ДЕПАРТАМЕНТ ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ТОМСКОЙ ОБЛАСТИ

ОБЛАСТНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

«ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

Специальность 09.02.07 Информационные системы и программирование

Пояснительная записка

к производственной практике

ОПП.23.09.02.07.601.5.ПЗ

|  |  |  |
| --- | --- | --- |
| Студент  «\_\_»\_\_\_\_\_\_\_\_ 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | П. Д. Левицкий |
| Руководитель  «\_\_»\_\_\_\_\_\_\_\_ 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Д. В. Смоляков |
|  |  |  |
|  |  |  |
|  |  |  |

Томск 2023

СОДЕРЖАНИЕ

|  |  |
| --- | --- |
| Введение | 3 |
| 1. ОБЩАЯ ЧАСТЬ | 5 |
| * 1. Анализ предметной области   2. Средства и среда разработки | 5  6 |
| 1. СПЕЦИАЛЬНАЯ ЧАСТЬ | 8 |
| * 1. Описание требований к   информационной системе  2.2. Диаграмма вариантов использования | 8  10 |
| 2.3 Диаграмма состояний | 11 |
| 2.4 Схема базы данных | 15 |
| 2.5 Словарь данных | 16 |
| 2.6 Пользовательские сценарии | 22 |
| 2.7 Прототипы интерфейсов | 24 |
| ЗАКЛЮЧЕНИЕ | 30 |
| ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ | 31 |
| ПРИЛОЖЕНИЕ A. Листинг программы | 32 |
| ПРИЛОЖЕНИЕ Б. Инструкция пользователя | 138 |
| ПРИЛОЖЕНИЕ В. Диаграммы состояний | 139 |

Введение

Индивидуальный предприниматель Жданов Александр Валерьевич

Предприятие создано 22.01.2010 года, основной вид деятельности розничная торговля.

В настоящее время у предприятия четыре торговых точки, два магазина продовольственных товаров, один магазин с детскими товарами, ещё один крупный магазин со смешанными товарами и ювелирный отдел.

В организационную структуру входят индивидуальный предприниматель, финансовый директор, бухгалтер, оператор, продавец-консультант, разнорабочий, уборщица. В штате предприятия числится семнадцать сотрудников.

Индивидуальный предприниматель (руководитель) решает вопросы организационной и хозяйственной деятельности предприятия, частично занимается закупкой и доставкой товаров.

Работа финансового директора заключается в том, что связано с деньгами, занимается планированием, контролирует затраты и налаживает внутренний контроль на предприятии и выплатой заработной платы работникам. Так же занимается вопросами персонала и ведением кадрового учёта на предприятии.

Бухгалтер занимается ведением и сдачей бухгалтерской и налоговой отчётности, расчётом заработной платы и налогов, ведением кассовых операций и работает с первичной документацией. Ведёт расчёты с поставщиками. Проводит инвентаризации.

Оператор формирует расходную и приходную документацию, оформляете накладные, проводит корректировку сведений в документации по фактам движения товара, оформляет возвратную документацию, проводит сверку остатков на складе и формирование отчётов по движению товара, подготавливает первичную бухгалтерскую документацию, вносит данные в базу, принимает участие в проведении инвентаризации.

Обязанности продавца-консультанта заключаются в приёме, выкладке и реализации товара, консультирование покупателей, упаковке, приёме и обмене товара, проверка исправности товара, соответствия его наименования нанесённому на упаковке, визуальный осмотр.

1. ОБЩАЯ ЧАСТЬ
   1. Анализ предметной области

Работа предприятия напрямую зависит от профессионализма сотрудников. Ответственно за учет кадров на предприятии несет Финансовый директор.

Основной функцией финансового директора при ведении учета кадров является сбор и хранение информации о сотрудниках и предоставление её руководителю предприятия в виде документов.

Для автоматизации ведения кадрового учета возникла необходимость разработать информационную систему, упрощающую сбор, хранение, актуализацию информации о сотрудниках. Данная информационная система позволит уменьшить время на поиск нужной информации, а также упростить доступ к ней, исключить ошибки при ведении кадрового учета и упростить процесс подготовки документации.

Целью производственной практики является: Разработка информационной системы учета кадров для ИП Жданов

Задачи производственной практики:

1. Проведение анализа конкурентов;

2. Выбор платформы и языка программирования;

3. Изучение документации;

4. Реализация программного кода;

5. Тестирование и исправление ошибок;

6. Составление документации и внедрение.

* 1. 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:я

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.

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 7.0.4 – легковесный ADO.NET провайдер, предоставляет возможность работы с SQLite.

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.СПЕЦИАЛЬНАЯ ЧАСТЬ

* 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 телеграм-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

17. Установка URL сайта, с которого парсер будет собирать данные;

18. Установка токена бота, к которому будет подключаться ИС;

19. Установка пути для экспортирующихся отчетов;

20. Установка пути для журналов;

