Описание программы небольшого интернет магазина shop.com

А). Описание программы

shop.com — это небольшой интернет магазин.
В магазине есть регистрация и авторизация пользователей магазина. Пользователи магазина могут быть менеджерами или пользователями (покупателями).
Пользователь магазина имеет следующие поля:
account: String
password: String
type: [‘manager’, ‘user’]

Менеджер (‘manager’) может выполнять следующие действия:

- создавать новый товар;
- добавлять N-ое количество товара на склад;
- производить выемку товара со склада;
- видеть все корзины;
- не может добавлять товар в корзину.

Пользователь(‘user’) может выполнять следующие действия:

- добавлять товар в корзину;
- удалять товар из корзины;
- видеть только свою корзину.

Товар имеет следующие поля:
name: String
description: String
count: Integer

В). Описание среды разработки

Оборудование:
description: Computer

- memory size: 7859MiB
- cpu product: Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz
- disk: 238.5G

ОС:

- Ubuntu 20.04 LTS

IDE:

- Visual Studio Code

DB:

- pgAdmin 4

Стек технологий:

- golang
- postgresql

С). Структура программы

Структура программы представлена на рисунке ниже.

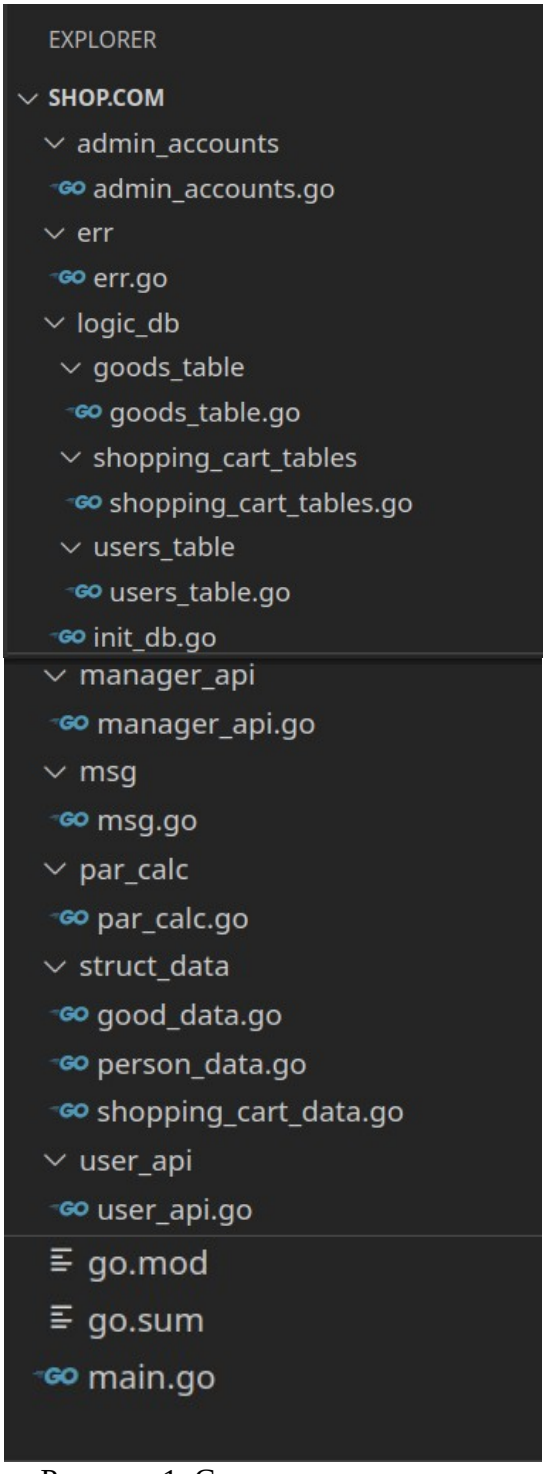

Рисунок 1. Структура программы.

D). Схема БД

Схема БД представлена на рисунке ниже.

