

ДЕПАРТАМЕНТ ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ  
ТОМСКОЙ ОБЛАСТИ  
ОБЛАСТНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
«ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

Специальность 09.02.07 Информационные системы и программирование

УТВЕРЖДАЮ  
Заместитель директора по  
учебно-методической работе  
Е.А. Родзик  
«\_\_» \_\_\_\_\_ 2023 г.

РАЗРАБОТКА TELEGRAM-БОТА С МОДУЛЕМ НАСТРОЙКИ ДЛЯ ВЕБ-  
САЙТА ОБЪЕДИНЕНИЯ ТОМСКИХ ВУЗОВ

Пояснительная записка  
к дипломному проекту  
ДП.23.09.02.07.603.5.ПЗ

Студент		
«__» _____ 2023 г.	_____	П.Д. Левицкий
Руководитель		
«__» _____ 2023 г.	_____	Д.В. Муха
Консультант по специальности		
«__» _____ 2023 г.	_____	Д.В. Смоляков
Консультант по экономической части		
«__» _____ 2023 г.	_____	О.М. Керб
Нормоконтролёр		
«__» _____ 2023 г.	_____	А.Ю. Маюнова

ДОПУСТИТЬ К ЗАЩИТЕ

Председатель ПЦК		
«__» _____ 2023 г.	_____	А.Ю. Маюнова

Томск 2023

**ГРАФИК**

## выполнения дипломного проекта

Группа 603

Дипломант Левицкий П. Д.

Срок сдачи проекта с отзывом руководителя	<b>01.06.2023</b>
---	-------------------

Наименование разделов проекта	Объем работы в %	Срок исполнения по плану	% выполнения на 20.04.2023	% выполнения на 15.05.2023	% выполнения на 25.05.2023
Общая часть	7%	20.04.2023	100%		
Специальная часть	10%	20.04.2023	100%		
Организационно- экономическая часть	18%	15.05.2023	15%	100%	
Графическая часть	20%	16.05.2023	80%	100%	
Экспериментальная часть	20%	21.05.2023	75%	100%	
Оформление пояснительной записки	25%	25.05.2023	50%	90%	100%
Сдача проекта в готовом виде руководителю	100%	31.05.2023			

Дни консультаций	
Дата	время
20.04.2023	8:30
25.04.2023	8:30
29.04.2023	8:30
13.05.2023	8:30
20.05.2023	8:30
27.05.2023	8:30
31.05.2023	17:50

Выполнение проекта		
дата	%	подпись руководителя

**ПРИМЕЧАНИЕ:** Составление пояснительной записки производится параллельно с выполнением соответствующих разделов проекта.

Дипломант: Левицкий П. Д.

«\_\_» \_\_\_\_\_ 2023 г.

Руководитель: Муха Д. В.

«\_\_» \_\_\_\_\_ 2023 г.

ДЕПАРТАМЕНТ ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ  
ТОМСКОЙ ОБЛАСТИ  
ОГБПОУ «ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

*Специальность 09.02.07 – «Информационные системы и программирование»*

СОГЛАСОВАНО  
заведующий отделением ПО  
\_\_\_\_\_ В.Е. Курочкин  
«\_\_» \_\_\_\_\_ 2023 г.

**ЗАДАНИЕ**

на дипломное проектирование

студенту группы 603

Левицкий Павел Дмитриевич  
(Фамилия, Имя, Отчество)

Дата выдачи: 10 апреля 2023 г.

Дата окончания: 01 июня 2023 г.

**I ТЕМА ДИПЛОМНОГО ПРОЕКТА**

Разработка telegram-бота с модулем настройки для веб-сайта объединения  
ТОМСКИХ ВУЗОВ.

**II ТЕХНИЧЕСКИЕ УСЛОВИЯ**

Процессор 11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz, оперативная память 16гб  
DDR4 SDRAM, операционная система Windows 11 Домашняя для одного  
языка

### III СОДЕРЖАНИЕ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

- 1) Общая часть
- 2) Специальная часть
- 3) Экономическая часть
- 4) Заключение
- 5) Перечень используемых источников
- 6) Приложение А. Листинг кода
- 7) Приложение Б. Инструкция пользователя
- 8) Приложение В. Диаграммы состояний

### IV ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ ПРОЕКТА

C# v10, .NET v6.0, Microsoft Visual studio 2022, DB Browser for SQLite, draw.io, HtmlAgilityPack v1.11.46, MaterialDesignThemes v4.8.0, Microsoft.Data.Sqlite v7.0.5, Telegram.Bot v18.0.0.

### V ЭКОНОМИЧЕСКАЯ ЧАСТЬ ПРОЕКТА

Основная заработная плата персонала, дополнительная плата персонала, отчисления в социальные статьи, стоимость работ на ЭВМ, расчет стоимости материалов, расчет косвенных затрат на разработку.

### VI ЛИТЕРАТУРА

Хабр. Использование диаграммы вариантов использования UML при проектировании программного обеспечения, IT-GOST.RU. Теория и практика UML. Диаграмма состояний, METANIT.COM. Полное руководство по языку программирования C# 10 и платформе .NET 6, Microsoft Learn. Общие сведения о WPF, METANIT.COM. Введение в WPF, ИНТУИТ. Практикум 9: Пример технического задания для рецензирования.

## VII СРОКИ ВЫПОЛНЕНИЯ ПРОЕКТА

В соответствии с графиком выполнения дипломного проекта.

Руководитель дипломного проекта \_\_\_\_\_ / Д.В. Смялков  
(подпись) (расшифровка подписи)

Дипломное задание рассмотрено на заседании предметной цикловой комиссии

Протокол № \_\_\_\_ от « \_\_\_\_ » \_\_\_\_\_ 2023 г.

Председатель предметной цикловой комиссии \_\_\_\_\_ А.Ю. Маюнова  
(подпись)

Задание к исполнению получил(а) студент \_\_\_\_\_ / П. Д. Левицкий  
(подпись) (расшифровка подписи)

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	3
1 ОБЩАЯ ЧАСТЬ .....	5
1.1 Анализ предметной области .....	5
1.2 Средства и среда разработки.....	6
2 СПЕЦИАЛЬНАЯ ЧАСТЬ.....	8
2.1 Описание требований к информационной системе.....	8
2.2 Диаграмма вариантов использования .....	10
2.3 Диаграмма состояний .....	11
2.4 Схема базы данных .....	15
2.5 Словарь данных .....	16
2.6 Пользовательские сценарии .....	21
2.7 Прототипы интерфейсов .....	23
3 ЭКОНОМИЧЕСКАЯ ЧАСТЬ.....	29
3.1 Расчет затрат на разработку программы и решение задачи на ЭВМ.....	29
3.2 Расчет экономического эффекта и определение срока окупаемости .....	34
ЗАКЛЮЧЕНИЕ .....	38
ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ .....	39
ПРИЛОЖЕНИЕ А. Листинг программы .....	40
ПРИЛОЖЕНИЕ Б. Инструкция пользователя .....	1488
ПРИЛОЖЕНИЕ В. Диаграммы состояний.....	1

## Введение

ТУСУР - Томский государственный университет систем управления и радиоэлектроники, признанный лидер в сфере подготовки квалифицированных кадров для высокотехнологичных отраслей экономики, аэрокосмического и оборонного комплексов страны, внедряющий инновационные образовательные и исследовательские программы, прикладные разработки новой техники, аппаратуры и систем управления. Университет уверенно держит первенство в реализации программ инновационного развития, выпускники ТУСУРа составляют кадровую основу многих предприятий как в России, так и за рубежом. Руководство университета участвовало в создании проекта "Консорциум "Объединение томских вузов"".

Сам проект в свою очередь призван упростить поступление абитуриентов в высшие учебные заведения, входящие в состав этого консорциума. На данный момент в состав консорциума входят ТУСУР, ТПУ, ТГПУ, ТГАСУ и ТГУ, также принимает участие СибГМУ.

Основное предназначение сайта проекта - подбор направлений обучения и подача заявки на поступление на выбранное направление. Направления обучения содержатся только от тех университетов, которые входят в состав консорциума.

Стоит сказать, что сам проект ориентирован также и на иностранных абитуриентов, по этой причине на веб-сайте присутствует страница с прохождением тестирования на знание русского языка.

Как руководитель проекта, университет стремится к увеличению охвата аудитории в лице потенциальных абитуриентов, по этой причине ему необходимо получить готовое решение, выполняющее данную задачу.

Было предложено несколько возможных вариантов реализации программного продукта, выполняющего задачу расширения аудитории, из них были следующие: создание чат-бота на готовой платформе, позволяющей провести интеграцию с Telegram и непосредственное написание Telegram-бота, используя стандартные средства разработки.

Первый вариант не подходит по нескольким причинам - подобные платформы не предполагают написание программного кода, но лишь использование визуального конструктора с весьма ограниченным функционалом и они не позволяют использовать динамические источники данных.



# 1 ОБЩАЯ ЧАСТЬ

## 1.1 Анализ предметной области

Работа сервиса напрямую зависит от актуальности данных, которыми располагает сервис. В противном случае использование сервиса пользователями будет не только бесполезным, но и вредным. По этой причине продукт должен собирать данные с веб-сайта проекта при каждом запуске и иметь возможность вручную обновить собранные данные.

Основная атомарная единица, хранящаяся на ресурсе – карточка направления.

В свою очередь карточка направления содержит в себе такие данные, как название университета, название направления, уровень обучения, форма обучения, код программы, продолжительность, квалификация, язык обучения, ФИО куратора, телефон, почта и стоимость за год обучения.

Также на веб-сайте проекта доступна форма выбора направления по уровню обучения (квалификации) и направлению, результатом является набор карточек направлений, при нажатии кнопки «Поступить» на одной из них открывается страница, позволяющая оставить заявку на поступление.

Исходя из описанного выше необходимо написать Telegram-бота, генерирующего карточки направлений, отбирающиеся из общего числа полученных карточек и в соответствии с выбором пользователя (потенциального абитуриента) отправляющего ее в чат. К такому сообщению должна быть прикреплена кнопка для обратной связи.

Задачи дипломного проекта:

- 1) Исследование технологий по созданию ботов;
- 2) Обзор решений для создания чат-ботов;
- 3) Анализ конкурентов;
- 4) Формулирование сценария работы бота;
- 5) Выбор платформы и языка программирования;
- 6) Исследование доступных библиотек;
- 7) Анализ возможностей доступных библиотек;
- 8) Реализация программного кода;
- 9) Исправление ошибок логики и ошибок интерфейса;
- 10) Составление документации.

## 1.2 Средства и среда разработки

На этапе проектирования продукта были использовано средство draw.io – ресурс для отрисовки диаграмм, обладает всем необходимым функционалом, таким, как обширная коллекция фигур и возможностью экспорта диаграмм в .png, прост в использовании и обладает кроссплатформенностью, применялся для отрисовки логической модели базы данных и построении остальных диаграмм.

На этапе разработки программного кода были использованы следующие средства:

Microsoft Visual studio 2022 - интегрированная среда разработки, позволяющая написание программного кода, предоставляющая средства отладки и сборки кода, а также последующей публикации приложений. Помимо стандартного редактора и отладчика, которые есть в большинстве IDE, Visual Studio включает в себя компиляторы, средства автозавершения кода, графические конструкторы и многие другие функции для повышения качества процесса разработки.

Среда разработки располагает редактируемым дизайном, огромным количеством расширений и приятным UI.

В качестве языка программирования был выбран C#, являющийся объектно- и компонентно-ориентированным языком программирования. Обладает рядом положительных качеств:

C# – объектно-ориентированный, простой и в то же время мощный язык программирования, позволяющий разработчикам создавать многофункциональные приложения;

C# относится к языкам компилируемого типа, поэтому он обладает всеми преимуществами таких языков;

C# объединяет лучшие идеи современных языков программирования Java, C++, Visual Basic и т.д;

Из-за большого разнообразия синтаксических конструкций и возможности работать с платформой .Net, C# позволяет быстрее, чем любой другой язык, разрабатывать программные решения;

C# отличается надежностью и наличием большого количества синтаксических конструкций.

Nuget-пакет HtmlAgilityPack v1.11.46 – библиотека, необходимая для работы парсера HTML-страницы средствами C#. Поддерживает XPATH, необходимый для парсинга HTML-документа.

SQLite — компактная встраиваемая СУБД. Удобно использовать в случаях, когда не требуется разделенное хранение данных по типу клиент-сервер, так как движок является не отдельно работающим процессом, а библиотекой, с которой выполняется построение программы. В свою очередь это позволяет время отклика и упрощает программу в целом.

Nuget-пакет Telegram.Bot v18.0.0 – предоставляет возможность работы с Telegram Bot API и непосредственного написания логики бота и подключения к боту по токену.

Nuget-пакет Microsoft.Data.Sqlite v7.0.4 – легковесный ADO.NET провайдер, предоставляет возможность работы с SQLite.

## 2 СПЕЦИАЛЬНАЯ ЧАСТЬ

### 2.1 Описание требований к информационной системе

Информационная система предоставляет возможность сбора, хранения и выдачи данных о карточках специальности посредством опроса в телеграм-боте.

Администратор может запускать и останавливать бота, вручную запускать парсер, просматривать различную статистику по работе бота, просматривать генерируемые ИС файлы журналов, просматривать полученные ботом сообщения в реальном времени, устанавливать URL сайта, с которого собираются данные, устанавливать, изменять и отключать пароль, изменять настройки ИС (пути к файлам журналов, токен бота, URL страницы), просматривать статистику и осуществлять поиск по программам университетов, просматривать и осуществлять поиск по карточкам направлений, просматривать статистику по длительности сессий, экспортировать данные по карточкам направлений и программам университетов.

Пользователями информационной системы (Далее - ИС) являются: администратор, пользователь.

Функционал администратора реализован непосредственно на стороне ИС, функционал пользователя - в виде чата с Telegram-ботом.

Функционал пользователя:

- 1) Переход на веб-сайт проекта;
- 2) Переход на веб страницу прохождения тестирования на знание русского языка;
- 3) Выбор уровня обучения;
- 4) Выбор университета;

- 5) Выбор программы обучения;
- 6) Получение полных данных о выбранной программе обучения;
- 7) Переход в почтовое приложение или на веб-сайт для последующей связи с куратором по почте, указанным в карточке;
- 8) Выход в главное меню.

Функционал администратора:

- 1) Установка пароля, если это первый запуск приложения;
- 2) Смена существующего пароля;
- 3) Отключение установленного пароля;
- 4) Вход по паролю, если включен;
- 5) Запуск бота;
- 6) Остановка бота;
- 7) Просмотр полученных ботом сообщений;
- 8) Просмотр сведений об ошибках Telegram API;
- 9) Ручной запуск парсера;
- 10) Просмотр сведений о полученных карточках направлений;
- 11) Умный поиск по названию направления среди карточек направлений;
- 12) Умный поиск по названию университета среди карточек направлений;
- 13) Сортировка таблицы с карточками направлений;
- 14) Просмотр статистики направлений по университетам
- 15) Умный поиск по названию университета среди статистики программ по университетам;

- 16) Умный поиск по количеству направлений среди статистики программ по университетам;
- 17) Сортировка таблицы со статистикой направлений по университетам
- 18) Установка URL сайта, с которого парсер будет собирать данные;
- 19) Установка токена бота, к которому будет подключаться ИС;
- 20) Установка пути для экспортирующихся отчетов;
- 21) Установка пути для журналов;
- 22) Очистка окна живого журнала;

