FACULTATEA DE AUTOMATICA SI CALCULATOARE BUCURESTI

# DELICE CAKE

# SHOP

# DOCUMENTATIE

PROIECT REALIZAT DE ALICE FLORENTA SUIU, BOGDAN CONSTANTIN NUTU SI CONSTANTIN RADUCANU

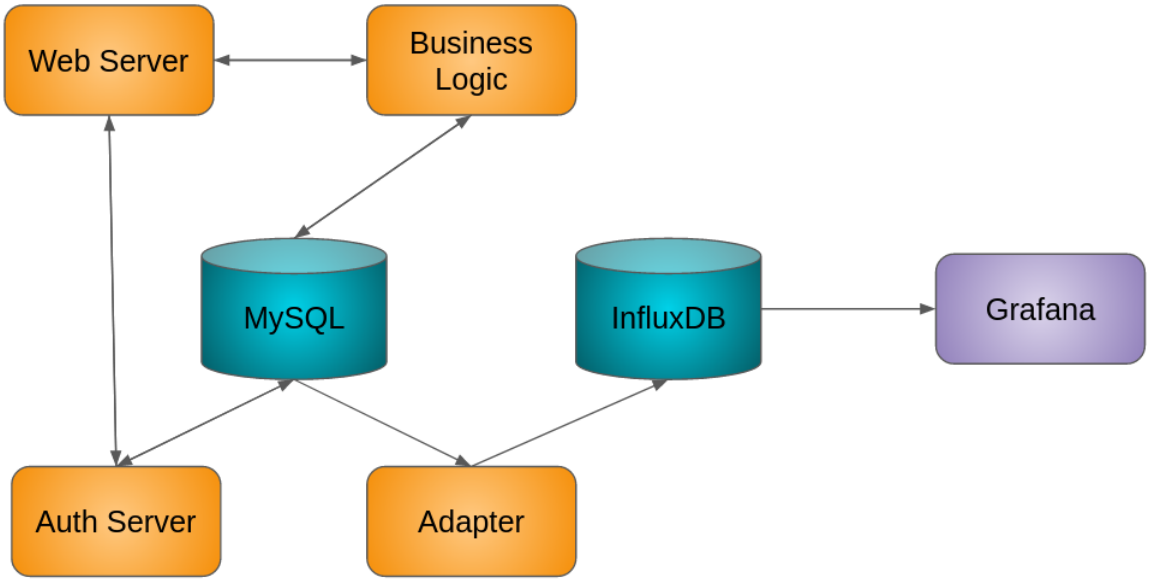
# 1. Introducere

Proiectul consta într-o aplicație de gestiune a unei cofetarii online. Astfel, aplicatia este formata din urmatoarele componente:

- baza de date create in MySQL
- un server pentru backend, pentru partea de business logic, creat in Python 3 cu biblioteca Flask
- un server Web creat cu Apache HTTP server ce expune o interfata grafica realizata cu Vanilla JavaScript si HTML.
- un serviciu de monitorizare a cofetariei realizat cu Grafana
- baza de date creata in InfluxDB
- un adapter scris în Python 3, ce preia date din baza de date creata in MySQL, le prelucrează și le stochează în baza de date create in InfluxDB, aceste date fiind folosite de serviciul de monitorizare
- un server de autentificare realizat în Python 3 cu biblioteca Flask

Fiecare microserviciu al aplicației va fi reprezentat de cate un container, iar intreaga aplicatie va fi asamblata intr-un docker-compose.

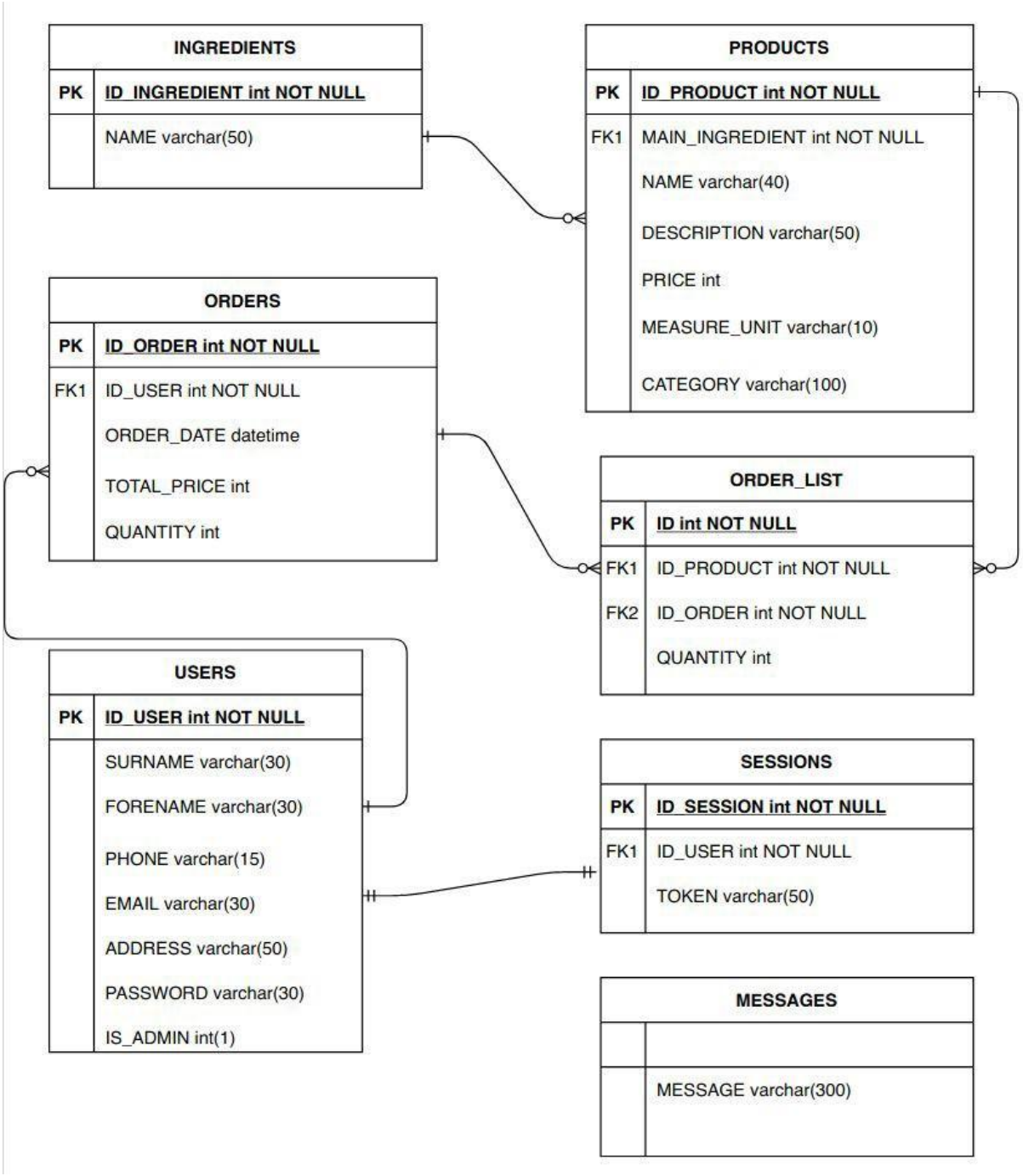
## 2. Diagrama arhitecturala a aplicatiei




