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»

Выполнил:

ст. гр. ПИ–18б

Моргунов А.Г.

Руководитель:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_кафедры \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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, сокеты, методы, алгоритмы взаимодействия с сервером, возможности Django, криптографические алгоритмы.

Объект исследования – клиентское приложение – сайт для агрегационной системы.

Результаты работы – сайт, написанный при помощи цDjango 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 РАЗРАБОТРА 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.

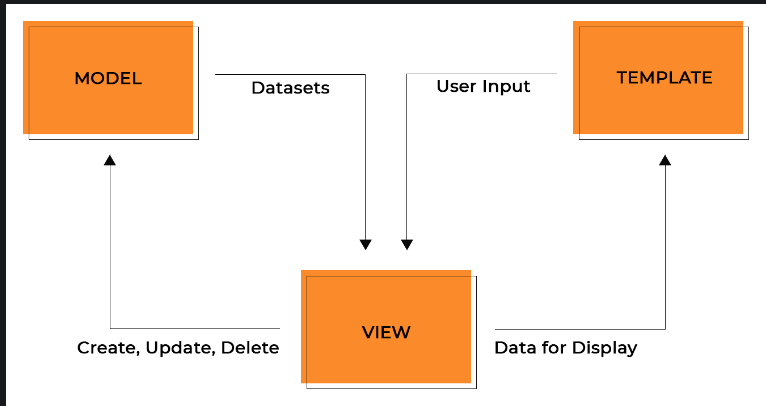
Для настройки агрегатора клиент использует API сервера, которое предоставляется для клиентов. Клиенты могут создавать новые каналы-источники, каналы-приемники и устанавливать отношение между ними.

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 исползует 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.

- Moдуль сокетов.

1.2.1 Упрощенная схема приложения

Система содержит 2 основных модуля:

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

Упрощенная схема приложения приведена на рисунке 1.2.

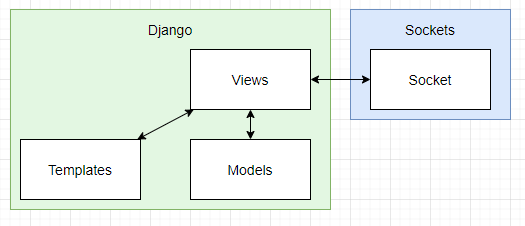


Рисунок 1.2 – Упрощенная схема приложения

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)

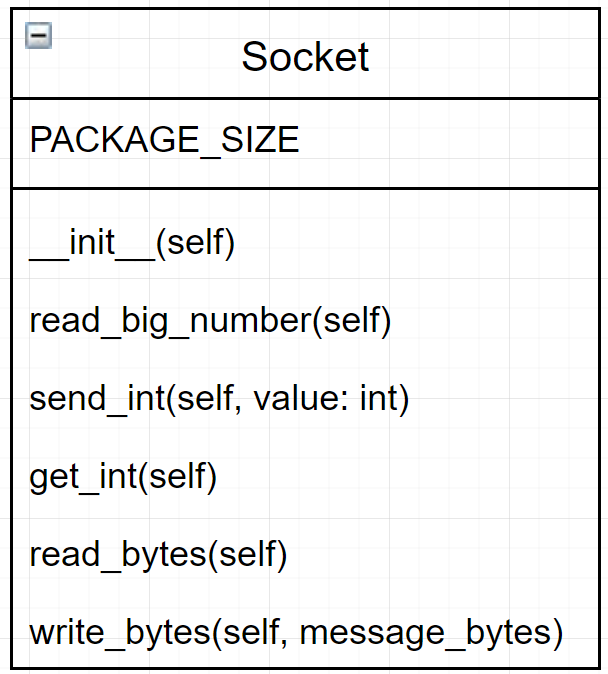


Рисунок 1.3 – Класс Socket

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 происходи 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 выводится пользователю. Каждая форма описывается в виде класса. Структура классов форм представлена на рисунке 1.4.