## 2.2 Диаграмма вариантов использования

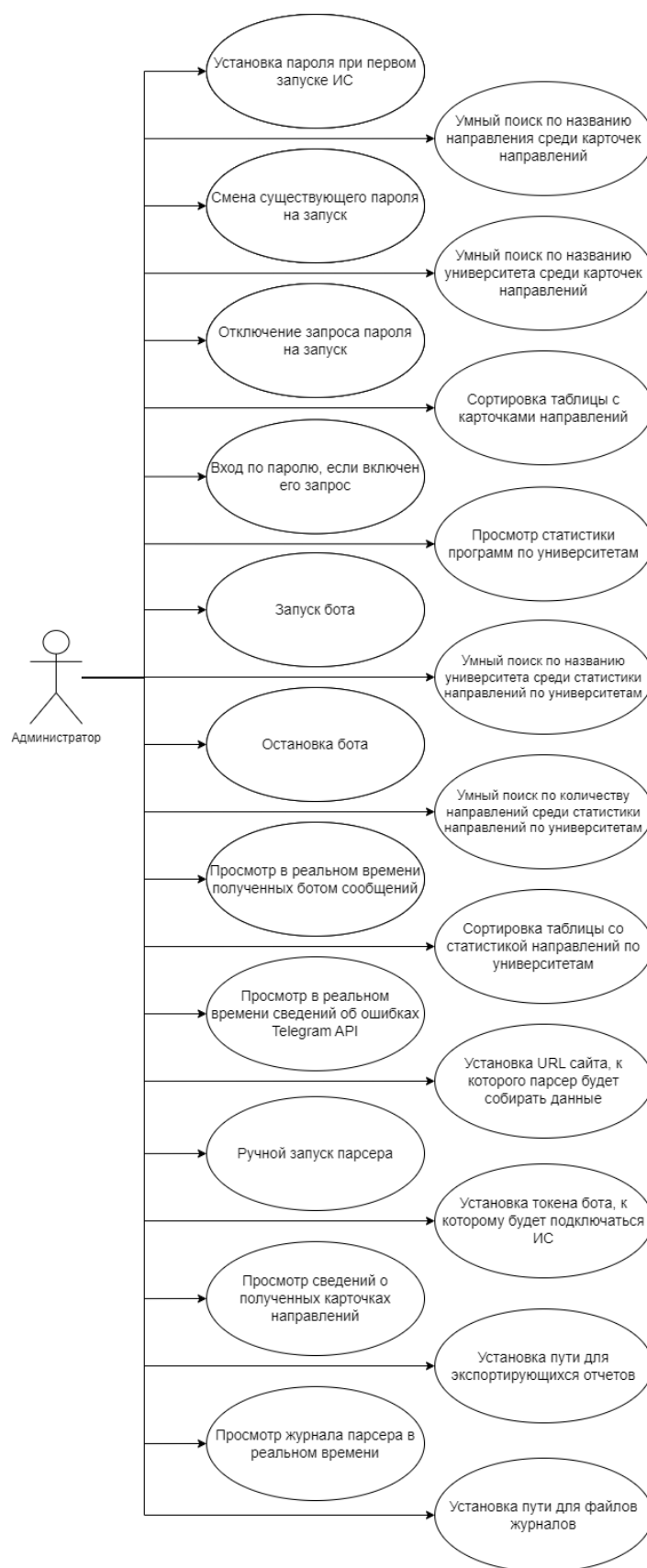


Рисунок 1 – Диаграмма вариантов использования для администратора



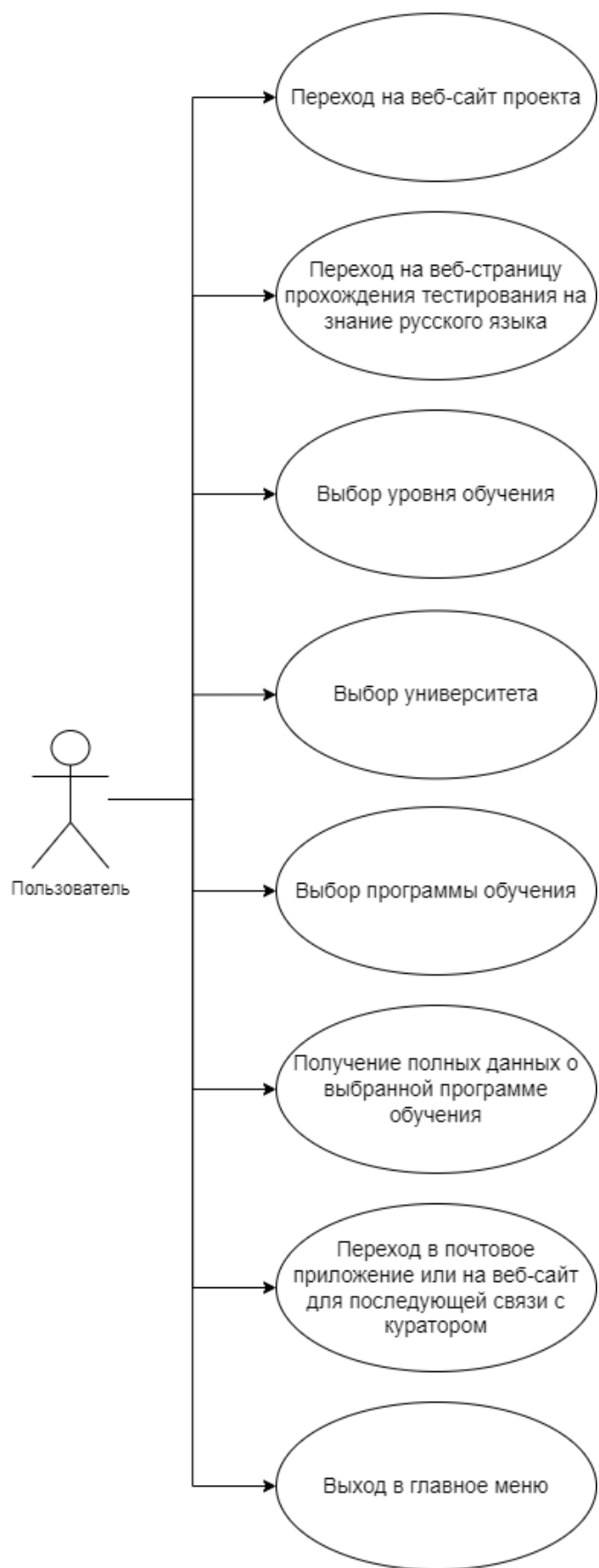


Рисунок 2 – Диаграмма вариантов использования для пользователя

## 2.3 Схема базы данных

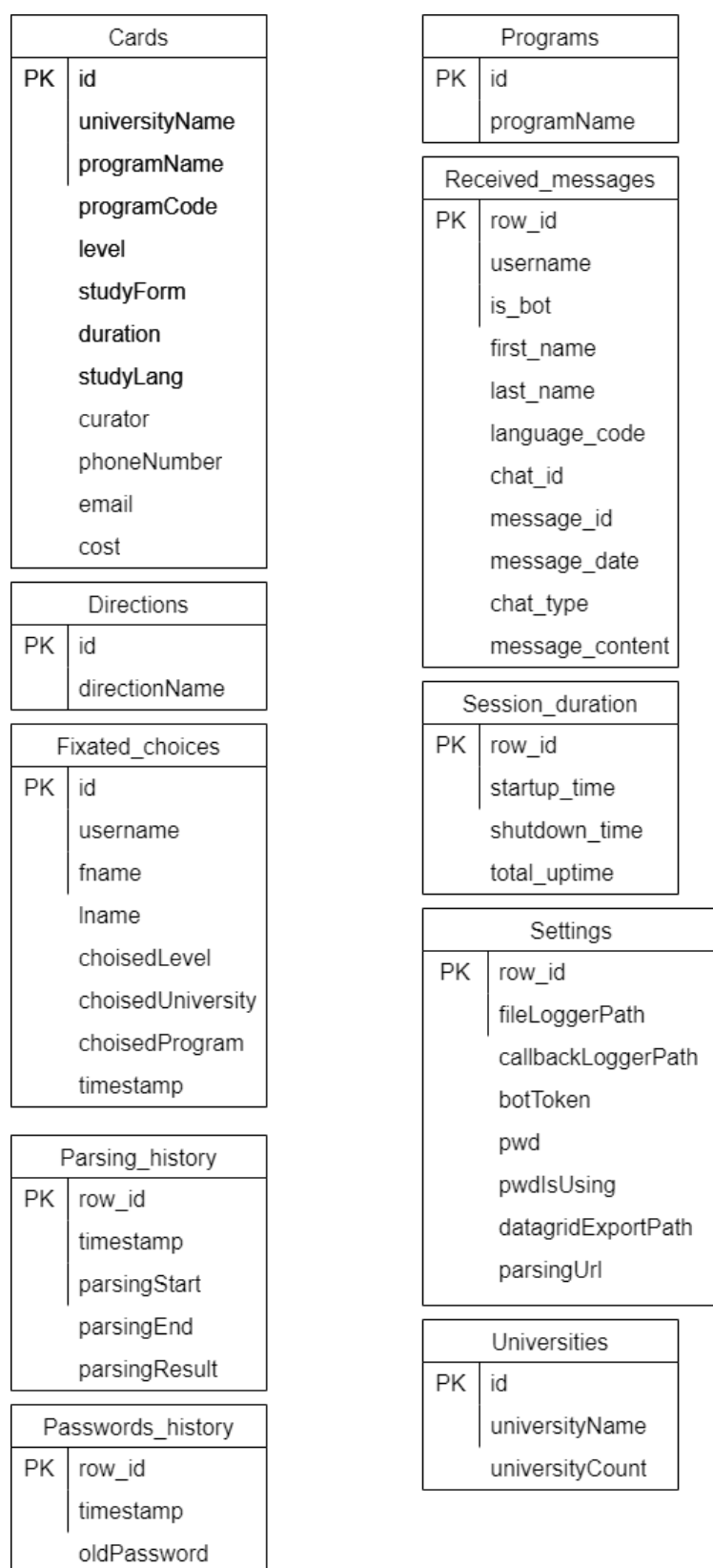


Рисунок 3 – Схема базы данных

## 2.4 Словарь данных

Таблица 1 – Cards (Карточки направлений)

Название поля	Тип данных	Описание
id (PK)	INT	Идентификатор карточки (уникальный)
universityName	TEXT	Название университета
programName	TEXT	Название программы
programCode	TEXT	Код программы
level	TEXT	Уровень обучения
studyForm	TEXT	Форма обучения
duration	TEXT	Длительность
studyLang	TEXT	Язык обучения
curator	TEXT	Куратор
phoneNumber	TEXT	Номер телефона
email	TEXT	Почта
cost	TEXT	Стоимость обучения

Таблица 2 – Directions (Направления)

Название поля	Тип данных	Описание
id (PK)	INT	Идентификатор направления (уникальный)
directionName	TEXT	Название направления (уникальное)

Таблица 3 – Fixated\_choices (Зафиксированные выборы пользователей)

Название поля	Тип данных	Описание
id (PK)	INT	Идентификатор набора выборов (уникальный)
username	TEXT	Никнейм пользователя
fname	TEXT	Имя пользователя
lname	TEXT	Фамилия пользователя
choisedLevel	TEXT	Выбранный уровень обучения
choisedUniversity	TEXT	Выбранный университет
choisedProgram	TEXT	Выбранная программа
timestamp	TEXT	Время фиксации результата

Таблица 4 – Parsing\_history (История парсинга)

Название поля	Тип данных	Описание
row_id (PK)	INT	Идентификатор строки (уникальный)
timestamp	TEXT	Время, затраченное на парсинг
parsingStart	TEXT	Время начала парсинга
parsingEnd	TEXT	Время конца парсинга
parsingResult	INT	Результат парсинга (количество полученных карточек)

Таблица 5 – Passwords\_history (История установленных паролей)

Название поля	Тип данных	Описание
row_id (PK)	INT	Идентификатор строки (уникальный)
timestamp	TEXT	Дата смены пароля
oldPassword	TEXT	Неактуальный на момент записи пароль

Таблица 6 – Programs (Программы обучения)

Название поля	Тип данных	Описание
id (PK)	INT	Идентификатор программы обучения (уникальный)
programName	TEXT	Названия программы

Таблица 7 – Received\_messages (Принятые ботом сообщения)

Название поля	Тип данных	Описание
row_id (PK)	INT	Идентификатор строки (уникальный)
username	TEXT	Никнейм пользователя
is_bot	TEXT	Пользователь – бот?
first_name	TEXT	Имя пользователя
last_name	TEXT	Фамилия пользователя
language_code	TEXT	Код языка пользователя
chat_id	TEXT	Идентификатор чата

message_id	TEXT	Идентификатор принятого сообщения
message_date	TEXT	Дата получения ботом сообщения
chat_type	TEXT	Тип чата с пользователем
message_content	TEXT	Содержимое принятого сообщения

Таблица 8 – Session duration (Длительность сессий)

Название поля	Тип данных	Описание
row_id (PK)	INT	Идентификатор строки (уникальный)
startup_time	TEXT	Дата и время запуска бота
shutdown_time	TEXT	Дата и время остановки бота
total_uptime	TEXT	Интервал работы бота

Таблица 9 – Settings (Настройки)

Название поля	Тип данных	Описание
row_id (PK)	INT	Идентификатор строки (уникальный)
fileLoggerPath	TEXT	Путь к журналу принятых сообщений
callbackLoggerPath	TEXT	Путь к журналу выборов пользователей
botToken	TEXT	Токен telegram-бота

Продолжение таблицы 9

pwd	TEXT	Пароль на вход в GUI
pwdIsUsing	TEXT	Вход по паролю активирован?
datagridExportPath	TEXT	Путь экспорта данных из таблицы карточек направлений
parsingUrl	TEXT	URL, требуемый парсеру

Таблица 10 – Universities (Университеты)

Название поля	Тип данных	Описание
Id (PK)	INT	Идентификатор университета (уникальный)
universityName	TEXT	Название университета
universityCount	INT	Количество направлений

## 2.5 Пользовательские сценарии

После первого запуска программы откроется окно, предлагающее установить пароль. Для этого необходимо ввести пароль и повторить, после чего нажать кнопку "Next".

Если пароли совпадают между собой, длина более трех символов и такой пароль не был использован ранее - пароль устанавливается, после чего открывается главное окно приложения.

Для использования входа в приложение необходимо на вкладке "Settings" установить флажок "Use this password", и при следующем перезапуске приложение запросит установленный ранее пароль.

Существуют поля для смены пароля. Для смены пароля необходимо ввести новый пароль дважды в соответствующие поля и нажать кнопку «Set password», после этого откроется окно ввода старого пароля. Для смены пароля необходимо указать старый пароль и нажать кнопку «Next», вследствие чего будет открыто окно настроек с уведомлением об успешной смене пароля на новый.

Также на данной вкладке при первом запуске необходимо настроить параметры, необходимые для работы бота. Их можно установить посредством ввода в соответствующие поля и нажатия соответствующих кнопок записи.

Бота возможно запустить с главной вкладки, но не ранее окончания работы парсера. Для контроля окончания работы парсера необходимо перейти на вкладку "Parser". Живой журнал даст знать об окончании процедуры парсинга, также есть возможность ручного перезапуска парсера.



Далее перейти на вкладку "Main" и нажать кнопку "Start", после чего отобразится сообщение о прослушивании бота с информацией о боте и дате старта прослушивания.

При необходимости можно остановить бота, остановить бота и выйти из приложения, запустить экземпляр командной строки или ставить живой журнал на паузу, очищать окно вывода или экспортировать его содержимое.

На вкладке "Parsed cards" выводятся данные по всем полученным парсером карточкам направлений, также расположено два поля умного поиска - по университету и по названию направления.

На вкладке "Parsed universities" данные по всем полученным парсером университетам и количеством направлений по каждому из них. Здесь также расположено два поля для умного поиска - по названию университета и по количеству направлений.

## 2.6 Прототипы интерфейсов

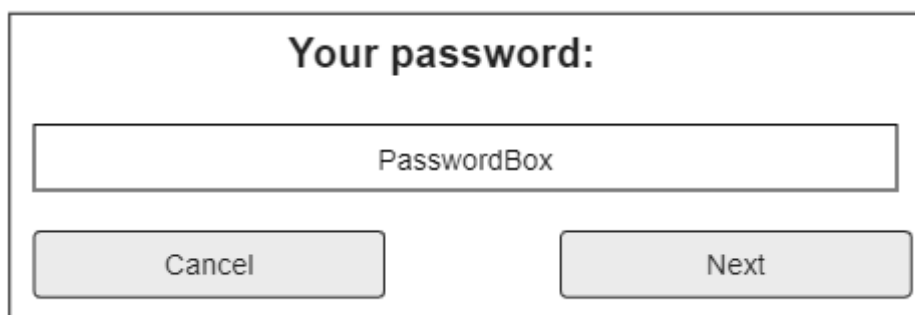
Окно установки пароля (Рисунок 4)



The diagram shows a rectangular window titled "Set a new password:". Inside the window, there are two identical rectangular input fields stacked vertically, each labeled "PasswordBox". Below these input fields, there are two rectangular buttons: "Cancel" on the left and "Next" on the right.

Рисунок 4 – Окно установки пароля

Окно входа (Рисунок 5)



The diagram shows a rectangular window titled "Your password:". Inside the window, there is a single rectangular input field labeled "PasswordBox". Below this input field, there are two rectangular buttons: "Cancel" on the left and "Next" on the right.

Рисунок 5 – Окно входа

## Вкладка «Main» окна botserver\_standard (Рисунок 6)

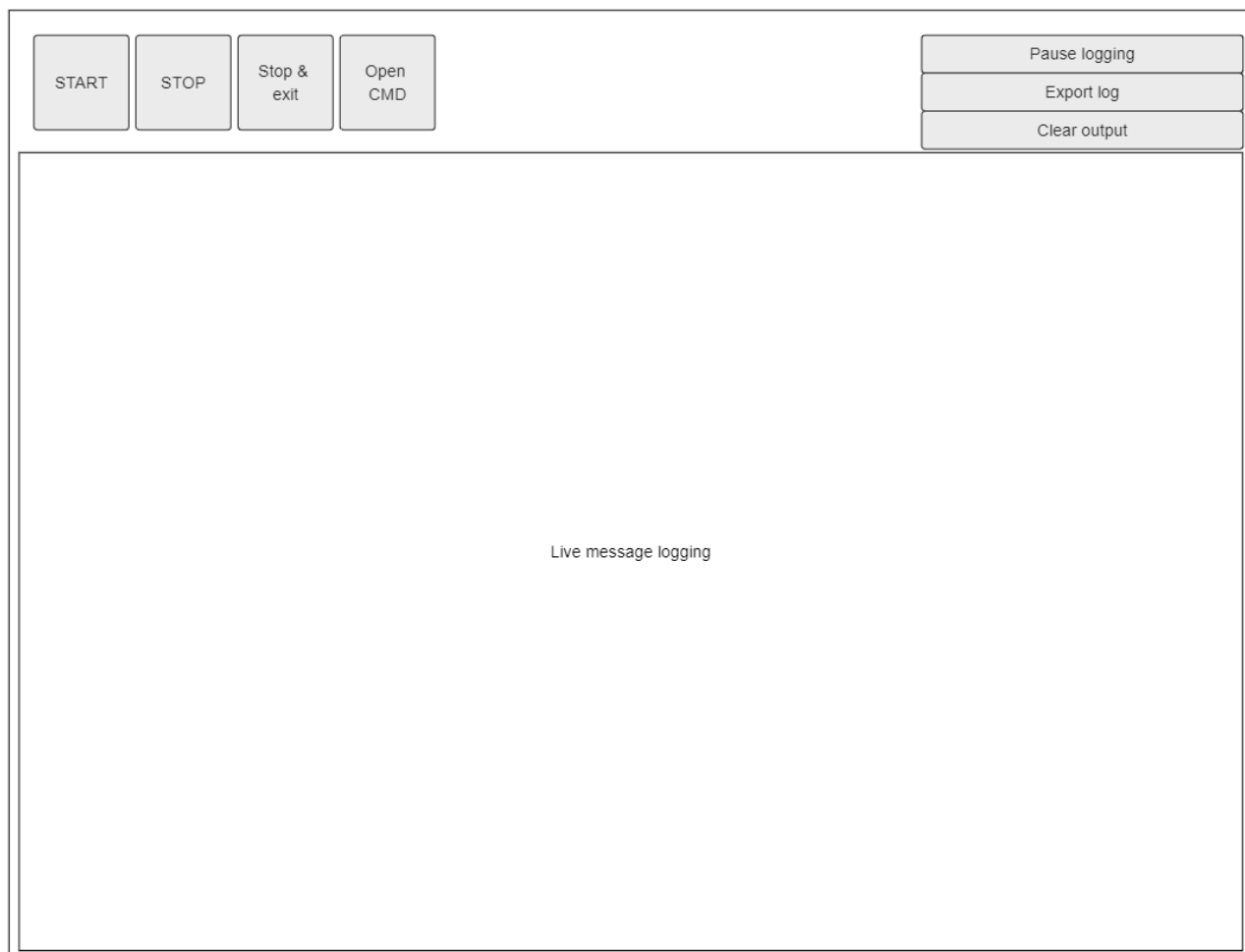


Рисунок 6 – Вкладка «Main» окна botserver\_standard

## Вкладка «Parser» окна botserver\_standard (Рисунок 7)

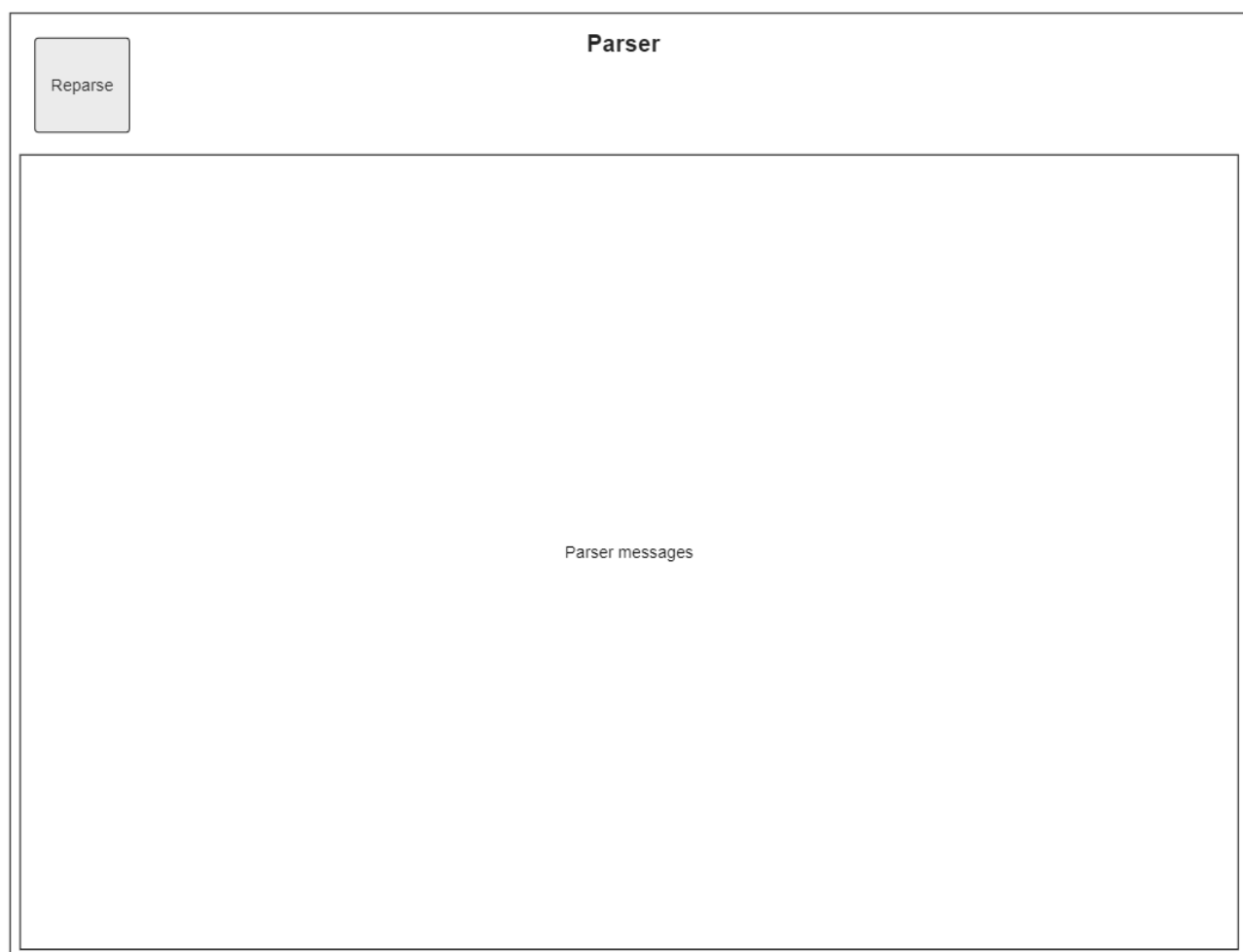


Рисунок 7 – Вкладка «Parser» окна botserver\_standard

## Вкладка «Parsed cards» окна botserver\_standard (Рисунок 8)

Поиск по названию программы :

