

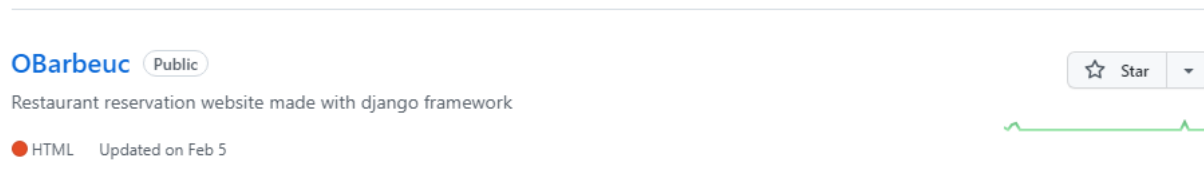


Gestion du patrimoine Obarbeuc

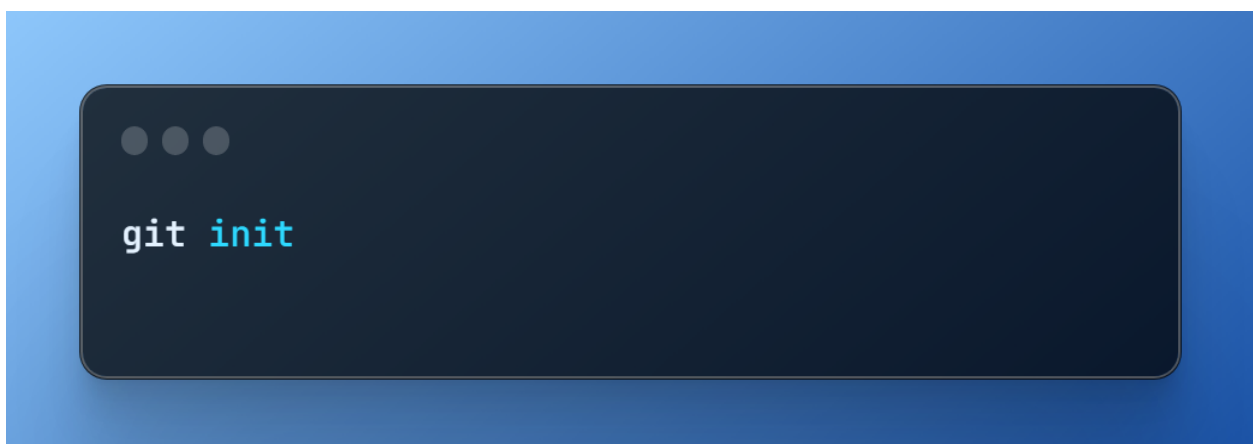
Alouache Yannis

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

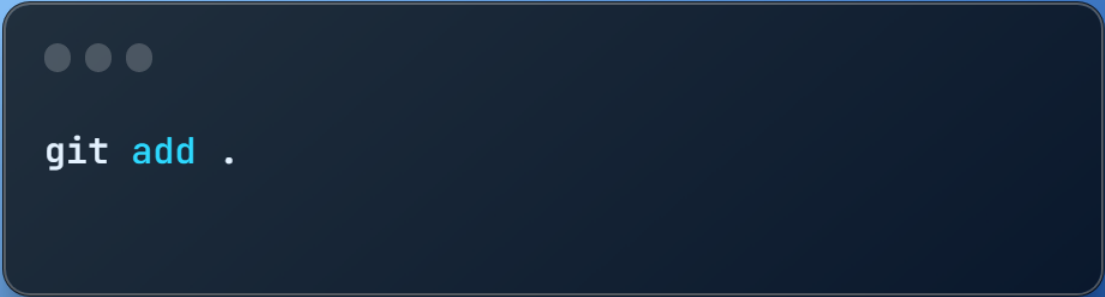
1) Créer un dépôt pour le projet : une fois connecté à GitHub, cliquer sur le bouton "New" pour créer un nouveau dépôt. Donner un nom à au dépôt (Dans notre cas Obarbeuc).



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 :



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 remote add` suivi du lien du dépôt.

