

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР
ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
КАФЕДРА ПРОГРАММНОЙ ИНЖЕНЕРИИ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту по дисциплине
«Системное программирование»
на тему: «Объектно-ориентированная реализация программной
системы SILUR в среде Linux»

Руководитель:

_____ кафедры

Выполнил:

ст. гр. ПИ–186

Моргунов А.Г.

ДОНЕЦК – 2021

РЕФЕРАТ

Пояснительная записка к курсовому проекту содержит: 44 страниц, 47 рисунка, 6 источников, 4 приложения.

Цель работы – закрепить полученные знания в области проектирования клиент-серверных систем, а также спроектировать и реализовать клиентское приложение для взаимодействия с сервером агрегации. Для достижения поставленной цели курсового проекта необходимо:

- проанализировать возможности Django;
- разработать комплекс модулей;
- осуществить техническое и рабочее проектирование сайта;
- реализовать эмуляторы, а также провести тестирования спроектированных программ;
- реализовать авторский протокол передачи данных.

Методы исследования – научные источники по агрегационным системам, сокет, методы, алгоритмы взаимодействия с сервером, возможности Django, криптографические алгоритмы.

Объект исследования – клиентское приложение – сайт для агрегационной системы.

Результаты работы – сайт, написанный при помощи Django на Python с поддержкой протоколов передачи данных сервера.

DJANGO, PYTHON, АРХИТЕКТУРА, САЙТ, SOCKET, АГРЕГАТОР, БАЗА ДАННЫХ, AES, RSA, DJANGO TEMPLATE LANGUAGE

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 ПРОЕКТИРОВАНИЕ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ	6
1.1 Web-клиент	6
1.2 Архитектура проекта	6
1.2.1 Упрощенная схема приложения	7
1.2.2 Модуль сокетов	8
1.2.3 Модуль Views	9
1.2.4 Модуль Templates	9
1.2.5 Модуль Models	10
1.3 Переменные сессии	11
2 РАЗРАБОТКА WEB-КЛИЕНТА	12
2.1 Выбор средств реализации. Обоснование выбора	12
2.2 Реализация модулей	12
2.2.1 Модуль сокетов	12
2.2.2 Модуль Views	13
2.2.3 Модуль Templates	15
2.2.3.1 Django template language	15
2.2.3.2 Формы	17
2.2.4 Модуль Models	19
3 ОПИСАНИЕ АВТОРСКОГО ПРОТОКОЛА	21
3.1 Взаимодействие с сервером	21
3.2 Протокол передачи данных	21
3.2.1 Инициализация	22
3.2.2 Аутентификация	23
3.2.3 Достоинства и недостатки авторского протокола	28
ВЫВОДЫ	30
ПЕРЕЧЕНЬ ССЫЛОК	31
ПРИЛОЖЕНИЕ А. ТЕХНИЧЕСКОЕ ЗАДАНИЕ	32

ПРИЛОЖЕНИЕ Б. ЭКРАННЫЕ ФОРМЫ.....	36
ПРИЛОЖЕНИЕ В. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ	41
ПРИЛОЖЕНИЕ Г. ЛИСТИНГ КОДА	42

ВВЕДЕНИЕ

Веб-сайт – совокупность файлов, документов, отраженных при помощи языка программирования таким образом, чтобы их видели пользователи сети Интернет. Другими словами, сайты включают в себя любую текстовую, графическую, аудио- или видеовизуальную информацию, собранную на странице или нескольких страницах [1].

Веб ресурсы в современном мире являются чрезвычайно важным аспектом информационной сферы. Многие разработчики и пользователи стали отказываться от десктопных приложений в пользу веб-приложений из-за того, что они доступны с любого устройства без установки.

В первом разделе описывается проектирование проекта. В частности, архитектура проекта, его модули и выполняемые ими задачи.

Во втором разделе описывается разработка web-клиента. В частности, выбор средств реализации, описание реализации модулей.

В третьем разделе описывается авторский протокол передачи данных между клиентом и сервером.

Целью курсового проекта является закрепление знаний, проектирования, реализация и понимание общей концепции работы клиент-серверного приложения со стороны клиента.

1 ПРОЕКТИРОВАНИЕ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

1.1 Web-клиент

Приложение SILUR агрегирует сообщения из каналов-источников Telegram в другие каналы-приёмники.

В рамках данного курсового проекта разрабатывалась клиентская часть системы SILUR. Которая выполняет следующие работы: предоставляет пользователю интерфейс для взаимодействия с сервером, сохраняет пользовательскую сессию, получает информацию с сервера и отображает ее пользователю.

Для настройки агрегатора клиент использует API сервера, которое предоставляется для клиентов. Клиенты могут создавать новые каналы-источники, каналы-приемники и устанавливать отношение между ними.

Для взаимодействия с сервером в рамках курсового проекта разрабатывался авторский протокол.

1.2 Архитектура проекта

Архитектура проекта обусловлена его направленностью, а именно web направленностью. Для разработки был выбран фреймворк Django, что и определило основную архитектуру проекта.

Django использует архитектуру MVT — Model-View-Template. Model — это модель, которая представляет данные. При разработке в роли модели выступает сервер приложения. View — это представления, которые взаимодействуют с данными. Template — это шаблоны, с помощью которых генерируется пользовательский интерфейс (см. рис. 1.1).

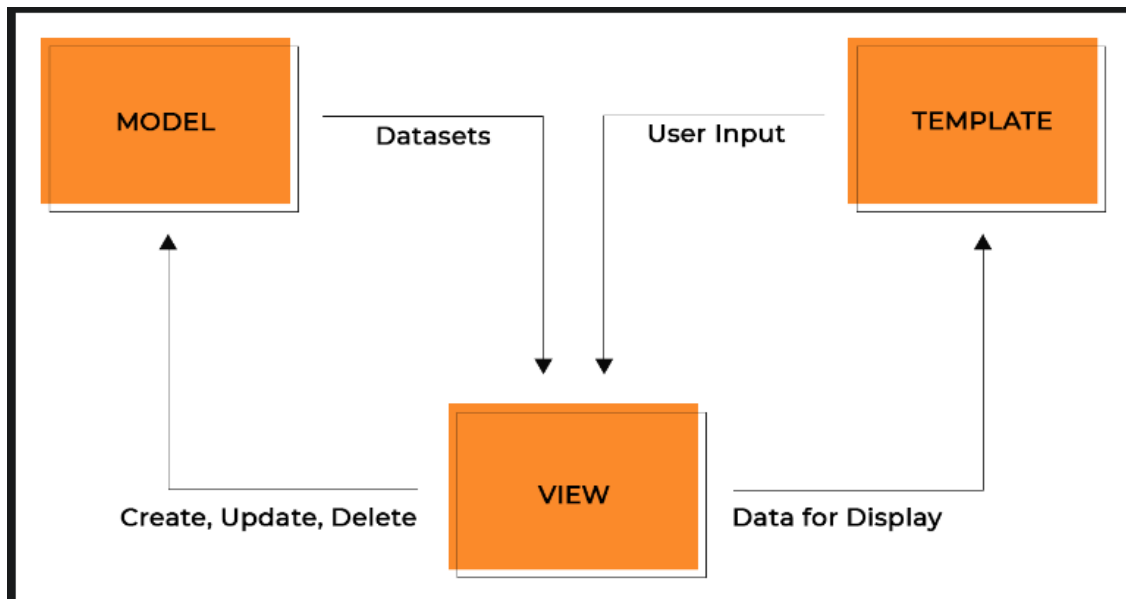


Рисунок 1.1 — Схема архитектуры MVT в Django

В программе есть несколько модулей:

- Модуль Django.
- Модуль сокетов.

1.2.1 Упрощенная схема приложения

Система содержит 2 основных модуля:

- 1) Модуль Django – обеспечивает основную функциональность клиента. Обработывает пользовательский ввод, информацию с сервера, обращается к сокетам для передачи данных между клиентом и сервером;
- 2) Модуль сокетов – предназначен для передачи данных между клиентом и сервером. Также шифрует данные, которые пересылаются.

Упрощенная схема приложения приведена на рисунке 1.2.

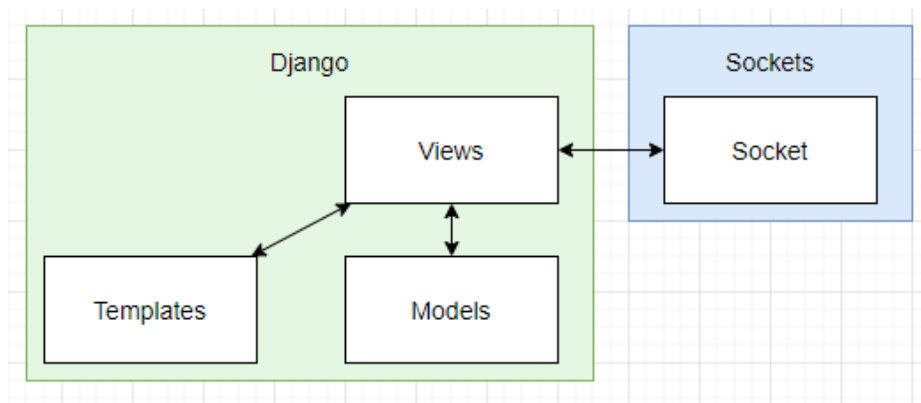


Рисунок 1.2 – Упрощенная схема приложения

Модуль сокетов содержит:

1) Socket – класс, который реализует соединение и передачу информации между клиентом и сервером.

Модуль Django содержит:

- 1) Views – модуль, который осуществляет маршрутизацию.
- 2) Templates – модуль, который отвечает за генерацию web-страниц для пользователя.
- 3) Models – модуль, который отвечает за отображение данных в виде классов.

1.2.2 Модуль сокетов

Модуль сокетов состоит из класса Socket (см. рис. 1.3)

Socket
PACKAGE_SIZE
__init__(self) read_big_number(self) send_int(self, value: int) get_int(self) read_bytes(self) write_bytes(self, message_bytes)

Рисунок 1.3 – Класс Socket

Класс Socket отвечает за прием и отправку данных на сервер.

1.2.3 Модуль Views

Структура модуля Views определяется Django. В этом модуле содержатся функции-обработчики веб-страниц, а также происходит связывание URL-адресов с определенными функциями обработки. Таким образом для каждого адреса можно определить отдельный обработчик.

1.2.4 Модуль Templates

Модуль Templates – это модуль, который содержит HTML-файлы. Django имеет такой инструмент, как Django template language, который позволяет генерировать HTML-файлы динамически. Это дает возможность создавать HTML конструкции, которые определенным образом зависят от данных.