## 3. Descrierea componentelor

**Baza de date in MySQL** este formată dintr-o serie de tabele ce reține datele clienților/userilor, ale comenzilor, produselor, lista cu ingredientele principale din fiecare produs, precum și un table în care se salvează sesiunea de conectare a unui client și un tabel în care sunt logate mesajele.

## Diagrama bazei de date



**Descrierea tabelelor**

a) **USERS**

Acest tabel contine informatiile necesare despre un user al aplicatiei. In momentul in care se creeaza un cont nou de user din aplicatia web, se insereaza in tabel datele furnizate. Este important ca userul sa completeze o **adresa de e-mail** si o **parola**, credentiale ce vor fi folosite la logarea in aplicatie.

Exista doua categorii de utilizatori: **administratorul** care poate sa adauge produse, ingrediente, comenzi si poate vizualiza comenzile tuturor utilizatorilor, si **clientul obisnuit** ce poate vizualiza produsele, ingredientele existente si poate plasa comenzi.

b) **INGREDIENTS**

Acest tabel contine denumirea unui ingredient principal ce intra in componenta unui produs.

c) **PRODUCTS**

Acest tabel contine date despre produsele existente in cofetarie. Caracteristicile unui produs: denumire, descriere, pret, unitate de masura, categoria produsului (mini-prajitura, placinta, tort) si ingredientul principal, acesta este dat de id-ul ingredientului.

Cu ajutorul id-ului de la ingredient s-a realizat o cheie straina ce face legatura cu tabela de ingrediente.

d) **ORDERS**

Acest tabel contine date despre comenzile efectuate de user. Caracteristicile unei comenzi sunt: data efectuarii comenzii, pretul total al comenzii calculate dinamic, cantitatea produsului selectat pentru comanda, precum si userul care a creat comanda.

In tabela este salvat id-ul userului care a efectuat comanda. Cu ajutorul acestuia s-a creat constrangerea de cheie straina.

e) **ORDER_LIST**

Acest tabel are la baza o entitate de intersectie care contine cele doua legaturi straine realizate prin ID_ORDER si ID_PRODUCT, avand un ID propriu pentru a simplifica lucrurile.

f) **SESSIONS**

La logarea in aplicatie sau la crearea unui cont i se asociaza user-ului un token – string ce denota faptul ca acesta este autentificat in aplicatie. Acest token este salvat in tabela de sesiuni. Fiecarui user i se asociaza un unic token.

La deconectare se sterge tokenul aferent userului din baza de date.

Tabela contine un ID_USER pentru a face legatura straina cu tabela de useri. De asemenea, tabela contine si un ID unic per inregistrare.

Baza de date **cake_shop** impreuna cu tabelele din diagrama de mai sus au fost create si initializate cu ajutorul unui script. Totodata, in acest script am definit si o serie de **proceduri stocate**, folosite de serverul de backend și de cel de autentificare pentru a interactiona cu baza de date, și o serie de **triggere**.

Exemplu de **procedura stocata** definita:

```
CREATE PROCEDURE group_products_ingr()
BEGIN
    SELECT A.NAME, COUNT(B.NAME) NR_PROD
    FROM INGREDIENTS A, PRODUCTS B
    WHERE B.MAIN_INGREDIENT = A.ID_INGREDIENT
    GROUP BY A.NAME;
END $$
```

Procedura din imagine grupeaza produsele din baza de date in functie de ingredientul principal. Aceasta este folosita pentru realizarea unui raport cu privire la produsele care contin un anumit ingredient principal selectat de catre utilizator.