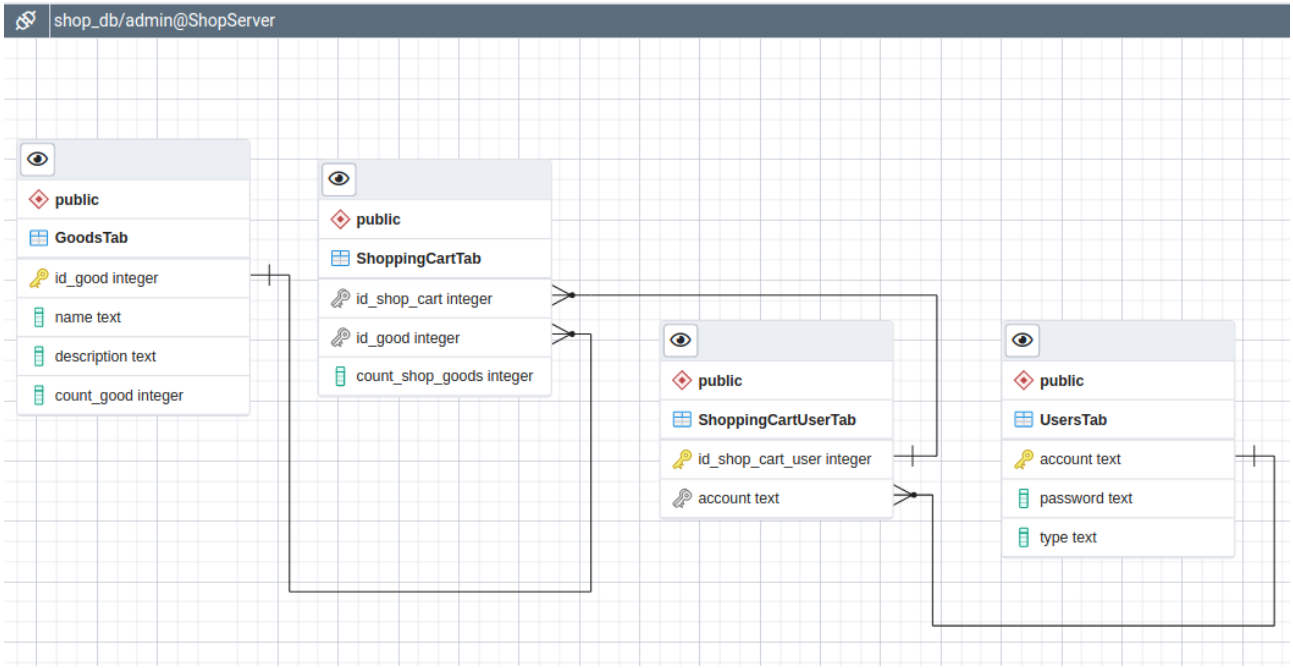

Рисунок 2. Схема БД.

E). Описание инициализации БД

Для соединения с БД используется пакет из *github.com/lib/pq*. Ниже приведены параметры БД для соединения.

```
const (
host = "localhost"
port = 5432
user = "admin"
password = "1234"
dbname = "shop_db"
)
```

Используется пакет *logic_db* для инициализации БД.

F). Описание API

1. API регистрации и аутентификации.
1.1 Регистрация нового пользователя.

```
func RegisterNewAccountApi(db *sql.DB, account, password, typePerson string)
```

На вход подается проинициализированная БД, имя аккаунта, пароль и тип аккаунта (‘manager’ или ‘user’). Имя аккаунта должно быть уникальным.
В БД запишется новый аккаунт. Если в БД уже есть такой аккаунта, то выведется сообщение об этом.

Пример вызова:
```
admin_accounts.RegisterNewAccountApi(db, "NewManager", "98765", "manager")
admin_accounts.RegisterNewAccountApi(db, "NewUser", "qwerty", "user")
```

1.2 Обновление пароля существующего аккаунта.
```
func UpdatePasswordByAccountApi(db *sql.DB, account, password string)
```
На вход подается проинициализированная БД, имя аккаунта и новый пароль. В БД обновится пароль по введенному аккаунту.
Пример вызова:
```
admin_accounts.UpdatePasswordByAccountApi(db, "NewUser", "NEWPASS")
```

1.3 Удаление существующего аккаунта.
```
func DeleteAccountApi(db *sql.DB, account string)
```
На вход подается проинициализированная БД и имя аккаунта. В БД удалится данные аккаунта.
Пример вызова:
```
admin_accounts.DeleteAccountApi(db, "NewUser12")
```