Также модуль Templates включает в себя такой инструмент Django как формы. Формы нужны для описания форм ввода данных, которые будут выводиться пользователю. Каждая форма описывается в виде класса. Структура классов форм представлена на рисунке 1.4.

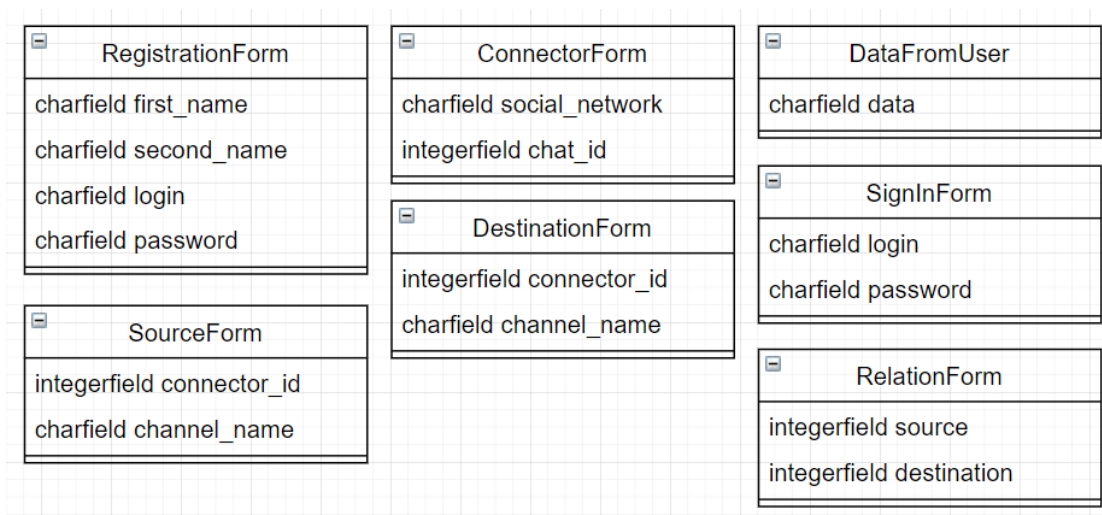


Рисунок 1.4 – Диаграмма классов форм

На диаграмме классов видно, что классы не связаны между собой. Это обусловлено тем, что эти классы предназначены для того, чтобы генерировать HTML-формы. Все приведенные классы являются наследниками класса Form, который предоставляется Django. Django автоматизирует проверку правильности введенных данных, что позволяет ускорить процесс разработки.

1.2.5 Модуль Models

Модуль Models отвечает за группировку данных в виде классов. Каждая сущность, которая должна выводиться пользователю, описывается отдельным классом. Структура модуля Models приведена на рисунке 1.5

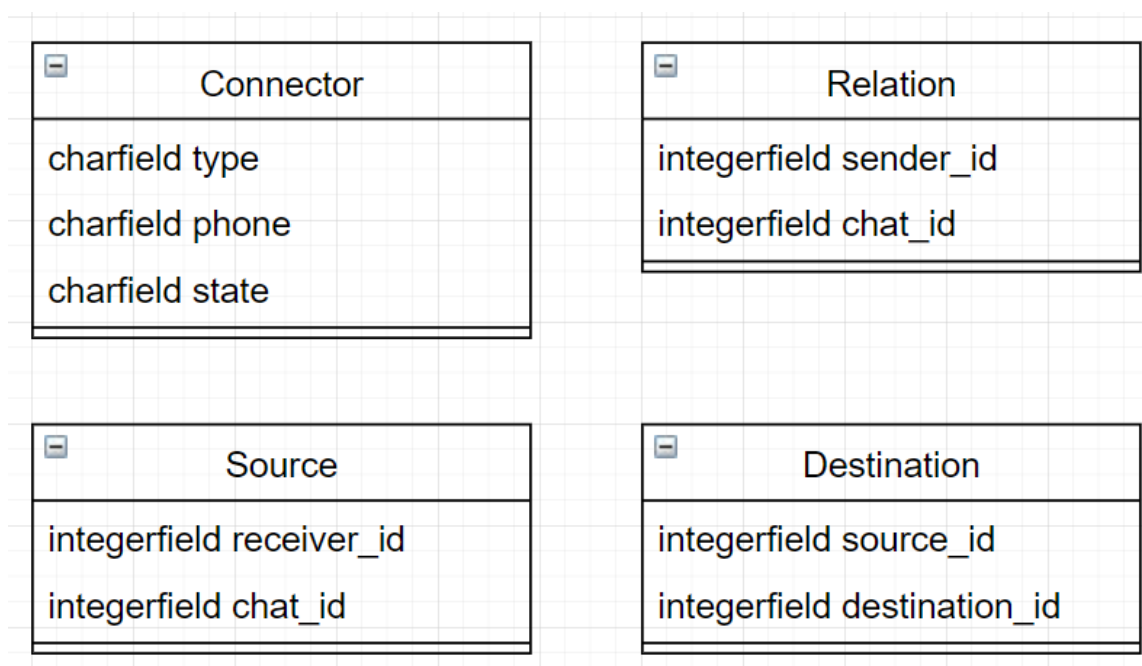


Рисунок 1.5 – Диаграмма классов модуля Models

На диаграмме классов видно, что классы не связаны между собой. Это обусловлено тем, что эти классы используются для отображения данных пользователю или для удобной работы с данными определенной сущности. Однако взаимодействие между этими классами не предусмотрено. Также все эти классы являются наследниками класса Model, который предоставляется Django.

1.3 Переменные сессии

Для хранения данных приложение использует один из механизмов Django, а именно сессии. Они реализованы с помощью SQLite. Это переменные, которые хранятся в БД, при этом для каждой пользовательской сессии эти переменные принимают разные значения.

Таким способом сохраняется подписанный токен пользователя и его логин, что позволяет реализовать авторизацию пользователя без пароля.

2 РАЗРАБОТКА WEB-КЛИЕНТА

2.1 Выбор средств реализации. Обоснование выбора

Для написания клиентской части использовался язык Python с фреймворком Django. Выбор такой комбинации средств разработки обусловлен гибкостью языка, наличием большого количества пользовательских библиотек, которые могут решить любую тривиальную задачу, что должно ускорить разработку.

Также нужно отметить Django. Это чрезвычайно мощный и гибкий инструмент, который исключительно хорошо подходит под разрабатываемое приложение.

2.2 Реализация модулей

2.2.1 Модуль сокетов

Отправка данных осуществляется с помощью команды `write_bytes`. Реализация `write_bytes` приведена на рисунке 2.1

```
def write_bytes(self, message_bytes) -> None:
    self.send_int(len(message_bytes))
    size = len(message_bytes)
    sent_bytes = 0
    while size != sent_bytes:
        sent_bytes += self.socket.send(message_bytes[sent_bytes:])
    # print(f"Sent: {sent_bytes}")
```

Рисунок 2.1 – Реализация метода `write_bytes`

```
def send_int(self, value: int) -> None:
    self.socket.send(value.to_bytes(length=4, byteorder='big'))
```

Рисунок 2.2 – Реализация метода `send_int`

Алгоритм передачи данных, следующий: вычисление размера сообщения, перевод размера в сетевой порядок байт, отправка размера, отправка сообщения.

Метод `read_bytes` симметричен (см. рис. 2.3).

```
def read_bytes(self) -> bytes:
    size = self.get_int()
    storage = bytearray()
    while len(storage) != size:
        storage += self.socket.recv(min(self.PACKAGE_SIZE, size - len(storage)))
    # print("Read")
    return bytes(storage)
```

Рисунок 2.3 – Реализация метода `read_bytes`

```
def get_int(self) -> int:
    int_size = 4
    result = self.socket.recv(int_size)
    result = int.from_bytes(result, byteorder='big')
    return result
```

Рисунок 2.4 – Реализация метода `get_int`

Алгоритм получения данных: прочитайте размер сообщения (4 байта – 32 бита), создайте буфер, переведите размер из сетевого порядка байт в порядок байт для платформы, читайте, пока объем считанной информации не будет равен размеру полученного сообщения.

2.2.2 Модуль Views

Для того, чтобы отобразить страницу пользователю необходимо создать функцию-обработчик запроса, а также связать URL с этой функцией-обработчиком (см. рис. 2.5 – 2.6)

```
def homepage(request):
    response = render(request, 'silur/homepage.html')
    return response
```

Рисунок 2.5 – Функция-обработчик главной страницы

```
path('silur/', views.homepage, name='homepage'),
```

Рисунок 2.6 – Связывание URL 'silur/' с функцией-обработчиком

В функциях обработчиках можно определять метод HTTP запроса, что позволяет по-разному обрабатывать GET и POST запросы (см. рис. 2.7).

```
def connectors(request):
    session = restore_session(request)
    if request.method == 'POST':
        method = request.POST['method']
        method, obj = method.split(' ')
        if method == 'Add':
            session.sign_out()
            if obj == 'connector':
                return redirect('create_connector')
            elif obj == 'destination':
                return redirect('create_destination')
            elif obj == 'source':
                return redirect('create_source')
```

Рисунок 2.7 – Пример обработки POST запроса

Django предоставляет возможность перехода на другую страницу при помощи метода HttpResponseRedirect (см. рис. 2.8)

```
return HttpResponseRedirect('/silur/sign_in')
```

Рисунок 2.8 – Переход на страницу /silur/sign_in

При получении данных от пользователя обрабатывается POST запрос (см. рис. 2.9).

```
if request.method == 'POST':
    print(request.POST)
    method = request.POST['method']
    method, obj = method.split(' ')
```

Рисунок 2.9 – Обработка POST запроса

Также при помощи функции render можно передавать данные в шаблоны для генерации веб-страниц (см. рис. 2.10).

```
data = {
    'connectors': connectors
}
session.sign_out()
return render(request, 'silur/connectors.html', data)
```

Рисунок 2.10 – Передача данных в шаблон

2.2.3 Модуль Templates

2.2.3.1 Django template language

Django предоставляет такой инструмент, как Django template language (Далее DTL) [2]. DTL позволяет создавать шаблоны, которые можно расширять. Метод создания шаблона приведен на рисунке 2.11.

```
{% block main %}{% endblock %}
```

Рисунок 2.11 – Создание блока при помощи DTL

Такие блоки можно расширять в других HTML документах (см. рис. 2.12).

```
{% extends 'silur/base/wrapper.html' %}

{% block homepage_active %}active{% endblock %}
{% block homepage_aria-current %}aria-current="page"{% endblock %}

{% block main %}
    <div class="container"...>
{% endblock %}
```

Рисунок 2.12 – Расширение шаблона wrapper.html

Для расширения нужно указать расширяемый документ, а затем указать расширяемый блок. Все, что будет написано в указанном блоке будет подставлено в исходный документ при его генерации.

