МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ "НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО"

ЛАБОРАТОРНАЯ РАБОТА №3

РЕАЛИЗАЦИЯ СЕРВЕРНОЙ ЧАСТИ НА DJANGO REST. ДОКУМЕНТИРОВАНИЕ API

РАБОТУ ВЫПОЛНИЛ:
Костылев Иван K33401

РАБОТУ ПРОВЕРИЛ:
Говоров Антон Игоревич

Санкт-Петербург
2022 г.

Ход работы

1. **Практическая работа №3.1.**

1.1. Реализовать эндпоинты для добавления и просмотра навыков методом, описанном в работе

Skills Create

```
POST /war/skills/create/
```

```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "Success": "Skill 'Fast running' created successfully."
}
```

Skills

```
GET /war/skills/
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "skills": [
        {
            "id": 1,
            "title": "Fast running"
        }
    ]
}
```

1.2. Реализовать эндпоинты для вывода полной информации о всех войнах и их профессиях (в одном запросе);
Для этого необходимо добавить новый сериализатор, который будет сериализовать поле profession отдельно, с помощью его собственного сериализатора.

Warrior Profession

Вывод полной информации о всех войнах и их профессиях (в одном запросе).

```
GET /war/warriors/professions/
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "warriors": [
        {
            "id": 1,
            "profession": {
                "id": 1,
                "title": "New profession",
                "description": "Very important profession"
            },
            "race": "t",
            "name": "Ванюшка",
            "level": 110,
            "skill": [
                1,
                2,
                3,
                6,
```

В нашем случае достаточно отнаследоваться от уже существующего сериализатора

```
class WarriorProfessionSerializer(WarriorSerializer):
    profession = ProfessionSerializer(read_only=True)
```

1.3. Реализовать эндпоинты для вывода полной информации о всех войнах и их скиллах (в одном запросе);
Выполняется аналогично предыдущему (только нужно добавить параметр many=True).

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "warriors": [
        {
            "id": 1,
            "skill": [
                {
                    "id": 1,
                    "title": "Fast running"
                },
                {
                    "id": 2,
                    "title": "Pass Web"
                },
                {
                    "id": 3,
                    "title": "Work as Android Developer"
                },
                {
                    "id": 6,
                    "title": "Time management"
                },
                {
                    "id": 4,
                    "title": "Pull up"
```

```
class WarriorSkillsSerializer(WarriorSerializer):
    skill = SkillSerializer(many=True, read_only=True)
```

1.4. Реализовать эндпоинты для вывода информации о воине (по id), его профессиях и скиллах;
Для получения данных об одном воине используем в качестве базового класса RetrieveAPIView

Вывод полной информации о войне (по id), его профессиях и скилах.

```
GET /war/warriors/2/
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 2,
    "profession": {
        "id": 6,
        "title": "Безработный",
        "description": "01 февраля с 00:00 до 23:59"
    },
    "skill": [
        {
            "id": 3,
            "title": "Work as Android Developer"
        },
        {
            "id": 5,
            "title": "Fight"
        },
        {
            "id": 4,
            "title": "Pull up"
```

```python
class WarriorInfoView(RetrieveAPIView):
    """
    Вывод полной информации о войне (по id), его профессиях и скилах.
    """
    queryset = Warrior.objects.all()
    serializer_class = WarriorProfessionsSkillsSerializer
```

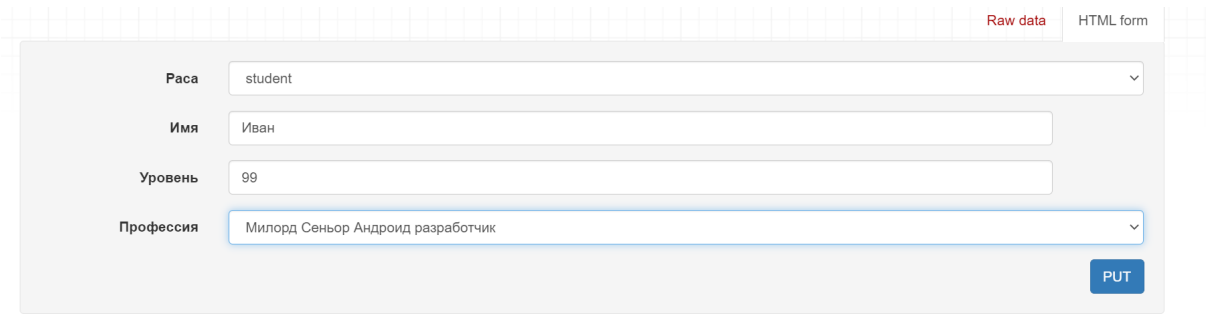
1.5. Удаление воина по id;

Унаследуем наш контрол от DestroyAPIView. В url будет передаваться id воина

