بسمه تعالی

گزارش فاز اول پروژه‌ي نهایی درس امنیت داده و شبکه

نام و نام خانوادگی: امیرحسین کوچاری
شماره دانشجویی: 95170651
نام و نام خانوادگی: حامد عبدی
شماره دانشجویی: 96109782

ترم تحصیلی ۲ - ۱۴۰۰

نحوه‌ی راه اندازی پروژه

برای راه اندازی پروژه، در مرحله‌ی اول نیاز به نصب کتابخانه‌ها داریم. با استفاده از دستور pip install -r requirements، اقدام به نصب کتابخانه‌های موردنیاز میکنیم. سپس در پوشه‌ی سرور، دستور python manage.py runserver را اجرا میکنیم و پس از آن در پوشه‌ی کلاینت، فایل main.py را ران کرده و پروژه شروع به کار می‌کند.

شرح منطق پیاده سازی

در این قسمت، دو قسمت کلاینت و سرور را تعریف میکنیم:

کلاینت

در بخش مربوط به کلاینت، از فایل main شروع میکنیم.

main.py

client > main.py > ...

```python
from commands import path, defined_commands

command = None
while command != 'exit':
    inputs = input(f"{path} > ").split()
    command = inputs[0]
    if command in defined_commands:
        defined_commands[command](*inputs[1:])

```

در main.py، تا زمانی که دستور exit وارد نشده باشد، رابط کاربری اقدام به دریافت کامند از کاربر میکند و اگر کامند وارد شده بین مواردی بود که در لیست کامندها وجود داشت، آن را به سمت سرور ارسال کرده و خروجی آن را دریافت میکند.

اولین کامند، مربوط به لاگین کاربر است. پس از لاگین کردن کاربر، یه توکن در هدر کاربر قرار میگیرد که از آن پس در هر درخواست، از این توکن برای احراز هویت کاربر استفاده خواهد شد.

در صورتی که کاربر به سیستم لاگین نکرده باشد، به او یادآوری میشود که نیاز به لاگین کردن دارد.

```
def login():
    user_name = input("enter your user name:")
    password = input("enter your password:")
    data = {
        'user_name': user_name,
        'password': password,
    }
    response = post(f"{constants.SERVER_URL}/login", data=data)
    if response.status_code == 201:
        print("login succeeded")
        global auth_key, root_token
        json_response = response.json()
        auth_key = json_response.get('authorization')
        root_token = json_response.get('root_token')
    else:
        print("login failed")
```

```
def login_required(func):
    def new_func(*args, **kwargs):
        if not auth_key:
            print('pls login if you have account or signup if you don\'t')
            return
        return func(*args, **kwargs)

    return new_func
```

همچنین برای ثبت نام کاربر نیز تابع sign up وجود دارد که اطلاعات کاربر را گرفته و در دیتابیس (به صورت هش شده) ذخیره میکند تا در هنگام لاگین کردن،اطلاعات ورودی را با آن تطبیق دهد.

```
def signup():
    first_name = input("enter your first name:")
    last_name = input("enter your last name:")
    user_name = input("enter your user name:")
    password = input("enter your password:")

    data = json.dumps(dict(list=list(), type='directory'))
    token = post_file(data, auth_key)
    data = {
        'first_name': first_name,
        'last_name': last_name,
        'user_name': user_name,
        'password': password,
        'root_token': token
    }
    response = post(f"{constants.SERVER_URL}/signup", data=data)
    if response.status_code == 201:
        print("signup succeeded")
    else:
        print("signup failed")
```

بقیه‌ی دستورات نیز همگی در فایل commands.py قرار دارند و هر یک کار خواسته شده را انجام میدهند. تمامی مواردی که برای انجام به لاگین بودن کاربر نیاز دارند، با login required مشخص شده‌اند.