Также можно передавать данные, которые будут обрабатываться в HTML документе. Передача параметров, а также генерация документа происходит при помощи метода render (см. рис. 2.13 – 2.14).

```
server_response = session.get_connectors()
connectors = []
server_connectors = server_response['connectors']
for connector_info in server_connectors:
    connector = Connector(connector_info['id'], type=connector_info['type'],
                          state=connector_info['meta_info']['state'])
    connectors.append(connector)
data = {
    'connectors': connectors
}
session.sign_out()
return render(request, 'silur/connectors.html', data)
```

Рисунок 2.13 – Получение и передача данных в HTML документ


```

{% for connector in connectors %}
  <form method="post">
    {% csrf_token %}
    <h4>Connector {{ connector.id }}</h4>
    state: {{ connector.state }}<br>
    type: {{ connector.type }}<br>
  </form>
{% endfor %}

```

Рисунок 2.14 – Обработка данных в HTML-документе

В приведенном в рисунке 2.13 отрывке кода происходит получение информации о соединениях, а также передача его в HTML-документ. В документе эти данные обрабатываются с помощью DTL. Получившаяся страница приведена на рисунке 2.15.

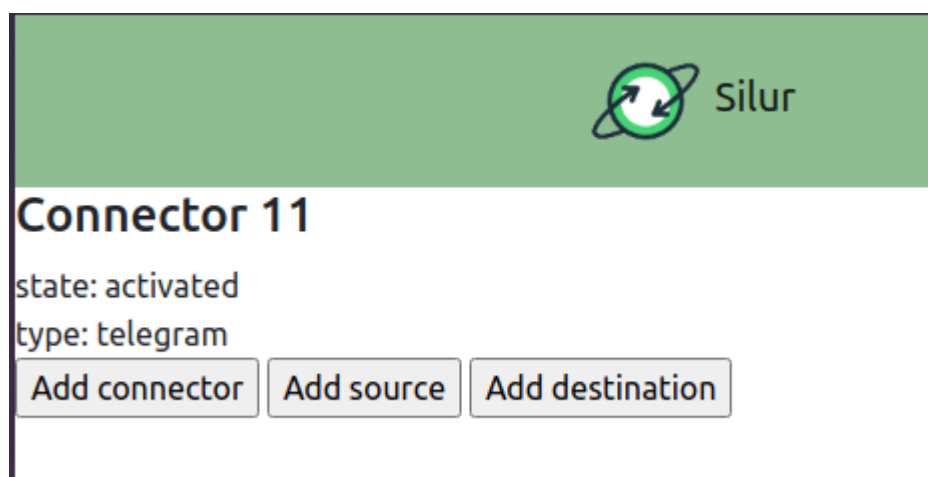


Рисунок 2.15 – Сгенерированная страница

2.2.3.2 Формы

Также частью модуля Templates являются формы. Формы описываются как классы, а затем используются в шаблонах для обеспечения пользовательского ввода на сайте. При описании полей класса можно указывать ограничения для полей (например, максимальная

длина у текстового поля). Пример описания класса формы приведен на рисунке 2.16.

```
class DestinationForm(forms.Form):
    connector_id = forms.IntegerField(label='Connector')
    channel_name = forms.CharField(max_length=200, label='Channel name')
```

Рисунок 2.16 – Описание класса формы

Для использования формы необходимо в функции-обработчике создать объект класса формы и передать его как параметр в шаблон, а в шаблоне обработать переданную форму с помощью DTL. Важно при обработке формы в HTML документе использовать тэги <form>. (см. рис. 2.17-2.18)

```
form = ConnectorForm()
data = {
    'form': form,
    'button_name': 'Create connector'
}
return render(request, 'silur/form_template.html', data)
```

Рисунок 2.17 – Создание объекта формы и передача его в шаблон

```
<form method="post">
{% csrf_token %}
    {% for field in form %}
        {{ field.errors }}<br>
        {{ field.label_tag }}<br>
        {{ field }}<br>
    {% endfor %}
<br>
<input type="submit" value="{{ button_name }}">
</form>
```

Рисунок 2.18 – Обработка формы при помощи DTL

Сразу после тэга <form> идет строка {% csrf_token %}. Это необходимо для того, чтобы при обработке POST запроса убедиться в том, что запрос не был подделан.

При обработке POST запроса, данные для которого были получены из объекта класса формы можно проверить правильность данных средствами Django (см. рис. 2.19)

```
def create_destination(request):  
    session = restore_session(request)  
    if request.method == 'POST':  
        form = DestinationForm(request.POST)  
        if form.is_valid():  
            data = form.cleaned_data
```

Рисунок 2.19 – Проверка правильности данных POST запроса

Переменная data будет содержать json, в котором будут храниться данные, введенные пользователем.

2.2.4 Модуль Models

Для создания модели необходимо создать соответствующий класс (см. рис. 2.20).

```
class Destination(models.Model):  
    sender_id = models.IntegerField()  
    chat_id = models.IntegerField()
```

Рисунок 2.20 – Создание класса модели

В разрабатываемом приложении модели используются для группировки данных, полученных с сервера (см. рис. 2.20).

```
server_connectors = server_response['connectors']
for connector_info in server_connectors:
    connector = Connector(connector_info['id'], type=connector_info['type'],
                          state=connector_info['meta_info']['state'])
    connectors.append(connector)
data = {
    'connectors': connectors
}
session.sign_out()
return render(request, 'silur/connectors.html', data)
```

Рисунок 2.20 – Группировка данных в объекты и передача их в шаблон

Это нужно для использования DTL при выводе данных для пользователей (см. рис. 2.21).

```
{% for connector in connectors %}
<form method="post">
  {% csrf_token %}
  <h4>Connector {{ connector.id }}</h4>
  state: {{ connector.state }}<br>
  type: {{ connector.type }}<br>
</form>
{% endfor %}
```

Рисунок 2.21 – использование DTL для обработки объекта модели

Получившаяся страница приведена на рисунке 2.22.

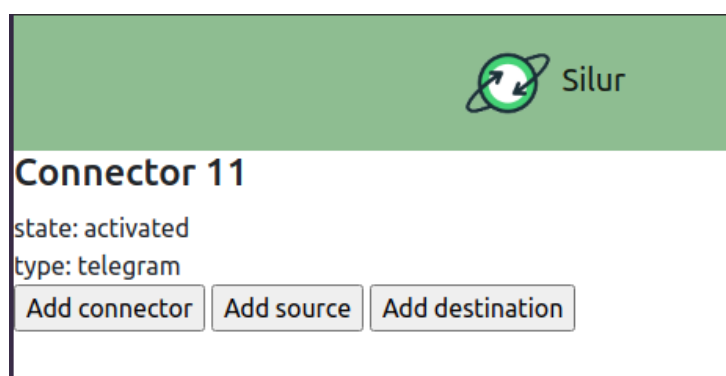


Рисунок 2.22 – Сгенерированная страница

3 ОПИСАНИЕ АВТОРСКОГО ПРОТОКОЛА

3.1 Взаимодействие с сервером

Для использования протокола необходимо соединение с сервером, в рамках курсового проекта была выбрана архитектура, описанная ниже. Для каждого пользователя создается собственное соединение с сервером.

3.2 Протокол передачи данных

Не существует идеального протокола для передачи данных. Многие действительно хорошие протоколы (MTProto и другие) привязаны к определенным приложениям или задачам компаний, что не позволяет произвольно использовать данные протоколы. Протоколы для безопасной передачи данных, которые используются для произвольных задач, слишком обобщенные, что плохо сказывается на безопасности, а также содержат много дополнительной информации, которая не всегда бывает нужна, что сказывается на производительности.

Исходя из этих недостатков можно сделать вывод, что для обеспечения высокопроизводительной безопасной передачи данных необходимо создавать собственный протокол исходя из бизнес-логики приложений или сервисов.

Существует стандартный подход к созданию протокола безопасной передачи данных, состоящий из двух частей: алгоритм работы протокола на этапе аутентификации и алгоритм работы протокола на этапе передачи данных.

Алгоритм работы протокола на этапе аутентификации необходим для того, чтобы предоставить доступ пользователю к ресурсу, при этом следует учитывать, что по умолчанию канал передачи данных небезопасный.

Алгоритм работы протокола на этапе передачи данных учитывает, что уже получены необходимые сведения (ключи) для передачи данных по защищенному каналу связи. В разработку данной части протокола входит проектирование структуры сообщения передачи данных, а также общий алгоритм шифрования данных, алгоритм проверки целостности и подлинности данных. Рассмотрим подробнее суть предлагаемого протокола для безопасной передачи данных в программном обеспечении для агрегации сообщений.

3.2.1 Инициализация

Первый шаг алгоритма аутентификации в клиент-серверном взаимодействии заключается в том, что пользователь подключается к серверу. После чего необходимо создать безопасный канал передачи данных.

Создание безопасного канала между клиентом и сервером заключается в том, чтобы сгенерировать ключи при помощи алгоритма Диффи-Хеллмана [3] на стороне клиента и сервера, после чего отправлять все данные, зашифрованные симметричным алгоритмом AES [4]. На рис. 3.1 представлена стадия инициализации авторского протокола.

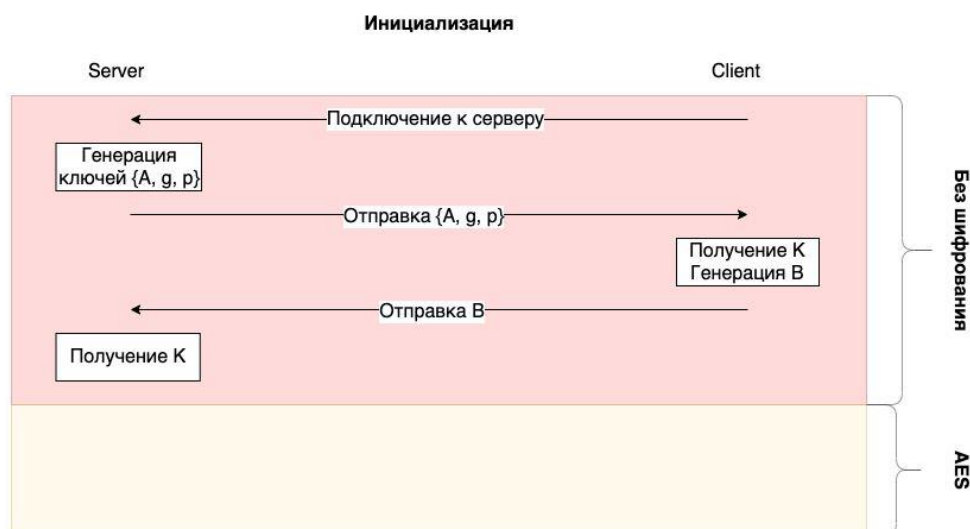


Рисунок 3.1 – Стадия инициализации

На рисунке 3.2 представлена проекция данного протокола в коде со стороны клиента.

```
def __init__(self):  
    self.socket = Socket()  
    self.aes_socket = AESocket(self.socket)
```

Рисунок 3.2 – Клиентская сессия создает Socket и AESocket

На рисунке 3.2 изображено поле `self.socket`, которое устанавливает соединение с сервером, а затем создается `AESocket` из обычного сокета, что приводит к обмену AES ключами через Diffie-Hellman с сервером. Клиентская сессия создается при подключении пользователя к серверу.

