Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»
Факультет инфокоммуникационных технологий

Лабораторная работа №3
«РЕАЛИЗАЦИЯ СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ СРЕДСТВАМИ DJANGO И DJANGORESTFRAMEWORK»

по дисциплине

«Web-программирование»

Выполнил:

студент III курса ФИКТ

группы К33402

Ф.И.О. Кондрашов Егор Юрьевич

Проверил:

Говоров А. И.

Санкт-Петербург

2021

Цель работы: овладеть практическими навыками и умениями реализации web-сервисов средствами Django.

Выполнение работы:

Recycle Starter - сервис для сбора макулатуры и её дальнейшей переработки, разработанный в рамках конкурса ITMO Future.

Доступен по адресу: https://recycle.itmo.ru/

Исходный код бэкенда: https://github.com/e-kondr01/rcs_back

Ниже приведена часть кода из приложения containers_app.

Модели базы данных и их методы (models.py):

```
import datetime
import time
from secrets import choice
from string import ascii_letters, digits
from tempfile import NamedTemporaryFile
from typing import List, Union

import pdfkit
from django.conf import settings
from django.contrib.auth import get_user_model
from django.core.mail import EmailMessage
from django.db import models
from django.db.models import Sum
from django.db.models.functions import Coalesce
from django.db.models.query import QuerySet
from django.template.loader import render_to_string
from django.utils import timezone

from rcs_back.containers_app.utils.qr import generate_sticker
from rcs_back.utils.model import get_eco_emails

tz = timezone.get_default_timezone()


class EmailToken(models.Model):
    """Модель токена для:
    активации контейнера через email;
    создания сбора всех контейнеров через email"""
```

```python
TOKEN_LENGTH = 32

token = models.CharField(
    max_length=32,
    blank=True,
    verbose_name="токен"
)

is_used = models.BooleanField(
    default=False,
    verbose_name="использован"
)

def generate_token(self) -> str:
    """Генерирует рандомный токен"""
    token = ''.join(choice(
        ascii_letters + digits
    ) for _ in range(self.TOKEN_LENGTH))
    return token

def set_token(self) -> None:
    """Задать значение поля token"""
    while True:
        token = self.generate_token()
        # Проверка на уникальность
        if not EmailToken.objects.filter(
            token=token
        ).first():
            break
    self.token = token

def use(self) -> None:
    """Использует токен"""
    if not self.is_used:
        self.is_used = True
        self.save()

def __str__(self) -> str:
    return f"токен №{self.pk}"

class Meta:
    verbose_name = "токен для email"
```

```
        verbose_name_plural = "токены для email"


class BaseBuilding(models.Model):
    """Абстрактный класс для общих методов
    здания и корпуса"""

    def current_mass(self) -> int:
        """Возвращает накопившуюся массу бумаги
        по зданию/корпусу"""
        current_mass = 0
        container: Container
        for container in self.containers.filter(
            _is_full=True
        ):
            current_mass += container.mass()
        return current_mass

    def meets_mass_takeout_condition(self) -> bool:
          """Выполняются ли в здании/корпусе условия для сбора по общей
массе"""
        return bool(self.takeout_condition.mass and
                    self.current_mass() >= self.takeout_condition.mass)

    def meets_time_takeout_condition(self) -> bool:
        """Выполняются ли в здании/корпусе условия для сбора
        по времени."""
        container: Container
        for container in self.containers.all():
            if container.check_time_conditions():
                return True
        return False

    def containers_for_takeout(self) -> QuerySet:
        """Возвращает QuerySet полных контейнеров"""
        return self.containers.filter(
            _is_full=True
        ).filter(
            status=Container.ACTIVE
        )

    def container_count(self) -> int:
        """Кол-во активных контейнеров"""
        return self.containers.filter(status=Container.ACTIVE).count()

    class Meta:
        abstract = True


class Building(BaseBuilding):
    """ Модель здания """

    address = models.CharField(
        max_length=2048,
        verbose_name="адрес"
    )

    get_container_room = models.CharField(
        max_length=64,
        verbose_name="аудитория, в которой можно получить контейнер",
        blank=True
    )

    get_sticker_room = models.CharField(
        max_length=64,
        verbose_name="аудитория, в которой можно получить стикер",
        blank=True
    )

    sticker_giver = models.CharField(
        max_length=256,
        verbose_name="контакты человека, который выдаёт стикер",
        blank=True
    )

    _takeout_notified = models.BooleanField(
        default=False,
        verbose_name="послано оповещение о необходимоси сбора"
    )

    precollected_mass = models.PositiveSmallIntegerField(
        default=0,
        verbose_name="масса, собранная до старта сервиса"
    )

    passage_scheme = models.ImageField(
```

```python
        null=True,
        blank=True,
        verbose_name="схема проезда"
    )

    detect_building_part = models.BooleanField(
        default=False,
        verbose_name="определять номер корпуса по аудитории"
    )

    def street_name(self) -> str:
        """Возвращает только улицу"""
        if "," in self.address:
            return self.address[:self.address.find(",")]
        else:
            return self.address

    def needs_takeout(self) -> bool:
        """Нужно ли вынести бумагу?"""
        if hasattr(self, "building_parts"):
            for bpart in self.building_parts.all():
                if bpart.needs_takeout():
                    return True
        return (self.meets_mass_takeout_condition() or
                self.meets_time_takeout_condition())

    def check_conditions_to_notify(self) -> None:
        """Проверяет условия на вынос и если нужно,
        отправляет email-Оповещение о необходимости сбора"""
        if not self._takeout_notified and self.needs_takeout():

            self._takeout_notified = True
            self.save()

            self.takeout_condition_met_notify()

    def takeout_condition_met_notify(self) -> None:
        """Оповещение о необходимости сбора"""
        emails = self.get_worker_emails()
        if emails:

            due_date = timezone.now().date() + datetime.timedelta(days=1)
```

```
           token = EmailToken.objects.create()
           token.set_token()
           token.save()
           link = "https://" + settings.DOMAIN
                                                                link      +=
f"/api/container-takeout-requests?token={token.token}"
           link += f"&building={self.pk}"

           msg = render_to_string("takeout_condition_met.html", {
               "address": self.address,
               "due_date": due_date,
               "containers": self.containers_for_takeout(),
               "link": link,
               "has_building_parts": hasattr(self, "building_parts"),
           }
           )

           email = EmailMessage(
               "Оповещение от сервиса RecycleStarter",
               msg,
               None,
               emails
           )
           email.content_subtype = "html"
           containers_html_s = render_to_string(
               "containers_for_takeout.html", {
                   "containers": self.containers_for_takeout(),
