

EasyChef URLs

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

Accounts URLs

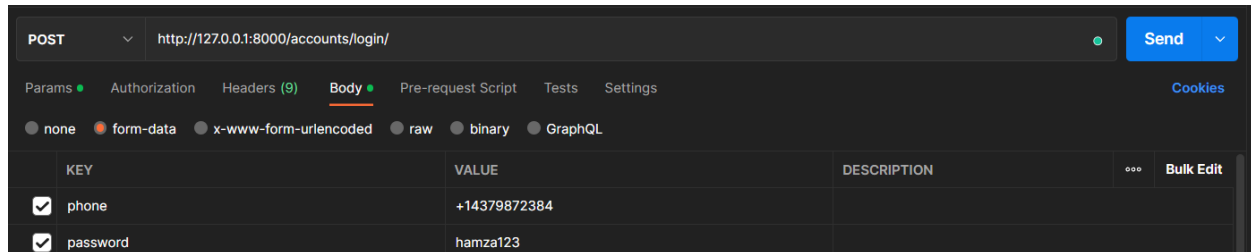
```
urlpatterns = [
    path('login/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('profile/', GetProfileView.as_view()),
    path('register/', RegisterView.as_view()),
    path('edit-profile/<int:pk>/', EditProfileView.as_view()),
]
```

Recipe URLs

```
urlpatterns = [
    path('create/', CreateRecipeView.as_view()),
    path('details/<int:recipe_id>/', GetRecipeView.as_view()),
    path('create/comment/<int:recipe_id>/', CreateCommentView.as_view()),
    path('like/<int:recipe_id>/', LikeRecipeView.as_view()),
    path('edit/<int:id>/', EditRecipeView.as_view()),
    path('delete/<int:recipe_id>/', DeleteRecipeView.as_view()),
    path('search/', SearchRecipeView.as_view()),
    path('favourite/<int:recipe_id>/', FavouriteRecipeView.as_view()),
    path('rate/<int:recipe_id>/', RateRecipeView.as_view())
]
```

Postman URL Guide

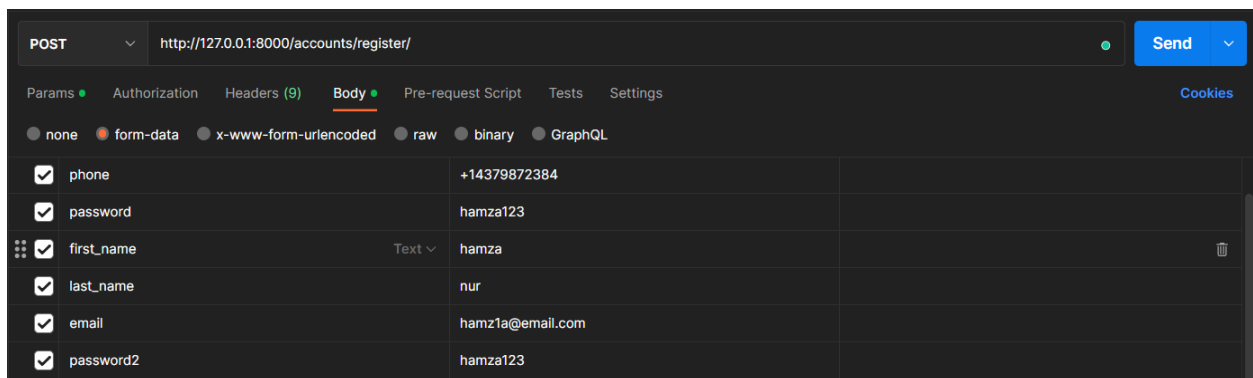
The login/ URL takes in a phone and password field in form-data to log the user in. **(POST)**



The profile/ URL displays the currently logged-in user's info. If the user is not logged in an authentication error will be returned. **(GET)**

```
1 {
2   "id": 2,
3   "first_name": "hamza",
4   "last_name": "nur",
5   "email": "hamz1a@email.com",
6   "phone": "+14379872384"
7 }
```

The register/ URL takes in a phone, first_name, last_name, email, password and password2. These are all required fields where password and password2 also must match in order to register this account. **(POST)**



The edit-profile/<int:pk>/ URL is similar to the register/ URL but instead of creating a new user, it just updates the user's info. The user profile is the one specified in the URL, "pk", Simply send in your new details, and if any field is not included, it will simply remain unchanged. If the specified user does not exist, an error will be returned. IsAuthenticated (**PUT**)

