

Колледж космического машиностроения и технологий

**КУРСОВОЙ ПРОЕКТ**

**По МДК.01.02 «Прикладное программирование»**

**Тема: «Приложение “Таск-анализатор репозиториев”. Клиентная часть»**

Выполнил студент

Волков М.А.

Группа П1-18

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_ (Дата сдачи работы)

Принял преподаватель

Гусятинер Л.Б.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Оценка)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Подпись)

**Королев 2021 г.**

Оглавление

[Введение 2](#_Toc72338220)

[1 Теоретическая часть 3](#_Toc72338221)

[1.1 Описание предметной области 3](#_Toc72338222)

[1.2 Описание существующих разработок 6](#_Toc72338223)

[1.2.1 Google Classroom 6](#_Toc72338224)

[1.2.2 Pear Deck 6](#_Toc72338225)

[2 Проектная часть 8](#_Toc72338226)

[2.1 Диаграмма прецедентов 8](#_Toc72338227)

[2.2 Выбор инструментов 9](#_Toc72338228)

[2.2.1 Язык программирования 9](#_Toc72338229)

[2.2.2 Среда разработки программного обеспечения 10](#_Toc72338230)

[2.2.3 База данных 11](#_Toc72338231)

[2.3 Проектирование сценария 12](#_Toc72338232)

[2.4 Диаграммы классов 13](#_Toc72338233)

[2.5 Описание главного модуля. 13](#_Toc72338234)

[2.6 Описание спецификаций к модулям 14](#_Toc72338235)

[2.7 Описание модулей 15](#_Toc72338236)

[2.8 Описание тестовых наборов модулей 17](#_Toc72338237)

[2.9 Описание применения средств отладки 19](#_Toc72338238)

[3 Эксплуатационная часть 20](#_Toc72338239)

[3.1 Руководство оператора 20](#_Toc72338240)

[3.1.1 Назначение программы 20](#_Toc72338241)

[3.1.2 Условия выполнения программы 20](#_Toc72338242)

[3.1.3 Выполнение программы 20](#_Toc72338243)

[3.1.4 Сообщение оператору 24](#_Toc72338244)

[3.2 To-Do лист 24](#_Toc72338245)

[Заключение 25](#_Toc72338246)

[Список литературы и интернет ресурсов 26](#_Toc72338247)

[Приложения 27](#_Toc72338248)

# Введение

Данный курсовой проект заключается в разработке настольного приложения. Разработка настольного приложения будет происходить на языке Pascal.

В первой части будут рассмотрены предметная область и существующие продукты по данной теме.

Во второй части будут описаны инструменты разработки программы и модули программы.

В третьей части описаны взаимодействие пользователя с программой и сообщения оператору.

В заключительной части будут приведены общие выводы по проекту.

# Теоретическая часть

## Описание предметной области

**Git**— распределённая система контроля версий, которая даёт возможность разработчикам отслеживать изменения в файлах и работать над одним проектом совместно с коллегами. Она была разработана в 2005 году Линусом Торвальдсом, создателем Linux, чтобы другие разработчики могли вносить свой вклад в ядро Linux.

У Git две основных задачи: первая — хранить информацию о всех изменениях в вашем коде, начиная с самой первой строчки, а вторая — обеспечение удобства командной работы над кодом.

Git известен своей скоростью, простым дизайном, поддержкой нелинейной разработки, полной децентрализацией и возможностью эффективно работать с большими проектами.

Подход Git к хранению данных похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок.

Преимущества Git:

**Бесплатный и open-source**. Можно бесплатно скачать и вносить любые изменения в исходный код;

**Небольшой и быстрый**. Выполняет все операции локально, что увеличивает его скорость. Кроме того, Git локально сохраняет весь репозиторий в небольшой файл без потери качества данных;

**Резервное копирование**. Git эффективен в хранении backup, поэтому известно мало случаев, когда кто-то терял данные при использовании Git;

**Простое ветвление**. В других системах контроля версий создание веток— утомительная и трудоёмкая задача, так как весь код копируется в новую ветку. В Git управление ветками реализовано гораздо проще и эффективнее.

Можно настроить свой собственный сервер для Git, но гораздо проще использовать Github. Это сервисы, предоставляющие сервера для repositories git и сопутствующие инструменты, совершенно бесплатно. На Github же бесплатно можно создавать только open repository. Close repository нужен для того, чтобы к вашему коду имели доступ только те, кому вы разрешите, а не все пользователи интернета.

Repository – это хранилище с кодом.

Commit – это фиксация изменений. Commit является минимальной “единицей” в git.

Branch – это альтернативная реальность кода.

Git работает так, что существует main repository и программист из него делает local repository, командой git clone, когда происходит клонирование, прописывают git commit и таким образом в local repository, программисты могут работать отдельно. Если им нужно передать друг другу данные, для этого есть команды:git push(который передает данные в main repository) и git pull(который принимает данные из main repository). Так же есть branch, который из себя представляет альтернативное изменение к какому-то commit. Если изменить branch, то commit не изменится.

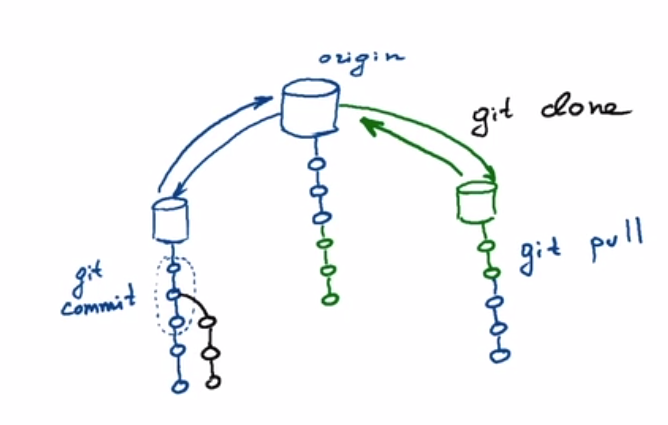


Рисунок 1. Структура Git

**Дистанционное обучение** (ДО) — взаимодействие учителя и учащихся между собой на расстоянии, отражающее все присущие учебному процессу компоненты (цели, содержание, методы, организационные формы, средства обучения) и реализуемое специфичными средствами Интернет-технологий или другими средствами, предусматривающими интерактивность.

6 основных черт дистанционного обучение: гибкость, адаптивность, доступность, интерактивность, универсальность, самообучение.

GitHub — сервис онлайн-хостинга repository, обладающий всеми функциями распределённого контроля версий и функциональностью управления исходным кодом — всё, что поддерживает Git и даже больше.

Github стал популярен за счет того, что имел обширные возможности для социального взаимодействия, фактически — социальная сеть для разработки open-source приложений.

**GitHub** также является одним из **крупнейших онлайн-хранилищ** (англ) совместной работы по всему миру.

## Описание существующих разработок

### Google Classroom

Педагоги, которым нужен более интерактивный вариант, найдут «Pear Deck» настоящим удовольствием. В дополнение к размещению ваших слайд-презентаций, эта программа позволяет оценивать любое количество ваших слайдов.

Студенты могут рисовать прямо в презентации, вводить свои собственные заметки, добавлять числа, перетаскивать символы или отвечать на вопросы с несколькими вариантами ответов. Если мало контента в презентации, это тоже может помочь. Pear Deck в партнерстве с NewSela создавал интерактивные темы для ежедневного обучения прямо из заголовков.

