

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

Факультет «Инфокоммуникационных технологий»
Направление подготовки «09.03.03 Прикладная информатика»
Бакалаврская программа «Мобильные и сетевые технологии»

Лабораторная работа №3

по дисциплине «Web программирование»

«Django Rest Framework, документирование API»

Выполнил

_____ / Комаров Г. Ю., К33402
(подпись) (Фамилия И.О., группа)

Проверил

_____ / Говоров А. И.

Дата _____

**Санкт-Петербург
2021**

Выполнение работы

Часть 1. Практическая работа №3.1

1. В рамках практической работы реализованы:

- Методы для работы с профессиями

professions Методы для работы с профессиями			
GET	/war/professions	Список профессий	✓ 🔒
POST	/war/professions	Создать профессию	✓ 🔒
GET	/war/professions/{id}	Получить профессию по ID	✓ 🔒
PUT	/war/professions/{id}	Редактировать профессию	✓ 🔒
PATCH	/war/professions/{id}	Редактировать профессию	✓ 🔒
DELETE	/war/professions/{id}	Удалить профессию	✓ 🔒

```
19 class Profession(models.Model):
20     """
21     Описание профессии
22     """
23
24     class Meta:
25         verbose_name = 'Профессия'
26         verbose_name_plural = 'Профессии'
27
28     title = models.CharField(max_length=120, unique=True, verbose_name='Название')
29     description = models.TextField(verbose_name='Описание')
```

```
6 class ProfessionSerializer(serializers.ModelSerializer):
7     class Meta:
8         model = Profession
9         fields = '__all__'
10
```

```
9 @extend_schema_view(
10     create=profession_create_schema,
11     retrieve=profession_retrieve_schema,
12     list=profession_list_schema,
13     update=profession_update_schema,
14     partial_update=profession_update_schema,
15     destroy=profession_delete_schema,
16 )
17 class ProfessionViewSet(ModelViewSet):
18     serializer_class = ProfessionSerializer
19     queryset = Profession.objects.all()
```

- Методы для работы с умениями

skills Методы для работы с умениями			
GET	/war/skills	Список умений	⌵ 🔒
POST	/war/skills	Создать умение	⌵ 🔒
GET	/war/skills/{id}	Получить умение по ID	⌵ 🔒
PUT	/war/skills/{id}	Редактировать умение	⌵ 🔒
PATCH	/war/skills/{id}	Редактировать умение	⌵ 🔒
DELETE	/war/skills/{id}	Удалить умение	⌵ 🔒

```
4 class Skill(models.Model):
5     """
6     Описание умений
7     """
8
9     class Meta:
10         verbose_name = 'Умение'
11         verbose_name_plural = 'Умения'
12
13         title = models.CharField(max_length=120, unique=True, verbose_name='Наименование')
14
15     def __str__(self):
16         return self.title
```

```
12 class SkillSerializer(serializers.ModelSerializer):
13     class Meta:
14         model = Skill
15         fields = '__all__'
```

```
22 @extend_schema_view(
23     create=skill_create_schema,
24     retrieve=skill_retrieve_schema,
25     list=skill_list_schema,
26     update=skill_update_schema,
27     partial_update=skill_update_schema,
28     destroy=skill_delete_schema,
29 )
30 class SkillViewSet(ModelViewSet):
31     serializer_class = SkillSerializer
32     queryset = Skill.objects.all()
```

