Съдържание

Въведение

EpayRD е опитен проект за тестване на връзка на React - Django REST приложение с Epay. Той може да бъде полезен като ръководство за програмисти на **React** и **Django**, които се нуждаят от подобна функционалност.

Epay е популярна в България разплащателна система, поради което е важна за местния пазар. Тя предоставя интерфейс към външни програми и техническа документация за ползването му. Включени са примери с **PHP** и **Perl**. Интерфейсът има особености, типични за времето, когато е бил първоначално създаден.

Настоящият проект се фокусира върху връзката с **Epay**. Извън неговия обхват остават въпроси свързани със стандартни мерки за сигурност, с валидация на данните и с интегриране на **React** и **Django.**

Проектът включва пример на създаване на заявка за плащане от регистриран търговец към регистриран потребител.

Кодът е създаден и тестван в **Linux** среда с **Python 3.10.6**.

Коментар на интерфейса

Разработчиците на приложения регистрират примерен търговец и примерен клиент. Примерният търговец получава:

- Идентификационен номер на търговец (MIN) и
- секретната дума (secret)

Заявката за плащане трябва да съдържа няколко реквизита. Те са:

- Идентификационен номер на търговец (MIN)
- Номер на фактура (INVOICE - само цифри)
- Сума (AMOUNT)
- Крайно време за валидност (EXP_TIME)

Също така могат да се попълнят и допълнителни полета като описание, вид валута, обратна връзка при усешно плащане и др. В настоящия проект се ползват:

- описание (DESCR) и
- кодова таблица за описанието (ENCODING)

След като приложението е подготвило необходимата информация, то трябва да произведе многоредов низ (**data**), който изглежда по следния начин:

```
MIN=D460900632
INVOICE=0010000003
AMOUNT=22.80
EXP_TIME=21.09.2022 18:13:30
DESCR=Test
ENCODING=utf-8
```

За да се подаде заявка към **Epay** следва да се извършват следните действия:

1. **data** се кодира с **base64**, съгласно RFC 3548 при EOL=". Резултатът е едноредов низ, който ще наречем **ENCODING**.
2. **ENCODING** се хешира с HMAC с алгоритъм SHA-1 като се ползва секретната дума на търговеца.
   Означаваме резултата с **CHECKSUM**.
3. Съставя се форма, която има следния вид:

```
<form action="https://www.epay.bg/" method=post>
    <input type=hidden name=PAGE value="paylogin">
    <input type=hidden name=ENCODED value="[ENCODED]">
    <input type=hidden name=CHECKSUM value="[CHECKSUM]">
    <input type=submit>
</form>
```

На етап разработка и тестване във формата се подава:

```
action="https://demo.epay.bg/"
```

Организация на проекта

Проектът е съвкупност от два под-проекта, които следва да работят заедно. Единият е React приложение, което е видимо за потребителя, а другият е Django REST програма, която предоставя услуги. Ще ги наречем съответно: **epayr** и **epayd**.

По същество това са две независими програми, които са направени с различни технологии. Те могат както да се интегрират, така и да работят по отделно, включително на различни сървъри.

epayr е клиента, който в съответствие с бизнес логиката си трябва да подготви необходимите данни за създаване на заявка за плащане. **epayd** е бекенда, задачата на когото е да създаде **base64** кодиран низ и да го хешира със секрета на търговеца.

Двете програми комуникират по HTTP. За целта е дефиниран следния интерфейс:

Epayr изпраща **JSON POST** заявка със съдържание, което изглежда по следния начин:

```
{
   "MIN": "D460900632",
   "INVOICE": "0010000003",
   "AMOUNT": 22.80,
   "sec": 3600,
   "DESCR": "Test",
   "ENCODING": "utf-8"
}
```

Тук **sec** е времето на валидност на заявката в секунди. С главни букви са въведени параметрите, които са дефинирани от **Epay**.

Epayd връща отговор със **статус 201** и тяло от вида:

```
{
        "ENCODED":
        "TUlOPUQ0NjA5MDA2MzIKSU5WT0lDRT0wMDEwMDAwMDAzCkFNT1VOVD
        0yMi44MApFWFBfVElNRT0yMy4xMC4yMDIyIDExOjQzOjI2CkRFU0NSPVRlc3
        QKRU5DT0RJTkc9dXRmLTgK",
        "CHECKSUM": "8707eab027ac2eb5ffb2f60493c48700e825b114"
}
```

В случай на грешка отговорът следва да бъде със статус 400 и тяло от вида:

```
{
        “error”: “error note”
}
```

Така дефинфираният интерфейс позволява **epayd** да обслужва всякакъв клиент, който изпълнява протокола, дори и да не е **React** приложение. Под формата на микросървиз той може да предоставя услугата и на клиенти, които ползват друг бекенд за останалата си потребности.

EpayRD. е главна директория на проекта, в която е инициализирано GIT хранилище. Двата под-проекта са съответно в под-директории: **epayr** и **epayd**.

Подпроект epayd

В директория **epayd**, се създава и активира виртуална среда за **Python**:

```
virtualenv venv

source ./venv/bin/activate
```

```
pip install -r requirements.txt
```

След това се създава самия **Django** проект:

```
django-admin startproject _core .

python manage.py startapp epay
```

epayd се състои от две проложения: **_core** и **epay**.