                                    "has_building_parts":  hasattr(self,
"building_parts"),
               }
           )
           pdf = pdfkit.from_string(containers_html_s, False)
           email.attach("containers.pdf",
                        pdf,
                        "application/pdf"
                        )
           email.send()

   def tank_takeout_notify(self) -> None:
       """Отправляет запрос на вывоз накопительного бака"""
       emails = []
       tank_takeout_companies = TankTakeoutCompany.objects.all()
       if tank_takeout_companies:
           company: TankTakeoutCompany
           for company in tank_takeout_companies:
               emails.append(company.email)
           phone = ""
           name = ""
           hoz_worker = self.get_hoz_workers().first()
           if hoz_worker:
               phone = hoz_worker.phone
               name = hoz_worker.name

           msg = render_to_string("tank_takeout.html", {
               "address": self.address,
               "phone": phone,
               "name": name
           }
           )

           email = EmailMessage(
               "Оповещение от сервиса RecycleStarter",
               msg,
               None,
               emails
           )
           email.content_subtype = "html"
           if self.passage_scheme:
               email.attach("passage.png",
                            self.passage_scheme.read(),
                            "image/png"
                            )
           email.send()

   def get_hoz_workers(self) -> QuerySet["User"]:
       """QuerySet из сотрудников хоз отдела"""
       hoz_workers = get_user_model().objects.filter(
           groups__name=settings.HOZ_GROUP
       ).filter(
           building=self
       )
       return hoz_workers

   def get_worker_emails(self) -> List[str]:
       """Возвращает email всех сотрудников эко отдела
       и email коменданта здания"""
       emails = get_eco_emails()
       hoz_worker = self.get_hoz_workers().first()
       if hoz_worker:
           emails.append(hoz_worker.email)
       return emails

   def calculated_collected_mass(self) -> int:
       """Собранная масса макулатуры, посчитанная как среднее"""
       mass = self.precollected_mass if self.precollected_mass else 0
       for request in self.containers_takeout_requests.filter(
           confirmed_at__isnull=False
       ):
           mass += request.mass()
       return mass

   def confirmed_collected_mass(self,
                                start_date: datetime.date = None,
                                end_date: datetime.date = None,
                                yearly: bool = False
                                ) -> int:
       """Суммарная масса собранной макулатуры,
       подтверждённая после вывоза бака.
       При указании start_date, возвращает массу макулатуры,
       собранную за месяц после start_date.
       При указании yearly=True, возвращает массу
       макулатуры, собранную за год после start_date"""
       confirmed_requests = self.tank_takeout_requests.filter(
           confirmed_mass__isnull=False
       )
       if start_date:
           if yearly:
               end_date = start_date.replace(
                   day=1
               ).replace(
                   year=start_date.year+1
               )
           else:
               if not end_date:
                                            end_date = (start_date +
datetime.timedelta(days=31)).replace(day=1)

           confirmed_requests = confirmed_requests.filter(
               confirmed_at__gte=start_date,
               confirmed_at__lt=end_date
           )
       mass = confirmed_requests.aggregate(
           summ_mass=Coalesce(Sum("confirmed_mass"), 0)
       )["summ_mass"]
       if self.precollected_mass and not start_date:
           mass += self.precollected_mass
       return mass

   def activated_containers(self, start_date: datetime.date) -> int:
       """Возвращает кол-во контейнеров, активированных
       в течение месяца с заданной даты"""
                                   end_date     =     (start_date     +
datetime.timedelta(days=31)).replace(day=1)
       return self.containers.filter(
           activated_at__gte=start_date,
           activated_at__lt=end_date
       ).count()

   def avg_fill_speed(self) -> Union[float, None]:
       """Средняя скорость сбора макулатуры (кг/месяц)"""
       if self.tank_takeout_requests.exists():
           if self.tank_takeout_requests.order_by(
                   "created_at")[0].confirmed_mass:
                                                         start_date    =
self.tank_takeout_requests.order_by("created_at")[
                   0].confirmed_at
                month_count = (timezone.now().year - start_date.year) *
\
                   12 + (timezone.now().month - start_date.month)
               if not month_count:
                   month_count = 1
               return self.confirmed_collected_mass() / month_count

       return None

   def __str__(self) -> str:
       return self.address

   class Meta:
       verbose_name = "здание"
       verbose_name_plural = "здания"
```

```
class BuildingPart(BaseBuilding):
   """Модель корпуса здания"""

   num = models.CharField(
       max_length=32,
       verbose_name="номер корпуса"
   )

   building = models.ForeignKey(
       to=Building,
       on_delete=models.CASCADE,
       related_name="building_parts",
       verbose_name="здание"
   )

   def needs_takeout(self) -> bool:
       """Нужно ли вынести бумагу?"""
       return (self.meets_mass_takeout_condition() or
               self.meets_time_takeout_condition())

   def __str__(self) -> str:
       return f"{str(self.building)}, корпус {self.num}"

   class Meta:
       verbose_name = "корпус здания"
       verbose_name_plural = "корпусы зданий"


class         Container(models.Model):                #         pylint:
disable=too-many-public-methods
   """ Модель контейнера """

   # Варианты статуса
   WAITING = 1
   ACTIVE = 2
   INACTIVE = 3
   RESERVED = 4
   STATUS_CHOICES = (
       (WAITING, "ожидает подключения"),
       (ACTIVE, "активный"),
       (INACTIVE, "не активный"),
       (RESERVED, "распечатан стикер, контейнер не выбран")
```

```
)

# Варианты вида
ECOBOX = 1
PUBLIC_ECOBOX = 2
OFFICE_BOX = 3
KIND_CHOICES = (
    (ECOBOX, "экобокс"),
    (PUBLIC_ECOBOX, "контейнер в общественном месте"),
    (OFFICE_BOX, "офисная урна")
)

# Масса бумаги, вмещающейся в вид контейнера, в кг
ECOBOX_MASS = 30
PUBLIC_ECOBOX_MASS = 15
OFFICE_BOX_MASS = 4

kind = models.PositiveSmallIntegerField(
    choices=KIND_CHOICES,
    verbose_name="вид контейнера"
)

building = models.ForeignKey(
    to=Building,
    on_delete=models.PROTECT,
    related_name="containers",
    verbose_name="здание"
)

building_part = models.ForeignKey(
    to=BuildingPart,
    on_delete=models.CASCADE,
    related_name="containers",
    blank=True,
    null=True,
    verbose_name="корпус"
)

floor = models.PositiveSmallIntegerField(
    verbose_name="этаж"
)

room = models.CharField(
    max_length=16,
    blank=True,
    verbose_name="аудитория"
)

description = models.CharField(
    max_length=1024,
    verbose_name="описание",
    blank=True
)

status = models.PositiveSmallIntegerField(
    choices=STATUS_CHOICES,
    default=ACTIVE,
    verbose_name="состояние"
)

activated_at = models.DateTimeField(
    null=True,