21. Очистка окна живого журнала;2.2. Диаграмма вариантов использования

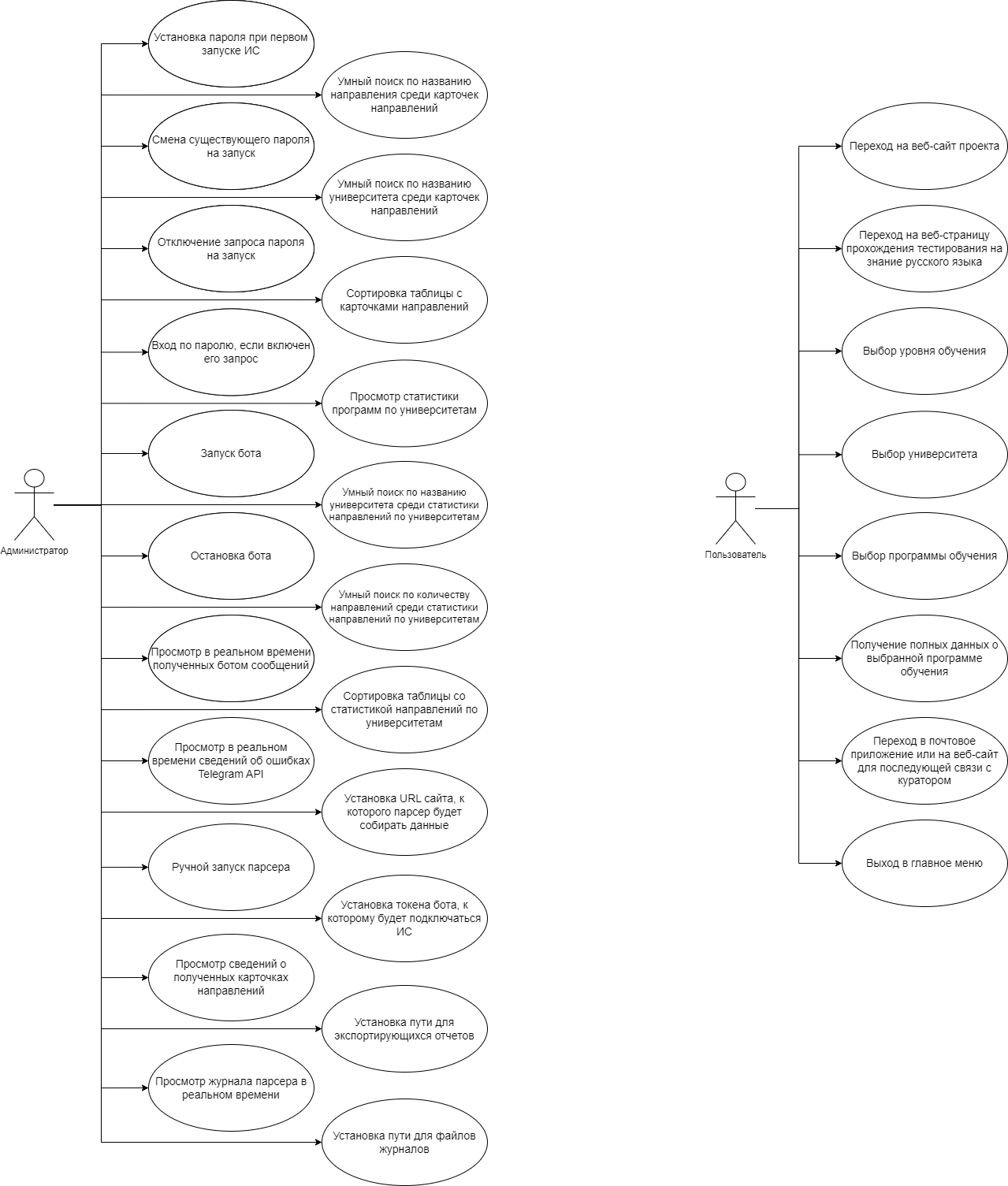


Рисунок 1 – Диаграмма вариантов использования для администратора

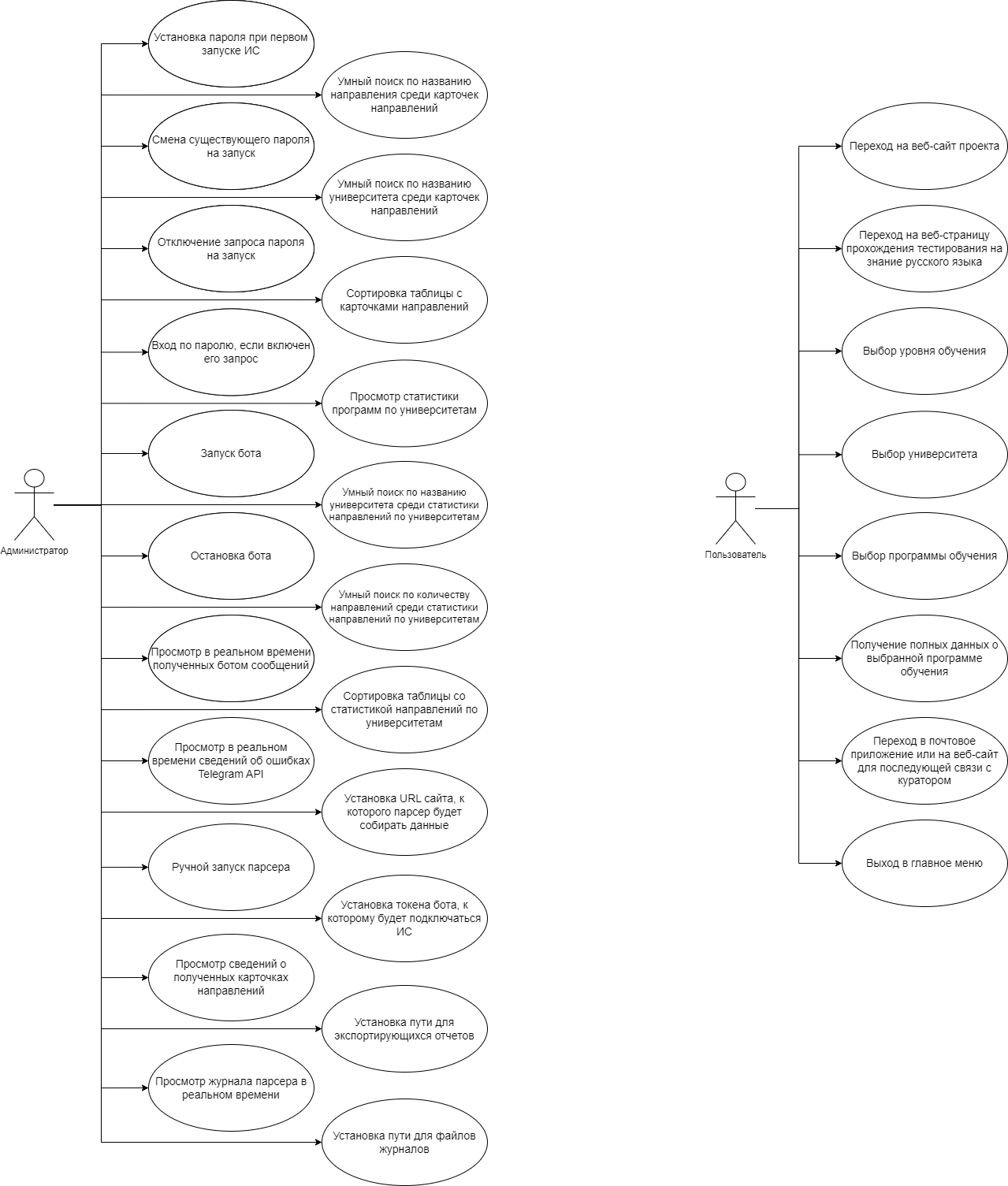


Рисунок 2 – Диаграмма вариантов использования для пользователя

2.4 Схема базы данных

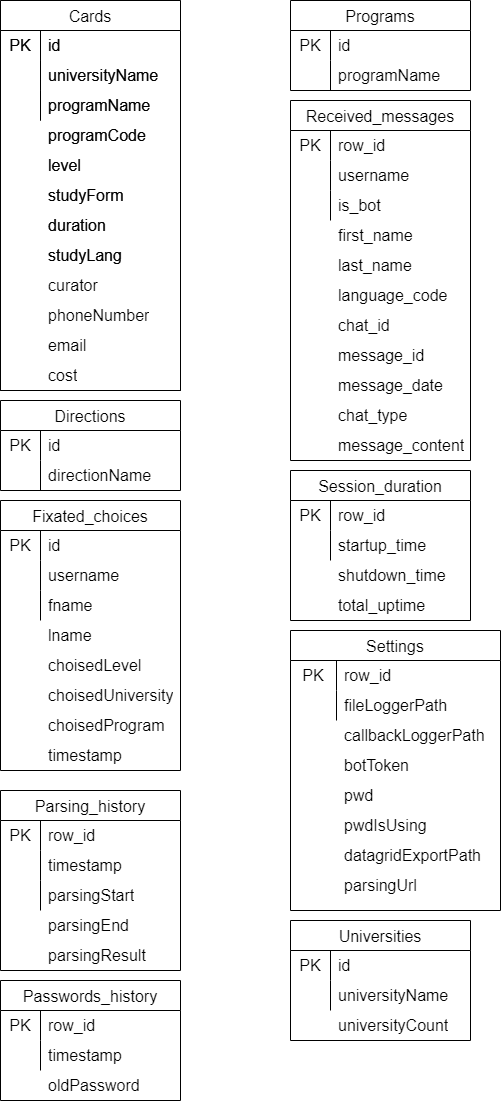


Рисунок 3 – блок схема базы данных

2.5 Словарь данных

Таблица 1 – Cards (Карточки направлений)

|  |  |  |  |
| --- | --- | --- | --- |
| № п-п | Название поля | Тип данных | Описание |
| 1 | id (PK) | INT | Идентификатор карточки (уникальный) |
| 2 | universityName | TEXT | Название университета |
| 3 | programName | TEXT | Название программы |
| 4 | programCode | TEXT | Код программы |
| 5 | level | TEXT | Уровень обучения |
| 6 | studyForm | TEXT | Форма обучения |
| 7 | duration | TEXT | Длительность |
| 8 | studyLang | TEXT | Язык обучения |
| 9 | curator | TEXT | Куратор |
| 10 | phoneNumber | TEXT | Номер телефона |
| 11 | email | TEXT | Почта |
| 12 | cost | TEXT | Стоимость обучения |

Таблица 2 – Directions (Направления)

|  |  |  |  |
| --- | --- | --- | --- |
| № п-п | Название поля | Тип данных | Описание |
| 1 | id (PK) | INT | Идентификатор направления (уникальный) |
| 2 | directionName | TEXT | Название направления  (уникальное) |

Таблица 3 – Fixated\_choices (Зафиксированные выборы пользователей)

|  |  |  |  |
| --- | --- | --- | --- |
| № п-п | Название поля | Тип данных | Описание |
| 1 | id (PK) | INT | Идентификатор набора выборов (уникальный) |
| 2 | username | TEXT | Никнейм пользователя |
| 3 | fname | TEXT | Имя пользователя |
| 4 | lname | TEXT | Фамилия пользователя |
| 5 | choisedLevel | TEXT | Выбранный уровень обучения |
| 6 | choisedUniversity | TEXT | Выбранный университет |
| 7 | choisedProgram | TEXT | Выбранная программа |
| 8 | timestamp | TEXT | Время фиксации результата |

Таблица 4 – Parsing\_history (История парсинга)

|  |  |  |  |
| --- | --- | --- | --- |
| № п-п | Название поля | Тип данных | Описание |
| 1 | row\_id (PK) | INT | Идентификатор строки (уникальный) |
| 2 | timestamp | TEXT | Время, затраченное на парсинг |
| 3 | parsingStart | TEXT | Время начала парсинга |
| 4 | parsingEnd | TEXT | Время конца парсинга |
| 5 | parsingResult | INT | Результат парсинга (количество полученных карточек) |

Таблица 5 – Passwords\_history (История установленных паролей)

|  |  |  |  |
| --- | --- | --- | --- |
| № п-п | Название поля | Тип данных | Описание |
| 1 | row\_id (PK) | INT | Идентификатор строки (уникальный) |
| 2 | timestamp | TEXT | Дата смены пароля |
| 3 | oldPassword | TEXT | Неактуальный на момент записи пароль |

Таблица 6 – Programs (Программы обучения)

|  |  |  |  |
| --- | --- | --- | --- |
| № п-п | Название поля | Тип данных | Описание |
| 1 | id (PK) | INT | Идентификатор программы обучения (уникальный) |
| 2 | programName | TEXT | Названия программы |

Таблица 7 – Received\_messages (Принятые ботом сообщения)

|  |  |  |  |
| --- | --- | --- | --- |
| № п-п | Название поля | Тип данных | Описание |
| 1 | row\_id (PK) | INT | Идентификатор строки (уникальный) |
| 2 | username | TEXT | Никнейм пользователя |
| 3 | is\_bot | TEXT | Пользователь – бот? |
| 4 | first\_name | TEXT | Имя пользователя |
| 5 | last\_name | TEXT | Фамилия пользователя |
| 6 | language\_code | TEXT | Код языка пользователя |
| 7 | chat\_id | TEXT | Идентификатор чата |

Продолжение таблицы 8

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | message\_id | TEXT | Идентификатор принятого сообщения |
| 9 | message\_date | TEXT | Дата получения ботом сообщения |
| 10 | chat\_type | TEXT | Тип чата с пользователем |
| 11 | message\_content | TEXT | Содержимое принятого сообщения |

Таблица 8 – Session\_duration (Длительность сессий)

|  |  |  |  |
| --- | --- | --- | --- |
| № п-п | Название поля | Тип данных | Описание |
| 1 | row\_id (PK) | INT | Идентификатор строки (уникальный) |
| 2 | startup\_time | TEXT | Дата и время запуска бота |
| 3 | shutdown\_time | TEXT | Дата и время остановки бота |
| 4 | total\_uptime | TEXT | Интервал работы бота |

Таблица 9 – Settings (Настройки)

|  |  |  |  |
| --- | --- | --- | --- |
| № п-п | Название поля | Тип данных | Описание |
| 1 | row\_id (PK) | INT | Идентификатор строки (уникальный) |
| 2 | fileLoggerPath | TEXT | Путь к журналу принятых сообщений |
| 3 | callbackLoggerPath | TEXT | Путь к журналу выборов пользователей |
| 4 | botToken | TEXT | Токен telegram-бота |

Продолжение таблицы 10

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | pwd | TEXT | Пароль на вход в GUI |
| 6 | pwdIsUsing | TEXT | Вход по паролю активирован? |
| 7 | datagridExportPath | TEXT | Путь экспорта данных из таблицы карточек направлений |
| 8 | parsingUrl | TEXT | URL, требуемый парсеру |

Таблица 10 – Universities (Университеты)

|  |  |  |  |
| --- | --- | --- | --- |
| № п-п | Название поля | Тип данных | Описание |
| 1 | Id (PK) | INT | Идентификатор университета (уникальный) |
| 2 | universityName | TEXT | Название университета |
| 3 | universityCount | INT | Количество направлений |

2.6 Пользовательские сценарии

После запуска программы откроется окно авторизации. Для начала работы необходимо войти в систему. Для этого необходимо ввести логин и пароль и нажать на кнопку «Войти». После прохождения авторизации пользователю откроется окно в зависимости от его роли.

Войдя в систему как сотрудник пользователь может изменить свои персональные данные во вкладке «Профиль», введя актуальные данные в требуемые поля и нажав на кнопку «Сохранить».

Также пользователю доступна вкладка «Посещаемость», где пользователь может посмотреть информацию о своей посещаемости и вкладка «Отпуск», где пользователь может увидеть статус своего заявления на отпуск, а также составить его. Для составления заявления на отпуск требуется нажать кнопку «Заявление на отпуск», для вызова формы. После её заполнения нужно нажать на кнопку «Создать», после чего создастся заявление.

Авторизовавшись как администратор пользователю доступна вкладка «Пользователи», где доступен список всех сотрудников фирмы с возможностью фильтрации и поиска. Для фильтрации требуется поставить галочку в нужный чекбокс. Для поиска нужно ввести ФИО в поле «Поиск…». Также на этой вкладке возможно создать пользователя: Нажав на кнопку «создать пользователя» откроется форма после заполнение которой требуется нажать на кнопку «Создать», возможно изменить персональные данные сотрудника, выбрав его из списка и нажав на кнопку «Изменить пользователя», после чего откроется форма изменение ,где нужно заменить данные о пользователя и нажать на кнопку «Сохранить», возможность удалять пользователя, выбрав его из списка и нажав на кнопку «Удалить пользователя».

Так же во вкладке «Пользователи» возможно сформировать договор трудоустройства нажав на кнопку «Принять на работу», заполнив форму и нажав на кнопку «Создать». Таким же образом формируются отчеты по посещаемости, по текучке кадров, по отпускам (Нажав на кнопки «отчет по посещаемости», «отчет по текучке кадров», «отчет по отпускам» соответственно). Чтобы уволить сотрудника выбрав сотрудника из списка и нажать на кнопку «Уволить сотрудника», заполнив форму нажать на кнопку «Создать», также возможно создать заявления на отпуск. Для этого требуется нажать кнопку «Заявление на отпуск», для вызова формы. После её заполнения нужно нажать на кнопку «Создать», после чего создастся заявление.

Во вкладке «Посещаемость» можно посмотреть посещаемость сотрудников по отделам, заполнив поле отдел и нажав на кнопку «Вывести».

Во вкладке «Отпуск» можно посмотреть заявки на отпуск сотрудников по отделам, заполнив поле отдел и нажав на кнопку «Вывести». Так же можно принять либо отклонить заявку выбрав её из списка и нажать на кнопку «Принять» или «Отклонить»

Во вкладке «Кадры» можно посмотреть все изменения кадров по отделам, заполнив поле отдел и нажав на кнопку «Вывести».