1.4 Авторизация аккаунта.
```
func AuthorizationByAccountAndPasswordApi(db *sql.DB, account, password string)
(string, bool)
```
На вход подается проинициализированная БД, имя аккаунта и пароль. Если не будет найден аккаунт или будет введен пароль, несовпадающий с сохраненным паролем в БД, то выводится соответствующие сообщения. В случае успеха, выведется сообщение об успешной авторизации.
На выходе вернется тип пользователя (‘manager’ или ‘user’) и true в случае успеха.
Пример вызова:
```
typePerson, result := admin_accounts.AuthorizationByAccountAndPasswordApi(db,
"NewManager", "98765")
fmt.Println(typePerson, "", result)
```
Вывод:
MessageAPI:
Successful account authorization.
manager true

1.5 Получения списка всех аккаунтов.
```
func GetAllAccountsApi(db *sql.DB)
```
На вход подается проинициализированная БД.
На выходе будет список аккаунтов.
Пример вызова:
```
persons := admin_accounts.GetAllAccountsApi(db)
fmt.Println(persons)
```
Вывод:
[{Bob 123456 manager} {Bob2 123456 manager} {Bob4 12 user} {Eric werr user} {NewManager12 98765 manager} {NewManager 98765 manager} {NewUser NEWPASS user}]

2. API менеджера.
API менеджера доступно к работе после авторизации и проверки типа пользователя.
2.1 Создание нового товара на склад.
```
func AddNewGoodsApi(db *sql.DB, name, description string, count uint32)
```

На вход подается проинициализированная БД, именование товара, описание товара и количество товара. В БД добавится новый товар. Именование товара не уникальное имя, т к может быть много товара с одинаковым именованием. В качестве уникального поля для товаров используется ИД товара (код товара).
Пример вызова:
```
manager_api.AddNewGoodsApi(db, "NewGoods1", "Description NewGoods1", 200)
```

2.2 Обновление количества товара на складе.
```
func UpdateCountGoodsByIdAndNameApi(db *sql.DB, id uint32, name string, count uint32)
```
На вход подается проинициализированная БД, ИД товара, именование товара и количество товара, которое будет обновлено. В БД заменится количество товара на складе на новое из функции.
Пример вызова:
```
manager_api.UpdateCountGoodsByIdAndNameApi(db, 10, "NewGoods1", 273)
```

2.3 Добавление товара на склад.
```
func AddCountGoodsByIdAndNameApi(db *sql.DB, id uint32, name string, adding_count
uint32)
```
На вход подается проинициализированная БД, ИД товара, именование товара и количество товара, которое нужно добавить на склад. В БД добавится количество товара на складе к существуещему на складе.
Пример вызова:
```
manager_api.AddCountGoodsByIdAndNameApi(db, 10, "NewGoods1", 42)
```

2.4 Выемка определенного количества товара со склада.
```
func DeleteCountGoodsByIdAndNameApi(db *sql.DB, id uint32, name string,
deleting_count uint32) uint32
```
На вход подается проинициализированная БД, ИД товара, именование товара и количество товара, которое нужно изъять со склада. В БД изымается количество товара на складе от существующего на складе. Если товара на складе не хватит для изъятия, то изымается товара столько сколько есть на складе, при этом выводится сообщение о том, что вы хотите изъять столько то товара, а изъято товара столько то со склада и что невозможно изъять столько то.
Пример вызова:
```
manager_api.DeleteCountGoodsByIdAndNameApi(db, 10, "NewGoods1", 42)
```
Вывод сообщения:
You want to pick up the goods in the amount of 277, but there are only 273 such goods in stock. It will be delete from the stock of goods in the amount of 273. And 4 goods are out of stock

2.5 Получение информации о товаре.
```
func GetGoodsByIdAndNameApi(db *sql.DB, id uint32, name string) struct_data.Goods
```
На вход подается проинициализированная БД, ИД товара, именование товара.
На выходе получится структура с описание товара: ИД товара, именование товара, описание товара и количество на складе.
Пример вызова:
```
goods := manager_api.GetGoodsByIdAndNameApi(db, 9, "NewGoods1")
fmt.Println(goods)
```
Вывод:
{9 NewGoods1 Description NewGoods1 200}

