

## Documentation Technique :

**Objectifs :** Ce site internet est conçu pour gérer des utilisateurs, des offres et des achats.  
Les utilisateurs peuvent s'inscrire et se connecter sur leur compte.  
Les offres peuvent être créées et supprimées par l'administrateur.  
Les utilisateurs peuvent effectuer des achats et consulter l'historique de leurs transactions.

### Technologies utilisées :

- Django pour le backend.
- Django REST Framework pour créer les APIs.
- JWT (JSON Web Token) pour l'authentification.
- React pour le frontend.
- Postgresql pour la base de données.
- Heroku pour développement

## Sommaire

<b>1.</b>	<b>CONFIGURER L'ENVIRONNEMENT DE DEVELOPPEMENT</b>	<b>2</b>
<b>2.</b>	<b>ORGANISATION DU PROJET</b>	<b>2</b>
<b>A.</b>	<b>CLE SECRETE</b>	<b>3</b>
<b>B.</b>	<b>APPLICATIONS INSTALLEES</b>	<b>3</b>
<b>C.</b>	<b>MIDDLEWARE</b>	<b>4</b>
<b>D.</b>	<b>AUTHENTIFICATION</b>	<b>4</b>
<b>E.</b>	<b>SÉCURITÉ</b>	<b>5</b>
<b>F.</b>	<b>URLS AU NIVEAU PROJET</b>	<b>5</b>
<b>G.</b>	<b>APPLICATION USER</b>	<b>6</b>
<b>H.</b>	<b>APPLICATION OFFRE</b>	<b>8</b>
<b>I.</b>	<b>APPLICATION PURCHASE</b>	<b>9</b>
<b>J.</b>	<b>DATABASE</b>	<b>10</b>
<b>K.</b>	<b>REACT COMPONENTS</b>	<b>12</b>
<b>3.</b>	<b>MANUEL UTILISATION POUR L'ADMINISTRATEUR</b>	<b>14</b>
<b>4.</b>	<b>TESTS</b>	<b>17</b>

## 1. Configurer l'environnement de développement:

```
git clone https://github.com/Theo2402/bloc3studi52
```

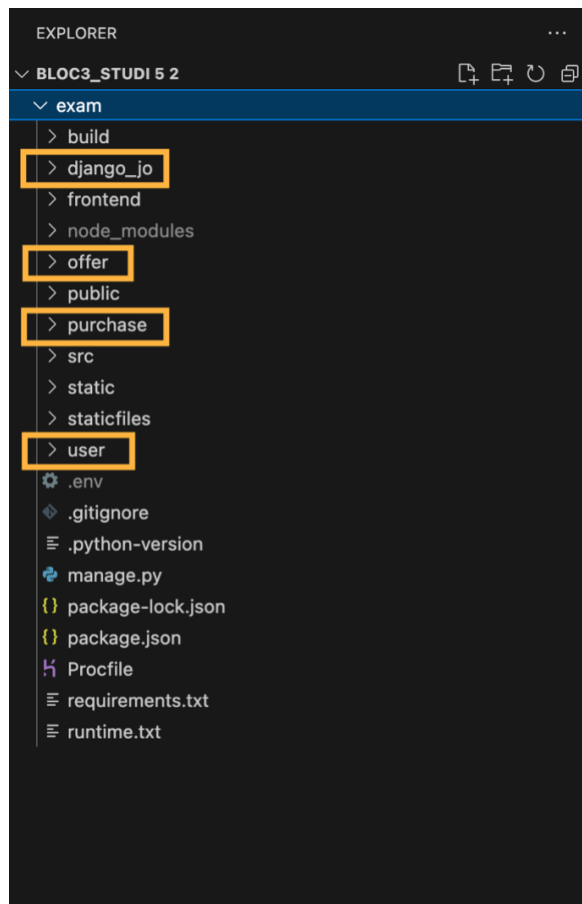
```
cd exam
```

```
python -m venv env  
source env/bin/activate
```

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

## 2. Organisation du Projet :

Backend avec Django:



django\_jo contient settings.py et urls.py au niveau projet.

Le fichier settings.py contient les configurations globales pour le projet Django. Voici les éléments clés et leur description :

### a. Clé secrète

SECRET\_KEY : La clé secrète est utilisée par Django pour sécuriser les cookies et les autres données sensibles. Elle doit être gardée secrète.