Во вкладке «Должности и отделы» возможно посмотреть все должности и отделы предприятия, а так создать их или удалить. Для создания должности или отдела нужно заполнить требуемое поле и нажать на кнопку «Создать», Для удаления же требуется выбрать запись из списка и нажать на кнопку «Удалить»

2.7 Прототипы интерфейсов

Окно установки пароля (Рисунок 4)



Рисунок 4 – Окно установки пароля

Окно входа (Рисунок 5)

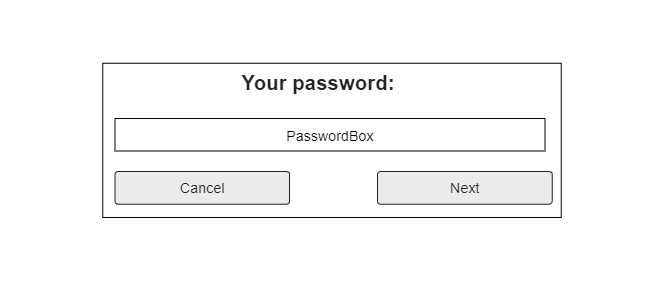


Рисунок 5 – Окно входа

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



Рисунок 6 – Вкладка «Main» окна botserver\_standard

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

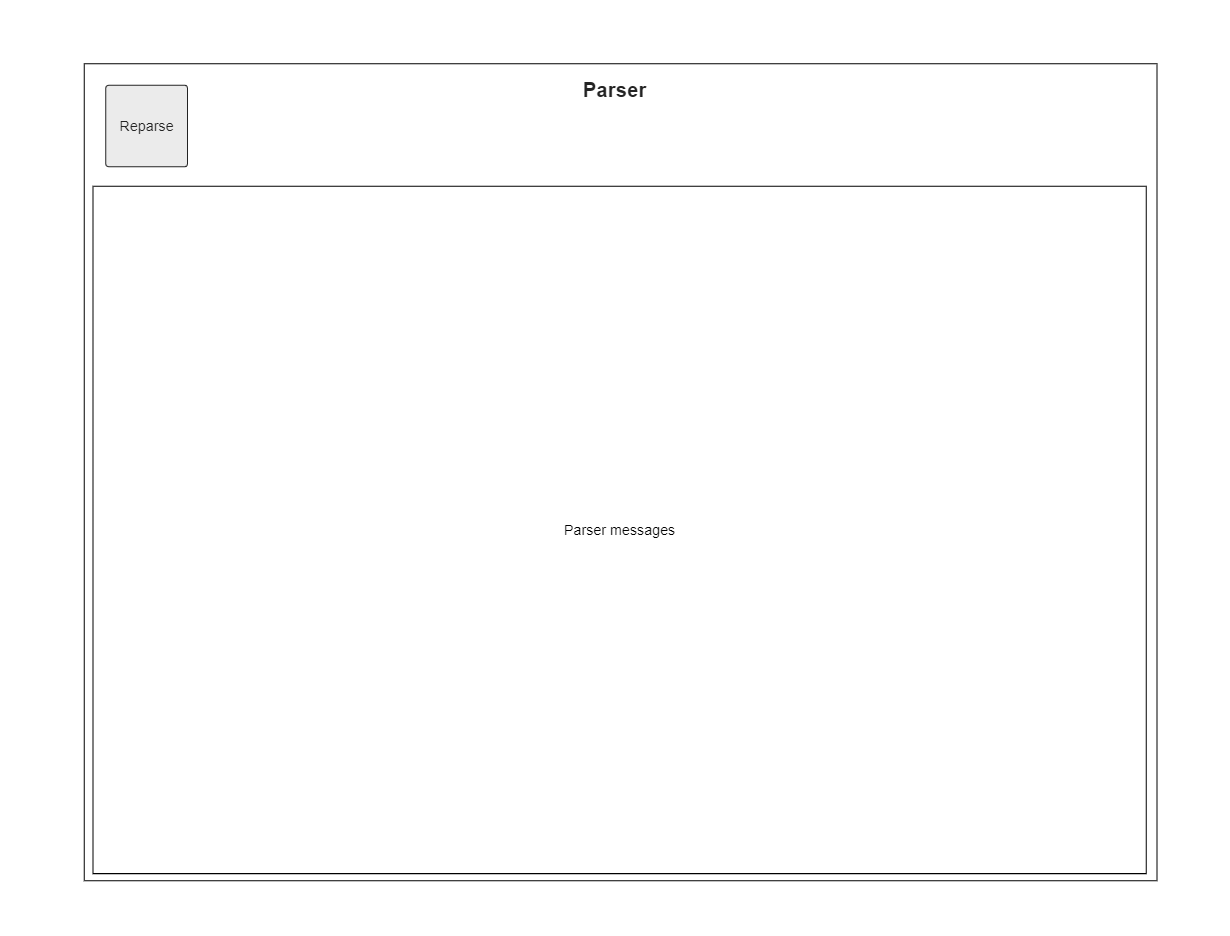


Рисунок 7 – Вкладка «Parser» окна botserver\_standard

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

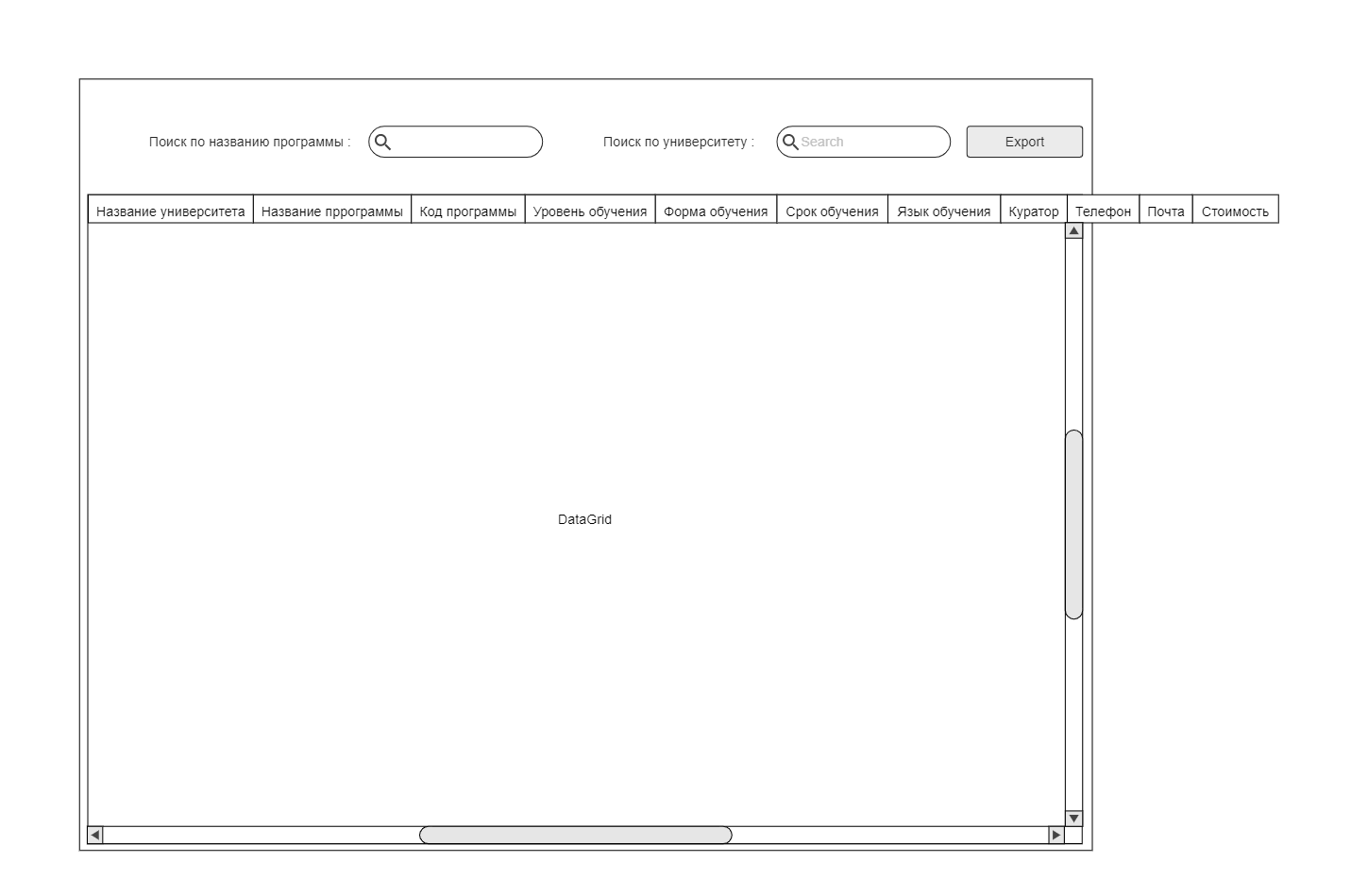


Рисунок 8 – Вкладка «Parsed cards» окна botserver\_standard

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



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

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

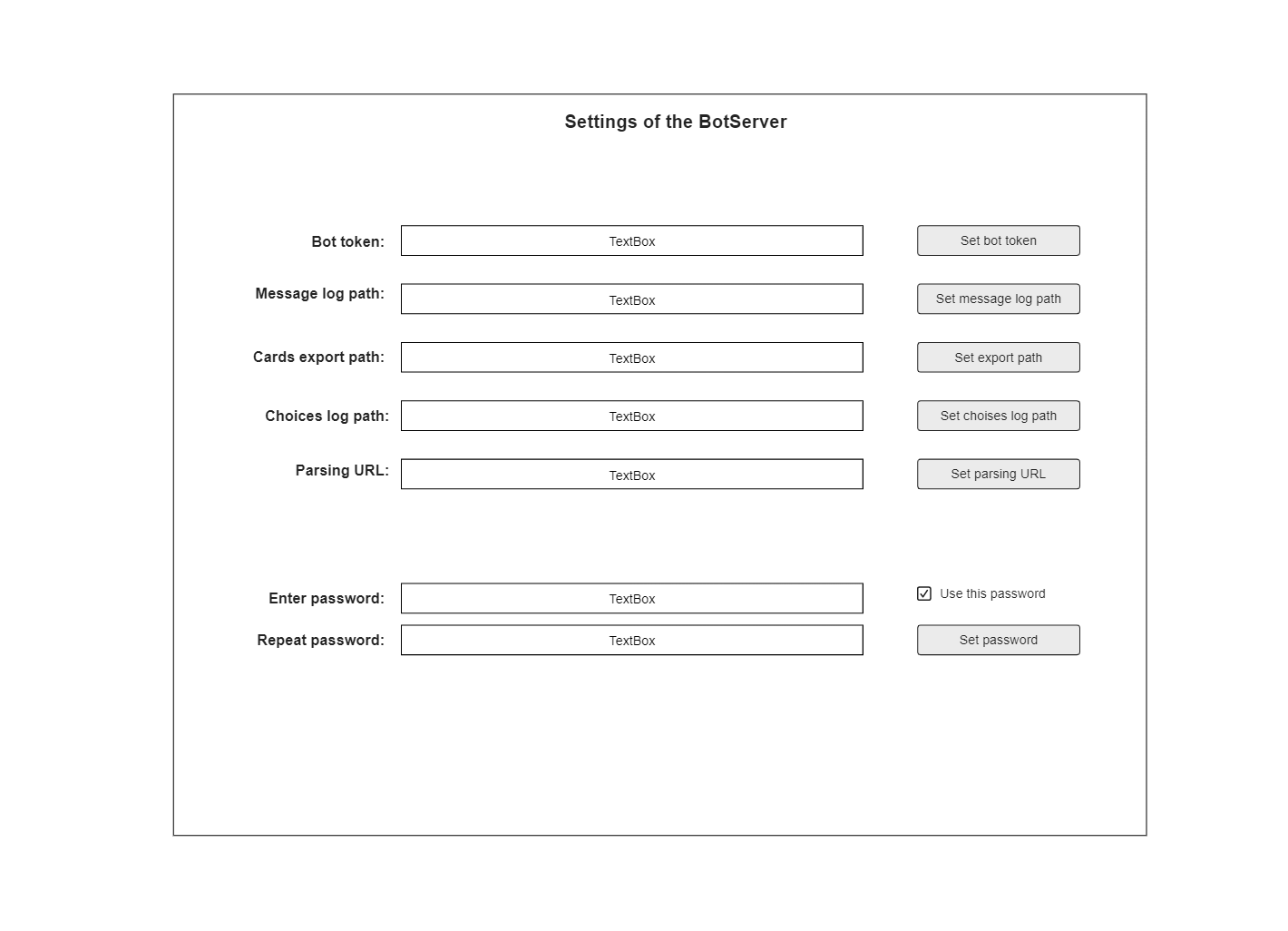


Рисунок 10 – Вкладка «Settings» окна «botserver\_standard»