- Методы для работы с воинами

warriors Методы для работы с воинами			^
GET	/war/warriors	Список воинов	⌵ 🔒
POST	/war/warriors	Создать воина	⌵ 🔒
GET	/war/warriors/{id}	Получить воина по ID	⌵ 🔒
PUT	/war/warriors/{id}	Редактировать воина	⌵ 🔒
PATCH	/war/warriors/{id}	Редактировать воина	⌵ 🔒
DELETE	/war/warriors/{id}	Удалить воина	⌵ 🔒

```
32 class SkillOfWarrior(models.Model):
33     """
34     Описание умений война
35     """
36
37     class Meta:
38         verbose_name = 'Умение воина'
39         verbose_name_plural = 'Умение воинов'
40
41     skill = models.ForeignKey('Skill', verbose_name='Умение', on_delete=models.CASCADE)
42     warrior = models.ForeignKey('Warrior', verbose_name='Воин', on_delete=models.CASCADE)
43     level = models.IntegerField(verbose_name='Уровень освоения умения')
44
45
46 class WarriorRaces(models.TextChoices):
47     """
48     Расы война
49     """
50
51     STUDENT = 'student', 'Студент'
52     DEVELOPER = 'developer', 'Разработчик'
53     TEAMLEAD = 'teamlead', 'Тимлид'
54
55
56 class Warrior(models.Model):
57     """
58     Описание война
59     """
60
61     class Meta:
62         verbose_name = 'Воин'
63         verbose_name_plural = 'Воины'
64
65     race = models.CharField(max_length=10, choices=WarriorRaces.choices, verbose_name='Раса')
66     name = models.CharField(max_length=120, verbose_name='Имя')
67     level = models.IntegerField(default=0, verbose_name='Уровень')
68     skills = models.ManyToManyField(Skill, through=SkillOfWarrior, related_name='warrior_skills', verbose_name='Умения')
69     profession = models.ForeignKey(Profession, on_delete=models.CASCADE, blank=True, null=True, verbose_name='Профессия')
```

```

18 class SkillOfWarriorSerializer(serializers.ModelSerializer):
19     class Meta:
20         model = SkillOfWarrior
21         fields = ['title', 'level']
22         read_only_fields = fields
23
24         title = serializers.CharField(source='skill', read_only=True)
25
26
27 class WarriorSerializer(serializers.ModelSerializer):
28     class Meta:
29         model = Warrior
30         fields = '__all__'
31
32         profession = serializers.SlugRelatedField(slug_field='title', queryset=Profession.objects.all())
33         skills = SkillOfWarriorSerializer(source='skillofwarrior_set', many=True, read_only=True)
34

```

```

35 @extend_schema_view(
36     create=warrior_create_schema,
37     retrieve=warrior_retrieve_schema,
38     list=warrior_list_schema,
39     update=warrior_update_schema,
40     partial_update=warrior_update_schema,
41     destroy=warrior_delete_schema,
42 )
43 class WarriorViewSet(ModelViewSet):
44     serializer_class = WarriorSerializer
45     queryset = Warrior.objects.all()

```

Так же была создана документация в формате OpenAPI v3 с помощью **drf-spectacular**, т. к. библиотека drf-yasg более не поддерживается и не всегда генерирует точную документацию.

Часть 2. Выполнение лабораторной работы

В качестве проекта для работы №3 и №4 была идея создать сервис для взаиморасчетов, где информация о тратах подгружается с помощью QR кода чека.

Чеки загружаются с помощью приложения ЧекСкан (такой способ был выбран ввиду закрытого API ФНС). После этого есть возможность указать в профиле токен ЧекСкан и синхронизировать чеки с их API. В чеке содержится общая сумма, список товаров, их кол-во и цена. После загрузки чеков в систему можно разбить каждый товар на несколько пользователей.

checks Методы для работы с чеками		^
GET	/api/checks	Получить список чеков
GET	/api/checks/{id}	Получить чек по ID
PUT	/api/checks/{id}/items	Установить взаиморасчет для чека
POST	/api/checks/sync	Синхронизировать чеки из ЧекСкан

Методы для работы с чеками

Check ▾ {	
id*	integer title: ID чека
items*	> [...]
createdAt*	string(\$date-time) readOnly: true title: Дата создания
updatedAt*	string(\$date-time) readOnly: true title: Дата обновления
qrData	string nullable: true title: Данные QR кода maxLength: 100
name*	string title: Наименование maxLength: 1000
address	string nullable: true title: Адрес maxLength: 1000
date*	string(\$date-time) title: Дата покупки
totalSum*	integer title: Сумма чека
fn	string nullable: true title: ФН maxLength: 20
fp	string nullable: true title: ФПД maxLength: 20
fd	string nullable: true title: ФД maxLength: 20
orgInn*	string title: ИНН организации maxLength: 20
orgName*	string title: Название организации maxLength: 20
orgAddress	string nullable: true title: Юр. адрес организации maxLength: 20
createdBy*	integer readOnly: true title: Кем создан
updatedBy*	integer readOnly: true title: Кем обновлён
users*	Пользователи > [...]
}	

Модель чека

CheckItem ▾ {	
id*	integer title: ID товара
createdAt*	string(\$date-time) readOnly: true title: Дата создания
updatedAt*	string(\$date-time) readOnly: true title: Дата обновления
name*	string title: Наименование maxLength: 1000
price*	integer title: Цена
quantity*	string(\$decimal) pattern: ^\d{0,5}(?:\.\d{0,10})?\$ title: Количество
sum*	integer title: Сумма
createdBy*	integer readOnly: true title: Кем создан
updatedBy*	integer readOnly: true title: Кем обновлён
receipt*	integer title: Чек
parts*	Пользователи > [...]
}	