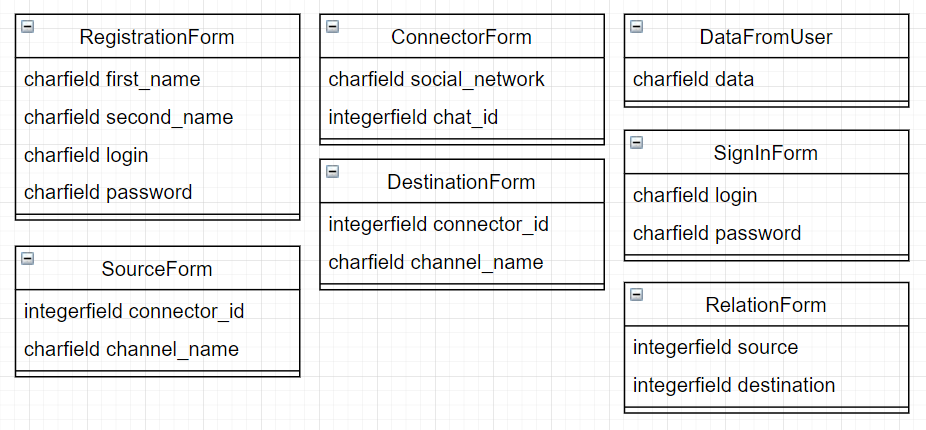


Рисунок 1.4 – Диаграмма классов форм

На диаграмме классов видно, что классы не связанны 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 выводится пользователю, описывается отдельным классом. Структура модуля Models приведена на рисунке 1.5

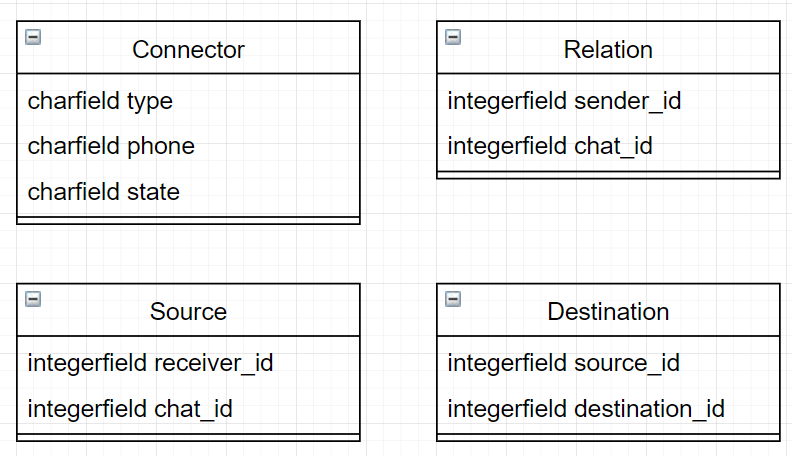


Рисунок 1.5 – Диаграмма классов модуля Models

На диаграмме классов видно, что классы не связанны 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 РАЗРАБОТРА 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 черезвычайно 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



Рисунок 2.1 – Реализация метода write\_bytes

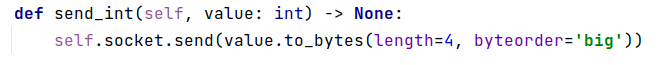


Рисунок 2.2 – Реализация метода send\_int

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).

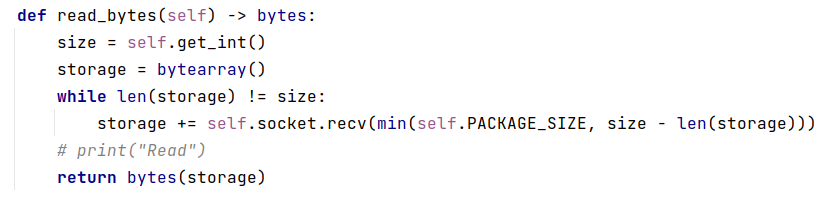
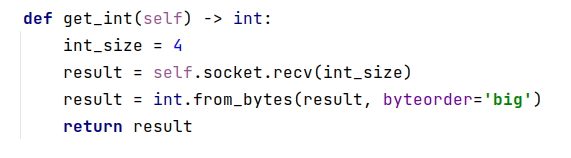


Рисунок 2.3 – Реализация метода read\_bytes

Рисунок 2.4 – Реализация метода get\_int

Алгоритм получения данных: прочитать размер сообщения (4 байта – 32 бита), создать буфер, перевести размер из сетевого порядка байт в порядок байт для платформы, читать, 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)

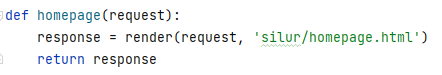


Рисунок 2.5 – Функция-обработчик главной страницы



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).

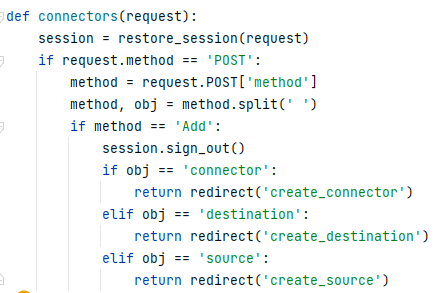


Рисунок 2.7 – Пример обработки POST запроса

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



Рисунок 2.8 – Переход на страницу /silur/sign\_in

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

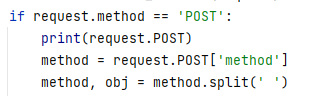


Рисунок 2.9 – Обработка POST запроса

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

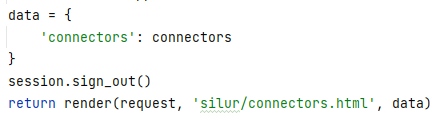


Рисунок 2.10 – Передача данных в шаблон

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.