3.2.2 Аутентификация

Первый «слой» безопасной передачи данных настроен на стадии инициализации, но он не является достаточно надежным, чтобы передавать критически важные данные: пароль или логин.

Стоит отметить, что следующие «слои» безопасности всегда генерируют новые ключи криптографических алгоритмов, для разных алгоритмов используют разные ключи для повышения криптостойкости.

Следующий шаг заключается в создании дополнительного «слоя» безопасности для передачи логина и пароля. На стороне сервера генерируются новые RSA [5] ключи (e , d , n). Публичный ключ для шифрования принимается от сервера, предварительно расшифрованный алгоритмом AES. Таким образом, создается безопасный канал для передачи данных с клиента на сервер (использование криптографических алгоритмов AES и RSA). Через данный канал передается логин и пароль, после чего сервер отвечает клиенту успехом или неудачей, шифруя ответ только алгоритмом AES. В случае успеха обрабатывается пароль

пользователя аналогичным образом. На рисунке 3.3 представлена стадия идентификации и аутентификации.

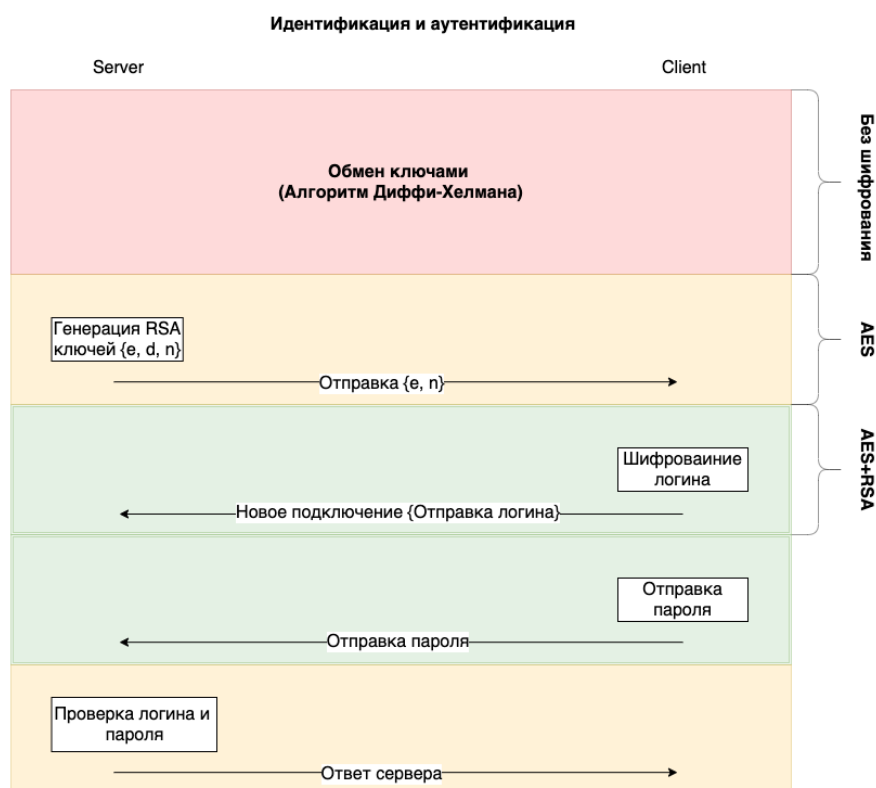


Рисунок 3.3 – Стадия идентификации и аутентификации

В коде это выражено следующим образом (см. рис. 3.4).

```
def sign_in_password(self, login: str, password: str):
    data = {
        'command': 'sign in',
        'method': 'password'
    }
    self.send_command_and_check_answer(data)
    rsa_socket_writer = RSASocketWriter(self.aes_socket)
    self.send_secret_string(login, rsa_socket_writer)
    self.send_secret_string(password, rsa_socket_writer)
    result = self.receive_answer()
    if result['status'] == 'ok':
        self.aes_socket.exchange_aes_key()
    return result
```

Рисунок 3.4 – Идентификация, аутентификация и авторизация

После успешной аутентификации необходимо создать сессионный ключ. Данный ключ генерируется при помощи алгоритма Диффи-Хеллмана поверх существующего AES соединения, в дальнейшем используется только последний ключ (см. рис. 3.5).

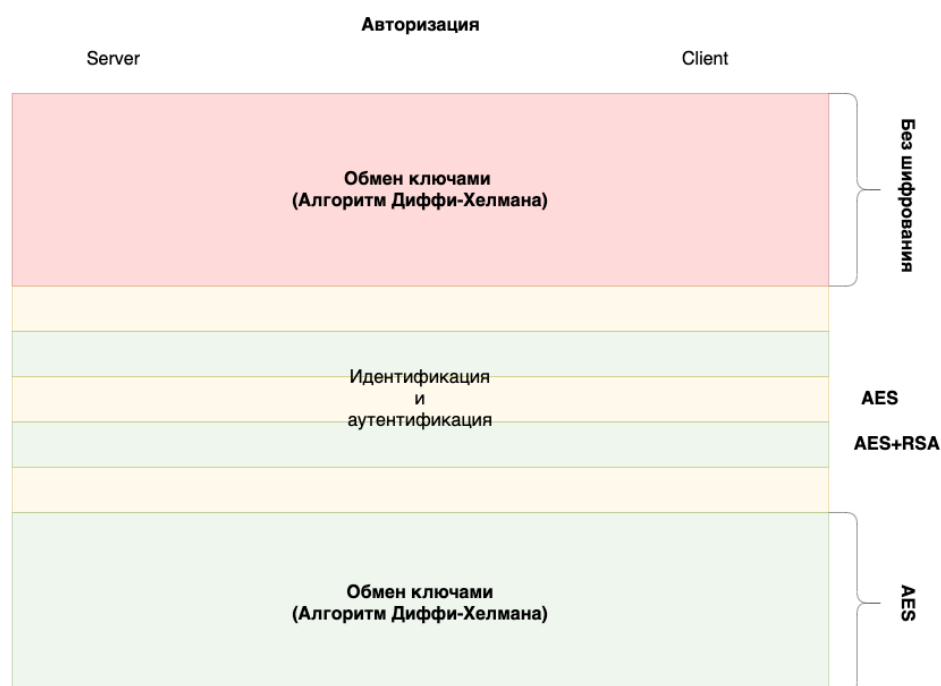


Рисунок 3.5 – Стадия авторизации

```
def exchange_aes_key(self):
    aes_key = get_key_with_dh(self)
    self.aes = AES(aes_key)
```

Рисунок 3.6 – Обмен ключами

Процесс авторизации закончен. Дальнейшая передача данных осуществляется по защищенному каналу.

Для получения доступа к данным необходимо взломать первый алгоритм Диффи-Хеллмана, публичный ключ RSA сервера и второй алгоритм Диффи-Хеллмана. Может показаться, что данные меры безопасности избыточны, но данный протокол рассчитан на то, что через несколько лет появятся вычислительные мощности в совокупности с

современными алгоритмами и искусственным интеллектом, которые будут способны взломать алгоритмы RSA и Диффи-Хеллмана за минимальное время. То есть, через несколько лет данные, которые недостаточно хорошо зашифрованы, будут легкодоступными для злоумышленников. Но при использовании предложенного алгоритма, взлом можно отложить на будущее, что дает гарантию безопасности данных на долгий период.

Частным случаем аутентификации является восстановление сессии, но для этого необходимо её создать и сохранить на сервере и клиенте. Создание сессии доступно только аутентифицированным клиентам, для этого клиент отправляет запрос серверу с командой «создать сессию». После чего клиент и сервер генерируют одинаковый токен (Diffie-Hellman) и сохраняют его. Дополнительно клиент создает подпись и отправляет ключ серверу для проверки подписи при следующем восстановлении сессии. При этом клиент сохраняет только подписанный токен. (см. рис 3.7).

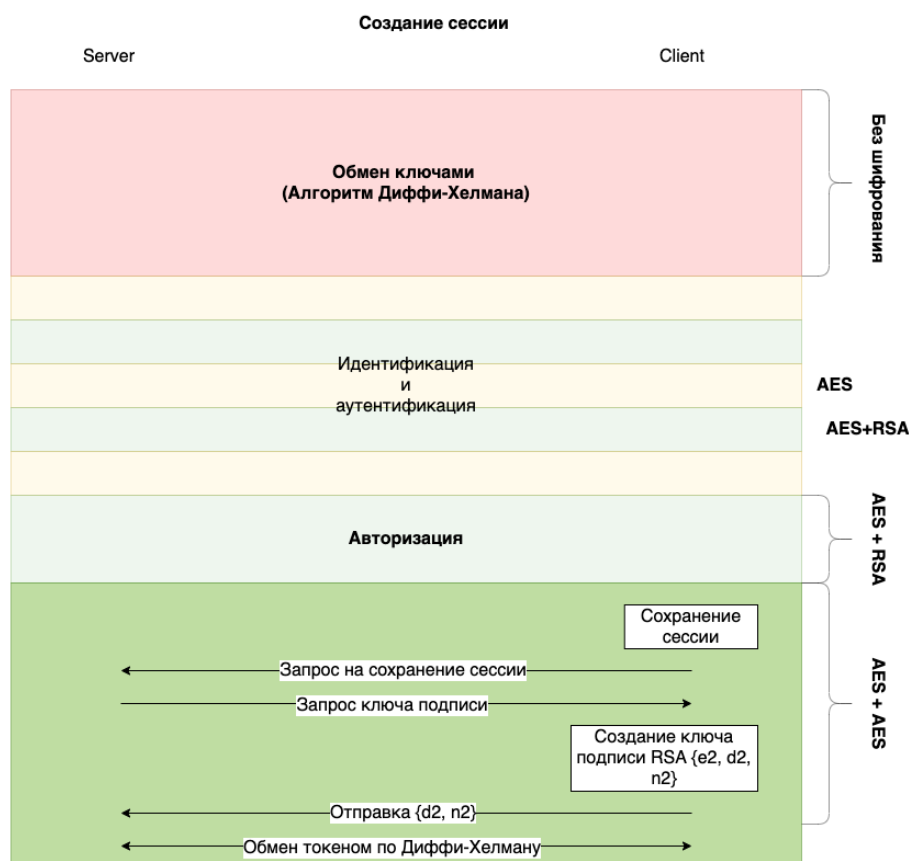


Рисунок 3.7 – Создание сессии

```
def create_session(request, login, password):
    new_session = ClientSession()
    response = new_session.sign_in_password(login, password)
    if response['status'] == 'ok':
        token = new_session.create_token()
        request.session['token'] = token
        request.session['login'] = login
    else:
        raise RuntimeError('Incorrect login or password')
    return new_session
```

Рисунок 3.8 – Создание сессии (токена) в коде

При восстановлении сессии отправляется логин и подписанный токен. Сессия восстанавливается только при успешном сравнении расшифрованного токена (при помощи ключа проверки подписи) пользователя с токеном из базы данных сервера (см. рис. 3.9).

