

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Институт информационных технологий и радиоэлектроники
Кафедра информатики и защиты информации

ОТЧЁТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №5
Финальный отчет

Преподаватель:
(доцент кафедры ИЗИ)

(подпись, дата)

Д.В. Мишин

Студенты:

(гр.ИСБ-117)

(подпись, дата)

И.С. Бедняцкий
А.А. Лысов
Е.Е. Калитин

Владимир 2021

Оглавление

Оглавление.....	2
Введение.....	4
1. Описание стека и бизнес-процессов разработанного программного обеспечения.....	5
1.1 Стек технологий.....	5
1.2 Структура модулей серверной части	5
1.3 Функциональные схемы бизнес-процессов системы	6
1.4 Структура хранения данных	8
1.4.1 Регистрационный центр	8
1.4.2 Сертификационный центр.....	9
1.4.3 Клиентский центр	10
2. Описание реализованного программного обеспечения	11
2.1 Регистрационный центр	11
2.2 Сертификационный центр.....	12
2.3 Клиентский центр	12
2.4 Криптография	14
2.4.1 Межмодульное взаимодействие	14
2.4.2 Цифровая подпись сертификата	16
2.4.3 Обмен файлами	18
2.4.3.1 Используемые алгоритмы	18
2.4.3.2 Реализация	18
3. Документация	20
3.1 Инструкции для администратора по развертыванию модулей.....	20

3.2 Инструкция для администратора по использования административной панели	23
3.2.1 Административная панель регистрационного центра	24
3.2.2 Административная панель удостоверяющего центра	25
3.2.3 Административная панель клиентского центра	26
3.3 Инструкции оператора регистрационного центра	27
3.4 Инструкции оператора удостоверяющего центра	29
3.5 Инструкции пользователя клиентского центра	30
4. Тестирование	33
Заключение	34
ПРИЛОЖЕНИЕ 1	35
ПРИЛОЖЕНИЕ 2	37
ПРИЛОЖЕНИЕ 3	39
ПРИЛОЖЕНИЕ 4	41
ПРИЛОЖЕНИЕ 5	42
ПРИЛОЖЕНИЕ 6	44
ПРИЛОЖЕНИЕ 7	46
ПРИЛОЖЕНИЕ 8	49
ПРИЛОЖЕНИЕ 9	51
ПРИЛОЖЕНИЕ 10	53
ПРИЛОЖЕНИЕ 11	54

Введение

В данной лабораторной работе необходимо выполнить следующую задачу:

- Подготовить отчет и материалы для защиты проекта (ПЗ, презентация, видео демонстрации работы программ, документация на ПО)

1. Описание стека и бизнес-процессов разработанного программного обеспечения

1.1 Стек технологий

Для разработки программного обеспечения используется язык программирования Python 3.8 с использованием фреймворка Django. Django позволяет разрабатывать веб-приложения. Для реализации интерфейса администратора и оператора используются стандартный функционал Django: Django admin и Django templates.

Для взаимодействия серверных модулей между собой используется протокол http с использованием архитектуры Rest. Для этого используется библиотека Django-rest-framework.

Так как Django является синхронным фреймворком, для параллельного выполнения задач серверных модулей используется очередь задач Celery с брокером Redis.

В качестве базы данных везде используется PostgreSQL. Для развертки модулей на импортонезависимом программном обеспечении используются технологии docker и docker-compose.

1.2 Структура модулей серверной части

Серверная часть состоит из двух модулей:

- Регистрационный центр, на котором хранится репозиторий, список аннулированных сертификатов и реестр пользователей. Все запросы, связанные регистрацией, получением сертификата субъекта, проверка действительности сертификата, аннулированием сертификата от субъекта изначально обрабатывает регистрационный центр.
- Удостоверяющий центр, на котором хранится архив сертификатов. Обрабатывает запрос от регистрационного центра на регистрацию и аннулирование сертификатов, а также аннулирует сертификаты, в случае окончания времени действия, по периодической задаче.

1.3 Функциональные схемы бизнес-процессов системы

Серверные модули имеют пять бизнес-процессов, связанных с функционированием PKI. Далее приведены функциональные схемы UML этих бизнес-процессов.