Приложение _core

_core съдържа конфигурацията на системата.

Във файла **_core/settings.py** в **MIDDLEWARE** е добавено: 'corsheaders.middleware.CorsMiddleware':

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'corsheaders.middleware.CorsMiddleware',
]
```

Това е необходимо, за да осигурим CORS функционалност на проекта.

В **INSTALLED_APPS** са включени: '**rest_framework**', '**corsheaders**' и **‘epay’**:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'corsheaders',
    'epay',
]
```

Променено е системното време:

```
TIME_ZONE = 'Europe/Sofia'
```

Във файла **_core/urls.py** е добавен път към **epay**:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('epay/', include('epay.urls')),
]
```

Приложение epay

epay представлява **бекенда**, който работи директно с **React фронтенда** по отношение на разплащанията. Негова задача също така е да съхранява **секрета** на **търговеца**.

Във файла **epay/models.py** е дефиниран

```
class Merchant(models.Model):.
```

Чрез него се съхранява секрета на търговеца в базата данни. Съответстващата таблица се създава, като от командния ред се изпълняват командите:

```
python manage.py makemigrations и

python manage.py migrate
```

В **epay/admin.py** се регистрира класа **Merchant**.

```
admin.site.register(Merchant)
```

Това го прави достъпен през административния интерфейс на **Django**.

Администратор се създава, като се следва диалога, стартиран с командата:

```
python manage.py createsuperuser
```

Администраторът следва да се логне в системата на адрес:

http://127.0.0.1:8000/admin/,

след което да въведе в таблицата **Merchants** данните за регистрирания търговец.
Този подход за съхраняване на секрета на търговеца позволява на приложението да работи и с повече от един търговец.

Във файла **epay/view.py** е дефинирана функция **pay**, която приема обръщенията от **фронтенда**:

```
@api_view(['POST'])
def pay(request):       # /epay/pay/
```

Тя очаква **JSON POST** заявка на адрес: http://localhost:8000/epay/pay/ и трябва да върне кодирания и хеширания низ, съгласно изискванията на **Epay**.

В **epay/utils.py** е дефинирана функция

```
def prepare_payment(dic: dict):,
```

която върши работата по същество. Тя получава като параметър речник с входните данни, извлича секрета на търговеца, съставя дефинирания от **Epay** многоредов низ, кодира го **base_64**, хешира го и връща речник с резултата. За целта ползва услугите на функциите:

```
def b64_encode(message: str, cp: str = 'utf-8'):
```

и

```
def hash_sha1(key: str, message: str, cp: str = 'utf-8'):.
```

Освен параметъра **sec** функцията може да получи алтенативен параметър **untilTime**. В този случай времето на изтичане на заявката, трябва да е определено при клиента и да се педаде във формата, дефинирана от **Epay**.
Това е удобно за дебъгване, но за предпочитане е да се ползва **sec,** тъй като при реална работа клиентът може да е в друга часова зона. Ако са зададени и двата параметъра се изпълнява **untilTime**.

Функцията

```
def b64_decode(b64_message: str, cp: str = 'utf-8'):
```

е добавена за пълнота и може да бъде полезна при тестване.

Подпроект epayr

Приложението се създава като в директоря **EpayRD** се изпълнява командата:

```
npx create-react-app epayr
```

След това всички дейности по този подпроект се извършват в директория **EpayRD/epayr**.

По същество това е опростено **React** приложение, което се състои от две форми: **appForm** и **epForm**.
Първата е дефинирана във файла **App.js** и се показва веднага, след след като приложението стартира. Втората е оформена като самостоятелен компонент, който е създаден във файла **epForm.js** и също се ползва в **App.js**.

Задача на формата в **appForm** е да направи **POST** заявка към **epayd**. Тогава в случай на успешна заявка **epForm** следва да направи **POST** заявка с включени **ENCODING** и **CHECKSUM** към **Epay**.

Особеното е, че двете заявки трябва да се изпълнят след еднократно натисване на бутона „**OK**“.

Формата **epForm** не съдържа видими за потребителя компоненти. Тя получава като входни параметри **PAGE**, **ENCODING** и **CHECKSUM**. При първоначално пускане на приложението **ENCODING** е празен низ. След като **appForm** извърши усешна заявка, той получава съответна стойност. Кодът

```
useEffect(() => {
        if (ENCODED && !isSubmitted) {
                formRef.current.submit();
                setIsSubmitted(true)
        }
}, [ENCODED]);
```

следи за тази промяна и когато тя се случи отправя заявката към **Epay**.

Заключение

Разгледания пример включва два подпроекта, които са направени с различни технологии. **React** е едно от водещите понасоящем средства за създаване на **фронтенд**. **Django** е високо ефективна платформа за сървърно програмиране. При внедряване те могат да работят както по отделно, така и да се интегрират. До колкото връзката между тях е по **HTTP**, **Django** приложението може да обслужва всякакъв клиент, който се придържа към договорения интерфейс. Тъй като **epayd** пази секрета на търговеца в базата данни, той може да обслужва повече от един търговец.