DEBUG : Le mode debug doit être désactivé en production pour des raisons de sécurité.

```
SECRET_KEY = config('SECRET_KEY')

DEBUG = config('DEBUG', default=False, cast=bool)
```

### b. Applications Installées

Liste des applications Django et des applications utilisées dans le projet. Cela inclut les applications user, offer, et purchase.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'user',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'corsheaders',
    'rest_framework',
    'rest_framework_simplejwt',
    'django_extensions',
    'offer',
    'purchase',
    'whitenoise.runserver_nostatic',
    'csp',
]
```

### c. Middleware

Le middleware est une série de hooks utilisés pour modifier les requêtes et les réponses de Django. On a notamment WhiteNoiseMiddleware pour servir les fichiers statiques et CSPMiddleware pour renforcer la sécurité via Content Security Policy.

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

### d. Authentication

Configuration avec JWT via Django REST Framework.

```
SIMPLE_JWT = {  
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=40),  
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),  
    'ROTATE_REFRESH_TOKENS': True,  
    'BLACKLIST_AFTER_ROTATION': True,  
    'ALGORITHM': 'HS256',  
    'SIGNING_KEY': SECRET_KEY,  
    'VERIFYING_KEY': None,  
    'AUTH_HEADER_TYPES': ('Bearer',),  
    'USER_ID_FIELD': 'id',  
    'USER_ID_CLAIM': 'user_id',  
    'AUTH_TOKEN_CLASSES': ('rest_framework_simplejwt.tokens.AccessToken',),  
    'TOKEN_TYPE_CLAIM': 'token_type',  
    'SLIDING_TOKEN_REFRESH_EXP_CLAIM': 'refresh_exp',  
    'SLIDING_TOKEN_LIFETIME': timedelta(minutes=5),  
    'SLIDING_TOKEN_REFRESH_LIFETIME': timedelta(days=1),  
}
```

## e. Sécurité

Paramètres de sécurité pour HTTPS, Content Security Policy (CSP), et configurations spécifiques pour le déploiement sur Heroku.

```
SECURE_SSL_REDIRECT = True
SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')
CSRF_COOKIE_SECURE = True
SESSION_COOKIE_SECURE = True
SECURE_HSTS_SECONDS = 3600
SECURE_HSTS_INCLUDE_SUBDOMAINS = True
SECURE_HSTS_PRELOAD = True
```

```
CSP_DEFAULT_SRC = ('self',)
CSP_STYLE_SRC = ('self', 'https://bloc3exam-a2922cc2f685.herokuapp.com')
CSP_SCRIPT_SRC = ('self', 'https://bloc3exam-a2922cc2f685.herokuapp.com')
CSP_CONNECT_SRC = ('self', 'https://bloc3exam-a2922cc2f685.herokuapp.com')
CSP_IMG_SRC = ('self', 'data:') #, 'https://bloc3exam-a2922cc2f685.herokuapp.com'
```

## f. URLs au niveau projet :

Le fichier urls.py au niveau du projet définit les URL patterns pour l'ensemble du projet Django. Il inclut les routes pour l'administration, l'authentification via JWT, et les routes pour les applications internes telles que : user, offer, et purchase.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/token/', MyTokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
    path('api/user/', include('user.urls')),
    path('api/offer/', include('offer.urls')),
    path('api/purchase/', include('purchase.urls')),
    path('api/register/', RegisterUserAPIView.as_view(), name='register'),
    path('', TemplateView.as_view(template_name='index.html')),
    path('favicon.ico', RedirectView.as_view(url='/static/favicon.ico', permanent=True)),
    re_path(r'^.*$', TemplateView.as_view(template_name='index.html')),
]
```

**Administration Django :** path('admin/', admin.site.urls)

Fournit l'accès à l'interface d'administration de Django où les administrateurs peuvent gérer les utilisateurs, les offres, les achats, et autres modèles.

**Authentification JWT :**

api/token/ : Endpoint pour obtenir un token JWT en échange des informations d'identification de l'utilisateur (nom d'utilisateur et mot de passe).

api/token/refresh/ : Endpoint pour rafraîchir un token JWT expiré en utilisant un token de rafraîchissement valide.

**Routes des applications :**

api/user/ : Inclus les routes définies dans user/urls.py, qui gèrent les opérations liées aux utilisateurs.

api/offer/ : Inclus les routes définies dans offer/urls.py, qui gèrent les opérations liées aux offres.

api/purchase/ : Inclus les routes définies dans purchase/urls.py, qui gèrent les opérations liées aux achats.

**Enregistrement des utilisateurs : api/register/**

Endpoint pour permettre à de nouveaux utilisateurs de s'enregistrer en fournissant les informations requises (nom d'utilisateur, mot de passe, email, etc.).

**TemplateView:**

Pour servir le template index.html en tant que page d'accueil du site web.

**Favicon :**

Redirige les requêtes pour favicon.ico vers le fichier favicon situé dans le répertoire des fichiers statiques.

**Fallback :**

Attrape toutes les autres URL non définies et sert le template index.html.

## g. Application user

**Objectifs de l'Application user :** L'application user gère l'authentification, l'autorisation et les informations des utilisateurs. Elle permet la création, la suppression et l'authentification des utilisateurs.

