بسم الله الرحمن الرحیم

دانشگاه صنعتی شریف

دانشکده‌ی مهندسی کامپیوتر

برنامه سازی وب

استاد: امید جعفری‌نژاد

تمرین سوم - DevOps

نازنین آذریان ۹۸۱۰۵۵۶۸

پویا اسمعیلی ۹۸۱۰۵۵۸۱

بنیامین بیضایی ۹۸۱۰۰۳۵۶

فاطمه عسگری ۹۸۱۰۵۹۲۱

نیم‌سال اول تحصیلی ۰۲-۰۱

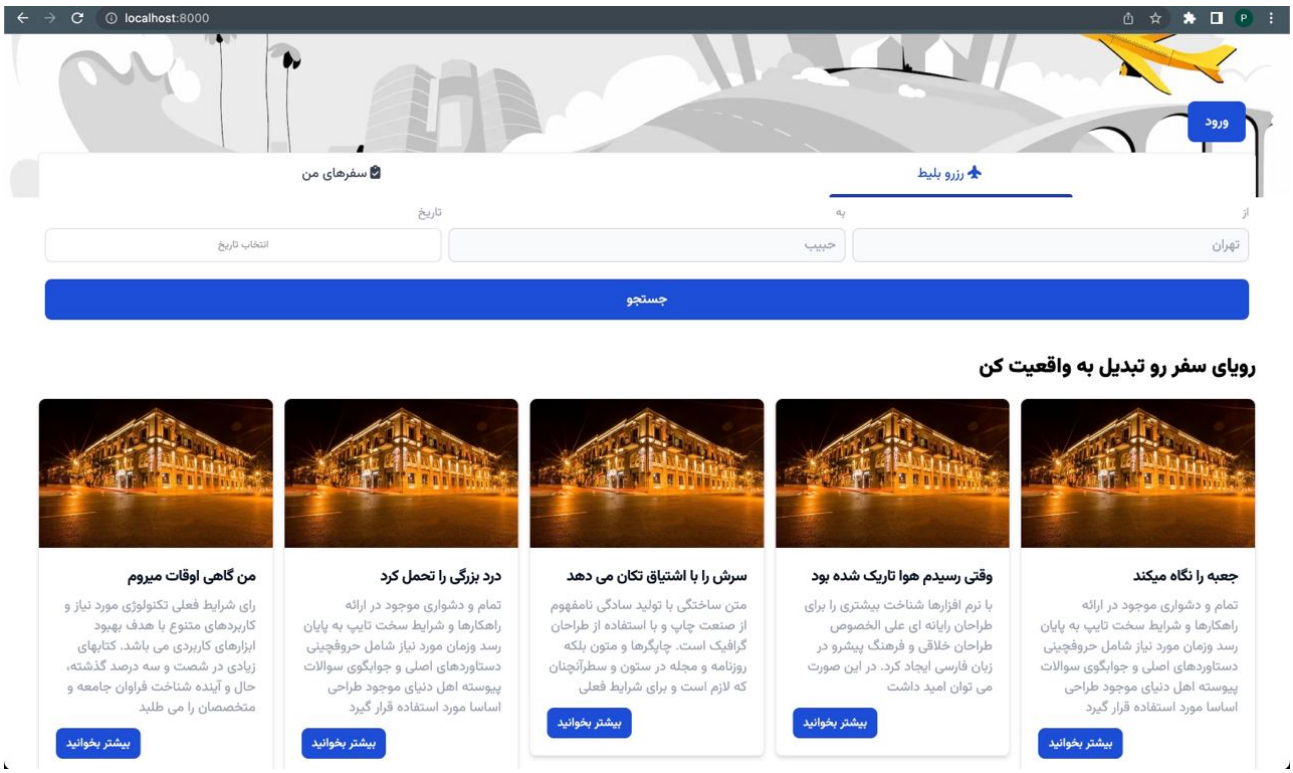
سوال اول)

در این سوال از ما خواسته شده پروژه‌ی فرانت تمرین اول را به صورت یک کانتینر بالا آورده و با استفاده از nginx یک پراکسی معکوس زده و با استفاده از قابلیت port forwarding در داکر این پروژه را روی پورت 8000 دستگاه لوکال خودمان بالا بیاوریم کد مربوط به این قسمت به شرح زیر است:

```
63
64  frontend:
65    platform: linux/amd64
66    image: ghcr.io/web-programming-fall-2022/airline-frontend:v0.0.0
67    depends_on:
68      - auth
69      - ticket
70
71  nginx:
72    image: nginx:latest
73    restart: always
74    ports:
75      - "8000:443"
76    volumes:
77      - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
78      - ./nginx/certs:/etc/nginx/certs:ro
79    depends_on:
80      - auth
81      - ticket
82
```

```
40
41          location ~* ^/ {
42              proxy_pass http://frontend:80;
43              proxy_set_header Host $host;
44          }
```