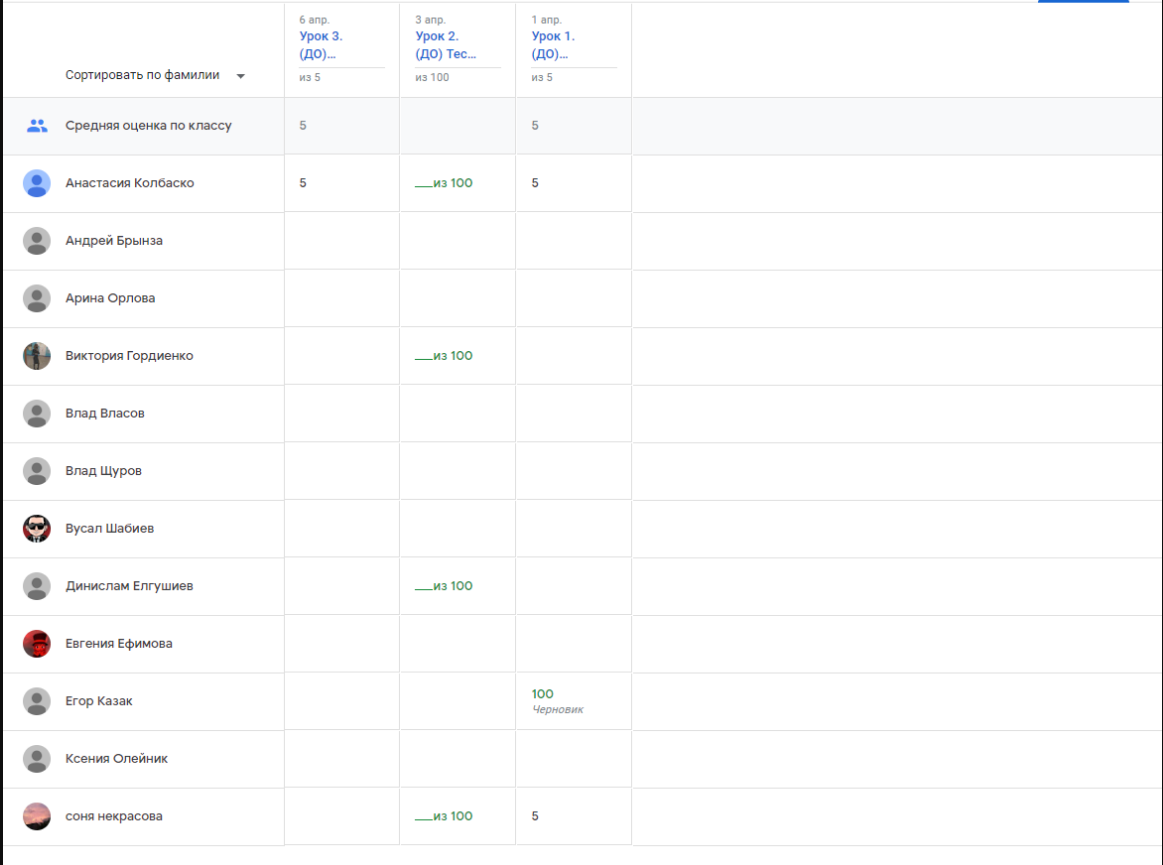


Рисунок 2. Работа с Google Classroom.

### Pear Deck

Педагоги, которым нужен более интерактивный компонент, найдут «Pear Deck» настоящим удовольствием. В дополнение к размещению ваших слайд-презентаций, эта программа позволяет оценивать любое количество ваших слайдов.

Студенты могут рисовать прямо в презентации, вводить свои собственные заметки, добавлять числа, перетаскивать символы или отвечать на вопросы с несколькими вариантами ответов. Если мало контента в презентации, это тоже может помочь. Pear Deck в партнерстве с Newsela создавал интерактивные темы для ежедневного обучения прямо из заголовков.

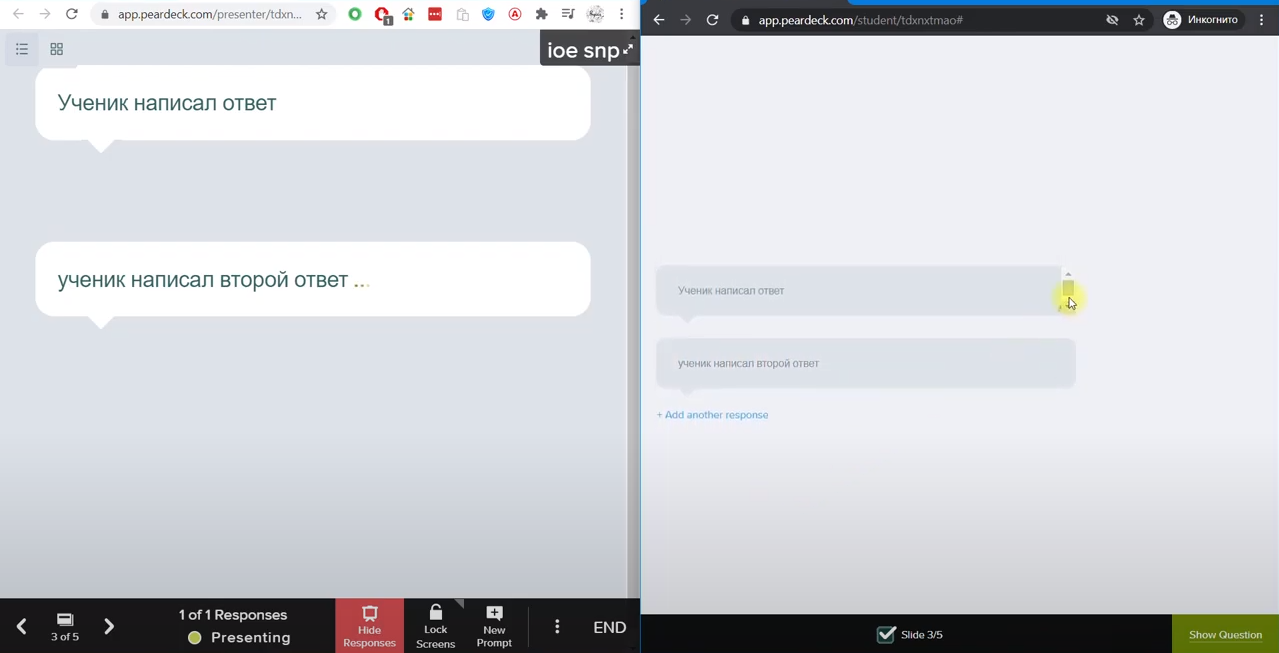


Рисунок 3. Работа с Pear Deck

# Проектная часть

## Диаграмма прецедентов

В данном разделе содержится диаграмма прецедентов для приложения, на которой показаны возможные функциональные отношения.

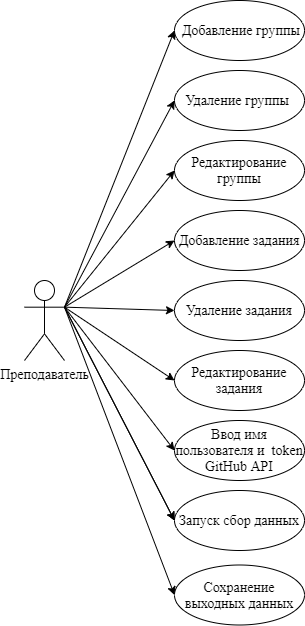


Рисунок 4. Диаграмма прецедентов.

## Выбор инструментов

В разработке приложения “Таск-анализатор репозиториев”, используется Pascal, в качестве СУБД использовался SQLite и в качестве IDE используется Lazarus и компилятор Free Pascal, который встроен в Lazarus.

### Язык программирования

**Pascal [1] -** Это чисто процедурный язык программирования, часто использующийся для обучения структурному программированию.

Pascal, который является ясным, логичным и гибким языком и приучает к хорошему стилю программирования. Кроме того, в средней школе основы программирования преподают именно на базе Pascal. Вместе с тем именно Pascal лежит в основе современной мощной системы визуального программирования Delphi, c помощью которой разрабатываются многие современные программные продукты.

Pascal был разработан Никлаусом Виртом в 1970; вопреки расхожему мнению, он не был исключительно учебным языком, а предназначался для практического применения. Прототипом послужил Algol. Первоначально язык компилировался в байт-код, подобно языку Java.

Особенностями языка являются строгая типизация и наличие средств структурного (процедурного) программирования. Pascal был одним из первых таких языков. Кроме того, язык предоставлял ряд встроенных структур данных: записи, массивы, файлы, множества и указатели.

Тем не менее, первоначально язык обладал множеством недостатков: невозможность передачи функциям массивов переменной длины, отсутствие нормальных средств работы с динамической памятью, ограниченная библиотека ввода-вывода, отсутствие средств для подключения функций, написанных на других языках, отсутствие средств раздельной компиляции и т.п.

