📕 Shopping App Doc

About The Project

Language, libraries, tools, and IDEs:

- **Language**
  - Go v1.20
- **Libraries**
  - go-gorm/gorm (ORM library for Golang)
  - jwt.io (JSON Web Tokens)
- **Tools**
  - PostgreSql
  - Docker
  - API platforms
    - Postman
- **IDE's**
  - GoLand (powerful code completion and nice debugging feature) or
  - Visual Studio Code

Installation

1. **Download the codebase**

```
  git clone https://github.com/emrecanbulat/ekinoks-shopping-app.git
```

2. **Fetch dependencies from `go.mod`**

```
go mod download
```

3. **Set `.env` values**

Run following command for generating a `.env` file from `.env.example`

```
cp .env.example .env
```

You will see variables like app variables, PostgreSql credentials in `.env`. You must fill these variables before running the application

> ‼️ **Troubleshooting on the *.env* path**
>
> if you get an error when you run the project that the values in your env file cannot be read;

- Open `cmd/api/main.go` file.
- Please set `godotenv.Load(".env" )` to `godotenv.Load("../../.env")` .

4 . **Run the Project**

```
go run .\cmd\api .
```

After running the following cURL command you should see `I'm OK.` message

`curl --request GET --url localhost:8080/v1/healthcheck`

```
 {
  "message":"I'm OK.",
  "status":"available",
  "system_info":{"environment":"development","version":"1.0.0"}
  }

» This message means your Go server is up. You can also see some system information here
```

Authorization

- **The first time you run the project, the seeder will automatically create an admin account.**

‼️ **NOTE: Seeder also adds fake users and fake products to the database**

```
    email:      "admin@admin.com",
    password:   "password"
```

- **You can log in now. Get a response from the following cURL;**

```
    curl --request POST --location 'localhost:8080/v1/tokens/authentication' \
    --header 'Content-Type: application/json' \
    --data-raw '{
    "email": "admin@admin.com",
    "password": "password"
    }'
```

- **Get “token” from the response and set as the Bearer token in the HTTP HEADERS part. Example;**

```
{
  "Authorization" : "Bearer {{token}}"
}
```

API Usage

- **PRODUCT**

1) Create Product *(this endpoint requires admin access)*

```
  curl --location 'localhost:8080/v1/products' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer {{token}}' \
--data '{
      "title": "title",
      "description": "description",
      "price": 30899,
      "brand": "brand",
      "category": [
          "cat1",
          "cat2"
      ]
}'
```

Response;

```
{
  "product": {
      "id": 22,
      "title": "title",
      "description": "description",
      "price": 30899,
      "brand": "brand",
      "category": [
          "cat1",
          "cat2"
      ]
  }
}
```

‼️ **Adding products requires some validation, otherwise you will get an error like this;**

```
{
    "error": {
        "brand": "must be provided",
        "category": "must be provided",   //should be a string array
        "description": "must be provided",
        "price": "must be provided",
        "title": "must be provided"
    }
}
```

2) Get Product

```
curl --location 'localhost:8080/v1/products/${id}' \
--header 'Authorization: Bearer {{token}}'

```

Response;

```
{
  "product": {
      "id": ${id},
      "title": "title",
      "description": "description",
      "price": 30899,
      "brand": "brand",
      "category": [
          "cat1",
          "cat2"
      ]
  }
}
```

3) Get Product List (filter can be used)

```
curl --location 'localhost:8080/v1/products?title=title&category=cat1&page=1&page_size=1&sort=-price&brand=brand'
--header 'Authorization: Bearer {{token}}'
```

Response;

```
{
  "meta": {
      "current_page": 1,
      "page_size": 1,
      "first_page": 1,
      "last_page": 22,
      "total_records": 22
  },
  "products": [
      {
          "id": 22,
          "title": "title",
          "description": "description",
          "price": 30899,
          "brand": "brand",
          "category": [
              "cat1",
              "cat2"
          ]
      }
  ]
}
```

❗ **You can use filter to list all products**

```
"title", "brand" and "category" are acceptable types for filtering.

example usage =>  localhost:8080/v1/products?title={$title}&category={$cat1}
```

❗ **You can also use sort for the response to return**

```
"id", "title", "price", "brand", "-id", "-title", "-price", "-brand" are acceptable types for sorting.

example usage =>  localhost:8080/v1/products?sort=price

if you want to descending sorting you will only need to use "-".
```

4) Update Product *(this endpoint requires admin access)*

```
curl --location --request PUT 'localhost:8080/v1/products/${id}' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer {{token}}' \
--data '{
      "title": "title (edit)",
      "description": "description (edit)",
      "price": 18899,
      "brand": "brand (edit)",
      "category": [
          "cat1 (edit)",
          "cat 2 (edit)"
      ]
}'
```

Response;

```
{
    "product": {
        "id": ${id},
        "title": "title (edit)",
        "description": "description (edit)",
        "price": 18899,
        "brand": "brand (edit)",
        "category": [
            "cat1 (edit)",
            "cat 2 (edit)"
        ]
    }
  }
```

❗ Update request, like create, require some validation. You are expected to fill in the inputs correctly.