2.6 Полная выемка товара со склада.
```
func DeleteGoodsByIdAndNameApi(db *sql.DB, id uint32, name string)
```
На вход подается проинициализированная БД, ИД товара, именование товара.
Из БД полность удалится запись о товаре.
Пример вызова:
```
manager_api.DeleteGoodsByIdAndNameApi(db, 9, "NewGoods1")
```

2.7 Получения списка всех товаров.
```
func GetAllGoodsApi(db *sql.DB) []struct_data.Goods {
```
На вход подается проинициализированная БД.
На выходе выведется полный список товаров, хранящихся на складе.
Пример вызова:
```
listGoods := manager_api.GetAllGoodsApi(db)
fmt.Println(listGoods)
```
Вывод:
[{1 good_one desc_one 14} {3 ff fdf 0} {5 34 34 3} {4 NewGoods1 Description NewGoods1 200} {6 NewGoods3 Description NewGoods3 56} {7 NewGoods1 Description NewGoods1 200} {8 NewGoods1 Description NewGoods1 200} {10 NewGoods1 Description NewGoods1 0}]

2.8 Удаление записей товаров со склада, если они закончились.
```
func DeleteGoodsIfCountGoodsEqualZeroApi(db *sql.DB)
```
На вход подается проинициализированная БД.
В БД обновятся записи на складе и удаляттся те товары, у которых количество равно 0.
Примечание. Функционал не позволит удалить записи товара, количество которых равно 0, но при этом они у кого-то из пользователей в корзине. Теоретически пользователь может вернуть товара на склад, отказавшись покупать или выбрав иное количество этого товара. В такой ситуации выводится ошибка.
Пример вызова:
```
manager_api.DeleteGoodsIfCountGoodsEqualZeroApi(db)
```

2.9 Просмотр всех корзин.
```
func GetAllShoppingCartsApi(db *sql.DB) []struct_data.ShoppingCartForOneUser
```
На вход подается проинициализированная БД.
На выходе выводятся все корзины со следующей структурой. Т е будет список корзин каждого пользователя, где одна корзина представлена следующим образом:
```
type ShoppingCartForOneUser struct {
Id uint32
NameAccount string
Goods []Goods
}
```

Id - ИД корзины, NameAccount - именование аккаунта пользователя, Goods - список товаров.

```
type Goods struct {
Id uint32
Name string
Description string
Count uint32
}
```

Id - ИД товара, Name - именование товара, Description — описание товара, Count - количество купленного товара.

Пример вызова:
```
shoppingCartForManyUser := manager_api.GetAllShoppingCartsApi(db)
fmt.Println(shoppingCartForManyUser)
```
Вывод:
[{2 Bob [{1 good_one desc_one 0}]} {9 Bob4 [{1 good_one desc_one 27} {3 ff fdf 3}]} {12 Eric [{1 good_one desc_one 7}]}]

3. API пользователя.
API пользователя доступно к работе после авторизации и проверки типа пользователя.

3.1 Создание новой корзины
```
func CreateNewShoppingCartApi(db *sql.DB, account string)
```
На вход подается проинициализированная БД и именование аккаунта.
Создается новая корзина в БД.
Пример вызова:
```
user_api.CreateNewShoppingCartApi(db, "Alice")
```

3.2 Добавление нового товара в корзину.
```
func AddNewGoodsIntoShoppingCartApi(db *sql.DB, id_shopping_cart, id_goods,
count_goods uint32)
```
На вход подается проинициализированная БД, ИД корзины, ИД товара и количество добавляемого товара.
В БД и в корзину добавятся товары с заданным количество, а если заданное количество больше количества товаров, расположенных на складе, тогда в корзину поместится столько товара, сколько хранится на складе. При этом выводится сообщение о том, что пользователь хотел добавить в корзину столько то товаров, но на складе меньше товаров, поэтому добавится в корзину товаров столько, сколько на складе, а остальное не добавится в корзину.
Пример вызова:
```
user_api.AddNewGoodsIntoShoppingCartApi(db, 13, 6, 5)
```
и
```
user_api.AddNewGoodsIntoShoppingCartApi(db, 13, 6, 55)
```
Вывод:
MessageAPI:
You want to pick up the goods in the amount of 55, but there are only 51 such goods in stock. It will be delete from the stock of goods in the amount of 51. And 4 goods are out of stock