Несмотря на то, что Pascal занимает маленькие места по разработке desktop, Pascal прост в освоении и с помощью его можно легко сделать интерфейс, который будет малотребовательный по ресурсам, отличие от других языков.

**Free Pascal [2] -** один из наиболее известных языков программирования, используется для обучения программированию в старших классах и на первых курсах вузов, является основой для ряда других языков. Free Pascal является очень мощным средством программирования, и вместе с тем за использование компиляторов студенту, школьнику и преподавателю не придется платить. Этим компилятором можно пользоваться абсолютно легально. Свободно распространяемые компиляторы с языка Free Pascal реализованы во многих дистрибутивах Linux, есть свободные компиляторы и для ОС Windows.

Преимущества Free Pascal на фоне остальных компиляторов – то, что он бесплатный и выходят все еще обновления. Free Pascal выступает как язык программирования и как компилятор. Использовать мы его будем как язык программирования, так и как компилятор.

### Среда разработки программного обеспечения

**Lazarus [3] -** это среда визуального программирования. Здесь программист получает возможность не просто создавать программный код, но и наглядно (визуально) показывать системе, что бы он хотел увидеть. Технология визуального программирования позволяет строить интерфейс будущей программы из специальных компонентов, реализующих нужные свойства. Количество таких компонентов достаточно велико. Каждый из них содержит готовый программный код и все необходимые для работы данные, что избавляет программиста от создания того, что уже создано ранее.

Подобный подход во много раз сокращает время написания программы. Кроме того, быстрота создания программного кода в Lazarus достигается за счет того, что значительная часть текста формируется автоматически. Среда визуального программирования Lazarus сочетает в себе компилятор (Free Pascal), объектно-ориентированные средства визуального программирования и различные технологии, облегчающие и ускоряющие создание программы.

На выбор IDE для проекта выбрался Lazarus, только из-за того, что нет альтернатив, кроме Delphi IDE, но Delphi IDE платный.

### База данных

**SQLite [4]** - это встроенная библиотека, которая реализует автономный, без серверный, нулевой конфигурации, транзакционный механизм СУБД SQL. Легко встраиваемая в приложения база данных. Так как это система базируется на файлах, то она предоставляет довольно широкий набор инструментов для работы с ней, по сравнению с сетевыми СУБД. При работе с этой СУБД обращения происходят напрямую к файлам (в этих файлах хранятся данные), вместо портов и сокетов в сетевых СУБД. Именно поэтому SQLite очень быстрый, а также мощный благодаря технологиям обслуживающих библиотек. Это база данных, которая настроена на нуль, что означает, как и другие базы данных, которые вам не нужно настраивать в вашей системе.

В качестве СУБД используется именно SQLite, потому что для него не нужен сервер. Но по сравнению с другими СУБД, он медленнее и у него нет системы пользователей, но именно из-за отсутствия сервера, используется SQLite.

Программа писалась на Pascal по следующим причинам:

|  |  |  |  |
| --- | --- | --- | --- |
| Язык программирования | С# | Pascal | Swift |
| Простота | 3 | 10 | 9 |
| Скорость разработки | 9 | 10 | 5 |
| Аудитория | 10 | 10 | 6 |
| Итог | 22 | 30 | 20 |

Таблица 1. Сравнение языков

## Проектирование сценария

В данном разделе приведен сценарий использования приложения пользователем (Рисунок 5).

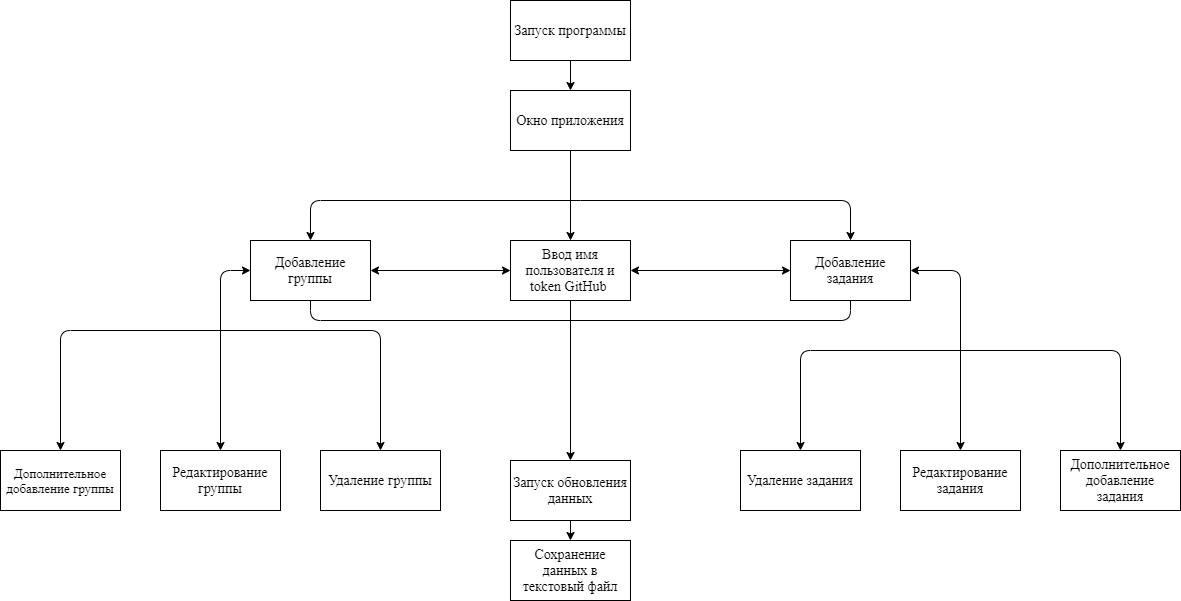


Рисунок 5. Сценарий использования приложения.

При запуске программы открывается окно приложения и у пользователя будет три действия на выбор, это: добавление группы, ввод имя пользователя и token GitHub, добавление задания. Неважно в каком порядке выполнять, главное сделать эти три действия, чтобы запустить обновление данных. Когда пользователь добавляет группу, появится 3 действия, это: редактирование группы, удаление группы и дополнительное добавление группы.

Когда пользователь добавляет задание, появится 3 действия, это: редактирование задания, удаление задания и дополнительное добавление задания.

После того, как пользователь обновил данные, их можно будет сохранить в текстовый файл.

## Диаграммы классов

В данном разделе располагается диаграмма классов приложения. (Рисунок 7).

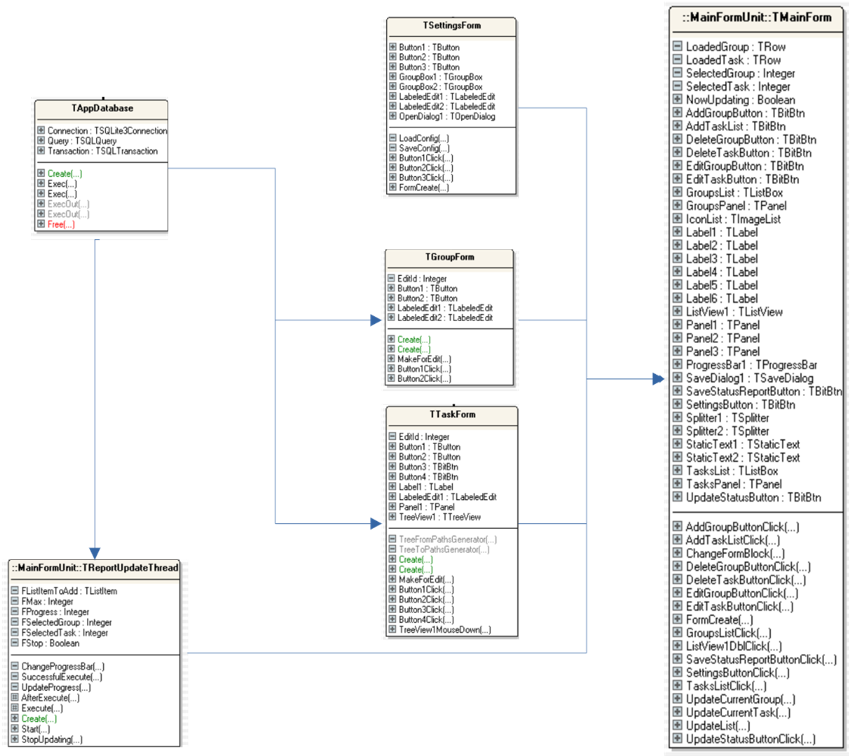


Рисунок 6. Диаграмма классов приложения

## Описание главного модуля.

Главный модуль состоит из главной функции, которая запускается при старте программы инициализирует класс Application и создает главную форму.