دستور rm:

```
@login_required
def rm(input_path, *options):
    absolute_path = input_path if input_path.startswith('~') else relative_to_absolute(input_path)
    if absolute_path == '~':
        print('not allowed')
        return
    try:
        token, data = find_file(absolute_path)
    except Exception:
        print(f"No such file or directory: {input_path}")
        return
    file_type = get_file_type(data)
    if file_type == 'directory':
        if not '-r' in options:
            print(f'cannot remove {input_path}: Is a directory')
            return
        dir = DirectoryManager(absolute_path)
        for item, _ in dir.list:
            rm(f"{absolute_path}/item", '-r')
    delete_file(token, auth_key)
    father_path = absolute_path[:absolute_path.rfind('/')]
    dir_manager = DirectoryManager(father_path)
    dir_manager.remove(token)
    dir_manager.put()
```

دستور mv:

```
@login_required
def mv(input_path, output_path):
    absolute_ipath = input_path if input_path.startswith('~') else relative_to_absolute(input_path)
    absolute_opath = input_path if input_path.startswith('~') else relative_to_absolute(output_path)
    token, data = find_file(absolute_ipath)
    out = find_file(absolute_opath)
    if out and get_file_type(out[1]) != 'directory':
        print('file exists')
        return
    father_ipath = absolute_ipath[:absolute_ipath.rfind('/')]
    father_opath = absolute_opath if out else absolute_opath[:absolute_opath.rfind('/')]
    file_name = absolute_ipath[absolute_ipath.rfind('/') + 1:] if out else absolute_opath[
                                                                           absolute_opath.rfind('/') + 1:]
    odir_manager = DirectoryManager(father_opath)
    odir_manager.add(file_name, token)
    odir_manager.put()
    idir_manager = DirectoryManager(father_ipath)
    idir_manager.remove(token)
    idir_manager.put()
```

دستور ls:

```
@login_required
def ls(input_path):
    absolute_path = input_path if input_path.startswith('~') else relative_to_absolute(input_path)
    try:
        _, data = find_file(absolute_path)
    except Exception:
        print(f"no such file or directory: {input_path}")
    try:
        dir = DirectoryManager(absolute_path)
        for item, _ in dir.list:
            print(item)
    except TypeError:
        pass
```

دستور mkdir:

```
@login_required
def mkdir(input_path):
    absolute_path = input_path if input_path.startswith('~') else relative_to_absolute(input_path)
    father_path = absolute_path[:absolute_path.rfind('/')]
    dir_manager = DirectoryManager(father_path)
    data = json.dumps(dict(list=list(), type='directory'))
    name = absolute_path[absolute_path.rfind('/') + 1:]
    token = post_file(data, auth_key)
    dir_manager.add(name, token)
    dir_manager.put()
```

دستور touch:

```
@login_required
def touch(input_path):
    absolute_path = input_path if input_path.startswith('~') else relative_to_absolute(input_path)
    father_path = absolute_path[:absolute_path.rfind('/')]
    dir_manager = DirectoryManager(father_path)
    data = json.dumps(dict(data='', type='file'))
    name = absolute_path[absolute_path.rfind('/') + 1:]
    token = post_file(data, auth_key)
    dir_manager.add(name, token)
    dir_manager.put()
```

دستور cd:

```
@login_required
def cd(input_path):
    absolute_path = input_path if input_path.startswith('~') else relative_to_absolute(input_path)
    try:
        _, data = find_file(absolute_path)
    except Exception:
        print(f"no such file or directory: {input_path}")
        return
    try:
        DirectoryManager(absolute_path)
    except TypeError:
        print(f'not a directory: {input_path}')
        return
    global path
    path = absolute_path
```

توابع find file و relative to absolute و کلاس directory manager نیز مرتبط با یافتن فایل ها و تبدیل آدرس های محلی به سراسری می باشند.

در فایل client requests نیز تعدادی تابع مرتبط با فایل‌ها و همچنین توابع رمزگذاری و رمزگشایی وجود دارد. لازم به ذکر است که در طی عملیات مربوط به رمزگذاری، خود هر فایل نیز به صورت هش شده در کنار هر فایل قرار گرفته و ذخیره می‌شود. در نتیجه امکان جا زدن فایل‌های دیگر در دیتابیس به جای فایل اصلی وجود نخواهد داشت.

در نهایت نیز در فایل crypto.py توابع مربوط به رمزگذاری و رمزگشایی قرار دارند که برای این کار از الگوریتم HS256 برای رمزگذاری استفاده کردیم.