Поиск по университету :

Название университета	Название программы	Код программы	Уровень обучения	Форма обучения	Срок обучения	Язык обучения	Куратор	Телефон	Почта	Стоимость
DataGrid										

Рисунок 8 – Вкладка «Parsed cards» окна botserver\_standard

### Вкладка «Parsed universities» окна «botserver\_standard» (Рисунок 9)

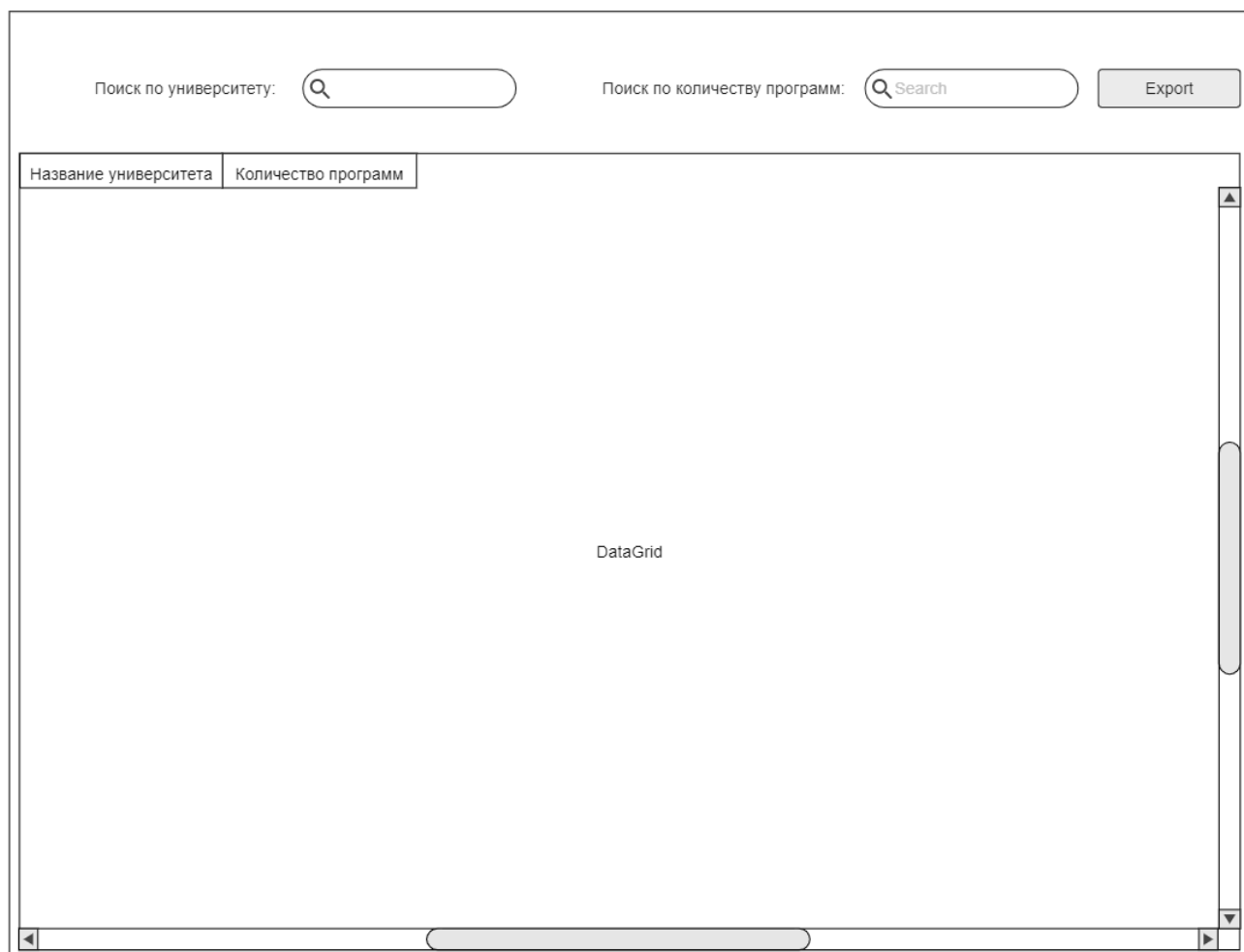


Рисунок 9 – Вкладка «Parsed universities» окна «botserver\_standard»

Вкладка «Settings» окна «botserver\_standard» (Рисунок 10)

**Settings of the BotServer**

Bot token:	<div>TextBox</div>	<div>Set bot token</div>
Message log path:	<div>TextBox</div>	<div>Set message log path</div>
Cards export path:	<div>TextBox</div>	<div>Set export path</div>
Choices log path:	<div>TextBox</div>	<div>Set choices log path</div>
Parsing URL:	<div>TextBox</div>	<div>Set parsing URL</div>
Enter password:	<div>TextBox</div>	<div><input checked="" type="checkbox"/> Use this password</div>
Repeat password:	<div>TextBox</div>	<div>Set password</div>

Рисунок 10 – Вкладка «Settings» окна «botserver\_standard»

## Заключение

По итогу выполнения дипломного проекта была успешно разработана информационная система со следующим функционалом:

Функционал пользователя:

- 1) Переход на веб-сайт проекта;
- 2) Переход на веб страницу прохождения тестирования на знание русского языка;
- 3) Выбор уровня обучения;
- 4) Выбор университета;
- 5) Выбор программы обучения;
- 6) Получение полных данных о выбранной программе обучения;
- 7) Переход в почтовое приложение или на веб-сайт для последующей связи с куратором по почте, указанным в карточке;
- 8) Выход в главное меню.

Функционал администратора:

- 1) Установка пароля, если это первый запуск приложения;
- 2) Смена существующего пароля;
- 3) Отключение установленного пароля;
- 4) Вход по паролю, если включен;
- 5) Запуск бота;
- 6) Остановка бота;
- 7) Просмотр полученных ботом сообщений;
- 8) Просмотр сведений об ошибках Telegram API;
- 9) Ручной запуск парсера;
- 10) Просмотр сведений о полученных карточках направлений;
- 11) Умный поиск по названию направления среди карточек направлений;



- 12) Умный поиск по названию университета среди карточек направлений;
- 13) Сортировка таблицы с карточками направлений;
- 14) Просмотр статистики направлений по университетам
- 15) Умный поиск по названию университета среди статистики программ по университетам;
- 16) Умный поиск по количеству направлений среди статистики программ по университетам;
- 17) Сортировка таблицы со статистикой направлений по университетам
- 18) Установка URL сайта, с которого парсер будет собирать данные;
- 19) Установка токена бота, к которому будет подключаться ИС;
- 20) Установка пути для экспортирующихся отчетов;
- 21) Установка пути для журналов;
- 22) Очистка окна живого журнала;

## Перечень используемых источников

Ниже перечислены источники, использовавшиеся при составлении документации для данного проекта:

Хабр. Использование диаграммы вариантов использования UML при проектировании программного обеспечения // [Электронный ресурс] – Режим доступа: <https://habr.com/ru/post/566218/>. Дата обращения: 28.02.2023.

Worldskills. Проектирование диаграммы состояний UML (statechart diagram) // [Электронный ресурс] – Режим доступа: <https://nationalteam.worldskills.ru/skills/proektirovanie-diagrammy-sostoyaniy-uml-statechart-diagram/>. Дата обращения: 5.03.2023.

IT-GOST.RU. Теория и практика UML. Диаграмма состояний // [Электронный ресурс] – Режим доступа: [http://it-gost.ru/articles/view\\_articles/97](http://it-gost.ru/articles/view_articles/97). Дата обращения: 5.03.2023

METANIT.COM. Полное руководство по языку программирования C# 10 и платформе .NET 6 // [Электронный ресурс] – Режим доступа: <https://metanit.com/sharp/tutorial/> . Дата обращения: 5.03.2023

Microsoft Learn. Общие сведения о WPF // [Электронный ресурс] – Режим доступа: <https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/introduction-to-wpf?view=netframeworkdesktop-4.8> . Дата обращения: 5.03.2023.

METANIT.COM. Введение в WPF // [Электронный ресурс] – Режим доступа: <https://metanit.com/sharp/wpf/1.php> . Дата обращения: 5.03.2023.

ИНТУИТ. Практикум 9: Пример технического задания для рецензирования // [Электронный ресурс] – Режим доступа:

<https://intuit.ru/studies/courses/2195/55/lecture/15050?page=2> . Дата обращения:  
1.04.2023.

## Листинг кода приложения

**Содержимое файла App.xaml.cs:**

```
using Microsoft.Data.Sqlite;

using System.Windows;

namespace botserver_standard
{
    public partial class App : Application
    {
        protected override void OnStartup(StartupEventArgs e)
        {
            base.OnStartup(e);

            // OnStartup code next:

            Stats.StartupTimeFixator();

            Settings.connString = "Data Source = appDB.db";

            //восстановление структуры бд при необходимости

            try
            {
```

```
        DbWorker.DbQuerySilentSender(DbWorker.sqliteConn,
        DbWorker.dbStructureRessurrection);

    }

    catch

    {

        return;

    }

    finally

    {

        //чтение настроек

        using SqliteDataReader reader =
        DbWorker.SettingsReader(DbWorker.readSettings, DbWorker.sqliteConn);

    }

}

}
```

**Содержимое файла AskingPassword.xaml.cs:**

```

using System.Windows;

namespace botserver_standard
{
    public partial class AskingPassword : Window
    {
        public AskingPassword()
        {
            InitializeComponent();
        }

        private void NextBtn_Click(object sender, RoutedEventArgs e)
        {
            if (EnterPwdBox.Password == Settings.pwd) //если введенный
пароль корректен
            {
                this.DialogResult = true;
            }
        }

        public string Password
        {
            get { return EnterPwdBox.Password; }
        }
    }
}

```

**Содержимое файла Card.cs:**

```
using System.Collections.Generic;

namespace botserver_standard
{

    public class Card
    {
        public static List<Card> cards = new(); // упорядоченный набор
карточек (экземпляров классов). Нечитабельно при отладке(?)

        public int Id { get; set; }
        public string UniversityName { get; set; }
        public string ProgramName { get; set; }
        public string Level { get; set; }
        public string ProgramCode { get; set; }
        public string StudyForm { get; set; }
        public string Duration { get; set; }
        public string StudyLang { get; set; }
        public string Curator { get; set; }
        public string PhoneNumber { get; set; }
        public string Email { get; set; }
        public string Cost { get; set; }
```

```
public Card(int id, string universityName, string programName, string
level, string studyForm, string programCode, string duration, string studyLang,
string curator, string phoneNumber, string email, string cost)
{
    this.Id = id;
    this.UniversityName = universityName;
    this.ProgramName = programName;
    this.Level = level.ToLower();
    this.StudyForm = studyForm.ToLower();
    this.ProgramCode = programCode;
    this.Duration = duration.ToLower();
    this.StudyLang = studyLang.ToLower();
    this.Curator = curator;
    this.PhoneNumber = phoneNumber;
    this.Email = email;
    this.Cost = cost;
}
}
}
```



**Содержимое файла ChangeDefaultPwd.xaml.cs:**

```

using Microsoft.Data.Sqlite;
using System;
using System.Windows;

namespace botserver_standard
{
    public partial class ChangeDefaultPwd : Window
    {
        public ChangeDefaultPwd()
        {
            InitializeComponent();
            EnterPwdBox.MaxLength = 50;
            EnterPwdBox_Repeated.MaxLength = 50;
        }

        int rowsChanged;
        private void NextBtn_Click(object sender, RoutedEventArgs e)
        {
            string changeDefaultPwdQuery = $"UPDATE Settings SET pwd =
'{EnterPwdBox.Password}';";
            string setCheckboxQuery = $"UPDATE Settings SET pwdIsUsing =
'True';";

            DateTime updateMoment;
            string previousPwdWrite = $"INSERT INTO Passwords_history
(timestamp, oldPassword) VALUES ('{updateMoment = DateTime.Now}',
'{Settings.pwd}');"; //установка пароля по умолчанию и отключение его
запроса при старте

```

```

        if (EnterPwdBox.Password == EnterPwdBox_Repeated.Password
&& EnterPwdBox_Repeated.Password != Settings.pwd) // если юзер не ошибся
и пароль не равен предыдущему
        {
            rowsChanged =
DbWorker.DbQuerySilentSender(DbWorker.sqliteConn,
changeDefaultPwdQuery); //смена дефолтного пароля
//IsSetted?
            if (rowsChanged is 1) //если удачно
            {
                DbWorker.DbQuerySilentSender(DbWorker.sqliteConn,
previousPwdWrite); //запись истории паролей

                DbWorker.DbQuerySilentSender(DbWorker.sqliteConn,
setCheckboxQuery); //установка галки на использование пароля на старте

                using SqliteDataReader reader =
DbWorker.SettingsReader(DbWorker.readSettings, DbWorker.sqliteConn);
//обновление настроек приложения из бд

                MessageBox.Show("Пароль успешно установлен. На вкладке
\"Settings\" вы можете отключить его использование.", "Notice");

                this.DialogResult = true;
            }

```

```
else
    { MessageBox.Show("Непредвиденная ошибка", "Error"); }
}
else
{
    MessageBox.Show("Вы пытаетесь установить пароль по
умолчанию, либо пароли не совпадают", "Notice");
}

}

}
```

**Содержимое файла ConsoleWorker.cs:**

```

using System;
using System.IO;

namespace botserver_standard
{
    internal class ConsoleWorker
    {
        public static void CardOutputter()
        {
            MainWindow.AllocConsole();

            TextWriter stdOutWriter = new
StreamWriter(Console.OpenStandardOutput(), Console.OutputEncoding) {
AutoFlush = true };

            TextWriter stdErrWriter = new
StreamWriter(Console.OpenStandardError(), Console.OutputEncoding) {
AutoFlush = true };

            TextReader strInReader = new
StreamReader(Console.OpenStandardInput(), Console.InputEncoding);

            Console.SetOut(stdOutWriter);
            Console.SetError(stdErrWriter);
            Console.SetIn(strInReader);

            foreach (var item in Card.cards)
            {
                Console.WriteLine($"{item.Id} | {item.UniversityName} |
{item.ProgramName} | {item.Level} | " +

```

```
        $"{item.ProgramCode} | {item.StudyForm} | {item.Duration} |  
        {item.StudyLang} | " +  
        $"{item.Curator} | {item.PhoneNumber} | {item.Email} |  
        {item.Cost}");  
  
    }  
  
    Console.ReadKey();  
  
    MainWindow.FreeConsole();  
    }  
    }  
}
```

**Содержимое файла DatagridControls.cs:**

```
using System.Collections.Generic;
using System.IO;
using System.Windows;
using System.Windows.Input;

namespace botserver_standard
{
    public partial class MainWindow : Window
    {
        //parsedCards
        private void SearchByProgramName_KeyUp(object sender,
KeyEventArgs e)
        {
            List<Card> searchResult = new();
            string programNameFrag = SearchByProgramName.Text;
            foreach (var item in cardsView)
            {
                if (item.ProgramName.ToLower().Contains(programNameFrag))
                {
                    searchResult.Add(item);
                }
                else { continue; }
            }
            parsedCardsGrid.ItemsSource = searchResult;
        }
    }
}
```

```

private void SearchByUniversity_KeyUp(object sender, KeyEventArgs
e)
{
    List<Card> searchResult = new();
    string universityNameFrag = SearchByUniversity.Text;
    foreach (var item in cardsView)
    {
        if
(item.UniversityName.ToLower().Contains(universityNameFrag))
        {
            searchResult.Add(item);
        }
        else { continue; }
    }
    parsedCardsGrid.ItemsSource = searchResult;
}

private void CardsExportBtn_Click(object sender, RoutedEventArgs e)
{
    StreamWriter parsedCardsExport =
new(Settings.datagridExportPath);
    parsedCardsExport.WriteLine("Id Название
университета\tНазвание программы\tКод программы\tУровень
обучения\tФорма обучения\tДлительность обучения\tЯзык
обучения\tКуратор\tНомер телефона\tПочта\tСтоимость");
    foreach (var item in cardsView)
    {

```

```

parsedCardsExport.WriteLine($"{item.Id}\t{item.UniversityName}\t{item.Progra
mName}\t{item.ProgramCode}\t{item.Level}\t{item.StudyForm}\t{item.Duration
}\t{item.StudyLang}\t{item.Curator}\t{item.PhoneNumber}\t{item.Email}\t{item.
Cost}");

```

```

    }
    parsedCardsExport.Close();
}

```

```

//parsedUniversities

```

```

private void SearchByUniversityName_KeyUp(object sender,
KeyEventArgs e)
{
    List<UniversityEntryFreq> searchResult = new();
    string universityNameFrag = SearchByUniversityName.Text;
    foreach (var item in UniversityEntryFreq.universitiesFreqList)
    {
        if
(item.UniversityName.ToLower().Contains(universityNameFrag))
        {
            searchResult.Add(item);
        }
        else { continue; }
    }
    parsedUniversitiesGrid.ItemsSource = searchResult;
}

```



```

private void SearchByUniversityFreq_KeyUp(object sender,
KeyEventArgs e)
{
    List<UniversityEntryFreq> searchResult = new();
    string universityNameFrag = SearchByUniversityFreq.Text;
    foreach (var item in UniversityEntryFreq.universitiesFreqList)
    {
        if (item.Count.ToString().Contains(universityNameFrag))
        {
            searchResult.Add(item);
        }
        else { continue; }
    }
    parsedUniversitiesGrid.ItemsSource = searchResult;
}