The screenshot shows a REST client interface with a PUT request to `http://127.0.0.1:8000/accounts/edit-profile/2/`. The 'Body' tab is selected, and the data is formatted as 'form-data'. The request body contains the following fields:

Key	Value	Description
password	hamza123	
first_name	hamza	
last_name	nur	
email	hamza1a@email.com	
password2	hamza123	

The create/ URL creates a recipe with some required fields. The cuisine input must be one of the cuisine options: (Mexican, Italian, Greek, Spanish, Indian, American, Somalian, Algerian, Turkish, Japanese, Chinese, Vietnamese), any other option would throw an error. Diets can be multiple selections, and are done so by inputting multiple diet fields as seen below. The diets must also be one of the diet options: (Halal, Kosher, Vegan, Vegetarian, Gluten-Free, Dairy-Free, Keto, Carnivore, Pescatarian). Ingredient can also have multiple inputs by simply sending in multiple ingredient fields as seen below. Instructions follow a certain format: {number}g of {ingredient}. Instruction can also have multiple inputs by simply sending in multiple instruction fields. There are no formatting requirements for this field. IsAuthenticated (**POST**)

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:8000/recipes/create/`. The 'Body' tab is selected, and the data is formatted as 'form-data'. The request body contains the following fields:

KEY	VALUE
title	Omelette
description	In cuisine, an omelette is a dish made from beaten eggs, fried with
cuisine	American
diet	Halal
diet	Vegetarian
prep_time	5
servings	3
instruction	Break the mulitple egg
instruction	Put 5 grames of salt
ingredient	5g of salt
ingredient	10g of oil
image	Select Files

The details/<int:recipe_id>/ URL displays the details of the specified recipe “recipe_id” that is passed in through the URL. If the specified recipe does not exist, an error will be returned.

IsAuthenticated (**GET**)

```
1  {
2    "owner_name": "hamza nur",
3    "title": "Omelette",
4    "description": "In cuisine, an omelette is a dish made from beaten eggs, fried with butter or oil in a frying pan",
5    "cuisine": "American",
6    "image": null,
7    "prep_time": "5",
8    "servings": "3",
9    "instruction": [
10     "Break the egg",
11     "Put 5 grams of salt"
12   ],
13   "ingredient": [
14     "5g of salt",
15     "10g of oil"
16   ],
17   "diet": [
18     "Halal",
19     "Vegetarian"
20   ],
21   "comment": [],
22   "like": 0,
23   "rating": 0
24 }
```

The create/comment/<int:recipe_id>/ URL creates a comment on the recipe “recipe_id” that is passed in through the URL. If the specified recipe does not exist, an error will be returned.

IsAuthenticated (**POST**)

POST	http://127.0.0.1:8000/recipes/create/comment/5/					
Params	Authorization	Headers (10)	Body	Pre-request Script	Tests	Settings
<input type="radio"/> none	<input checked="" type="radio"/> form-data	<input type="radio"/> x-www-form-urlencoded	<input type="radio"/> raw	<input type="radio"/> binary	<input type="radio"/> GraphQL	
	KEY					VALUE
<input checked="" type="checkbox"/>	comment	Text				Instructions unclear, my wife left me.

The like/<int:recipe_id>/ URL likes the recipe “recipe_id” that is passed in through the URL. If the same user likes the recipe again, it will unlike the recipe. And of course if the user likes again it will re-like the recipe. If the specified recipe does not exist, an error will be returned.

IsAuthenticated (**POST**)

The edit/<int:id>/ URL modifies the recipe “id” that is passed in through the URL. This is very similar to create recipe but if fields are not sent it will remain unchanged. If the specified recipe does not exist, an error will be returned. IsAuthenticated (**PUT**)

PUT http://127.0.0.1:8000/recipes/edit/5/

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

KEY	VALUE
<input checked="" type="checkbox"/> title	Omelette
<input checked="" type="checkbox"/> description	EDITED In cuisine, an omelette is a dish made from beaten eggs, fri
<input checked="" type="checkbox"/> cuisine	American
<input checked="" type="checkbox"/> diet	Halal
<input checked="" type="checkbox"/> diet	Vegetarian
<input checked="" type="checkbox"/> prep_time	5
<input checked="" type="checkbox"/> servings	3
<input checked="" type="checkbox"/> instruction	Break the multiple egg
<input checked="" type="checkbox"/> instruction	Put 5 grams of salt
<input checked="" type="checkbox"/> ingredient	5g of salt
<input checked="" type="checkbox"/> ingredient	10g of oil
<input checked="" type="checkbox"/> image	Select Files

