Dựa trên các file đã đọc, tôi sẽ viết tài liệu chi tiết về app users:

**Tài liệu App Users**

**1. Tổng quan**

App Users là một ứng dụng Django quản lý người dùng với các tính năng mở rộng từ Django's built-in User model. App cung cấp API endpoints để quản lý thông tin người dùng và profile.

**2. Models**

**2.1 User Model**

 Kế thừa từ AbstractUser của Django

 Các trường bổ sung:

 email: Email address (unique, required)

 phone\_number: Số điện thoại (optional)

 avatar: Ảnh đại diện (optional)

 birth\_date: Ngày sinh (optional)

**2.2 UserProfile Model**

 Model 1-1 với User để lưu thông tin bổ sung

 Các trường:

 bio: Tiểu sử (TextField, max 500 ký tự)

 position: Chức vụ

 department: Phòng ban

 address: Địa chỉ

**3. Serializers**

**3.1 UserProfileSerializer**

 Serialize các trường của UserProfile

 Fields: bio, position, department, address

**3.2 UserSerializer**

 Serialize thông tin User

 Fields:

 Thông tin cơ bản: id, username, email, first\_name, last\_name

 Thông tin bổ sung: full\_name, phone\_number, avatar, birth\_date

 Thông tin profile: nested profile field

 Thông tin thời gian: date\_joined, last\_login (read-only)

**3.3 UserCreateSerializer**

 Dùng cho việc tạo user mới

 Fields: username, email, password, first\_name, last\_name, phone\_number

 Password được xử lý an toàn với write\_only=True

**4. Views/API Endpoints**

**4.1 UserViewSet**

Base URL: /api/users/

**Các endpoints chính:**

 **List Users**

 GET /api/users/

 Yêu cầu xác thực

 Trả về danh sách users

 **Create User**

 POST /api/users/

 Không yêu cầu xác thực

 Body: thông tin user theo UserCreateSerializer

 **Get User Detail**

 GET /api/users/{id}/

 Yêu cầu xác thực

 Trả về thông tin chi tiết của user

 **Update User**

 PUT/PATCH /api/users/{id}/

 Yêu cầu xác thực

 Body: thông tin cần update

**Custom endpoints:**

 **Get Current User**

 GET /api/users/me/

 Yêu cầu xác thực

 Trả về thông tin user đang đăng nhập

 **Update Profile**

 PUT/PATCH /api/users/update\_profile/

 Yêu cầu xác thực

 Body: có thể update cả thông tin user và profile

{

"first\_name": "...",

"last\_name": "...",

"profile": {

"bio": "...",

"position": "...",

"department": "...",

"address": "..."

}

}

**5. Permissions**

 Tạo user mới: Cho phép tất cả người dùng (AllowAny)

 Các thao tác khác: Yêu cầu xác thực (IsAuthenticated)

**6. Tính năng đặc biệt**

 **Tự động tạo profile**:

 UserProfile được tự động tạo khi User được tạo (thông qua signals)

 Quan hệ 1-1 giữa User và UserProfile

 **Full name handling**:

 Cung cấp full\_name trong API responses

 Tự động tạo từ first\_name và last\_name

 **Flexible Updates**:

 Hỗ trợ cập nhật từng phần (PATCH)

 Có thể cập nhật user và profile cùng lúc

 **Security**:

 Password được xử lý an toàn (write-only)

 Email là unique và required

 Các endpoints được bảo vệ bởi authentication

**7. Sử dụng**

 **Tạo user mới**:

POST /api/users/

{

"username": "example",

"email": "example@email.com",

"password": "secure\_password",

"first\_name": "First",

"last\_name": "Last",

"phone\_number": "1234567890"

}

**Cập nhật profile**:

PATCH /api/users/update\_profile/

{

"first\_name": "New First Name",

"profile": {

"bio": "New bio",

"position": "Developer"

}

}

**Lấy thông tin user hiện tại**:

GET /api/users/me/

Tôi sẽ phân tích chi tiết từng file và ý nghĩa của code trong app users:

apps/users/

├── migrations/ # Chứa các file migrations database

├── \_\_pycache\_\_/ # Cache của Python

├── \_\_init\_\_.py # File đánh dấu package Python

├── admin.py # Cấu hình Django admin

├── apps.py # Cấu hình app

├── forms.py # Định nghĩa các form

├── models.py # Định nghĩa models

├── serializers.py # Serializers cho REST API

├── signals.py # Signals handlers

├── tests.py # Unit tests

├── urls.py # URL routing

└── views.py # Views/API endpoints

**2. Chi tiết từng file**

**2.1. models.py**

class User(AbstractUser):

email = models.EmailField(\_('email address'), unique=True)

phone\_number = models.CharField(\_('phone number'), max\_length=15, blank=True)

avatar = models.ImageField(upload\_to='avatars/', null=True, blank=True)

birth\_date = models.DateField(\_('birth date'), null=True, blank=True)

REQUIRED\_FIELDS = ['email']

**Phân tích:**

 Kế thừa từ AbstractUser để tận dụng các fields có sẵn như username, password

 email: Bắt buộc và unique, dùng để xác thực

 phone\_number: Số điện thoại, không bắt buộc (blank=True)

 avatar: Lưu ảnh đại diện trong thư mục 'avatars/'

 birth\_date: Ngày sinh, cho phép null và blank

 REQUIRED\_FIELDS: Thêm email vào danh sách fields bắt buộc khi tạo superuser

class UserProfile(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE, related\_name='profile')

bio = models.TextField(\_('biography'), max\_length=500, blank=True)

position = models.CharField(\_('position'), max\_length=100, blank=True)

department = models.CharField(\_('department'), max\_length=100, blank=True)

address = models.TextField(\_('address'), blank=True)