```

```

private void UniversitiesExportBtn_Click(object sender,
RoutedEventArgs e)
{
    StreamWriter parsedUniversitiesExport =
new("exportUniversities.txt");
    parsedUniversitiesExport.WriteLine("Id Название
университета\tКоличество программ");
    foreach (var item in UniversityEntryFreq.universitiesFreqList)
    {
        parsedUniversitiesExport.WriteLine($"{item.UniversityName}\t{item.Count}");
    }
    parsedUniversitiesExport.Close();
}

```

```

    }
}
}

```

### Содержимое файла DbWorker.cs:

```

using Microsoft.Data.Sqlite;
using System;

namespace botserver_standard
{
    internal class DbWorker
    {
        public static bool pwdSetResult;
        public static SqliteConnection sqliteConn = new(Settings.connString);

        //восстановление структуры БД, если файл не найден

        public static readonly string dbStructureRessurrection =
            "CREATE TABLE IF NOT EXISTS Received_messages (username
            TEXT, is_bot INTEGER, first_name TEXT, last_name TEXT, language_code
            TEXT, chat_id INTEGER, message_id INTEGER, message_date TEXT,
            chat_type TEXT, message_content BLOB);" +
            "\r\nCREATE TABLE IF NOT EXISTS Settings (logPath TEXT,
            connString TEXT, botToken TEXT, pwd TEXT, pwdIsUsing TEXT, prsFilePath
            TEXT);" +
            "\r\nCREATE TABLE IF NOT EXISTS Cards (id INTEGER NOT
            NULL UNIQUE, universityName TEXT, programName TEXT, programCode

```

TEXT, level TEXT, studyForm TEXT, duration TEXT, studyLang TEXT, curator TEXT, phoneNumber TEXT, email TEXT, cost TEXT, PRIMARY KEY(id)) WITHOUT ROWID" +

"\r\nCREATE TABLE IF NOT EXISTS Session\_duration (startup\_time TEXT, shutdown\_time TEXT, total\_uptime TEXT);" +

"\r\nCREATE TABLE IF NOT EXISTS Universities (id INTEGER NOT NULL, universityName TEXT);" +

"\r\nCREATE TABLE IF NOT EXISTS Directions (id INTEGER NOT NULL, directionName TEXT);" +

"\r\nCREATE TABLE IF NOT EXISTS Programs (id INTEGER NOT NULL, programName TEXT);" +

"\r\nCREATE TABLE IF NOT EXISTS Parsing\_history (timestamp TEXT, parsingStart TEXT, parsingEnd TEXT, parsingResult INTEGER);" +

"\r\nCREATE TABLE IF NOT EXISTS Passwords\_history (timestamp TEXT, oldPassword TEXT);";

public static string readTokenFromDb = "SELECT botToken FROM Settings";

public static readonly string received\_messagesConsoleOutput = "SELECT \* FROM Received\_messages";

public static readonly string readSettings = "SELECT \* FROM Settings";

public static int DbQuerySilentSender(SqliteConnection sqliteConn, string queryText) //no feedback

```
{
    sqliteConn.Open();
```

```

        SqliteCommand command = new()
        {
            Connection = sqliteConn, //соединение для выполнения запроса
            CommandText = queryText //текст запроса
        };
        int rowsChanged = command.ExecuteNonQuery(); //выполнение
запроса и возврат количества измененных строк
        sqliteConn.Close(); //закрытие соединения
        return rowsChanged;
    }

    public static SqliteDataReader SettingsReader(string queryText,
SqliteConnection sqliteConn) //оновление настроек из бд
    {
        sqliteConn.Open(); //открытие соединения
        SqliteCommand command = new() //инициализация экземпляра
SqliteCommand
        {
            Connection = sqliteConn, //соединение для выполнения запроса
            CommandText = queryText //текст запроса
        };
        SqliteDataReader reader = command.ExecuteReader();

        if (reader.HasRows) // если есть строки
        {
            while (reader.Read()) // построчное чтение данных
            {
                Settings.fileLoggerPath =
Convert.ToString(reader["fileLoggerPath"]);
            }
        }
    }

```

```
        Settings.callbackLoggerPath =  
Convert.ToString(reader["callbackLoggerPath"]);  
        Settings.botToken = Convert.ToString(reader["botToken"]);  
        Settings.pwd = Convert.ToString(reader["pwd"]);  
        Settings.pwdIsUsing =  
Convert.ToBoolean(reader["pwdIsUsing"]);  
        Settings.datagridExportPath =  
Convert.ToString(reader["datagridExportPath"]);  
        Settings.parsingUrl = Convert.ToString(reader["parsingUrl"]);  
  
    }  
}  
sqliteConn.Close();  
return reader;  
}  
  
}  
}
```

**Содержимое файла MainTab.cs:**

```
using System;

using System.Collections.Generic;

using System.IO;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Threading;

using System.Windows;

using Telegram.Bot.Exceptions;

using Telegram.Bot.Types;

using Telegram.Bot;

using Telegram.Bot.Polling;

using Telegram.Bot.Types.ReplyMarkups;

using System.Linq;

namespace botserver_standard

{

    public partial class MainWindow : Window

    {

        //maintab methods
```

```

static string? selectedLevel;

static string? selectedUniversity;

static string? selectedProgram;

string? firstname;

private async void BotStartBtn_Click(object sender, RoutedEventArgs
e)

{

    LiveLogOutput.Clear();


    using CancellationTokenSource OnBotLoadCts = await
OnBotLoadMsg();


    // отправка запроса отмены для остановки

    OnBotLoadCts.Cancel();


    var receiverOptions = new ReceiverOptions

    {

        AllowedUpdates = { }, // receive all update types

    };


    await Task.Factory.StartNew(() =>
TgBot.botClient.StartReceiving(updateHandler: HandleUpdateAsync,

```

pollingErrorHandler:

HandleErrorAsync,

cancellationToken:

TgBot.MainBotCts.Token, receiverOptions: receiverOptions)); //ok

```

        async Task HandleUpdateAsync(ITelegramBotClient botClient,
        Update update, CancellationToken cancellationToken)

```

```

    {

```

```

        Message message = update.Message;

```

```

        if (message is null) { goto Eight; }

```

```

        #region sqlQueries править запросы на запись в бд

```

```

        //запись принятых сообщений в бд

```

```

        string recievedMessageToDbQuery = $"INSERT INTO
        Received_messages(username, is_bot, first_name, last_name, language_code,
        chat_id, message_id, message_date, chat_type, message_content) " +

```

```

        $"VALUES('@{message.Chat.Username}', '0',
        '{message.Chat.FirstName}', '{message.Chat.LastName}', 'ru', '{message.Chat.Id}',
        '{message.MessageId}', '{DateTime.Now}', '{message.Chat.Type}',
        '{message.Text}')";

```

```

        //запись принятых фотографий в бд

```

```

        string recievedPhotoMessageToDbQuery = $"INSERT INTO
        Received_messages(username, is_bot, first_name, last_name, language_code,
        chat_id, message_id, message_date, chat_type, message_content) " +

```



```

        $"VALUES('@{message.Chat.Username}', '0',
        '{message.Chat.FirstName}', '{message.Chat.LastName}', 'ru', '{message.Chat.Id}',
        '{message.MessageId}', '{DateTime.Now}', '{message.Chat.Type}',
        '{message.Photo}')";

```

```

        #endregion

```

```

        if (update.Type is
        Telegram.Bot.Types.Enums.UpdateType.Message && message.Text is null)
        //suggestion if recieved not text message

```

```

        {

```

```

            await botClient.SendTextMessageAsync(chatId:
            message.Chat.Id, text: $"Пожалуйста, выберите один из доступных
            вариантов:", replyMarkup: TelegramBotKeypads.mainMenuKeypad,
            cancellationToken: cancellationToken);

```

```

            LiveLogger_message(message); // живой лог

```

```

            FileLogger_message(message, message.Text, message.Chat.Id,
            Settings.fileLoggerPath); // логгирование в файл

```

```

            return;

```

```

        }

```

```

        if (update.Type is
        Telegram.Bot.Types.Enums.UpdateType.Message && message.Text.ToLower()
        == "/start") //if recieved Message update type

```

```

        {

```

```

    firstname = update.Message.Chat.FirstName;

    if (message.Text.ToLower() == "/start") //if recieved this text
    {
        LiveLogger_message(message); // живой лог

        FileLogger_message(message, message.Text,
message.Chat.Id, Settings.fileLoggerPath); // логгирование в файл

        DbWorker.DbQuerySilentSender(DbWorker.sqliteConn,
recievedMessageToDbQuery);

        await botClient.SendTextMessageAsync(chatId:
message.Chat.Id, text: $"Добро пожаловать, {firstname}!", replyMarkup:
TelegramBotKeypads.mainMenuKeypad, cancellationToken: cancellationToken);

        return;
    }

    else if (message.Text is not null && message.Text.ToLower() is
not "/start")
    {
        await botClient.SendTextMessageAsync(chatId:
message.Chat.Id, text: $"Пожалуйста, выберите один из доступных
вариантов:", replyMarkup: TelegramBotKeypads.mainMenuKeypad,
cancellationToken: cancellationToken);

        LiveLogger_message(message); // живой лог

```

```

        FileLogger_message(message, message.Text,
message.Chat.Id, Settings.fileLoggerPath); // логгирование в файл

```

```

        DbWorker.DbQuerySilentSender(DbWorker.sqliteConn,
recievedMessageToDbQuery);

```

```

        return;

```

```

    }

```

```

        await botClient.SendTextMessageAsync(chatId:
message.Chat.Id, text: $"Добро пожаловать, {firstname}!", replyMarkup:
TelegramBotKeypads.mainMenuKeypad, cancellationToken: cancellationToken);

```

```

    }

```

Eight:

```

        if (update.Type is
Telegram.Bot.Types.Enums.UpdateType.CallbackQuery)

```

```

        {

```

```

            if (update.CallbackQuery.Data is "toHome") //действия при
нажатии На главную

```

```

            {

```

```

                string telegramMessage = $"Добро пожаловать,
{firstname}!";

```

```

                await

```

```

botClient.EditMessageTextAsync(update.CallbackQuery.Message.Chat.Id,
update.CallbackQuery.Message.MessageId, telegramMessage, replyMarkup:
TelegramBotKeypads.mainMenuKeypad, parseMode:

```

```
Telegram.Bot.Types.Enums.ParseMode.Html, cancellationToken:
cancellationToken);

    }
```

```
        if (update.CallbackQuery.Data is "programChoose") //если
ответ был programChoose, то изменить сообщение на следующее...
```

```
        {

            string telegramMessage = "Выберите желаемый уровень
подготовки:";

```

```
            await
botClient.EditMessageTextAsync(update.CallbackQuery.Message.Chat.Id,
update.CallbackQuery.Message.MessageId, telegramMessage, replyMarkup:
TelegramBotKeypads.levelChoosingKeypad, parseMode:
Telegram.Bot.Types.Enums.ParseMode.Html, cancellationToken:
cancellationToken);

```

```
        }
```

```
        if (update.CallbackQuery.Data.Contains("_level")) //если ответ
содержал в себе level, то изменить сообщение на следующее...
```

```
        {

            selectedLevel =
update.CallbackQuery.Data.Replace("_level", string.Empty) as string;

            //UniversityEntryFreq.universitiesFreqList;

```

```

List<InlineKeyboardButton> parsedUniversitiesButtons =
new(); //

        foreach (var item in
UniversityEntryFreq.universitiesFreqList)
        {

parsedUniversitiesButtons.Add(InlineKeyboardButton.WithCallbackData(text:
item.UniversityName, callbackData: Convert.ToString(item.UniversityName) +
"_university"));

        }

parsedUniversitiesButtons.Add(InlineKeyboardButton.WithCallbackData(text:
"🏠", callbackData: "toHome"));

        var dynamicUniversityChoosingKeypad = new
InlineKeyboardMarkup(parsedUniversitiesButtons);

        string telegramMessage = "Пожалуйста, выберите
необходимый университет:";

        await
botClient.EditMessageTextAsync(update.CallbackQuery.Message.Chat.Id,
update.CallbackQuery.Message.MessageId, telegramMessage, replyMarkup:
dynamicUniversityChoosingKeypad, parseMode:
Telegram.Bot.Types.Enums.ParseMode.Html, cancellationToken:
cancellationToken);

```

```

    }

    if (update.CallbackQuery.Data.Contains("_university"))
    {

        selectedUniversity =
update.CallbackQuery.Data.Replace("_university", string.Empty) as string;

        string telegramMessage = "Подобранные программы
обучения:\n\n";

        // фильтрация карточек на основании выборов
абитуриента

        List<Card> filteredCardsByEnrollee = new();

        foreach (var item in cardsView) //переписать цикл на фор
для нормальной нумерации направлений

        {

            if (selectedLevel == item.Level && selectedUniversity ==
item.UniversityName)

                filteredCardsByEnrollee.Add(item);

        }

        //составление сообщения с номерами направлений

        foreach (var item in filteredCardsByEnrollee)

```

```

    {
        telegramMessage +=
$"{item.Id}:\t{item.ProgramName}\n";
    }

//извлечение идентификаторов из отфильтрованных
направлений

List<string> cardIds = new();

foreach (var item in filteredCardsByEnrollee)
{
    cardIds.Add(Convert.ToString(item.Id));
}

//генерация кнопок на основе отфильтрованных карточек

var filteredUniversitiesButtons = new
List<List<InlineKeyboardButton>>();

for (int i = 0; i < cardIds.Count; i += 3)

    filteredUniversitiesButtons.Add(new
List<InlineKeyboardButton>(cardIds.Skip(i).Take(3).Select(id =>
InlineKeyboardButton.WithCallbackData(id))));

var dynamicProgramChoosingKeypad = new
InlineKeyboardMarkup(filteredUniversitiesButtons);

```

```
//отправка сообщения
```

```
await
```

```
botClient.EditMessageTextAsync(update.CallbackQuery.Message.Chat.Id,
update.CallbackQuery.Message.MessageId, telegramMessage, replyMarkup:
dynamicProgramChoosingKeypad, parseMode:
Telegram.Bot.Types.Enums.ParseMode.Html, cancellationToken:
cancellationToken);
```

```
}
```

```
//отправка финального сообщения с данными о выбранном
направлении
```

```
if (int.TryParse(update.CallbackQuery.Data, out int
isIntegerValue) is true)
```

```
{
```

```
selectedProgram = update.CallbackQuery.Data;
```

```
Card finalSelectedCard =
cardsView[Convert.ToInt32(selectedProgram)];
```

```
string telegramMessage = $"Мы подобрали для вас
следующее направление:\n" +
```

```
 $"Университет:\t{finalSelectedCard.UniversityName}\n" +
```

```
 $"Программа:\t{finalSelectedCard.ProgramName}\n" +
```



```

        $"Код
программы:\t{finalSelectedCard.ProgramCode}\n" +

        $"Уровень
образования:\t{finalSelectedCard.Level}\n" +

        $"Форма
обучения:\t{finalSelectedCard.StudyForm}\n" +

        $"Длительность
обучения:\t{finalSelectedCard.Duration}\n" +

        $"Язык
обучения:\t{finalSelectedCard.StudyLang}\n" +

        $"Куратор:\t{finalSelectedCard.Curator}\n" +

        $"Номер
телефона:\t{finalSelectedCard.PhoneNumber}\n" +

        $"Почта:\t{finalSelectedCard.Email}\n" +

        $"Стоимость
обучения:\t{finalSelectedCard.Cost}";

```

```

InlineKeyboardMarkup lastButtonsKeypad = new(
    new[]
    {
        // first row
        new[]
        {

```

```

        InlineKeyboardButton.WithUrl(text: "✉", url:
$"mailto:{finalSelectedCard.Email}"),

        },

        // second row

        new[]

        {

            InlineKeyboardButton.WithCallbackData(text: "↩",
callbackData: "toHome"),

            },

        });

    await

    botClient.EditMessageTextAsync(update.CallbackQuery.Message.Chat.Id,
update.CallbackQuery.Message.MessageId, telegramMessage, replyMarkup:
lastButtonsKeypad, parseMode: Telegram.Bot.Types.Enums.ParseMode.Html,
cancellation_token: cancellationToken);

    LiveLogger_callBack(update.CallbackQuery,
finalSelectedCard);

    ChoicesToDb(update.CallbackQuery, finalSelectedCard);

    FileLogger_callBack(update.CallbackQuery,
Settings.callbackLoggerPath, finalSelectedCard);

```

```
}
```

```
}
```

```
}
```

```
Task HandleErrorAsync(ITelegramBotClient botClient, Exception
exception, CancellationToken cancellationToken) //обработчик ошибок API
{
    var ErrorMessage = exception switch
    {
        ApiRequestException apiRequestException
            => $"Telegram API
Error:\n[{apiRequestException.ErrorCode}]\n{apiRequestException.Message}\nTi
mestamp: {DateTime.Now}",
        _ => exception.ToString()
    };

    Dispatcher.Invoke(() =>
    {
        return LiveLogOutput.Text += $"{ErrorMessage}\n" + "-----
-----
\n";

    });

    return Task.CompletedTask;
}
```

```
}
```

```
}
```

```
public async Task<CancellationTokenSource> OnBotLoadMsg()
{
    CancellationTokenSource OnBotLoadCts = new();

    User me = await TgBot.botClient.GetMeAsync();

    LiveLogOutput.Text += $"Начато прослушивание бота
@{me.Username} с именем {me.FirstName} в {DateTime.Now}\n";

    LiveLogOutput.Text += "-----
-----\n";

    return OnBotLoadCts;
}
```

```
public void LiveLogger_message(Message? message)
{
    Dispatcher.Invoke(() =>
    {
        return LiveLogOutput.Text += $"Получено сообщение
'{message.Text}' от пользователя @{message.Chat.Username} так же
известного, как {message.Chat.FirstName} {message.Chat.LastName} в чате
{message.Chat.Id} в {DateTime.Now}.\n" +
```

```

"-----
-----\n";

    });

}

public void LiveLogger_callBack(CallbackQuery callbackQuery, Card
card)

{
    Dispatcher.Invoke(() =>
    {
        return LiveLogOutput.Text += $"Пользователь
@{callbackQuery.From.Username}, так же известный, как
{callbackQuery.From.FirstName} {callbackQuery.From.LastName} выбрал
уровень {selectedLevel}, университет {selectedUniversity} и программу
{card.ProgramName} в {DateTime.Now}\n" +
        "-----
-----\n";

    });

}

public static async void FileLogger_message(Message message, string
messageText, long chatId, string logPath) //логгирование полученных
сообщений в файл

{

```

```

        using StreamWriter logWriter = new(logPath, true);
//инициализация экземпляра Streamwriter

        await logWriter.WriteLineAsync($"Получено сообщение
'{messageText}' от пользователя @{message.Chat.Username}, так же
известного, как {message.Chat.FirstName} {message.Chat.LastName} в чате
{chatId} в {DateTime.Now}"); //эхо

        await logWriter.WriteLineAsync("-----
-----");

    }

    public static async void FileLogger_callBack(CallbackQuery
callbackQuery, string logPath, Card card) //логгирование callback в файл
    {

        using StreamWriter logWriter = new(logPath, true);
//инициализация экземпляра Streamwriter

        await logWriter.WriteLineAsync($"Пользователь
@{callbackQuery.From.Username}, так же известный, как
{callbackQuery.From.FirstName} {callbackQuery.From.LastName} выбрал
уровень {selectedLevel}, университет {selectedUniversity} и программу
{card.ProgramName} в {DateTime.Now}");

        await logWriter.WriteLineAsync("-----
-----");