Le modèle est créé pour que chaque utilisateur ait un nom, email, rôle (admin ou pas) et un uuid (clé unique créée lors de la création d'un compte qui sera ajoutée à une deuxième clé pour former le QR code).

```

from django.db import models

# Create your models here.
from django.contrib.auth.models import AbstractUser
import uuid

class UserProfile(AbstractUser):
    uuid = models.UUIDField(default=uuid.uuid4, editable=True, unique=True)
    name = models.CharField(max_length=100)
    email = models.EmailField(unique=True)
    role = models.CharField(max_length=50)

```

#### User API:

- GET /api/user/ - Lister tous les utilisateurs.
- POST /api/user/ - Créer un nouvel utilisateur.
- GET /api/user/{id}/ - Récupérer les détails d'un utilisateur spécifique.
- DELETE /api/user/{id}/ - Supprimer un utilisateur.
- POST /api/register/ - Enregistrer un nouvel utilisateur.
- POST /api/token/ - Obtenir un token JWT.
- POST /api/token/refresh/ - Rafraîchir un token JWT.

Gérer l'authentification des utilisateurs et l'obtention des tokens JWT (MyTokenObtainPairView)

```

21
22 from rest_framework_simplejwt.serializers import TokenObtainPairSerializer
23 from rest_framework_simplejwt.views import TokenObtainPairView
24 #from .custom_token import CustomRefreshToken
25
26 class MyTokenObtainPairSerializer(TokenObtainPairSerializer):
27     @classmethod
28     def get_token(cls, user):
29         token = super().get_token(user)
30         #token = CustomRefreshToken.for_user(user)
31         print(token)
32         # Custom claims
33         token['is_admin'] = user.is_staff
34         token['username'] = user.username
35         return token
36
37 class MyTokenObtainPairView(TokenObtainPairView):
38     serializer_class = MyTokenObtainPairSerializer
39

```

Enregistrer un nouvel utilisateur (RegisterUserAPIView)

```
60 from rest_framework import status
61 from rest_framework.response import Response
62 from rest_framework.views import APIView
63 from .serializers import UserSerializer
64
65 class RegisterUserAPIView(APIView):
66     def post(self, request, *args, **kwargs):
67         #print("Received data:", request.data) # Incoming data
68         serializer = UserSerializer(data=request.data)
69         if serializer.is_valid():
70             serializer.save()
71             return Response(serializer.data, status=status.HTTP_201_CREATED)
72         else:
73             print("Errors:", serializer.errors)
74             return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
75
```

Supprimer un utilisateur (UserViewSet):

```
10 User = get_user_model()
11
12 class UserViewSet(viewsets.ModelViewSet):
13     queryset = User.objects.all()
14     serializer_class = UserSerializer
15
16     def destroy(self, request, *args, **kwargs):
17         user = get_object_or_404(UserProfile, uuid=kwargs['pk'])
18         user.delete()
19         return Response(status=status.HTTP_204_NO_CONTENT)
20
```

## h. Application offre:

**Objectifs de l'application offer :** L'application offer gère les offres disponibles dans le système. Elle permet la création, la suppression et la récupération des offres.

Une offre aura les champs : titre, description et prix.

```
exam > offer > models.py > ...
1 from django.db import models
2
3 # Create your models here.
4 from django.db import models
5
6 class Offer(models.Model):
7     title = models.CharField(max_length=200)
8     description = models.TextField()
9     price = models.DecimalField(max_digits=6, decimal_places=2)
10
```



#### Offer API:

- GET /api/offer/ - Lister toutes les offres.
- POST /api/offer/ - Créer une nouvelle offre.
- GET /api/offer/{id}/ - Récupérer les détails d'une offre spécifique.
- DELETE /api/offer/{id}/ - Supprimer une offre.

### i. Application purchase:

**Objectifs de l'Application purchase :** L'application purchase gère les achats effectués par les utilisateurs. Elle permet la création, la suppression et la récupération des achats.

Le modèle contient les champs suivants : user, offer, purchase\_date, et safe\_key. Pour chaque achat on aura ainsi un utilisateur, une offre, une date d'achat et une clé de sécurité associée.

```
from django.db import models

# Create your models here.
from django.utils.crypto import get_random_string
from user.models import UserProfile
from offer.models import Offer

def generate_safe_key():
    return get_random_string(length=32)

class Purchase(models.Model):
    user = models.ForeignKey(UserProfile, on_delete=models.CASCADE)
    offer = models.ForeignKey(Offer, on_delete=models.CASCADE)
    purchase_date = models.DateTimeField(auto_now_add=True)
    safe_key = models.CharField(max_length=200, default=generate_safe_key)

    def save(self, *args, **kwargs):
        if not self.safe_key:
            self.safe_key = get_random_string(length=32)
        super().save(*args, **kwargs)
```

#### Purchase API:

- GET /api/purchase/purchase/ - Lister tous les achats.
- POST /api/purchase/purchase/ - Créer un nouvel achat.
- GET /api/purchase/purchase/{id}/ - Récupérer les détails d'un achat spécifique.
- DELETE /api/purchase/purchase/{id}/ - Supprimer un achat.

