بسمه تعالی

گزارش فاز دوم 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. در ابتدا با استفاده از دستور pip install -r requirements، اقدام به نصب کتابخانه‌های موردنیاز میکنیم. سپس در پوشه‌ی سرور،ابتدا دستور python manage.py migrate و سپس 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
import os
import sys

from commands import defined_commands, cache

hashseed = os.getenv('PYTHONHASHSEED')
if not hashseed:
    os.environ['PYTHONHASHSEED'] = '0'
    os.execv(sys.executable, [sys.executable] + sys.argv)

command = None
while command != 'exit':
    cache.clear()
    from commands import path
    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. همچنین با دستور log out، کاربری که لاگین کرده است از سیستم خارج می‌شود و سامانه مجددا برای لاگین کردن یک کاربر آماده می‌شود.

```python
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 == 200:
        print("login succeeded")
        global auth_key, root_token, root_write_key, my_user_name
        json_response = response.json()
        auth_key = json_response.get('authorization')
        root_token = json_response.get('root_token')
        root_write_key = json_response.get('root_write_key')
        read_private_key(user_name)
        my_user_name = user_name
    else:
        print("login failed")


def logout():
    global auth_key, root_token, root_write_key, my_user_name
    auth_key = None
    root_token = None
    root_write_key = None
    my_user_name = None
    print("logout succeeded")
```

```python
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.

```python
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: ")
    public_key = generate_rsa_keys(user_name)
    token, write_key = post_file()
    data = {
        'first_name': first_name,
        'last_name': last_name,
        'user_name': user_name,
        'password': password,
        'root_token': token,
        'public_key': public_key
    }
    response = post(f"{constants.SERVER_URL}/signup", data=data)
    global auth_key, root_token, root_write_key, my_user_name
    json_response = response.json()
    auth_key = json_response.get('authorization')
    root_token = json_response.get('root_token')
    root_write_key = write_key
    data = json.dumps(dict(list=list(), type='directory'))
    put_file(token, data, write_key, auth_key)
    my_user_name = user_name
    if response.status_code == 200:
        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:

```python
@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
```

دستور vim: (یک فیل را باز می‌کند و کاربر می‌تواند محتوای آن را ویرایش کند)

```
@login_required
def vim(input_path):
    absolute_path = input_path if input_path.startswith('~') else relative_to_absolute(input_path)
    try:
        token, data, write_key = find_file(absolute_path)
    except Exception:
        print(f"No such file: {input_path}")
        return
    if not write_key:
        print(f" you don't have permission to write on this file: {input_path}")
        return
    file_type = get_file_type(data)
    if file_type == 'directory':
        print(f'cannot edit {input_path}: Is a directory')
        return
    name = absolute_path[absolute_path.rfind('/') + 1:]
    json_data = json.loads(data)
    data = json_data['data']
    with open(name, 'w') as file:
        file.write(data)
    os.system(f'vim {name}')
    with open(name, 'r') as file:
        data = ''.join(file.readlines())
    os.remove(name)
    json_data['data'] = data
    key = find_key(absolute_path)
    put_file(token, json.dumps(json_data), write_key, auth_key, key)
```

دستور cat: (یک فایل را فقط برای خواندن باز می‌کند)

```
@login_required
def cat(input_path):
    absolute_path = input_path if input_path.startswith('~') else relative_to_absolute(input_path)
    try:
        token, data, write_key = find_file(absolute_path)
    except Exception:
        print(f"No such file: {input_path}")
        return
    file_type = get_file_type(data)
    if file_type == 'directory':
        print(f'cannot cat {input_path}: Is a directory')
        return
    data = json.loads(data)
    print(data['data'])
```

دستور share:

```
@login_required
def share(input_path, username, options='-r'):
    absolute_path = input_path if input_path.startswith('~') else relative_to_absolute(input_path)
    token, data, write_key = find_file(absolute_path)
    name = absolute_path[absolute_path.rfind('/') + 1:]
    data = {'token': token, 'key': jwt_sign(token), 'from': my_user_name, 'file_name': name}
    if 'w' in options:
        data['write_key'] = write_key
    create_share(token, data, username, auth_key)
```