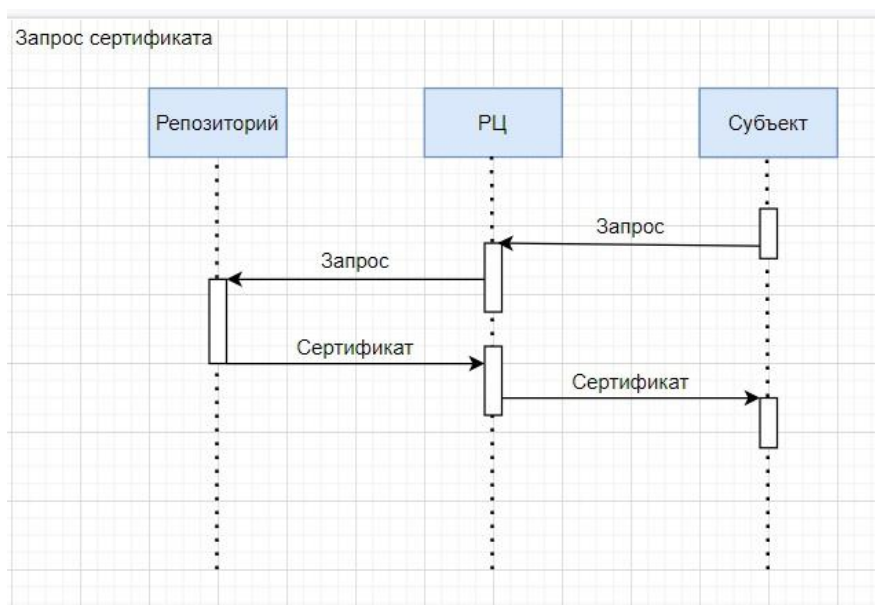


Рисунок 1 Функциональная схема запроса сертификата субъекта

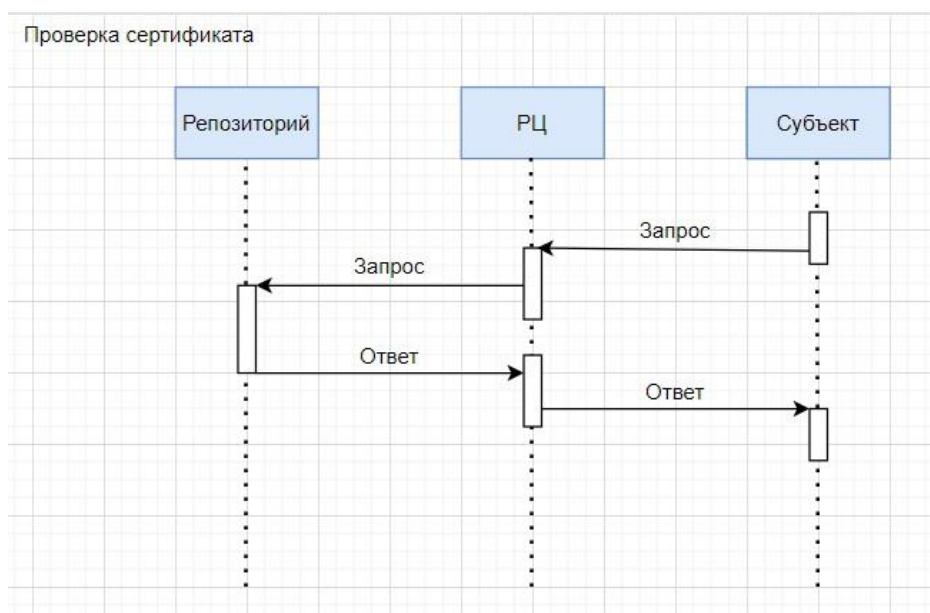


Рисунок 2 Функциональная схема проверки действительности сертификата

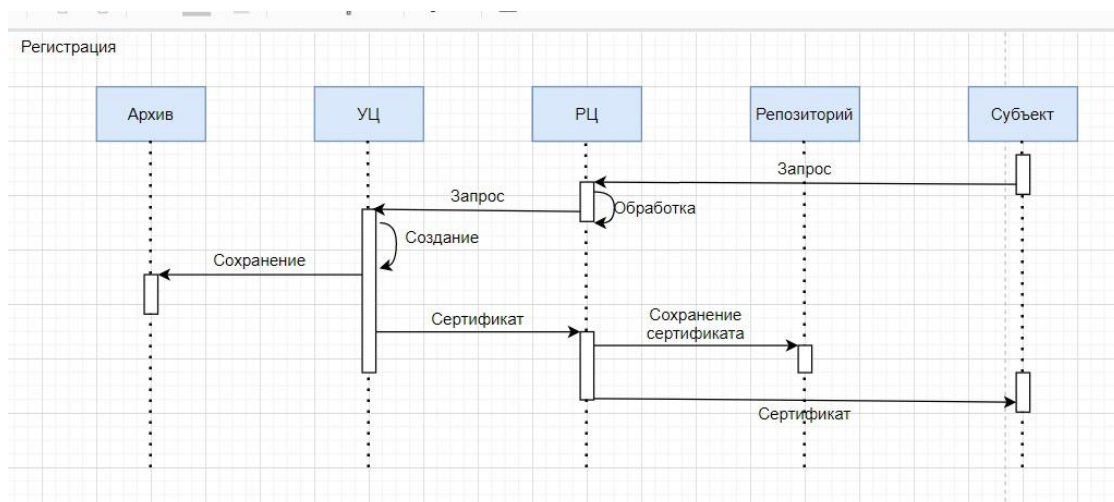


Рисунок 3 Функциональная регистрация сертификата субъекта

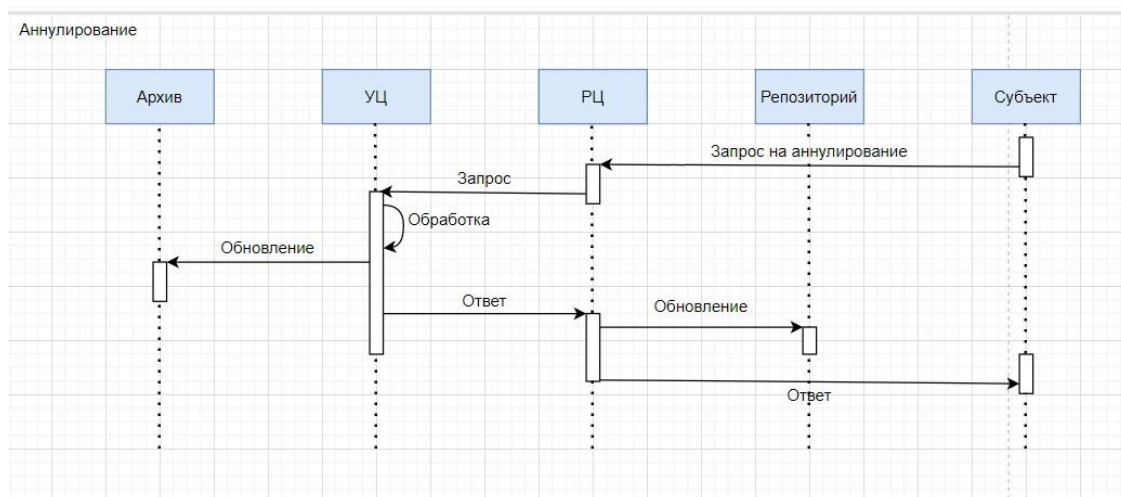


Рисунок 4 Функциональная схема аннулирования сертификата

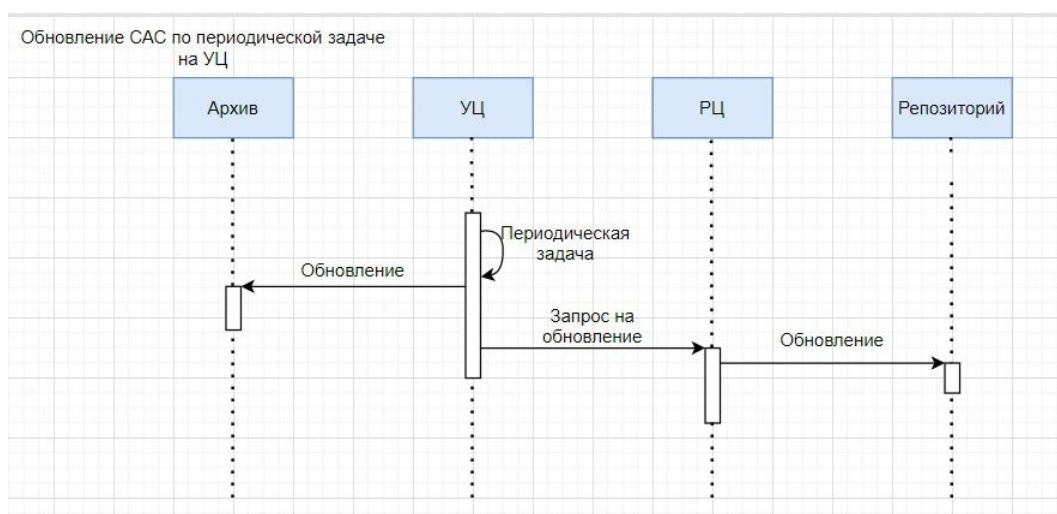


Рисунок 5 Функциональная схема аннулирования сертификата в случае истечения срока действия по периодической задаче со стороны удостоверяющего центра

1.4 Структура хранения данных

1.4.1 Регистрационный центр

Репозиторий, список аннулированных сертификатов и реестр пользователей хранятся на регистрационном модуле в СУБД PostgreSQL.

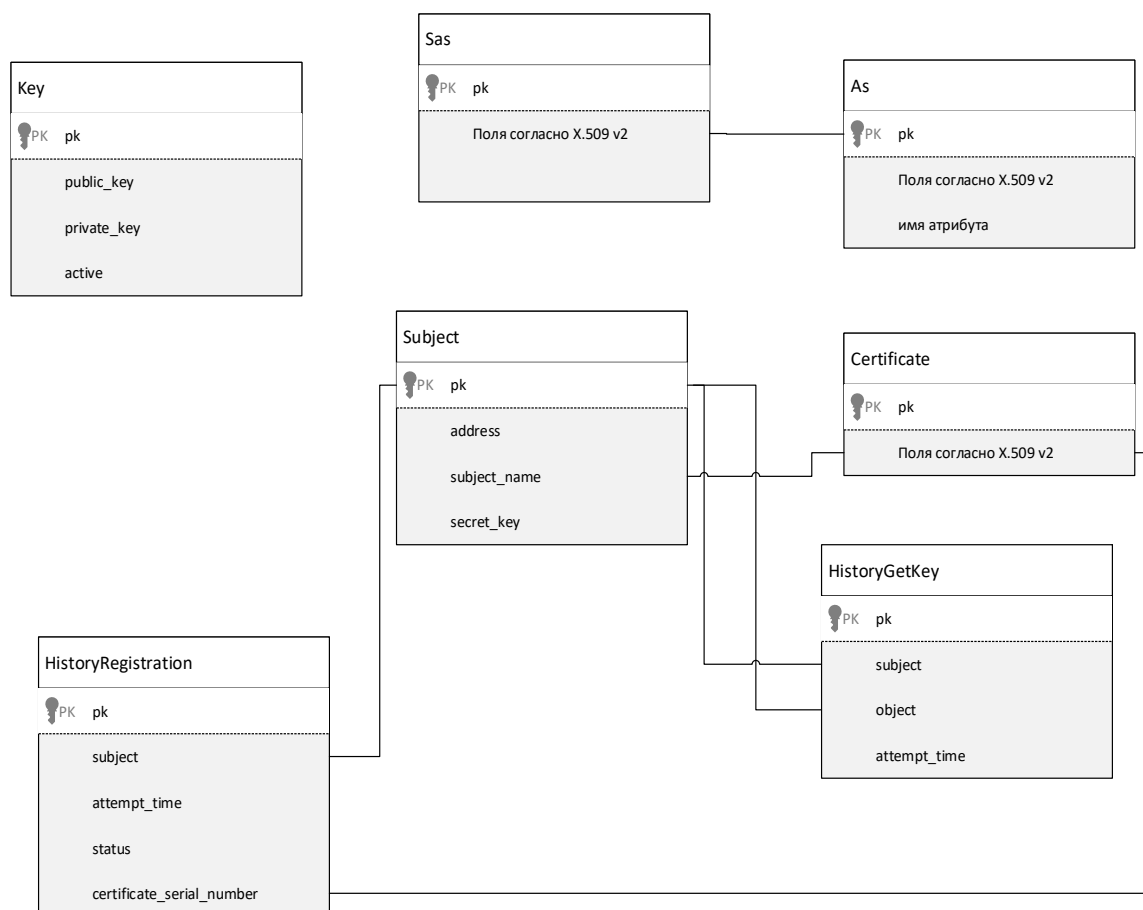


Рисунок 6 БД регистрационного центра

Key – ключи для RSA;

Sas – список аннулированных сертификатов, где As – информация об одном аннулированном сертификате;

Subject – субъекты;

Certificate – репозиторий сертификатов;

HistoryRegistration – история попыток регистрации сертификатов;

HistoryGetKey – история запросов сертификатов субъектов.

Описание моделей базы данных находится по адресу <https://github.com/iliyB/PKI/blob/main/registration/reg/models.py>.

1.4.2 Сертификационный центр

Архив сертификатов хранится на удостоверяющем модулей в СУБД PostgreSQL.

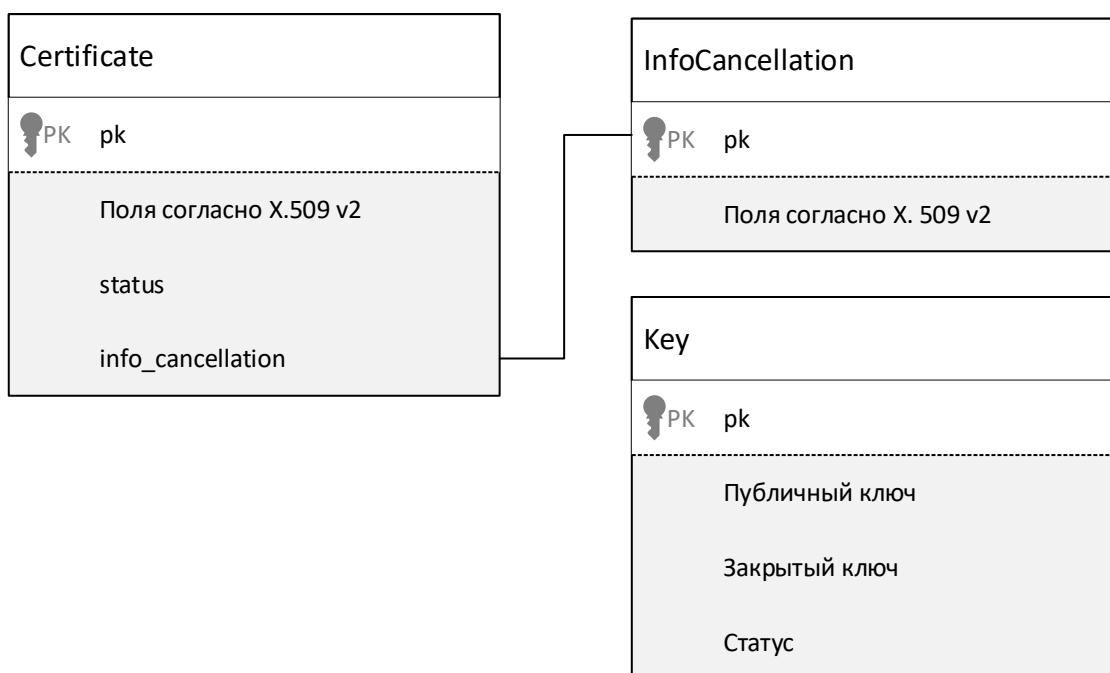


Рисунок 7 БД удостоверяющего центра

Key – ключи для RSA;

Certificate – архив сертификатов, где status указывает на действительность сертификата

InfoCancellation – информация об аннулировании сертификата.

Описание моделей базы данных находится по адресу <https://github.com/iliyB/PKI/blob/main/certification/cert/models.py>.

1.4.3 Клиентский центр

Клиентский модуль хранит данные пользователей и их сертификаты в СУБД PostgresSql.