Модель товара

CheckItemPartRequest ▾ {	
user*	string
part*	string(\$decimal) pattern: ^\d{0,5}(?:\.\d{0,10})?\$ title: Количество
item*	integer title: Товар
}	

Ассоциативная сущность: товар, пользователь, количество

users Методы для работы с аккаунтом текущего пользователя		⌵
GET	/api/users/profile Получить информацию о текущем пользователе	⌵ 🔒
PUT	/api/users/profile Редактировать информацию о текущем пользователе	⌵ 🔒
PATCH	/api/users/profile Редактировать информацию о текущем пользователе	⌵ 🔒
DELETE	/api/users/profile Деактивировать аккаунт	⌵ 🔒
PUT	/api/users/profile/avatar Обновить аватар текущего пользователя	⌵ 🔒

Методы для работы с пользователем

```

Profile {
  id* integer
  readOnly: true
  username* string
  title: Имя пользователя
  readOnly: true
  email* string($email)
  title: E-Mail
  readOnly: true
  firstName string
  title: Имя
  maxLength: 150
  lastName string
  title: Фамилия
  maxLength: 150
  avatarUrl* string($uri)
  readOnly: true
  checkScanToken string
  nullable: true
  title: Токен ЧекСкан
}

```

Модель пользователя

auth Методы библиотеки авторизации			^
POST	/api/auth/login/	Авторизация пользователя	✓ 🔒
POST	/api/auth/logout/	Выход	✓ 🔒
POST	/api/auth/password/change/	Смена пароля	✓ 🔒
POST	/api/auth/password/reset/	Запрос на сброс пароля	✓ 🔒
POST	/api/auth/password/reset/confirm/	Подтверждение сброса пароля	✓ 🔒
POST	/api/auth/registration/	Регистрация пользователя	✓ 🔒
POST	/api/auth/registration/resend-email/	Повторная отправка e-mail	✓ 🔒
POST	/api/auth/registration/verify-email/	Подтверждение e-mail	✓ 🔒
POST	/api/auth/token/refresh/	Обновить JWT токен	✓
POST	/api/auth/token/verify/	Валидировать JWT токен	✓

Методы библиотеки авторизации (dj-rest-auth)

Часть 3. Документация с помощью MkDocs

В результате выполненной работы была изучена система документации MkDocs. Ввиду того, что имелась полная API схема, было принято решение воспользоваться пакетом [openapi-generator](#) для формирования файлов в формате Markdown.

CheckSplitter Docs

Документация OpenAPI

Авторизация

Чеки

Пользователь

Documentation for CheckSplitter API

Documentation for API Endpoints

Class	Method	HTTP request	Description
AuthApi	authLoginCreate	POST /api/auth/login/	Авторизация пользователя
AuthApi	authLogoutCreate	POST /api/auth/logout/	Выход
AuthApi	authPasswordChangeCreate	POST /api/auth/password/change/	Смена пароля
AuthApi	authPasswordResetConfirmCreate	POST /api/auth/password/reset/confirm/	Подтверждение сброса пароля
AuthApi	authPasswordResetCreate	POST /api/auth/password/reset/	Запрос на сброс пароля
AuthApi	authRegistrationCreate	POST /api/auth/registration/	Регистрация пользователя
AuthApi	authRegistrationResendEmailCreate	POST /api/auth/registration/resend-email/	Повторная отправка e-mail
AuthApi	authRegistrationVerifyEmailCreate	POST /api/auth/registration/verify-	Подтверждение e-

Table of contents

Documentation for API Endpoints

Documentation for Models

Documentation for Authorization

jwtAuth

Документация MkDocs

CheckSplitter Docs

Документация OpenAPI

Авторизация

Чеки

Пользователь

checksItemsUpdate

checksItemsUpdate(id, CheckItemPartRequest)

Установить взаиморасчет для чека

Parameters

Name	Type	Description	Notes
id	Integer	A unique value identifying this Check.	[default to null]
CheckItemPartRequest	List		

Return type

null (empty response body)

Authorization

jwtAuth

HTTP request headers

- Content-Type: application/json
- Accept: Not defined

Документация MkDocs

Вывод

В результате выполненной работы были получены практические навыки по использованию DRF и MkDocs.