Листинг 1 Главная функция

program KkmtTesting;

{$mode objfpc}{$H+}

uses

{$IFDEF UNIX}

cthreads,

{$ENDIF}

{$IFDEF HASAMIGA}

athreads,

{$ENDIF}

Interfaces, // this includes the LCL widgetset

Forms, MainFormUnit, GroupFormUnit, taskformUnit, SettingsFormUnit;

{$R \*.res}

begin

RequireDerivedFormResource:=True;

Application.Scaled:=True;

Application.Initialize;

Application.CreateForm(TMainForm, MainForm);

Application.Run;

end.

## Описание спецификаций к модулям

Всего разработано 6 модулей:

* **MainFormUnit**
* **SettingsFormUnit**
* **TaskFormUnit**
* **GroupFormUnit**
* **AppDatabase**
* **Utils**

## Описание модулей

**MainFormUnit** – Модуль, который содержит класс главной формы программы.

Рассмотрим функцию запуска обновления выполненных веток и построим блок схему:

Листинг 2. Функция обновления

procedure TMainForm.UpdateStatusButtonClick(Sender: TObject);

begin

if not NowUpdating then

begin

if (SelectedGroup = -1) and (SelectedTask = -1) then exit;

Label2.Caption := StaticText2.Caption;

Label4.Caption := StaticText1.Caption;

Thread := TReportUpdateThread.Create(True, SelectedGroup, SelectedTask);

Thread.Start;

end

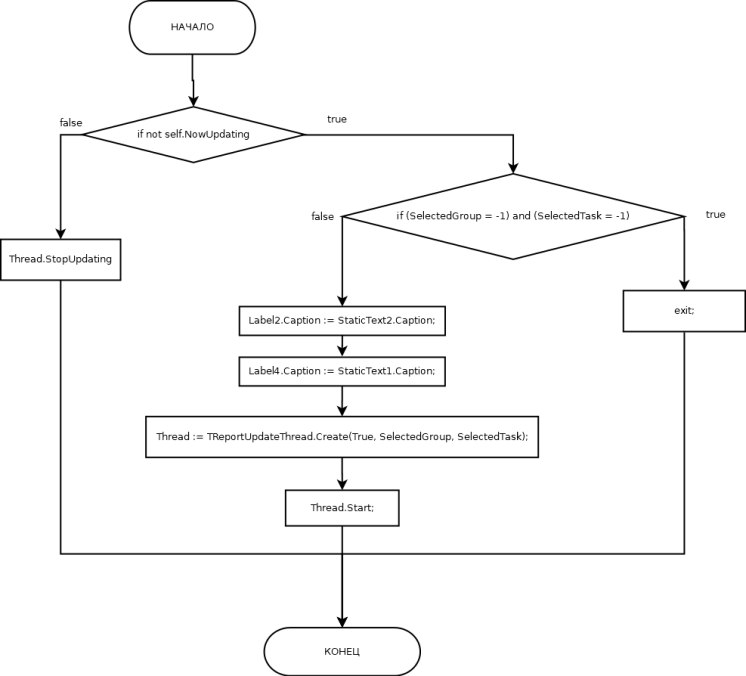
else

begin

Thread.StopUpdating;

end;

end;

Рисунок 7. Блок схема класса MainFormUnit

**SettingsFormUnit** - Модуль с классом формы настроек программы.

Имеет два текстовых поля:

* Логин пользователя GitHub
* Персональный токен для этого логина пользователя

При нажатии кнопки "Сохранить" записывает данные в поля в файл config.json в папке программы.

**TaskFormUnit** - Модуль с классом формы для добавления, изменения заданий.

Имеет поле ввода имени задания, а также дерево с ожидаемой структурой ветки начиная от корня ветки.

Также имеет три кнопки:

* Добавить элемент дерева как ребенка выделенного элемента
* Удалить выделенный элемент
* Изменить название выделенного элемента

**GroupFormUnit** - Модуль с классом формы для редактирования, добавления групп и их репозитория для проверки.

Имеет два поля:

* Имя группы
* Уникальный идентификатор репозитория группы

При сохранении данных этой формы выполняется проверка на существование репозитория.

**AppDatabase** - Модуль содержащий классы и функции для работы с базой данных приложения.

Имеет в себе класс TAppDatabase с методами для выполнения запросов с параметрами.

* Exec - для выполнения простого SQL запроса
* ExecOut - для выполнения SQL запроса с возвращением результата

**Utils** - Хранит в себе функции различного назначения.

* CreateGitHubHTTP - для инициализации объекта TFPHttpClient для работы с GitHub API
* OpenDialog - для открытия формы в виде простого диалога

## Описание тестовых наборов модулей

В некоторых модулях производится проверка данных на корректность, чтобы сообщить пользователю о проблеме и предотвратить аварийное завершение программы.

В GroupFormUnit происходит проверка на существование репозитория при нажатии на кнопку "Button1".

**Тест 1. Проверка имя группы**

Проверка происходит на пустоту имени группы и на пробелы. При непрохождении проверки выводится messagebox с сообщением "Имя группы не должно быть пустым или иметь пробелы".

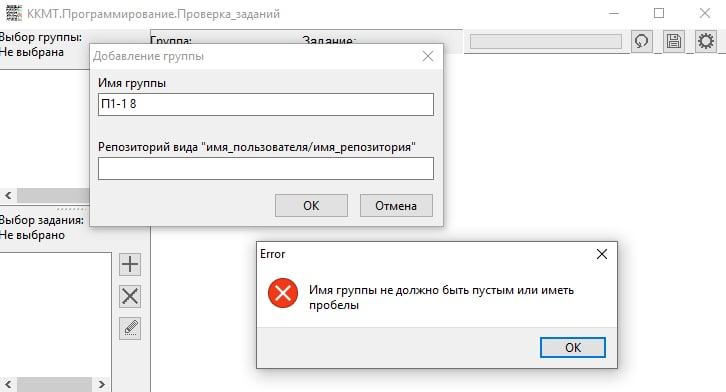


Рисунок 8. Ошибка при не правильном вводе группы

**Тест 2. Проверка заполнения вида репозитория**

Проверка на существование репозитория. Если он не существует, то будет показана ошибка "Репозиторий не найден".

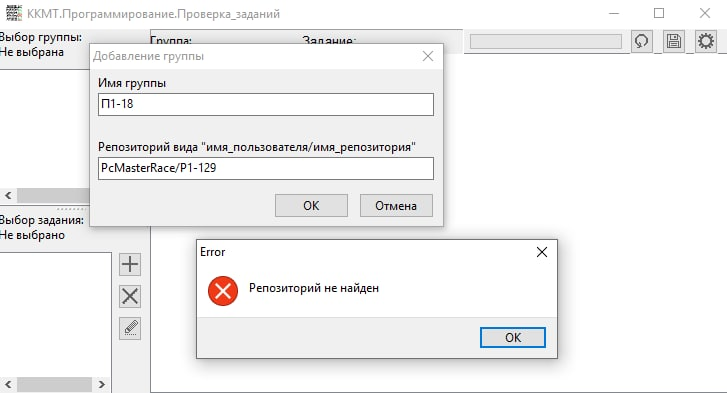


Рисунок 9. Ошибка при неправильном вводе репозитория

## Описание применения средств отладки

Во время разработки программы мы столкнулись с багом при котором не было проверки репозитория на существование. Поэтому при запуске обновления списка выполнения задания программа выдавала ошибку и не могла продолжить работу дальше.