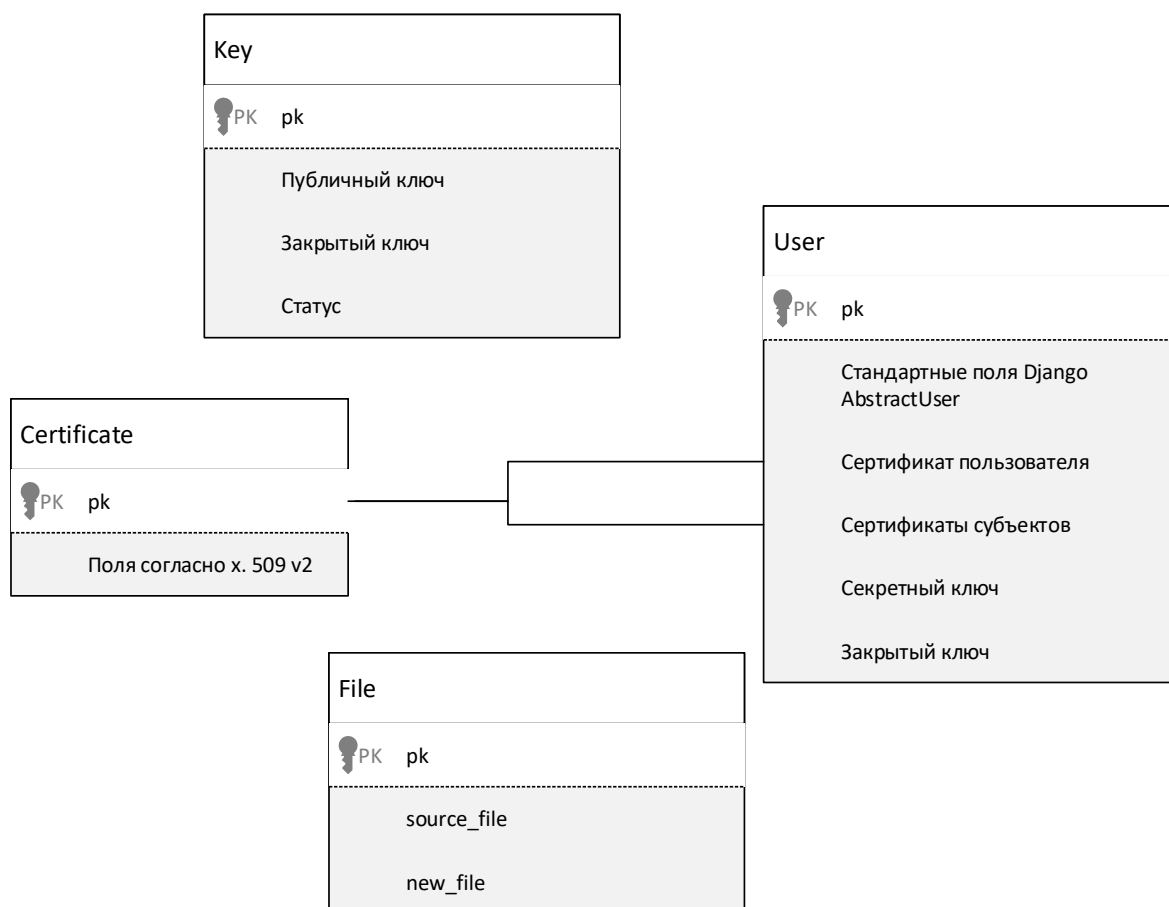


Рисунок 8 БД клиентского центра

Key – ключи для RSA;

Certificate – сертификаты пользователей;

User – модель пользователя системы;

File – модель для временного хранения преобразованных файлов.

Описание моделей базы данных находится по адресу <https://github.com/iliyB/PKI/blob/main/client/cli/models.py>.

2. Описание реализованного программного обеспечения

2.1 Регистрационный центр

Полный код регистрационного центра находится по адресу <https://github.com/iliyB/PKI/tree/main/registration>.

```
urlpatterns = [
    path('login/', views.LoginView.as_view(), name='login_url'),
    path('logout/', views.logout_user, name='logout_url'),
    path('certificate/', views.CertificateListView.as_view(), name='certificate_list_url'),
    path('certificate/<int:pk>', views.CertificateDetailView.as_view(), name='certificate_detail_url'),
    path('sas/', views.SasListView.as_view(), name='sas_list_url'),
    path('sas/sas/<int:pk>', views.AsDetailView.as_view(), name='as_detail_url'),
    path('subjects/', views.SubjectListView.as_view(), name='subject_list_url'),
    path('subjects/<int:pk>', views.SubjectDetailView.as_view(), name='subject_detail_url'),
    path('history-registration/', views.HistoryRegistrationListView.as_view(), name='history_registration_list_url'),
    path('history-get-key/', views.HistoryGetKeyListView.as_view(), name='history_get_key_list_url'),

    path('api/registration/', rest_views.RegistrationView.as_view()),
    path('api/get-key/', rest_views.GetKeyView.as_view()),
    path('api/check-key/', rest_views.CheckKeyView.as_view()),
    path('api/cancellation/', rest_views.CancelledView.as_view()),
    path('api/cert-key/', rest_views.GetCertificationKeyView.as_view()),
    path('api/periodic-canc/', rest_views.PeriodicCancellationView.as_view())
]
```

Рисунок 9 Список url-адресов регистрационного центра

Первая часть списка используется для взаимодействия регистрационного центра с оператором, вторая – для взаимодействия с другими модулями PKI.

В приложении 1 представлен код описания взаимодействия оператора с регистрационным центром. Отображение интерфейса взаимодействия регистрационного центра с оператором происходит посредством html-шаблонов, находящихся в папке “templates”.

В приложении 2 и приложении 3 представлен код описания взаимодействия регистрационного центра с другими модулями PKI.

Регистрационный центр так же ведет учет запросов субъектов на регистрацию сертификатов и запросов субъектов на сертификаты других субъектов.

2.2 Сертификационный центр

Полный код регистрационного центра находится по адресу <https://github.com/iliyB/PKI/tree/main/certification>.

```
urlpatterns = [
    path('login/', views.LoginView.as_view(), name='login_url'),
    path('logout/', views.logout_user, name='logout_url'),
    path('', views.CertificateListView.as_view(), name='certificate_list_url'),
    path('<int:pk>', views.CertificateDetailView.as_view(), name='certificate_detail_url'),

    path('api/registration/', rest_views.RegistrationView.as_view()),
    path('api/cancellation/', rest_views.CancelledView.as_view()),
    path('api/reg-key/', rest_views.GetRegistrationKeyView.as_view()),
]
```

Рисунок 10 Список адресов удостоверяющего центра

В приложении 4 представлен код описания взаимодействия оператора с удостоверяющим центром. Отображение интерфейса взаимодействия удостоверяющего центра с оператором происходит посредством html-шаблонов, находящихся в папке “templates”.

В приложении 5 и приложении 6 представлен код описания взаимодействия удостоверяющего центра с другими модулями PKI.

2.3 Клиентский центр

Полный код регистрационного центра находится по адресу <https://github.com/iliyB/PKI/tree/main/client>.

```
urlpatterns = [
    path('login/', views.LoginView.as_view(), name='login_url'),
    path('logout/', views.logout_user, name='logout_url'),
    path('my-certificate/', views.MyCertificateView.as_view(), name='my_certificate_url'),
    path('subject-certificates/', views.MySubjectCertificateView.as_view(), name='subject_certificates_url'),
    path('encrypt/', views.EncryptFileView.as_view(), name='encrypt_url'),
    path('decrypt/', views.DecryptFileView.as_view(), name='decrypt_url'),
    path('registration/', views.RegistrationCertificateView.as_view(), name='registration_url'),
    path('cancellation/', views.CancelledView.as_view(), name='cancellation_url'),
    path('check-key/<int:pk>', views.CheckKeyView.as_view(), name='check_key_url'),

    path('get-key/', views.GetKeyView.as_view(), name='get_key_url'),
    path('api/registration/', rest_views.RegistrationView.as_view()),
    path('api/cancellation/', rest_views.CancelledView.as_view()),
    path('api/cert-key/', rest_views.GetCertificationKeyView.as_view())
]
```

Рисунок 11 Список адресов клиентской части

В приложении 7 представлен код описания взаимодействия пользователя с клиентской частью. Отображение интерфейса взаимодействия клиентской части с пользователями происходит посредством html-шаблонов, находящихся в папке “templates”.

В приложении 8 и приложении 9 представлен код описания взаимодействия клиентской части с другими модулями РКІ.

2.4 Криптография

2.4.1 Межмодульное взаимодействие

Ключи RSA для регистрационного и сертификационного центра генерируются при развертке центра с помощью команды:

python manage.py add_keys

```
class Command(BaseCommand):
    help = 'Cli-команда, создающая ключи RSA для ЦС.'
    def handle(self, *args, **kwargs) -> None:
        """
        ~~~~~
        "manage.py add_keys"
        ~~~~~

        private_key = RSA.generate(2048)
        public_key = private_key.publickey()

        key, _ = Key.objects.get_or_create(type=Key.KeyType.Cert, active=True)

        key.private_key.save(
            f'private_key.pem',
            ContentFile(private_key.export_key('PEM')),
            save=True
        )
        key.public_key.save(
            f'public_key.pem',
            ContentFile(public_key.export_key('PEM')),
            save=True
        )

        self.stdout.write(self.style.SUCCESS('The RSA keys generated.'))
```

Рисунок 12 Функция генерации ключей при развертке сервисов

Для передачи открытого ключа сервиса используется периодическая задача, которая отправляет свой ключ другим сервисам.

```
@app.task
def send_key_cli():
    key = Key.objects.filter(
        active=True,
        type=Key.KeyType.Cert
    ).first()
    public_key = RSA.import_key(
        open(key.public_key.path).read()
    )
    public_key = public_key.export_key('PEM').decode()
    response = requests.post(
        f'{settings.CLIENT_ADDRESS}/api/cert-key/',
        data={
            'key': public_key
        }
    )
    response.raise_for_status()
```

Рисунок 13 Пример функции отправки публичного ключа

```

@app.task
def send_key_cli():
    key = Key.objects.filter(
        active=True,
        type=Key.KeyType.Cert
    ).first()
    public_key = RSA.import_key(
        open(key.public_key.path).read()
    )
    public_key = public_key.export_key('PEM').decode()
    response = requests.post(
        f'{settings.CLIENT_ADDRESS}/api/cert-key/',
        data={
            'key': public_key
        }
    )
    response.raise_for_status()

```

Рисунок 14 Пример функции отправки публичного ключа
 Функции, используемые для шифрования запросов сервисами.

```

def encrypt(public_key, certificate: {}) -> bytes:

    byte = bytes(str(certificate), encoding='utf8')
    cipher = PKCS1_OAEP.new(public_key)

    return cipher.encrypt(byte)

def decrypt(private_key, byte: bytes) -> {}:

    cipher = PKCS1_OAEP.new(private_key)
    decrypt_byte = cipher.decrypt(byte)

    return ast.literal_eval(decrypt_byte.decode('utf-8'))

```

Рисунок 15 Функции шифрования и дешифрования запросов

2.4.2 Цифровая подпись сертификата

Функции, используемые для создания и проверки цифровой подписи.

```
def create_signature(private_key, certificate: {}) -> bytes:

    hash = _get_hash_from_json(certificate)

    signature = PKCS1_v1_5.new(private_key)
    return signature.sign(hash)

def check_signature(public_key, certificate: {}, _signature: bytes) -> bool:

    hash = _get_hash_from_json(certificate)
    signature = PKCS1_v1_5.new(public_key)

    return signature.verify(hash, _signature)
```

Рисунок 16 Функции создания и проверки цифровой подписи

Полный файл с криптографическими функциями представлен в ПРИЛОЖЕНИИ 10.