❗ You do not need to fill in all the fields when updating. You just need to change the value you want to update. `For example;`

```
curl --location --request PUT 'localhost:8080/v1/products/${id}' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer {{token}}' \
--data '{
      "title": "updated title",
}'
```

4) Delete Product *(this endpoint requires admin access)*

```
curl --location --request DELETE 'localhost:8080/v1/products/${id}' \
--header 'Authorization: Bearer {{token}}'
```

Response;

```
{
  "message": "product successfully deleted"
}
```

- **USER**

1) User Create & Register

```
curl --location 'localhost:8080/v1/users' \
--header 'Content-Type: application/json' \
--data-raw '{
    "full_name":"User",
    "email":"user@user.com",
    "password":"password",
    "address": "address",
    "phone":"0000000000"
}'
```

Response;

```
{
  "access_token": {
      "expiry": "2023-05-03 15:20:23",
      "token": "{{token}}"
  },
  "user": {
      "id": ${id},
      "full_name": "User",
      "email": "user@user.com",
      "phone": "0000000000",
      "address": "address"
  }
}
```

‼️ There is only one endpoint for user registration. Users can register to the system using this endpoint. At the same time, the admin can create a new user by sending a request to this endpoint. After registering the user is considered logged in and receives a token in response. So you don't need to buy a new token after registration. You can use this token for 24 hours.

- **ORDER**

1) New Order

```
curl --location 'localhost:8080/v1/orders' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer {{token}}' \
--data '{
      "product_id": ${id},
      "payment_type": "Credit_card",
      "amount_paid": 30899
}'
```

Response;

```
{
  "order_details": {
      "amount_paid": 30899,
      "id": 10,
      "order_date": "2023-05-02 15:28:44",
      "payment_type": "Credit_card",
      "status": "processing"
  },
  "product": {
      "brand": "brand",
      "category": [
          "cat1",
          "cat2"
      ],
      "description": "description",
      "price": 30899,
      "title": "title"
  }
}
```

❗ The ordering endpoint also requires validations. You must fill in the id, price and payment method fields correctly.

```
"Cash" and "Credit_card" are acceptable types for payment_type.
```

2) Show Order

```
curl --location 'localhost:8080/v1/orders/${id}' \
--header 'Authorization: Bearer {{token}}' \
--data ''
```

Response;

```
{
  "order_details": {
      "amount_paid": 30899,
      "order_date": "2023-05-02 15:28:44",
      "payment_type": "Credit_card",
      "status": "processing"
  },
  "product": {
      "brand": "brand",
      "category": [
          "cat1",
          "cat2"
      ],
      "description": "description",
      "price": 30899,
      "title": "title"
  },
  "user": {
      "address": "address",
      "email": "admin@admin.com",
      "full_name": "Admin",
      "id": 1,
      "phone": "0000000000"
  }
}
```

3) Order List *(this endpoint requires admin access)*

```
curl --location 'localhost:8080/v1/orders?page_size=1' \
--header 'Authorization: Bearer {{token}}' \
--data ''
```

Response;

```
{
    "meta": {
        "current_page": 1,
        "page_size": 1,
        "first_page": 1,
        "last_page": 10,
        "total_records": 10
    },
    "orders": [
        {
            "id": 1,
            "user": {
                "id": 1,
                "full_name": "Admin",
                "email": "admin@admin.com",
                "phone": "0000000000",
                "address": "address"
            },
            "product": {
                "id": 8,
                "title": "title (edit)",
                "description": "description (edit)",
                "price": 18899,
                "brand": "brand (edit)",
                "category": [
                    "cat1 (edit)",
                    "cat 2 (edit)"
                ]
            },
            "status": 0,
            "payment_type": "Cash",
            "amount_paid": 30899
        }
    ]
}
```

‼️ You can find the required postman collection for testing in the project.

‼️ The project has been developed to run on Docker as well. For this you need to change the "POSTGRES_HOST" field in your .env file.

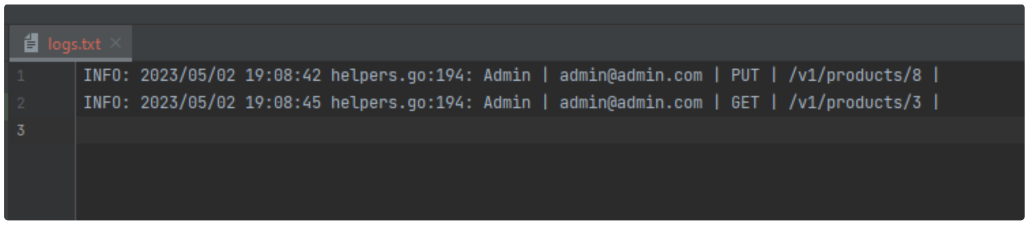
```
#Local development
#POSTGRES_HOST="127.0.0.1"

#Docker
POSTGRES_HOST="host.docker.internal"
```

User Activity(log)