سرور

در سمت سرور نیز به ترتیب از فایل models در پوشه‌ی app شروع میکنیم. در این فایل دو کلاس یوزر و فایل وجود دارد. کلاس یوزر شامل اطلاعات وارد شده توسط کاربر به اضافه‌ی یک توکن برای احراز هویت است. کلاس فایل نیز شامل دیتای خود فایل و یک توکن برای هر فایل است که برای رمزگذاری آن فایل استفاده می‌شود (برای رسیدن به این هدف که هر کلید باید فقط برای رمزگذاری یک فایل استفاده شود).

```python
class User(models.Model):
    AUTH_TOKEN_EXPIRATION_TIME = 7 * 24 * 60 * 60  # one week
    first_name = models.CharField(max_length=32)
    last_name = models.CharField(max_length=32)
    user_name = models.CharField(max_length=32, unique=True)
    password = models.CharField(max_length=128)
    root_token = models.CharField(max_length=32, default=get_random_string)

    def get_authorization(self):
        payload = dict(
            id=self.pk,
            user_name=self.user_name,
            expires=int(time.time() + User.AUTH_TOKEN_EXPIRATION_TIME)
        )
        return jwt_encode(payload)


class File(models.Model):
    token = models.CharField(max_length=32, default=get_random_string, unique=True)
    data = models.TextField()
```

فایل views در سرور نیز وظیفه‌ی انجام عملیات‌های مورد نیاز روی دیتابیس را برعهده دارد. دستورات مربوط به file view همگی نیاز به احراز هویت دارند که با تگ need authorization مشخص شده‌اند.

```python
class FileView(APIView):
    @need_authorization
    def get(self, request, token):
        try:
            file = File.objects.get(token=token)
        except File.DoesNotExist:
            return Response(status=404)
        return Response(dict(data=file.data))

    def post(self, request):
        data = request.POST['data']
        file = File.objects.create(data=data)
        return Response(dict(token=file.token))

    @need_authorization
    def put(self, request, token):
        try:
            file = File.objects.get(token=token)
        except File.DoesNotExists:
            return Response(status=404)
        data = request.POST['data']
        file.data = data
        file.save()
        return Response(dict(data=file.data))

    @need_authorization
    def delete(self, request, token):
        try:
            File.objects.get(token=token).delete()
        except File.DoesNotExist:
            return Response(status=404)
        return Response()
```

ریکوئست‌های مربوط به ساین آپ و لاگین نیز در این فایل بررسی می‌شوند.

```
def login(request):
    user_name = request.POST.get('user_name')
    password = request.POST.get('password')
    try:
        user = User.objects.get(user_name=user_name)
    except User.DoesNotExist:
        return Response(status=404)
    if user.password != get_hashed_pass(password):
        return Response(status=401)
    return Response(dict(authorization=user.get_authorization(), root_token=user.root_token))


def signup(request):
    first_name = request.POST.get('first_name')
    last_name = request.POST.get('last_name')
    user_name = request.POST.get('user_name')
    password = request.POST.get('password')
    root_token = request.POST.get('root_token')
    User.objects.create(first_name=first_name, last_name=last_name, user_name=user_name,
                        password=get_hashed_pass(password), root_token=root_token)
    return Response(status=201)
```

در نهایت نیز در فایل utils، توابع مربوط به هش کردن پسورد کاربر، احراز هویت از روی توکن و دریافت اطلاعات کاربر از روی دیتابیس وجود دارد.

```
def get_user(authorization):
    payload = jwt_decode(authorization)
    if payload['expires'] > time.time():
        return None
    return User.objects.get(user_name=payload['user_name'])
```

```
def need_authorization(func):
    def new_func(request, *args, **kwargs):
        authorization = request.headers.get('AUTHORIZATION')
        if not authorization:
            return Response(status=401)
        user = get_user(authorization)
        if not user:
            return Response(status=401)
        request.user = user
        return func(request, *args, **kwargs)

    return new_func
```

```
def get_hashed_pass(raw_password):
    combo_password = raw_password + settings.SECRET_KEY
    hashed_password = bcrypt.hashpw(combo_password.encode(), salt)
    return hashed_password
```