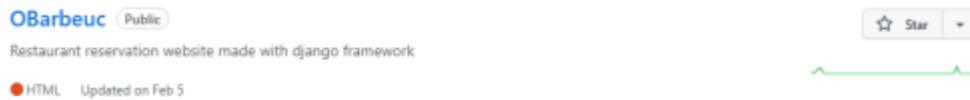


## Gestion du patrimoine Obarbeuc

### Procédure d'utilisation d'une solution de gestion de version : Github

1 ) Créer un dépôt pour le projet : une fois connecté à GitHub, cliquer sur le bouton "New" pour créer un nouveau dépôt. Donner un nom à au dépôt (Dans notre cas Obarbeuc).



2 ) Initialiser le dépôt avec les fichiers du projet : ouvrir une invite de commande et naviguer jusqu'au répertoire contenant les fichiers du projet Obarbeuc. Puis entrer la commande suivante pour initialiser le dépôt Git :

```
git init
```

3 ) Ajouter les fichiers au dépôt : utiliser la commande git add pour ajouter les fichiers de votre projet au dépôt. Par exemple, pour ajouter tous les fichiers de votre projet Obarbeuc, entrez la commande suivante :

```
git add
```

4 ) Créer un commit : une fois que tous les fichiers sont ajoutés, utilisez la commande git commit pour créer un commit qui représente l'état actuel de votre projet. Assurez-vous d'ajouter un message clair qui décrit les modifications apportées au projet.

```
git commit -m "Ajout du framework Django et du squelette du projet"
```

5 ) Ajouter le dépôt distant : pour synchroniser le dépôt local avec le dépôt GitHub, utilisez la commande git remoteadd suivi du lien du dépôt.

```
git remote add origin https://github.com/Yannis-Alouache/Obarbeuc.git
```

6 ) Pousser les modifications vers GitHub : utilisez la commande git push pour pousser les modifications vers votre dépôt distant.

```
git push -u origin master
```

7 ) Mettre à jour le dépôt : pour apporter des modifications au projet Obarbeuc, utiliser les commandes git add et git commit pour ajouter et enregistrer ces modifications dans votre dépôt local. Une fois que je suis satisfait des modifications, j'utilise la commande git push pour mettre à jour le dépôt distant sur GitHub.

```
git add .
git commit -m "Ajout de fonctionnalités à Obarbeuc"
git push
```

### Les Normes pour les messages de commit

Pour les messages de commit j'ai utilisé une norme qui permet de se repérer facilement et de suivre efficacement les modifications apportées au projet.

Pour les ajouts de code :

```
git add .
git commit -m "[+] added reservation module"
git push
```

Pour la suppression de code :

```
git add .
git commit -m "[-] removed files"
git push
```

Pour la modification de code :

```
git add .
git commit -m "[~] modified reservation module"
git push
```

Les Normes de programmation

- Utilisez des outils de vérification de code tels que Pylint pour détecter les erreurs potentielles et les problèmes de style de code.
- Utilisez les messages d'erreurs de Django pour fournir des informations aux utilisateurs. Évitez d'utiliser des messages d'erreur génériques et fournissez des messages explicites et utiles.
- Respectez le modèle MVC (Model-View-Controller) de Django. Évitez de placer de la logique métier dans vos templates et de mettre trop de code dans vos vues.
- Utilisez des noms de variables et de fonctions explicites et descriptifs. Évitez d'utiliser des noms courts ou ambigus.

Pour les fonctions et procédures : première lettre du premier mot en minuscule ensuite première lettre des autres mots en Majuscule

```
def addReservation():
    ...
```

Pour les classes : Première lettre en majuscule





```
class ReservationForm(forms.ModelForm):
    ...
```

Pour les variables : première lettre du premier mot en minuscule ensuite première lettre des autres mots en Majuscule

```
myVariable = ...
```

## **Respect des normes Github**

Commits on Mar 1, 2022

[+]reservation et limite de table RedCHK committed on Mar 1, 2022	 f9376a7	◀
[+] Ajout d'un lien vers le compte Facebook  Algamor59 committed on Mar 1, 2022	 3204cde	◀
[+] changement du texte de la page d'accueil RedCHK committed on Mar 1, 2022	 5a2c6ae	◀

J'ai réussi à respecter les normes établies pour les commits sur GitHub. J'ai pris soin d'écrire des messages de commit concis et précis, en fournissant des informations utiles sur les changements que j'ai apportés au code. En utilisant une syntaxe cohérente et en suivant les conventions établies pour la structure des messages de commit, j'ai pu rendre le processus de collaboration plus facile pour moi-même et pour mes collègues de travail. J'espère continuer à suivre les normes établies pour les commits de mes projets à l'avenir.