Exemplu de **trigger** definit:

```
CREATE TRIGGER TGR_ORDER_LIST
    AFTER INSERT ON ORDER_LIST
    FOR EACH ROW
BEGIN
    DECLARE QUANTITY_CMD, PRICE_PROD INT;

    SELECT QUANTITY INTO QUANTITY_CMD
    FROM ORDERS
    WHERE ID_ORDER = NEW.ID_ORDER;

    SELECT PRICE INTO PRICE_PROD
    FROM PRODUCTS
    WHERE ID_PRODUCT = NEW.ID_PRODUCT;

    UPDATE ORDERS SET TOTAL_PRICE = QUANTITY_CMD * PRICE_PROD
    WHERE ID_ORDER = NEW.ID_ORDER;

    INSERT INTO MESSAGES(MESSAGE) VALUES(CONCAT('S-a inserat o comanda in lista de comenzi!
        Comanda are id-ul: ', NEW.ID_ORDER));

END $$
```

Acest trigger ajuta la calculul pretului total al unei comenzi.

Pentru crearea microserviciului ce reprezinta baza de date, am creat un container cu versiunea 8.0.22 a imaginii de MySQL de pe Docker Hub. Serviciul obtinut este unul persistent, datele pastrandu-se in tabele de la o rulare la alta.

**Business Logic** consta dintr-un server scris in Python ce expune numeroase endpoint-uri folosite de catre aplicatia Web. Pentru fiecare tabel exista doua endpoint-uri de baza – unul pentru **cereri de tip GET** care selecteaza elementele din baza de date si le transmite in format JSON catre Web si unul pentru **cereri de tip POST** care adauga datele primite de la Web in baza de date in tabelul corespunzator.

Exemplu de functie ce faciliteaza o **cerere de tip GET** pentru afisarea in pagina Web a ingredientelor principale disponibile in produsele din cofetarie.

```
@app.route('/ingredients', methods=['GET'])
def get_ingredients():
    connection = mysql.connector.connect(**config)
    response = {}
    try:
        cursor = connection.cursor()
        cursor.callproc('select_all_ingredients')
        ingredients = []
        result = []
        for row in cursor.stored_results():
            result = row.fetchall()
        for id, name in result:
            ingredient = {}
            ingredient['ID_INGREDIENT'] = int(id)
            ingredient['NAME'] = name
            ingredients.append(ingredient)

        response["data"] = ingredients
        cursor.close()
        connection.close()
        response["status"] = SUCCESS
    except:
        response["status"] = SERVER_ERROR
    return json.dumps(response)
```

Exemplu de functie ce faciliteaza o **cerere de tip POST** pentru adaugarea in baza de date a unui ingredient principal nou, adaugare realizata prin completarea unui formular din pagina Web.

```
@app.route('/ingredients', methods=['POST'])
def add_ingredient():
    connection = mysql.connector.connect(**config)
    response = {}
    try:
        cursor = connection.cursor()
        data = request.json
        name = data['NAME']
        cursor.callproc('add_ingredient', [name])
        cursor.close()
        connection.close()
        response['status'] = SUCCESS
    except:
        response['status'] = SERVER_ERROR
    return json.dumps(response)
```

Conexiunea cu baza de date se face cu ajutorul bibliotecii **mysql.connector** din **Python**, iar conexiunea cu aplicatia Web se face cu ajutorul adresei ip '**0.0.0.0**' si a portului **5000** expus de server.

Pentru construirea rapoartelor si a graficelor, serverul expune cate un endpoint pentru fiecare filtrare realizata. De exemplu, endpointul /group_products_ingr faciliteaza realizarea unui raport cu privire la produsele care contin un anumit ingredient principal selectat de catre utilizator.

```python
@app.route('/group_products_ingr', methods=['GET'])
def group_products_ingr():
    connection = mysql.connector.connect(**config)
    response = {}
    try:
        cursor = connection.cursor()
        cursor.callproc('group_products_ingr')
        res = []
        results = []
        for row in cursor.stored_results():
            res = row.fetchall()
        for name, nr_prod in res:
            result = {}
            result['NAME'] = name
            result['NR_PROD'] = int(nr_prod)
            results.append(result)
        cursor.close()
        connection.close()
        response["data"] = results
        response['status'] = SUCCESS
    except:
        response["status"] = SERVER_ERROR
    return json.dumps(response)
```

Pentru a trimite date de la server la aplicatia Web si invers am folosit formatul JSON. Exemplul din imagine arata mesajul primit de catre aplicatia Web in momentul in care s-a facut o **cerere GET** pe toate produsele grupate in functie de categorie.

```
{
    "data": [
        {
            "CATEGORY": "cake",
            "NR_PROD": 3
        },
        {
            "CATEGORY": "mini-cake",
            "NR_PROD": 4
        },
        {
            "CATEGORY": "pie",
            "NR_PROD": 3
        }
    ],
    "status": 200
}
```


Pentru crearea microserviciului ce reprezinta serverul de backend, am creat un container cu o imagine de Python 3.6. Serviciul obtinut comunica cu serviciul reprezentat de baza de date MySQL si cu serviciul reprezentat de serverul Web.

**Componenta de autentificare** consta dintr-un server scris in Python 3 cu ajutorul bibliotecii Flask ce expune doua endpoint-uri unul pentru **login** si unul pentru **logout**.

La logarea in aplicatie sau la crearea unui cont i se asociaza utilizatorului un token – string ce denota faptul ca acesta este autentificat in aplicatie. Acest token este salvat in tabela de sesiuni. Fiecarui user i se asociaza un unic token. Existenta acestui token valideaza faptul ca utilizatorul este conectat in aplicatie si astfel acesta poate executa operatii.

La deconectare se sterge tokenul aferent utilizatorului din baza de date.