The delete/<int:recipe_id>/ URL deletes the recipe “recipe_id” that is passed in through the URL. If the specified recipe does not exist, an error will be returned. IsAuthenticated (**DELETE**)

The search/ URL allows you to search all recipes and filter by title, diet, ingredient, cuisine, owner_name and prep_time. First, navigate to params and simply enter in the field you want to filter. For ingredient pass in “ingredient__ingredient” into the key and for diet pass in “diet__diet” into the key. The search field below can be used to filter by title or owner_name. IsAuthenticated (**GET**)

The favourite/<int:recipe_id>/ URL favourites the recipe “recipe_id” that is passed in through the URL. If the specified recipe does not exist, an error will be returned. If the same user favourites this recipe again, it will un-favourite the recipe for them. And of course if the user favourites this recipe again it will favourite the recipe again. IsAuthenticated (**POST**)

GET http://127.0.0.1:8000/recipes/search/?diet__diet=Carnivore&ingredient__ingredient=mango&prep_time=120&search=Cheese cake&Cuisine=American

Params **Authorization** Headers (9) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
<input checked="" type="checkbox"/> diet__diet	Carnivore
<input checked="" type="checkbox"/> ingredient__ingredient	mango
<input checked="" type="checkbox"/> prep_time	120
<input checked="" type="checkbox"/> search	Cheese cake
<input checked="" type="checkbox"/> Cuisine	American
Key	Value

The `rate/<int:recipe_id>/` URL rates the recipe “recipe_id” that is passed in through the URL. If the specified recipe does not exist, an error will be returned. In order to rate the recipe, you must send in a rating field with an integer between 1 and 5. If the user rates the recipe again, the previous rating will be overwritten with the new rating. Recipe_id. IsAuthenticated (**POST**)

POST	http://127.0.0.1:8000/recipes/rate/5/	
Params	Authorization	Headers (10)
Body		
Pre-request Script		
Tests		
Settings		
none form-data x-www-form-urlencoded raw binary GraphQL		
KEY	VALUE	
<input checked="" type="checkbox"/> rating	5	

The `get-shopping-list/` URL displays the current user’s shopping list. Containing the total of all ingredients from all recipes added to the shopping list. IsAuthenticated (**GET**)

```
1  [
2    "Recipes created": [
3      [
4        16,
5        "Chocolate Cheesecake"
6      ],
7      [
8        17,
9        "Chocolate Cheesecake"
10     ],
11     [
12       18,
13       "Chocolate Cheesecake"
14     ],
15     [
16       19,
17       "Chocolate Cheesecake"
18     ]
19   ],
20   "Recipes favourited": [
21     [
22       1,
23       "Chocolate Cheesecake"
24     ]
25   ],
26   "Recipes liked": [],
27   "Recipes commented on": [
28     [
29       17,
30       "Chocolate Cheesecake"
31     ],
32     [
33       1,
34       "Chocolate Cheesecake"
35     ]
36   ],
37   "Recipes rated": [
38     [
39       1,
40       "Chocolate Cheesecake"
41     ]
42   ]
43 ]
```

The add-to-shopping-list/<int:recipe_id>/ URL adds the recipe with id 'recipe_id' to the current user's shopping list. If the specified recipe does not exist, an error will be returned.

IsAuthenticated **(POST)**

The my-recipes/ URL displays the current user's recipes as well as all the favourited, liked, commented on and rated recipes of the current user. IsAuthenticated **(GET)**

```
1  {
2      "Recipes": [
3          "Chocolate Cheesecake",
4          "Chocolate Cheesecake",
5          "Chocolate Cheesecake"
6      ],
7      "Total": [
8          "1500g of chocolate",
9          "750g of cheese"
10     ]
11 }
```

Accounts Model

```
class NewUser(AbstractUser):
    phone = PhoneNumberField(null=False, blank=False, unique=True)
    username = None
    USERNAME_FIELD = 'phone'
    EMAIL_FIELD = 'email'
    REQUIRED_FIELDS = ['first_name', 'last_name', 'password', 'email']
```

The model defines a custom user model NewUser that inherits from Django's built-in AbstractUser model. The NewUser model adds two new fields to the user model: phone and avatar.

The phone field is of type PhoneNumberField from the phonenumber_field package, which stores phone numbers as a string with the country code. This field is required and unique for each user.

The avatar field is of type ImageField from Django's models module, which stores image files uploaded by users. The field has a default value of 'uploads/default4', which is a default image displayed when the user has not uploaded their own image. The upload_to argument specifies the directory where uploaded images will be stored.