После того как мы добавили проверку на наличие репозитория в диалог добавления новой группы, баг исчез (Рисунок 10)

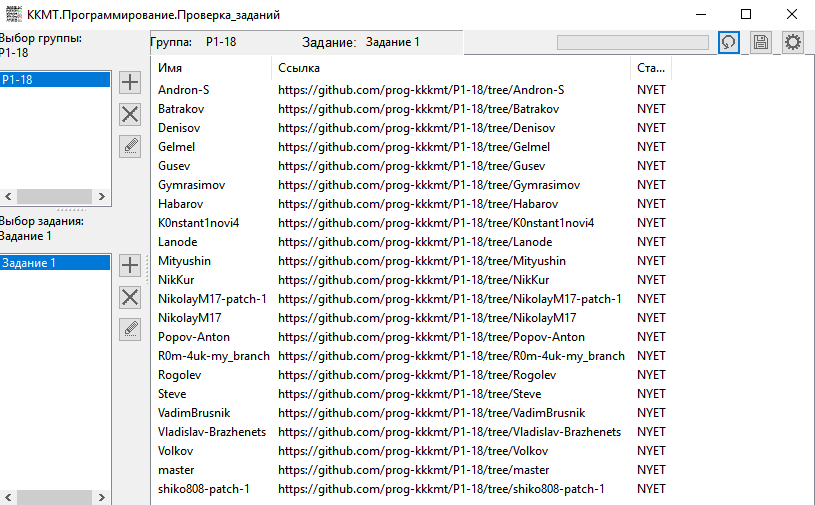


Рисунок 10. Успешная выдача результатов.

# Эксплуатационная часть

## Руководство оператора

### Назначение программы

Данная программа считывает список студентов заданной группы, смотрит есть ли заданное задание на сайте GitHub, и выводит список тех, кто выполнил и не выполнил задание.

### Условия выполнения программы

Операционная система: Windows, Linux.

Операционная память: 2 гб.

Доступного места на диске: 40 Мб.

### Выполнение программы

При запуске программы открывается окно приложения:

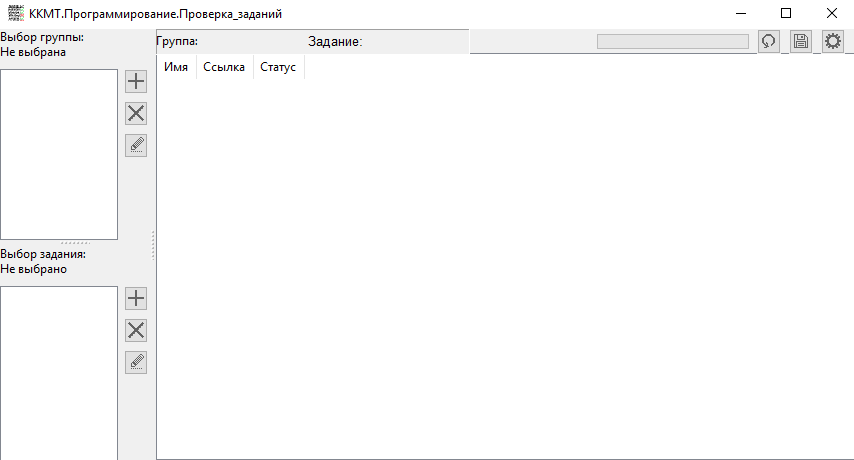


Рисунок 11. Окно приложения

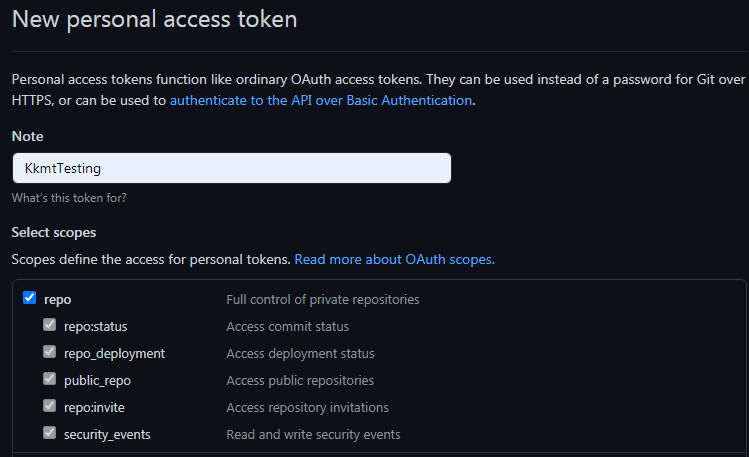
Для начала, нужно зайти на GitHub для получения token, чтобы авторизоваться в приложении. Получить token можно в GitHub. Чтобы получить токен, нужно нажать на аватарку в правом углу сверху и после нажать Settings. После того, как перешли в свой профиль, нажимаете Developer settings. После перехода в Developer settings, вам нужно нажать Personal access token. Чтобы сгенерировать токен, нажмите Generate new token. После нажатия на кнопку, нужно будет ввести название токена и поставить галочку в кладке repo и после нажать ниже Generate token. 

Рисунок 12. Получение token GitHub API

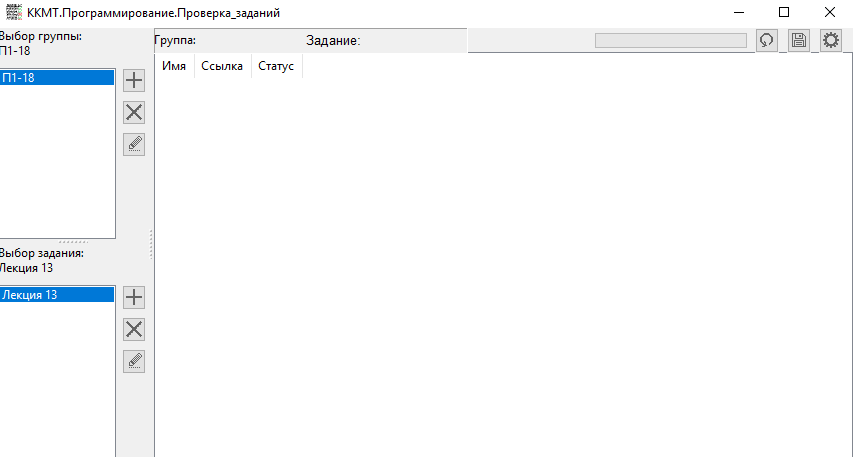
После того, как введен токен и имя, нужно выбрать группу и задание, для этого надо нажать плюсик. 

Рисунок 13. Выбор группы и задания

Затем надо ввести имя группы и задание в специальные строки, которые высвечиваются при нажатии плюсиков.

Мы выбрали группу и задание, поэтому чтобы получить результат, вам нужно нажать на закругленную стрелочку.

Статус выдает NYET, когда у студента отсутствует задание, а когда у студента есть это задание, выдает DA.

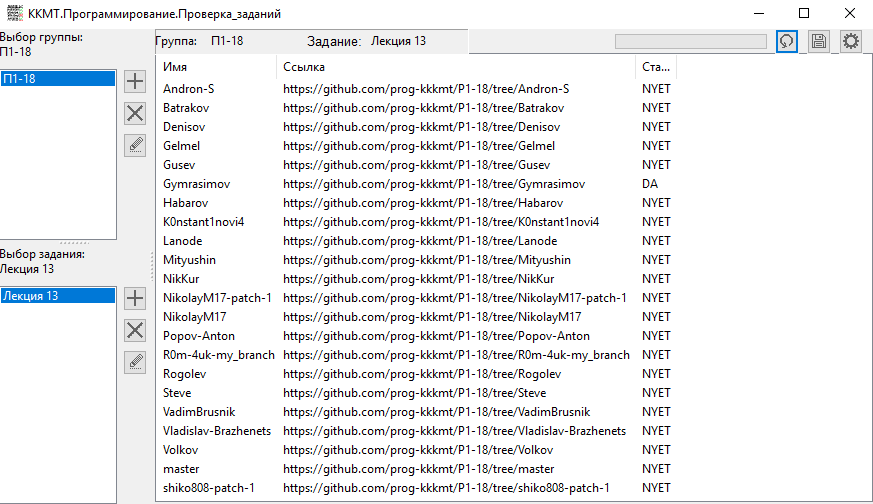


Рисунок 14. Вывод данных

Можно сохранить результат, для этого вам нужно нажать на значок дискеты в правом верхнем углу.