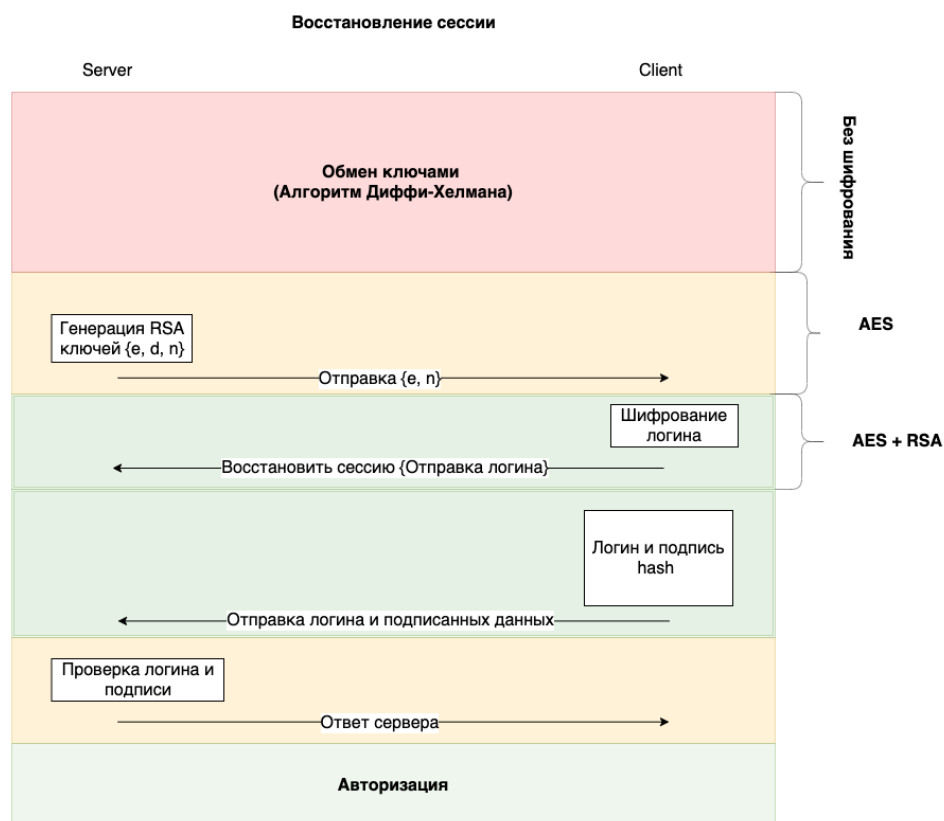


Рисунок 3.9 – Восстановление сессии

```

def restore_session(request):
    restored_session = ClientSession()
    login = request.session['login']
    token = request.session['token']
    restored_session.sign_in_token(login, token)
    return restored_session
  
```

Рисунок 3.10 – Восстановление сессии в коде

3.2.3 Достоинства и недостатки авторского протокола

В основе протокола лежит оригинальная комбинация симметричного алгоритма шифрования AES, протокола Диффи-Хеллмана для обмена 2048-битными RSA-ключами между двумя устройствами и SHA-256 [6].

Достоинства предлагаемого протокола:

1. Первый этап шифрования устанавливается до аутентификации, что позволяет полностью скрыть информацию о пользователе.

2. Аутентификация происходит по защищенному каналу связи.

3. Для шифрования сообщений используется симметричное шифрование.

4. Авторизацию пользователи могут выполнить, как при помощи пароля, так и без пароля при помощи сессионных токенов.

Недостатки.

Недостатки протокола будут выявлены в процессе программной реализации и тестирования программного обеспечения в реальных условиях.

Также стоит заметить, что протокол привязан к приложению, в будущем структура приложения будет развиваться и увеличиваться, что скажется на структуре и алгоритмах шифрования сообщений, а также алгоритме аутентификации.

ВЫВОДЫ

Во время выполнения курсового проекта были получены навыки в области проектирования и реализации клиентской части клиент-серверного приложения.

На базе данных знаний была разработан авторский протокол передачи данных, используемый в системе агрегирования. Преимущества которого заключается в повышенной криптостойкости за счет уникальной комбинации существующих криптографических алгоритмов, а также возможность повторной авторизации пользователя без пароля.

Результатом является клиентское приложение – сайт. При этом в реализации были учтены конфиденциальность пользователя, безопасность передачи данных, а также возможность дальнейшего развития – добавление большинства стилей и дальнейшая поддержка сервера. Авторский протокол передачи данных, идеально подходит для данной задачи и не добавляет дополнительной нагрузки, как в случае использования других протоколов передачи данных.

В дальнейшей данный проект можно расширить другими возможностями: добавление анимации на сайт, поддержка мобильных устройств, создание множества стилей, добавление темной и светлой темы.

ПЕРЕЧЕНЬ ССЫЛОК

1. Что такое веб-ресурс [электронный ресурс]. – Режим доступа: <https://php.zone/post/veb-resurs-hto-eto-takoe> (дата обращения 23.12.2021).
2. The Django template language [электронный ресурс]. – Режим доступа: <https://docs.djangoproject.com/en/4.0/ref/templates/language/> (дата обращения 23.12.2021).
3. Diffie-Hellman Key Agreement Method [электронный ресурс]. – Режим доступа: <https://datatracker.ietf.org/doc/html/rfc2631> (дата обращения 05.11.2021).
4. The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model [электронный ресурс]. – Режим доступа: <https://datatracker.ietf.org/doc/html/rfc3826> (дата обращения 05.11.2021).
5. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptograph Specifications Version 2.1 [электронный ресурс]. – Режим доступа: <https://datatracker.ietf.org/doc/html/rfc3447> (дата обращения 05.11.2021).
6. US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF) [электронный ресурс]. – Режим доступа: <https://datatracker.ietf.org/doc/html/rfc6234> (дата обращения 05.11.2021).

ПРИЛОЖЕНИЕ А. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР
ГОУ ВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»

ТЕХНИЧЕСКОЕ ЗАДАНИЕ
К КУРСОВОЙ РАБОТЕ
ПО КУРСУ: «Системное программирование»
НА ТЕМУ: «Объектно-ориентированная реализация программной
системы SILUR в среде Linux»

Выдано:
студенту группы ПИ-186
Моргунову А. Г.

Руководитель:
Коломойцева И.А.
Филипишин Д.А.
Ногтев Е.А.

Донецк, 2021

1 Основание для разработки (основанием для разработки является задание на курсовую работу, выданное кафедрой программной инженерии)

2 Цель разработки (целью разработки является Объектно-ориентированная реализация клиентской части программной системы SILUR)

3 Требования к программе:

программа должна содержать объектно-ориентированную модель заданной предметной области с использованием технологии Django.

– приложение должно быть многопоточным: поток для управление главным циклом агрегатора и поток для работы с клиентскими командами (модуль socket), для сложных вычислительных работ необходимо произвести анализ и по возможности распараллелить процессы.

4 Требования к программной документации:

- пояснительная записка;
- руководство пользователя.

5 Условие задачи

Создание клиентской части проекта SILUR.

1. Создать клиентскую часть агрегирующей системы, которая позволяет пересылать сообщения из Telegram каналов в другие Telegram каналы.

2. Обеспечить связь с серверной частью системы. Для взаимодействия с сервером предоставить протокол безопасной передачи данных.

Клиент взаимодействует с сервером при помощи API протокола, т.е. командам, например, создать связь агрегации, удалить связь агрегации, добавить ключевые слова для агрегации, просмотреть список агрегации и т.д. На команды клиента сервер отвечает сообщениями, которые могут

содержать код ошибки, а также полезную нагрузку (информацию), например, список агрегирующих каналов, историю агрегации и т.д.

6 Этапы разработки

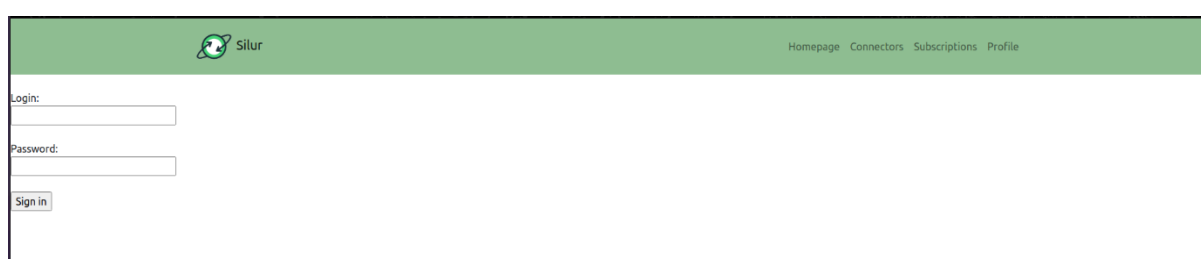
№ этапа	Наименование этапа	Срок выполнения
1.	Выдача задания, составление ТЗ и его утверждение	2 недели
2.	Техническое проектирование. Разработка алгоритмов	2-4 недели
3.	Рабочее проектирование. Определение структур данных	4-5 недели
4.	Написание программы	5-9 недели
5.	Отладка программы	10-12 недели
6.	Написание пояснительной записки	13 неделя
7.	Защита курсовой работы	13-14 недели

ПРИЛОЖЕНИЕ Б. ЭКРАННЫЕ ФОРМЫ



The screenshot shows the user profile page of the Silur application. At the top, there is a green header bar with the Silur logo on the left and navigation links (Homepage, Connectors, Subscriptions, Profile) on the right. Below the header, a blue bar displays the user's login: 'Login: ArseniiM'. On the left side, there are two buttons: 'Sign in' and 'Register'.