Рисунок 2.11 – Создание блока при помощи DTL

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



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).

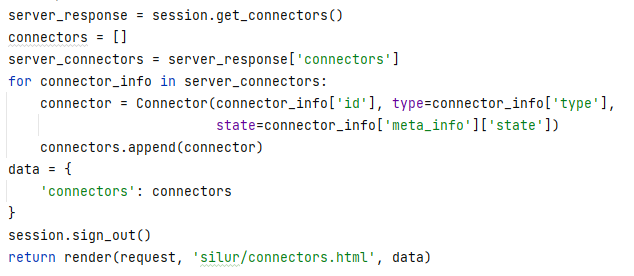


Рисунок 2.13 – Получение и передача данных в HTML документ

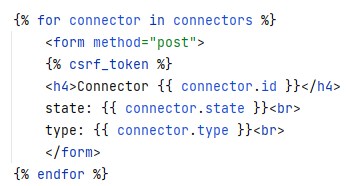


Рисунок 2.14 – Обработка данных в HTML-документе

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.

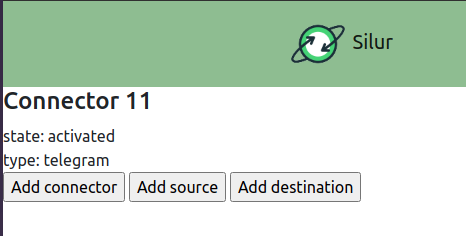


Рисунок 2.15 – Сгенерированная страница

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 моно указывать ограничения для полей (например, максимальная длина у текстового поля). Пример описания класса формы приведен на рисунке 2.16.

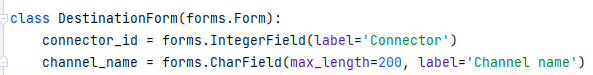


Рисунок 2.16 – Описание класса формы

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)

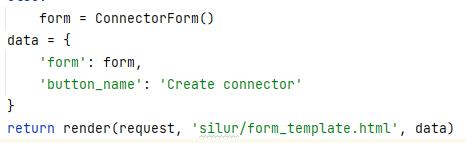


Рисунок 2.17 – Создание объекта формы и передача его в шаблон

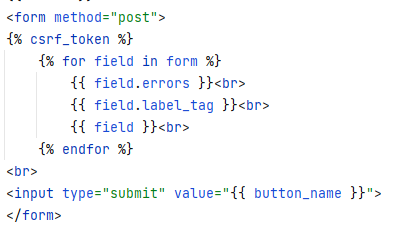


Рисунок 2.18 – Обработка формы при помощи DTL

Сразу после тэга <form> идет строка {% csrf\_token %}. Это необходимо для того, чтобы при обработке POST запроса убедится 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)

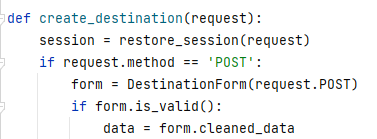


Рисунок 2.19 – Проверка правильности данных POST запроса

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

2.2.4 Модуль Models

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

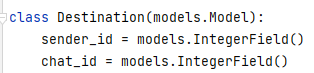


Рисунок 2.20 – Создание класса модели

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

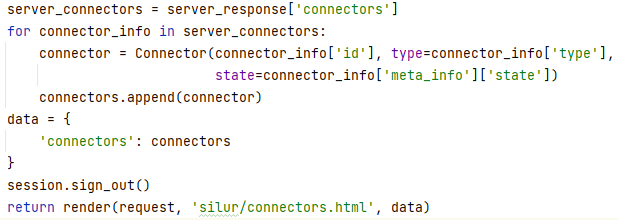


Рисунок 2.20 – Группировка данных в объекты и передача их в шаблон

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

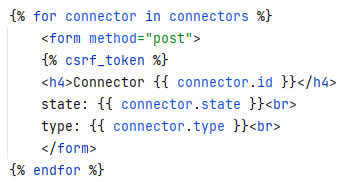


Рисунок 2.21 – использование DTL для обработки объекта модели

Получившаяся страница приведена на рисунке 2.22.