Сохраненный результат можно найти по пути: KKMT.Tasks(текущая версия)\build\Зависимость от выбора сборки\reports

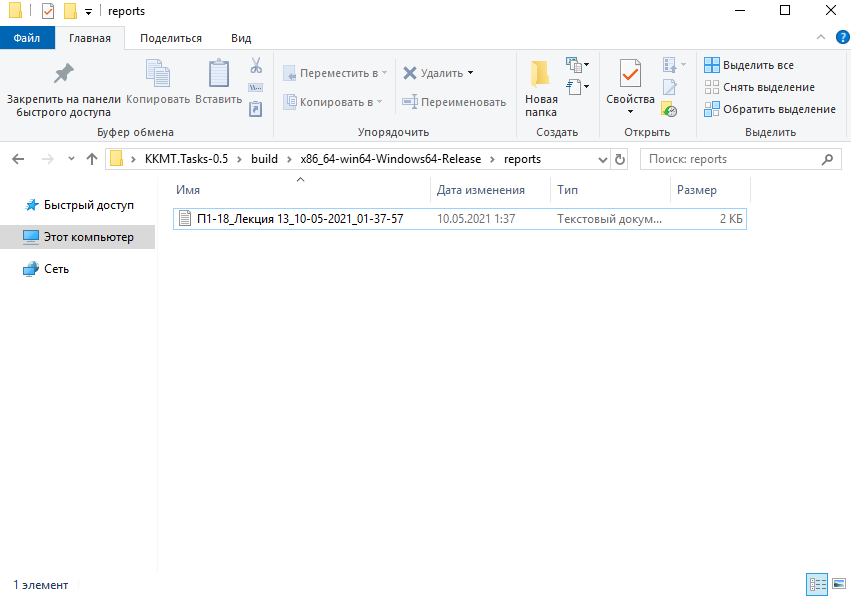


Рисунок 15. Путь к отчету

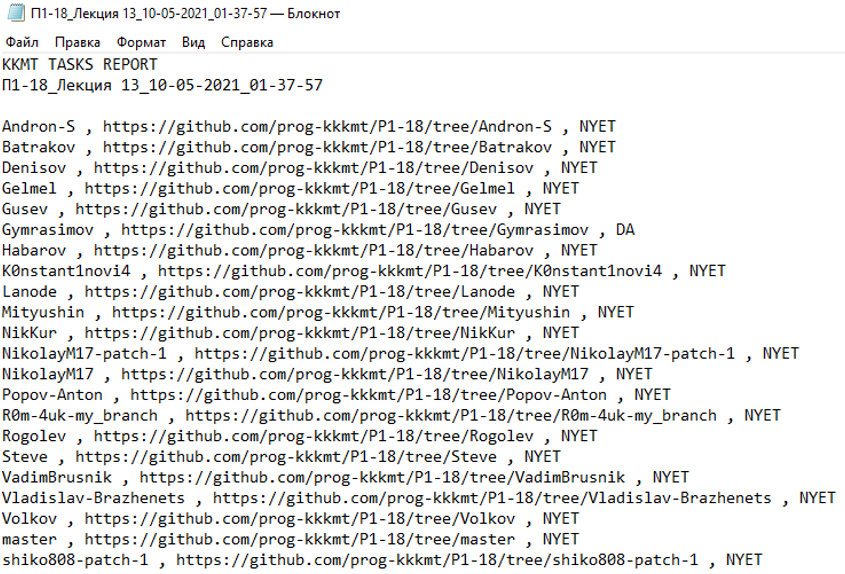


Рисунок 16. Сохраненный список.

### Сообщение оператору

При неправильном вводе данных в форму регистрации или авторизации появиться сообщение с допущенной ошибкой:

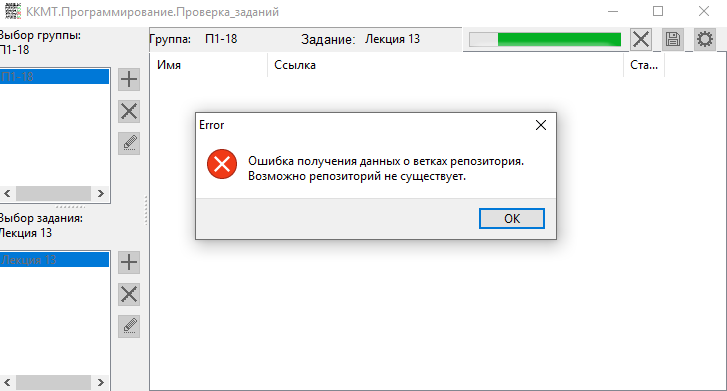


Рисунок 17. Сообщение об ошибке

**Причина:** не введен token GitHub API.

**Действия программы:** программа больше не может обрабатывать результат.

**Действия оператора:** ввести token GitHub API.

## To-Do лист

* Улучшить дизайн программы
* Упростить программу
* Добавить детальную проверку заданий

# Заключение

В результате этой работы было разработано приложение на языке Pascal для удобства отслеживания заданий студентов и сокращения времени преподавателя на проверку заданий.

Это приложение будет удобно во время дистанционного обучения так как облегчит нагрузку на преподавателя дома.

# Список литературы и интернет ресурсов

1. Ресурсы про дистанционное обучение

https://ru.wikipedia.org/wiki/Дистанционное\_обучение

<https://www.hp.com/us-en/shop/tech-takes/best-distance-learning-tools-for-teachers>

1. Git и GitHub

<https://blog.vistro.ru/git/>

https://tproger.ru/translations/difference-between-git-and-github/

https://[ru.stackoverflow.com/questions/116959/Для-чего-нужен-github](https://ru.stackoverflow.com/questions/116959/Для-чего-нужен-github)

1. Паскаль и Lazarus

<http://progopedia.ru/language/pascal/>

<https://teacher.ucoz.net/Lection/Pascal/Lazarus.pdf>

1. SQLite

<https://devacademy.ru/article/sqlite-vs-mysql-vs-postgresql>

# Приложения

**Приложение 1. MainFormUnit.pas**

unit MainFormUnit;

{$mode objfpc}{$H+}

interface

uses

Classes, SysUtils, Forms, Controls, Graphics, Dialogs, StdCtrls, CheckLst,

ExtCtrls, Grids, ValEdit, ComCtrls, SQLite3, jsonparser, fpjson, fphttpclient,

Types, LCLIntf, Buttons,

Utils, AppDatabase,

GroupFormUnit, TaskFormUnit, SettingsFormUnit;

type

{ TMainForm }

// Класс главной формы

TMainForm = class(TForm)

// Кнопка добавления группы

AddGroupButton: TBitBtn;

// Кнопока добавления группы

AddTaskList: TBitBtn;

// Кнопка удаления группы

DeleteGroupButton: TBitBtn;

// Кнопка удаления таска

DeleteTaskButton: TBitBtn;

// Кнопка редактирования группы

EditGroupButton: TBitBtn;

// Группа редактирования таска

EditTaskButton: TBitBtn;

IconList: TImageList;

Panel3: TPanel;

// Кнопка сохранения репорта

SaveStatusReportButton: TBitBtn;

Panel1: TPanel;

Panel2: TPanel;

ProgressBar1: TProgressBar;

// Кнопка настроек

SettingsButton: TBitBtn;

Splitter1: TSplitter;

Splitter2: TSplitter;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Label4: TLabel;

Label5: TLabel;

Label6: TLabel;

// Лист групп

GroupsList: TListBox;

StaticText1: TStaticText;

StaticText2: TStaticText;

// Лист тасков

TasksList: TListBox;

// Главный лист с статусом выполнения заданий

ListView1: TListView;

TasksPanel: TPanel;

GroupsPanel: TPanel;