در نهایت با وارد کردن دستور docker compose up در ترمینال میتوان فرانت را روی پورت ۸۰۰۰ localhost مشاهده کرد:



سوال دوم)

در این سوال در ابتدا کانتینری محتوی postgres بالا آورده‌ایم که در زیر قابل مشاهده است:

```yaml
services:
  postgres:
    image: postgres:14.5
    restart: always
    environment:
      POSTGRES_PASSWORD: DPZ4ZCtygy5rGQRMjkp1DkD6OuAucZtAwzeO6rcWg0
      POSTGRES_USER: postgres
    ports:
      - "5432:5432"
    volumes:
      - ./postgres_init/init.sh:/docker-entrypoint-initdb.d/init.sh
      - ./postgres_init:/app
      - ./postgres:/var/lib/postgresql/data
    healthcheck:
      test: [ "CMD-SHELL", "pg_isready -U postgres" ]
      interval: 10s
      timeout: 5s
      retries: 5
```

سپس یک redis برای کش authentication سرویس بالا آورده‌ایم:

```
redis:
  image: redis:7.0.8
  volumes:
    - ./redis:/data
```

از ما خواسته شده که در صورتی که redis پایین آمد سرور از کار نیفتد و دادها در postgres ذخیره و بازیابی شود:

```
func (m *JWTManager) CheckUnauthorizedToken(ctx context.Context, tokenString string) error {  1 usage  Pouya Esmaili *
    resp := m.RDB.Get(ctx, tokenString)
    if resp.Err() == redis.Nil {
        _, err := m.Storage.GetUnauthorizedToken(tokenString)
        if err != nil {
            m.RDB.SetEx(ctx, tokenString, value: "true", time.Minute*10)
            return nil
        }
        m.RDB.SetEx(ctx, tokenString, value: "false", time.Minute*10)
    } else if resp.Err() != nil {
        _, err := m.Storage.GetUnauthorizedToken(tokenString)
        if err != nil {
            m.RDB.SetEx(ctx, tokenString, value: "true", time.Minute*10)
            return nil
        }
        m.RDB.SetEx(ctx, tokenString, value: "false", time.Minute*10)
    } else if resp.Val() == "true" {
        return nil
    }
    return errors.New( text: "token is unauthorized")
}
```

در ادامه به جای گرفتن shell و ساختن جداول و وارد کردن دادهای csv دیتابیس‌ها، از راهکاری به نام initdb استفاده کردیم که راهکار بسیار تمیز و مناسبی است. Initdb به این صورت عمل می‌کند که در بار اولی که دیتابیس بالا می‌آید تمام کوئری‌های لازم را می‌زند و دیتابیس‌ها را می‌سازد و سپس با زدن کوئری، دیتا را داخل دیتابیس می‌ریزد. کد این بخش و همچنین نتیجه در زیر قابل مشاهده است:

```bash
#!/bin/bash
set -e

export USER=$POSTGRES_USER
export PGPASSWORD=$POSTGRES_PASSWORD

if ! psql -lqt -U $USER | cut -d \| -f 1 | grep -qw auth; then
  psql -U $USER -c "CREATE DATABASE auth;"
fi

if ! psql -lqt -U $USER | cut -d \| -f 1 | grep -qw bank; then
  psql -U $USER -c "CREATE DATABASE bank;"
fi

if ! psql -lqt -U $USER | cut -d \| -f 1 | grep -qw ticket; then
  psql -U $USER -c "CREATE DATABASE ticket;"
  psql -U $USER -d ticket -a -f /app/ticket.sql
fi
```

سوال سوم)