Заключение

В результате прохождения производственной практики была разработана информационная система учета кадров для ИП Жданов, которая без ошибок и сбоев выполняет заявленные функции:

1. Изменение персональных данных сотрудников

2. Просмотр посещаемости сотрудников

3. Просмотр отпусков сотрудников

4. Составление заявок на отпуск

5. Просмотр и изменение статус заявок на отпуск

6. Создание\удаление должностей

7. Создание\удаление отделов

8. Просмотр сотрудников

9. Поиск сотрудников

10. Фильтрация сотрудников

11. Составление трудового договора

12. Составление соглашения о расторжении трудового договора

13. Составление отчета по посещаемости

14. Составление отчета по отпускам

15. Составление отчета о текучке кадров

В результате был разработан функционал облегчающий процесс учета кадров в ИП Жданов.

Перечень используемых источников

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

1. 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.
2. 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.
3. IT-GOST.RU. Теория и практика UML. Диаграмма состояний // [Электронный ресурс] – Режим доступа: <http://it-gost.ru/articles/view_articles/97>. Дата обращения: 5.03.2023
4. METANIT.COM. Полное руководство по языку программирования С# 10 и платформе .NET 6 // [Электронный ресурс] – Режим доступа: <https://metanit.com/sharp/tutorial/> . Дата обращения: 5.03.2023
5. 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.
6. METANIT.COM. Введение в WPF // [Электронный ресурс] – Режим доступа: <https://metanit.com/sharp/wpf/1.php> . Дата обращения: 5.03.2023.
7. 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.

**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.dbStructureRessurection);

}

catch

{

return;

}

finally

{

//чтение настроек

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

}

}

}

}

**Содержимое файла AskingPassword.xaml**

<Window x:Class="botserver\_standard.AskingPassword"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:botserver\_standard"

mc:Ignorable="d"

Width="450"

Height="150"

Background="#FF4A2E98"

ResizeMode="NoResize"

WindowStyle="None"

WindowStartupLocation="CenterScreen">

<Grid>

<Label

Margin="0,10,0,0"

HorizontalAlignment="Center"

VerticalAlignment="Top"

Content="Your password:"

FontFamily="Cascadia Mono"

FontSize="15"

Foreground="#FFB4B4B4" />

<PasswordBox

x:Name="EnterPwdBox"

HorizontalAlignment="Center"

VerticalContentAlignment ="Center"

Margin="0,42,0,0"

VerticalAlignment="Top"

Width="410" Height="22"

Background="#FF5F3CC0"

Foreground="#FFB4B4B4"/>

<Button

x:Name="CancelBtn"

Background="#FF5F3CC0"

Content="Cancel"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Margin="20,93,0,0"

Height="22"

HorizontalAlignment="Left"

Width="155"

VerticalAlignment="Top" IsCancel="True" />

<Button

x:Name="NextBtn"

Background="#FF5F3CC0"

Click="NextBtn\_Click"

Content="Next"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Margin="275,93,0,0"

Height="22"

HorizontalAlignment="Left"

Width="155"

VerticalAlignment="Top" />

</Grid>

</Window>

**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(?)

/\*

Всего строк: 6001

Строк на одну карточку: 12

Карточек: 500

\*/

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:**

<Window x:Class="botserver\_standard.ChangeDefaultPwd"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:botserver\_standard"

mc:Ignorable="d"

Width="450"

Height="150"

Background="#FF4A2E98"

ResizeMode="NoResize"

WindowStyle="None"

WindowStartupLocation="CenterScreen">

<Grid>

<Label

Margin="0,10,0,0"

HorizontalAlignment="Center"

VerticalAlignment="Top"

Content="Set a new password:"

FontFamily="Cascadia Mono"

FontSize="15"

Foreground="#FFB4B4B4" />

<PasswordBox

x:Name="EnterPwdBox"

HorizontalAlignment="Center"

VerticalContentAlignment ="Center"

Margin="0,42,0,0"

VerticalAlignment="Top"

Width="410" Height="22"

Background="#FF5F3CC0"

Foreground="#FFB4B4B4"/>

<Button

x:Name="CancelBtn"

Background="#FF5F3CC0"

Content="Cancel"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Margin="20,104,0,0"

Height="22"

HorizontalAlignment="Left"

Width="155"

VerticalAlignment="Top" IsCancel="True" />

<Button

x:Name="NextBtn"

Background="#FF5F3CC0"

Click="NextBtn\_Click"

Content="Next"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Margin="275,104,0,0"

Height="22"

HorizontalAlignment="Left"

Width="155"

VerticalAlignment="Top" />

<PasswordBox

x:Name="EnterPwdBox\_Repeated"

HorizontalAlignment="Center"

VerticalContentAlignment ="Center"

Margin="0,69,0,0"

VerticalAlignment="Top"

Width="410" Height="22"

Background="#FF5F3CC0"

Foreground="#FFB4B4B4"/>

</Grid>

</Window>

**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.ProgramName}\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 dbStructureRessurection =

"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? choisedLevel;

static string? choisedUniversity;

static string? choisedProgram;

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 приныятых 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.ToLower() == "/start") //if recieved Message update type

{

var 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:

///главное меню (выбор программы) -

///выбор уровня (фиксированный ответ)

///выбор вуза(фиксированный ответ)

///выбор направления(доступные ответы генерятся на основе предыдущих записанных callbackData)

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!";

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);

}

//отправка клавиатуры выбора уровня

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);

}

//отправка клавиатуры выбора университета

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

{

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

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

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

}

//отправка клавиатуры выбора программы

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

{

string telegramMessage = "Подобранные программы обучения:";

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

// фильтрация карточек на основании выборов юзера

List<Card> filteredCardsByClient = new();

foreach (var item in cardsView)

{

if (choisedLevel == item.Level && choisedUniversity == item.UniversityName)

filteredCardsByClient.Add(item);

}

//генерация кнопок на основе отфильтрованных карточек

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

foreach (var item in cardsView)

{

if (item.Level == choisedLevel && item.UniversityName == choisedUniversity)

filteredUniversitiesButtons.Add(InlineKeyboardButton.WithCallbackData(text: item.ProgramName, callbackData: Convert.ToString(item.Id)));

}

var dynamicProgramChoosingKeypad = new InlineKeyboardMarkup(filteredUniversitiesButtons);

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 сообщения с данными о выбранном направлении

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

{

choisedProgram = update.CallbackQuery.Data;

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

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: "Связаться", url: $"mailto:{finalSelectedCard.Email}"),

},

// second row

new[]

{

InlineKeyboardButton.WithCallbackData(text: "На главную", callbackData: "toHome"),

},

});

await botClient.EditMessageTextAsync(update.CallbackQuery.Message.Chat.Id, update.CallbackQuery.Message.MessageId, telegramMessage, replyMarkup: lastButtonsKeypad, parseMode: Telegram.Bot.Types.Enums.ParseMode.Html, cancellationToken: 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}\nTimestamp: {DateTime.Now}",

\_ => exception.ToString()

};

Dispatcher.Invoke(() =>

{

return LiveLogOutput.Text += $"{ErrorMessage}\n" + "-------------------------------------------------------------------------------------------------------------\n";

});

return Task.CompletedTask;

}

}

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)

{

ConsoleWorker.CardOutputter();

}

#region other right stackpanel buttons

private void OutputPauseBtn\_Click(object sender, RoutedEventArgs e)

{

MessageBox.Show("Not implemented");

}

private void ExportBtn\_Click(object sender, RoutedEventArgs e)

{

MessageBox.Show("Not implemented");

}

private void OutputClrBtn\_Click(object sender, RoutedEventArgs e)

{

LiveLogOutput.Clear();

}

#endregion

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 {choisedLevel}, университет {choisedUniversity} 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 {choisedLevel}, университет {choisedUniversity} 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}', '{choisedLevel}', '{choisedUniversity}', '{card.ProgramName}', '{DateTime.Now}')";

DbWorker.DbQuerySilentSender(DbWorker.sqliteConn, query); //запись истории паролей

}

public static async Task PrepairedMessageSender(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);

}

}

}

**Содержимое файла MainWindow.xaml:**

<Window

x:Class="botserver\_standard.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:botserver\_standard"

Title="botserver\_standard"

Width="1050"

Height="800"

Background="#FF4A2E98"

WindowStartupLocation="CenterScreen"

mc:Ignorable="d">

<Grid>

<TabControl x:Name="tabControl">

<TabItem

x:Name="mainTab"

Background="#FF4A2E98"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Header="Main">

<Grid Background="#FF4A2E98">

<TextBox

x:Name="LiveLogOutput"

Margin="10,126,10,10"

Background="Black"

FontFamily="Cascadia Mono"

Foreground="White"

IsReadOnly="True"

TextWrapping="Wrap" />

<StackPanel HorizontalAlignment="Right" Width="355">

<Button

x:Name="OutputPauseBtn"

Height="40"

Background="#FF5F3CC0"

Click="OutputPauseBtn\_Click"

Content="Pause logging"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4" />

<Button

x:Name="ExportBtn"

Height="40"

Background="#FF5F3CC0"

Click="ExportBtn\_Click"

Content="Export log"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4" />

<Button

x:Name="OutputClrBtn"

Height="40"

Background="#FF5F3CC0"

Click="OutputClrBtn\_Click"

Content="Clear output"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4" />

</StackPanel>

<Button

x:Name="CmdOpenBtn"

Background="#FF5F3CC0"

Click="CmdOpenBtn\_Click"

Content="Open cmd"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Margin="265,10,0,0"

Width="80"

HorizontalAlignment="Left"

Height="80"

VerticalAlignment="Top" />

<Button

x:Name="BotLogoutBtn"

Background="#FF5F3CC0"

Click="StopBotBtn\_Click"

Content="STOP"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Margin="95,10,0,0"

HorizontalAlignment="Left"

Width="80"

Height="80"

VerticalAlignment="Top" />

<Button

x:Name="BotStartBtn"

Background="#FF5F3CC0"

Click="BotStartBtn\_Click"

Content="START"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Margin="10,10,0,0"

HorizontalAlignment="Left"

Width="80"

IsDefault="True"

Height="80"

VerticalAlignment="Top" />

<Button

x:Name="StopExitBotBtn"

Background="#FF5F3CC0"