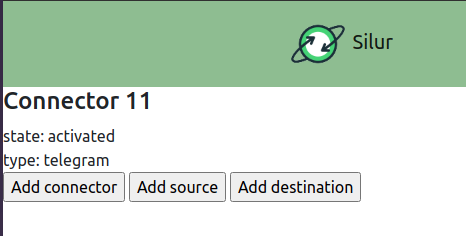


Рисунок 2.22 – Сгенерированная страница

# 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.

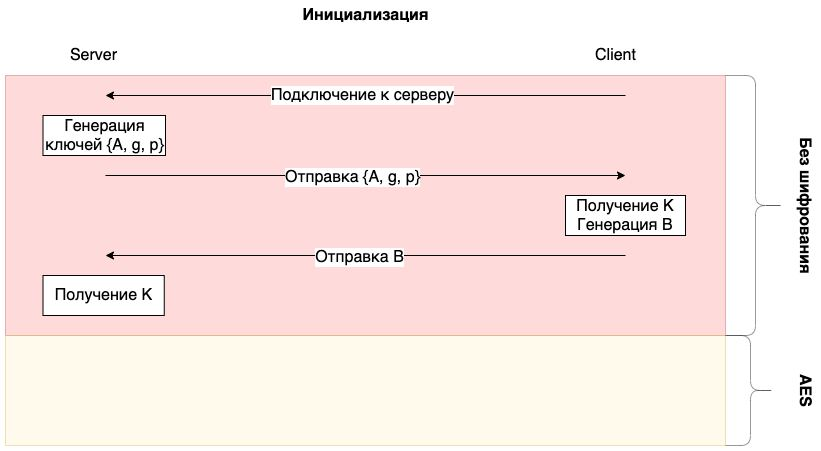


Рисунок 3.1 – Стадия инициализации

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

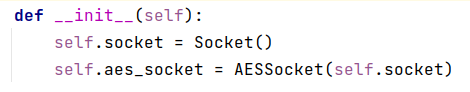


Рисунок 3.2 – Клиентская сессия создает Socket и AESSocket

На рисунке 3.2 изображено поле self.socket, которое устанавливает соединение с сервером, а затем создается AESSocket из обычного сокета, что приводит к обмену AES ключами через Deffie-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.

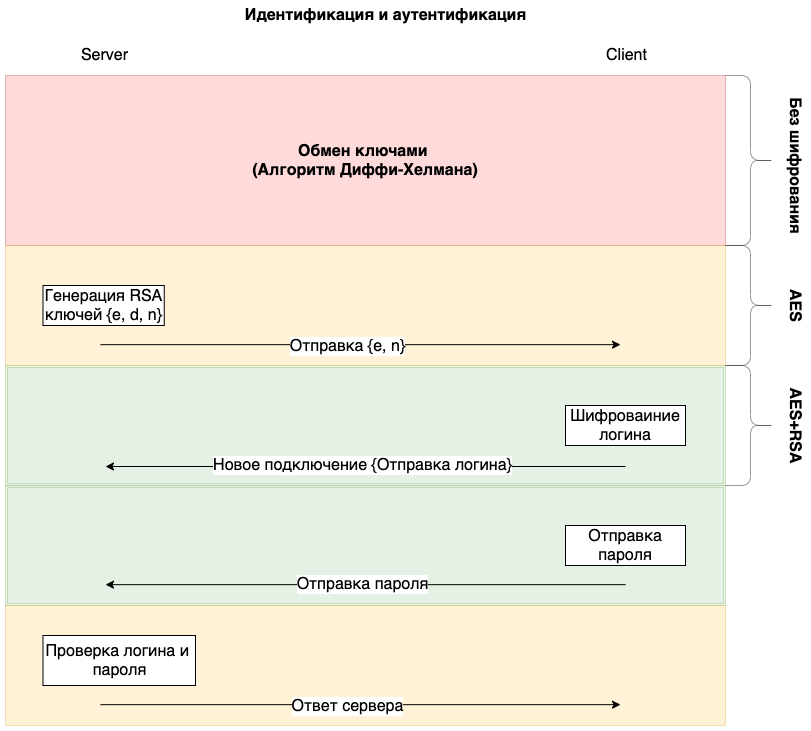
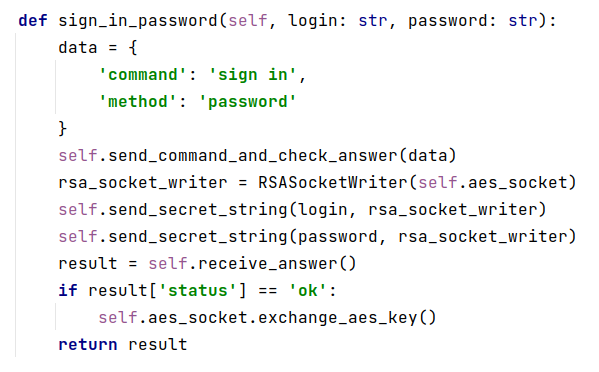


Рисунок 3.3 – Стадия идентификации и аутентификации

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

 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).