**Phân tích:**

 OneToOneField: Mối quan hệ 1-1 với User

 related\_name='profile': Cho phép truy cập profile từ user: user.profile

 on\_delete=models.CASCADE: Xóa profile khi xóa user

 Các fields khác đều optional (blank=True)

**2.2. serializers.py**

class UserProfileSerializer(serializers.ModelSerializer):

class Meta:

model = UserProfile

fields = ('bio', 'position', 'department', 'address')

**Phân tích:**

 Serializer đơn giản cho UserProfile

 Chỉ serialize các fields cơ bản

class UserSerializer(serializers.ModelSerializer):

profile = UserProfileSerializer(read\_only=True)

full\_name = serializers.SerializerMethodField()

class Meta:

model = User

fields = ('id', 'username', 'email', 'first\_name', 'last\_name',

'full\_name', 'phone\_number', 'avatar', 'birth\_date',

'profile', 'date\_joined', 'last\_login')

read\_only\_fields = ('date\_joined', 'last\_login')

**Phân tích:**

 Nested serializer với profile

 Custom field full\_name được tính toán động

 read\_only\_fields: Các fields không cho phép update

 Bao gồm đầy đủ thông tin user và profile

class UserCreateSerializer(serializers.ModelSerializer):

password = serializers.CharField(write\_only=True)

class Meta:

model = User

fields = ('username', 'email', 'password', 'first\_name',

'last\_name', 'phone\_number')

**Phân tích:**

 Serializer riêng cho việc tạo user

 password là write\_only để bảo mật

 Chỉ bao gồm các fields cần thiết khi tạo user

**2.3. views.py**

class UserViewSet(viewsets.ModelViewSet):

queryset = User.objects.all()

permission\_classes = [permissions.IsAuthenticated]

def get\_serializer\_class(self):

if self.action == 'create':

return UserCreateSerializer

return UserSerializer

**Phân tích:**

 Sử dụng ModelViewSet để tự động tạo CRUD endpoints

 Dynamic serializer dựa vào action

 Mặc định yêu cầu authentication

@action(detail=False, methods=['get'])

def me(self, request):

"""Endpoint để lấy thông tin user hiện tại"""

serializer = self.get\_serializer(request.user)

return Response(serializer.data)

**Phân tích:**

 Custom action không cần ID (detail=False)

 Trả về thông tin của user đang đăng nhập

 Sử dụng serializer mặc định của viewset

@action(detail=False, methods=['put', 'patch'])

def update\_profile(self, request):

user = request.user

profile = user.profile

# Cập nhật thông tin user

user\_serializer = UserSerializer(

user,

data=request.data,

partial=True

)

user\_serializer.is\_valid(raise\_exception=True)

user\_serializer.save()

# Cập nhật thông tin profile

profile\_data = request.data.get('profile', {})

profile\_serializer = UserProfileSerializer(

profile,

data=profile\_data,

partial=True

)

profile\_serializer.is\_valid(raise\_exception=True)

profile\_serializer.save()

**Phân tích:**

 Hỗ trợ cả PUT (update toàn bộ) và PATCH (update một phần)

 Xử lý update cả user và profile trong một request

 partial=True: Cho phép update từng phần

 Validate dữ liệu trước khi save

@receiver(post\_save, sender=User)

def create\_user\_profile(sender, instance, created, \*\*kwargs):

"""Tạo UserProfile khi User được tạo"""

if created:

UserProfile.objects.create(user=instance)

**Phân tích:**

 Sử dụng Django signals để tự động tạo profile

 Kích hoạt sau khi user được tạo (post\_save)

 Chỉ tạo profile khi user mới được tạo (created=True)

**2.5. urls.py**

from rest\_framework.routers import DefaultRouter

from .views import UserViewSet

router = DefaultRouter()

router.register('users', UserViewSet)

urlpatterns = router.urls

**Phân tích:**

 Sử dụng DRF Router để tự động tạo URLs

 Tạo các endpoints chuẩn RESTful:

 GET /users/ - List users

 POST /users/ - Create user

 GET /users/{id}/ - Retrieve user

 PUT /users/{id}/ - Update user

 DELETE /users/{id}/ - Delete user

 GET /users/me/ - Current user

 PUT/PATCH /users/update\_profile/ - Update profile

**3. Flow hoạt động**

**3.1. Tạo User mới:**

 Request đến POST /users/

 UserCreateSerializer validate data

 User được tạo trong database

 Signal trigger tạo UserProfile

 Response trả về thông tin user

**3.2. Update Profile:**

 Request đến PATCH /users/update\_profile/

 Authenticate user

 Update thông tin user

 Update thông tin profile

 Response trả về data đã update

**3.3. Get User Info:**

 Request đến GET /users/me/

 Authenticate user

 UserSerializer serialize user và profile

 Response trả về full user data

**4. Security Features**

 Password hashing tự động

 Authentication required cho hầu hết endpoints

 Email verification (unique)

 Safe password handling (write\_only)

 Permission-based access control

CREATE TABLE users\_user (

id INT PRIMARY KEY,

username VARCHAR(150) UNIQUE,

email VARCHAR(254) UNIQUE,

password VARCHAR(128),

first\_name VARCHAR(150),

last\_name VARCHAR(150),

phone\_number VARCHAR(15),

avatar VARCHAR(100),

birth\_date DATE,

date\_joined TIMESTAMP,

last\_login TIMESTAMP

);

CREATE TABLE users\_userprofile (

id INT PRIMARY KEY,

user\_id INT UNIQUE REFERENCES users\_user(id),

bio TEXT,

position VARCHAR(100),

department VARCHAR(100),

address TEXT

);