## j. Database:

La configuration de la base de données se trouve dans le fichier settings.py. Le projet utilise PostgreSQL comme système de gestion de base de données (SGBD) principal.

Django utilise le système de migrations pour appliquer des changements aux modèles de manière sûre.

Les commandes suivantes permettent de créer et appliquer les migrations :

*cd exam*

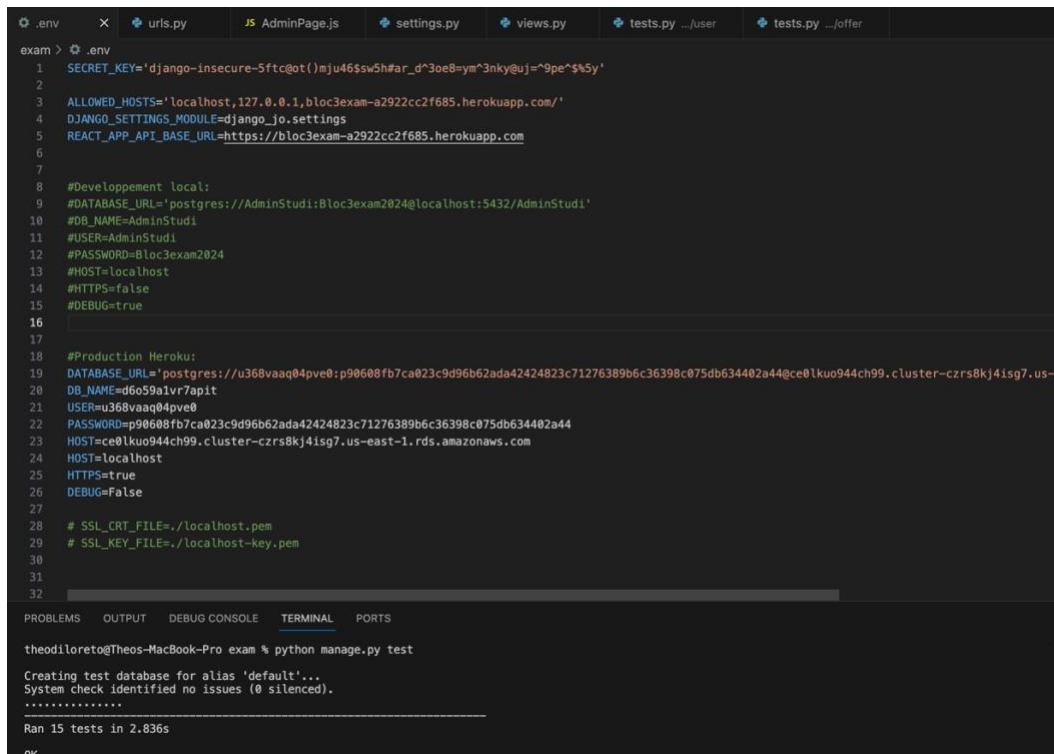
*python manage.py makemigrations* (Créer des fichiers de migration pour les modifications apportées aux modèles.)

*python manage.py migrate* (Appliquer les migrations à la base de données)

Sécurité de la base de données :

Pour les environnements de production, la connexion à la base de données est sécurisée avec SSL.

Les informations sensibles telles que les identifiants de la base de données sont stockées dans des variables d'environnement (fichier .env) et gérées via decouple.config.



```
.env
1 SECRET_KEY='django-insecure-5ftc@ot()mju46$sw5h#ar_d^3oe8=ym^3mky@uj=~9pe*$s$y'
2
3 ALLOWED_HOSTS='localhost,127.0.0.1,bloc3exam-a2922cc2f685.herokuapp.com/'
4 DJANGO_SETTINGS_MODULE=django_jo.settings
5 REACT_APP_API_BASE_URL=https://bloc3exam-a2922cc2f685.herokuapp.com
6
7
8 #Development local:
9 #DATABASE_URL='postgres://AdminStudi:Bloc3exam2024@localhost:5432/AdminStudi'
10 #DB_NAME=AdminStudi
11 #USER=AdminStudi
12 #PASSWORD=Bloc3exam2024
13 #HOST=localhost
14 #HTTPS=false
15 #DEBUG=true
16
17
18 #Production Heroku:
19 DATABASE_URL='postgres://u368vaaq04pve0:p90608fb7ca023c9d96b62ada42424823c71276389b6c36398c075db634402a44@ce01kuo944ch99.cluster-czrs8kj4isg7.us-east-1.rds.amazonaws.com'
20 DB_NAME=d6o59a1vr7apit
21 USER=u368vaaq04pve0
22 PASSWORD=p90608fb7ca023c9d96b62ada42424823c71276389b6c36398c075db634402a44
23 HOST=ce01kuo944ch99.cluster-czrs8kj4isg7.us-east-1.rds.amazonaws.com
24 HOST=localhost
25 HTTPS=true
26 DEBUG=False
27
28 # SSL_CERT_FILE=./localhost.pem
29 # SSL_KEY_FILE=./localhost-key.pem
30
31
32
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
theodiloreto@Theos-MacBook-Pro exam % python manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
Ran 15 tests in 2.036s
OK
```