// Кнопка обновления репорта

UpdateStatusButton: TBitBtn;

// Нажатие на кнопку добавления группы

procedure AddGroupButtonClick(Sender: TObject);

// Нажатие на кнопку добавления таска

procedure AddTaskListClick(Sender: TObject);

// Нажатие лист групп

procedure GroupsListClick(Sender: TObject);

// Двойное нажатие на лист с статусами заданий

procedure ListView1DblClick(Sender: TObject);

// Нажатие на кнопку сохранения репорта

procedure SaveStatusReportButtonClick(Sender: TObject);

// Нажатие на кнопку настроек

procedure SettingsButtonClick(Sender: TObject);

// Нажатие на лист тасков

procedure TasksListClick(Sender: TObject);

// Нажатие на кнопку обновления статусов заданий

procedure UpdateStatusButtonClick(Sender: TObject);

// Нажатие на кнопку удаления группы

procedure DeleteGroupButtonClick(Sender: TObject);

// Нажатие на кнопку удаления таска

procedure DeleteTaskButtonClick(Sender: TObject);

// Нажатие на кнопку редактирования группы

procedure EditGroupButtonClick(Sender: TObject);

// Нажатие на кнопку редактирования таска

procedure EditTaskButtonClick(Sender: TObject);

// Создание формы

procedure FormCreate(Sender: TObject);

// Функция обновления листа

procedure UpdateList(TableName: String; UpdatingList: TListBox);

//

procedure UpdateCurrentGroup;

//

procedure UpdateCurrentTask;

// Toggle блокировку формы

procedure ChangeFormBlock(AEnabled: Boolean);

private

// Айди выбранной группы

SelectedGroup: Integer;

// Айди выбранного таска

SelectedTask: Integer;

// Айди загруженной группы

LoadedGroup: Integer;

// Айди загруженного таска

LoadedTask: Integer;

public

// Обновляется ли сейчас статус задания

NowUpdating: Boolean;

end;

var

MainForm: TMainForm;

implementation

{$R \*.lfm}

type

{ TReportUpdateThread }

TReportUpdateThread = class(TThread)

private

// Нужна ли остановка обновления

FStop: Boolean;

// Количество веток для проверки

FMax: Integer;

// Уже проверено веток

FProgress: Integer;

// Айди выбранной группы

FSelectedGroup: Integer;

// Айди выбранного таска

FSelectedTask: Integer;

// Лист итемов для добавления в вывод

FListItemToAdd: TListItem;

// Изменить прогресс бар

procedure ChangeProgressBar;

// Обновить прогресс

procedure UpdateProgress;

// Проверка была выполнена успешно

procedure SuccessfulExecute;

protected  
 // Главная функция потока

procedure Execute; override;

// Вызовится в любом случае после завершения работы

procedure AfterExecute(Sender: TObject);

public

constructor Create(CreateSuspended: Boolean; SelectedGroup, SelectedTask: Integer);

// Запуск проверки

procedure Start;

// Остановка проверки

procedure StopUpdating;

end;

var

// Поток проверки заданий

Thread: TReportUpdateThread;

{ TMainForm }

procedure TMainForm.FormCreate(Sender: TObject);

begin

UpdateList('groups', GroupsList);

UpdateList('tasks', TasksList);

SelectedGroup := -1;

SelectedTask := -1;

LoadedGroup := -1;

LoadedTask := -1;

end;

procedure TMainForm.UpdateStatusButtonClick(Sender: TObject);

begin

if not NowUpdating then

begin

if (SelectedGroup = -1) and (SelectedTask = -1) then exit;

Label2.Caption := StaticText2.Caption;

Label4.Caption := StaticText1.Caption;

Thread := TReportUpdateThread.Create(True, SelectedGroup, SelectedTask);

Thread.Start;

end

else

begin

Thread.StopUpdating;

end;

end;

procedure TMainForm.SaveStatusReportButtonClick(Sender: TObject);

var

AppDB: TAppDatabase;

ResultTable: TTable;

GroupName, TaskName: String;

ReportContents: TStringList;

FileName: String;

ListItem: TListItem;

begin

if (LoadedGroup = -1) or (LoadedTask = -1) then

exit;

AppDB := TAppDatabase.Create;

ResultTable := AppDB.ExecOut('SELECT name FROM groups WHERE id=:id', [par('id', LoadedGroup)]);

GroupName := ResultTable[0].KeyData['name'];

ResultTable := AppDB.ExecOut('SELECT name FROM tasks WHERE id=:id', [par('id', LoadedTask)]);

TaskName := ResultTable[0].KeyData['name'];

ResultTable.Free;

AppDB.Free;

ReportContents := TStringList.Create;

FileName := Format('%s\_%s\_%s',

[GroupName, TaskName, FormatDateTime('DD-MM-YYYY\_hh-nn-ss', Now)]);

ReportContents.Add('KKMT TASKS REPORT');

ReportContents.Add(Format('%s', [FileName]));

ReportContents.Add('');

for ListItem in ListView1.Items do

ReportContents.Add(Format('%s , %s , %s',

[ListItem.Caption, ListItem.SubItems[0], ListItem.SubItems[1]]));

ForceDirectories(Application.Location+'/reports');

ReportContents.SaveToFile(Application.Location+'/reports/'+FileName+'.txt');

end;

procedure TMainForm.SettingsButtonClick(Sender: TObject);

var

ModalForm: TSettingsForm;

begin

ModalForm := TSettingsForm.Create(nil);

ModalForm.ShowModal;

ModalForm.Free;

end;

procedure TMainForm.ListView1DblClick(Sender: TObject);

begin

if ListView1.ItemIndex < 0 then

exit;

OpenURL(ListView1.Items[ListView1.ItemIndex].SubItems.Strings[0]);

end;

{ === GroupsPanel === }

procedure TMainForm.GroupsListClick(Sender: TObject);

begin

if GroupsList.ItemIndex < 0 then

exit;

SelectedGroup := TIntObj(GroupsList.Items.Objects[GroupsList.ItemIndex]).I;

StaticText2.Caption := GroupsList.GetSelectedText;

end;

procedure TMainForm.AddGroupButtonClick(Sender: TObject);

var

ModalForm: TGroupForm;

begin

ModalForm := TGroupForm.Create(MainForm);

ModalForm.ShowModal;

ModalForm.Free;

//OpenDialog(TGroupForm);

UpdateList('groups', GroupsList);

end;

procedure TMainForm.DeleteGroupButtonClick(Sender: TObject);

var

AppDB: TAppDatabase;

begin

if GroupsList.ItemIndex >= 0 then

begin

AppDB := TAppDatabase.Create;

AppDB.Exec('DELETE FROM groups WHERE id=:id',

[par('id', TIntObj(GroupsList.Items.Objects[GroupsList.ItemIndex]).I)]);

UpdateCurrentGroup;

UpdateList('groups', GroupsList);

end;

end;

procedure TMainForm.EditGroupButtonClick(Sender: TObject);

var

ModalForm: TGroupForm;

begin

if GroupsList.ItemIndex >= 0 then

begin

ModalForm := TGroupForm.Create(MainForm, TIntObj(GroupsList.Items.Objects[GroupsList.ItemIndex]).I);

ModalForm.ShowModal;

if ModalForm.ModalResult = mrOK then

begin

UpdateCurrentGroup;

UpdateList('groups', GroupsList);

end;

ModalForm.Free;

end;

end;

{ === TasksPanel === }

procedure TMainForm.TasksListClick(Sender: TObject);

begin

if TasksList.ItemIndex < 0 then

exit;

SelectedTask := TIntObj(TasksList.Items.Objects[TasksList.ItemIndex]).I;

StaticText1.Caption := TasksList.GetSelectedText;

end;

procedure TMainForm.AddTaskListClick(Sender: TObject);

var

ModalForm: TTaskForm;

begin

ModalForm := TTaskForm.Create(MainForm);

ModalForm.ShowModal;

ModalForm.Free;

UpdateList('tasks', TasksList);

end;

procedure TMainForm.DeleteTaskButtonClick(Sender: TObject);

var

AppDB: TAppDatabase;

begin

if TasksList.ItemIndex >= 0 then

begin

AppDB := TAppDatabase.Create;