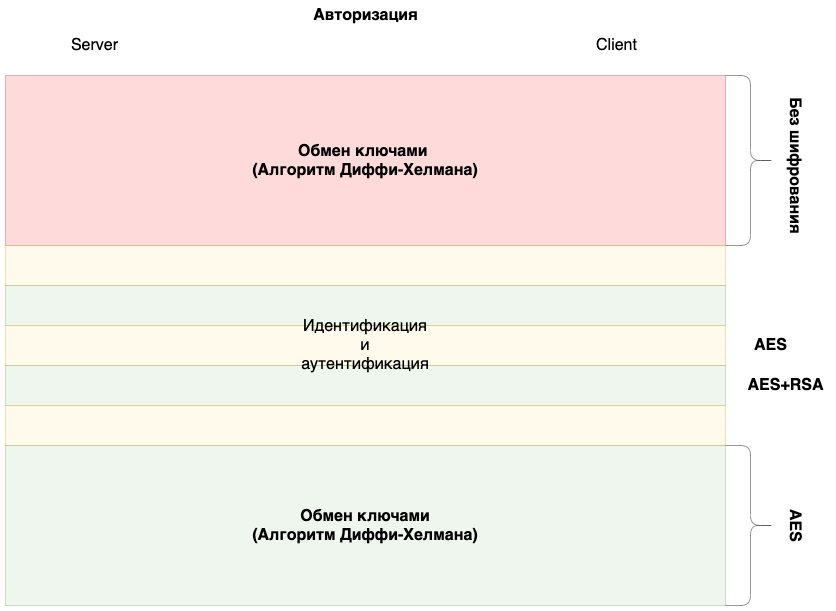


Рисунок 3.5 – Стадия авторизации

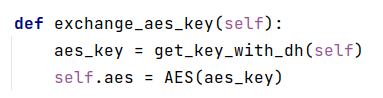


Рисунок 3.6 – Обмен ключами

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 (Deffie-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).

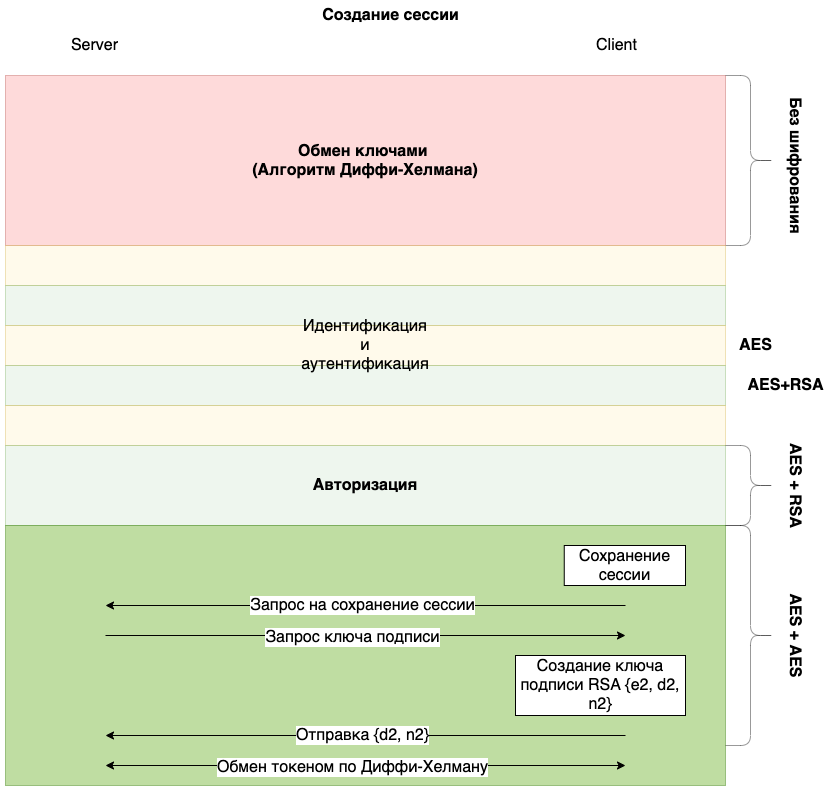


Рисунок 3.7 – Создание сессии

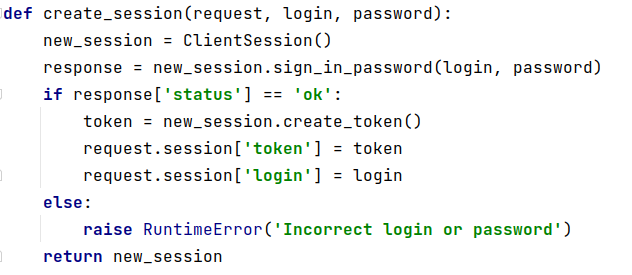


Рисунок 3.8 – Создание сессии (токена) в коде

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).