Pour Heroku, la base de données utilise le service heroku-postgresql avec le plan essentiel-0.

Pour appliquer les migrations de la base de données :

```
heroku run python manage.py makemigrations
```

```
heroku run python manage.py migrate
```

The screenshot shows the Salesforce Platform interface for a Datastore. The breadcrumb trail is "Datastores > postgresql-graceful-92822". Below this, there are tabs for "SERVICE heroku-postgresql", "PLAN essential-0", and "BILLING APP bloc3exam". A navigation bar includes "Overview" (selected), "Durability", "Settings", and "Dataclips". The "HEALTH" section shows a green checkmark and the word "Available". Below this, a row of status indicators shows "PRIMARY Yes", "VERSION 16.2", "CREATED 2 months ago", "MAINTENANCE Unsupported", and "ROLLBACK Unsupported". The "UTILIZATION" section displays three metrics: "0 of 20" for connections, "8.5 MB of 1 GB" for data size, and "12 of 4,000" for tables. Each metric has a green dot and the text "IN COMPLIANCE".

### Sécurité:

Heroku prend en charge plusieurs mesures de sécurité pour protéger la base de données :

- **SSL** : Les connexions à la base de données sont sécurisées par SSL.
- **Variables d'Environnement** : Pour stocker les informations sensibles.

Development avec Heroku:

The screenshot shows the "App Information" page in the Heroku dashboard. It displays the following details for the app "bloc3exam":

- App Name**: bloc3exam
- Region**: Europe
- Stack**: heroku-24
- Framework**: Python
- GitHub repo**: Theo2402/bloc3studi52
- Heroku git URL**: https://git.heroku.com/bloc3exam.git


## Configuration sur Heroku :

### Config Vars

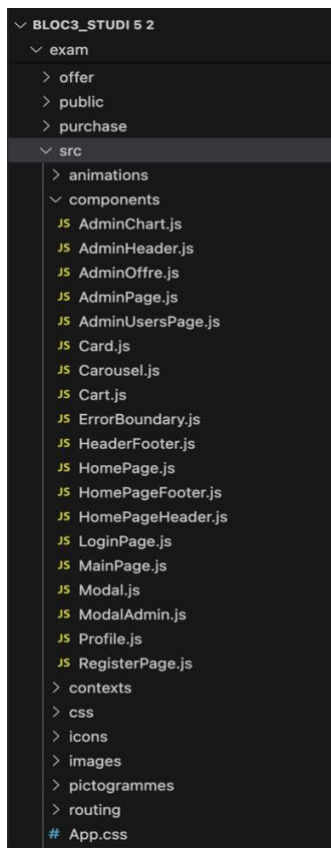
Config vars change the way your app behaves. In addition to creating your own, some addons come with their own.

### Config Vars

Hide Config Vars

ALLOWED_HOSTS	bloc3exam-a2922cc2f685.herokuapp.com	 
DATABASE_URL	postgres://ubs9h052rggt2m:p459fc3862d	 
DEBUG	False	 
DISABLE_COLLECTSTATIC	1	 
EMAIL_HOST_PASSWORD	Bloc3exam2024	 
EMAIL_HOST_USER	AdminStudi	 
REACT_APP_API_BASE_URL	https://bloc3exam-a2922cc2f685.herokuapp.com	 
SECRET_KEY	django-insecure-5ftc@ot()mju46\$sw5h#a	 
KEY	VALUE	 
		

## k. React components:



Les différents composants React ainsi que les fichiers CSS et icônes/images/animations/pictogrammes se situent dans le dossier source (src).