Цифровая подпись сертификата создаётся при создании сертификата на сертификационном центре.

```
certificate = Certificate.objects.create(
    serial_number=User.objects.make_random_password(20),
    id_algorithm_signature='sha',
    publisher_name=settings.CERTIFICATION_NAME,
    start_time=edit_current_time(),
    end_time=edit_current_time() + timedelta(days=90),
    subject_name=subject_name,
    public_key=public_key
)

sign = {
    'serial_number': certificate.serial_number,
    'id_algorithm_signature': certificate.id_algorithm_signature,
    'publisher_name': certificate.publisher_name,
    'start_time': certificate.start_time,
    'end_time': certificate.end_time,
    'subject_name': certificate.subject_name,
    'public_key': certificate.public_key,
}

key = Key.objects.filter(
    active=True,
    type=Key.KeyType.Cert
).first()
private_key = RSA.import_key(
    open(key.private_key.path).read()
)

signature = create_signature(private_key, sign)
certificate.signature = signature
certificate.save()
```

Рисунок 17 Создание цифровой подписи сертификата

Проверка цифровой подписи происходит при получении сертификата клиентским модулем.

```
signature = serializer.validated_data['signature']
# signature = bytes(signature, encoding='utf8')

key = Key.objects.filter(
    active=True,
    type=Key.KeyType.Cert
).first()

public_key = RSA.import_key(
    open(key.public_key.path).read()
)

sign = {
    'serial_number': serializer.validated_data['serial_number'],
    'id_algorithm_signature': serializer.validated_data['id_algorithm_signature'],
    'publisher_name': serializer.validated_data['publisher_name'],
    'start_time': serializer.validated_data['start_time'],
    'end_time': serializer.validated_data['end_time'],
    'subject_name': serializer.validated_data['subject_name'],
    'public_key': serializer.validated_data['public_key'],
}

if not check_signature(public_key, sign, signature):
    return Response(status=400)
certificate = serializer.save()
user = get_user_model().objects.get(username=certificate.subject_name)
user.certificate = certificate
user.save()

return Response(status=status.HTTP_200_OK)
```

Рисунок 18 Проверка цифровой подписи сертификата

2.4.3 Обмен файлами

2.4.3.1 Используемые алгоритмы

Для шифрования и дешифрования используются RSA ключи размером 2048 бит и симметричный алгоритм AES. Для подписи, шифрования и дешифрования файлов будем использовать следующую последовательность:

1. Алиса подписывает сообщение своей цифровой подписью и шифрует ее открытым ключом Боба (асимметричным алгоритмом RSA).
2. Алиса генерирует случайный сеансовый ключ и шифрует этим ключом сообщение (с помощью симметричного алгоритма AES)
3. Сеансовый ключ шифруется открытым ключом Боба (асимметричным алгоритмом RSA).
4. Боб расшифровывает сеансовый ключ своим закрытым ключом.
5. При помощи полученного, таким образом, сеансового ключа Боб расшифровывает зашифрованное сообщение Алисы.
6. Боб расшифровывает и проверяет подпись Алисы.

2.4.3.2 Реализация

Ключи RSA пользователей генерируются при их создании.

```
def save(self, *args, **kwargs):
    super(User, self).save(*args, **kwargs)
    if not self.private_key:
        private_key = RSA.generate(2048)
        self.private_key.save(
            f'{self.username}.pem',
            ContentFile(private_key.export_key('PEM')),
            save=True
        )
```

Рисунок 19 Генерация закрытого ключа пользователя при создании

Для проведения манипуляций над файлом его надо изначально сохранить, поэтому в БД клиентского центра была добавлена таблица, хранящая исходный файл и файл после манипуляций. В модели таблицы также прописаны функции шифрования и дешифрования исходного файлов, использующие утилиты из Приложения 1, после которых новый файл сохраняется в поле “new_file”. Так же после проведения манипуляций с файлом через 30 минут запускается задача, которая удаляет исходный и преобразованный файл из таблицы БД.

```
@app.task
def delete_file(file_id: int):
    file = File.objects.get(id=file_id)
    file.delete()
```

Рисунок 20 Периодическая задача

```
class File(models.Model):
    source_file = models.FileField(upload_to="source_file", blank=True, null=True)
    new_file = models.FileField(upload_to="new_file", blank=True, null=True)
    def __str__(self):
        return str(self.pk)
    def encrypt(self, private_key, public_key):
        byte = encrypt_file(self.source_file.path, private_key, public_key)
        self.new_file.save(
            f'{str(get_user_model().objects.make_random_password())}',
            ContentFile(bytes(byte)),
            save=True
        )
    def decrypt(self, private_key, public_key):
        b, byte = decrypt_file(self.source_file.path, private_key, public_key)
        self.new_file.save(
            f'{str(get_user_model().objects.make_random_password())}',
            ContentFile(bytes(byte)),
            save=True
        )
        return b
    def delete(self, *args, **kwargs):
        if os.path.isfile(self.source_file.path):
            os.remove(self.source_file.path)
        if os.path.isfile(self.new_file.path):
            os.remove(self.new_file.path)
        super(File, self).delete(*args, **kwargs)
```

Рисунок 21 Модель файла с функциями шифрования и дешифрования

После преобразования файла пользователя предоставляется ссылка на скачивание преобразованного файла.

Функции шифрования и дешифрования файлов представлены в приложении 11.

3. Документация

3.1 Инструкции для администратора по развертыванию модулей

Для развертывания необходимо установить следующие библиотеки/программы: git, docker, docker-compose.

Следующие примеры будут показаны для ОС Linux. Для ОС Windows способы установки библиотек и запуска докера могут отличаться.

Шаг 1. Устанавливаем необходимые библиотеки

```
sudo apt install git, docker-ce, docker-compose -y
```

Шаг 2. Запускаем демон для докера

```
sudo systemctl start docker
```

Шаг 3. Клонировем репозиторий

```
git clone https://github.com/iliyB/PKI.git
```

Шаг 4. Подготовка

Если все модули находятся на одной машине, необходимо создать сеть docker:

```
docker network create pki_network
```

В ином случае необходимо изменить настройки модулей, файлы settings/settings.py в каждом модуля:

REGISTRATION_ADDRESS – адрес РЦ

CLIENT_ADDRESS – адрес клиента

CERTIFICATION_ADDRESS – адрес УЦ

Шаг 5. Разворачиваем докер

Для РЦ:

```
cd registration
```

```
docker-compose up --build
```

Для УЦ:

```
cd certification
```

```
docker-compose up --build
```

Для клиента:

```
cd client
```

```
docker-compose up --build
```

Шаг 6. Проводим миграции и создаем суперпользователя, создать ключи RSA

Для РЦ:

```
docker exec -ti registration-web bash
```

```
python manage.py makemigrations
```

```
python manage.py createsuperuser
```

```
python manage.py add_keys
```

Для УЦ:

```
docker exec -ti certification-web bash
```

```
python manage.py makemigrations
```

```
python manage.py createsuperuser
```

```
python manage.py add_keys
```

Для клиента:

```
docker exec -ti client-web bash
```

```
python manage.py makemigrations
```

```
python manage.py createsuperuser
```

Регистрационный модуль запускается разворачивается на 8002 порту, удостоверяющий – на 8001, клиентский – на 8005.

Docker файлы для развертывания регистрационного центра:

- <https://github.com/iliyB/PKI/blob/main/registration/docker-compose.yml>
- <https://github.com/iliyB/PKI/blob/main/registration/Dockerfile>

Docker файлы для развертывания удостоверяющего центра:

- <https://github.com/iliyB/PKI/blob/main/certification/docker-compose.yml>
- <https://github.com/iliyB/PKI/blob/main/certification/Dockerfile>

Docker файлы для развертывания клиентского центра:

- <https://github.com/iliyB/PKI/blob/main/client/docker-compose.yml>
- <https://github.com/iliyB/PKI/blob/main/client/Dockerfile>

3.2 Инструкция для администратора по использованию административной панели

Регистрационный модуль запускается разворачивается на 8002 порту, удостоверяющий – на 8001, клиент – 8005. Далее представлены адреса для входа в административную панель Django:

- url для РЦ: ip:8002/admin/;
- url для УЦ: ip:8001/admin/;
- url для клиента: ip:8005/admin/.

Для входа в административную панель необходимо ввести данные, используемые на шаге 5 при развертке модуля.

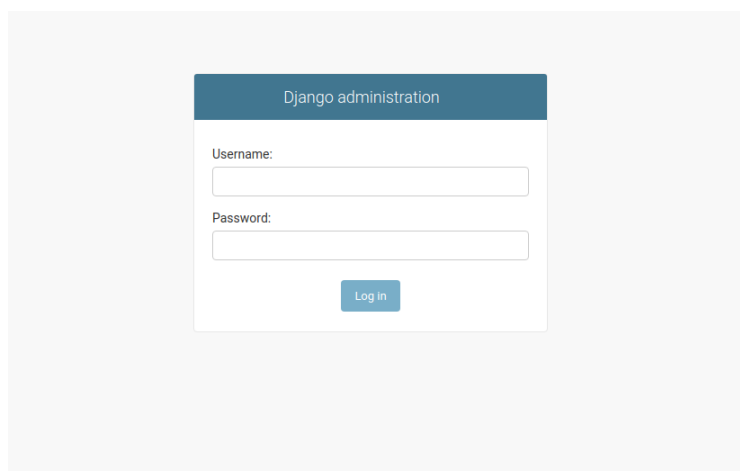


Рисунок 12 Окно авторизации суперпользователя Django

3.2.1 Административная панель регистрационного центра

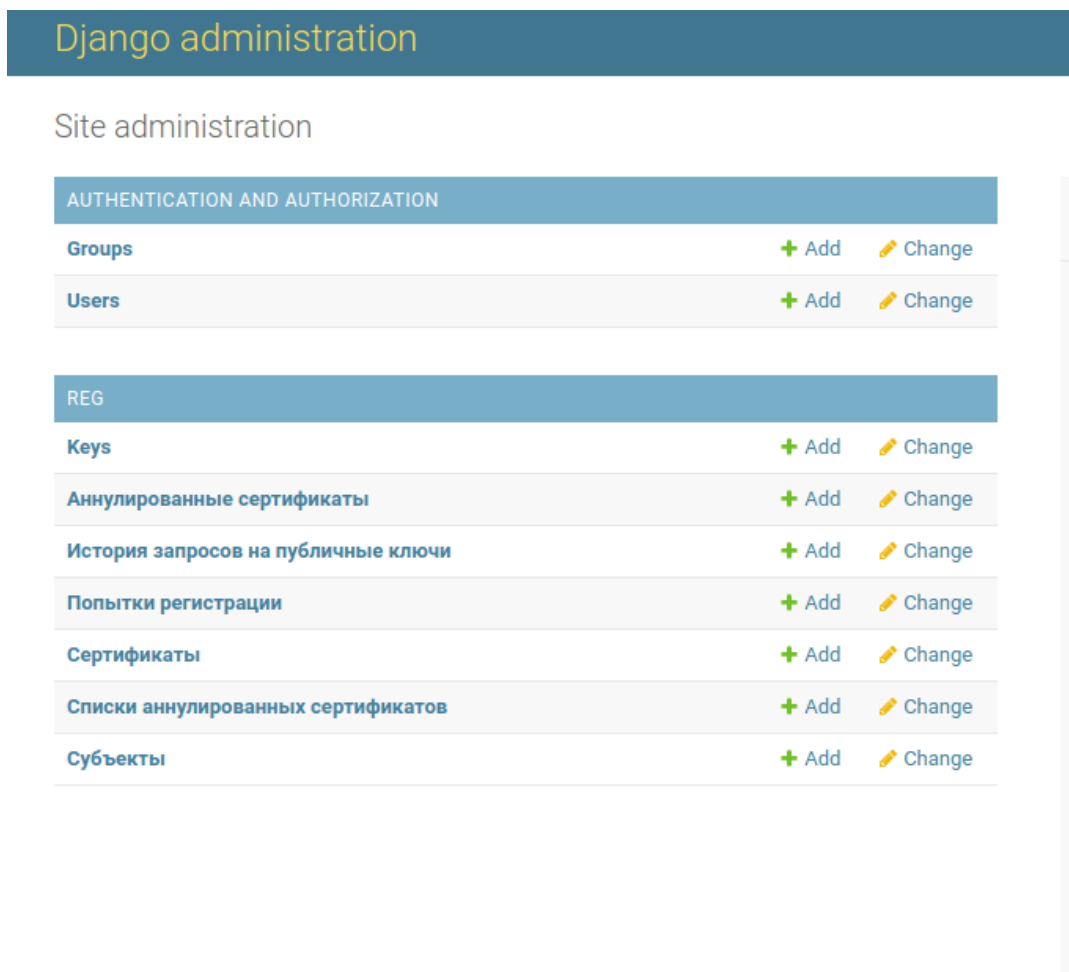


Рисунок 13 Административная панель регистрационного центра

Здесь представлены все таблицы, существующие в БД. Суперадминистратор имеет права реализовывать абсолютно все действия над данными.