    blank=True,
    verbose_name="время активации"
)

email = models.EmailField(
    verbose_name="почта (для связи)",
    blank=True
)

phone = models.CharField(
    max_length=24,
    verbose_name="номер телефона (для связи)",
    blank=True
)

_is_full = models.BooleanField(
    default=False,
    verbose_name="полный (для сортировки)"
)

avg_takeout_wait_time = models.DurationField(
    blank=True,
    null=True,
    verbose_name="среднее время ожидания выноса контейнера"
)

avg_fill_time = models.DurationField(
    blank=True,
    null=True,
    verbose_name="среднее время заполнения контейнера"
)

requested_activation = models.BooleanField(
    default=False,
    verbose_name="запрошена активация"
)

def mass(self) -> int:
    """Возвращает массу контейнера по его виду"""
    mass_dict = {
        self.ECOBOX: self.ECOBOX_MASS,
        self.PUBLIC_ECOBOX: self.PUBLIC_ECOBOX_MASS,
        self.OFFICE_BOX: self.OFFICE_BOX_MASS
    }

    if self.kind in mass_dict:
        return mass_dict[self.kind]
    else:
        return 0

def collected_mass(self,
                   start_date: datetime.date = None,
                   end_date: datetime.date = None) -> int:
            """"Рассчитанная суммарная масса, собранная из этого
контейнера"""
    reports = self.full_reports.filter(
        emptied_at__isnull=False
    )
    if start_date and end_date:
        reports.filter(
            emptied_at__gte=start_date,
            emptied_at__lte=end_date
        )
    takeout_count = reports.count()
    return takeout_count * self.mass()

def is_active(self) -> bool:
```

```python
        """Активен ли контейнер?"""
        return self.status == self.ACTIVE

    def is_public(self) -> bool:
        """Находится в общественном месте?"""
        return self.kind == self.PUBLIC_ECOBOX

    def last_full_report(self) -> Union["FullContainerReport", None]:
        """Возвращает самый новый и
        незакрытый FullContainerReport
        для этого контейнера"""
        report = self.full_reports.order_by(
            "-reported_full_at"
        ).first()
        if report and not report.emptied_at:
            return report
        else:
            return None

    def last_emptied_report(self) -> Union["FullContainerReport", None]:
        """Возвращает последний закрытый FullContainerReport
        для этого контейнера"""
        reports = self.full_reports.order_by(
            "-reported_full_at"
        )
        if reports:
            if reports[0].emptied_at:
                return reports[0]
            if len(reports) > 1 and reports[1].emptied_at:
                return reports[1]
            else:
                return None
        else:
            return None

    def empty_from(self) -> Union[datetime.datetime, None]:
        """Возвращает, с какого момента контейнер является пустым"""
        if not self.is_full():
            if self.last_emptied_report():
                return self.last_emptied_report().emptied_at
            else:
                return self.activated_at
        else:
            return None

    def ignore_reports_count(self) -> int:
        """Возвращает количество сообщений о заполненности,
             которое  нужно  игнорировать,  если  контейнер  в  общественом
месте"""
        if not self.is_public():
            return 0
        if (self.building_part and
                self.building_part.takeout_condition.ignore_reports):
            return self.building_part.takeout_condition.ignore_reports
        else:
            return self.building.takeout_condition.ignore_reports

    def is_full(self) -> bool:
        """Полный ли контейнер?
        Учитывается количество сообщений, которые надо игнорировать."""
        if self.is_active() and self.last_full_report():

            if not self.is_public():
                return True

            if self.last_full_report().by_staff:
                return True

            ignore_count = self.ignore_reports_count()
            return self.last_full_report().count > ignore_count

        else:
            return False

    def add_report(self, by_staff: bool = False):
        """Фиксируем сообщение о заполненности и
        проверяем полноту контейнера"""
        report: FullContainerReport = self.last_full_report()

        if report:
            # При повторном сообщении о заполнении нужно
            # увеличить кол-во сообщений
            if by_staff:
                report.by_staff = True
            report.count += 1
            report.save()
        else:
            # При первом сообщение о заполненности контейнера
            # нужно создать FullContainerReport
            FullContainerReport.objects.create(
                container=self,
                by_staff=by_staff
            )

        time.sleep(5)  # Ждём сохранения в БД
        # Если выполняются условия для вывоза по
        # кол-ву бумаги, нужно сообщить
        self.check_fullness()

    def handle_empty(self):
        """При опустошении контейнера нужно запомнить время
        и пересчитать среднее время выноса"""
        last_full_report = self.last_full_report()
        if last_full_report:
            last_full_report.emptied_at = timezone.now()
            last_full_report.save()
            time.sleep(5)  # Ждём сохранения в БД
                                         self.avg_takeout_wait_time    =
self.calc_avg_takeout_wait_time()
        self._is_full = False  # Для сортировки
        self.save()

    def check_fullness(self) -> None:
        """Проверяет, полный ли контейнер. Если полный,
        то сохраняет для сортировки и проверяет,
        не выполнилось ли условие по массе"""
        if self.is_full() and not self._is_full:
            self._is_full = True  # Для сортировки
            report: FullContainerReport = self.last_full_report()
            report.filled_at = timezone.now()
            report.save()
            self.save()
            time.sleep(5)  # Ждём сохранения в БД
            self.avg_fill_time = self.calc_avg_fill_time()
            self.save()
            self.building.check_conditions_to_notify()

    def get_time_condition_days(self) -> Union[int, None]:
```

```
    """Возвращает максимальное кол-во дней, которое
    этот контейнер может быть заполнен по условию"""
    if self.is_public():
        if (self.building_part and
                self.building_part.takeout_condition.public_days):
            return self.building_part.takeout_condition.public_days
        else:
            return self.building.takeout_condition.public_days
    else:
        if (self.building_part and
                self.building_part.takeout_condition.office_days):
            return self.building_part.takeout_condition.office_days
        else:
            return self.building.takeout_condition.office_days

def check_time_conditions(self) -> bool:
    '''Выполнены ли условия "не больше N дней"'''
    if self.is_active() and self.get_time_condition_days():

        if self.is_full():
            days_full = self.cur_takeout_wait_time().days
            return days_full >= self.get_time_condition_days()
        else:
            return False
    else:
        # Если такого условия нет, то False
        return False

def cur_fill_time(self) -> Union[datetime.timedelta, None]:
    """Текущее время заполнения контейнера.
    Если None - то уже заполнен (либо не активен)"""
    if self.is_active() and self.empty_from():