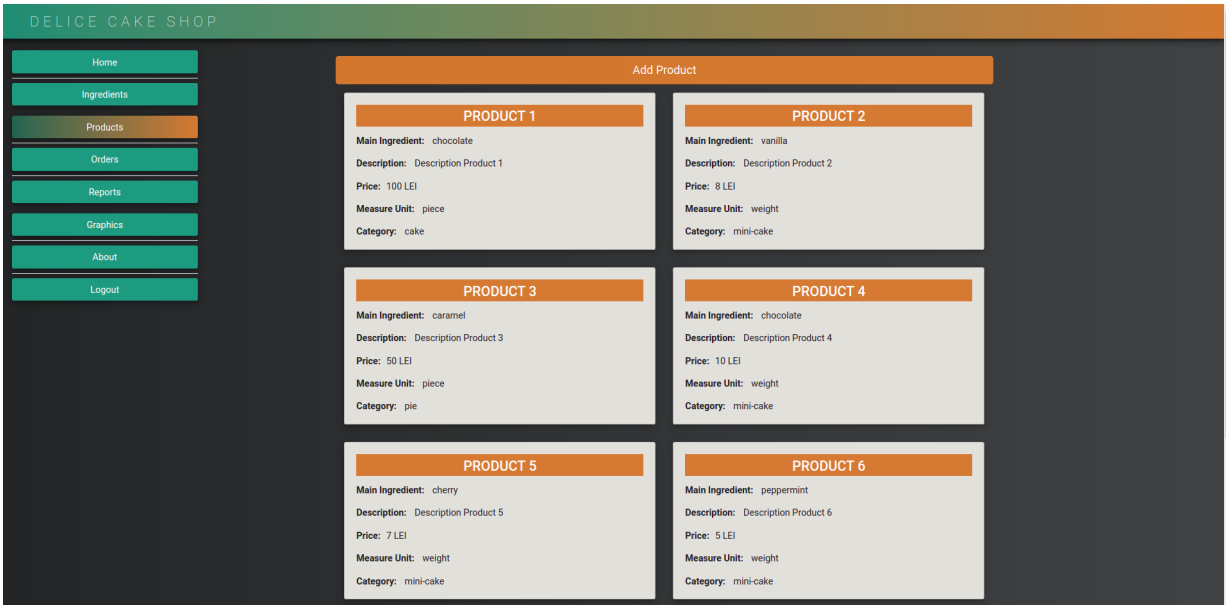
```python
@app.route('/logout/<token>', methods=['POST'])
def logout(token):
    connection = mysql.connector.connect(**config)
    response = {}
    try:
        cursor = connection.cursor()
        print("logout " + token)
        cursor.callproc('delete_session_token', [token])
        cursor.close()
        connection.close()
        response['status'] = SUCCESS
    except:
        response["status"] = SERVER_ERROR
    return json.dumps(response)
```

Pentru crearea microserviciului ce reprezinta serverul de autentificare, am creat un container cu o imagine de Python 3.6. Serviciul obtinut comunica cu serviciul reprezentat de baza de date MySQL si cu serviciul reprezentat de serverul Web.

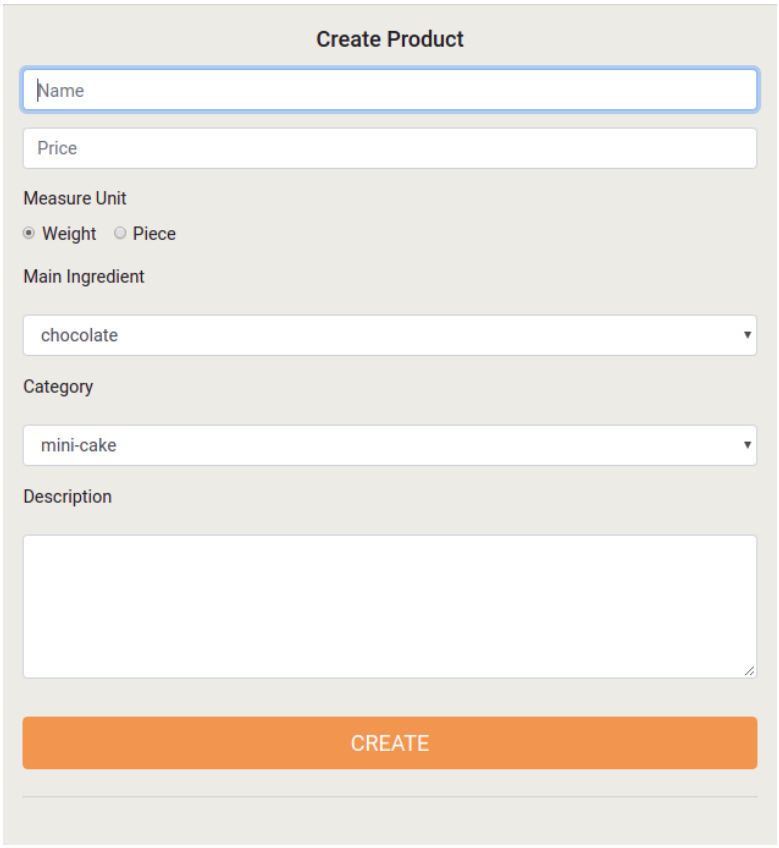
**Serverul Web** creat cu **Apache HTTP server** expune o interfata grafica realizata cu **Vanilla JavaScript**, **HTML** si **CSS**.

Aplicatia Web prezinta urmatoarelor pagini:

- ✓ **PRODUCTS** – pagina ce afiseaza produsele disponibile in cofetarie.