Таблицы:

- Users — таблица, используемые для хранения авторизационных данных операторов;
- Keys — ключи RSA;
- Субъекты — реестр пользователей PKI;
- Сертификаты — публичный репозиторий;
- Списки аннулированных сертификатов, аннулированные сертификаты — САС;

- Попытки регистрации — хранит лог истории попыток регистрации сертификатов субъектами;
- История запросов на публичные ключи — хранит лог историй всех запросов на сертификаты субъектов.

3.2.2 Административная панель удостоверяющего центра

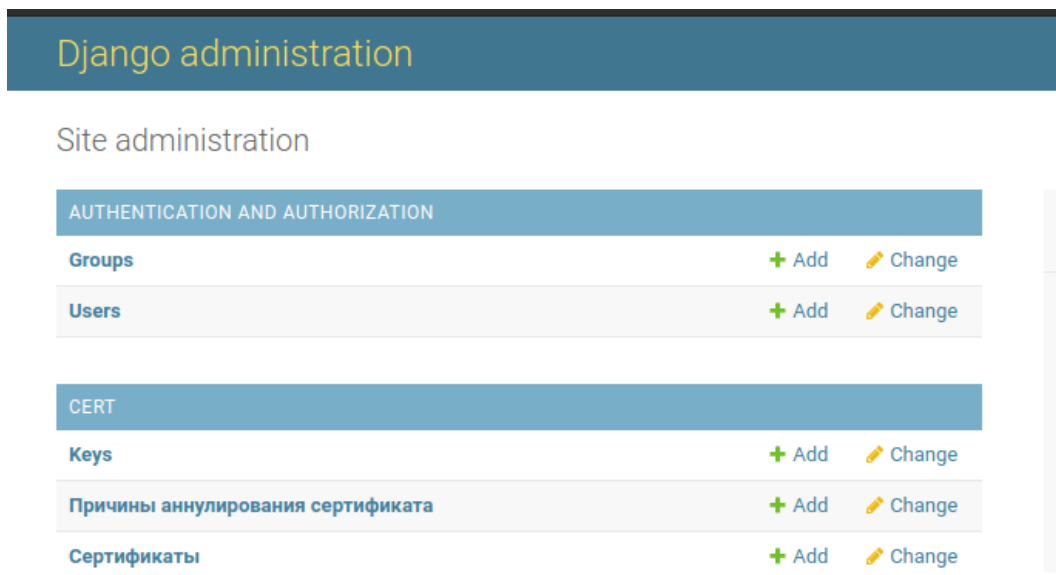


Рисунок 14 Административная панель удостоверяющего центра

Здесь представлены все таблицы, существующие в БД. Суперадминистратор имеет права реализовывать абсолютно все действия над данными.

Таблицы:

- Users — таблица, используемые для хранения авторизационных данных операторов;
- Keys — ключи RSA;
- Сертификаты — архив сертификатов;
- Причины аннулирования сертификатов — хранит структуры, привязываемые к аннулированным сертификатам.

3.2.3 Административная панель клиентского центра



Рисунок 25 Панель администратора

Здесь представлены все таблицы, существующие в БД. Суперадминистратор имеет права реализовывать абсолютно все действия над данными.

Таблицы:

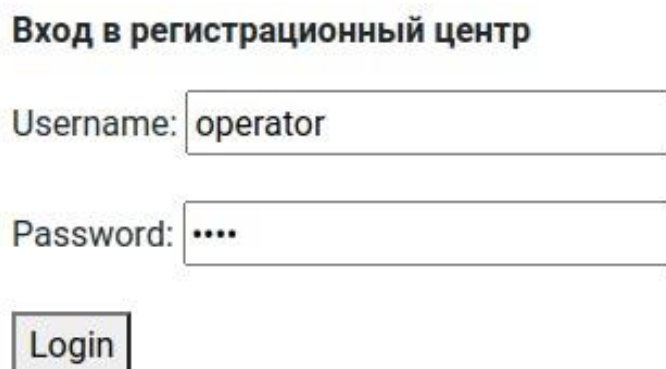
Users — таблица, хранящая информация о пользователях клиентской части;

Keys — ключи RSA;

Сертификаты — сертификаты пользователей, хранящиеся на клиентской части.

3.3 Инструкции оператора регистрационного центра

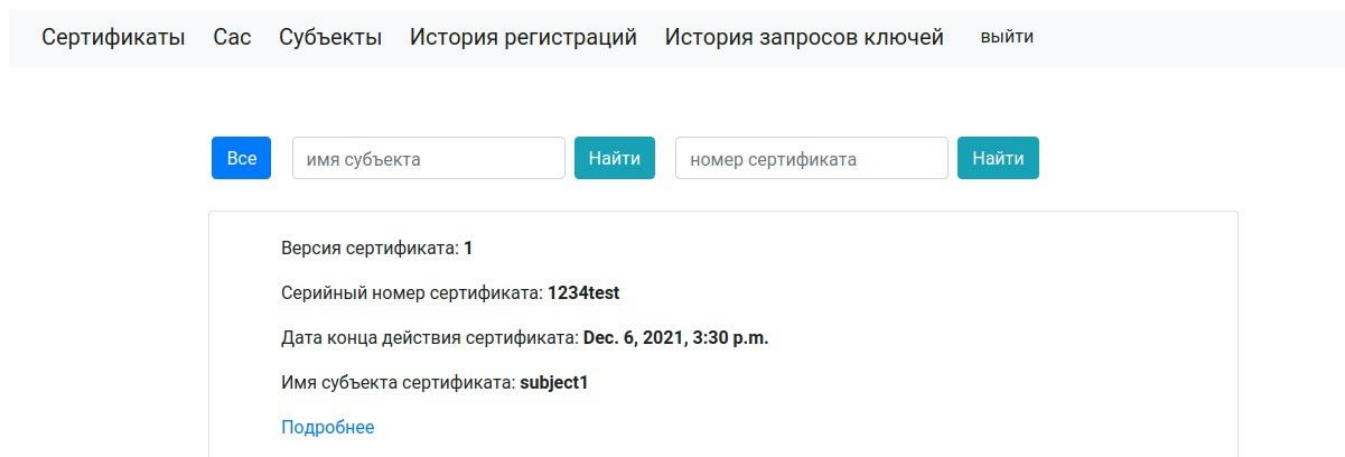
Для входа в регистрационный центр необходимо ввести в адресной строке браузера следующий url - ip:8002/login/. Откроется окно, в котором необходимо ввести авторизационные данные оператора.



The screenshot shows a login interface titled "Вход в регистрационный центр". It contains two input fields: "Username:" with the value "operator" and "Password:" with masked characters "....". Below the fields is a "Login" button.

Рисунок 26 Вход в регистрационный центр

После успешного входа следующие окно.



The screenshot shows a dashboard with a top navigation bar containing links: "Сертификаты", "Сас", "Субъекты", "История регистраций", "История запросов ключей", and "выйти". Below the navigation bar is a search section with a "Все" button, two input fields ("имя субъекта" and "номер сертификата"), and two "Найти" buttons. A large box displays certificate details: "Версия сертификата: 1", "Серийный номер сертификата: 1234test", "Дата конца действия сертификата: Dec. 6, 2021, 3:30 p.m.", and "Имя субъекта сертификата: subject1". A "Подробнее" link is at the bottom.

Рисунок 27 Сертификаты регистрационного центра

Вкладка «Сертификаты»

Показывает текущий публичный репозиторий, размещенный на регистрационном центре. Позволяет искать сертификаты по имени субъекта сертификата и серийному номеру сертификата.

Вкладка «Сас»

Показывает текущий список аннулированных сертификатов, размещенный на регистрационном центре.

Вкладка «Субъекты»

Показывает реестр пользователей, размещенный на регистрационном центре. Позволяет искать субъекты по имени. Для каждого отдельного субъекта имеется страница, на которой выводится все связанная с ним история регистраций и запросов сертификатов.

The screenshot shows the 'Subjects' tab selected in a navigation bar. Below the navigation bar, there is a search interface with a blue 'Все' button, a text input field containing 'имя субъекта', and a blue 'Найти' button. Below the search bar, there is a list of subjects, with 'subject1' and 'subject2' visible as examples.

Рисунок 28 Вкладка Субъекты

Вкладка «История регистраций»

Показывает все попытки регистраций сертификатов субъектами.

Вкладка «История запросов ключей»

Показывает все запросы запросов сертификатов субъектов.

Вкладка «выйти»

Закрывает текущую сессию оператора.

3.4 Инструкции оператора удостоверяющего центра

Для входа в удостоверяющий центр необходимо ввести в адресной строке браузера следующий url - ip:8001/login/. Откроется окно, в котором необходимо ввести авторизационные данные оператора.

Вход в удостоверяющий центр

- Please enter a correct username and password. Note that both fields may be case-sensitive.

Username:

Password:

Рисунок 29 Пример попытки ввода неправильных данных

После успешного входа откроется вкладка «Сертификаты».

Сертификаты [выйти](#)

Все

Действующие

Аннулированные

Версия сертификата: 1

Серийный номер сертификата: 12

Дата конца действия сертификата: Nov. 5, 2021, 8:18 a.m.

Имя субъекта сертификата: sdf

Является ли сертификат действующим: True

[Подробнее](#)

Версия сертификата: 1

Серийный номер сертификата: 13212412

Дата конца действия сертификата: Nov. 5, 2021, 8:36 a.m.