        fill_time = timezone.now() - self.empty_from()
        return fill_time
    else:
        return None

def cur_takeout_wait_time(self) -> Union[datetime.timedelta, None]:
    """Текущее время ожидания выноса контейнера"""
    if (self.is_active() and
        self.last_full_report() and
            self.last_full_report().filled_at):
        try:
                wait_time = (timezone.now() -
                             self.last_full_report().filled_at)
                return wait_time
            except AttributeError:
                return None
        else:
            return None

    def calc_avg_fill_time(self) -> Union[datetime.timedelta, None]:
        """Считает среднее время заполнения контейнера"""
        reports = self.full_reports.filter(
            filled_at__isnull=False
        ).order_by("filled_at")
        if (self.activated_at and reports) or len(reports) > 1:
            sum_time = datetime.timedelta(seconds=0)
            count = 0
            if self.activated_at:
                sum_time += reports[0].filled_at - self.activated_at
                count += 1
            for i in range(len(reports) - 1):
                if reports[i].emptied_at:
                    fill_time = reports[i+1].filled_at - \
                        reports[i].emptied_at
                    sum_time += fill_time
                    count += 1
            avg_fill_time = sum_time / count
            return avg_fill_time
        else:
            return None

    def calc_avg_takeout_wait_time(self) -> Union[datetime.timedelta,
None]:
        """Считает среднее время ожидания выноса контейнера"""
        reports = self.full_reports.filter(
            emptied_at__isnull=False
        )
        if not reports:
            return None
        else:
            sum_time = datetime.timedelta(seconds=0)
            for report in reports:
                sum_time += report.takeout_wait_time()
            avg_takeout_wait_time = sum_time / len(reports)
            return avg_takeout_wait_time

    def request_activation(self) -> None:
        """Запросить активацию контейнера"""
        if not self.is_active():
            token = EmailToken.objects.create()
            token.set_token()
            token.save()
            self.requested_activation = True
            self.save()
            self.activation_request_notify(token)

    def activation_request_notify(self, token: EmailToken) -> None:
        """Отправляет запрос на активацию экологу и коменданту здания"""
        emails = self.building.get_worker_emails()
        if emails:
            activation_link = "https://" + settings.DOMAIN +
"/api/containers/"
            activation_link += str(self.pk)
            activation_link += f"/activate?token={token.token}"
             msg = render_to_string("container_activation_request.html",
{
                "container": self,
                "activation_link": activation_link
            }
            )

            email = EmailMessage(
                "Запрос активации контейнера на сайте RecycleStarter",
                msg,
                None,
                emails
            )
            email.content_subtype = "html"
            email.send()

    def activate(self) -> None:
        """Активировать контейнер"""
        self.status = Container.ACTIVE
        self.requested_activation = False
        self.save()

    def public_add_notify(self) -> None:
        """Отправляет сообщение с инструкциями для активации
        добавленного контейнера"""
        is_ecobox = self.kind == Container.ECOBOX

        msg = render_to_string("public_container_add.html", {
            "is_ecobox": is_ecobox,
            "container_room": self.building.get_container_room,
            "sticker_room": self.building.get_sticker_room,
            "sticker_giver": self.building.sticker_giver
        }
        )

        email = EmailMessage(
            "Добавление контейнера в сервисе RecycleStarter",
            msg,
            None,
            [self.email]
        )
        email.content_subtype = "html"
        with NamedTemporaryFile() as tmp:
            sticker_im = generate_sticker(self.pk)
            sticker_im.save(tmp.name, "pdf", quality=100)
            email.attach("sticker.pdf", tmp.read(), "application/pdf")
            email.send()

    def detect_building_part(self) -> Union[BuildingPart, None]:
        """Определяет корпус по номеру аудитории"""
        if (self.room and self.building.detect_building_part
                and not self.building_part):
            char: str
            for char in self.room:
                if char.isdigit():
                    if self.building.building_parts.filter(
                        num=char
                    ).first():
                        return self.building.building_parts.filter(
                            num=char
                        ).first()
                    break
        return None

    def correct_fullness(self) -> None:
        """Этот метод используется для корректировки
        ошибок (заполненный в сервисе контейнер на самом деле пустой),
        поэтому не вызываем handle_empty_container
        (там устанавливается время опустошения)"""
        last_report: FullContainerReport = self.last_full_report()
        if last_report:
            last_report.delete()
            time.sleep(5)  # Ждём сохранения в БД
            self._is_full = False  # Для сортировки
            self.avg_fill_time = self.calc_avg_fill_time()
            self.save()

    def __str__(self) -> str:
        return f"Контейнер №{self.pk}"

    class Meta:
        verbose_name = "контейнер"
        verbose_name_plural = "контейнеры"


class FullContainerReport(models.Model):
    """Модель, хранящая информацию о том, когда
    был заполнен и очищен контейнер"""

    reported_full_at = models.DateTimeField(
        auto_now_add=True,
        verbose_name="первый раз получено"
    )

    filled_at = models.DateTimeField(
        null=True,
        blank=True,
        verbose_name="заполнен в"
    )

    container = models.ForeignKey(
        to=Container,
        on_delete=models.CASCADE,
        related_name="full_reports",
        verbose_name="контейнер"
    )

    count = models.SmallIntegerField(
        default=1,
        verbose_name="количество сообщений"
    )

    emptied_at = models.DateTimeField(
        verbose_name="контейнер вынесен",
        blank=True,
        null=True
    )

    by_staff = models.BooleanField(
        default=False,
        verbose_name="сотрудником"
    )

    def takeout_wait_time(self) -> Union[datetime.timedelta, None]:
        """Возвращает время ожидания выноса"""
        if self.emptied_at and self.filled_at:
            return self.emptied_at - self.filled_at
        else:
            return None

    def __str__(self) -> str:
        return (f"Контейнер №{self.container.pk} заполнен, "

f"{self.reported_full_at.astimezone(tz).strftime('%d.%m.%Y')}")

    class Meta:
        verbose_name = "контейнер заполнен"
        verbose_name_plural = "контейнеры заполнены"


class TankTakeoutCompany(models.Model):
    """Модель компании, ответственной за вывоз бака"""

    email = models.EmailField(
        verbose_name="email"
    )

    def __str__(self) -> str:
        return self.email

    class Meta:
        verbose_name = "компания, вывоза бака"
        verbose_name_plural = "компании, вывоз бака"
```