3.3 Обновление количества товаров в корзине.
```
func UpdateCountGoodsInShoppingCartApi(db *sql.DB, id_shopping_cart, id_goods,
count_goods uint32)
```
На вход подается проинициализированная БД, ИД корзины, ИД товара и количество товара (в большую или меньшую сторону относительно текущего количества товаров в корзине).
В БД и в корзине обновятся товары с заданным количество, а если заданное количество больше количества товаров, расположенных на складе, тогда в корзину поместится столько товара, сколько хранится на складе. При этом выводится сообщение о том, что пользователь

хотел добавить в корзину столько то товаров, но на складе меньше товаров, поэтому добавится в корзину товаров столько, сколько на складе, а остальное не добавится в корзину.
Если количество товара увеличивается в корзине, то на складе количество этого товара уменьшается.
Если количество товара уменьшается в корзине, то на складе происходит возврат товара, его количество увеличивается.
Пример вызова:
```
user_api.UpdateCountGoodsInShoppingCartApi(db, 9, 1, 2)
```
и
```
user_api.UpdateCountGoodsInShoppingCartApi(db, 9, 1, 100)
```
Вывод:
MessageAPI:
You want to pick up the goods in the amount of 100, but there are only 39 such goods in stock. It will be delete from the stock of goods in the amount of 39. And 58 goods are out of stock

3.4 Удаление товаров из корзины.
```
func DeleteGoodsByIdShoppingCartAndIdGoodsApi(db *sql.DB, id_shopping_cart, id_goods uint32)
```
На вход подается проинициализированная БД, ИД корзины и ИД товара.
Из БД и корзины удалится товар с ИД товаром. Когда товар удаляется из корзины, этот товар возвращается на склад в том количестве, в котором он находился в корзине.
Пример вызова:
```
user_api.DeleteGoodsByIdShoppingCartAndIdGoodsApi(db, 13, 6)
```

3.5 Получение всех товаров из одной корзины.
```
func GetAllGoodsFromShoppingCartByIdApi(db *sql.DB, id_shopping_cart uint32) struct_data.ShoppingCartForOneUser
```
На вход подается проинициализированная БД и ИД корзины.
На выходе выводятся все товары из одной корзины со структурой, представленной выше при выводе всех корзин менеджером. Т е будет список товаров одной корзины.
Пример вызова:
```
shoppingCartForOneUser := user_api.GetAllGoodsFromShoppingCartByIdApi(db, 9)
fmt.Println(shoppingCartForOneUser)
```
Вывод:
{9 Bob4 [{3 3 goods good goods 3 3} {1 good_one desc_one 42}]}

3.6 Многократный запуск получения всех товаров из одной корзины.
```
func RunMultipleDbAccessesApi(db *sql.DB, number_of_requests int)
```
На вход подается проинициализированная БД и количество одновременный обращений к БД.
На выходе выводятся все товары из одной корзины для заданного числа раз вызова. Вывод осуществляется вызовом в горутине *GetAllGoodsFromShoppingCartByIdApi*.
Пример вызова:
```
par_calc.RunMultipleDbAccessesApi(db, 100)
```
Вывод:
{9 Bob4 [{3 3_goods good_goods_3 3} {1 good_one desc_one 42}]}
{9 Bob4 [{3 3_goods good_goods_3 3} {1 good_one desc_one 42}]}
{9 Bob4 [{3 3_goods good_goods_3 3} {1 good_one desc_one 42}]}
...
{9 Bob4 [{3 3_goods good_goods_3 3} {1 good_one desc_one 42}]}
{9 Bob4 [{3 3_goods good_goods_3 3} {1 good_one desc_one 42}]}

G). Описание вспомогательного функционала

В программе используется дополнительный функционал для вывода ошибок, сообщений, работы непосредственно с запросами к БД. Это такие пакеты как *msg, err, goods_table, shopping_cart_tables, users_table, struct_data*.

Пакет msg.
Пример функции вывода сообщений:

```
func MsgAboutIncorrectAccount() {
	MsgApi()
	fmt.Println("Account with this naming already exists. Please try a different account name.")
}
```

Пакет err.
Пример функции вывода ошибки:

```
func PrintError(err error) {
	if err != nil {
		panic(err)
	}
}
```