Click="StopExitBotBtn\_Click"

Content="Stop &amp; exit"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Margin="180,10,0,0"

HorizontalAlignment="Left"

Width="80"

IsCancel="True"

Height="80"

VerticalAlignment="Top" />

</Grid>

</TabItem>

<TabItem

x:Name="parserTab"

Background="#FF4A2E98"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Header="Parser">

<Grid Background="#FF4A2E98">

<TextBox

x:Name="ParserLogOutput"

Margin="10,127,10,9"

Background="Black"

FontFamily="Cascadia Mono"

Foreground="White"

IsReadOnly="True"

TextWrapping="Wrap" />

<Button

x:Name="ReparseBtn"

Background="#FF5F3CC0"

Click="ParserPeparseBtn\_Click"

Content="Reparse"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4" Margin="10,42,0,0"

HorizontalAlignment="Left"

Width="80"

IsDefault="True"

Height="80"

VerticalAlignment="Top" />

<Label

Margin="0,10,0,0"

HorizontalAlignment="Center"

VerticalAlignment="Top"

Content="Parser"

FontFamily="Cascadia Mono"

FontSize="15"

Foreground="#FFB4B4B4" Width="62" />

</Grid>

</TabItem>

<TabItem

x:Name="parsedCardsTab"

Background="#FF4A2E98"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Header="Parsed cards">

<Grid Background="#FF4A2E98">

<DataGrid x:Name="parsedCardsGrid" AutoGenerateColumns="False" CanUserAddRows="False" CanUserDeleteRows="False" CanUserResizeRows="False" IsReadOnly="True" RenderTransformOrigin="0.504,1.97" Foreground="Black" BorderBrush="#FFABADB3" Background="#FF5F3CC0" Margin="0,42,0,0">

<DataGrid.Columns>

<!--<DataGridTextColumn Header="Id:" Binding="{Binding cardsView.Id}"/>-->

<DataGridTextColumn Header="Название университета" Binding="{Binding UniversityName}"/>

<DataGridTextColumn Header="Название программы" Binding="{Binding ProgramName}"/>

<DataGridTextColumn Header="Код программы" Binding="{Binding ProgramCode}"/>

<DataGridTextColumn Header="Уровень обучения" Binding="{Binding Level}"/>

<DataGridTextColumn Header="Форма обучения" Binding="{Binding StudyForm}"/>

<DataGridTextColumn Header="Срок обучения" Binding="{Binding Duration}"/>

<DataGridTextColumn Header="Язык обучения" Binding="{Binding StudyLang}"/>

<DataGridTextColumn Header="Куратор" Binding="{Binding Curator}"/>

<DataGridTextColumn Header="Телефон" Binding="{Binding PhoneNumber}"/>

<DataGridTextColumn Header="Почта" Binding="{Binding Email}"/>

<DataGridTextColumn Header="Стоимость" Binding="{Binding Cost}"/>

</DataGrid.Columns>

</DataGrid>

<TextBox

x:Name = "SearchByProgramName"

HorizontalAlignment ="Left"

Margin="264,10,0,0"

TextWrapping="Wrap"

VerticalContentAlignment="Center"

VerticalAlignment="Top"

Width="199"

Height="27"

KeyUp="SearchByProgramName\_KeyUp"

Background="#FF5F3CC0"

Foreground="#FFABADB3"/>

<TextBox

x:Name = "SearchByUniversity"

HorizontalAlignment ="Left"

Margin="709,10,0,0"

TextWrapping="Wrap"

VerticalContentAlignment="Center"

VerticalAlignment="Top"

Width="199" Height="27"

KeyUp="SearchByUniversity\_KeyUp"

Background="#FF5F3CC0"

Foreground="#FFABADB3"/>

<Label

Margin="10,10,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

Content="Поиск по названию программы:"

FontFamily="Cascadia Mono"

FontSize="15"

Foreground="#FFB4B4B4" Width="254" />

<Label

Margin="497,10,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

Content="Поиск по университету:"

FontFamily="Cascadia Mono"

FontSize="15"

Foreground="#FFB4B4B4" Width="207" />

<Button x:Name="CardsExportBtn" Content="Export" HorizontalAlignment="Left" Margin="913,10,0,0" VerticalAlignment="Top" Height="27" Width="71" Background="#FF5F3CC0" Foreground="#FFB4B4B4" Click="CardsExportBtn\_Click"/>

</Grid>

</TabItem>

<TabItem

x:Name="parsedUniversitiesTab"

Background="#FF4A2E98"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Header="Parsed universities">

<Grid Background="#FF4A2E98">

<!--<DataGrid

x:Name = "parsedUniversitiesGrid"

Margin="0,42,0,0"

IsReadOnly="True"

Foreground="Black"

BorderBrush="#FFABADB3"

Background="#FF5F3CC0"/>-->

<DataGrid x:Name="parsedUniversitiesGrid" AutoGenerateColumns="False" CanUserAddRows="False" CanUserDeleteRows="False" CanUserResizeRows="False" IsReadOnly="True" RenderTransformOrigin="0.504,1.97" Foreground="Black" BorderBrush="#FFABADB3" Background="#FF5F3CC0" Margin="0,42,0,0">

<DataGrid.Columns>

<!--<DataGridTextColumn Header="Id:" Binding="{Binding cardsView.Id}"/>-->

<DataGridTextColumn Header="Название университета" Binding="{Binding UniversityName}"/>

<DataGridTextColumn Header="Количество программ" Binding="{Binding Count}"/>

</DataGrid.Columns>

</DataGrid>

<TextBox

x:Name = "SearchByUniversityName"

HorizontalAlignment ="Left"

Margin="220,10,0,0"

TextWrapping="Wrap"

VerticalContentAlignment="Center"

VerticalAlignment="Top"

Width="199"

Height="27"

KeyUp="SearchByUniversityName\_KeyUp"

Background="#FF5F3CC0"

Foreground="#FFABADB3"/>

<TextBox

x:Name = "SearchByUniversityFreq"

HorizontalAlignment ="Left"

Margin="769,10,0,0"

TextWrapping="Wrap"

VerticalContentAlignment="Center"

VerticalAlignment="Top"

Width="112" Height="27"

KeyUp="SearchByUniversityFreq\_KeyUp"

Background="#FF5F3CC0"

Foreground="#FFABADB3"/>

<Label

Margin="10,10,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

Content="Поиск по университету:"

FontFamily="Cascadia Mono"

FontSize="15"

Foreground="#FFB4B4B4" Width="254" />

<Label

Margin="497,10,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

Content="Поиск по количеству программ:"

FontFamily="Cascadia Mono"

FontSize="15"

Foreground="#FFB4B4B4" Width="267" />

<Button

x:Name="UniversitiesExportBtn"

Content="Export"

HorizontalAlignment="Left"

Margin="913,10,0,0"

VerticalAlignment="Top"

Height="27"

Width="71"

Background="#FF5F3CC0"

Foreground="#FFB4B4B4"

Click="UniversitiesExportBtn\_Click"/>

</Grid>

</TabItem>

<TabItem

x:Name="settingsTab"

Background="#FF4A2E98"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Header="Settings">

<Grid Background="#FF4A2E98">

<TextBox

x:Name="SettingsTokenInput"

Width="495"

Height="22"

Margin="230,71,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

VerticalContentAlignment="Center"

Background="#FF5F3CC0"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

TextWrapping="NoWrap" />

<Label

Width="169"

Height="22"

Margin="56,71,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

HorizontalContentAlignment="Right"

Content="Bot token:"

FontFamily="Cascadia Mono"

FontWeight="Bold"

Foreground="#FFB4B4B4" />

<Label

Width="169"

Height="25"

Margin="56,101,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

HorizontalContentAlignment="Right"

Content="Messages log path:"

FontFamily="Cascadia Mono"

FontWeight="Bold"

Foreground="#FFB4B4B4" />

<Label

Width="169"

Height="26"

Margin="56,130,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

HorizontalContentAlignment="Right"

Content="Cards export path:"

FontFamily="Cascadia Mono"

FontWeight="Bold"

Foreground="#FFB4B4B4" />

<TextBox

x:Name="MessagesLogRootInput"

Width="495"

Height="22"

Margin="230,101,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

VerticalContentAlignment="Center"

Background="#FF5F3CC0"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

TextWrapping="NoWrap" />

<TextBox

x:Name="datagridExportPath"

Width="495"

Height="22"

Margin="230,131,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

VerticalContentAlignment="Center"

Background="#FF5F3CC0"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

TextWrapping="NoWrap" />

<Label

Margin="0,10,0,0"

HorizontalAlignment="Center"

VerticalAlignment="Top"

Content="Settings of the BotServer"

FontFamily="Cascadia Mono"

FontSize="15"

Foreground="#FFB4B4B4" />

<Button

x:Name="SettingsFilePathSetBtn"

Background="#FF5F3CC0"

Click="ChoicesLogRootInputSetBtn\_Click"

Content="Set choices log path"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Margin="0,160,66,0"

Height="22"

VerticalAlignment="Top" HorizontalAlignment="Right" Width="155" />

<Button

x:Name="SetTokenBtn"

Background="#FF5F3CC0"

Click="SetTokenBtn\_Click"

Content="Set bot token"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Margin="0,71,66,0"

Height="22"

VerticalAlignment="Top" HorizontalAlignment="Right" Width="155" />

<Button

x:Name="SetLogPathBtn"

Background="#FF5F3CC0"

Click="SetMessageLogPathBtn\_Click"

Content="Set message log path"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Margin="0,101,66,0"

Height="22"

VerticalAlignment="Top" HorizontalAlignment="Right" Width="155" />

<Button

x:Name="SetDbPathBtn"

Background="#FF5F3CC0"

Click="SetDatagridExportPathBtn\_Click"

Content="Set export path"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Margin="0,131,66,0"

Height="22"

VerticalAlignment="Top" HorizontalAlignment="Right" Width="155" />

<Label

Width="169"

Height="24"

Margin="56,161,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

HorizontalContentAlignment="Right"

Content="Choices log path:"

FontFamily="Cascadia Mono"

FontWeight="Bold"

Foreground="#FFB4B4B4" />

<TextBox

x:Name="ChoicesLogRootInput"

Width="495"

Height="22"

Margin="230,161,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

VerticalContentAlignment="Center"

Background="#FF5F3CC0"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

TextWrapping="NoWrap" />

<Button

x:Name="ParsingUrlSetBtn"

Background="#FF5F3CC0"

Content="Set parsing URL"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Margin="0,189,66,0"

Height="22"

VerticalAlignment="Top"

Click="ParsingUrlSetBtn\_Click" HorizontalAlignment="Right" Width="155" />

<Label

Width="175"

Height="24"

Margin="50,190,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

HorizontalContentAlignment="Right"

Content="Parsing URL:"

FontFamily="Cascadia Mono"

FontWeight="Bold"

Foreground="#FFB4B4B4" FontSize="11" />

<TextBox

x:Name="UrlSet"

Width="495"

Height="22"

Margin="230,191,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

VerticalContentAlignment="Center"

Background="#FF5F3CC0"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

TextWrapping="NoWrap" />

<Label

Width="169"

Height="24"

Margin="56,260,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

HorizontalContentAlignment="Right"

Content="Enter password:"

FontFamily="Cascadia Mono"

FontWeight="Bold"

Foreground="#FFB4B4B4" />

<Label

Width="169"

Height="24"

Margin="56,291,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

HorizontalContentAlignment="Right"

Content="Repeat password:"

FontFamily="Cascadia Mono"

FontWeight="Bold"

