-in users view the lines of a previously-made order of theirs.

req query:

Specify the order from which you want lines

```
?idorder=3
```

req body: none

res (success):

```
{
    "result": "success",
    "origin": "model",
    "body": {
        "lines": [
            {
                "idorder": 3,
                "idproduct": 2,
                "qty": 2
            },
            {
                "idorder": 3,
                "idproduct": 6,
                "qty": 1
            },
            ...
        ]
    }
}
```

res (error):

404

```
{
```

```
    "result": "error",
    "origin": "model",
    "body": {
        "msg": [
            "No lines were found."
                                or
            "No order was found."
        ],
        "idorder": 666
    }
}
```

For more possible errors see Error Guide

- / POST user

Lets logged-in users place an order.

req query: none

req body:

```
[
     {
          "idproduct": 2,
          "qty": 4
     },
     {
          "idproduct": 666,
          "qty": 4,
          optional "recurring": "m" or "w" or "d"
     },
     ...
]
```

res (success):

```
{
    "result": "success",
    "origin": "model",
    "body": {
        "msg": [
            "Your order was placed."
        ]
```

```
    }
}
```

res (error):

503

```
{
    "result": "error",
    "origin": "inventory",
    "body": {
        "msg": [
            "There is not enough in stock of product 666.",
            "None of the re-stocks were commited."
        ],
        "idproduct": "666"
    }
}
```

404

```
{
    "result": "error",
    "origin": "model",
    "body": {
        "msg": [
            "idproduct 666 was not found.",
            "None of the re-stocks were commited."
        ],
        "idproduct": "666"
    }
}
```

For more possible errors see Error Guide

/api/review

- / GET public

See reviews on products.
Some reviews are comments to other reviews. This is indicated by their idparent value, which contains the idreview their parent review, on which they were commented. These are called **sub-reviews**.

req query:

all are optional

Pagination with:

```
?amount=
&batch=
```

Combine with 1 of the following:
(Sending more than 1 will likely not have the wanted effect)

```
&idreview=
&iduser=
&idproduct=
&idparent=
```

- `idreview` looks for a specific review, whether it's a regular or sub-review.
- `iduser` looks for all reviews made by a user, whether regular or sub-review.
- `idproduct` looks for all **non-sub reviews** made on a product.
- `idparent` looks for all reviews with the specified parent (so all are guaranteed to be sub-reviews)

req body: none

res (success):

```
{
    "result": "success",
    "origin": "model",
    "body": {
        "reviews": [
            {
                "idreview": 1,
                "iduser": 6,
                "idproduct": 1,
                "datetime": "2023-02-24T11:08:58.000Z",
                "rating": 5,
                "content": "Fantastic!",
                "idparent": null
            },
            ...
        ]
    }
}
```

res (error):

404

```
{
    "result": "error",
    "origin": "model",
    "body": {
        "msg": [
            "No reviews were found."
        ]
    }
}
```

For more possible errors see Error Guide

- / POST user

Post a review on a product

req query: none.

req body:

It's possible to post a sub-review (comment made on another, "parent" review) by setting idproduct as the idorder of the parent review.

```
{
     "idproduct": 6,
     "rating": 4,
     optional "content": "lorem ipsum",
     optional "idparent": 666
}
```

res (success):

```
{
    "result": "success",
    "origin": "model",
    "body": {
        "msg": [
            "Review added."
        ]
    }
}
```

res (error):

404

```
{
    "result": "error",
    "origin": "model",
    "body": {
        "msg": [
            "Review made a reference to a user, product, or
parent review which doesn't exist."
        ]
    }
}
```

For more possible errors see Error Guide

Error Guide

It is possible to get many kinds of errors, but all are structured the same way:

```
{
    "result": "error"
    "origin": "..."
    "body":
    {
        "msg": [
            "...",
            ...
        ]
    }
}
```

This same structure is also followed by success messages, but they may or may not have `msg`.

The details for why an error happened are included in `msg` but sometimes are not explicit, for security reasons.

Possible `origin`s are:
`validation | model | middleware | inventory`

- `validation` errors are always given out with status `400`

They happen when the request was not formatted properly.
A typical validation error would look like:

```
{
    "result": "error",
    "origin": "validation",
    "body": {
        "msg": [
            "\"email\" must be a string",
            "\"phone\" is required"
        ]
    }
}
```

- `model` errors are given out with either status `404` or `409` , or `401`

They happen when:

- `404`: A requested resource or action is not found (ex: trying post a review on a product which doesn't exist)
- `409`: A requested resource or action causes a conflict (ex: registeration with a taken email)
- `401`: wrong email or password, or out of attempts, when trying to log in.

- `middleware` errors are given out with either status `400` or `401` or `403`

They indicate an error related to the token.

- `inventory` errors are always given out with status `503`

They indicate when there is not enough in stock of the requested product at the requested quantity.

Response Guide

Non-error responses also have a consistent structure, as follows:

```
{
```

```
    "result": "success",
    "origin": "model",
    "body":
    {
        "msg": [
            "...",
            ...
        ],
        data*: [{...},...] or string (if data is "token")
    }
}
```

"body" is guaranteed to have either "msg" or data, but not necessarily both.
*data is a placeholder, the actual key will be
"orders" or
"reviews" or
"products" or
"lines" or
"token".
depending on which API was used.