```python
class WarriorDeleteView(DestroyAPIView):
    """
    Удаление война по id.
    """
    queryset = Warrior.objects.all()
    serializer_class = WarriorSerializer
```

1.6. Редактирование информации о воине

Реализуем этот функционал с помощью UpdateAPIView



Редактирование информации о воине

```
PUT /war/warriors/2/edit/
```

```
HTTP 200 OK
Allow: PUT, PATCH, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "message": "warrior Иван updated successfully"
}
```

```python
class WarriorEditView(UpdateAPIView):
    """
    Редактирование информации о воине
    """
    queryset = Warrior.objects.all()
    serializer_class = WarriorSerializer
    lookup_field = 'pk'

    def update(self, request, *args, **kwargs):
        instance = self.get_object()
        serializer = self.get_serializer(instance, data=request.data, partial=True)

        if serializer.is_valid():
            serializer.save()
            return Response({"message": f"warrior {instance} updated successfully"})

        else:
            return Response({"message": "failed", "details": serializer.errors})
```

2. Реализация REST API для выбранного варианта. Нами был выбран сайт для прогноза погоды.

2.1. Модели базы данных

Наша база данных будет содержать всего 3 модели: User, Town и Country. Этого хватит для реализации наших потребностей.

```python
class Country(models.Model):
    name = CharField(max_length=200)
    continent = CharField(max_length=10, choices=CONTINENTS)

    def __str__(self):
        return f'{self.name} in {self.continent}'


class Town(models.Model):
    name = CharField(max_length=200)
    country = ForeignKey(Country, on_delete=models.CASCADE)
    lon = FloatField()
    lat = FloatField()

    def __str__(self):
        return f'{self.name} in {self.country}'
```

```python
class User(AbstractUser):
    towns = ManyToManyField(Town, blank=True)
    days_count = IntegerRangeField(default=3, min_value=1, max_value=10)
```

2.2. REST API (Views)

```python
class CreateUserAPIView(CreateAPIView):
    serializer_class = UserSerializer
    queryset = User.objects.all()


class UserAPIView(ListAPIView):
    serializer_class = UserSerializer
    queryset = User.objects.all()


class GetUserInfoAPIView(RetrieveAPIView):
    serializer_class = UserSerializer
    queryset = User.objects.all()
```

```python
class UpdateUserAPIView(RetrieveUpdateAPIView):
    serializer_class = UserSerializer
    queryset = User.objects.all()


class CreateCountryAPIView(CreateAPIView):
    serializer_class = CountrySerializer
    queryset = Country.objects.all()


class GetCountriesAPIView(RetrieveAPIView):
    def get(self, request, **kwargs):
        countries = Country.objects.all()

        serializer = CountrySerializer(countries, many=True)

        return JsonResponse({'towns': serializer.data})
```

```python
class CreateTownAPIView(CreateAPIView):
    serializer_class = TownSerializer
    queryset = Town.objects.all()


class GetTownsAPIView(RetrieveAPIView):
    def get(self, request, **kwargs):
        towns = Town.objects.all()

        serializer = TownSerializer(towns, many=True)

        return JsonResponse({'towns': serializer.data})
```

Нами изначально был реализован весь API для работы с базой данных. В следующем пункте мы добавим работу с токенами.

2.3. Подключение Djoser

```python
    path('auth/', include('djoser.urls')),
    re_path('auth/', include('djoser.urls.authtoken')),
```

На рисунке выше видно, как мы подключаем стандартные эндпоинты, предоставляемые Djoser. С помощью настроек укажем, что взаимодействовать по API могут только авторизованные пользователи. Фрагмент кода представлен ниже.

```python
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework.authentication.TokenAuthentication',
    ),
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',
    )
}
```

2.4. Документация с помощью MkDocs

Нами была составлена документация для всего взаимодействия по API нашего приложения с помощью Mkdocs. Работа с mkdocs достаточно удобна, в ней используется простой язык разметки Markdown. Подключение специальной темы material, мы преобразуем внешний вид нашего документа. На скриншоте снизу представлена структура документа в формате .yml.

```yaml
site_name: My.WEATHER
nav:
  - Home: index.md
  - API:
      - User's actions: user.md
      - Towns and contries: towns_contries.md
      - Weather: weather.md
  - About authors: about.md
theme: material
```

Выводы

В качестве результата данной работы является готовый сервер, позволяющий регистрироваться новым пользователям, авторизоваться по токенам, хранить информацию о пользователях (города / страны), добавлять новые города и страны для получения прогноза погоды.