```

```
}
```

```
public void ChoicesToDb(CallbackQuery callbackQuery, Card card)
```

```
{
```

```
    string query = $"INSERT INTO Fixated_choices (username, fname, lname, choisedLevel, choisedUniversity, choisedProgram, timestamp) " +
```

```
        $"VALUES ('@{callbackQuery.From.Username}', '{callbackQuery.From.FirstName}', '{callbackQuery.From.LastName}', '{selectedLevel}', '{selectedUniversity}', '{card.ProgramName}', '{DateTime.Now}')";
```

```
    DbWorker.DbQuerySilentSender(DbWorker.sqliteConn, query);
```

```
    //запись истории паролей
```

```
}
```

```
public static async Task PreparedMessageSender(ITelegramBotClient botClient, string sendingMessage, long chatId, CancellationToken cancellationToken)
```

```
{
```

```
    await botClient.SendTextMessageAsync(chatId: chatId,
```

```
        text: sendingMessage,
```

```
        cancellationToken: cancellationToken);
```

```
}
```

```

    public static async Task<Task>
    ParrotedMessageSender(ITelegramBotClient botClient, Message? message, long?
    chatId, CancellationToken cancellationToken) //отправка пользователю текста
    его сообщения

    {
        if (message is not null)
        {
            await botClient.SendTextMessageAsync(
                chatId: chatId,
                text: $"I received the following message:\n{message.Text}",
                cancellationToken: cancellationToken);
        }

        else await ErrorToChatSender(botClient, chatId, cancellationToken);
        return Task.CompletedTask;
    }

    public static async Task ErrorToChatSender(ITelegramBotClient
    botClient, long? chatId, CancellationToken cancellationToken)

    {
        await botClient.SendTextMessageAsync(
            chatId: chatId,
            text: $"botserver_ standard error. message.Text is null?",

```



```
cancellationToken: cancellationToken);  
  
}  
  
#region кнопки  
  
private void StopBotBtn_Click(object sender, RoutedEventArgs e)  
{  
  
    Stats.ShutdownTimeFixator();  
  
    Stats.UpTimeWriter();  
  
    TgBot.MainBotCts.Cancel();  
  
    LiveLogOutput.Clear();  
  
    LiveLogOutput.Text = "Бот был остановлен.";  
  
}  
  
  
private void StopExitBotBtn_Click(object sender, RoutedEventArgs e)  
{  
  
    Stats.ShutdownTimeFixator();  
  
    Stats.UpTimeWriter();  
  
    TgBot.MainBotCts.Cancel();  
  
    Environment.Exit(0);  
  
}  
  
  
private void CmdOpenBtn_Click(object sender, RoutedEventArgs e)
```

```
{  
    Task.Factory.StartNew(() => ConsoleWorker.CardOutputter());  
}  
  
private void LogExportBtn_Click(object sender, RoutedEventArgs e)  
{  
    string pathPart = $"livelog_{DateTime.Now}.txt".Replace(":", "_");  
    string parserOutPath = Settings.baseLogPath + pathPart;  
    StreamWriter parserExport = new(parserOutPath);  
    parserExport.WriteLine(LiveLogOutput.Text);  
    parserExport.Close();  
}  
  
private void LogClearBtn_Click(object sender, RoutedEventArgs e)  
{  
    LiveLogOutput.Clear();  
}  
#endregion  
}  
}
```

**Содержимое файла MainWindow.xaml.cs:**

```
using System;
using System.IO;
using System.Runtime.InteropServices;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Media;

namespace botserver_standard
{
    public partial class MainWindow : Window
    {
        //импорт библиотек для запуска консоли
        [DllImport("kernel32.dll")]
        public static extern bool AllocConsole();
        [DllImport("kernel32.dll")]
        public static extern bool FreeConsole();

        public MainWindow()
        {
            InitializeComponent();

            if (Settings.pwdIsUsing is true)
            {
                AskingPassword askPwd = new();
                if (askPwd.ShowDialog() == true)
                {
                    MessageBox.Show("Добро пожаловать!");
                }
            }
        }
    }
}
```

```

        UseThisPwdCheckbox.IsChecked = false; //обновление
состояния чекбокса в окне
    }
    else
    {
        MessageBox.Show("Операция отменена.");
        Environment.Exit(0);
    }
}

UseThisPwdCheckbox.IsChecked = Settings.pwdIsUsing;

if (Settings.pwd is
"YtcPoTIZPA0WpUdc~SMCaTjL7Kvt#ne3k*{Tb|H2Kx4t227gXy") // setting
new pwd if now setted default
{
    ChangeDefaultPwd changeDefaultPwd = new();
    changeDefaultPwd.ShowDialog();
    if (this.DialogResult is true)
    {
        UseThisPwdCheckbox.IsChecked = true;
    }
}
SetPwdBox.MaxLength = 50;
SetRepeatedPwdBox.MaxLength = 50;
Task.Factory.StartNew(() => CardParser(DbWorker.sqliteConn));
//ok
}

```

```
private void TabControl_SelectionChanged(object sender,
System.Windows.Controls.SelectionChangedEventArgs e)
{
    if (mainTab.IsSelected)
    {
        HomePic.Foreground = Brushes.RoyalBlue;
        ParserPic.Foreground = Brushes.LightGray;
        ParsedCardsPic.Foreground = Brushes.LightGray;
        ParsedUnivsPic.Foreground = Brushes.LightGray;
        SettingsPic.Foreground = Brushes.LightGray;
        AboutPic.Foreground = Brushes.LightGray;
    }

    if (parserTab.IsSelected)
    {
        HomePic.Foreground = Brushes.LightGray;
        ParserPic.Foreground = Brushes.RoyalBlue;
        ParsedCardsPic.Foreground = Brushes.LightGray;
        ParsedUnivsPic.Foreground = Brushes.LightGray;
        SettingsPic.Foreground = Brushes.LightGray;
        AboutPic.Foreground = Brushes.LightGray;
    }

    if (parsedCardsTab.IsSelected)
    {
        HomePic.Foreground = Brushes.LightGray;
        ParserPic.Foreground = Brushes.LightGray;
        ParsedCardsPic.Foreground = Brushes.RoyalBlue;
        ParsedUnivsPic.Foreground = Brushes.LightGray;
    }
}
```

```
SettingsPic.Foreground = Brushes.LightGray;
AboutPic.Foreground = Brushes.LightGray;
}

if (parsedUniversitiesTab.IsSelected)
{
    HomePic.Foreground = Brushes.LightGray;
    ParserPic.Foreground = Brushes.LightGray;
    ParsedCardsPic.Foreground = Brushes.LightGray;
    ParsedUnivsPic.Foreground = Brushes.RoyalBlue;
    SettingsPic.Foreground = Brushes.LightGray;
    AboutPic.Foreground = Brushes.LightGray;
}

if (settingsTab.IsSelected)
{
    HomePic.Foreground = Brushes.LightGray;
    ParserPic.Foreground = Brushes.LightGray;
    ParsedCardsPic.Foreground = Brushes.LightGray;
    ParsedUnivsPic.Foreground = Brushes.LightGray;
    SettingsPic.Foreground = Brushes.RoyalBlue;
    AboutPic.Foreground = Brushes.LightGray;
}

if (aboutTab.IsSelected)
{
    HomePic.Foreground = Brushes.LightGray;
    ParserPic.Foreground = Brushes.LightGray;
    ParsedCardsPic.Foreground = Brushes.LightGray;
```

```
ParsedUnivsPic.Foreground = Brushes.LightGray;  
SettingsPic.Foreground = Brushes.LightGray;  
AboutPic.Foreground = Brushes.RoyalBlue;
```

```
}
```

```
}
```

```
}
```

```
}
```

**Содержимое файла Parser.cs:**

```
using HtmlAgilityPack;

using Microsoft.Data.Sqlite;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using System.Windows.Threading;

namespace botserver_standard

{

    public partial class MainWindow : Window

    {

        List<Card> cardsView = new();

        List<UniversityEntryFreq> universityFreqListView = new();

        public async void CardParser(SqliteConnection sqliteConn)

        {

            Dispatcher.Invoke(() =>

            {
```



```

ParserLogOutput.AppendText("-----
-----\n");

ParserLogOutput.Text += $"{DateTime.Now} | Парсер
запущен...\n";

});

Dispatcher.Invoke(() =>
{
    ParserLogOutput.Text += $"{DateTime.Now} | Получение
данных из сети...\n";

});

var parsingUrl = "https://studyintomsk.ru/programs-main/";

var web = new HtmlWeb();

HtmlDocument document;

document = web.Load(parsingUrl); //loading html

/*

/html/body/div[2]/div/div[3]/div[5]/select - программы подготовки

/html/body/div[3]/div[3] - карточки со сдвигами

/html/body/div[2]/div/div[3]/div[3]/select - вузы

/html/body/div[2]/div/div[3]/div[1]/select - уровни

/html/body/div[2]/div/div[3]/div[4]/select - языки

```

```

*/

Dispatcher.Invoke(() =>
{
    ParserLogOutput.Text += $"{DateTime.Now} | Выбор
узлов...\n";

});

if (document is null)
{
    return;
}

var cardsValue =
document.DocumentNode.SelectNodes("/html/body/div[3]/div[3]/div/div/div/div/d
iv");

string noTabsDoc = string.Empty; //первичная строка с сырыми
данными

Dispatcher.Invoke(() =>
{
    ParserLogOutput.Text += $"{DateTime.Now} | Обработка
полученных данных...\n";

});

foreach (var item in cardsValue)

```

```

{
    noTabsDoc += item.InnerText; //node is single row?
}

noTabsDoc = noTabsDoc.Replace("\t", "\n"); //замена табуляций

List<string> cardsList = new(); //лист с правильными данными,
идушими подряд

cardsList = noTabsDoc.Split('\n').ToList(); //построчная запись
данных (в том числе и мусора)

//удаление мусора из листа

Dispatcher.Invoke(() =>
{
    ParserLogOutput.Text += $"{DateTime.Now} | Очистка
полученных данных...\n";

});

#region data cleaning

string itemToRemove = "Уровень обучения";

cardsList.RemoveAll(x => x == itemToRemove);

itemToRemove = "Форма обучения";

cardsList.RemoveAll(x => x == itemToRemove);

```

```
itemToRemove = "Код программы ";  
cardsList.RemoveAll(x => x == itemToRemove);
```

```
itemToRemove = "Продолжительность";  
cardsList.RemoveAll(x => x == itemToRemove);
```

```
itemToRemove = "Степень или квалификация";  
cardsList.RemoveAll(x => x == itemToRemove);
```

```
itemToRemove = "Язык обучения";  
cardsList.RemoveAll(x => x == itemToRemove);
```

```
itemToRemove = "Куратор";  
cardsList.RemoveAll(x => x == itemToRemove);
```

```
itemToRemove = "за год обучения";  
cardsList.RemoveAll(x => x == itemToRemove);
```

```
itemToRemove = "Поступить";  
cardsList.RemoveAll(x => x == itemToRemove);
```

```
itemToRemove = "Нет подходящей программы?";
```

```
cardsList.RemoveAll(x => x == itemToRemove);
```

```
itemToRemove = "Напишите нам об этом и мы придумаем для  
вас индивидуальное решение.";
```

```
cardsList.RemoveAll(x => x == itemToRemove);
```

```
itemToRemove = "Получить решение";
```

```
cardsList.RemoveAll(x => x == itemToRemove);
```

```
itemToRemove = "Основная программа О программе";
```

```
cardsList.RemoveAll(x => x == itemToRemove);
```

```
//symbols
```

```
itemToRemove = "\n";
```

```
cardsList.RemoveAll(x => x == itemToRemove);
```

```
itemToRemove = "";
```

```
cardsList.RemoveAll(x => x == itemToRemove);
```

```
itemToRemove = string.Empty;
```

```
cardsList.RemoveAll(x => x == itemToRemove);
```

```
#endregion
```

//строки для передачи в атрибуты экземпляра класса

int Id = 0; // идентификатор экземпляра класса. Задать в бд?

//для прыжков по строкам всех карточек в листе

int row0 = 0;

int row1 = 1;

int row2 = 2;

int row3 = 3;

int row4 = 4;

int row5 = 5;

//int row6 = 6; //пропуск повторяющегося атрибута

int row7 = 7;

int row8 = 8;

int row9 = 9;

int row10 = 10;

int row11 = 11;

int cardCounter = 0;

int cardsTotalRows = cardsList.Count / 12;

Dispatcher.Invoke(() =>

```

{
    ParserLogOutput.Text += $"{DateTime.Now} | Запись
данных...\n";
});

foreach (string line in cardsList)
{
    Card.cards.Add(new Card(Id, cardsList[row0], cardsList[row1],
cardsList[row2],
cardsList[row3], cardsList[row4], cardsList[row5],
cardsList[row7], cardsList[row8], cardsList[row9],
cardsList[row10], cardsList[row11]));

    Id++;

    //прыжок на строки следующей карточки

    row0 += 12;

    row1 += 12;

    row2 += 12;

    row3 += 12;

    row4 += 12;

    row5 += 12;

    //row6 += 12; //пропуск повторяющегося атрибута

    row7 += 12;

    row8 += 12;

```

```
row9 += 12;
```

```
row10 += 12;
```

```
row11 += 12;
```

```
cardCounter++;
```

```
if (cardCounter == cardsTotalRows)
```

```
    break;
```

```
}
```

```
string clearCardsDb = "DELETE FROM Cards;";
```

```
DbWorker.DbQuerySilentSender(DbWorker.sqliteConn,  
clearCardsDb);
```

```
//запись полученных карточек в бд
```

```
foreach (var item in Card.cards)
```

```
{
```

```
    string cardsToDb = $"INSERT INTO Cards(id, universityName,  
programName, programCode, level, studyForm, duration, studyLang, curator,  
phoneNumber, email, cost) " +
```

```
    $"VALUES('{item.Id}', '{item.UniversityName}',  
'{item.ProgramName}', '{item.ProgramCode}', '{item.Level}', '{item.StudyForm}',  
'{item.Duration}', '{item.StudyLang}', '{item.Curator}', '{item.PhoneNumber}',  
'{item.Email}', '{item.Cost}')";
```



```
DbWorker.DbQuerySilentSender(DbWorker.sqliteConn,  
cardsToDb);
```

```
}
```

```
//вывод данных из бд на вкладку карточек
```

```
int id;
```

```
string? universityName;
```

```
string? programName;
```

```
string? level;
```

```
string? programCode;
```

```
string? studyForm;
```

```
string? duration;
```

```
string? studyLang;
```

```
string? curator;
```

```
string? phoneNumber;
```

```
string? email;
```

```
string? cost;
```

```
string queryText = "SELECT * FROM Cards";
```

```
sqliteConn.Open(); //открытие соединения
```

```
SqlCommand command = new() //инициализация экземпляра
```

```
SqlCommand
```

```

{
    Connection = sqliteConn, //соединение для выполнения запроса
    CommandText = queryText //текст запроса
};

SqliteDataReader reader = command.ExecuteReader();

if (reader.HasRows) // если есть строки
{
    while (reader.Read()) // построчное чтение данных
    {
        //public Card(int id, string universityName, string
programName, string level, string studyForm, string programCode, string duration,
string studyLang, string curator, string phoneNumber, string email, string cost)

        id = Convert.ToInt32(reader["Id"]);
        universityName = Convert.ToString(reader["universityName"]);
        programName = Convert.ToString(reader["programName"]);
        programCode = Convert.ToString(reader["programCode"]);
        level = Convert.ToString(reader["level"]);
        studyForm = Convert.ToString(reader["studyForm"]);
        duration = Convert.ToString(reader["duration"]);
        studyLang = Convert.ToString(reader["studyLang"]);
        curator = Convert.ToString(reader["curator"]);
    }
}

```

```

        phoneNumber = Convert.ToString(reader["phoneNumber"]);

        email = Convert.ToString(reader["email"]);

        cost = Convert.ToString(reader["cost"]);

        cardsView.Add(new Card(id, universityName, programName,
level, studyForm, programCode, duration, studyLang, curator, phoneNumber,
email, cost));

    }

}

sqliteConn.Close();

//выделение уникальных вузов

List<string> universitiesList = new();

int universityRow = 0;

int rowCounter = 0;

foreach (string line in cardsList)

{

    universitiesList.Add(cardsList[universityRow]);

    universityRow += 12;

    rowCounter++;

    if (rowCounter >= cardsTotalRows)

        break;

```

```

    }

    foreach (string item in universitiesList.Distinct())
    {
        UniversityEntryFreq.universitiesFreqList.Add(new
UniversityEntryFreq(item, universitiesList.Where(x => x == item).Count()));
    }

    string clearUniversitiesFreqDb = "DELETE FROM Universities;";

    DbWorker.DbQuerySilentSender(DbWorker.sqliteConn,
clearUniversitiesFreqDb);

    id = 0;

    //запись полученных карточек в бд

    foreach (var item in UniversityEntryFreq.universitiesFreqList)
    {
        string universitiesToDb = $"INSERT INTO Universities(id,
universityName, universityCount) " +

        $"VALUES('{id}', '{item.UniversityName}', '{item.Count}')";

        DbWorker.DbQuerySilentSender(DbWorker.sqliteConn,
universitiesToDb);

        id++;
    }

```

```
//вывод данных из бд на вкладку карточек
```

```
string? freqUniversityName;
```

```
int freqUniversityCount;
```

```
queryText = "SELECT * FROM Universities";
```

```
sqliteConn.Open(); //открытие соединения
```

```
SqliteCommand freqCommand = new() //инициализация  
экземпляра SqliteCommand
```

```
{
```

```
    Connection = sqliteConn, //соединение для выполнения запроса
```

```
    CommandText = queryText //текст запроса
```

```
};
```

```
SqliteDataReader freqReader = freqCommand.ExecuteReader();
```

```
if (freqReader.HasRows) // если есть строки
```

```
{
```

```
    while (freqReader.Read()) // построчное чтение данных
```

```
    {
```

```
        freqUniversityName =
```

```
Convert.ToString(freqReader["universityName"]);
```

```

        freqUniversityCount =
Convert.ToInt32(freqReader["universityCount"]);

        universityFreqListView.Add(new
UniversityEntryFreq(freqUniversityName, freqUniversityCount));
    }
}

sqliteConn.Close();

Dispatcher.Invoke(() =>
{
    parsedCardsGrid.ItemsSource = cardsView;
});

Dispatcher.Invoke(() =>
{
    parsedUniversitiesGrid.ItemsSource = universityFreqListView;
});

Dispatcher.Invoke(() =>
{
    ParserLogOutput.Text += $"{DateTime.Now} | Парсинг
окончен.\n{Card.cards.Count} карточек было
добавлено;\n{universityFreqListView.Count} университетов было
добавлено.\n";

```

```
ParserLogOutput.Text += "-----  
-----\n";  
  
    });  
  
    }  
  
    }  
  
    }
```

**Содержимое файла ParserTab.cs:**

```

using System;
using System.IO;
using System.Threading.Tasks;
using System.Windows;

namespace botserver_standard
{
    public partial class MainWindow : Window
    {
        private void ParserPeparseBtn_Click(object sender, RoutedEventArgs
e)
        {
            universityFreqListView.Clear();
            Card.cards.Clear();
            Dispatcher.Invoke(() =>
            {
                ParserLogOutput.Text += $"{DateTime.Now} | Reparsing...\n";

            });
            Task.Factory.StartNew(() => CardParser(DbWorker.sqliteConn));
//ok
        }

        private void ParserExportBtn_Click(object sender, RoutedEventArgs e)
        {
            //string parserOutPath = Settings.baseLogPath +
            $"parser_{DateTime.Now}.txt";