A terminal window with a dark blue background and three small circles in the top left corner. It displays the command `git remote add origin https://github.com/Yannis-Alouache/0Barbeuc.git` in a light blue monospace font.

```
git remote add origin https://github.com/Yannis-Alouache/0Barbeuc.git
```

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

A terminal window with a dark blue background and three small circles in the top left corner. It displays the command `git push -u origin master` in a light blue monospace font.

```
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.


A terminal window with a dark blue background and rounded corners, set against a light blue gradient background. The window has three small grey circles in the top-left corner. It contains three lines of text in a monospaced font: 'git add .' on the first line, 'git commit -m "Ajout de fonctionnalités à Obarbeuc"' on the second line, and 'git push' on the third line.

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

Les Normes pour les message de commit

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

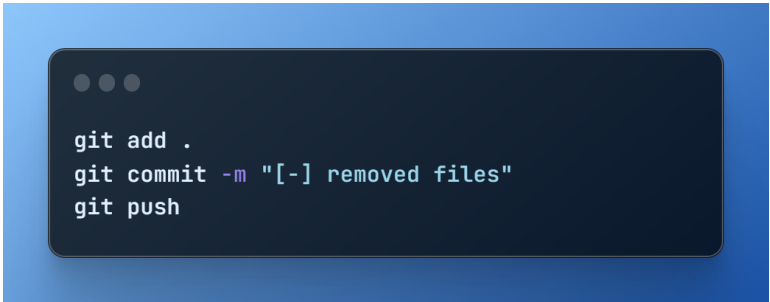
Pour les ajouts de code :

A terminal window with a dark blue background and three window control buttons in the top left. It contains the following text:

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

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

Pour les suppression de code :

A terminal window with a dark blue background and three window control buttons in the top left. It contains the following text:

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

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

Pour les modification de code :

A terminal window with a dark blue background and three window control buttons in the top left. It contains the following text:

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

```
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 fonction et procedure : première lettre du premier mot en minuscule
ensuite première lettre des autres mots en Majuscule

A code editor window with a dark background and blue border. It contains the text:

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

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

Pour les classes : Première lettre en majuscule

A code editor window with a dark background and blue border. It contains the text:

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

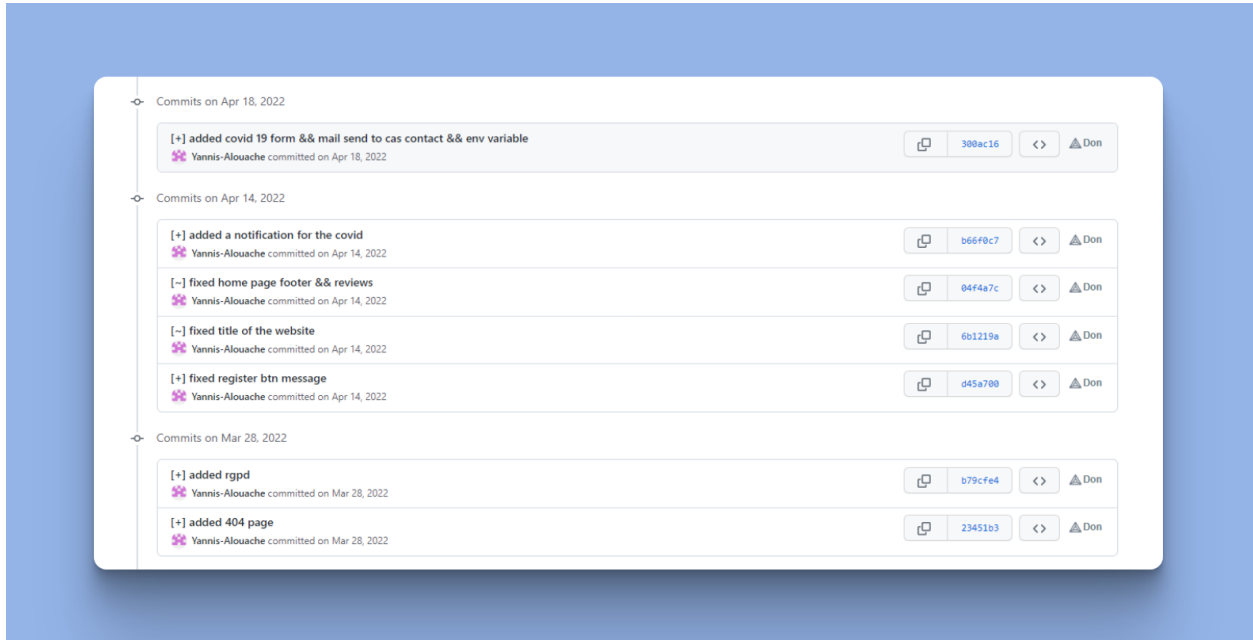
```
class ReservationForm(forms.ModelForm):  
    ...
```

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



```
myVariable = ...
```


Respect des normes Github



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.