Сериализаторы (serializers.py):

```
from rest_framework import serializers

from .models import Building, BuildingPart, Container, FullContainerReport


class BuildingPartSerializer(serializers.ModelSerializer):
    class Meta:
        model = BuildingPart
        fields = [
            "id",
            "num"
        ]


class BuildingSerializer(serializers.ModelSerializer):
    """Сериализатор здания"""
    building_parts = BuildingPartSerializer(
        many=True, read_only=True
    )

    class Meta:
        model = Building
        fields = [
            "id",
            "address",
            "building_parts",
            "detect_building_part"
        ]


class BuildingShortSerializer(serializers.ModelSerializer):
    """Сериализатор здания только с адресом"""
    class Meta:
        model = Building
        fields = [
            "id",
            "address"
        ]


class FullContainerReportSerializer(serializers.ModelSerializer):
    """Сериализатор для заполнения контейнера"""
    container = serializers.PrimaryKeyRelatedField(
        queryset=Container.objects.filter(status=Container.ACTIVE)
    )

    class Meta:
        model = FullContainerReport
        fields = [
            "id",
            "container"
        ]


class ContainerSerializer(serializers.ModelSerializer):
    """ Сериализатор контейнера"""
    building = BuildingShortSerializer()
    building_part = BuildingPartSerializer()

    class Meta:
        model = Container
        fields = [
            "id",
            "kind",
            "mass",
            "building",
            "building_part",
            "floor",
            "room",
            "description",
            "is_full",
            "status",
            "requested_activation",
            "email",
            "phone",
            "cur_fill_time",
            "cur_takeout_wait_time",
            "avg_fill_time",
            "avg_takeout_wait_time"
```

```python
        ]


class ChangeContainerSerializer(serializers.ModelSerializer):
    """ Сериализатор контейнера с id у здания и корпуса"""

    class Meta:
        model = Container
        fields = [
            "id",
            "building",
            "building_part",
            "kind",
            "floor",
            "room",
            "description",
            "status",
            "email",
            "phone",
        ]


class ContainerPublicAddSerializer(serializers.ModelSerializer):
    """Сериализатор для добавления контейнера рандомами"""

    class Meta:
        model = Container
        fields = [
            "id",
            "email",
            "phone",
            "building",
            "building_part",
            "floor",
            "room",
            "description",
            "kind"
        ]
        extra_kwargs = {
            "email": {"required": True},
            "phone": {"required": True}
        }
    class PublicFeedbackSerializer(serializers.Serializer):  # pylint:
disable=abstract-method
        """Сериализатор для оставления обратной связи"""
        email = serializers.EmailField()
        container_id = serializers.IntegerField(required=False)
        msg = serializers.CharField(max_length=4096)
```