Рисунок Б.1 – Профиль



The screenshot shows the login page of the Silur application. It features a green header bar with the Silur logo and navigation links. Below the header, there are two input fields labeled 'Login:' and 'Password:'. A 'Sign in' button is located below the password field.

Рисунок Б.2 – Авторизация



The screenshot shows the registration page of the Silur application. It features a green header bar with the Silur logo and navigation links. Below the header, there are four input fields labeled 'First name:', 'Second name:', 'Login:', and 'Password:'. A 'Register' button is located below the password field.

Рисунок Б.3 – Регистрация

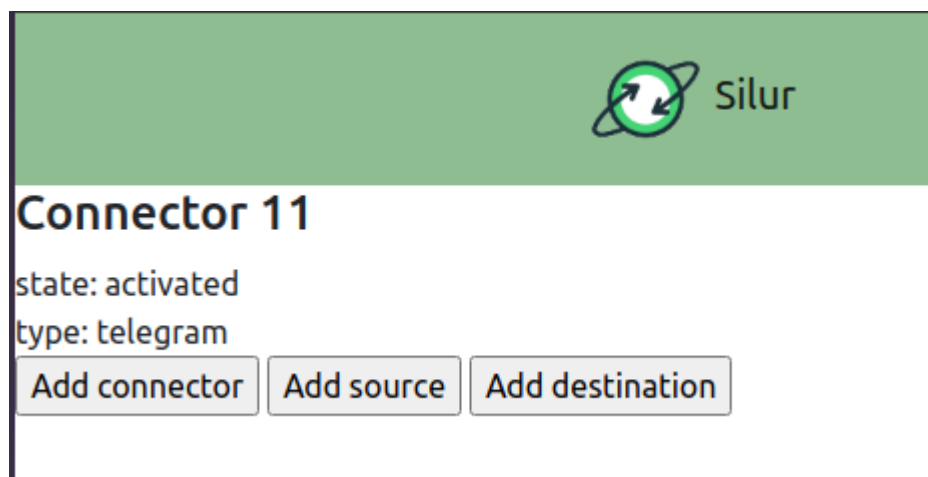


Рисунок Б.4 – Соединения

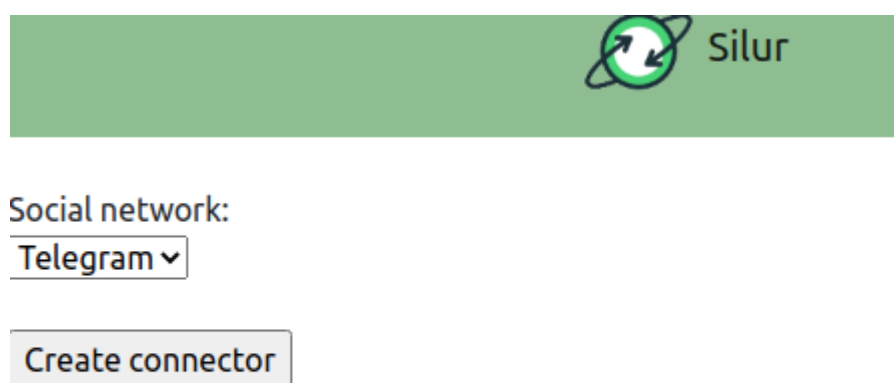


Рисунок Б.5 – Создание соединения


 Silur
Connector: ▾
Channel name:

Рисунок Б.6 – Создание источника


 Silur
Connector: ▾
Channel name:

Рисунок Б.7 – Создание приёмника

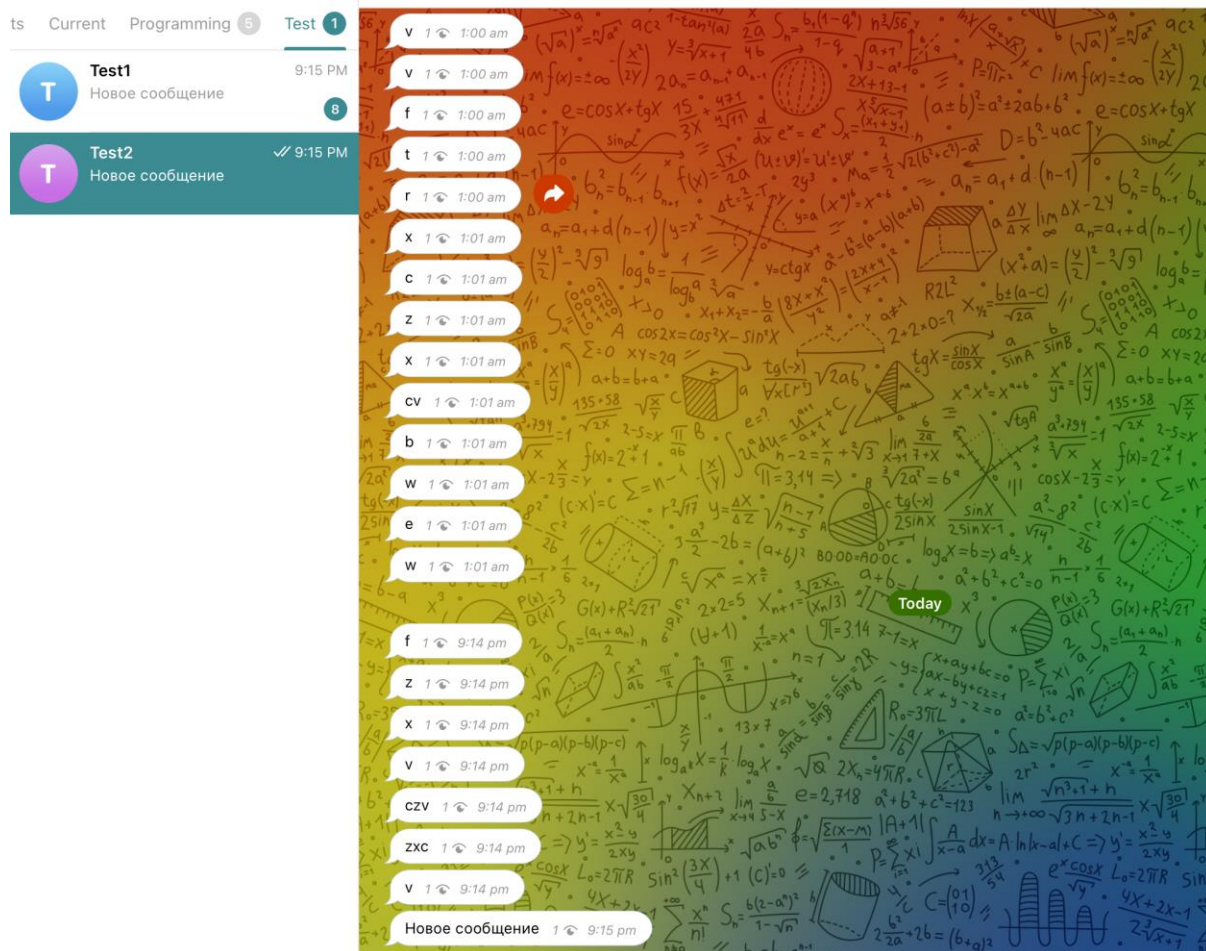


Рисунок Б.8 – Отправка сообщения в канал-источник

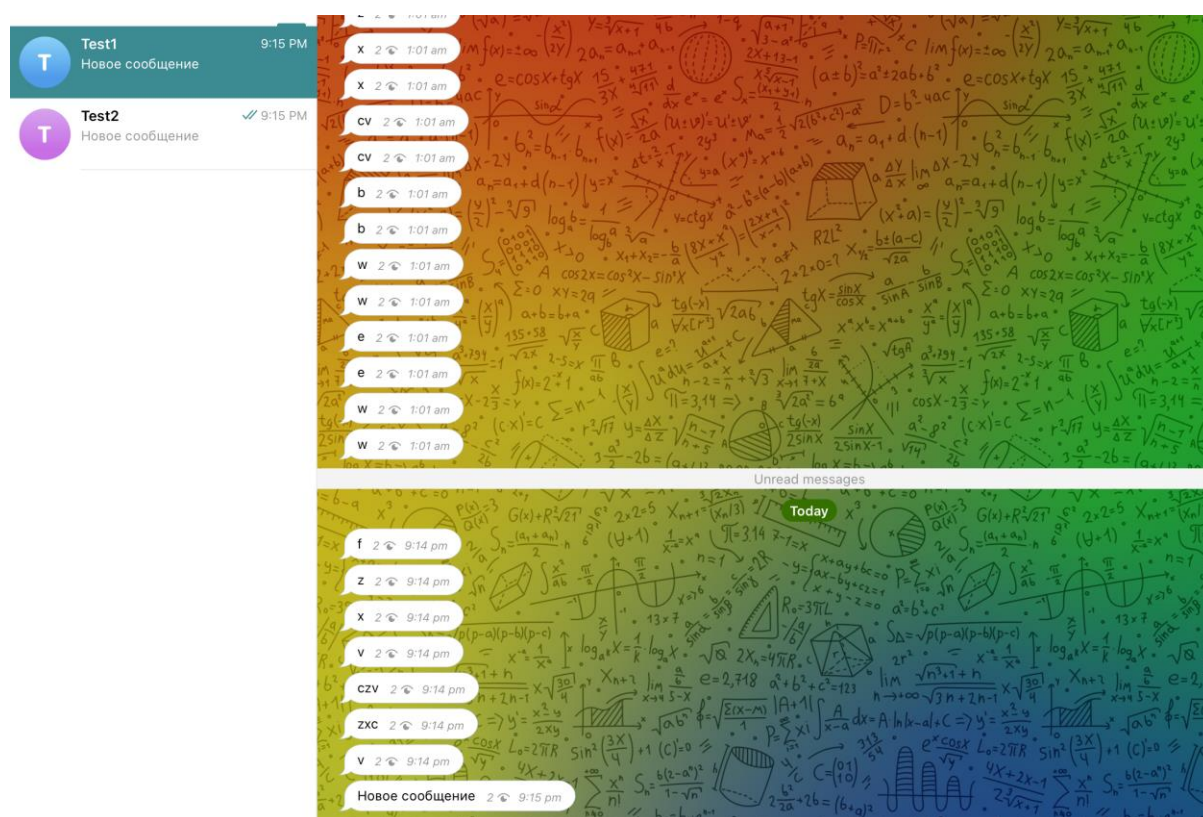


Рисунок Б.9 – Агрегация сообщения в канал-приёмник

ПРИЛОЖЕНИЕ В. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для начала работы необходимо зайти на страницу Profile по кнопке в навигационном меню. Далее нужно зарегистрироваться или войти в уже существующую учетную запись. Для регистрации нужно нажать на кнопку Register, ввести имя, фамилию, логин и пароль, нажать на кнопку Register. Для входа в существующую учетную запись нужно нажать на Sign in, ввести логин, пароль, нажать на кнопку Sign in.

