

TP8 – Supprimer et modifier (pizza et ingrédient)

Contexte et Objectifs

Nous savons maintenant créer des formulaires, les récupérer, alimenter la base de données, et la lire. Autrement dit, les aspects C (create) et R (read) du CRUD. Il nous reste à voir les aspects U (update) et D (delete).

Le premier objectif est de supprimer une pizza. Ensuite, nous verrons comment modifier une pizza. Puis, nous adapterons tout ceci aux ingrédients.

La dernière étape sera la modification d'une composition ou sa suppression.

Les liens de modification et de suppression dans pizzas.html

Dans le template pizzas.html, créer pour chaque pizza listée un lien de modification et un lien de suppression personnalisés, de la forme :

```
<a href="/pizzas/{{piz.idPizza}}/">détails</a>  
<a href="/pizzas/{{piz.idPizza}}/update/">modifier</a>  
<a href="/pizzas/{{piz.idPizza}}/delete/">supprimer</a>
```

Les urls correspondantes dans urls.py et les vues dans view.py n'ont pas encore été créées, c'est la suite du programme.

L'url de suppression dans urls.py

Créez, dans urls.py, une nouvelle ligne path qui fera correspondre le lien précédent de suppression à la vue (pas encore créée) supprimerPizza(request, pizza_id). Ce path sera de la forme

```
path('pizzas/<int:pizza_id>/delete/', views.supprimerPizza),
```

La « vue » de suppression dans views.py

1. Créez la vue `supprimerPizza(request, pizza_id)` dans le fichier `views.py`. Cette vue devra :
 - a. récupérer la pizza à supprimer (grâce à la méthode `get`),
 - b. appeler la méthode `delete()` sur cette pizza,
 - c. récupérer la liste de toutes les pizzas grâce à la méthode `all()` comme dans la vue `pizzas`,
 - d. appeler le template `pizzas.html` en lui fournissant la liste des pizzas (comme dans la vue `pizzas`).
2. Vérifiez qu'après avoir créé une pizza de test, sans ingrédient ajouté, vous pouvez la supprimer. Contrôlez l'effet dans `sqlitebrowser`.
3. Créez une nouvelle pizza, ajoutez-lui des ingrédients, et vérifiez avec `sqlitebrowser` que les ingrédients sont apparus dans la composition de la pizza créée. Puis supprimez la pizza, et vérifiez que les lignes correspondant à cette pizza dans la table `Composition` ont aussi disparu, ainsi que la pizza dans la table `Pizza`.

L'url de création du formulaire de modification dans urls.py

Créez, dans `urls.py`, une nouvelle ligne `path` qui fera correspondre le lien précédent de modification à la vue (pas encore créée)

```
afficherFormulaireModificationPizza(request, pizza_id)
```

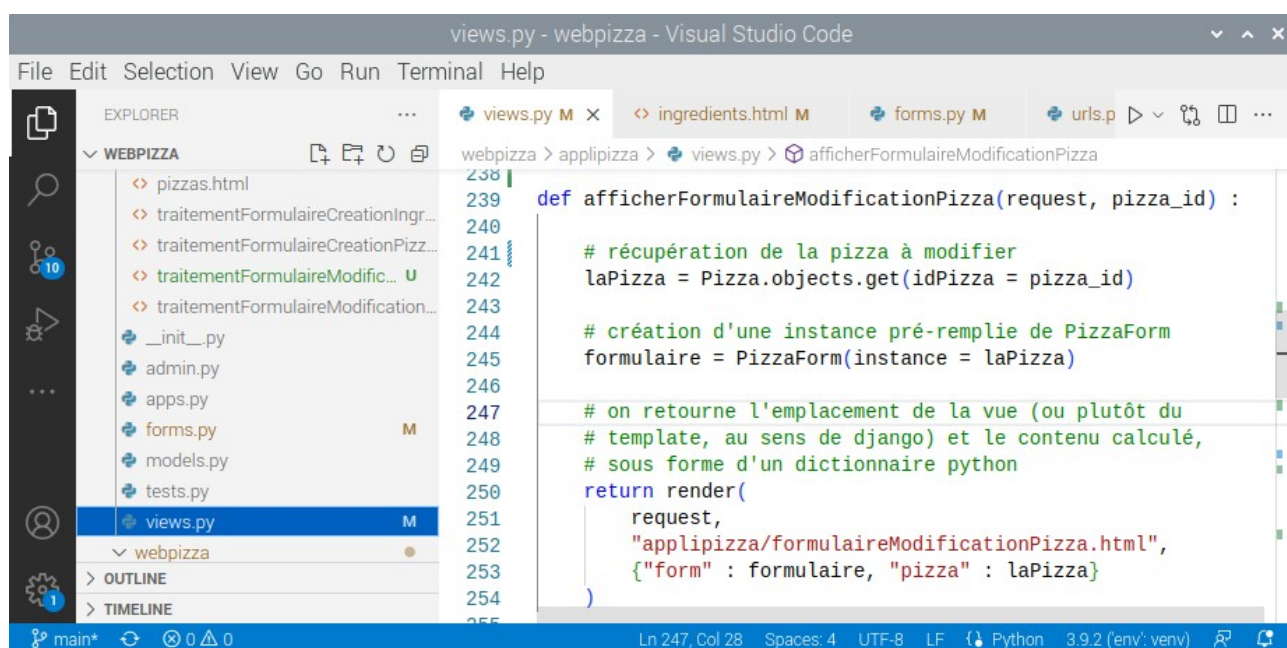
Ce `path` sera de la forme

```
path('pizzas/<int:pizza_id>/update/', views.afficherFormulaireModificationPizza),
```

La « vue » formulaire de modification dans views.py

Créez la vue `afficherFormulaireModificationPizza(request, pizza_id)` dans le fichier `views.py`. Cette vue devra :

- recupérer la pizza à afficher dans le formulaire (grâce à la méthode `get`),
- créer un formulaire `PizzaForm`, en lui passant comme instance la pizza à modifier,
- appeler `formulaireModificationpizza.html`, le template qui sera chargé d'afficher le formulaire prérempli.