```



```
string pathPart = $"parser_{DateTime.Now}.txt".Replace(":", "_");  
string parserOutPath = Settings.baseLogPath + pathPart;  
StreamWriter parserExport = new(parserOutPath);  
parserExport.WriteLine(ParserLogOutput.Text);  
parserExport.Close();  
    }  
}  
}
```

**Содержимое файла Settings.cs:**

```

namespace botserver_standard
{
    internal class Settings
    {
        public static string? fileLoggerPath = null; // путь к логу (добавить
изменение пути лога в интерфейсе?)
        public static string? callbackLoggerPath = null;
        public static string? connString = null; //путь к бд. setted in app!
        public static string? botToken = null; //токен бота
        public static string? pwd; //пароль на запуск. Может быть отключен,
см. ниже
        public static bool pwdIsUsing = false; //пароль
используется(чекбокс)?
        public static string? datagridExportPath = null;
        public static string? parsingUrl = null; //URL страницы, подлежащей
парсингу

        public static string? baseLogPath =
"C:\\Users\\creat\\source\\repos\\botserver_standard\\bin\\Debug\\net6.0-
windows\\logs\\";
    }
}

```

**Содержимое файла SettingsTab.cs:**

```

using Microsoft.Data.Sqlite;
using System;
using System.Windows;

namespace botserver_standard
{
    public partial class MainWindow : Window
    {
        private void SetTokenBtn_Click(object sender, RoutedEventArgs e)
        //ok
        {
            string query = $"UPDATE Settings SET botToken =
'{SettingsTokenInput.Text.Trim(' ')}';";
            int rowsChanged =
DbWorker.DbQuerySilentSender(DbWorker.sqliteConn, query);

            //IsChanged?
            if (rowsChanged is 1)
            {
                using SqliteDataReader reader =
DbWorker.SettingsReader(DbWorker.readSettings, DbWorker.sqliteConn); //from
db to fields

                string tokenLeftPart = SettingsTokenInput.Text[..^35]; //
:AAF3nNDIYNfryOulNHKtsxlhuGo_roxXYXI

```

```

        SettingsTokenInput.Text = $"Token has been changed to
{tokenLeftPart}";
    }
    else
    {
        SettingsTokenInput.Text = "Unforeseen error";
    }
}

```

```

private void SetMessageLogPathBtn_Click(object sender,
RoutedEventArgs e) //ok
{
    string query = $"UPDATE Settings SET fileLoggerPath =
'{MessagesLogRootInput.Text}';";
    int rowsChanged =
DbWorker.DbQuerySilentSender(DbWorker.sqliteConn, query);

    //IsChanged?
    if (rowsChanged is 1)
    {
        using SqlDataReader reader =
DbWorker.SettingsReader(DbWorker.readSettings, DbWorker.sqliteConn);

        MessagesLogRootInput.Text = $"file path has been setted.";
    }
    else
    {
        MessagesLogRootInput.Text = "Unforeseen error";
    }
}

```

```

    }

    private void SetDatagridExportPathBtn_Click(object sender,
RoutedEventArgs e)
    {
        string query = $"UPDATE Settings SET datagridExportPath =
'{datagridExportPath.Text}';";
        int rowsChanged =
DbWorker.DbQuerySilentSender(DbWorker.sqliteConn, query);

        //IsChanged?
        if (rowsChanged is 1)
        {
            using SqliteDataReader reader =
DbWorker.SettingsReader(DbWorker.readSettings, DbWorker.sqliteConn);

            datagridExportPath.Text = $"file path has been setted.";
        }
        else
        {
            datagridExportPath.Clear();
            datagridExportPath.Text += "Unforseen error";
        }
    }

    private void ChoicesLogRootInputSetBtn_Click(object sender,
RoutedEventArgs e)
    {

```

```

        string query = $"UPDATE Settings SET callbackLoggerPath =
'{ChoicesLogRootInput.Text}';";
        int rowsChanged =
        DbWorker.DbQuerySilentSender(DbWorker.sqliteConn, query);

        //IsChanged?
        if (rowsChanged is 1)
        {
            using SqliteDataReader reader =
            DbWorker.SettingsReader(DbWorker.readSettings, DbWorker.sqliteConn);

            ChoicesLogRootInput.Text = $"Choices log path has been
setted.";
        }
        else
        {
            ChoicesLogRootInput.Text = "Unforseen error";
        }
    }

    private void ParsingUrlSetBtn_Click(object sender, RoutedEventArgs
e)
    {
        string query = $"UPDATE Settings SET parsingUrl =
'{UrlSet.Text}';";
        int rowsChanged =
        DbWorker.DbQuerySilentSender(DbWorker.sqliteConn, query);

        //IsChanged?

```

```

        if (rowsChanged is 1)
        {
            using SqliteDataReader reader =
DbWorker.SettingsReader(DbWorker.readSettings, DbWorker.sqliteConn);

            ChoicesLogRootInput.Text = $"Parsing URL has been setted.";
        }
        else
        {
            ChoicesLogRootInput.Text = "Unforseen error";
        }
    }

private void SetPwdBtn_Click(object sender, RoutedEventArgs e)
{
    if (SetPwdBox.Password.Length >3 &&
SetRepeatedPwdBox.Password.Length >3 && SetPwdBox.Password ==
SetRepeatedPwdBox.Password)
    {
        int rowsChanged;
        StupidWall stupidWall = new();

        if (stupidWall.ShowDialog() == true) // if checking is success
        {
            if (stupidWall.EnterPwdBox.Password == Settings.pwd)
            {
                MessageBox.Show("Авторизация пройдена");
            }
        }
    }
}

```

```

        string updatePwdQuery = $"UPDATE Settings SET pwd =
        '{SetPwdBox.Password}';";

        rowsChanged =
        DbWorker.DbQuerySilentSender(DbWorker.sqliteConn, updatePwdQuery); //
        обновление бд

        //IsChanged?
        if (rowsChanged is 1)
        {
            using SqliteDataReader reader =
            DbWorker.SettingsReader(DbWorker.readSettings, DbWorker.sqliteConn);
            //обновление локальных настроек из бд

            PwdChangeNotifier.Content = $"Your password has been
            changed at {DateTime.Now}.";
        }
        else { PwdChangeNotifier.Content = "Unforseen error. Use
        old password."; }
    }

    if (stupidWall.EnterPwdBox.Password != Settings.pwd)
    {
        MessageBox.Show("Неверный пароль"); //if checking is
        failed
    }
}

else

```



```

        {
            MessageBox.Show("Cancelled"); // if cancelled
        }
    }

    else
    {
        MessageBox.Show("Passwords are not the same, try again",
"Error");

        SetRepeatedPwdBox.Clear();
    }
}

public void UseThisPwdCheckbox_Checked(object sender,
RoutedEventArgs e)
{
    string setCheckboxQuery = $"UPDATE Settings SET pwdIsUsing =
'True'";

    DbWorker.DbQuerySilentSender(DbWorker.sqliteConn,
setCheckboxQuery); // обновление бд

    using SqlDataReader reader =
DbWorker.SettingsReader(DbWorker.readSettings, DbWorker.sqliteConn);
//обновление локальных настроек из бд

}

private void UseThisPwdCheckbox_Unchecked(object sender,
RoutedEventArgs e)
{

```

```

StupidWall stupidWall = new();

if (stupidWall.ShowDialog() == true)
{

    MessageBox.Show("Запрос пароля отключен.");
    UseThisPwdCheckbox.IsChecked = false; //обновление
состояния чекбокса в окне

    string updatePwdIsUsingQuery = $"UPDATE Settings SET
pwdIsUsing = 'False'";
    DbWorker.DbQuerySilentSender(DbWorker.sqliteConn,
updatePwdIsUsingQuery); //обновление состояния чекбокса в бд
    }
else
{
    MessageBox.Show("Операция отменена.");
    UseThisPwdCheckbox.IsChecked = true;
}
}
}
}

```

**Содержимое файла Stats.cs:**

```
using System;
namespace botserver_standard
{
    public static class Stats
    {
        static DateTime startupTime;
        static DateTime shutdownTime;
        public static void StartupTimeFixator()
        {
            startupTime = DateTime.Now;
        }

        public static void ShutdownTimeFixator()
        {
            shutdownTime = DateTime.Now;
        }
        public static void UpTimeWriter()
        {
            TimeSpan upTime = shutdownTime - startupTime;
            string query = $"INSERT INTO Session_duration(startup_time,
shutdown_time, total_uptime)" +
                $"VALUES('{startupTime}', '{shutdownTime}',
'{upTime}')";
            DbWorker.DbQuerySilentSender(DbWorker.sqliteConn, query);
        }
    }
}
```

**Содержимое файла StupidWall.xaml.cs:**

```
using System.Windows;

namespace botserver_standard
{
    public partial class StupidWall : Window
    {
        public StupidWall()
        {
            InitializeComponent();

            private void NextBtn_Click(object sender, RoutedEventArgs e)
            {

                if (EnterPwdBox.Password == Settings.pwd) //если введенный
пароль корректен
                {
                    this.DialogResult = true;
                }
            }

            public string Password
            {
                get { return EnterPwdBox.Password; }
            }
        }
    }
```

**Содержимое файла TelegramBot.cs:**

```
using System.Threading;
using Telegram.Bot;

namespace botserver_standard
{
    internal class TgBot
    {
        public static TelegramBotClient botClient = new(Settings.botToken);
//инициализация клиента
        public static CancellationTokenSource MainBotCts = new();
    }
}
```

**Содержимое файла TelegramBotKeypads.cs:**

```

using Telegram.Bot.Types.ReplyMarkups;

namespace botserver_standard
{
    internal class TelegramBotKeypads
    {
        public static readonly InlineKeyboardMarkup mainMenuKeypad =
new(
    new[]
    {
        // second row
        new[]
        {
            InlineKeyboardButton.WithCallbackData(text: "Выбрать
программу обучения", callbackData: "programChoose"),
            //InlineKeyboardButton.WithCallbackData(text: "Сменить язык",
callbackData: "langSwitch"),
        },
        // first row
        new[]
        {
            InlineKeyboardButton.WithUrl(text: "Посетить веб-сайт
проекта", url: "https://studyintomsk.ru/"),
            InlineKeyboardButton.WithUrl(text: "Проверить знание
русского языка", url: "https://studyintomsk.2i.tusur.ru/"),
        },
    });

```

```

public static readonly InlineKeyboardMarkup levelChoosingKeypad =
new(
    // keyboard
    new[]
    {
        // first row
        new[]
        {
            InlineKeyboardButton.WithCallbackData(text: "Бакалавриат",
callbackData: "бакалавриат_level"),
            InlineKeyboardButton.WithCallbackData(text: "Магистратура",
callbackData: "магистратура_level"),
            InlineKeyboardButton.WithCallbackData(text: "Специалитет",
callbackData: "специалитет_level"),
        },
        // second row
        new[]
        {
            InlineKeyboardButton.WithCallbackData(text: "🏠",
callbackData: "toHome"),
        },
    });

//old universities keypad
public static readonly InlineKeyboardMarkup
universityChoosingKeypad = new( //MUST BE PARSED!!!

```

```

// keyboard
new[]
{
    new[]
    {
        InlineKeyboardButton.WithCallbackData(text: "ТГАСУ",
callbackData: "ТГАСУ_university"),
        InlineKeyboardButton.WithCallbackData(text: "ТГПУ",
callbackData: "ТГПУ_university"),
        InlineKeyboardButton.WithCallbackData(text: "ТГУ",
callbackData: "ТГУ_university"),
    },
    new[]
    {
        InlineKeyboardButton.WithCallbackData(text: "ТПУ",
callbackData: "ТПУ_university"),
        InlineKeyboardButton.WithCallbackData(text: "ТУСУР",
callbackData: "ТУСУР_university"),
    },
    // third row
    new[]
    {
        InlineKeyboardButton.WithCallbackData(text: "🏠",
callbackData: "toHome"),
    },
});
}
}

```



**Содержимое файла UniversityEntryFreq.cs:**

```
using System.Collections.Generic;

namespace botserver_standard
{
    internal class UniversityEntryFreq
    {
        public static List<UniversityEntryFreq> universitiesFreqList = new();

        public string UniversityName { get; set; }
        public int Count { get; set; }

        public UniversityEntryFreq(string universityName, int count)
        {
            this.UniversityName = universityName;
            this.Count = count;
        }
    }
}
```

**Содержимое файла UniversityProgramButton.cs:**

```
namespace botserver_standard
{
    public class UniversityProgramButton
    {

        public int Id { get; set; } //buttonID = cardId (callbackData)
        public string Text { get; set; } //programName, buttonText

        public UniversityProgramButton(string programName, int id)
        {
            this.Id = id;
            this.Text = programName;
        }
    }
}
```

## Инструкция пользователя

После первого запуска приложения откроется окно установки нового пароля (Рисунок 22).

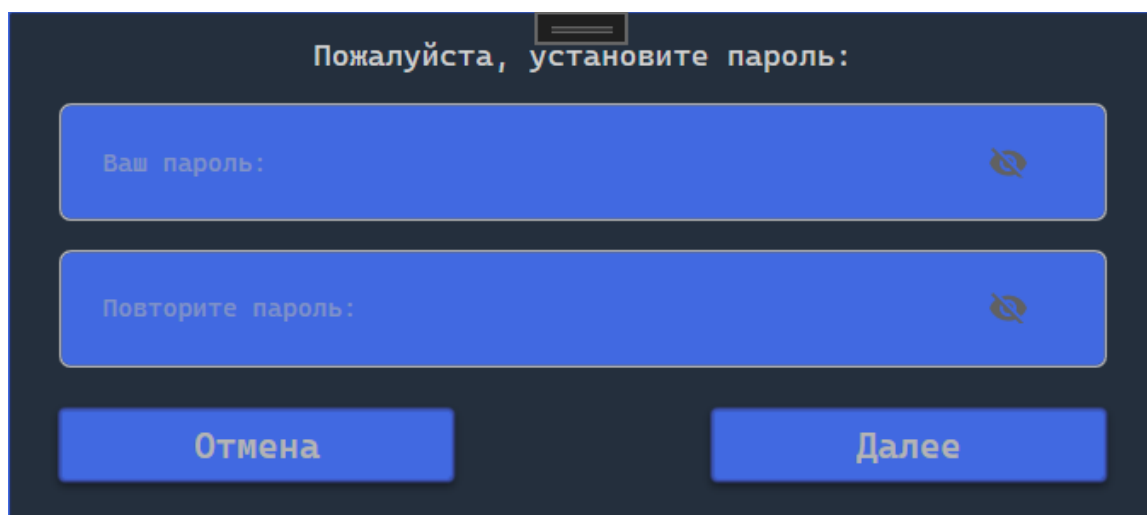


Рисунок 22 – Окно установки нового пароля

Далее необходимо ввести в соответствующие поля пароль и повторить его, затем нажать кнопку «Next» (Рисунок 23), после чего появится уведомление (Рисунок 24), откроется главное окно и во вкладке «Settings» флажок «Use this password» будет установлен (Рисунок 25).

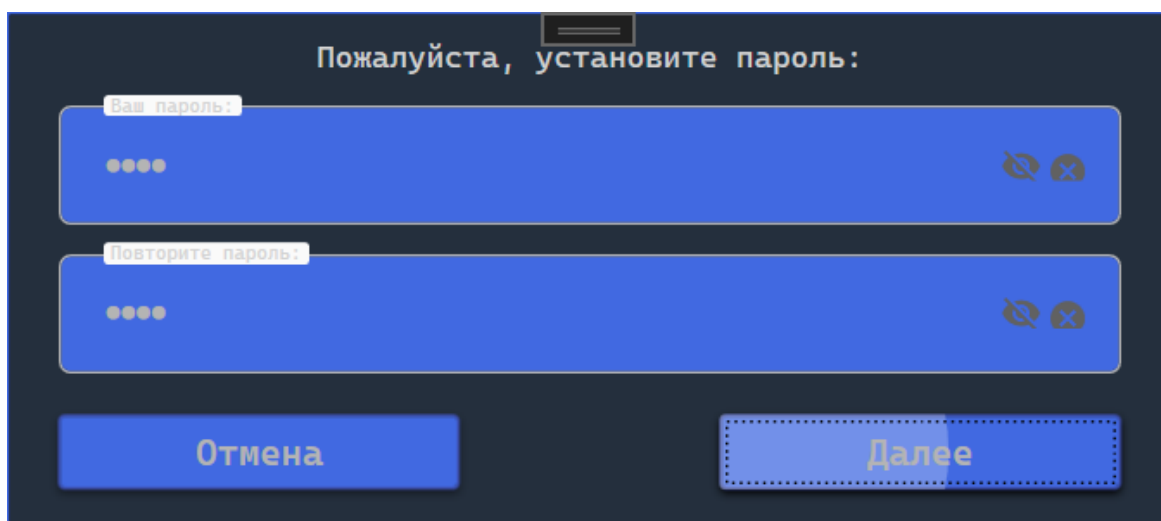


Рисунок 23 – Ввод пароля в окно установки нового пароля

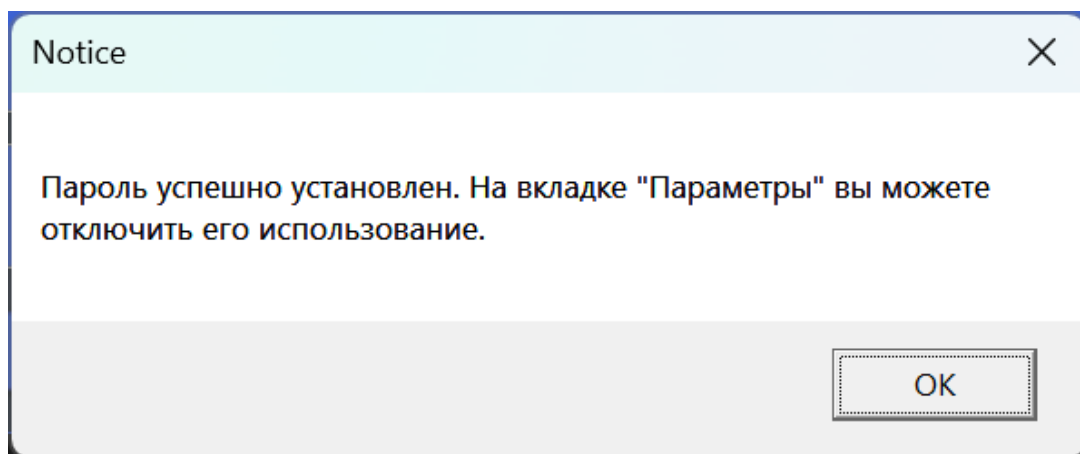


Рисунок 24 – Окно уведомления

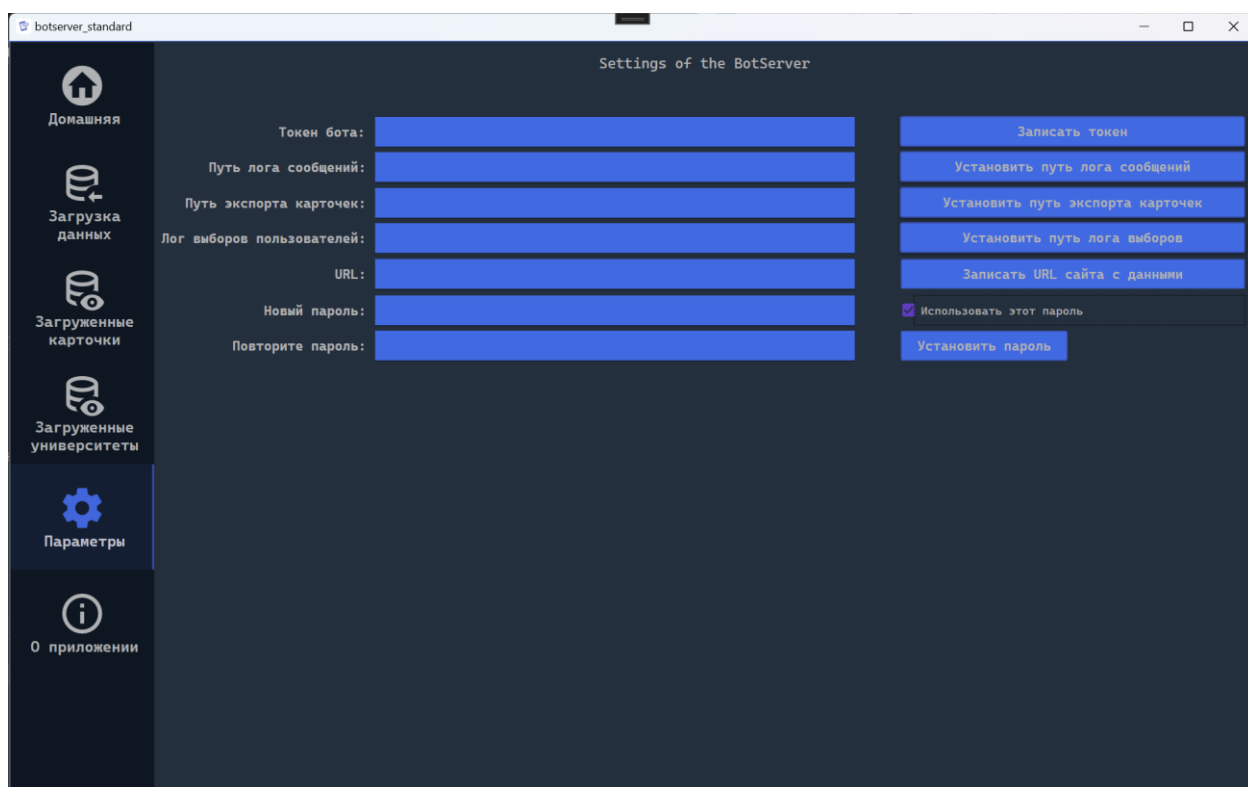


Рисунок 25 – Установленный флажок «Использовать этот пароль» во вкладке «Параметры»