دستور fetch: (با هر بار وارد کردن این دستور، فایل‌هایی که در پوشه آن کاربر وجود دارند -فایل‌هایی که خودش ساخته و فایل‌هایی که با اون به اشتراک گذاشته شده- به روز رسانی می‌شوند)

```
def _fetch_one_share(data):
    cache.clear()
    file_name = data['file_name']
    token = data['token']
    write_key = data.get('write_key')
    key = data['key']
    user_name = data['from']
    dir_path = f'~/shares/{user_name}'
    user_dir = find_file(dir_path)
    if not user_dir:
        mkdir(dir_path)
    cache.clear()
    dir_manager = DirectoryManager(dir_path)
    dir_manager.add(file_name, token, write_key, key)
    dir_manager.put()


@login_required
def fetch_shares():
    share_path = '~/shares'
    share_dir = find_file(share_path)
    if share_dir:
        dir = DirectoryManager(share_path)
        for _, token, write_key, _ in dir.list:
            delete_file(token, write_key, auth_key)
        dir.list.clear()
        dir.put()
    else:
        mkdir(share_path)
    cache.clear()
    shares = get_my_shares(auth_key)
    for token, data in shares:
        _fetch_one_share(data)
```

دستور revoke: (اشتراک گذاری میان یک فایل که با یک کاربر دیگر انجام شده را باطل می‌کند)

```
@login_required
def revoke(input_path):
    absolute_path = input_path if input_path.startswith('~') else relative_to_absolute(input_path)
    token, data, write_key = find_file(absolute_path)
    name = absolute_path[absolute_path.rfind('/') + 1:]
    new_token, new_write_key = post_file()
    put_file(new_token, data, new_write_key, auth_key)
    father_path = absolute_path[:absolute_path.rfind('/')]
    dir_manager = DirectoryManager(father_path)
    dir_manager.remove(name, token)
    dir_manager.add(name, new_token, new_write_key)
    dir_manager.put()
    delete_file(token, write_key, auth_key)
```

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. همچنین دستور help، تمامی دستورات موجود در سیستم را به کاربر نمایش می‌دهد.

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. همچنین برخی توابع مربوط به به اشتراک گذاری فایل‌ها نیز در این قسمت قرار دارند.

در نهایت نیز در فایل crypto.py توابع مربوط به رمزگذاری و رمزگشایی متقارن قرار دارند که برای این کار از الگوریتم HS256 برای رمزگذاری استفاده کردیم و در توابع مربوط به رمزگذاری نامتقارن نیز از الگوریتم rsa استفاده شده است. همچنین تابع jwt sign نیز با گرفتن توکن هر فایل و هش کردن ترکیب کلید خصوصی و توکن فایل، برای هر فایل یک کلید مجزا تولید می‌کند.

سرور

در سمت سرور نیز به ترتیب از فایل models در پوشه‌ی app شروع میکنیم. در این فایل سه کلاس یوزر و فایل و share 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). کلاس share نیز مربوط به اشتراک گذاری فایل‌ها است که فایل و یوزر مربوطه را به یکدیگر متصل می‌سازد. به کلاس فایل نیز یک write_key اضافه شده است که برای دادن دسترسی نوشتن به کاربران استفاده می‌شود.

```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()
```

```python
class Share(models.Model):
    file = models.ForeignKey(File, on_delete=models.CASCADE)
    to_user = models.ForeignKey(User, on_delete=models.CASCADE)
    data = models.TextField()
```

همچنین در قسمت دیتا، توکن آن فایل، کلید رمز و یوزری که این فایل را به اشتراک گذاشته قرار می‌گیرد. اگر دسترسی نوشتن (w) نیز وجود داشت، اول از تابع write key استفاده می‌کند و سپس با کلید عمومی کاربری که فایل را با او به اشتراک گذاشته است، رمز می‌کند.

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()
```

توابع مربوط به مشاهده‌ی فایل‌های شیر شده از روی دیتابیس نیز در این قسمت واقع شده‌اند.

```python
class ShareView(APIView):
    @need_authorization
    def get(self, request):
        shares = Share.objects.filter(to_user=request.user)
        result = [(share.file_id, share.data) for share in shares]
        return Response(data=result)

    @need_authorization
    def post(self, request):
        to_username = request.POST['username']
        token = request.POST['token']
        data = request.POST['data']
        try:
            to_user = User.objects.get(user_name=to_username)
        except User.DoesNotExist:
            return Response(status=404)
        Share.objects.create(to_user=to_user, file_id=token, data=data)
        return Response(status=201)
```

همچنین تابع get user public نیز در این قسمت است. این تابع وظیفه‌ی دریافت کلیدهای عمومی کاربران مختلف (که بر روی دیتابیس ذخیره شده است) را بر عهده دارد.

```python
def get_user_public(request, user_name):
    try:
        user = User.objects.get(user_name=user_name)
    except User.DoesNotExist:
        return HttpResponse(status=404)
    return JsonResponse(dict(user_name=user.user_name, public_key=user.public_key))
```

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

```python
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.

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

```python
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
```

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