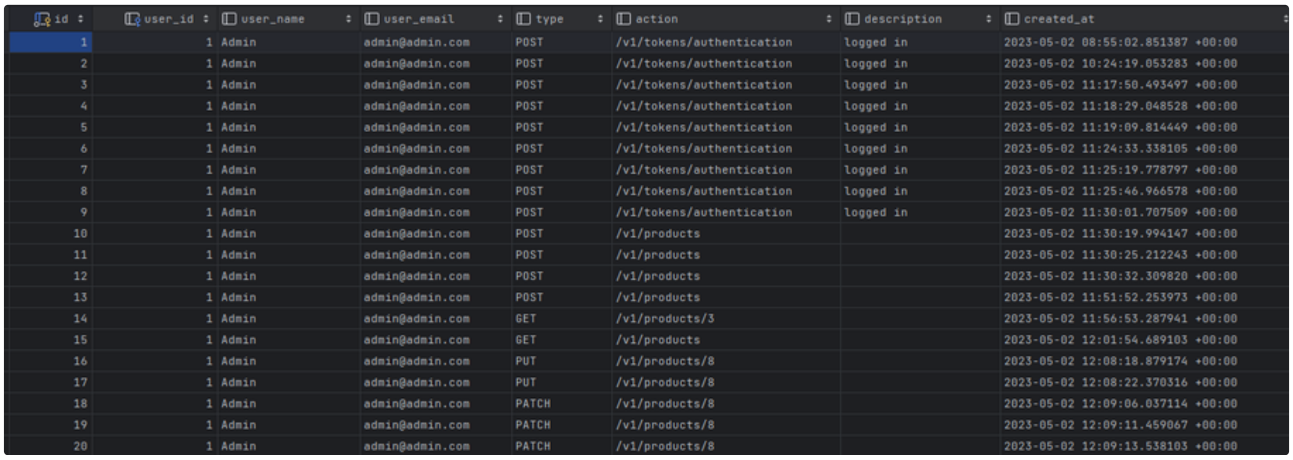
User activities are stored in the **database** and `log.txt` file. No API endpoint has been created for this section.

`log.txt`;



```
INFO: 2023/05/02 19:08:42 helpers.go:194: Admin | admin@admin.com | PUT | /v1/products/8 |
INFO: 2023/05/02 19:08:45 helpers.go:194: Admin | admin@admin.com | GET | /v1/products/3 |
```

`Database`;



id	user_id	user_name	user_email	type	action	description	created_at
1	1	Admin	admin@admin.com	POST	/v1/tokens/authentication	logged in	2023-05-02 08:55:02.851387 +00:00
2	1	Admin	admin@admin.com	POST	/v1/tokens/authentication	logged in	2023-05-02 10:24:19.053283 +00:00
3	1	Admin	admin@admin.com	POST	/v1/tokens/authentication	logged in	2023-05-02 11:17:50.493497 +00:00
4	1	Admin	admin@admin.com	POST	/v1/tokens/authentication	logged in	2023-05-02 11:18:29.048528 +00:00
5	1	Admin	admin@admin.com	POST	/v1/tokens/authentication	logged in	2023-05-02 11:19:09.814449 +00:00
6	1	Admin	admin@admin.com	POST	/v1/tokens/authentication	logged in	2023-05-02 11:24:33.338105 +00:00
7	1	Admin	admin@admin.com	POST	/v1/tokens/authentication	logged in	2023-05-02 11:25:19.778797 +00:00
8	1	Admin	admin@admin.com	POST	/v1/tokens/authentication	logged in	2023-05-02 11:25:46.966578 +00:00
9	1	Admin	admin@admin.com	POST	/v1/tokens/authentication	logged in	2023-05-02 11:30:01.707509 +00:00
10	1	Admin	admin@admin.com	POST	/v1/products		2023-05-02 11:30:19.994147 +00:00
11	1	Admin	admin@admin.com	POST	/v1/products		2023-05-02 11:30:25.212243 +00:00
12	1	Admin	admin@admin.com	POST	/v1/products		2023-05-02 11:30:32.309820 +00:00
13	1	Admin	admin@admin.com	POST	/v1/products		2023-05-02 11:51:52.253973 +00:00
14	1	Admin	admin@admin.com	GET	/v1/products/3		2023-05-02 11:56:53.287941 +00:00
15	1	Admin	admin@admin.com	GET	/v1/products		2023-05-02 12:01:54.689103 +00:00
16	1	Admin	admin@admin.com	PUT	/v1/products/8		2023-05-02 12:08:18.879174 +00:00
17	1	Admin	admin@admin.com	PUT	/v1/products/8		2023-05-02 12:08:22.370316 +00:00
18	1	Admin	admin@admin.com	PATCH	/v1/products/8		2023-05-02 12:09:06.037114 +00:00
19	1	Admin	admin@admin.com	PATCH	/v1/products/8		2023-05-02 12:09:11.459067 +00:00
20	1	Admin	admin@admin.com	PATCH	/v1/products/8		2023-05-02 12:09:13.538103 +00:00