Представления (views.py):

```
    from tempfile import NamedTemporaryFile
    from typing import Union

    from django.conf import settings
    from django.db.models import Q
    from        django.http.response        import        HttpResponse,
HttpResponseRedirect
    from django.shortcuts import get_object_or_404
    from rest_framework import generics, permissions, status, views
    from rest_framework.response import Response

    from rcs_back.containers_app.models import Building, BuildingPart,
Container, EmailToken
    from rcs_back.utils.mixins import UpdateThenRetrieveModelMixin

    from .serializers import (
        BuildingPartSerializer,
        BuildingSerializer,
        ChangeContainerSerializer,
        ContainerPublicAddSerializer,
        ContainerSerializer,
        FullContainerReportSerializer,
        PublicFeedbackSerializer,
    )
    from .tasks import (
        container_add_report,
        container_correct_fullness,
        public_container_add_notify,
    )
    from .utils.email import send_public_feedback
    from .utils.qr import generate_sticker
```

```python
class FullContainerReportView(generics.CreateAPIView):
    """View для заполнения контейнера"""
    permission_classes = [permissions.AllowAny]
    serializer_class = FullContainerReportSerializer

    def perform_create(self, serializer) -> Union[Container, None]:
        if "container" in serializer.validated_data:
            container = serializer.validated_data["container"]
            by_staff = self.request.user.is_authenticated
            #  Фиксируем сообщение о заполненности и
            #  проверяем полноту контейнера
            container_add_report.delay(container.pk, by_staff)
            return container
        return None

    def create(self, request, *args, **kwargs):
        """Изменённый create для того чтобы возвращать кол-во
        дней, через которое опустошат контейнер"""
        serializer = self.get_serializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        container: Container = self.perform_create(serializer)
        headers = self.get_success_headers(serializer.data)
        resp = {}
        if container:
            resp[
                "time_condition_days"
            ] = container.get_time_condition_days() + 1
        return Response(resp,
                        status=status.HTTP_201_CREATED,
                        headers=headers)


class ContainerDetailView(UpdateThenRetrieveModelMixin,
                          generics.RetrieveUpdateDestroyAPIView):
    """ View для CRUD-операций с контейнерами """
    queryset = Container.objects.filter(
        ~Q(status=Container.RESERVED)
    )
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]
    retrieve_serializer = ContainerSerializer
    update_serializer = ChangeContainerSerializer
    def get_serializer_class(self):
        if self.request.method == "GET":
            return self.retrieve_serializer
        else:
            return self.update_serializer


class ContainerListView(generics.ListAPIView):
    """ View для CRUD-операций с контейнерами """

    serializer_class = ContainerSerializer

    filterset_fields = [
        "building",
        "building_part",
        "floor",
        "status"
    ]

    allowed_sorts = [
        "building",
        "building_part",
        "floor",
        "status",
        "is_full",
        "description"
    ]

    def get_queryset(self):
        """Сортировка"""
        queryset = Container.objects.filter(
            ~Q(status=Container.RESERVED)
        )
        if (self.request.user.is_authenticated and
            self.request.user.groups.filter(
                name=settings.HOZ_GROUP) and
                self.request.user.building):
            queryset = queryset.filter(
                building=self.request.user.building
            )
        if "is_full" in self.request.query_params:
                                                 is_full_param     =
self.request.query_params.get("is_full")
```

```
                is_full = not is_full_param == "false"
                queryset = Container.objects.filter(
                    _is_full=is_full
                )

            if "sort_by" in self.request.query_params:
                sort = self.request.query_params.get("sort_by")

                if sort not in self.allowed_sorts:
                    return queryset

                if sort == "is_full":
                    sort = "_is_full"  # Чтобы не путать фронт

                if "order_by" in self.request.query_params:
                                                                  order_by    =
self.request.query_params.get("order_by")
                    if order_by == "desc":
                        sort = "-" + sort

                return queryset.order_by(sort)

            return queryset


    class BuildingListView(generics.ListAPIView):
        """Списко зданий (для опций при создании контейнера)"""
        serializer_class = BuildingSerializer
        queryset = Building.objects.all()
        permission_classes = [permissions.AllowAny]


    class ContainerPublicAddView(generics.CreateAPIView):
        """Добавление своего контейнера с главной страницы"""
        queryset = Container.objects.all()
        permission_classes = [permissions.AllowAny]
        serializer_class = ContainerPublicAddSerializer

        def perform_create(self, serializer):
            building = serializer.validated_data["building"]

            #  Если в заданном здании есть распечатанные стикеры,
            #  то нужно использовать их id
            if Container.objects.filter(
                status=Container.RESERVED
            ).filter(
                building=building
            ).exists():
                container: Container = Container.objects.filter(
                    status=Container.RESERVED
                ).filter(
                    building=building
                ).first()
                container.email = serializer.validated_data["email"]
                container.phone = serializer.validated_data["phone"]
                if "building_part" in serializer.validated_data:
                                             container.building_part   =
serializer.validated_data[
                        "building_part"]
                else:
                                             container.building_part   =
container.detect_building_part()
                container.floor = serializer.validated_data["floor"]