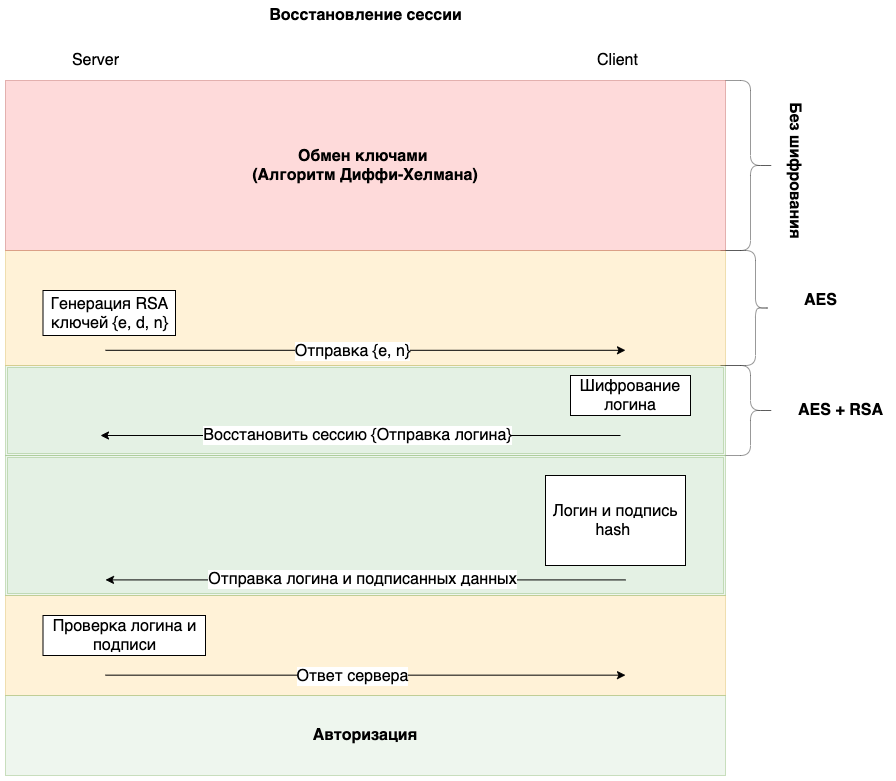


Рисунок 3.9 – Восстановление сессии

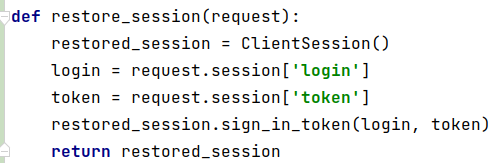


Рисунок 3.10 – Восстановление сессии в коде

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-chto-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: h 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-18б

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 |
|  | Выдача задания, составление ТЗ и его утверждение | 2 недели |
|  | Техническое проектирование. Разработка алгоритмов | 2-4 недели |
|  | Рабочее проектирование. Определение структур данных | 4-5 недели |
|  | Написание программы | 5-9 недели |
|  | Отладка программы | 10-12 недели |
|  | Написание пояснительной записки | 13 неделя |
|  | Защита курсовой работы | 13-14 недели |