### Respect des normes de Programmation

```
class Account(AbstractBaseUser):
    email = models.EmailField(
        verbose_name="email",
        max_length=255,
        unique=True,
        error_messages={'unique': 'Un compte existe déjà avec cette email'})
    ...
```

```
def homeView(request):
    todayLunchs = TodayLunch.objects.all()
    context = {
        'todayLunchs' : todayLunchs
    }
    return render(request, "home.html", context)
```

J'ai réussi à garder un environnement de code sain en respectant les norme de programmation que j'avais prévu.

# Développer la présence d'Obarbeuc

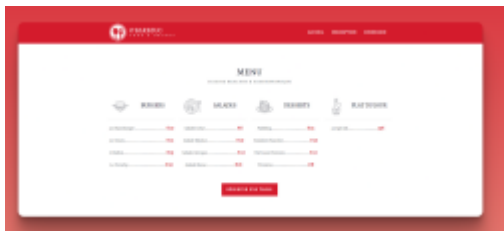
## Documentation Technique

Le site web Obarbeuc est un site de restauration en ligne développé en Python avec le framework Django. Ce document décrit les fonctionnalités du site, les technologies utilisées et les processus de développement et de déploiement.

- Connexion et inscription : Les utilisateurs peuvent créer un compte et se connecter pour accéder aux fonctionnalités du site.



- Consultation du menu : Les utilisateurs peuvent consulter le menu complet du restaurant avec les prix et les descriptions.



- Réservation : Les utilisateurs peuvent réserver une table en choisissant la date et l'heure de leur choix.



- Consultation des avis client : Les utilisateurs peuvent consulter les avis laissés par les autres clients pour avoir une idée sur la qualité des plats proposés.

La solution web Obarbeuc est développée en Python avec le framework Django.



Les autres technologies utilisées sont : HTML, CSS et JavaScript pour la partie front-end.



MySQL pour la base de données.



Git pour le contrôle de version.



Xampp pour le “déploiement” local.



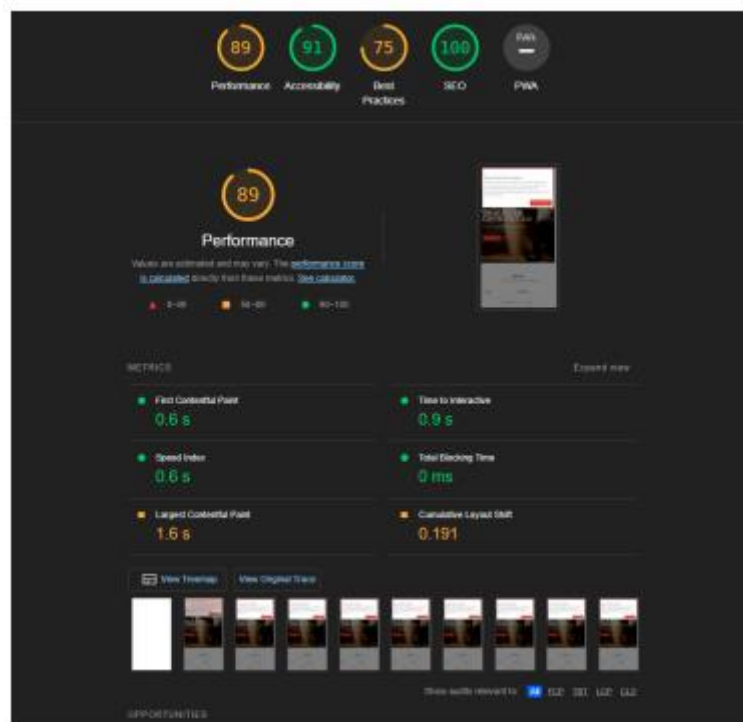
Google Lighthouse est un outil open-source d'audit et de test de performance web. Cet outil, créé par Google, permet aux développeurs et aux propriétaires de sites web d'évaluer la qualité et les performances de leur site en fournissant des recommandations pour améliorer la vitesse, l'accessibilité, la qualité et la sécurité du site web.