                if "room" in serializer.validated_data:
                    container.room = serializer.validated_data["room"]
                if "description" in serializer.validated_data:
                    container.description = serializer.validated_data[
                        "description"]
                container.kind = serializer.validated_data["kind"]
                container.status = Container.WAITING
                container.save()

            else:
                container = serializer.save(status=Container.WAITING)
                                            container.building_part    =
container.detect_building_part()
                container.save()

            public_container_add_notify.delay(container.pk)


     class PublicFeedbackView(views.APIView):
        """View для обратной связи на главной странице"""
        serializer_class = PublicFeedbackSerializer
        permission_classes = [permissions.AllowAny]
```

```
    def post(self, request, *args, **kwargs):
        serializer = self.serializer_class(data=request.data)
        serializer.is_valid(raise_exception=True)
        email = serializer.validated_data["email"]
        container_id = 0
        if "container_id" in serializer.validated_data:
                                              container_id     =
serializer.validated_data["container_id"]
        msg = serializer.validated_data["msg"]
        send_public_feedback(email, msg, container_id)
        resp = {
            "status": "email sent"
        }
        return Response(resp)


class BuildingPartView(generics.ListAPIView):
    """View списка корпусов"""
    serializer_class = BuildingPartSerializer
    queryset = BuildingPart.objects.all()
    filterset_fields = ["building"]
    permission_classes = [permissions.AllowAny]


class EmptyContainerView(views.APIView):
    """View для отметки контейнера пустым
    экоотделом"""

    def post(self, request, *args, **kwargs):
        if "pk" in self.kwargs:
            container: Container = Container.objects.filter(
                pk=self.kwargs["pk"]
            ).first()
            if container:
                container_correct_fullness.delay(container.pk)

        return Response(status=status.HTTP_204_NO_CONTENT)


class ContainerStickerView(views.APIView):
    """Возвращает стикер контейнера"""
    permission_classes = [permissions.AllowAny]
    def get(self, request, *args, **kwargs):
        with NamedTemporaryFile() as tmp:
            fname = f"container-sticker-{self.kwargs['pk']}"
            sticker_im = generate_sticker(self.kwargs["pk"])
            sticker_im.save(tmp.name, "pdf", quality=100)
            file_data = tmp.read()
            response = HttpResponse(
                file_data,
                headers={
                    "Content-Type": "application/pdf",
                    "Content-Disposition":
                    f'attachment; filename={fname}'
                }
            )
            return response


class ContainerActivationRequestView(views.APIView):
    """View для запроса активации контейнером"""
    permission_classes = [permissions.AllowAny]

    def post(self, request, *args, **kwargs):
        container = get_object_or_404(
            Container, pk=self.kwargs["pk"]
        )
        if container.status != container.WAITING:
            resp = {
                       "error": "This container has already been
activated"
            }
            return Response(
                resp,
                status=status.HTTP_400_BAD_REQUEST
            )
        if container.requested_activation:
            resp = {
                    "error": "This container has already requested
activation"
            }
            return Response(
                resp,
                status=status.HTTP_400_BAD_REQUEST
            )
```

```python
        container.request_activation()

        resp = {
            "success": "email sent"
        }
        return Response(resp)


class ContainerActivationView(views.APIView):
    """View для активации контейнера через письмо"""
    permission_classes = [permissions.AllowAny]

    def get(self, request, *args, **kwargs):
        container = get_object_or_404(
            Container, pk=self.kwargs["pk"]
        )
        if container.is_active():
            title = "Повторная активация"
            text = "Контейнер уже был активирован"
            msg_status = "info"
        elif "token" in self.request.query_params:
            r_token = self.request.query_params.get("token")
            token: EmailToken = EmailToken.objects.filter(
                token=r_token
            ).first()
            if token and not token.is_used:
                container.activate()
                token.use()
                title = "Успешная активация"
                text = "Контейнер успешно активирован"
                msg_status = "success"
            elif token:
                title = "Повторная активация"
                text = "Контейнер уже был активирован"
                msg_status = "info"
            else:
                title = "Ошибка активации"
                text = "Неверный токен для активации"
                msg_status = "error"
        else:
            title = "Ошибка активации"
            text = "Неверный токен для активации"
```

```
                 msg_status = "error"
                                                                    redirect_path       =
f"/result?title={title}&text={text}&status={msg_status}"
             return HttpResponseRedirect(
                            redirect_to="https://"  +  settings.DOMAIN  +
redirect_path
             )


      class ContainerCountView(views.APIView):
         """Количество контейнеров по зданиям"""
         permission_classes = [permissions.AllowAny]

         def get(self, request, *args, **kwargs):
             resp = []
             building: Building
             for building in Building.objects.all():
                 building_dict = {}
                 building_dict["id"] = building.pk
                 building_dict["building"] = building.street_name()
                 building_dict["count"] = building.container_count()
                 # В тоннах до десятых
                                                              building_dict["mass"]    =
building.confirmed_collected_mass(
                 ) // 100 / 10
                 resp.append(building_dict)
             return Response(resp)
```

