Splitwise - Command Line Interface

Command Line Interface

A command line interface is a way of interacting with a computer program where the user issues commands to the program in the form of successive lines of text (command lines). The program responds with output to the command lines issued by the user.

CLI is a very common way of interacting with programs. For example, the `git` program is a CLI program. It has multiple commands like `git add`, `git commit`, `git push` etc. Each of these commands takes in some arguments and performs some action.

A CLI is different from an API in that an API is a way for programs to interact with each other, whereas a CLI is a way for humans to interact with programs. Also, API calls are usually made over the network, whereas CLI commands are usually made on the same machine.

Creating a CLI with Django

You can create a CLI with Django by using the `django-admin` command. This command is used to create a new Django project. You can create a new Django project by running the following command:

```
$ django-admin startproject splitwise
```

We'll start with creating a command to register a new user. We'll call this command `register`.

To create a new command, you can use the `BaseCommand` class from `django.core.management.base`. This class is the base class for all commands in Django. You can create a new command by creating a new class that inherits from `BaseCommand` and then override the `handle` method to implement the logic for your command.

```
# splitwise/management/commands/register.py
from django.core.management.base import BaseCommand

class Command(BaseCommand):
    help = 'Register a new user'

    def handle(self, *args, **options):
        self.stdout.write(self.style.SUCCESS('Register!'))
```

To register this command with Django, you need to create a `management/commands` directory inside your app and then create a file with the name of your command inside this directory. In this case, we created a `register.py` file inside the `management/commands` directory.

Now, you can run this command using the `manage.py` command:

```
$ python manage.py register
Register!
```

Passing Arguments to a Command

You can also pass arguments to your command. For example, you can pass the username, phone number and password of the user you want to register. You can do this by overriding the `add_arguments` method of the `BaseCommand` class.

```
# splitwise/management/commands/register.py

from django.core.management.base import BaseCommand

class Command(BaseCommand):
    help = 'Register a new user'

    def add_arguments(self, parser):
        parser.add_argument('username', type=str)
        parser.add_argument('password', type=str)
        parser.add_argument('phone', type=str)

    def handle(self, *args, **options):
        username = options['username']
        password = options['password']
        phone = options['phone']
        self.stdout.write(self.style.SUCCESS(f'Register {username}!'))
```

Now, you can run this command by passing the username and password as arguments:

```
$ python manage.py register Tantia Tope
Register Tantia!
```

Creating a User

Now that we have the username and password of the user, we can create a new user in the database. Let's create the user model first.

```
# splitwise/models.py

from django.db import models

class User(models.Model):
    username = models.CharField(max_length=100)
    phone = models.CharField(max_length=100)
    password = models.CharField(max_length=100)
```

Now you can use the `User` model to create a new user in the database.

```
# splitwise/management/commands/register.py

from django.core.management.base import BaseCommand
from splitwise.models import User

class Command(BaseCommand):
    help = 'Register a new user'

    def add_arguments(self, parser):
        parser.add_argument('username', type=str)
        parser.add_argument('password', type=str)
        parser.add_argument('phone', type=str)

    def handle(self, *args, **options):
        username = options['username']
        password = options['password']
        phone = options['phone']
        user = User.objects.create(username=username, password=password, phone=phone)
        self.stdout.write(self.style.SUCCESS(f'Register {username}!'))
```

You can also encode the password using the `make_password` function from `django.contrib.auth.hashers`. This function will hash the password and return the hashed password.

```
# splitwise/management/commands/register.py

from django.core.management.base import BaseCommand
from django.contrib.auth.hashers import make_password

from splitwise.models import User

class Command(BaseCommand):
    help = 'Register a new user'

    def add_arguments(self, parser):
        parser.add_argument('username', type=str)
        parser.add_argument('password', type=str)
        parser.add_argument('phone', type=str)

    def handle(self, *args, **options):
        username = options['username']
        password = options['password']
        phone = options['phone']
        user = User.objects.create(username=username, password=make_password(password), ph
        self.stdout.write(self.style.SUCCESS(f'Register {username}!'))
```

Creating a Group

Now, we will create a command to create a new group. First, we'll create a `Group` model.

```
# splitwise/models.py
from django.db import models

class Group(models.Model):
    name = models.CharField(max_length=100)
    created_by = models.ForeignKey(User, on_delete=models.DO_NOTHING)
    created_at = models.DateTimeField(auto_now_add=True)
    members = models.ManyToManyField(User, related_name='groups')
    admins = models.ManyToManyField(User, related_name='admin_groups')
```

Now, we'll create a command to create a new group.

```
# splitwise/management/commands/add_group.py

from django.core.management.base import BaseCommand

class Command(BaseCommand):
    help = 'Create a new group'

    def add_arguments(self, parser):
        parser.add_argument('name', type=str)
        parser.add_argument('created_by', type=str)
        parser.add_argument('members', type=str, nargs='+')
        parser.add_argument('admins', type=str, nargs='+')

    def handle(self, *args, **options):
        name = options['name']
        created_by = options['created_by']
        members = options['members']
        admins = options['admins']
        Group.objects.create(name=name, created_by=created_by, members=members, admins=adm
        self.stdout.write(self.style.SUCCESS(f'Create group {name}!'))
```

Here, we have used the `nargs` argument to specify that the `members` and `admins` arguments can take multiple values. This is because a group can have multiple members and admins. To specify multiple values, you can pass the values separated by spaces.

```
$ python manage.py add_group Roommates u1 u1 u2 u3 u1 u2
Create group Roommates!
```

Adding a User to a Group

Now, we'll create a command to add a user to a group. We'll call this command `add_user_to_group`.

```python
# splitwise/management/commands/add_user_to_group.py

from django.core.management.base import BaseCommand

class Command(BaseCommand):
    help = 'Add a user to a group'

    def add_arguments(self, parser):
        parser.add_argument('user', type=int)
        parser.add_argument('group', type=int)

    def handle(self, *args, **options):

        username = options['user']
        group = options['group']
        user = User.objects.get(id=user)

        group = Group.objects.get(id=group)
        group.members.add(user)
        group.save()

        self.stdout.write(self.style.SUCCESS(f'Add {username} to {group}!'))
```

Conclusion

In this document, we learned how to create a command line interface with Django. We created commands to register a new user, create a new group and add a user to a group. In the next chapter, we'll see how to implement the settle up command and the debt simplification algorithm.