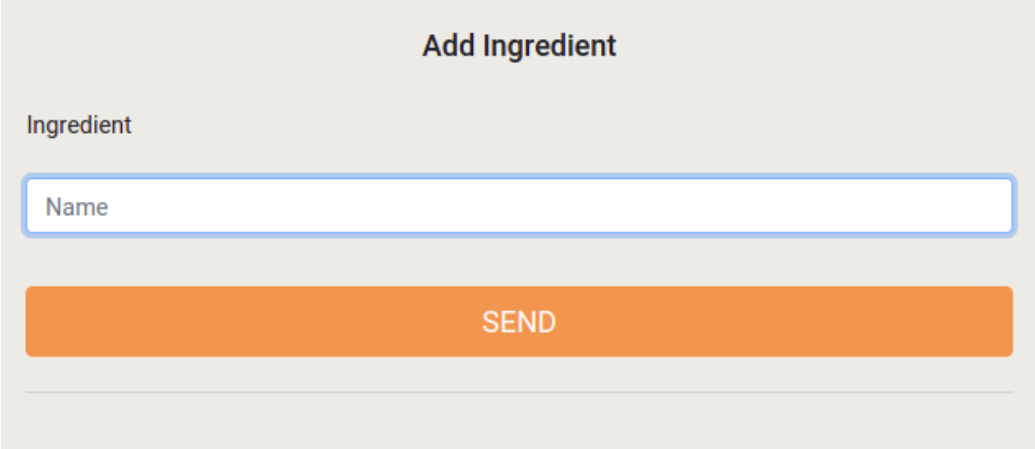



Utilizatorul cu rol de administrator poate sa adauge un nou produs in baza de date prin intermediul formularului de mai jos.

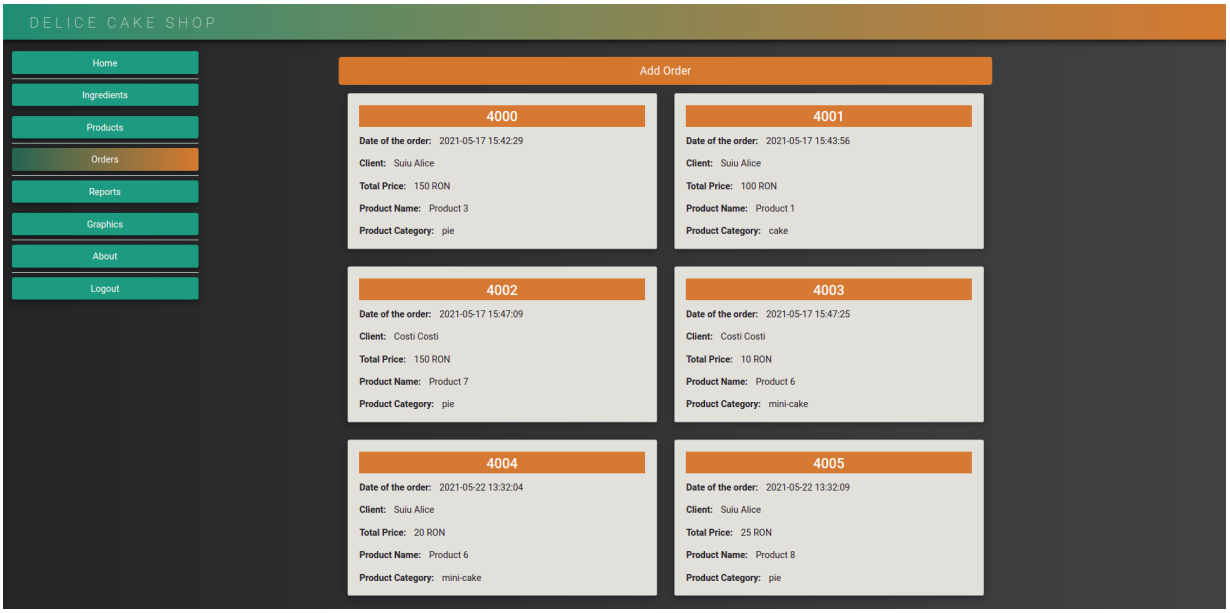



✓ **INGREDIENTS** – pagina ce afiseaza ingredientele principale pe care le pot contine produsele expuse.

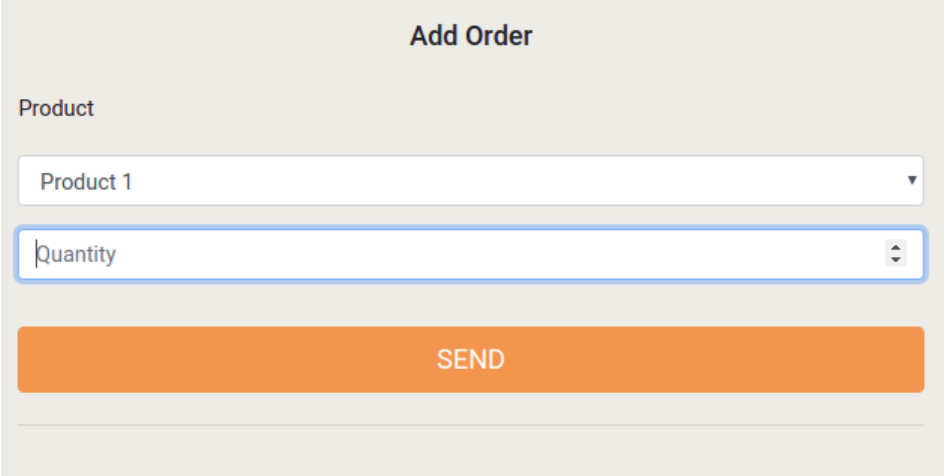

Utilizatorul cu rol de administrator poate sa adauge un nou ingredient in baza de date prin intermediul formularului de mai jos.