Имя субъекта сертификата: sdfsd

Является ли сертификат действующим: False

[Подробнее](#)

Рисунок 30 Архив сертификатов

Она показывает архив сертификатов, размещенный на удостоверяющем центре. Позволяет искать сертификаты по имени субъекта, серийному номеру сертификата и по статус сертификата, действующий или аннулированный.

Также имеется возможность просмотра подробной информации о каждом сертификате.

3.5 Инструкции пользователя клиентского центра

Для входа в клиентский центр необходимо ввести в адресной строке браузера следующий url - ip:8005/login/. Откроется окно, в котором необходимо ввести авторизационные данные пользователя.

Вход в клиентский центр

Username:

Password:

Рисунок 31 Вход в клиентский центр

Для регистрации сертификата пользователя необходимо перейти во вкладку “Мой сертификат” и нажать зарегистрировать новый сертификат.

Мой сертификат	Сертификаты субъектов	Зашифровать	Расшифровать	выйти
----------------	-----------------------	-------------	--------------	-------

Рисунок 32 Регистрация сертификата

После нажатия кнопки, в этой вкладке появится информация о сертификате пользователя.

Мой сертификат Сертификаты субъектов Зашифровать Расшифровать выйти

Версия сертификата: 1

Серийный номер сертификата: qNYEaFdvBep7ZjnK77Ee

Дата конца действия сертификата: March 2, 2022, 4:49 p.m.

Имя субъекта сертификата: client2

Аннулировать

Рисунок 33 Сертификат пользователя

На кнопку аннулировать данный сертификат будет аннулирован.

Во вкладке “Сертификаты субъектов” представлены субъекты, чьи сертификаты есть у пользователя.

Мой сертификат Сертификаты субъектов Зашифровать Расшифровать выйти

Запросить сертификат

Версия сертификата: 1

Серийный номер сертификата: jvwahuDbB7FhUuEUJDGh

Дата конца действия сертификата: March 2, 2022, 4:47 p.m.

Имя субъекта сертификата: client1

Проверить

Рисунок 34 Сертификаты субъектов

Для запроса сертификата пользователя необходимо ввести его имя и нажать кнопку запросить сертификат. Для проверки, действителен ли еще сертификат, необходимо нажать кнопку проверить на сертификате. Если сертификат действителен, он останется во вкладке, если нет – пропадет.

Для шифрования файла необходимо перейти во вкладку “Зашифровать”.

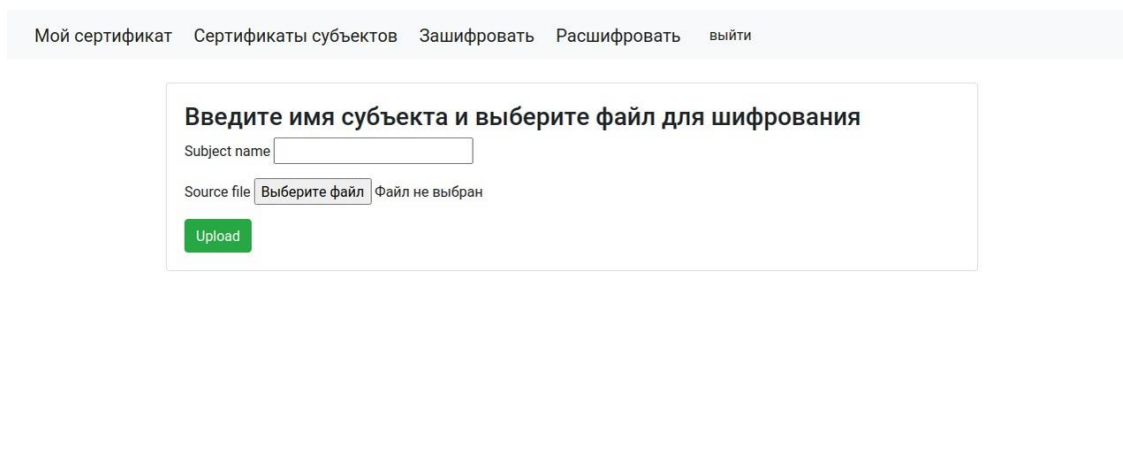


Рисунок 35 Шифрование файла

В данной вкладке необходимо ввести имя субъекта и загрузить файл для шифрования. Если у пользователя нет сертификата введенного субъекта, то после нажатия на кнопку появиться ошибка.

В том случае, если сертификат данного субъекта присутствует после нажатия на кнопку появиться ссылка на скачивание преобразованного файла.

Для дешифрования файла необходимо перейти во вкладку “Расшифровать”.

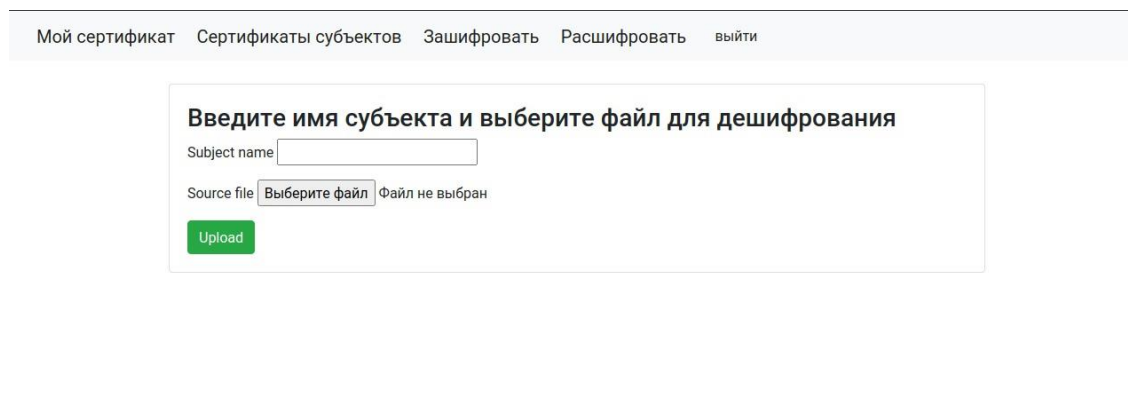


Рисунок 36 Дешифрование файла

В данной вкладке необходимо ввести имя субъекта и загрузить файл для дешифрования. Если у пользователя нет сертификата введенного субъекта, то после нажатия на кнопку появиться ошибка.

В том случае, если сертификат данного субъекта присутствует после нажатия на кнопку появиться ссылка на скачивание преобразованного файла.

Если электронные подписи файлов не совпадают, то появиться предупреждение.

4. Тестирование

Тестирование будет проводиться путем фиксации взаимодействия с модулями РКІ на видеозапись. Все видеозаписи тестирования расположены по адресу –

<https://drive.google.com/drive/folders/1MO9DjYevr5I0D3CPtpG5F32nw3KFQ8Df>.

Функциональные схемы регистрации, получения сертификата пользователя, шифрования и дешифрования файла представлены в видеозаписи “test1”.

Функциональные схемы аннулирования сертификата и проверки действительности сертификата представлены в видеозаписи “test2”.

Интерфейс оператора регистрационного центра представлен в видеозаписи “test3”.

Интерфейс оператора сертификационного центра представлен в видеозаписи “test4”.

Заключение

В данной лабораторной работе была выполнена поставленная задача:

- Подготовлен отчет и материалы для защиты проекта (ПЗ, презентация, видео демонстрации работы программ, документация на ПО)

ПРИЛОЖЕНИЕ 1

Файл <https://github.com/iliyB/PKI/blob/main/registration/reg/views.py>

```
from django.contrib.auth import logout
from django.contrib.auth.decorators import login_required
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth.views import LoginView
from django.db.models import Q
from django.shortcuts import redirect
from django.views.generic import ListView, DetailView
from django.contrib.auth.mixins import LoginRequiredMixin

from .models import Certificate, Sas, Subject, HistoryRegistration,
HistoryGetKey, As

class LoginUser(LoginView):
    form_class = AuthenticationForm
    template_name = "registration/login.html"

@login_required
def logout_user(request):
    logout(request)
    return redirect("login_url")

class CertificateListView(LoginRequiredMixin, ListView):
    model = Certificate

    def get_queryset(self):
        queryset = super(CertificateListView, self).get_queryset()
        subject = self.request.GET.get('subject', None)
        number = self.request.GET.get('number', None)
        if subject:
            return queryset.filter(subject_name__icontains=subject)
        elif number:
            return queryset.filter(serial_number__icontains=number)
        else:
            return queryset.all()

class CertificateDetailView(LoginRequiredMixin, DetailView):
    model = Certificate

class SasListView(LoginRequiredMixin, ListView):
    model = Sas

class AsDetailView(LoginRequiredMixin, DetailView):
    model = As

class SubjectListView(LoginRequiredMixin, ListView):
    model = Subject

    def get_queryset(self):
        queryset = super(SubjectListView, self).get_queryset()
```

```

        subject = self.request.GET.get('subject', None)
        if subject:
            return queryset.filter(subject_name__icontains=subject)
        else:
            return queryset.all()

class SubjectDetailView(LoginRequiredMixin, DetailView):
    model = Subject

class HistoryRegistrationListView(LoginRequiredMixin, ListView):
    model= HistoryRegistration

    def get_queryset(self):
        queryset = super(HistoryRegistrationListView, self).get_queryset()
        subject = self.request.GET.get('subject', None)
        if subject:
            return queryset.filter(subject__subject_name__icontains=subject)
        else:
            return queryset.all()

class HistoryGetKeyListView(LoginRequiredMixin, ListView):
    model = HistoryGetKey

    def get_queryset(self):
        queryset = super(HistoryGetKeyListView, self).get_queryset()
        subject = self.request.GET.get('subject', None)
        if subject:
            return queryset.filter(Q(subject__subject_name__icontains=subject) |
Q(object__subject_name__icontains=subject))
        else:
            return queryset.all()