**AdminChart.js** permet de visualiser le nombre d'achat de tickets par jour.

**AdminHeader.js** correspond au Header (en-tête) de la page administrateur.

**AdminOffre.js** permet à l'administrateur d'ajouter ou supprimer des offres.

**AdminPage.js** permet la gestion des utilisateurs, des achats et des offres dans l'interface administrateur.

**AdminUserPage.js** permet de gérer et afficher les informations des utilisateurs (nom, l'email et l'UUID ) ainsi que leurs achats associés (offre, la date de l'achat, la clé de sécurité (safe key), la clé combinée et le QR code). Il permet aussi de supprimer des utilisateurs.

**Card.js** permet de voir sur la page d'accueil quelques sports olympiques sous forme de cartes.

**Carousel.js** permet de faire défiler les cartes présentant les sports.

**Cart.js** permet à l'utilisateur d'ajouter une offre à son panier

**ErrorBoundary.js** peut être ignorer. Il a été utiliser pour afficher les erreurs JS.

**HeaderFooter.js** permet de créer un Header et Footer sur toutes les pages du sites internet sauf la page d'accueil.

**HomePage.js** correspond à la page d'accueil du site. Il utilise les composants pour ajouter un header (HomePageHeader.js), footer (HomePageFooter.js) et le Carousel.

**HomePageFooter.js** correspond au footer de la page d'accueil du site.

**HomePageHeader.js** correspond au header de la page d'accueil du site.

**LoginPage.js** permet aux utilisateurs de s'identifier sur le site.

**MainPage.js** correspond à la page ou les offres sont visibles et ou l'utilisateur peut ajouter à son panier les offres qui l'intéressent.

**Modal.js** peut être ignorer. Il a été créer mais n'a pas été utilisé.

**ModalAdmin.js** est utilisé pour afficher une boîte de dialogue modale c'est à dire une boîte de dialogue superposée à la l'interface utilisateur. L'utilisateur (ici l'administrateur) est obligé d'interagir avec elle avant de revenir à l'interface principale. Dans ce cas précis, ce modal est utilisé pour la suppression des offres dans l'espace administrateur.

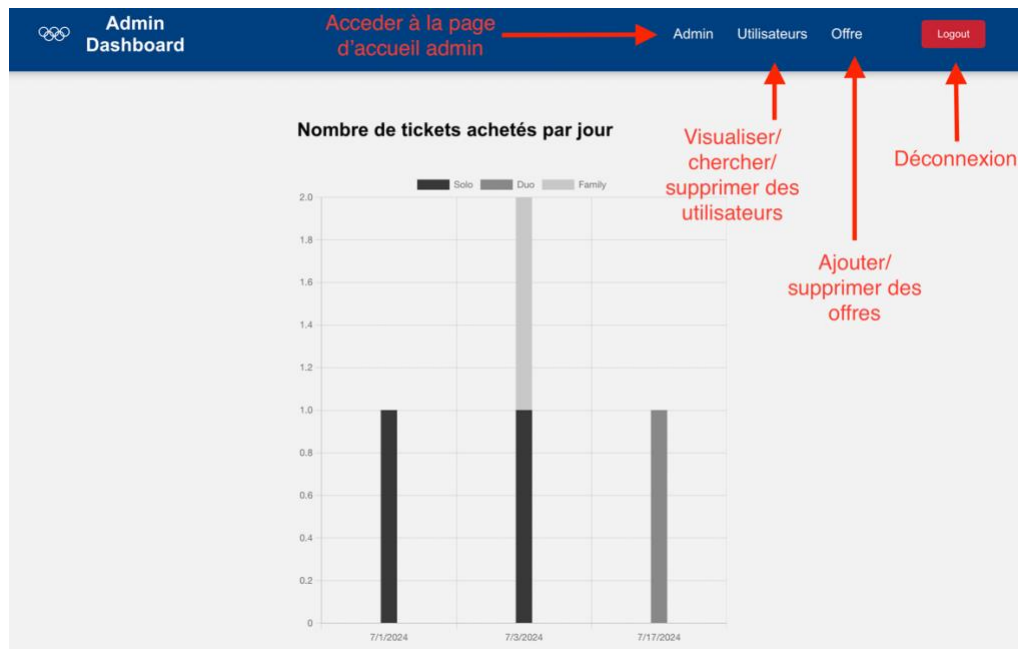
**Profile.js** permet aux utilisateurs d'accéder à leur profile et de voir leur billet acheté et leur QR code.

**RegisterPage.js** permet aux utilisateurs non-inscrits de créer un compte à partir du prénom, nom, email et mot de passe sécurisé. A la fin de la création de son compte l'utilisateur sera redirigé vers la page de Login.