✓ **ORDERS** – pagina ce afiseaza comenzile efectuate – ca user admin se pot vedea toate comenzile efectuate de toti userii din sistem.

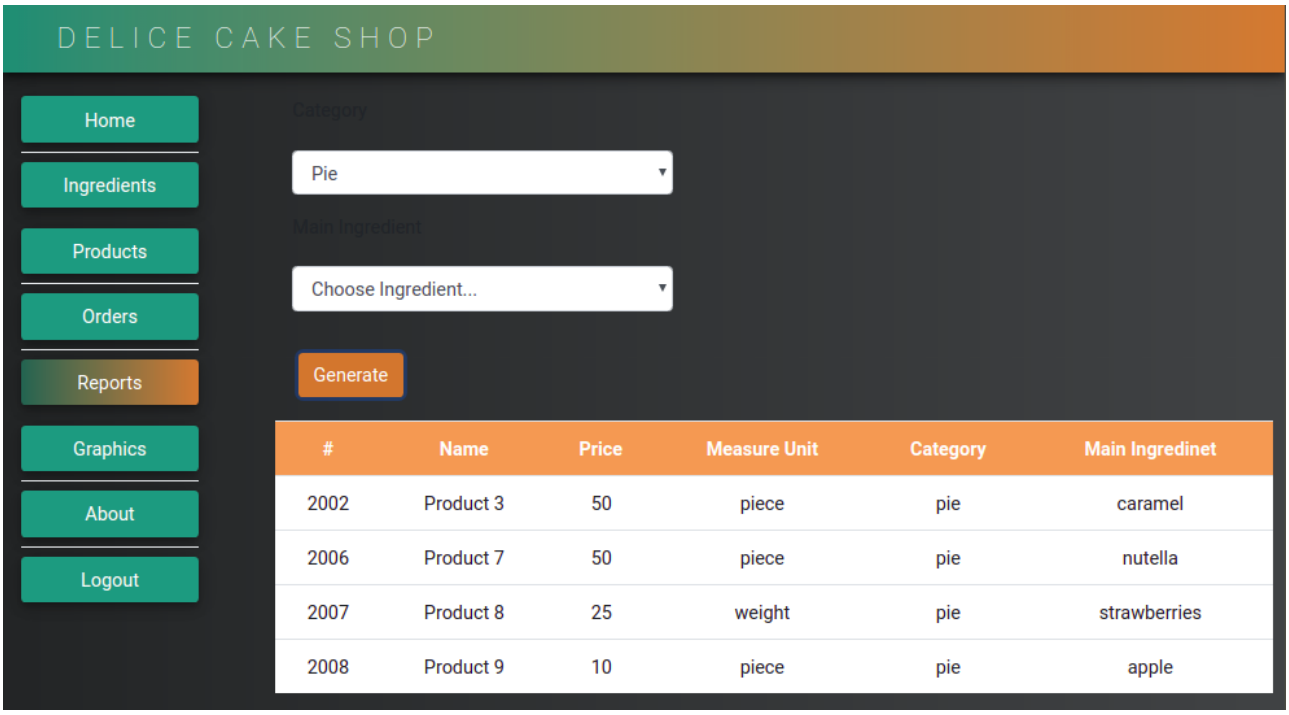



Orice utilizator autentificat in aplicatie poate sa adauge comenzi prin intermediul formularului de mai jos.



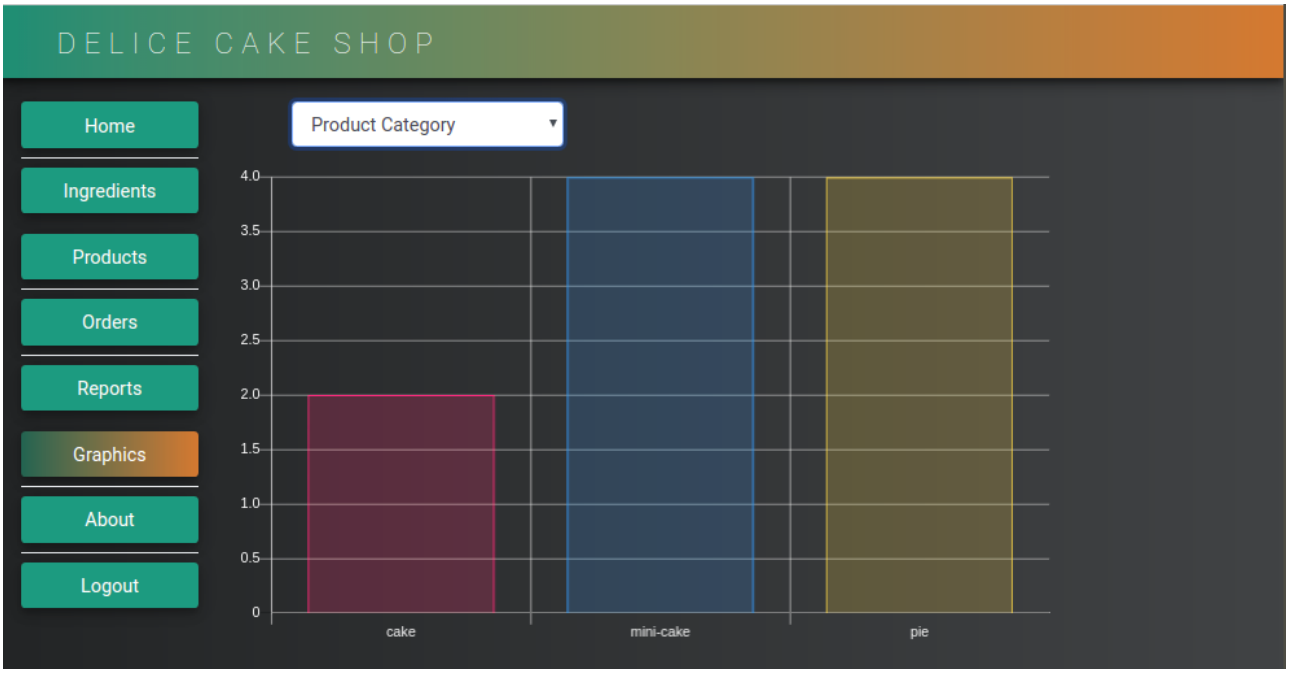

✓ **REPORTS** – pagina ce contine rapoartele aplicatiei. Se pot obtine rapoarte in functie de:

- ingredientul principal
- categoria produselor
- data plasarii comenzilor
- pretul total al comenzilor



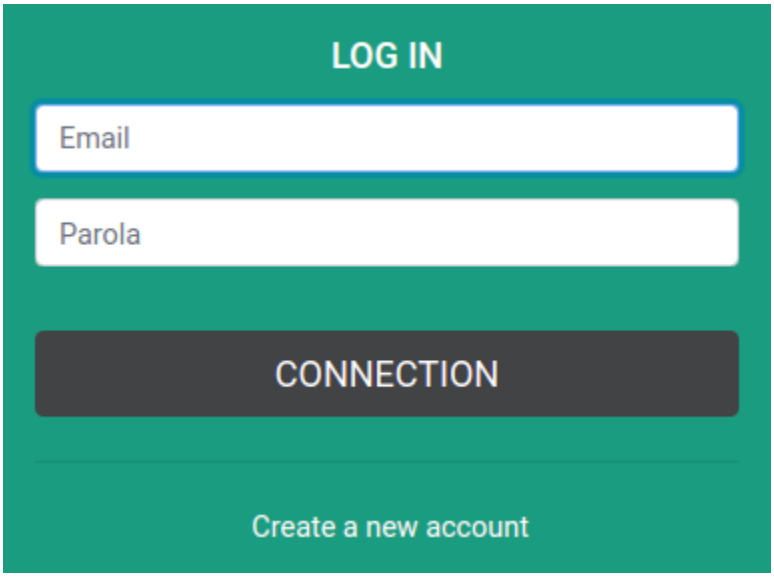
✓ **GRAPHICS** – pagina ce contine graficele aplicatiei. Se pot obtine grafice in functie de:

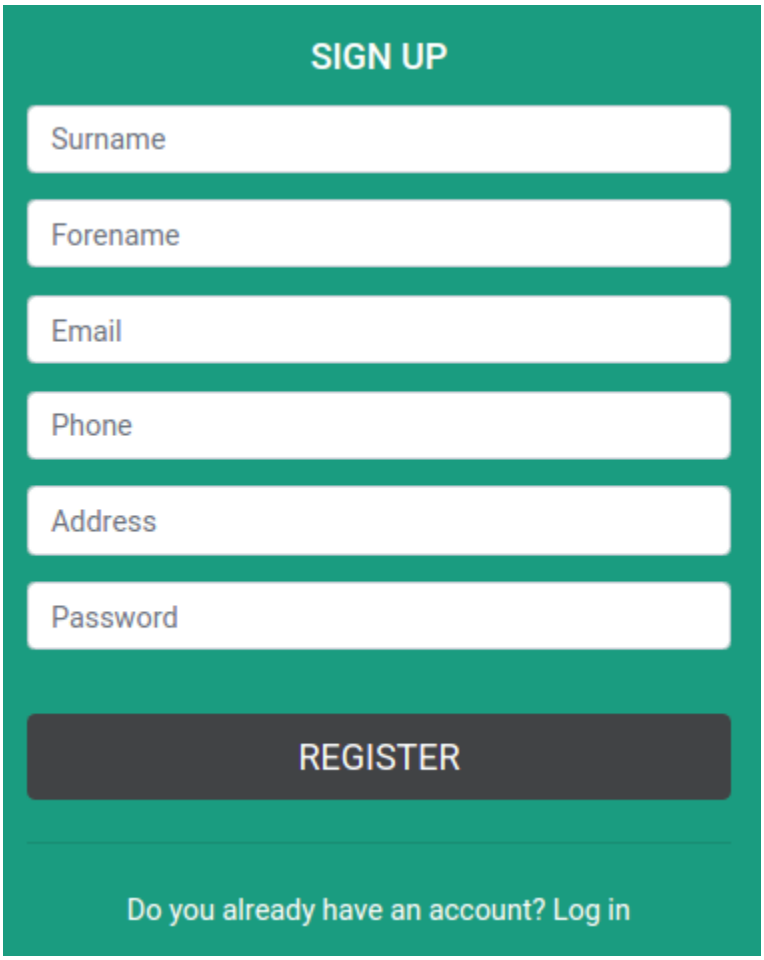
- categoria produselor (tort, mini-prajitura, placinta)
- ingredientul principal




✓ **ABOUT** – pagina ce afiseaza documentatia aplicatiei.

Aplicatia Web permite adaugarea unui nou utilizator in sistem, precum si autentificarea pe baza de e-mail si parola.






Pentru crearea microserviciului ce reprezinta serverul Web, am creat un container cu o imagine de httpd:2.4. Serviciul obtinut serveste ca server Web pentru aplicatia online. Frontend-ul este realizat cu Vanilla JavaScript si cu HTML. Serviciul comunica cu serverul de backend si cu cel pentru autentificare, ambele scrise in Python 3 cu ajutorul bibliotecii Flask.