Foreground="#FFB4B4B4" />

<Button

x:Name="SetPwdBtn"

Background="#FF5F3CC0"

Click="SetPwdBtn\_Click"

Content="Set password"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Margin="0,292,66,0"

Height="22"

VerticalAlignment="Top" HorizontalAlignment="Right" Width="155" />

<PasswordBox

x:Name="SetPwdBox"

HorizontalAlignment="Left"

VerticalContentAlignment ="Center"

Margin="230,261,0,0"

VerticalAlignment="Top"

Width="495" Height="22"

Background="#FF5F3CC0"

Foreground="#FFB4B4B4"/>

<PasswordBox

x:Name="SetRepeatedPwdBox"

HorizontalAlignment="Left"

VerticalContentAlignment ="Center"

Margin="230,292,0,0"

VerticalAlignment="Top"

Width="495" Height="22"

Background="#FF5F3CC0"

Foreground="#FFB4B4B4"/>

<CheckBox

x:Name="UseThisPwdCheckbox"

Content="Use this pwd"

Margin="0,261,66,0"

VerticalAlignment="Top"

Background="#FF5F3CC0"

Foreground="#FFB4B4B4"

Height="22"

Checked="UseThisPwdCheckbox\_Checked"

Unchecked="UseThisPwdCheckbox\_Unchecked" HorizontalAlignment="Right" Width="154"/>

<Label x:Name="PwdChangeNotifier"

Width="411"

Height="24"

Margin="230,319,0,0"

HorizontalAlignment="Left"

VerticalAlignment="Top"

HorizontalContentAlignment="Center"

Content=""

FontFamily="Cascadia Mono"

FontWeight="Bold"

Foreground="#FFB4B4B4" />

</Grid>

</TabItem>

<TabItem

x:Name="aboutTab"

Background="#FF4A2E98"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Header="About">

<Grid Background="#FF4A2E98" />

</TabItem>

</TabControl>

</Grid>

</Window>

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

using System;

using System.Runtime.InteropServices;

using System.Threading.Tasks;

using System.Windows;

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

}

}

}

**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();

public async void CardParser(SqliteConnection sqliteConn)

{

Dispatcher.Invoke(() =>

{

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

ParserLogOutput.Text += $"{DateTime.Now} | Parsing in process...\n";

});

Dispatcher.Invoke(() =>

{

ParserLogOutput.Text += $"{DateTime.Now} | Getting data from web...\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} | Selecting nodes...\n";

});

if (document is null)

{

return;

}

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

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

Dispatcher.Invoke(() =>

{

ParserLogOutput.Text += $"{DateTime.Now} | Processing received data...\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, идущими подряд

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

//удаление мусора из листа

Dispatcher.Invoke(() =>

{

ParserLogOutput.Text += $"{DateTime.Now} | Cleaning data...\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

/\*

Всего строк: 6501

Строк на одну карточку: 12

Карточек: 500

\*/

//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} | Writing data...\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

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

{

//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;

}

//ТУСУР - 78 раз

//ТПУ - 203 раз

//ТГПУ - 52 раз

//ТГАСУ - 45 раз

//ТГУ - 122 раз

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();

List < UniversityEntryFreq > universityFreqListView= new();

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} | Parsing is done.\n{Card.cards.Count} cards have been added;\n{universityFreqListView.Count} universities have been added.\n";

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

});

}

}

}

**Содержимое файла ParserTab.cs:**

using System;

using System.Threading.Tasks;

using System.Windows;

namespace botserver\_standard

{

public partial class MainWindow : Window

{

private void ParserPeparseBtn\_Click(object sender, RoutedEventArgs e)

{

Card.cards.Clear();

Dispatcher.Invoke(() =>

{

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

});

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

}

}

}

**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

}

}

**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]; // :AAF3nNDlYNfryOulNHKtsxlhuGo\_roxXYXI

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

}

else

{

SettingsTokenInput.Text = "Unforseen error";

}

}

private void SetMessageLogPathBtn\_Click(object sender, RoutedEventArgs e) //ok

{

string query = $"UPDATE Settings SET logPath = '{MessagesLogRootInput.Text}';";

int rowsChanged = DbWorker.DbQuerySilentSender(DbWorker.sqliteConn, query);

//IsChanged?

if (rowsChanged is 1)

{

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

MessagesLogRootInput.Text = $"file path has been setted.";

}

else

{

MessagesLogRootInput.Text = "Unforseen 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 SqliteDataReader 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:**

<Window x:Class="botserver\_standard.StupidWall"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:botserver\_standard"

mc:Ignorable="d"

Width="450"

Height="150"

Background="#FF4A2E98"

ResizeMode="NoResize"

WindowStyle="None"

WindowStartupLocation="CenterScreen">

<Grid>

<Label

Margin="0,10,0,0"

HorizontalAlignment="Center"

VerticalAlignment="Top"

Content="Your password:"

FontFamily="Cascadia Mono"

FontSize="15"

Foreground="#FFB4B4B4" />

<PasswordBox

x:Name="EnterPwdBox"

HorizontalAlignment="Center"

VerticalContentAlignment ="Center"

Margin="0,42,0,0"

VerticalAlignment="Top"

Width="410" Height="22"

Background="#FF5F3CC0"

Foreground="#FFB4B4B4"/>

<Button

x:Name="CancelBtn"

Background="#FF5F3CC0"

Content="Cancel"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Margin="20,93,0,0"

Height="22"

HorizontalAlignment="Left"

Width="155"

VerticalAlignment="Top" IsCancel="True" />

<Button

x:Name="NextBtn"

Background="#FF5F3CC0"

Click="NextBtn\_Click"

Content="Next"

FontFamily="Cascadia Mono"

Foreground="#FFB4B4B4"

Margin="275,93,0,0"

Height="22"

HorizontalAlignment="Left"

Width="155"

VerticalAlignment="Top" />

</Grid>

</Window>

**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[]

{

// first row

new[]

{

InlineKeyboardButton.WithCallbackData(text: "Выбрать программу обучения", callbackData: "programChoose"),

InlineKeyboardButton.WithUrl(text: "Проверить знание русского языка", url: "https://studyintomsk.2i.tusur.ru/"),

},

// second row

new[]

{

InlineKeyboardButton.WithUrl(text: "Посетить веб-сайт проекта", url: "https://studyintomsk.ru/"),

//InlineKeyboardButton.WithCallbackData(text: "Сменить язык", callbackData: "langSwitch"),

},

});

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: "На главную", callbackData: "toHome"),

},

});

public static readonly InlineKeyboardMarkup universityChoosingKeypad = new(

// keyboard

new[]

{

// first row

new[]

{

InlineKeyboardButton.WithCallbackData(text: "ТГАСУ", callbackData: "ТГАСУ\_university"),

InlineKeyboardButton.WithCallbackData(text: "ТГПУ", callbackData: "ТГПУ\_university"),

InlineKeyboardButton.WithCallbackData(text: "ТГУ", callbackData: "ТГУ\_university"),

},

// second row

new[]

{

InlineKeyboardButton.WithCallbackData(text: "ТПУ", callbackData: "ТПУ\_university"),

InlineKeyboardButton.WithCallbackData(text: "ТУСУР", callbackData: "ТУСУР\_university"),

//InlineKeyboardButton.WithCallbackData(text: "ФГБОУ ВО СибГМУ", callbackData: "sixth\_university"),

},

// third row

new[]

{

InlineKeyboardButton.WithCallbackData(text: "На главную", 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 UniversityEntryFreq.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;

}

}

}

**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;

}

}

}

После запуска приложения откроется окно для авторизации (Рисунок 22).

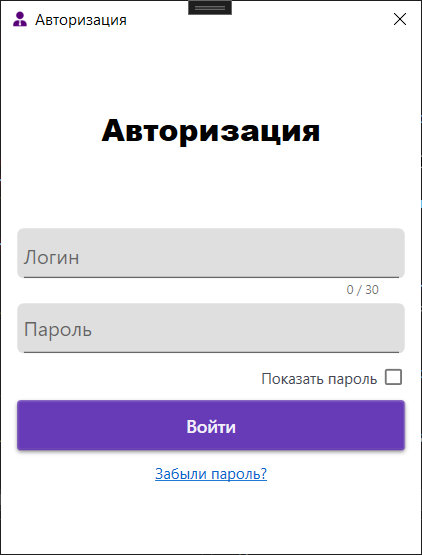
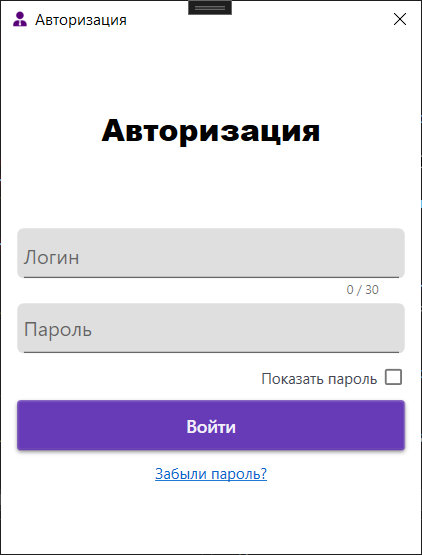
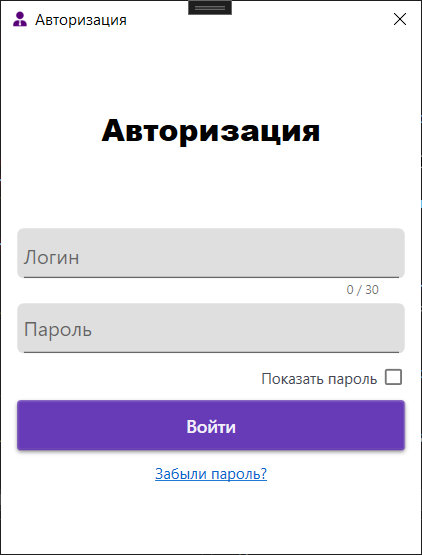


Рисунок 22 – Окно авторизации

В системе предусмотрено две роли: администратор и сотрудник. Войдя в систему как сотрудник откроется основное окно сотрудника, в котором доступны разделы





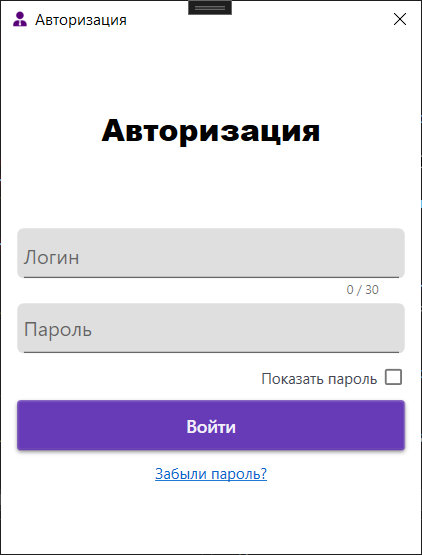


Диаграмма состояния «Установка пароля»