После авторизации можно приступать к работе с приложением. Для начала нужно создать соединение. Для этого нужно зайти в Connectors в навигационном меню, нажать на кнопку Add connectors. Выбрать тип соединения, нажать Create connector, ввести номер телефона, нажать Send, ввести код аутентификации, нажать Send.

Теперь нужно создать источник и получателя. Для создания источника нужно нажать Add source, выбрать соединение, ввести имя канала в формате @ChannelName, нажать на кнопку Save source.

Для создания получателя нужно нажать Add destination, выбрать соединение, ввести имя канала в формате @ChannelName, нажать на кнопку Save destination.

Теперь нужно создать подписку. Для этого нужно перейти в Subscriptions в навигационном меню, нажать на кнопку Add subscription, выбрать источник, выбрать получателя, нажать кнопку Save relation.

ПРИЛОЖЕНИЕ Г. ЛИСТИНГ КОДА

Utl.py

```
from django.urls import path, re_path
```

```
from . import views
```

```
urlpatterns = [
    path('silur/', views.homepage, name='homepage'),
    path('silur/connectors/', views.connectors, name='connectors'),
    path('silur/myprofile/', views.myprofile, name='myprofile'),
    path('silur/subscriptions/', views.subscriptions, name='subscriptions'),
    path('silur/registration/', views.registration, name='registration'),
    path('silur/sign_in/', views.sign_in, name='sign_in'),
    path('silur/create_source/', views.create_source, name='create_source'),
    path('silur/create_destination/', views.create_destination, name='create_destination'),
    # path('silur/relation/', views.relation, name='relation'),
    path('silur/create_connector/', views.create_connector, name='create_connector'),
    path('silur/create_relation/', views.create_relation, name='create_relation'),
    path("", views.homepage),
    path('silur/<str:data_info>', views.user_data_input, name='data_input'),
]
```

Forms.py

```
from django import forms
```

```
class AddSubscriptionForm(forms.Form):
    source = forms.IntegerField(label='Source')
    destination = forms.IntegerField(label='Destination')
```

```
class ConnectorForm(forms.Form):
    social_network_types = [('telegram', 'Telegram')]
    social_network = forms.CharField(max_length=200,
    widget=forms.Select(choices=social_network_types,
    label='Social network')
```

```
class SourceForm(forms.Form):
    connector_id = forms.IntegerField(label='Connector')
    channel_name = forms.CharField(max_length=200, label='Channel name')
```

```
class DestinationForm(forms.Form):
    connector_id = forms.IntegerField(label='Connector')
    channel_name = forms.CharField(max_length=200, label='Channel name')
```

```
class RegistrationForm(forms.Form):
    first_name = forms.CharField(max_length=200, label='First name')
    second_name = forms.CharField(max_length=200, label='Second name')
    login = forms.CharField(max_length=200, label='Login')
    password = forms.CharField(widget=forms.PasswordInput(), label='Password')
```

```
class SignInForm(forms.Form):
    login = forms.CharField(max_length=200, label='Login')
    password = forms.CharField(widget=forms.PasswordInput(), label='Password')
```

```
class DataFromUser(forms.Form):
    data = forms.CharField(max_length=100)
```

```
class RelationForm(forms.Form):
    source = forms.IntegerField(label='Source')
    destination = forms.IntegerField(label='Destination')
```

```
label='Constraint type')
```

Models.py

```
from django.db import models
```

```
class Connector(models.Model):
    type = models.CharField(max_length=200)
    phone = models.CharField(max_length=200)
    state = models.CharField(max_length=200)
```

```
class Source(models.Model):
    receiver_id = models.IntegerField()
    chat_id = models.IntegerField()
    # channel_name = models.CharField(max_length=200)
```

```
class Destination(models.Model):
    sender_id = models.IntegerField()
    chat_id = models.IntegerField()
    # channel_name = models.CharField(max_length=200)
```

```
class Relation(models.Model):
    source_id = models.IntegerField()
    destination_id = models.IntegerField()
```

views.py

```
import random
```

```
from django.shortcuts import render, redirect
from .forms import *
from .SilurCryptoSockets.ClientSession import ClientSession
import json
from .models import *
from django.http import HttpResponseRedirect, HttpResponse
from .SilurCryptoSockets.AESSocket import AESSocket
```

```
# session = ClientSession()
# session.sign_in_password('Login', 'Password')
sessions = {}
```

```
def is_authorized(request):
    if request.session.session_key is not None:
        return True
    else:
        return False
```

```
def create_session(request, login, password):
    new_session = ClientSession()
    response = new_session.sign_in_password(login, password)
    if response['status'] == 'ok':
        token = new_session.create_token()
        request.session['token'] = token
        request.session['login'] = login
    else:
        raise RuntimeError('Incorrect login or password')
    return new_session
```

```
def restore_session(request):
    restored_session = ClientSession()
    login = request.session['login']
    token = request.session['token']
    restored_session.sign_in_token(login, token)
    return restored_session
```

```
def homepage(request):
    response = render(request, 'silur/homepage.html')
    return response
```

```
def connectors(request):
    session = restore_session(request)
    if request.method == 'POST':
        method = request.POST['method']
        method, obj = method.split(' ')
        if method == 'Add':
            session.sign_out()
            if obj == 'connector':
                return redirect('create_connector')
            elif obj == 'destination':
                return redirect('create_destination')
            elif obj == 'source':
                return redirect('create_source')
        # print(method)
    server_response = session.get_connectors()
    connectors = []
    # server_response = {'connectors': [{'id': 8, 'meta_info': {'state': 'activated'}, 'type': 'telegram'}],
    #                   'description': 'The server sent your connectors', 'status': 'ok'}
    server_connectors = server_response['connectors']
    for connector_info in server_connectors:
        connector = Connector(connector_info['id'], type=connector_info['type'],
        state=connector_info['meta_info']['state'])
    connectors.append(connector)
    data = {
        'connectors': connectors
    }
    session.sign_out()
    return render(request, 'silur/connectors.html', data)
```

```
def myprofile(request):
    if request.method == 'POST':
        print(request.POST)
        method = request.POST['method']
        if method == 'Sign in':
            return HttpResponseRedirect('/silur/sign_in')
        elif method == 'Register':
            return HttpResponseRedirect('/silur/registration')
        elif method == 'Sign out':
            pass
    request.session['token']))

    login = request.session.get('login', 'Guest')
    # if login == 'Guest':
    buttons = ['Sign in', 'Register']
    # else:
    #     buttons = ['Sign out']
    data = {
        'login': login,
        'token': request.session.get('token', 0),
        'buttons': buttons
    }
    return render(request, 'silur/myprofile.html', data)
```

```
def subscriptions(request):
    session = restore_session(request)
    if request.method == 'POST':
        print(request.POST)
        method = request.POST['method']
        method, obj = method.split(' ')
        if method == 'Add':
            session.sign_out()
            return redirect('create_relation')
    senders = session.get_senders()
    server_destinations = []
    for sender in senders['senders']:
        server_response_destinations = session.get_destinations(sender['id'])
        server_destinations.append(server_response_destinations['destinations'])
```

```
destinations_with_sources = []
for destination_list in server_destinations:
    for destination in destination_list:
        destination_obj = Destination(destination['id'], chat_id=destination['meta_info']['chat_id'],
        sender_id=destination['sender_id'])
        # print(f'Destination ID: {destination["id"]}')
        server_response = session.get_relations(destination['id'])
        # server_response = server_response_relations.pop()
        relations = server_response['relations']
        # print(relations)
        server_receivers = session.get_receivers()
        for receiver in server_receivers['receivers']:
            server_sources = session.get_sources(receiver['id'])
            sources = []
            for relation in relations:
                server_source = find_in_dictionary_set(server_sources['sources'], 'id', relation['source_id'])
                # print(f'Server source: {server_source}')
                source = Source(server_source['id'], chat_id=server_source['meta_info']['chat_id'],
                receiver_id=server_source['receiver_id'])
            sources.append(source)
        destinations_with_sources.append((destination_obj, sources))
# print(destinations_with_sources)
data = {
    'destinations': destinations_with_sources,
}
session.sign_out()
return render(request, 'silur/subscriptions.html', data)
```

```
def registration(request):
    error = ""
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            session = ClientSession()
```

```

result = session.register(data['login'], data['password'],
                          data['first_name'], data['second_name'])
# session.sign_out()
if result['status'] == 'ok':
    try:
        session = create_session(request, data['login'], data['password'])
        return redirect('myprofile')
    except RuntimeError as re:
        error = re.args[0]
        session.sign_out()
    else:
        error = "Cannot create user"
        session.sign_out()
else:
    form = RegistrationForm()
data = {
    'form': form,
    'error': error
}
return render(request, 'silur/registration.html', data)

def sign_in(request):
    error = ""
    if request.method == 'POST':
        form = SignInForm(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            try:
                session = create_session(request, data['login'], data['password'])
                session.sign_out()
                # sessions[data['login']] = session
                return redirect('myprofile')
            except RuntimeError as re:
                error = re.args[0]
                print(error)
            else:
                form = SignInForm()
        data = {
            'form': form,
            'button_name': 'Sign in',
            'error': error
        }
    return render(request, 'silur/form_template.html', data)

def user_data_input(request, data_info):
    if request.method == 'POST':
        form = DataFromUser(request.POST)
        # print(form.data)
        # print(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            session = sessions[request.session.session_key]
            response = session.send_user_data(data['data'])
            if response['status'] == 'need value':
                return HttpResponseRedirect('/silur/' + response['value name'] + '/')
            else:
                session.sign_out()
                del sessions[request.session.session_key]
                return HttpResponseRedirect('/silur/connectors/')
        else:
            form = DataFromUser()
    data = {
        'form': form,
        'data_info': data_info
    }
    return render(request, 'silur/user_data_input.html', data)

def create_destination(request):
    session = restore_session(request)
    if request.method == 'POST':
        form = DestinationForm(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            session.create_destination(data['connector_id'], data['channel_name'])
            session.sign_out()
            return HttpResponseRedirect('/silur/connectors/')
        else:
            form = DestinationForm()
    server_connectors = session.get_connectors()
    # server_connectors = {'connectors': [{'id': 8, 'meta_info': {'state': 'activated'}, 'type': 'telegram'}],
    #                      'description': 'The server sent your connectors', 'status': 'ok'}
    connectors_id = []
    for connector in server_connectors['connectors']:
        connectors_id.append((connector['id'], "Connector " + str(connector['id'])))

    data = {
        'form': form,
        'button_name': 'Save destination',
        'connectors': connectors_id
    }
    return render(request, 'silur/create_destination.html', data)

def create_relation(request):
    session = restore_session(request)
    if request.method == 'POST':
        form = RelationForm(request.POST)
        print(form.data)
        # print(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            session.create_relation(data['source'], data['destination'])
            session.sign_out()
            return redirect('subscriptions')
        else:
            form = RelationForm()

    receivers = session.get_receivers()
    sources = []
    for receiver in receivers['receivers']:
        server_response_sources = session.get_sources(receiver['id'])
        sources.append(server_response_sources['sources'])
    sources_id = []
    for source_list in sources:
        for source in source_list:
            sources_id.append((source['id'], "Source " + str(source['id'])))

    senders = session.get_senders()
    destinations = []
    for sender in senders['senders']:
        server_response_destinations = session.get_destinations(sender['id'])
        destinations.append(server_response_destinations['destinations'])
    destinations_id = []
    for destination_list in destinations:
        for destination in destination_list:
            destinations_id.append((destination['id'], "Destination " + str(destination['id'])))

    data = {
        'form': form,
        'sources': sources_id,
        'destinations': destinations_id,
        'button_name': 'Save relation',
    }
    session.sign_out()
    return render(request, 'silur/create_relation.html', data)

def create_source(request):
    session = restore_session(request)
    if request.method == 'POST':
        form = SourceForm(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            session.create_source(data['connector_id'], data['channel_name'])
            session.sign_out()
            return HttpResponseRedirect('/silur/connectors/')
        else:
            form = SourceForm()
    server_connectors = session.get_connectors()
    # server_connectors = {'connectors': [{'id': 8, 'meta_info': {'state': 'activated'}, 'type': 'telegram'}],
    #                      'description': 'The server sent your connectors', 'status': 'ok'}
    connectors_id = []
    for connector in server_connectors['connectors']:
        connectors_id.append((connector['id'], "Connector " + str(connector['id'])))

    data = {
        'form': form,
        'button_name': 'Save source',
        'connectors': connectors_id
    }
    return render(request, 'silur/create_source.html', data)

def create_connector(request):
    if request.method == 'POST':
        form = ConnectorForm(request.POST)
        print(form.data)
        print(request.POST)
        if form.is_valid():
            session = restore_session(request)
            sessions[request.session.session_key] = session
            response = session.create_connector_command()
            if response['status'] == 'need value':
                return HttpResponseRedirect('/silur/' + response['value name'] + '/')
            else:
                session.sign_out()
                del sessions[request.session.session_key]
        else:
            form = ConnectorForm()
    data = {
        'form': form,
        'button_name': 'Create connector'
    }
    return render(request, 'silur/form_template.html', data)

wrapper.html
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Silur</title>
    <link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">
    {# <link rel="stylesheet" href="{% static 'css/bootstrap-grid.css' %}">#}

    {# <link rel="stylesheet" href="{% static 'css/MyCss.css' %}">#}

    <script rel="{% static 'js/bootstrap.bundle.js' %}"></script>
    <script rel="{% static 'js/bootstrap.js' %}"></script>
    {% block addtohead %}{% endblock %}
</head>
<body>
    <header class="d-flex justify-content-between navbar navbar-light navbar-expand"
    style="background-color: darkseagreen">
    {# <nav class="d-flex justify-content-between navbar navbar-light navbar-expand"
    style="background-color: darkseagreen">#}
        <div class="container">
            <a class="navbar-brand" href="/silur/">
                
                Silur
            </a>
            <div class="d-inline-flex justify-content-end container collapse navbar-collapse">
                <ul class="navbar-nav">
                    <li class="nav-item">
                        <a class="nav-link {% block homepage_active %}{% endblock %}" {% block
                        homepage_aria_current %}{% endblock %} href="{% url 'homepage' %}">Homepage</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link {% block connectors_active %}{% endblock %}" {% block
                        connectors_aria_current %}{% endblock %} href="{% url 'connectors' %}">Connectors</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link {% block subscriptions_active %}{% endblock %}" {% block
                        subscriptions_aria_current %}{% endblock %} href="{% url 'subscriptions' %}">Subscriptions</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link {% block myprofile_active %}{% endblock %}" {% block
                        myprofile_aria_current %}{% endblock %} href="{% url 'myprofile' %}">Profile</a>
                    </li>
                </ul>
            </div>
        </div>
    {# </nav>#}
    </header>

    {% block main %}{% endblock %}

</body>
</html>

Deropdown_snippet.html
{#variables:#}