Для запуска бота необходимо дождаться завершения работы парсера.  
Отслеживать журнал работы парсера можно во вкладке «Parser» (Рисунок 26)

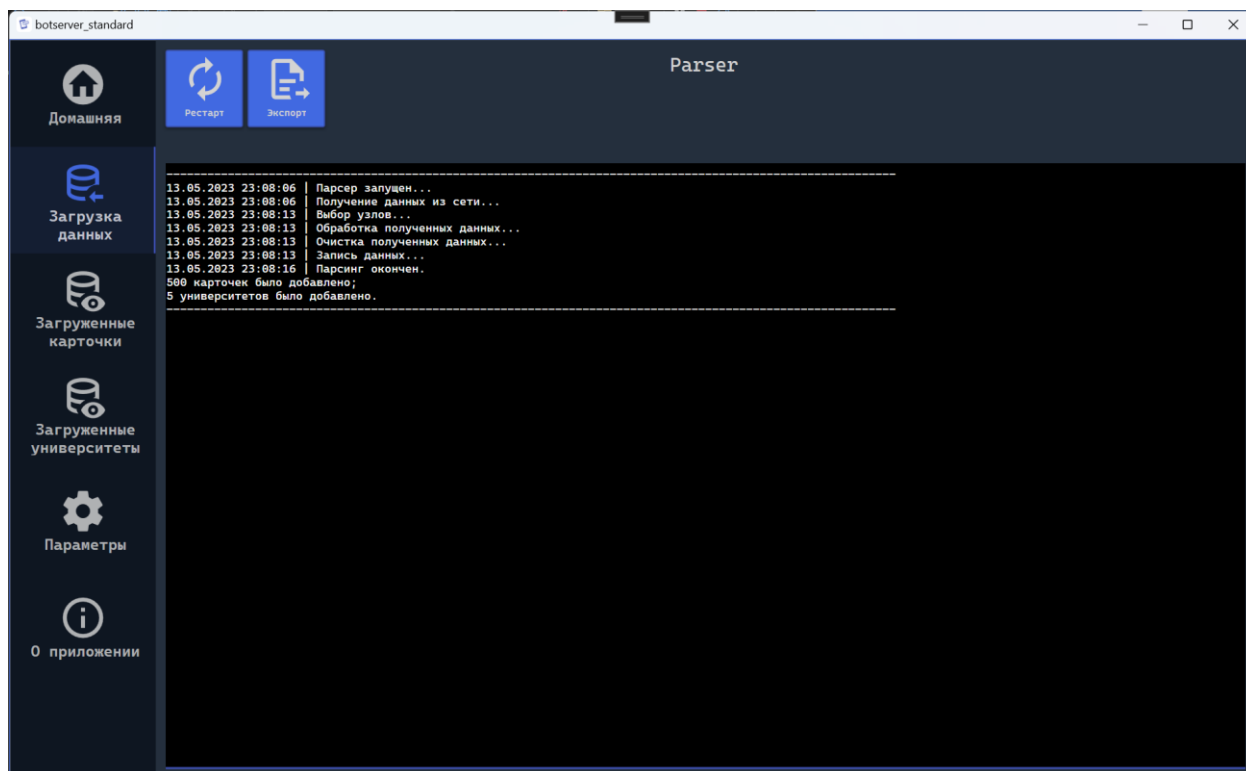


Рисунок 26 – Журнал парсера, сообщающий об окончании его работы

Также в этой вкладке можно перезапустить парсер для сбора свежих данных о карточках направлений, не перезапуская бота и экспортировать журнал парсера. Для обновления данных на вкладке присутствует кнопка «Рестарт». Это обновит полученные ранее данные карточек (Рисунок 27)

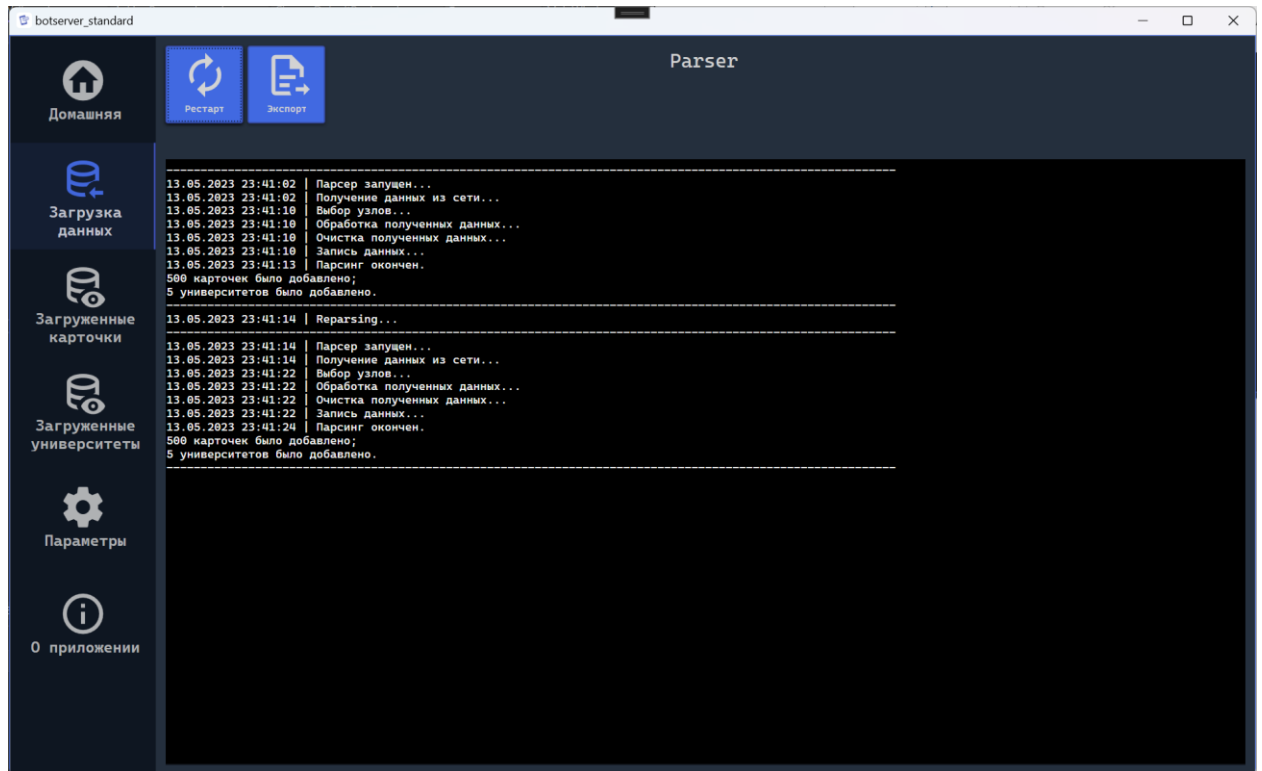


Рисунок 27 – Результат перезапуска парсера

После окончания работы парсера можно переходить к запуску бота. Для этого во вкладке «Домашняя» присутствует кнопка «Старт» (Рисунок 28).

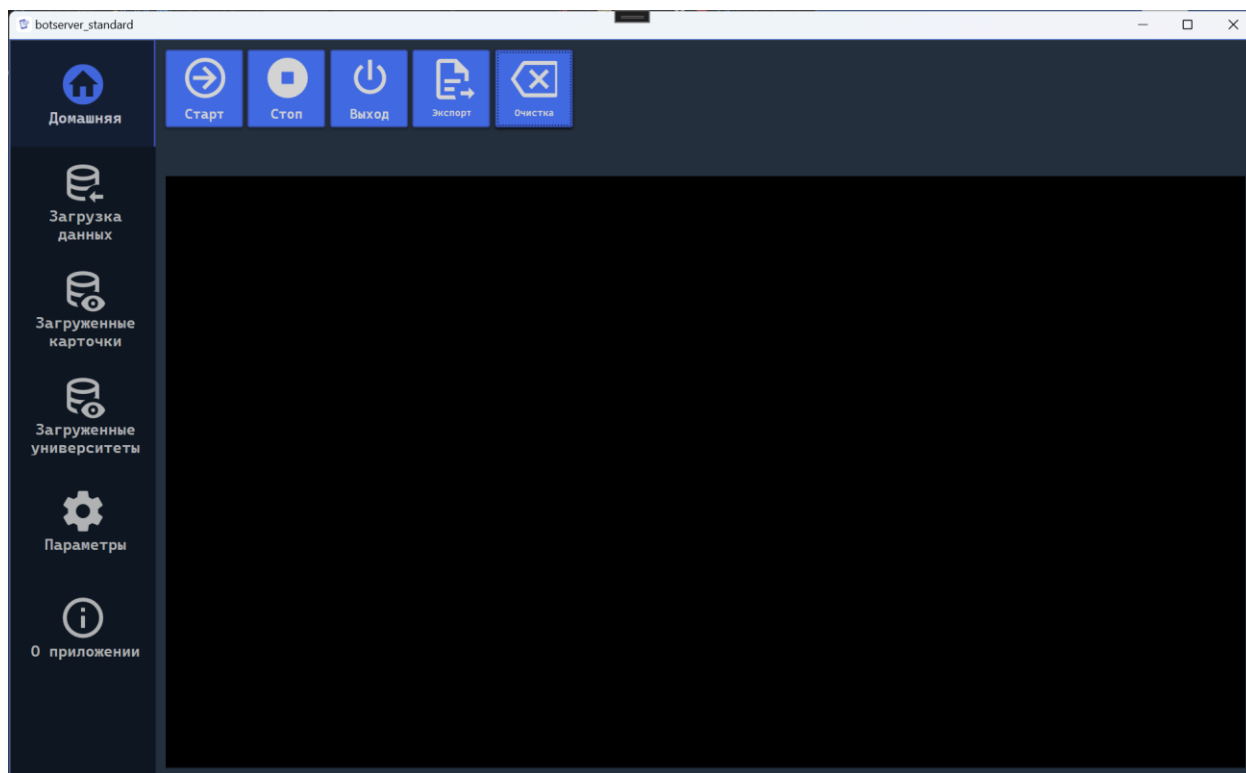


Рисунок 28 – Вкладка «Домашняя»

Также на этой вкладке есть кнопки «Стоп», «Выход», «Экспорт» и «Очистка»

- «Стоп» - позволяет остановить прием ботом сообщений без закрытия GUI;
- «Выход» - останавливает бота и закрывает GUI;
- «Экспорт» - позволяет экспортировать текущий живой журнал;
- «Очистка» - позволяет очистить вывод;

Для работы всего функционала после первого запуска бота необходимо настроить. Настройка производится в рассмотренной ранее вкладке «Параметры» (Рисунок 25). В соответствующие поля вводятся необходимые данные и нажимаются кнопки записи для записи или обновления данных. По результату записи данных выведутся уведомления (Рисунок 29, 30)

The screenshot shows the 'Settings of the BotServer' window. On the left is a sidebar with icons for 'Домашняя', 'Загрузка данных', 'Загруженные карточки', 'Загруженные университеты', 'Параметры' (selected), and '0 приложений'. The main area contains the following fields and buttons:

Field	Value	Action Button
Токен бота:	9680000000:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	Записать токен
Путь лога сообщений:	messages.txt	Установить путь лога сообщений
Путь экспорта карточек:	cards.txt	Установить путь экспорта карточек
Лог выборов пользователей:	choices.txt	Установить путь лога выборов
URL:	https://studyintomsk.ru/programs-main/?level=card-item&direction=card-1	Записать URL сайта с данными
Новый пароль:		<input type="checkbox"/> Использовать этот пароль Установить пароль
Повторите пароль:		

Рисунок 29 – Заполнение полей данными, необходимыми для работы бота

The screenshot shows the 'Settings of the BotServer' window after the data has been saved. The fields now display confirmation messages:

Field	Value	Action Button
Токен бота:	Токен изменен на 96800000000000	Записать токен
Путь лога сообщений:	Путь установлен.	Установить путь лога сообщений
Путь экспорта карточек:	Путь установлен.	Установить путь экспорта карточек
Лог выборов пользователей:	Путь установлен.	Установить путь лога выборов
URL:	URL записан.	Записать URL сайта с данными
Новый пароль:		<input type="checkbox"/> Использовать этот пароль Установить пароль
Повторите пароль:		

Рисунок 30 – Результат записи данных



На вкладке «Загруженные карточки» отображается список всех карточек направлений, которые смог собрать парсер. Эти данные можно экспортировать посредством нажатия кнопки «Export» (Рисунок 31).

Для поиска по карточкам присутствует два поля – «Поиск по названию программы» и «Поиск по университету».

Умный поиск был реализован с целью помочь администратору при необходимости найти определенную карточку и получить о ней исчерпывающую информацию. (Рисунок 32, Рисунок 33).

Название университета	Название программы	Код программы	Уровень обучения	Форма обучения	Срок обучения	Язык обучения	Куратор
TUSUR	Оптические системы и сети связи	11.03.02	бакалавриат	очная	4 года	русский	Полански
TUSUR	Системы беспроводной связи и "Интернета вещей"	11.03.02	бакалавриат	очная	4 года	русский	Полански
TUSUR	Видеоинформационные технологии	11.03.02	бакалавриат	очная	4 года	русский	Полански
TUSUR	Проектирование и технологии радиоэлектронных средств	11.03.03	бакалавриат	очная	4 года	русский	Денисова
TUSUR	Проектирование и технологии электронно-вычислительных средств	11.03.03	бакалавриат	очная	4 года	русский	Денисова
TUSUR	Технология электронных средств	11.03.03	бакалавриат	очная	4 года	русский	Денисова
TUSUR	Квантовая и оптическая электроника	11.03.04	бакалавриат	очная	4 года	русский	Каранский
TUSUR	Промышленная электроника	11.03.04	бакалавриат	очная	4 года	русский	Каранский
TUSUR	Микроэлектроника и твердотельная электроника	11.03.04	бакалавриат	очная	4 года	русский	Каранский
TUSUR	Фотоника нелинейных, волноводных и периодических структур	12.03.03	бакалавриат	очная	4 года	русский	Каранский
TUSUR	Электромагнитная совместимость	11.03.01	бакалавриат	очная	4 года	русский	Полански
TUSUR	Системы мобильной связи	11.03.02	бакалавриат	очная	4 года	русский	Полански
TUSUR	Защищенные системы и сети связи	11.03.02	бакалавриат	очная	4 года	русский	Полански
TUSUR	Автоматизированное управление бизнес-процессами и финансами	09.03.01	бакалавриат	очная	4 года	русский	Хабидуллин
TUSUR	Аналитические информационные системы	09.03.02	бакалавриат	очная	4 года	русский	Хабидуллин
TUSUR	Прикладная информатика в экономике	09.03.03	бакалавриат	очная	4 года	русский	Вадим Я.
TUSUR	Индустриальная разработка программных продуктов	09.03.04	бакалавриат	очная	4 года	русский	Вадим Я.
TUSUR	Безопасности автоматизированных систем	10.03.01	бакалавриат	очная	4 года	русский	Ильин Н.
TUSUR	Радиотехнические средства передачи, приема и обработки сигналов	11.03.01	бакалавриат	очная	4 года	русский	Полански
TUSUR	Микроволновая техника и антенны	11.03.01	бакалавриат	очная	4 года	русский	Полански
TUSUR	Управление разработками робототехнических комплексов	15.04.06	магистратура	очная	2 года	русский	Ситник А.
TUSUR	Управление качеством промышленной продукции и услуг	27.04.02	магистратура	очная	2 года	русский	Ситник А.
TUSUR	Управление и автоматизация технологических процессов и производств	27.04.04	магистратура	очная	2 года	русский	Хабидуллин
TUSUR	Компьютерное моделирование и обработка информации в технических системах	27.04.04	магистратура	очная	2 года	русский	Хабидуллин
TUSUR	Управление инновациями в электронной технике	27.04.05	магистратура	очная	2 года	русский	Ситник А.
TUSUR	Экономика и управление финансами предприятия	38.04.01	магистратура	очная	2 года	русский	Аксенова
ТГУПИ	Математическая физика	18.04.04	магистратура	очная	2 года	русский	Аксенова

Рисунок 31 – общий вид окна «Загруженные карточки»

botserver\_standard

Поиск по названию программы:  Поиск по университету:

Домашняя

Загрузка данных

Загруженные карточки

Загруженные университеты

Параметры

0 приложений

Исход

Название университета	Название программы	Код программы	Уровень обучения	Форма обучения	Срок обучения	Язык обу
ТУСУР	Микроэлектроника и твердотельная электроника	11.03.04	бакалавриат	очная	4 года	русский
ТУСУР	Микроволновая техника и антенны	11.03.01	бакалавриат	очная	4 года	русский
ТУСУР	Промышленная электроника и микропроцессорная техника	11.04.04	магистратура	очная	2 года	русский
ТУСУР	Микроволновая техника и антенны	11.04.01	магистратура	очная	2 года	русский
ТУСУР	Автоматизация проектирования микро- и нанозлектронных устройств для радиотехнических систем	09.04.01	магистратура	очная	2 года	русский
ТУСУР	Нанотехнологии в электронике и микросистемной технике	28.03.01	Бакалавриат	очная	4 года	русский

Рисунок 32 – Результат работы умного поиска по названию программы

botserver\_standard

Поиск по названию программы:  Поиск по университету:

Домашняя

Загрузка данных

Загруженные карточки

Загруженные университеты

Параметры

0 приложений

Исход

Название университета	Название программы	Код программы	Уровень обучения	Форма обучения	Срок об
ТГАСУ	Инженерная защита окружающей среды	20.03.01	бакалавриат	очная	4 года
ТГАСУ	Эксплуатация и обслуживание технологических объектов нефтегазового производства	21.03.01	бакалавриат	очная	4 года
ТГАСУ	Городской кадастр	21.03.02	бакалавриат	очная	4 года
ТГАСУ	Подъемно-транспортные, строительные, и дорожные машины	23.03.02	бакалавриат	очная	4 года
ТГАСУ	Архитектура	07.03.01	бакалавриат	очная	5 лет
ТГАСУ	Реконструкция и реставрация архитектурного наследия	07.03.02	бакалавриат	очная	5 лет
ТГАСУ	Дизайн архитектурной среды	07.03.03	бакалавриат	очная	5 лет
ТГАСУ	Промышленное и гражданское строительство	08.03.01	бакалавриат	очная	4 года
ТГАСУ	Городское строительство	08.03.01	бакалавриат	очная	4 года
ТГАСУ	Производство строительных материалов, изделий и конструкций	08.03.01	бакалавриат	очная	4 года
ТГАСУ	Инженерные системы жизнеобеспечения в строительстве	08.03.01	бакалавриат	очная	4 года
ТГАСУ	Автомобильные дороги	08.03.01	бакалавриат	очная	4 года
ТГАСУ	Автомобильные дороги	08.03.01	бакалавриат	очная	4 года
ТГУ	Перевод и обучение межкультурной коммуникации	45.03.02	бакалавриат	очная	4 года
ТГУ	Перевод и обучение межкультурной коммуникации	45.05.01	специалитет	очная	5 лет
ТГУ	Вычислительная механика и компьютерный инжиниринг	15.03.03	бакалавриат	очная	4 года
ТГУ	Промышленная и специальная робототехника	15.03.03	бакалавриат	очная	4 года
ТГУ	Термодинамика	16.03.01	бакалавриат	очная	4 года
ТГУ	Баллистика и гидроаэродинамика	24.03.03	бакалавриат	очная	4 года
ТГУ	Томская международная научная программа	27.03.05	бакалавриат	очная	4 года
ТГУ	Графика	53.05.03	специалитет	очная	5 лет
ТГУ	Издательское дело	42.03.03	бакалавриат	очная	4 года
ТГУ	Отечественная филология	45.03.01	бакалавриат	очная	4 года
ТГУ	Профессионально-деловая коммуникация на иностранных языках (английский и немецкий)	45.03.01	бакалавриат	очная	4 года
ТГУ	Русский язык	45.03.01	бакалавриат	очная	4 года
ТГУ	Фундаментальная и прикладная лингвистика	45.03.03	бакалавриат	очная	4 года
ТГУ	Физическая химия	07.04.01	бакалавриат	очная	4 года

Рисунок 33 – Результат работы умного поиска по названию университета

На вкладке «Загруженные университеты» отображается список всех университетов с количеством направлений в каждом. (Рисунок 34). Для поиска по карточкам присутствует два поля – «Поиск по университету» и «Поиск по количеству программ» (Рисунок 35, Рисунок 36). Умный поиск работает по тому же принципу, что и на вкладке, рассмотренной выше за исключением тех данных, по которым он ищет.



Название университета	Количество программ
ТУСУР	78
ТПУ	203
ТГПУ	52
ТГАСУ	45
ТГУ	122

Рисунок 34 – Общий вид вкладки «Загруженные университеты»

The screenshot shows the 'botserver\_standard' application window. The left sidebar contains navigation icons: 'Домашняя' (Home), 'Загрузка данных' (Load data), 'Загруженные карточки' (Loaded cards), 'Загруженные университеты' (Loaded universities), 'Параметры' (Parameters), and '0 приложений' (0 applications). The main area has two search filters: 'Поиск по университету:' (Search by university) with the value 'тг' and 'Поиск по количеству программ:' (Search by number of programs) which is empty. An 'Экспорт' (Export) button is in the top right. Below the filters is a table with two columns: 'Название университета' (University name) and 'Количество программ' (Number of programs). The table contains three rows of data.

Название университета	Количество программ
ТГПУ	52
ТГАСУ	45
ТГУ	122

Рисунок 35 - Результат работы умного поиска по университету

The screenshot shows the 'botserver\_standard' application window with the same sidebar as Figure 35. The search filters are: 'Поиск по университету:' (empty) and 'Поиск по количеству программ:' with the value '2'. The 'Экспорт' (Export) button is present. The table below shows search results filtered by the number of programs.

Название университета	Количество программ
ТПУ	263
ТГПУ	52
ТГУ	122

Рисунок 36 - Результат работы умного поиска по количеству программ

На вкладке «Параметры» осталась последняя нерассмотренная возможность – смена пароля. Для ее использования необходимо ввести в соответствующие поля новый пароль и повторить его, затем нажать кнопку «Set password» (Рисунок 37).

Появится окно, куда необходимо ввести старый пароль для подтверждения его смены (Рисунок 38). При вводе корректного пароля и подтверждении окно закроется, и на вкладке «Параметры» будет отображено уведомление о смене пароля (Рисунок 39)

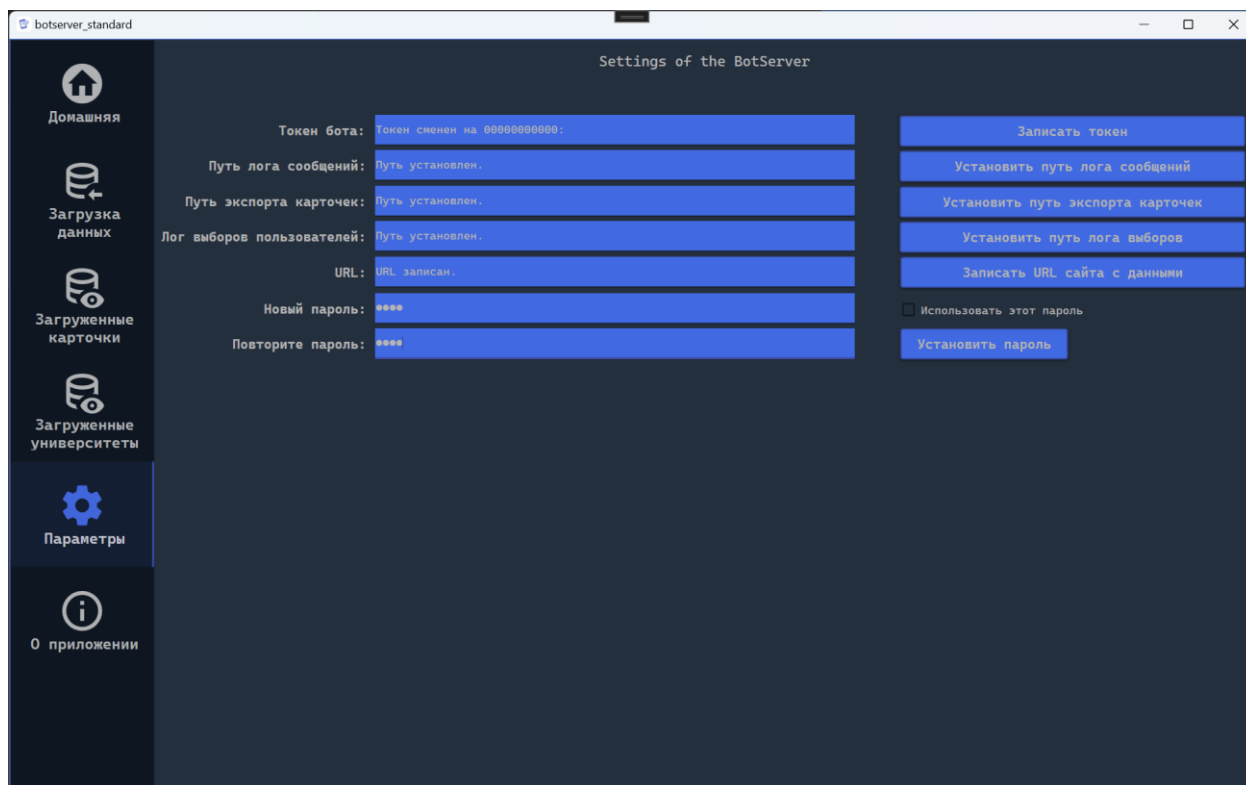
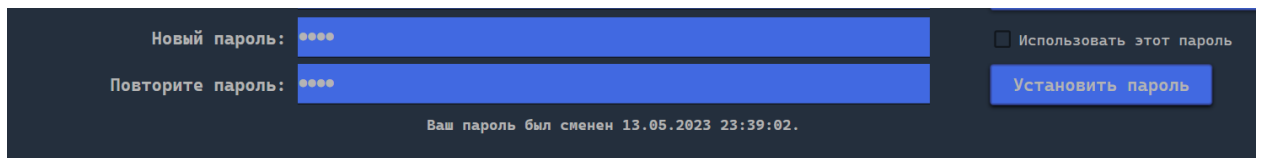


Рисунок 37 – Ввод нового пароля



The screenshot shows a dark-themed mobile application interface. At the top, there is a hamburger menu icon. Below it, a label "Ваш пароль:" is positioned above a large blue rectangular input field. Inside the input field, four dots represent masked characters, and a cursor is visible at the end. To the right of the input field are two small icons: a crossed-out eye and a close (X) button. Below the input field, there are two blue buttons: "Отмена" (Cancel) on the left and "Далее" (Next) on the right.

Рисунок 38 – Окно ввода пароля для подтверждения его смены



The screenshot displays a dark-themed mobile application interface for password confirmation. It features two blue input fields. The first is labeled "Новый пароль:" and the second is labeled "Повторите пароль:". Both fields contain four dots representing masked characters. To the right of the "Новый пароль:" field, there is a checkbox labeled "Использовать этот пароль". Below the input fields, a blue button labeled "Установить пароль" is visible. At the bottom center, a status message reads: "Ваш пароль был сменен 13.05.2023 23:39:02."

Рисунок 39 – Уведомление об успешной смене пароля

# Диаграммы состояний

## Диаграмма состояния «Установка пароля»

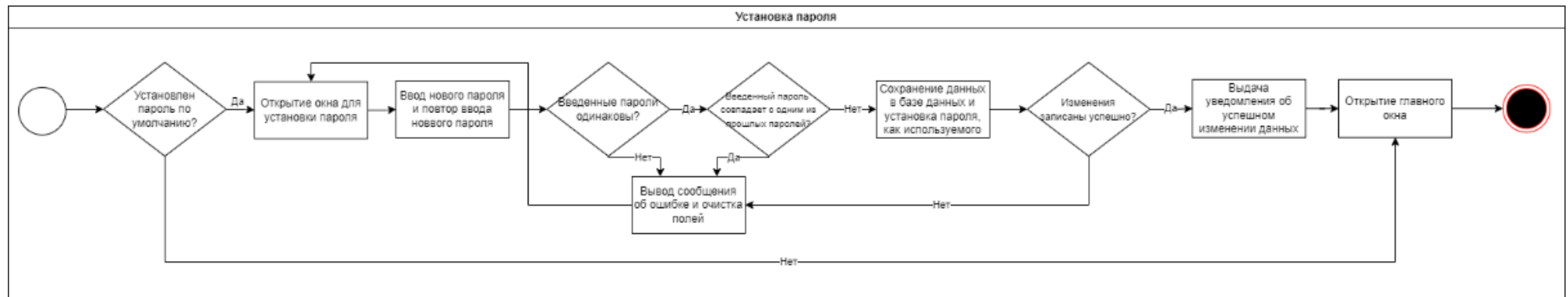


Рисунок 40 – Диаграмма состояния «Установка пароля»

## Диаграмма состояния «Вход по паролю»

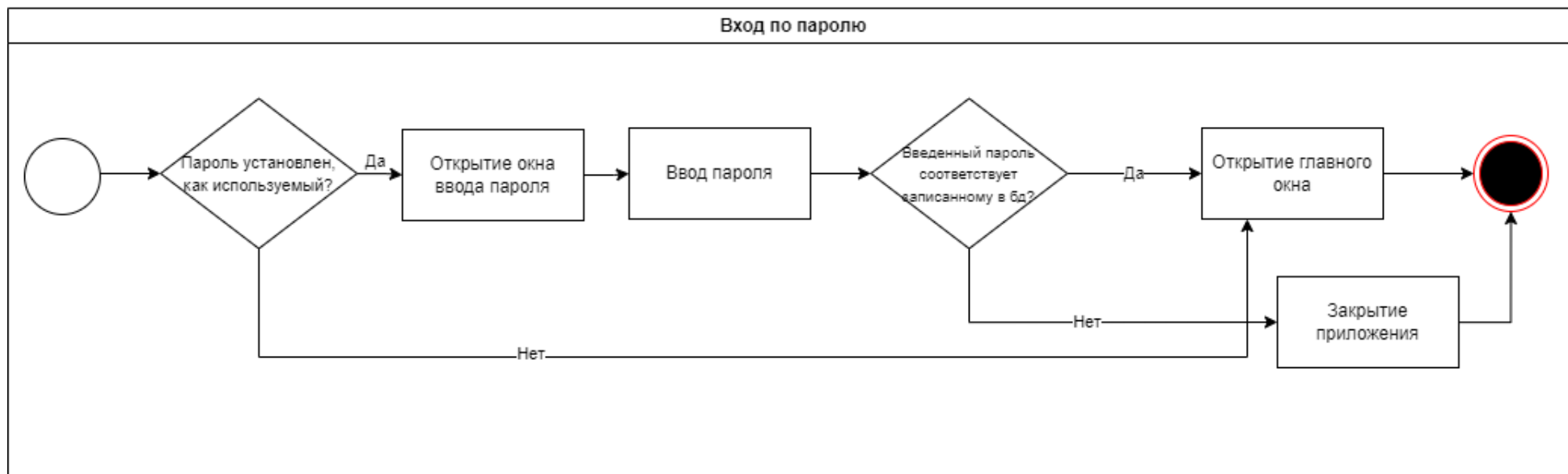


Рисунок 41 – Диаграмма состояния «Вход по паролю»



Диаграмма состояния «Отключение входа по паролю»



Рисунок 42 – Диаграмма состояния «Отключение входа по паролю»

Диаграмма состояния «Включение входа по паролю»

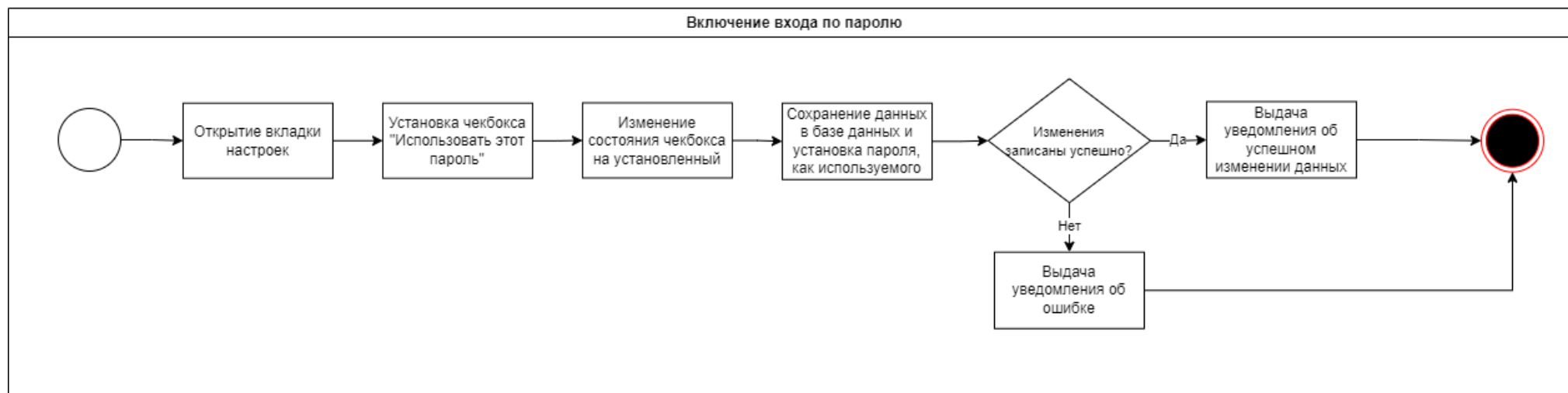


Рисунок 43 – Диаграмма состояния «Включение входа по паролю»

## Диаграмма состояния «Установка параметров»



Рисунок 44 – Диаграмма состояния «Установка параметров»

Диаграмма состояния «Выбор пользователем направления обучения»

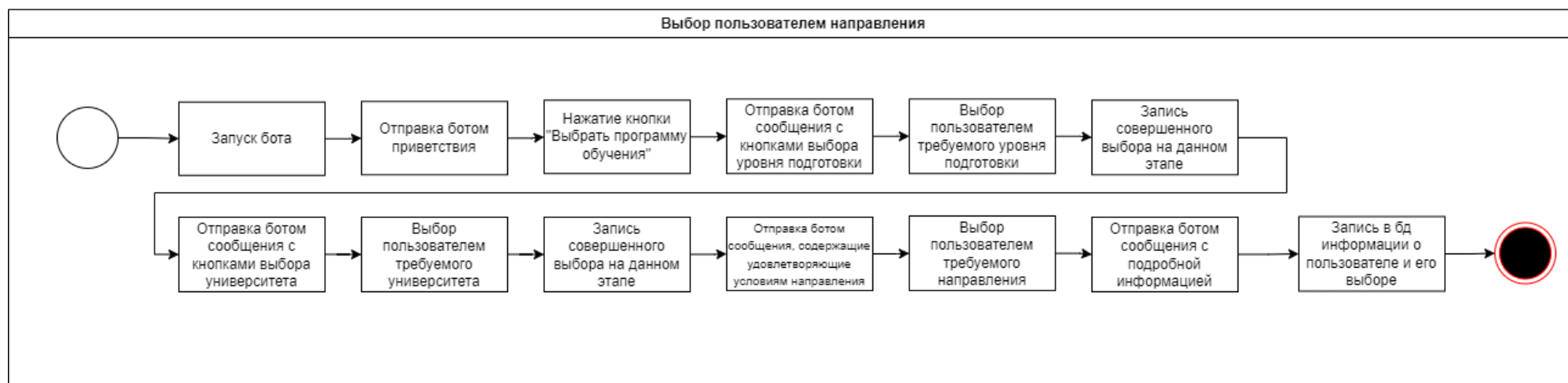


Рисунок 45 – Диаграмма состояния «Выбор пользователем направления обучения»