```

ПРИЛОЖЕНИЕ 2

Файл https://github.com/iliyB/PKI/blob/main/registration/reg/rest_views.py

```
from django.core.files.base import ContentFile
from rest_framework.response import Response
from rest_framework.views import APIView
from rest_framework import status

from .serializers import *
from .models import Subject
from .rest_tasks import *

class RegistrationView(APIView):

    def post(self, request):
        serializer = RequestRegistrationSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        subject_name = serializer.validated_data['subject_name']

        public_key = serializer.validated_data.pop('public_key')
        subject, created =
Subject.objects.update_or_create(**serializer.validated_data)

        if Certificate.objects.filter(subject_name=subject_name).exists():
            HistoryRegistration.objects.create(subject=subject, status=False)
            return Response(status=status.HTTP_400_BAD_REQUEST)

        registration.delay(public_key, subject_name)

        return Response(status=status.HTTP_200_OK)

class GetKeyView(APIView):

    def post(self, request):
        serializer = GetKeySerializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        object_name = serializer.validated_data['object_name']
        subject_name = serializer.validated_data['subject_name']
        if Certificate.objects.filter(subject_name=object_name).exists():
            object, __ = Subject.objects.get_or_create(subject_name=object_name)
            subject, __ =
Subject.objects.get_or_create(subject_name=subject_name)
            h = HistoryGetKey.objects.create(subject=subject, object=object)
            print(h)
            h.save()
            certificate = Certificate.objects.get(subject_name=object_name)
            return Response(CertificateSerializer(certificate).data)
        else:
            return Response(status=status.HTTP_400_BAD_REQUEST)

class CheckKeyView(APIView):

    def post(self, request):
        serializer = CheckKeySerializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        serial_number = serializer.validated_data['serial_number']
        if Certificate.objects.filter(serial_number=serial_number).exists():
```

```

        return Response(status=status.HTTP_200_OK)
    else:
        return Response(status=status.HTTP_404_NOT_FOUND)

class CancelledView(APIView):

    def post(self, request):
        serializer = RequestCancelledSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        subject_name = serializer.validated_data['subject_name']
        if Subject.objects.filter(subject_name=subject_name).exists():
            subject = Subject.objects.get(subject_name=subject_name)
            if subject.secret_key != serializer.validated_data['secret_key']:
                return Response(status=status.HTTP_403_FORBIDDEN)
            subject.save()

            cancellation.delay(subject_name, serializer.validated_data['code'])
            return Response(status=status.HTTP_200_OK)

        return Response(status=status.HTTP_404_NOT_FOUND)

class GetCertificationKeyView(APIView):

    def post(self, request, *args, **kwargs):
        public_key = request.data.get('key')

        key, created = Key.objects.get_or_create(
            active=True,
            type=Key.KeyType.Cert
        )

        if not created:
            return Response(status=200)

        key.public_key.save(
            f'public_key.pem',
            ContentFile(bytes(public_key, encoding='utf8')),
            save=True
        )

        return Response(status=200)

class PeriodicCancellationView(APIView):

    def post(self, request, *args, **kwargs):
        array = request.data.get('array')

        for serial_number in array:
            certificate = Certificate.objects.get(serial_number=serial_number)
            canc_certificate = As()
            canc_certificate.certificate_serial_number =
certificate.serial_number
            canc_certificate.reason_code = "6"
            canc_certificate.save()

            sas = Sas.objects.all().first()
            sas.certificate.add(canc_certificate)
            sas.save()
            certificate.delete()

```

ПРИЛОЖЕНИЕ 3

Файл https://github.com/iliyB/PKI/blob/main/registration/reg/rest_tasks.py

```
import requests
from Crypto.PublicKey import RSA

from django.conf import settings

from settings.celery import app
from .models import *
from .serializers import *

@app.task
def registration(public_key: str, subject_name: str) -> None:

    response = requests.post(
        f'{settings.CERTIFICATION_ADDRESS}/api/registration/',
        data={
            'subject_name': subject_name,
            'public_key': public_key,
        })

    response.raise_for_status()

    serializer = CertificateSerializer(data=response.json())
    serializer.is_valid(raise_exception=True)
    certificate = serializer.save()

    subject = Subject.objects.get(subject_name=subject_name)
    HistoryRegistration.objects.create(
        subject=subject,
        certificate_serial_number=serializer.validated_data['serial_number']
    )

    response = requests.post(
        f'{settings.CLIENT_ADDRESS}/api/registration/',
        data=CertificateSerializer(certificate).data
    )

    response.raise_for_status()

    return None

@app.task
def cancellation(subject_name: str, code: int) -> None:

    response = requests.post(
        f'{settings.CERTIFICATION_ADDRESS}/api/cancellation/',
        data={
            'subject_name': subject_name,
            'code': code
        })

    response.raise_for_status()

    serializer = ResponseCancelledSerializer(data=response.json())
    serializer.is_valid(raise_exception=True)
```

```

serial_number = serializer.validated_data['serial_number']
code = serializer.validated_data['code']

certificate = Certificate.objects.get(serial_number=serial_number)
canc_certificate = As()
canc_certificate.certificate_serial_number = certificate.serial_number
canc_certificate.reason_code = str(code)
canc_certificate.save()

sas = Sas.objects.all().first()
sas.certificate.add(canc_certificate)
sas.save()
certificate.delete()

response = requests.post(
    f'{settings.CLIENT_ADDRESS}/api/cancellation/',
    data={
        'serial_number': serial_number,
        'code': code
    }
)

response.raise_for_status()

return None

@app.task
def send_key():

    key = Key.objects.filter(
        active=True,
        type=Key.KeyType.Reg
    ).first()

    public_key = RSA.import_key(
        open(key.public_key.path).read()
    )

    public_key = public_key.export_key('PEM').decode()

    response = requests.post(
        f'{settings.CERTIFICATION_ADDRESS}/api/reg-key/',
        data={
            'key': public_key
        }
    )

    response.raise_for_status()

```


ПРИЛОЖЕНИЕ 4

Файл <https://github.com/iliyB/PKI/blob/main/certification/cert/views.py>

```
from django.contrib.auth import logout
from django.contrib.auth.decorators import login_required
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth.views import LoginView
from django.shortcuts import redirect
from django.views.generic import ListView, DetailView
from django.contrib.auth.mixins import LoginRequiredMixin

from .models import Certificate

class LoginUser(LoginView):
    form_class = AuthenticationForm
    template_name = "registration/login.html"

@login_required
def logout_user(request):
    logout(request)
    return redirect("login_url")

class CertificateListView(LoginRequiredMixin, ListView):
    model = Certificate

    def get_queryset(self):
        queryset = super(CertificateListView, self).get_queryset()
        subject = self.request.GET.get('subject', None)
        filter = self.request.GET.get('active', None)
        number = self.request.GET.get('number', None)

        if number:
            return queryset.filter(serial_number__icontains=number)
        elif subject:
            if filter is None:
                return queryset.filter(subject_name__icontains=subject)
            elif filter.lower() == 'true':
                return queryset.filter(active=True,
subject_name__icontains=subject)
            elif filter.lower() == 'false':
                return queryset.filter(active=False,
subject_name__icontains=subject)
            else:
                return queryset.filter(subject_name=subject)
        else:
            if filter is None:
                return queryset
            elif filter.lower() == 'true':
                return queryset.filter(active=True)
            elif filter.lower() == 'false':
                return queryset.filter(active=False)
            else:
                return queryset

class CertificateDetailView(LoginRequiredMixin, DetailView):
    model = Certificate
```

ПРИЛОЖЕНИЕ 5

Файл https://github.com/iliyB/PKI/blob/main/certification/cert/rest_views.py

```
from datetime import timedelta

from Crypto.PublicKey import RSA
from django.contrib.auth.models import User
from django.core.files.base import ContentFile
from rest_framework.response import Response
from rest_framework.views import APIView
from rest_framework import status

from django.conf import settings
from .serializers import *
from .models import *
from .utils import edit_current_time, create_signature

class RegistrationView(APIView):

    def post(self, request):
        serializers = RequestRegistrationSerializer(data=request.data)
        serializers.is_valid(raise_exception=True)
        subject_name = serializers.validated_data['subject_name']
        public_key = serializers.validated_data['public_key']

        certificate = Certificate.objects.create(
            serial_number=User.objects.make_random_password(20),
            id_algorithm_signature='sha',
            publisher_name=settings.CERTIFICATION_NAME,
            start_time=edit_current_time(),
            end_time=edit_current_time() + timedelta(days=90),
            subject_name=subject_name,
            public_key=public_key
        )

        sign = {
            'serial_number': certificate.serial_number,
            'id_algorithm_signature': certificate.id_algorithm_signature,
            'publisher_name': certificate.publisher_name,
            'start_time': certificate.start_time,
            'end_time': certificate.end_time,
            'subject_name': certificate.subject_name,
            'public_key': certificate.public_key,
        }

        key = Key.objects.filter(
            active=True,
            type=Key.KeyType.Cert
        ).first()

        private_key = RSA.import_key(
            open(key.private_key.path).read()
        )

        signature = create_signature(private_key, sign)

        certificate.signature = signature
        certificate.save()
```

```

        return Response(CertificateSerializer(certificate).data)

class CancelledView(APIView):

    def post(self, request):
        serializers = RequestCancelledSerializer(data=request.data)
        serializers.is_valid(raise_exception=True)
        subject_name = serializers.validated_data['subject_name']
        code = serializers.validated_data['code']

        if not Certificate.objects.filter(subject_name=subject_name,
active=True).exists():
            return Response(status=status.HTTP_400_BAD_REQUEST)

        certificate = Certificate.objects.get(subject_name=subject_name,
active=True)
        info = InfoCancellation.objects.create(
            revocation_date=edit_current_time(),
            reason_code=code,
            invalidity_date=edit_current_time()
        )

        certificate.active=False
        certificate.info_cancellation = info
        certificate.save()

        data = {
            'serial_number': certificate.serial_number,
            'code': code
        }
        return Response(data)

class GetRegistrationKeyView(APIView):

    def post(self, request, *args, **kwargs):
        public_key = request.data.get('key')

        key, created = Key.objects.get_or_create(
            active=True,
            type=Key.KeyType.Reg
        )

        if not created:
            return Response(status=200)

        key.public_key.save(
            f'public_key.pem',
            ContentFile(bytes(public_key, encoding='utf8')),
            save=True
        )

        return Response(status=200)

```

ПРИЛОЖЕНИЕ 6

Файл https://github.com/iliyB/PKI/blob/main/certification/cert/rest_tasks.py

```
import requests
from Crypto.PublicKey import RSA

from django.conf import settings

from settings.celery import app
from cert.models import Key, Certificate
from cert.utils import edit_current_time

from cert.models import InfoCancellation

@app.task
def periodic_cancellation():
    cancl = Certificate.objects.filter(end_time__lt=edit_current_time())

    array = []

    for cert in cancl:
        cert.status = False
        info = InfoCancellation.objects.create(
            revocation_date=edit_current_time(),
            reason_code=6,
            invalidity_date=edit_current_time()
        )
        cert.info_cancellation = info
        cert.save()
        array.append(cert.serial_number)

    response = requests.post(
        f'{settings.REGISTRATION_ADDRESS}/api/periodic-canc/',
        data={
            'array': array
        }
    )

    response.raise_for_status()

@app.task
def send_key_cli():
    key = Key.objects.filter(
        active=True,
        type=Key.KeyType.Cert
    ).first()

    public_key = RSA.import_key(
        open(key.public_key.path).read()
    )

    public_key = public_key.export_key('PEM').decode()

    response = requests.post(
        f'{settings.CLIENT_ADDRESS}/api/cert-key/',
        data={
```

```

        'key': public_key
    }
)

response.raise_for_status()

@app.task
def send_key_reg():

    key = Key.objects.filter(
        active=True,
        type=Key.KeyType.Cert
    ).first()

    public_key = RSA.import_key(
        open(key.public_key.path).read()
    )

    public_key = public_key.export_key('PEM').decode()

    response = requests.post(
        f'{settings.REGISTRATION_ADDRESS}/api/cert-key/',
        data={
            'key': public_key
        }
    )

    response.raise_for_status()