Маршрутизация (urls.py):

```
      from django.urls import path

      from .views import (
         ContainerActivationRequestView,
         ContainerActivationView,
         ContainerDetailView,
         ContainerListView,
         ContainerPublicAddView,
         ContainerStickerView,
         EmptyContainerView,
      )
```

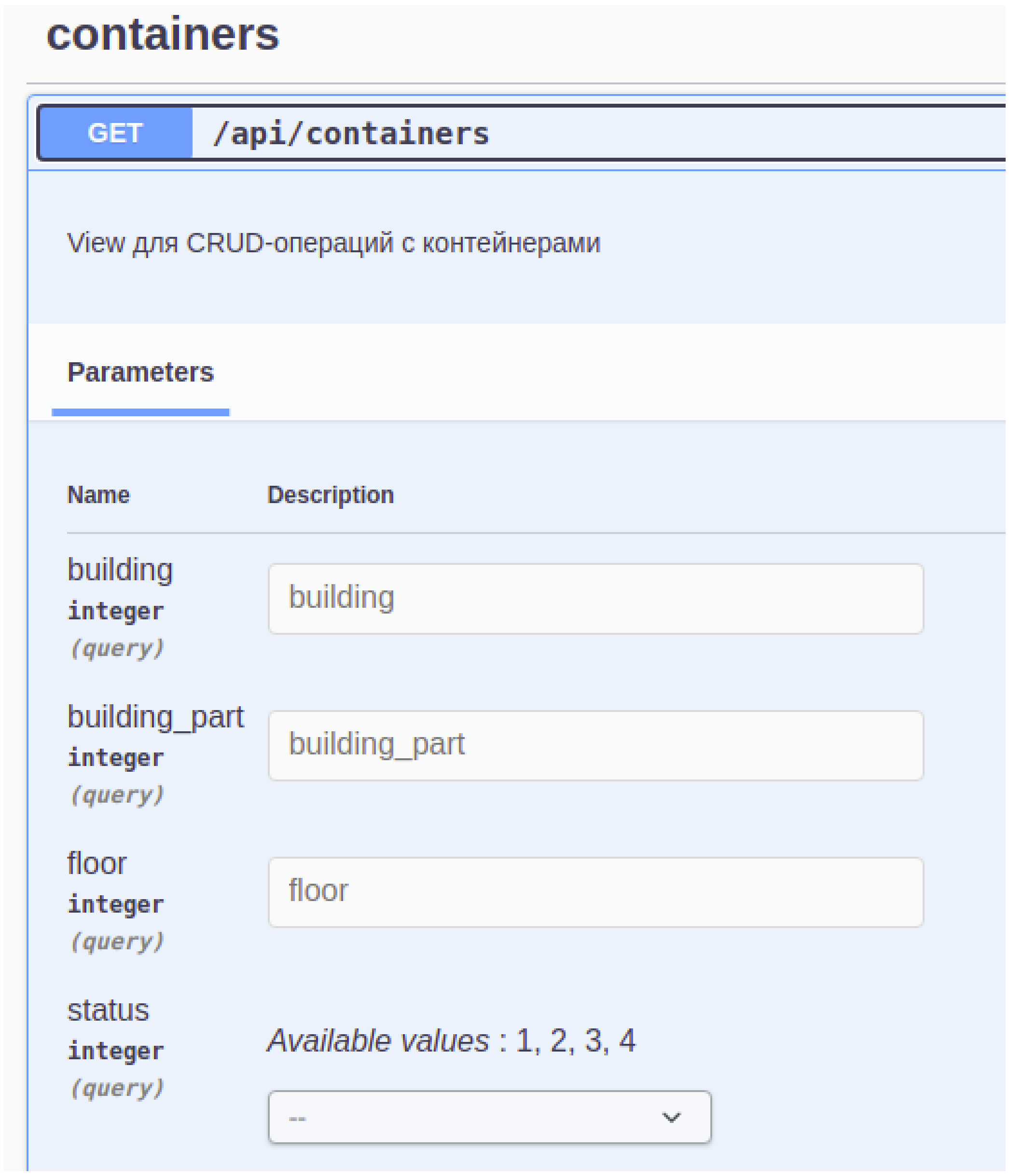
```
urlpatterns = [
   path("", ContainerListView.as_view()),
   path("/public-add", ContainerPublicAddView.as_view()),
   path("/<int:pk>", ContainerDetailView.as_view()),
   path("/<int:pk>/empty", EmptyContainerView.as_view()),
   path("/<int:pk>/sticker", ContainerStickerView.as_view()),
   path("/<int:pk>/request-activation",
        ContainerActivationRequestView.as_view()),
   path("/<int:pk>/activate",
        ContainerActivationView.as_view()),
]
```

В сервисе есть авторизация/регистрация средствами Djoser.

Документация

Документация оформлена как коллекция в Postman: https://documenter.getpostman.com/view/12771205/UVJeGcVd

Документация в Swagger-UI:


containers

GET /api/containers

View для CRUD-операций с контейнерами

Parameters

Name	Description
building integer (query)	building
building_part integer (query)	building_part
floor integer (query)	floor
status integer (query)	*Available values* : 1, 2, 3, 4 --

Документация с помощью MkDocs:

https://e-kondr01.github.io/rcs_back/

Вывод: средствами Django Rest Framework был реализован бэкенд для веб-сервиса Recycle Starter.