Google Lighthouse évalue la performance d'un site web en se basant sur plusieurs critères tels que :

La performance : la vitesse de chargement des pages et les optimisations liées à la performance

L'accessibilité : l'accessibilité du site pour les personnes en situation de handicap et les recommandations pour l'améliorer  
Le référencement : les recommandations pour améliorer le référencement du site

La qualité : la qualité du code HTML, CSS et JavaScript, la détection des erreurs et des Avertissements.



# Réalisation de Test Unitaire Django

## Introduction

Le projet Django Obarbeuc est un projet important pour la mise en place d'une application Web fiable et robuste dans le cadre de la réservation de table de restauration. Pour garantir la qualité et la fiabilité de l'application, l'utilisation des tests unitaires est indispensable.

## Les Tests Unitaire

Dans le cadre de ce projet, la classe TestCase de Django a été utilisée pour créer des tests unitaires.

```
class TestCase(TransactionTestCase):
    @classmethod
    def _enter_atomics(cls):
        """Open atomic blocks for multiple databases."""
        atomics = {}
        for db_name in cls._databases_names():
            atomics[db_name] = transaction.atomic(using=db_name)
            atomics[db_name].enter()
        return atomics
    ...
```

Un test est divisés en deux parties :

- SetUp : Initialise les méthodes et les données à tester
- Run : Lance le test et le compare avec le résultat attendu.

Un test est en fait une classe personnalisé composé de deux méthode qui hérite de la classe TestCase

```
class TodayLunchTestCase(TestCase):
    def setUp(self):
        TodayLunch.objects.create(title="Cheeseburger", price=35)
        TodayLunch.objects.create(title="SioBurger", price=25)
        TodayLunch.objects.create(title="BigBurger", price=10)

    def test_true_price(self):
        todayLunch_0 = TodayLunch.objects.get(title="Cheeseburger")
        self.assertEqual(todayLunch_0.getPrice(), 35)
    ....
```

Les tests ont été organisés dans un fichier nommé test.py.



La vérification des résultats a été effectuée à l'aide de la fonction assertEquals qui prend deux paramètres : la donnée et le résultat escompté.

Si la donnée est égale au résultat attendu le test est positif sinon on reçoit une notification sur ce qui à causer l'échec du test unitaire.

```
self.assertEqual(Notre Donnée, Le resultat attendu)
```

Dans le cadre de ce projet, les tests unitaires ont été utilisés pour tester les modèles et les urls de l'application. Cela permet de s'assurer que la logique derrière le code est appropriée et fonctionnera dans tous les cas.

Admettons que je lance ce test unitaire :

```
class UrlTestCase(TestCase):
    def test_url(self):
        response = self.client.get('/reservation')
        self.assertEqual(response.status_code, 200)
```

J'essaye d'accéder à la page réservation sans être connecté ce qui va me bloquer et me renvoyer en code de redirection 302.

```
=====
FAIL: test_url (main.tests.UrlTestCase)
-----
Traceback (most recent call last):
  File "C:\Users\chill\Desktop\dev\Obarbeuc\obarbeuc\main\tests.py", line 12, in test_url
    self.assertEqual(response.status_code, 200)
AssertionError: 302 != 200
-----
Ran 1 test in 0.033s
```

Comme nous le voyons avec "AssertionError: 302 != 200"

La page nous a renvoyé le code 302 au lieu du code 200 qui était attendu.

### Axe d'amélioration

Pour améliorer l'utilisation des tests unitaires dans le projet Obarbeuc, J'aurais pu éventuellement ajouter des tests pour les vues et les formulaires. Les tests de vues et de formulaires sont importants pour garantir le bon fonctionnement de l'application dans son ensemble. J'aurais aussi pu faire plus de tests pour couvrir tous les aspects du code de l'application.