```

ПРИЛОЖЕНИЕ 7

Файл <https://github.com/iliyB/PKI/blob/main/client/cli/views.py>

```
from django.contrib.auth import logout
from django.contrib.auth.decorators import login_required
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth.mixins import LoginRequiredMixin
from django.contrib.auth.views import LoginView
from django.shortcuts import redirect, render
from django.utils import timezone
from django.views import View
from django.views.generic import ListView

from Crypto.PublicKey import RSA

from .models import Certificate, File
from .forms import AddSubjectCertificateForm, FileForm
from .rest_tasks import *
from .utils import edit_current_time
from .tasks import delete_file

class LoginUser(LoginView):
    form_class = AuthenticationForm
    template_name = "registration/login.html"

@login_required
def logout_user(request):
    logout(request)
    return redirect("login_url")

class MyCertificateView(LoginRequiredMixin, ListView):
    queryset = Certificate.objects.all()
    template_name = 'cli/my_certificate.html'

    def get_queryset(self):
        return self.queryset.filter(user=self.request.user)

class CancelledView(LoginRequiredMixin, View):

    def get(self, request):
        user = request.user
        cancellation.delay(
            subject_name = user.username,
            secret_key = user.secret_key,
            code = 6
        )
        return redirect('subject_certificates_url')

class RegistrationCertificateView(LoginRequiredMixin, View):

    def get(self, request):
        user = request.user
        private_key = RSA.import_key(
```

```

        open(user.private_key.path).read(),
    )
    public_key = private_key.public_key().export_key('PEM')
    registration.delay(
        subject_name = user.username,
        public_key = public_key.decode(),
        secret_key = user.secret_key
    )
    return redirect('subject_certificates_url')

class MySubjectCertificateView(LoginRequiredMixin, ListView):
    queryset = Certificate.objects.all()
    template_name = 'cli/subject_certificate.html'

    def get_queryset(self):
        cert_pk = self.request.user.certificates.values_list('pk', flat=True)
        return self.queryset.filter(pk__in=cert_pk)

class CheckKeyView(LoginRequiredMixin, View):

    def get(self, request, pk):
        certificate = Certificate.objects.get(pk=pk)
        check_key.delay(
            serial_number=certificate.serial_number
        )
        return redirect('subject_certificates_url')

class GetKeyView(LoginRequiredMixin, View):

    def get(self, request):
        subject = request.GET.get('subject', None)

        if subject is not "":

            get_key.delay(
                subject_name = request.user.username,
                object_name = subject
            )
            return redirect('subject_certificates_url')

        return redirect('subject_certificates_url')

class EncryptFileView(LoginRequiredMixin, View):

    def get(self, request, *args, **kwargs):
        return render(request, 'cli/encrypt.html', context={'form':
FileForm()})

    def post(self, request, *args, **kwargs):
        form = FileForm(request.POST, request.FILES)

        if form.is_valid():
            subject_name = form.cleaned_data.pop('subject_name')

            user = request.user
            subject_certificate =
user.certificates.filter(subject_name=subject_name)

            if not subject_certificate.exists():
                form.add_error('subject_name', 'Нет сертификата данного
пользователя')
            return render(request, 'cli/encrypt.html', context={'form':
form})

```

```

        instance = form.save()

        private_key = RSA.import_key(
            open(user.private_key.path).read()
        )

        public_key = RSA.import_key(
            subject_certificate.last().public_key
        )

        instance.encrypt(private_key, public_key)
        delete_file.apply_async([instance.pk], eta=edit_current_time() +
timezone.timedelta(minutes=30))

        return render(request, 'cli/encrypt.html', context={'file':
instance})

        return render(request, 'cli/encrypt.html', context={'form': form})

class DecryptFileView(LoginRequiredMixin, View):

    def get(self, request, *args, **kwargs):
        return render(request, 'cli/decrypt.html', context={'form': FileForm()})

    def post(self, request, *args, **kwargs):
        form = FileForm(request.POST, request.FILES)

        if form.is_valid():
            subject_name = form.cleaned_data.pop('subject_name')

            user = request.user

            subject_certificate =
user.certificates.filter(subject_name=subject_name)

            if not subject_certificate.exists():
                form.add_error('subject_name', 'Нет сертификата данного
пользователя')
                return render(request, 'cli/decrypt.html', context={'form':
form})

            instance = form.save()

            private_key = RSA.import_key(
                open(user.private_key.path).read()
            )

            public_key = RSA.import_key(
                subject_certificate.last().public_key
            )

            if not instance.decrypt(private_key, public_key):
                form.add_error('subject_name', 'Электронная подпись не
совпадает')
                delete_file.apply_async([instance.pk], eta=edit_current_time() +
timezone.timedelta(minutes=30))

            return render(request, 'cli/decrypt.html', context={'form': form,
'file': instance})

        return render(request, 'cli/decrypt.html', context={'form': form})

```


ПРИЛОЖЕНИЕ 8

Файл https://github.com/iliyB/PKI/blob/main/client/cli/rest_views.py

```
from Crypto.PublicKey import RSA
from django.contrib.auth import get_user_model
from django.core.files.base import ContentFile
from rest_framework.response import Response
from rest_framework.views import APIView
from rest_framework import status

from .serializers import *
from .models import Certificate, Key
from .utils import check_signature

class CancelledView(APIView):

    def post(self, request, *args, **kwargs):
        serializers = ResponseCancelledSerializer(data=request.data)
        serializers.is_valid(raise_exception=True)

        serial_number = serializers.validated_data['serial_number']

        certificate = Certificate.objects.get(serial_number=serial_number)
        certificate.delete()

        return Response(status=status.HTTP_200_OK)

class RegistrationView(APIView):

    def post(self, request, *args, **kwargs):
        serializer = ResponseRegistrationSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)

        signature = serializer.validated_data['signature']
        # signature = bytes(signature, encoding='utf8')

        key = Key.objects.filter(
            active=True,
            type=Key.KeyType.Cert
        ).first()

        public_key = RSA.import_key(
            open(key.public_key.path).read()
        )
        sign = {
            'serial_number': serializer.validated_data['serial_number'],
            'id_algorithm_signature':
serializer.validated_data['id_algorithm_signature'],
            'publisher_name': serializer.validated_data['publisher_name'],
            'start_time': serializer.validated_data['start_time'],
            'end_time': serializer.validated_data['end_time'],
            'subject_name': serializer.validated_data['subject_name'],
            'public_key': serializer.validated_data['public_key'],
        }
        if not check_signature(public_key, sign, signature):
            return Response(status=400)
        certificate = serializer.save()
        user = get_user_model().objects.get(username=certificate.subject_name)
```

```

        user.certificate = certificate
        user.save()

    return Response(status=status.HTTP_200_OK)

class GetCertificationKeyView(APIView):

    def post(self, request, *args, **kwargs):
        public_key = request.data.get('key')

        key, created = Key.objects.get_or_create(
            active=True,
            type=Key.KeyType.Cert
        )

        if not created:
            return Response(status=200)

        key.public_key.save(
            f'public_key.pem',
            ContentFile(bytes(public_key, encoding='utf8')),
            save=True
        )

    return Response(status=200)

```

ПРИЛОЖЕНИЕ 9

Файл https://github.com/iliyB/PKI/blob/main/client/cli/rest_tasks.py

```
import requests

from django.contrib.auth import get_user_model
from django.conf import settings

from settings.celery import app
from .serializers import *

@app.task
def registration(subject_name: str, secret_key: str, public_key):

    data = {
        'subject_name': subject_name,
        'public_key': public_key,
        'secret_key': secret_key
    }

    response = requests.post(
        f'{settings.REGISTRATION_ADDRESS}/api/registration/',
        data=RequestRegistrationSerializer(data).data
    )

    response.raise_for_status()

@app.task
def get_key(subject_name: str, object_name: str):

    if Certificate.objects.filter(subject_name=object_name).exists():
        instance = Certificate.objects.get(subject_name=object_name)
    else:
        data = {
            'subject_name': subject_name,
            'object_name': object_name,
        }

        response = requests.post(
            '{} /api/get-key/'.format(settings.REGISTRATION_ADDRESS),
            data = RequestGetKeySerializer(data).data
        )

        response.raise_for_status()

        serializer = ResponseGetKeySerializer(data=response.json())
        serializer.is_valid(raise_exception=True)
        instance = serializer.save()

    user = get_user_model().objects.get(username=subject_name)
    user.certificates.add(instance)
    user.save()

@app.task
def check_key(serial_number: str):
```

```

data = {
    'serial_number': serial_number
}

response = requests.post(
    f'{settings.REGISTRATION_ADDRESS}/api/check-key/',
    data = RequestCheckKeySerializer(data).data
)

response.raise_for_status()

if not response.ok:
    certificate = Certificate.objects.filter(serial_number=serial_number)
    certificate.delete()

@app.task
def cancellation(subject_name: str, secret_key: str, code: int):

    data = {
        'subject_name': subject_name,
        'code': code,
        'secret_key': secret_key
    }

    response = requests.post(
        f'{settings.REGISTRATION_ADDRESS}/api/cancellation/',
        data = RequestCancelledSerializer(data).data
    )

    response.raise_for_status()

```

ПРИЛОЖЕНИЕ 10

Файлы:

<https://github.com/iliyB/PKI/blob/main/certification/cert/views.py>

<https://github.com/iliyB/PKI/blob/main/registration/reg/views.py>

<https://github.com/iliyB/PKI/blob/main/client/cli/views.py>

```
from django.utils import timezone
import ast

from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA
from Crypto.Cipher import PKCS1_OAEP

def edit_current_time():
    return timezone.localtime(timezone.now())

def encrypt(public_key, certificate: {}) -> bytes:

    byte = bytes(str(certificate), encoding='utf8')
    cipher = PKCS1_OAEP.new(public_key)

    return cipher.encrypt(byte)

def decrypt(private_key, byte: bytes) -> {}:

    cipher = PKCS1_OAEP.new(private_key)
    decrypt_byte = cipher.decrypt(byte)

    return ast.literal_eval(decrypt_byte.decode('utf-8'))

def create_signature(private_key, certificate: {}) -> bytes:

    hash = _get_hash_from_json(certificate)

    signature = PKCS1_v1_5.new(private_key)
    return signature.sign(hash)

def check_signature(public_key, certificate: {}, _signature: bytes) -> bool:

    hash = _get_hash_from_json(certificate)
    signature = PKCS1_v1_5.new(public_key)

    return signature.verify(hash, _signature)

def _get_hash_from_json(json: {}) -> bytes:

    byte = _get_bytes_from_json(json)
    return SHA.new(byte)

def _get_bytes_from_json(json: {}) -> bytes:

    string = ""
    for key, value in json.items():
        string += str(value)

    return bytes(string, encoding='utf-8')
```

ПРИЛОЖЕНИЕ 11

Файл https://github.com/iliyB/PKI/blob/main/client/cli/file_utils.py

[illegible]