```

```

{# - dropdown_field#}
{# - choices#}

<label for="{{ dropdown_field.id_for_label }}">
  {{ dropdown_field.label_tag }}
</label>
<select name="{{ dropdown_field.html_name }}" id="{{ dropdown_field.id_for_label }}">
  {% for choice_value, choice_name in choices %}
    <option value="{{ choice_value }}">{{ choice_name }}</option>
  {% endfor %}
</select><br>

Common_field_snippet.html
{#variables:#}
{# - dropdown_field#}
{# - choices#}

<label for="{{ dropdown_field.id_for_label }}">
  {{ dropdown_field.label_tag }}
</label>
<select name="{{ dropdown_field.html_name }}" id="{{ dropdown_field.id_for_label }}">
  {% for choice_value, choice_name in choices %}
    <option value="{{ choice_value }}">{{ choice_name }}</option>
  {% endfor %}
</select><br>

Homepage.html
{% extends 'silur/base/wrapper.html' %}

{% block homepage_active %}active{% endblock %}
{% block homepage_aria-current %}aria-current="page"{% endblock %}

{% block main %}
  <div class="container">
    <div class="row justify-content-center">
      <div class="col-8" style="background-color: gold">
        <h1>Welcome to aggregator Silur!</h1>
      </div>
    </div>
  </div>
{% endblock %}

Myprofile.html
{% extends 'silur/base/wrapper.html' %}

{% block myprofile_active %}active{% endblock %}
{% block myprofile_aria-current %}aria-current="page"{% endblock %}

{% block main %}
  <div class="container">
    <div class="row justify-content-center">
      <div class="col-10" style="background-color: lightskyblue">
        <div class="container">
          <div class="row">
            <div class="col">
              Login: {{ login }}
            </div>
          </div>
          <div class="row">
            <div class="col">
              <p>Token: {{ token }}</p>#}
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
  <form method="post">
  {% csrf_token %}
  {% for button in buttons %}
    <input type="submit" name="method" value="{{ button }}">
  {% endfor %}
  </form>
{% endblock %}

Create_relation.html
{% extends 'silur/base/wrapper.html' %}

{% block main %}
  <form method="post">
  {% csrf_token %}
  {% include 'silur/base/dropdown_snippet.html' with dropdown_field=form.source
choices=sources %}
  <br>
  {% include 'silur/base/dropdown_snippet.html' with dropdown_field=form.destination
choices=destinations %}

  <input type="submit" value="{{ button }}">
  </form>
{% endblock %}

Create_source.html
{% extends 'silur/base/wrapper.html' %}

{% block main %}
  <form method="post">
  {% csrf_token %}

  {% include 'silur/base/dropdown_snippet.html' with dropdown_field=form.connector_id
choices=connectors %}

  {% include 'silur/base/common_field_snippet.html' with form_field=form.channel_name %}
  <br>
  <input type="submit" value="{{ button }}">
  </form>
{% endblock %}

Create_destination.html
{% extends 'silur/base/wrapper.html' %}

{% block main %}
  <form method="post">
  {% csrf_token %}

  {% include 'silur/base/dropdown_snippet.html' with dropdown_field=form.connector_id
choices=connectors %}

  {% include 'silur/base/common_field_snippet.html' with form_field=form.channel_name %}
  <br>

```

```

    <input type="submit" value="{{ button }}">
  </form>
{% endblock %}

Form_template.html
{% extends 'silur/base/wrapper.html' %}

{% block main %}
  {{ error }}
  <form method="post">
  {% csrf_token %}
  {% for field in form %}
    {{ field.errors }}<br>
    {{ field.label_tag }}<br>
    {{ field }}<br>
  {% endfor %}
  <br>
  <input type="submit" value="{{ button }}">
  </form>
{% endblock %}

subscription.html
{% extends 'silur/base/wrapper.html' %}

{% block subscriptions_active %}active{% endblock %}
{% block subscriptions_aria-current %}aria-current="page"{% endblock %}

{% block main %}

  {% for destination, sources in destinations %}
    <h4>Destination {{ destination.id }} ({{ destination.chat_id }})</h4>
    <h4>Sender: {{ destination.sender_id }}</h4>
    {% for source in sources %}
      Source {{ source.id }} ({{ source.chat_id }})<br>
      Receiver: {{ source.receiver_id }}<br>
    {% endfor %}
  {% endfor %}
  <form method="post">
  {% csrf_token %}
  <input type="submit" name="method" value="Add subscription">
  </form>
{% endblock %}

sign_in.html
{% extends 'silur/base/wrapper.html' %}

{% block main %}
  <form method="post">
  {% csrf_token %}
  {% for field in form %}
    {{ field.errors }}<br>
    {{ field.label_tag }}<br>
    {{ field }}<br>
  {% endfor %}
  <input type="submit" value="Sign in">
  </form>
{% endblock %}

user_data_input.html
{% extends 'silur/base/wrapper.html' %}

{% block main %}
  <form method="post">
  {% csrf_token %}
  {% for field in form %}
    {{ field.errors }}<br>
    {{ data_info }}<br>
    {{ field }}<br>
  {% endfor %}
  <input type="submit" value="Send">
  </form>
{% endblock %}

registration.html
{% extends 'silur/base/wrapper.html' %}

{% block main %}
  {{ error }}
  <form method="post">
  {% csrf_token %}
  {% for field in form %}
    {{ field.errors }}<br>
    {{ field.label_tag }}<br>
    {{ field }}<br>
  {% endfor %}
  <input type="submit" value="Register">
  </form>
{% endblock %}

connectors.html
{% extends 'silur/base/wrapper.html' %}

{% block connectors_active %}active{% endblock %}
{% block connectors_aria-current %}aria-current="page"{% endblock %}

{% block main %}

  {% for connector in connectors %}
    <form method="post">
    {% csrf_token %}
    <h4>Connector {{ connector.id }}</h4>
    state: {{ connector.state }}<br>
    type: {{ connector.type }}<br>
    # <input type="submit" name="method" value="Delete connector {{ connector.id }}">#}
    </form>
  {% endfor %}
  <form method="post">
  {% csrf_token %}
  <input type="submit" name="method" value="Add connector">
  <input type="submit" name="method" value="Add source">
  <input type="submit" name="method" value="Add destination">
  </form>
{% endblock %}

```