# ПРИЛОЖЕНИЕ Б. ЭКРАННЫЕ ФОРМЫ

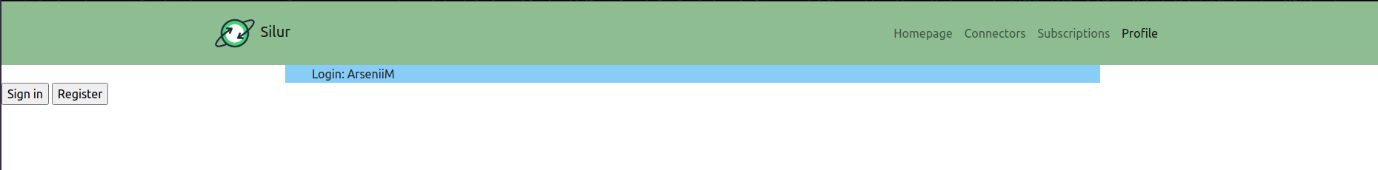


Рисунок Б.1 – Профиль



Рисунок Б.2 – Авторизация



Рисунок Б.3 – Регистрация

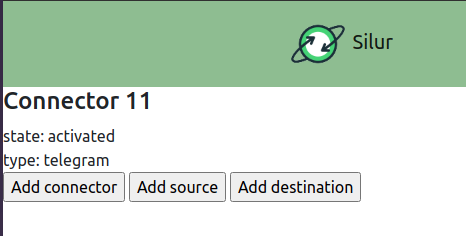


Рисунок Б.4 – Соединения

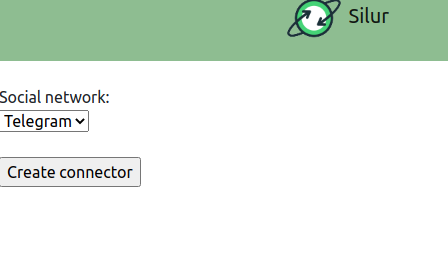


Рисунок Б.5 – Создание соединения

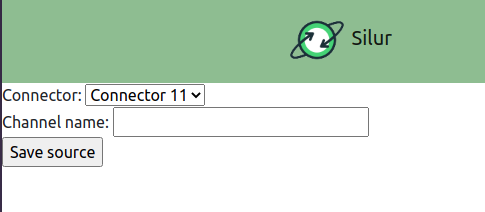


Рисунок Б.6 – Создание источника

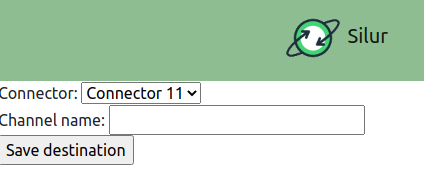


Рисунок Б.7 – Создание приёмника

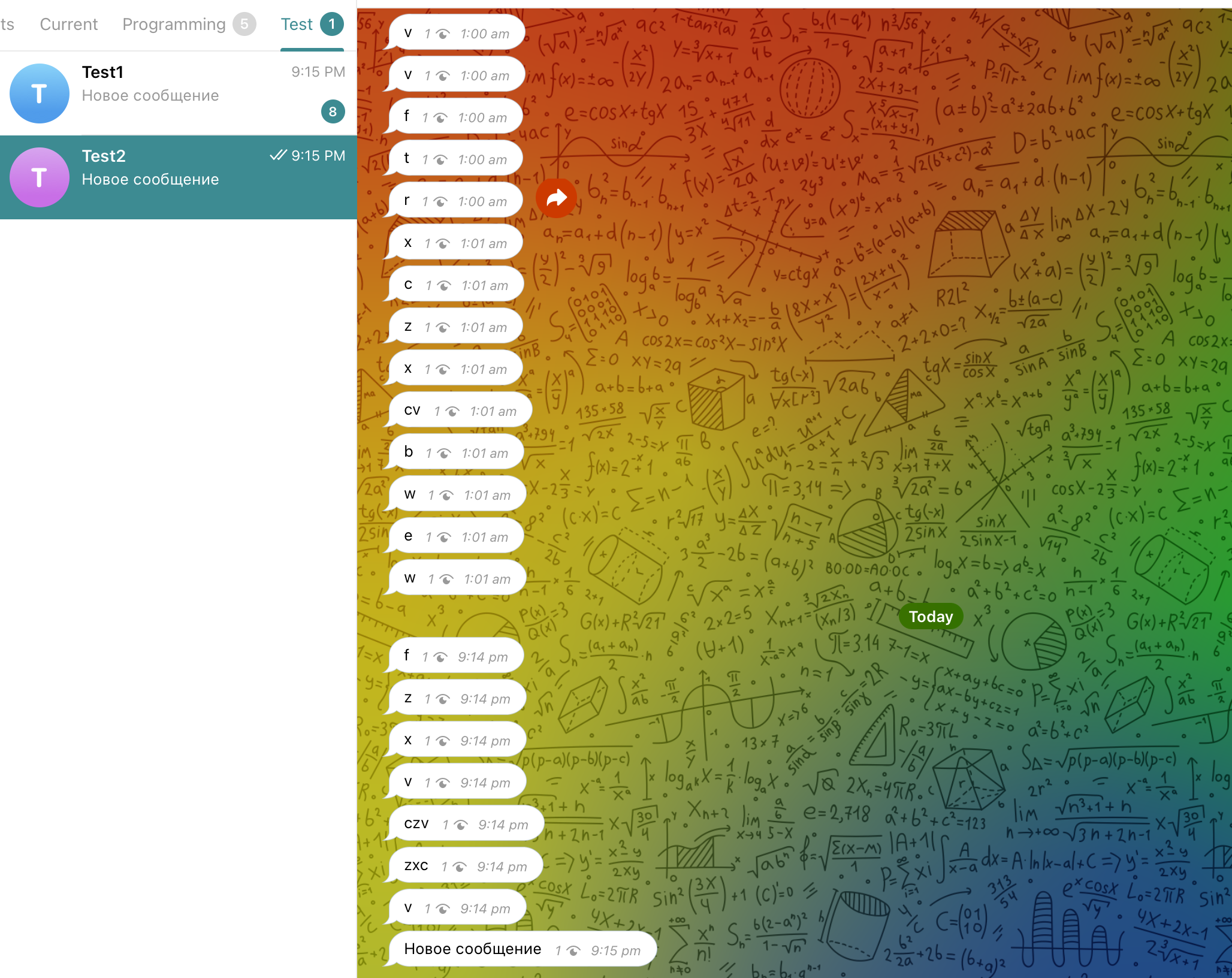


Рисунок Б.8 – Отправка сообщения в канал-источник

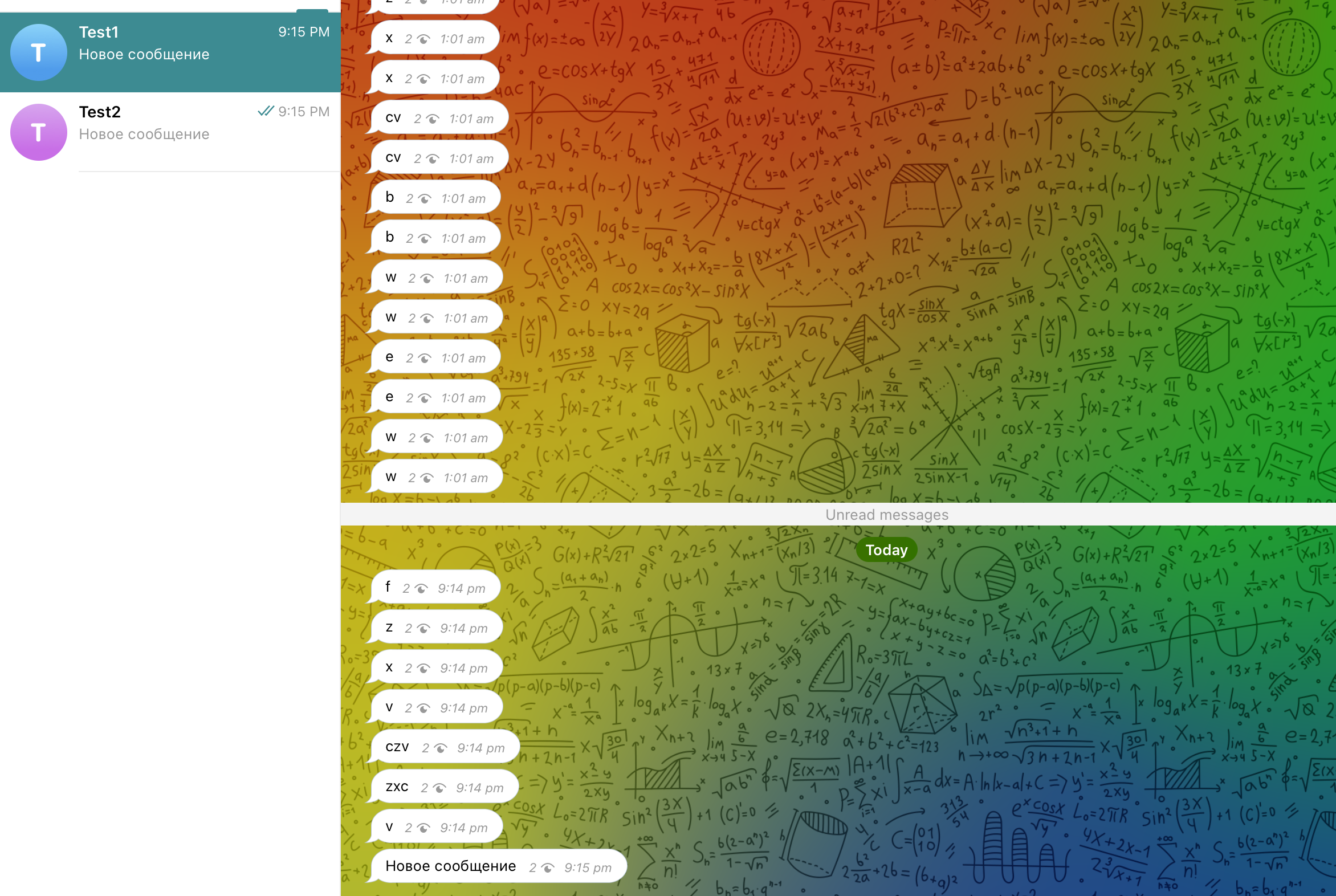


Рисунок Б.9 – Агрегация сообщения в канал-приёмник

# 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

**Utls.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.ClienSession import ClientSession

import json

from .models import \*

from .helper import find\_in\_dictionary\_set

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.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/">

<img src="{% static 'images/SilurGreenLogo-removebg.png' %}" height="60">

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\_name }}">

</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\_name }}">

</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\_name }}">

</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\_name }}">

</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 %}