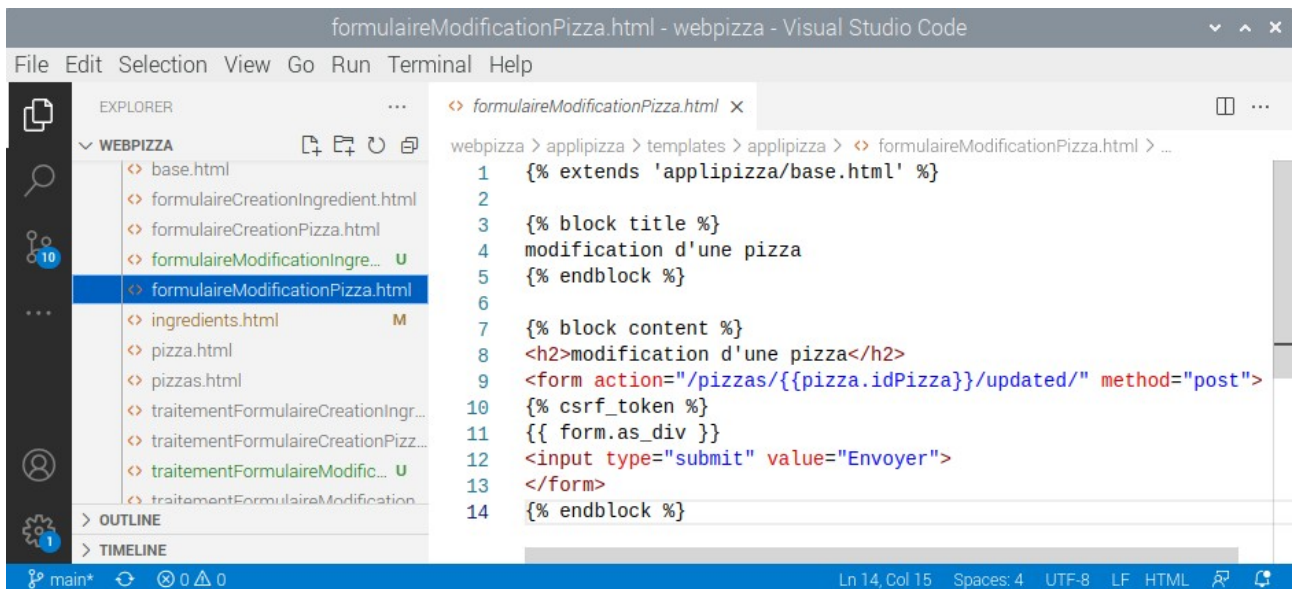


The screenshot shows the Visual Studio Code editor with the file `views.py` open. The Explorer sidebar on the left shows the project structure for `webpizza`, including files like `pizzas.html`, `traitementFormulaireCreationIngr...`, `traitementFormulaireCreationPizz...`, `traitementFormulaireModific...`, `traitementFormulaireModification...`, `__init__.py`, `admin.py`, `apps.py`, `forms.py`, `models.py`, `tests.py`, and `views.py`. The main editor area shows the code for the `afficherFormulaireModificationPizza` function. The code is as follows:

```
def afficherFormulaireModificationPizza(request, pizza_id) :  
    # récupération de la pizza à modifier  
    laPizza = Pizza.objects.get(idPizza = pizza_id)  
  
    # création d'une instance pré-remplie de PizzaForm  
    formulaire = PizzaForm(instance = laPizza)  
  
    # on retourne l'emplacement de la vue (ou plutôt du  
    # template, au sens de django) et le contenu calculé,  
    # sous forme d'un dictionnaire python  
    return render(  
        request,  
        "applipizza/formulaireModificationPizza.html",  
        {"form" : formulaire, "pizza" : laPizza}  
    )
```

Le template formulaireModificationPizza.html

Créez le template `formulaireModificationPizza.html`. Il affiche le formulaire calculé par la vue précédente, en indiquant un lien de traitement « updated » :



L'url de modification dans urls.py

Créez, dans `urls.py`, une nouvelle ligne path qui fera correspondre le lien précédent de modification à la vue (pas encore créée)

```
modifierPizza(request, pizza_id)
```

Ce path sera de la forme

```
path('pizzas/<int:pizza_id>/updated/', views.modifierPizza),
```

La « vue » modifierPizza dans views.py

Créez la vue `modifierPizza(request, pizza_id)` dans le fichier `views.py`. Cette vue devra :

- recupérer la pizza à modifier (grâce à la méthode `get`),
- recupérer le formulaire posté, avec pour instance la pizza récupérée,
- si le formulaire est valide, appeler sur lui la méthode `save()`, ce qui aura pour effet de mettre la pizza à jour dans la base de données,
- aller rechercher dans la base la pizza modifiée,

e. appeler `traitementFormulaireModificationpizza.html`, le template qui sera chargé d'afficher un message de confirmation de la modification (en précisant le nom de la pizza modifiée).

Le template `traitementFormulaireModificationPizza.html`

Créez le template `traitementFormulaireModificationPizza.html`. Il confirme simplement que la pizza (dont on précise le nom) a bien été modifiée.

Vérifiez maintenant que tout fonctionne.

Et pour les ingrédients ?

Recommencez tout pour les ingrédients. Le seul détail qui change est que justement, on n'affichera pas la vue détail d'un ingrédient. Donc on aura ce type de visuel :