The username field is set to None to indicate that the user should be identified by their phone number instead of a username.

The USERNAME_FIELD is set to 'phone' to specify that the phone field is used for authentication instead of the username field.

The EMAIL_FIELD is set to 'email' to specify the email field for the user model.

The REQUIRED_FIELDS specifies the fields that are required when creating a new user, which are 'first_name', 'last_name', 'password', and 'email'.

Overall, the model is designed to create a custom user model that uses phone numbers for authentication and stores user avatar images.

Recipe Model

```
from django.contrib.auth import get_user_model
from django.db import models
User = get_user_model()

class Recipe(models.Model):

    title = models.CharField(max_length=255)
    description = models.TextField(max_length=255)
    cuisine = models.CharField(max_length=255, choices=CUISINE_CHOICES,
default='Choose a cuisine')
    image = models.ImageField(upload_to='uploads/recipe_pictures/', blank=True)
    prep_time = models.CharField(max_length=255)
    servings = models.CharField(max_length=255)
    owner_name = models.CharField(max_length=255)
    owner = models.ForeignKey(User, on_delete=models.CASCADE, null=False)
```

```
class Diet(models.Model):
    recipe = models.ForeignKey(Recipe, related_name='diet',
on_delete=models.CASCADE)
    diet = models.CharField(choices=DIET_CHOICES, max_length=255,
default='Choose a diet')
```

The Recipe model includes fields for title, description, cuisine, image, prep_time, servings, owner_name, and owner. cuisine is a choice field with options for various cuisines. image is an image field that stores an image of the recipe. owner is a foreign key to the user model and specifies the user who created the recipe.

The Diet model includes fields for recipe and diet. recipe is a foreign key to the Recipe model, and diet is a choice field with options for various diets.

```
class Instruction(models.Model):
    recipe = models.ForeignKey(Recipe, related_name='instruction',
on_delete=models.CASCADE)
    instruction = models.CharField(max_length=255)

class Ingredient(models.Model):
    recipe = models.ForeignKey(Recipe, on_delete=models.CASCADE,
related_name='ingredient')
    ingredient = models.CharField(max_length=255)

class Comment(models.Model):
    recipe = models.ForeignKey(Recipe, on_delete=models.CASCADE,
related_name='comment')
    owner = models.ForeignKey(to=User, on_delete=models.CASCADE)
    time = models.DateTimeField(auto_now_add=True, auto_created=True)
```

```

comment = models.CharField(max_length=255)

class Like(models.Model):
    recipe = models.ForeignKey(Recipe, on_delete=models.CASCADE,
related_name='like')
    owner = models.ForeignKey(to=User, on_delete=models.CASCADE)

```

The Instruction model includes fields for recipe and instruction. recipe is a foreign key to the Recipe model, and instruction is a character field that stores instructions for making the recipe.

The Ingredient model includes fields for recipe and ingredient. recipe is a foreign key to the Recipe model, and ingredient is a character field that stores ingredients for making the recipe.

The Comment model includes fields for recipe, owner, time, and comment. recipe is a foreign key to the Recipe model, owner is a foreign key to the user model, time is a date time field that stores the time of the comment, and comment is a character field that stores the comment text.

The Like model includes fields for recipe and owner. recipe is a foreign key to the Recipe model, and owner is a foreign key to the user model. This model allows users to like a recipe.

```

class Favourite(models.Model):
    recipe = models.ForeignKey(Recipe, on_delete=models.CASCADE,
related_name='favourite')
    owner = models.ForeignKey(to=User, on_delete=models.CASCADE)

class ShoppingList(models.Model):
    owner = models.ForeignKey(to=User, on_delete=models.CASCADE)
    recipe = models.ForeignKey(Recipe, on_delete=models.CASCADE,
related_name='shopping_list')

class Rate(models.Model):
    recipe = models.ForeignKey(Recipe, on_delete=models.CASCADE,
related_name='rating')
    owner = models.ForeignKey(to=User, on_delete=models.CASCADE)
    rating = models.IntegerField()

```

The Favourite model includes fields for recipe and owner. recipe is a foreign key to the Recipe model, and owner is a foreign key to the user model. This model allows users to add a recipe to their favorites.

The ShoppingList model includes fields for owner and recipe. owner is a foreign key to the user model, and recipe is a foreign key to the Recipe model. This model allows users to add a recipe to their shopping list.

The Rate model includes fields for recipe, owner, and rating. recipe is a foreign key to the Recipe model, owner is a foreign key to the user model, and rating is an integer field that stores the user's rating for the recipe.