Les fichiers CSS correspondants aux composants React ont le même nom. Ainsi RegisterPage.js aura comme fichier CSS : RegisterPage.css

### 3. Manuel Utilisation pour l'administrateur

- Onglet page d'accueil Admin :



- Onglet Utilisateurs :

The screenshot shows the 'Admin Dashboard' header with navigation links: Admin, Utilisateurs, Offre, and a red Logout button. Below the header is a section titled 'Liste des utilisateurs et des achats'. A red arrow points to a search bar with the text 'Faire une recherche par clé (uuid) ou par nom d'utilisateur'. The search bar contains the text 'Search users or uuid...'. Below the search bar are three user cards for 'Tim', 'Bob', and 'Robin'. Each card displays the user's email and uuid, a 'Delete' button, and a dropdown menu labeled 'Achats'. Red arrows point to the 'Delete' buttons with the text 'Supprimer un utilisateur' and to the 'Achats' dropdown menu with the text 'Cliquer ici pour voir le détails de chaque utilisateur'.

Utilisateur	Email	UUID	Actions
Tim	tim@gmail.com	b483797a-3698-4526-bd95-e20a52119d29	Delete
Bob	tim@yahoo.com	ad2bcd5-09cf-46af-ae35-3738206d15e5	Delete
Robin	robin@gmail.com	fee4f8ce-d840-46e5-9ef8-96827a95bcd	Delete

**Admin Dashboard** Admin Utilisateurs Offre Logout

### Liste des utilisateurs et des achats

Search users or uuid...

**Tim**  
email: tim@gmail.com  
uuid: b483797a-3698-4526-bd95-e20a52119d29

Offer: Solo  
Purchase Date: 7/1/2024, 2:34:06 PM  
Safe Key: e5Male3CwdZkw8WxEE2K7QwhutMWsA2Z  
Combined Key: b483797a-3698-4526-bd95-e20a52119d29\_e5Male3CwdZkw8WxEE2K7QwhutMWsA2Z

Achats ▲

Delete

QR code

Clé créée lors de la création du compte

Type d'offre

Date de l'achat

Clé créée lors de l'achat

Clé combinant les deux clés précédentes

Supprimer un utilisateur

Pour supprimer un utilisateur, un message s'affichera pour confirmer sa suppression.

**Admin Dashboard** Admin Utilisateurs Offre Logout

### Liste des utilisateurs et des achats

Search users or uuid...

**Tim**  
email: tim@gmail.com  
uuid: b483797a-3698-4526-bd95-e20a52119d29

Achats ▼

Delete

**Bob**  
email: tim@yahoo.com  
uuid: ad2bcd5-09cf-46af-ae35-3738206d15e5

Achats ▼

Delete

**Robin**  
email: robin@gmail.com  
uuid: fee4f9ce-d840-46e5-9ef8-9682f7a95bcd

Achats ▼


Delete

Etes-vous sûr de vouloir supprimer cet utilisateur ?

Bob

Yes Non

- Onglet Offre pour l'administrateur:



Admin Dashboard

Admin

Utilisateurs

Offre

Logout

Ajouter une offre:

Add Offer

Offres existantes:

Solo

1 personne

25.00€

- 0 +

Add to Cart

Duo

2 personnes

45.00€

- 0 +

Add to Cart

Family

4 personnes

85.00€

- 0 +

Add to Cart

Family+

6 personnes

110.00€

- 0 +

Add to Cart

Ajouter une nouvelle offre

Liste des offres

Offres existantes:

Solo

1 personne

25.00€

- 0 +

Add to Cart

Duo

2 personnes

45.00€

- 0 +

Add to Cart

Family

4 personnes

85.00€


- 0 +

Add to Cart

Family+

6 personnes

110.00€

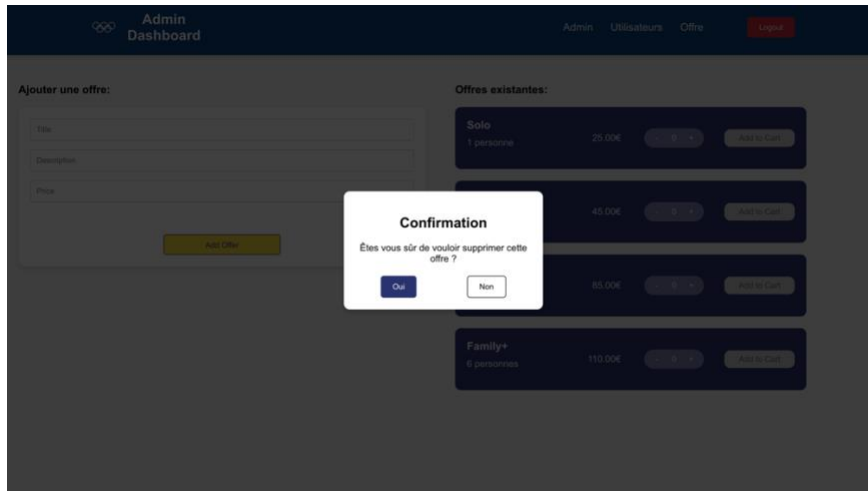


- 0 +

Add to Cart

Hover l'offre pour la supprimer





Confirmation pour supprimer l'offre.