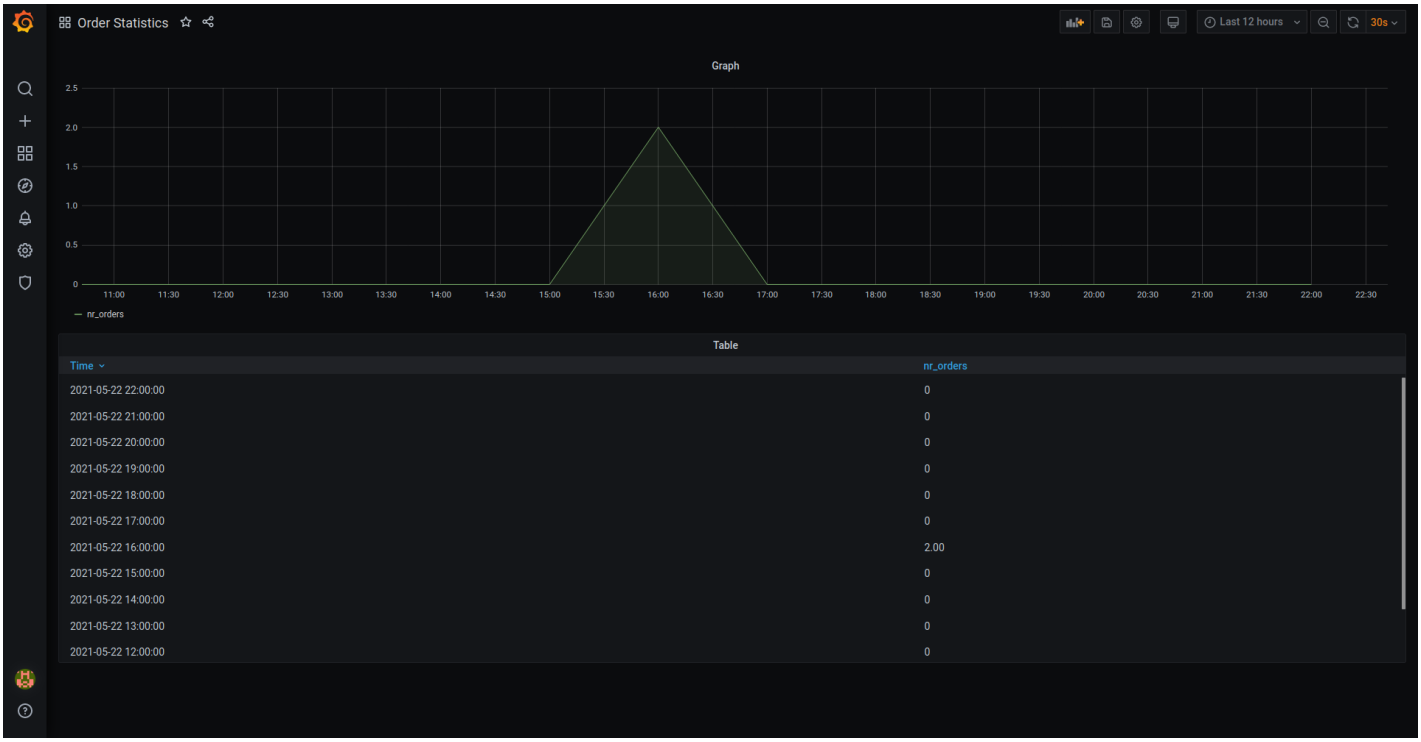
**Baza de date in InfluxDB** este creata cu ajutorul unei imagini de InfluxDB deja existenta pe Docker Hub. Aceasta comunica cu serviciul de monitorizare si cu serviciul reprezentat de adapter. Cheia de memorare va fi un timestamp pentru a putea realiza statistici pe anumite perioade de timp. Un exemplu de inregistrare din baza de date creata este prezentat in imaginea de mai jos:

```
{
   "measurement":"orders",
   "tags":{
      "user":"Suiu_Alice"
   },
   "time":"2021-05-22T13:32:04Z",
   "fields":{
      "quantity":4
   }
}
```


**Adapter** este un program scris in Python 3 cu scopul de a prelua datele din baza de date MySQL, a le prelucra si a le stoca in baza de date InfluxDB, aceste date fiind folosite de serviciul de monitorizare. Aspecte ce vor putea fi monitorizate: numarul de comenzi procesate intr-un anume moment de timp.

Pentru crearea microserviciului ce reprezinta adapterul, am creat un container cu o imagine de Python 3.6. Acest serviciu comunica cu cele doua baze de date MySQL si InfluxDB cu scopul celor mentionate mai sus.

Pentru crearea microserviciului ce reprezinta **componenta de monitorizare**, am creat un container cu o imagine de Grafana de pe Docker Hub. Acest serviciu va comunica cu serviciul reprezentat de baza de date InfluxDB. Serviciul va prelua informațiile din baza de date și va crea statistici (grafice) ce prezinta informații generale cu privire la numarul de comenzi plasate per ora.




## 4. Rularea aplicatiei

Intreaga aplicatie este formata din sapte microservicii asamblate intr-un docker-compose. Pentru a porni aplicatia trebuie rulata comanda **docker-compose up.**

Micorserviciile ce necesita configurarea unui username si a unei parole, au setate secrete asociate campurilor de configurare.

```
secrets/
├── adapter-password.txt
├── adapter-user.txt
├── authserver-password.txt
├── authserver-user.txt
├── grafana-admin-password.txt
├── grafana-admin-user.txt
├── mysql-password.txt
├── pyserver-password.txt
└── pyserver-user.txt
```

## 5. GitHub Link

https://github.com/alicesuiu/cake-shop