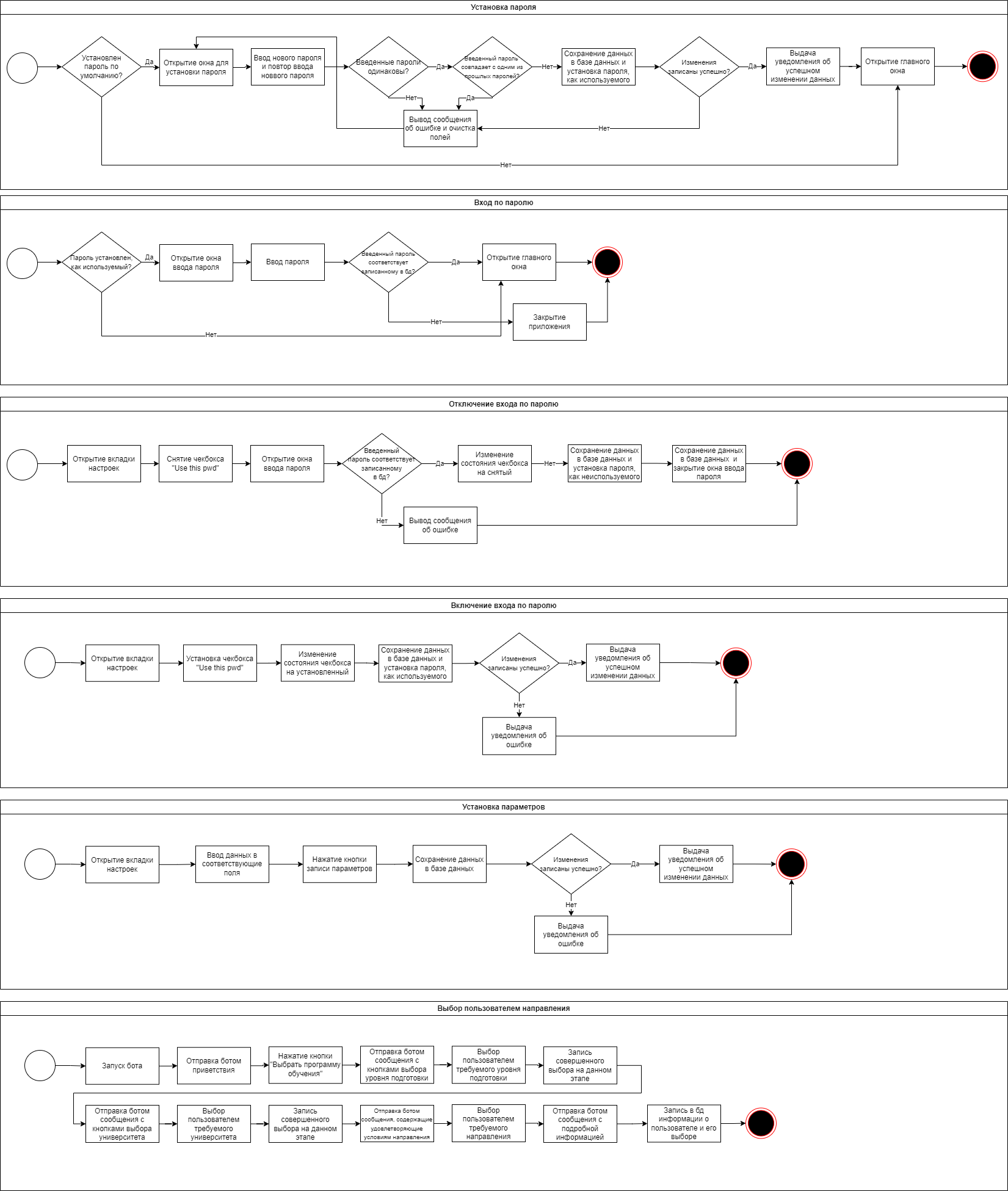
Рисунок Х – Диаграмма состояния «Установка пароля»

Диаграмма состояния «Вход по паролю»

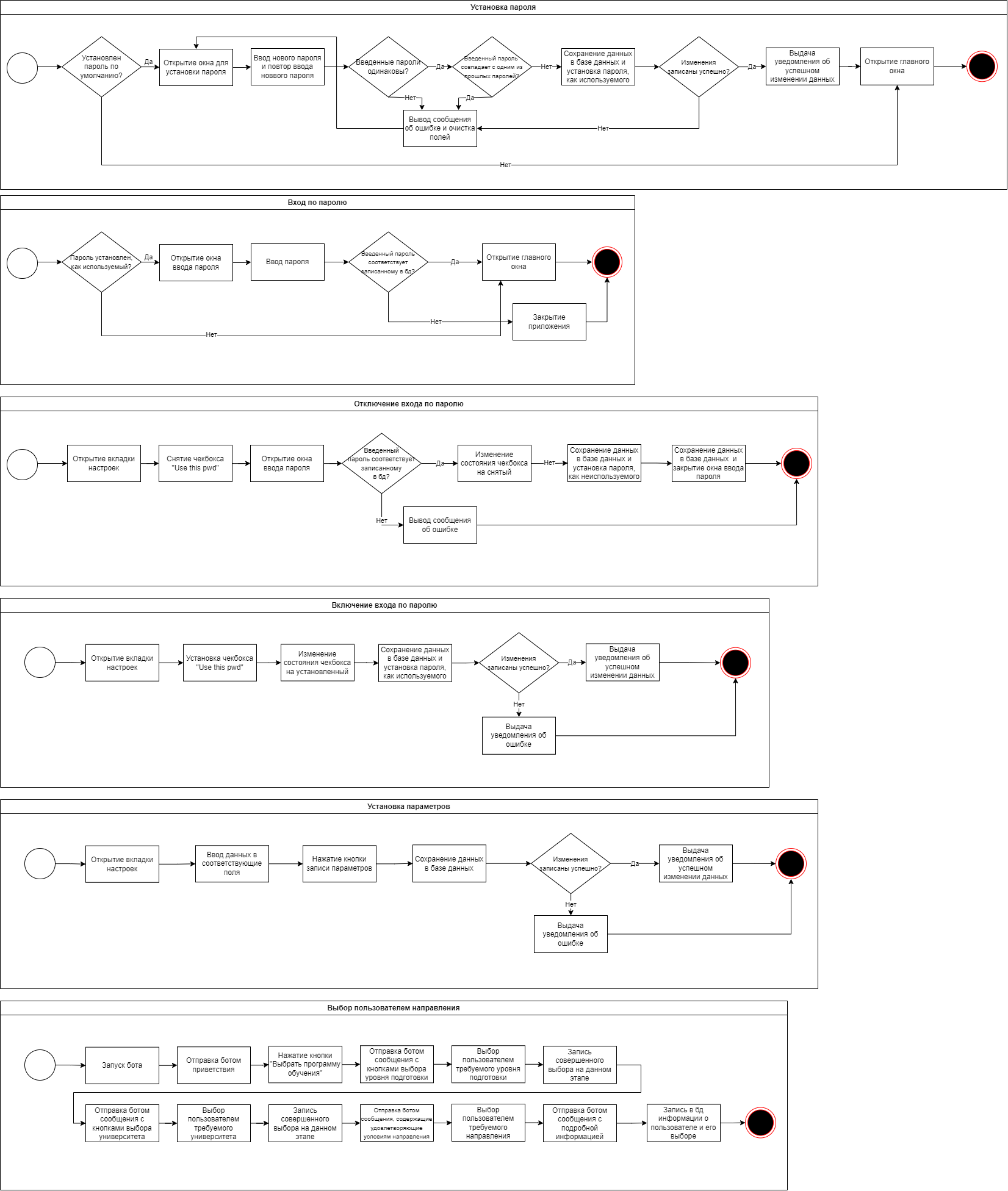


Рисунок Х – Диаграмма состояния «Вход по паролю»

Диаграмма состояния «Отключение входа по паролю»

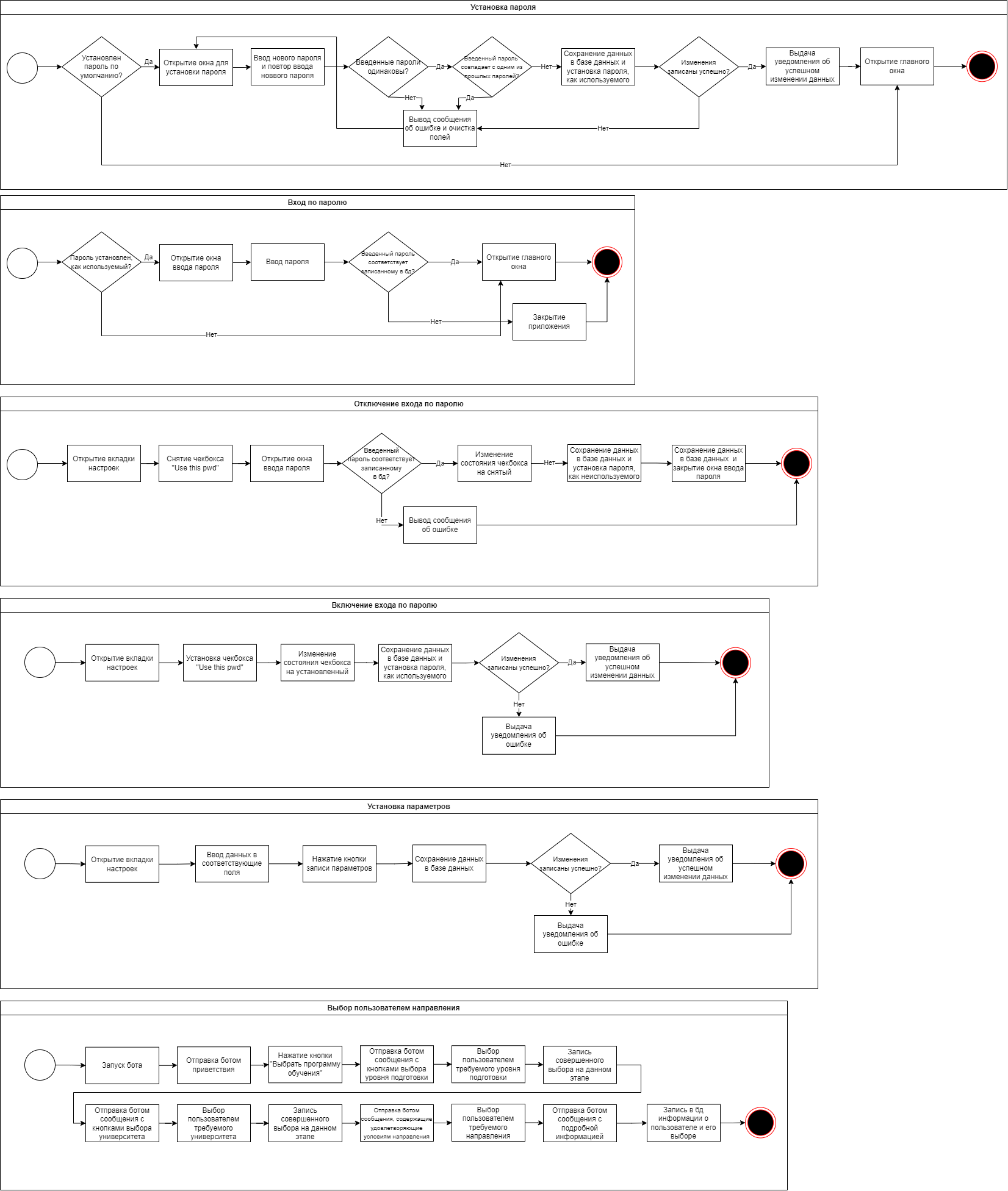


Рисунок Х – Диаграмма состояния «Отключение входа по паролю»

Диаграмма состояния «Включение входа по паролю»