#### 4. Tests:

Pour lancer les tests il suffit d'enlever les commentaires pour le développement local et mettre en commentaire la production Heroku dans le fichier .env:

```
exam > .env
1 SECRET_KEY='django-insecure-5ftc@ot()mju46$sw5h#ar_d^3oe8=ym^3nky@uj=~^9pe^$%5y'
2
3 ALLOWED_HOSTS='localhost,127.0.0.1,bloc3exam-a2922cc2f685.herokuapp.com/'
4 DJANGO_SETTINGS_MODULE=django_jo.settings
5 REACT_APP_API_BASE_URL=https://bloc3exam-a2922cc2f685.herokuapp.com
6
7
8 #Developpement local:
9 #DATABASE_URL='postgres://AdminStudi:Bloc3exam2024@localhost:5432/AdminStudi'
10 #DB_NAME=AdminStudi
11 #USER=AdminStudi
12 #PASSWORD=Bloc3exam2024
13 #HOST=localhost
14 #HTTPS=false
15 #DEBUG=true
16
17
18 #Production Heroku:
19 DATABASE_URL='postgres://u368vaaq04pve0:p90608fb7ca023c9d96b62ada42424823c71276389b6c36398c075db634402a44@ce0lkuo944ch9
20 DB_NAME=d6o59a1vr7apit
21 USER=u368vaaq04pve0
22 PASSWORD=p90608fb7ca023c9d96b62ada42424823c71276389b6c36398c075db634402a44
23 HOST=ce0lkuo944ch99.cluster-czrs8kj4isg7.us-east-1.rds.amazonaws.com
24 HOST=localhost
25 HTTPS=true
26 DEBUG=False
27
28 # SSL_CERT_FILE=./localhost.pem
29 # SSL_KEY_FILE=./localhost-key.pem
30
```

Enlever les commentaires

Mettre en commentaire

Dans settings.py, mettre en commentaire les données ci-dessous :

```
138
139 # Developpement Heroku (mettre en commentaire pour les Tests)
140 DATABASES = {
141     'default': {
142         'ENGINE': 'django.db.backends.postgresql',
143         'USER': 'ubs9h052rggt2m',
144         'PASSWORD': 'p459fc3862d6dc15e569eaa3e264768d034d1f3ce5e301706a25e97087f0d1ee0',
145         'HOST': 'c7u1tn6bvvsodf.cluster-czz5s0kz4scl.eu-west-1.rds.amazonaws.com',
146         'PORT': '5432',
147         'NAME': 'd412olpe262c9o',
148     }
149 }
150
```

Les tests effectués servent à tester les Endpoints pour les offres, achats et utilisateurs :

### 1. Pour les offres

- Créer une offre : Vérifie qu'un utilisateur authentifié peut créer une nouvelle offre. On s'attend ici à un statut "201 CREATED".
- Supprimer une offre : Vérifie qu'une offre peut être supprimée par un utilisateur authentifié (statut 204 NO CONTENT attendu).
- Obtenir une offre : Confirme qu'une offre peut être récupérée (statut 200 OK attendu).
- Mettre à jour une offre : Teste la mise à jour d'une offre existante (statut 200 OK attendu).
- Accès non autorisé : S'assure que les utilisateurs non authentifiés ne peuvent pas créer, mettre à jour ou supprimer des offres (statut 401 UNAUTHORIZED attendu).

### 2. Pour les achats

- Créer un achat : Valide qu'un utilisateur authentifié peut créer un achat (statut 200 OK attendu).
- Lister les achats : Vérifie que les achats peuvent être listés par un utilisateur authentifié (statut 200 OK attendu).
- Accès par le personnel : S'assure qu'un membre du personnel peut lister tous les achats (statut 200 OK attendu).
- Accès non autorisé : Confirme que les utilisateurs non authentifiés ne peuvent pas créer des achats (statut 401 UNAUTHORIZED attendu).

### 3. Pour les utilisateurs:

- Lister les utilisateurs : Vérifie que les utilisateurs peuvent être listés (statut 200 OK attendu).
- Créer un utilisateur : Teste la création d'un nouvel utilisateur (201 CREATED attendu).
- Supprimer un utilisateur : S'assure qu'un utilisateur peut être supprimé (204 NO CONTENT attendu).

Résultats des 15 tests effectués:

```
Destroying test database for alias 'default'...
theodiloreto@Theos-MacBook-Pro exam % python manage.py test

Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
-----
Ran 15 tests in 2.836s

OK
Destroying test database for alias 'default'...
theodiloreto@Theos-MacBook-Pro exam %
```