در این قسمت از ما خواسته شده هر سه قسمت authentication و ticket و bank را به صورت چندین کانتینر درآورده و سپس به کانتینرهای دیتابیس قسمت دوم وصل کنیم که کد آن در زیر قابل مشاهده است:

```yaml
  bank:
    platform: linux/amd64
    image: ghcr.io/web-programming-fall-2022/airline-bank:v1.0.0
    environment:
      DB_NAME: bank
      DB_USER: postgres
      DB_PASSWORD: DPZ4ZCtygy5rGQRMjkp1DkD6OuAucZtAwzeO6rcWg0
      DB_HOST: postgres
      DB_PORT: 5432
    depends_on:
      postgres:
        condition: service_healthy

  auth:
    platform: linux/amd64
    image: ghcr.io/web-programming-fall-2022/airline-auth:v1.0.5
    volumes:
      - ./auth/config.yml:/app/config.yml:ro
    command:
      - ./aauth
      - serve
      - -c
      - /app/config.yml
    depends_on:
      postgres:
        condition: service_healthy

  ticket:
    platform: linux/amd64
    image: ghcr.io/web-programming-fall-2022/airline-ticket:v1.0.0
    volumes:
      - ./ticket/.env:/app/.env:ro
    depends_on:
      postgres:
        condition: service_healthy
```

نهایتا هم برای آن‌ها با استفاده از nginx یک پراکسی معکوس زده‌ایم:

```
25
26          location ~* ^/bank {
27              proxy_pass http://bank:8000;
28              proxy_set_header Host $host;
29          }
30
31          location ~* ^/api/v1/auth {
32              proxy_pass http://auth:8080;
33              proxy_set_header Host $host;
34          }
35
36          location ~* ^/api/v1/ticket {
37              proxy_pass http://ticket:3000;
38              proxy_set_header Host $host;
39          }
```

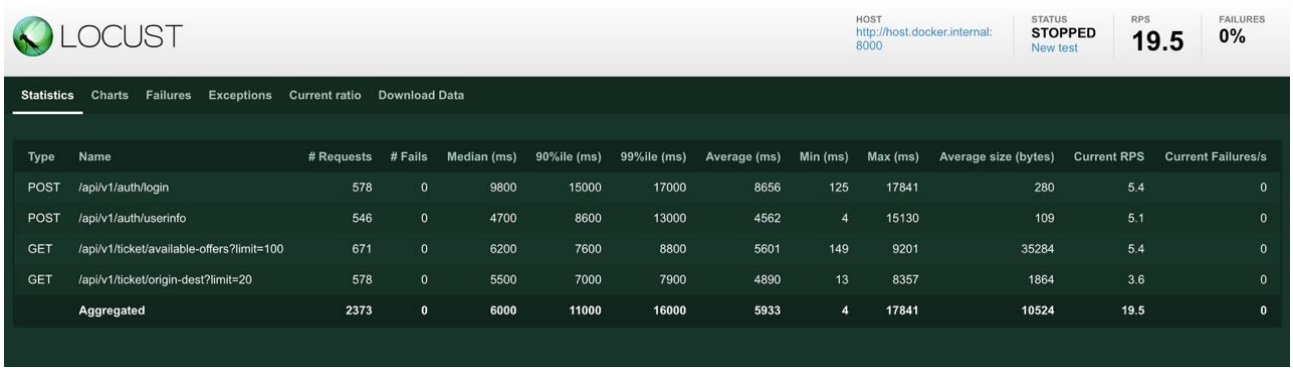
سوال چهارم)

همانطور که در قسمت‌های قبل هم نشان داده شد برای فرانت و بک کانتینرهای متعدد ساخته شده و حالا با داشتن کد زیر، با زدن docker compose up تمام پروژه اعم از بک و فرانت به صورت یکجا بالا می‌آید:

```
    server {
        listen 443;
        server_name airline.pouyaesmaili.ir;

        location ~* ^/bank {
            proxy_pass http://bank:8000;
            proxy_set_header Host $host;
        }

        location ~* ^/api/v1/auth {
            proxy_pass http://auth:8080;
            proxy_set_header Host $host;
        }

        location ~* ^/api/v1/ticket {
            proxy_pass http://ticket:3000;
            proxy_set_header Host $host;
        }

        location ~* ^/ {
            proxy_pass http://frontend:80;
            proxy_set_header Host $host;
        }
    }
```

سوال پنجم)

در نهایت با استفاده از ابزار locust پروژه را مورد تست قرار داده‌ایم:



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	/api/v1/auth/login	578	0	9800	15000	17000	8656	125	17841	280	5.4	0
POST	/api/v1/auth/userinfo	546	0	4700	8600	13000	4562	4	15130	109	5.1	0
GET	/api/v1/ticket/available-offers?limit=100	671	0	6200	7600	8800	5601	149	9201	35284	5.4	0
GET	/api/v1/ticket/origin-dest?limit=20	578	0	5500	7000	7900	4890	13	8357	1864	3.6	0
	Aggregated	**2373**	**0**	**6000**	**11000**	**16000**	**5933**	**4**	**17841**	**10524**	**19.5**	**0**