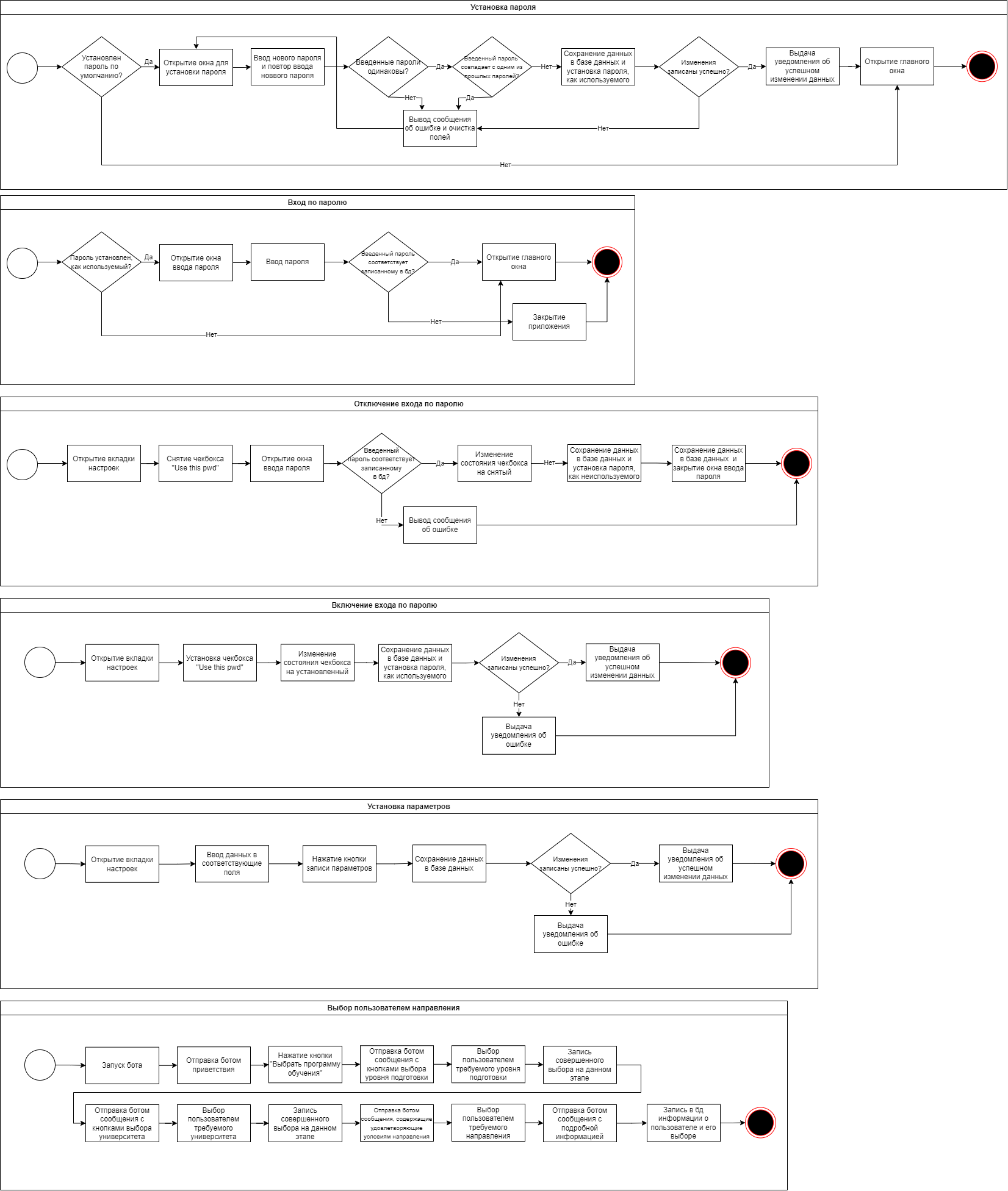


Рисунок Х – Диаграмма состояния «Включение входа по паролю»

Диаграмма состояния «Установка параметров»

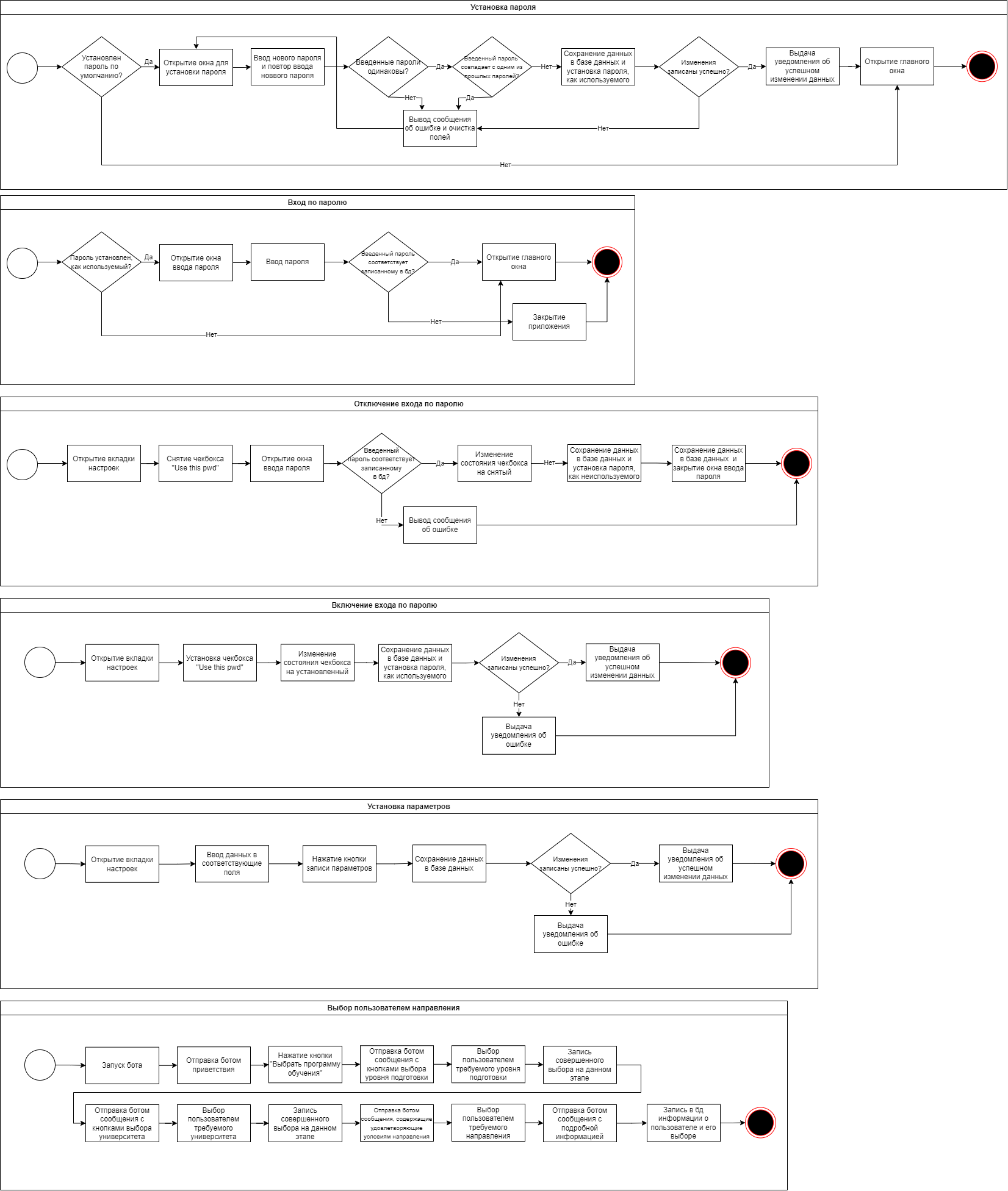


Рисунок Х – Диаграмма состояния «Установка параметров»

Диаграмма состояния «Выбор пользователем направления обучения»

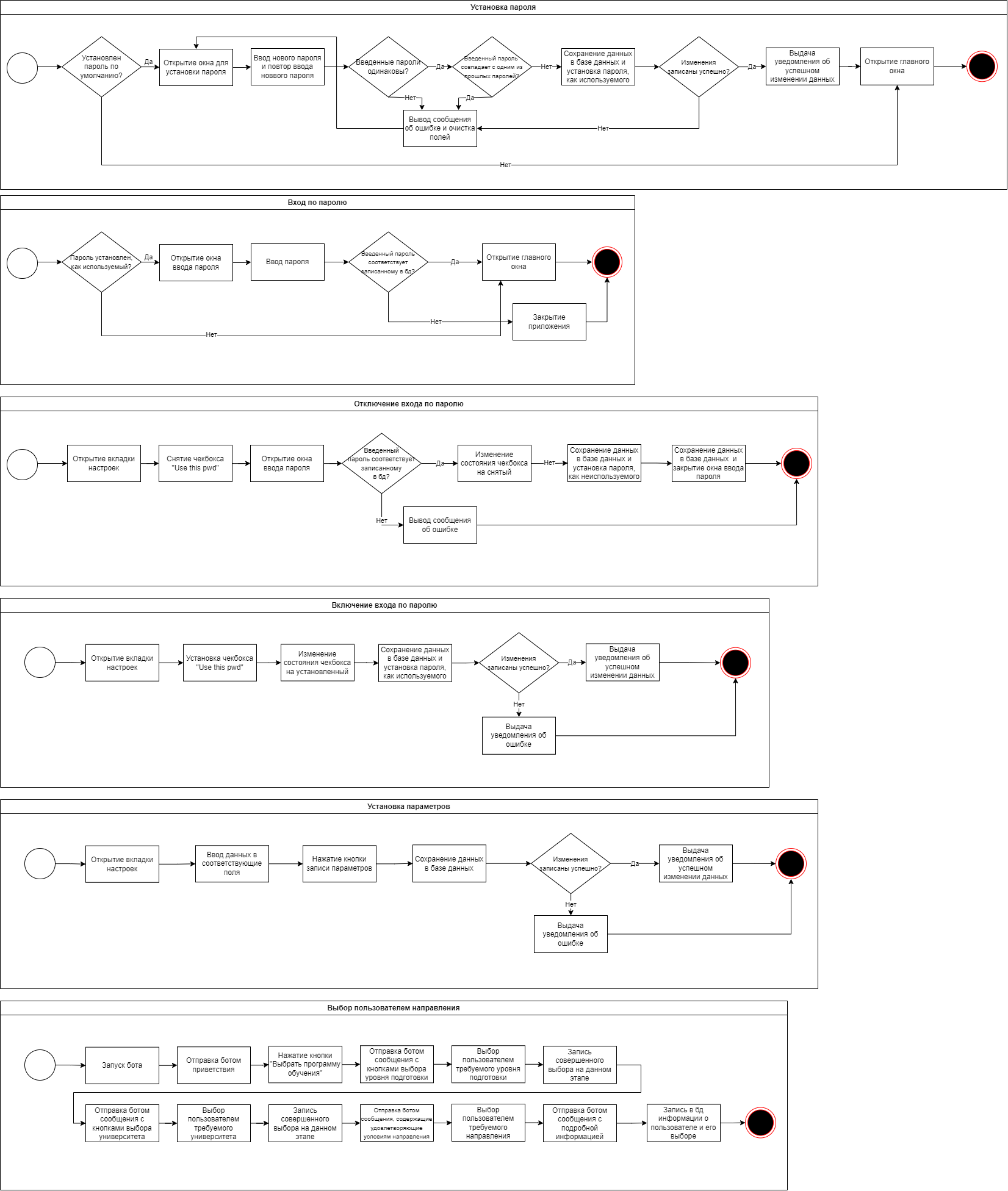


Рисунок Х – Диаграмма состояния «Выбор пользователем направления обучения»