AppDB.Exec('DELETE FROM tasks WHERE id=:id',

[par('id', TIntObj(TasksList.Items.Objects[TasksList.ItemIndex]).I)]);

UpdateCurrentTask;

UpdateList('tasks', TasksList);

end;

end;

procedure TMainForm.EditTaskButtonClick(Sender: TObject);

var

ModalForm: TTaskForm;

begin

if TasksList.ItemIndex >= 0 then

begin

ModalForm := TTaskForm.Create(MainForm, TIntObj(TasksList.Items.Objects[TasksList.ItemIndex]).I);

ModalForm.ShowModal;

if ModalForm.ModalResult = mrOk then

begin

UpdateCurrentTask;

UpdateList('tasks', TasksList);

end;

ModalForm.Free;

end;

end;

procedure TMainForm.UpdateList(TableName: String; UpdatingList: TListBox);

var

AppDB: TAppDatabase;

Table: TTable;

Row: TRow;

begin

AppDB := TAppDatabase.Create;

Table := AppDB.ExecOut('SELECT \* FROM '+TableName+';');

UpdatingList.Clear;

for Row in Table do

begin

UpdatingList.Items.AddObject(Row.KeyData['name'], TIntObj.Create(Row.KeyData['id']));

end;

end;

procedure TMainForm.UpdateCurrentGroup;

var

AppDB: TAppDatabase;

Table: TTable;

begin

if SelectedGroup <> -1 then

begin

AppDB := TAppDatabase.Create;

Table := AppDB.ExecOut('SELECT \* FROM groups WHERE id=:id;', [par('id', SelectedGroup)]);

if Table.Count = 0 then

begin

SelectedGroup := -1;

Label2.Caption := '';

end

else

begin

Label2.Caption := Table.Items[0].KeyData['name'];

end;

end;

if LoadedGroup <> -1 then

begin

Table := AppDB.ExecOut('SELECT \* FROM groups WHERE id=:id;', [par('id', LoadedGroup)]);

if Table.Count = 0 then

begin

LoadedGroup := -1;

ListView1.Clear;

end;

end;

Table.Free;

end;

procedure TMainForm.UpdateCurrentTask;

var

AppDB: TAppDatabase;

Table: TTable;

begin

if SelectedTask <> -1 then

begin

AppDB := TAppDatabase.Create;

Table := AppDB.ExecOut('SELECT \* FROM tasks WHERE id=:id;', [par('id', SelectedTask)]);

if Table.Count = 0 then

begin

SelectedTask := -1;

Label4.Caption := '';

end

else

begin

Label4.Caption := Table.Items[0].KeyData['name'];

end;

end;

if LoadedTask <> -1 then

begin

Table := AppDB.ExecOut('SELECT \* FROM tasks WHERE id=:id;', [par('id', LoadedTask)]);

if Table.Count = 0 then

begin

LoadedTask := -1;

ListView1.Clear;

end;

end;

Table.Free;

end;

procedure TMainForm.ChangeFormBlock(AEnabled: Boolean);

begin

GroupsList.Enabled := AEnabled;

AddGroupButton.Enabled := AEnabled;

DeleteGroupButton.Enabled := AEnabled;

EditGroupButton.Enabled := AEnabled;

TasksList.Enabled := AEnabled;

AddTaskList.Enabled := AEnabled;

DeleteTaskButton.Enabled := AEnabled;

EditTaskButton.Enabled := AEnabled;

SaveStatusReportButton.Enabled := AEnabled;

SettingsButton.Enabled := AEnabled;

end;

{ TReportUpdateThread }

constructor TReportUpdateThread.Create(CreateSuspended: Boolean; SelectedGroup, SelectedTask: Integer);

begin

inherited Create(CreateSuspended);

FreeOnTerminate := True;

FStop := False;

FProgress := 0;

FSelectedGroup := SelectedGroup;

FSelectedTask := SelectedTask;

FListItemToAdd := nil;

OnTerminate := @AfterExecute;

MainForm.ProgressBar1.Style := TProgressBarStyle.pbstMarquee;

MainForm.ProgressBar1.Position := 0;

MainForm.ListView1.Clear;

end;

procedure TReportUpdateThread.Start;

begin

MainForm.NowUpdating := True;

MainForm.UpdateStatusButton.ImageIndex := 1;

MainForm.ChangeFormBlock(False);

inherited Start;

end;

procedure TReportUpdateThread.StopUpdating;

begin

FStop := True;

end;

procedure TReportUpdateThread.AfterExecute(Sender: TObject);

begin

MainForm.ProgressBar1.Style := TProgressBarStyle.pbstNormal;

MainForm.ProgressBar1.Position := 0;

MainForm.NowUpdating := False;

MainForm.UpdateStatusButton.ImageIndex := 3;

MainForm.ChangeFormBlock(True);

end;

procedure TReportUpdateThread.ChangeProgressBar;

// this method is executed by the mainthread and can therefore access all GUI elements.

begin

MainForm.ProgressBar1.Style := TProgressBarStyle.pbstNormal;

MainForm.ProgressBar1.Max := FMax;

end;

procedure TReportUpdateThread.UpdateProgress;

// this method is executed by the mainthread and can therefore access all GUI elements.

begin

MainForm.ProgressBar1.StepIt;

MainForm.ListView1.Items.AddItem(FListItemToAdd);

end;

procedure TReportUpdateThread.SuccessfulExecute;

begin

MainForm.LoadedGroup := FSelectedGroup;

MainForm.LoadedTask := FSelectedTask;

end;

procedure TReportUpdateThread.Execute;

var

HTTP: TFPHttpClient;

AppDB: TAppDatabase;

Table: TTable;

GitHubId: String;

HttpResult: String;

BranchesJSON: TJSONArray;

Enum, Enum2: TJSONEnum;

Files: TStringList;

Counter: Integer;

begin

HTTP := CreateGitHubHTTP;

AppDB := TAppDatabase.Create;

Table := AppDB.ExecOut('SELECT github\_id FROM groups WHERE id=:id;', [par('id', FSelectedGroup)]);

if Table.Count <= 0 then

exit;

GitHubId := Table[0].KeyData['github\_id'];

Table := AppDB.ExecOut('SELECT \* FROM tasks WHERE id=:id;', [par('id', FSelectedTask)]);

if Table.Count <= 0 then

exit;

Files := TStringList.Create;

for Enum in TJSONArray(GetJSON(Table[0].KeyData['files'])) do

Files.Add(Enum.Value.AsString);

try

HttpResult := HTTP.Get(Format('https://api.github.com/repos/%s/branches', [GitHubId] ));

except

MessageDlg('Ошибка получения данных о ветках репозитория. Возможно репозиторий не существует.', mtError,

[mbOK], 0);

exit;

end;

BranchesJSON := TJSONArray(GetJSON(HttpResult));

FMax := BranchesJSON.Count;

Synchronize(@ChangeProgressBar);

for Enum in BranchesJSON do

begin

if FStop then break;

FListItemToAdd := TListItem.Create(MainForm.ListView1.Items);

FListItemToAdd.Caption := TJSONObject(enum.Value)['name'].AsString;

FListItemToAdd.SubItems.Add(Format('https://github.com/%s/tree/%s',

[GitHubId, TJSONObject(enum.Value)['name'].AsString]));

HttpResult := HTTP.Get(Format('https://api.github.com/repos/%s/git/trees/%s?recursive=1',

[GitHubId, TJSONObject(enum.Value).GetPath('commit.sha').AsString]));

Counter := Files.Count;

for Enum2 in TJSONObject(GetJSON(HttpResult)).Arrays['tree'] do

begin

if Files.IndexOf(TJSONObject(Enum2.Value).Strings['path']) <> -1 then

Dec(Counter);

end;

if Counter = 0 then

// something good

FListItemToAdd.SubItems.Add('DA')

else

// somethinf bad

FListItemToAdd.SubItems.Add('NYET');

Synchronize(@UpdateProgress);

end;

BranchesJSON.Free;

AppDB.Free;

HTTP.Free;

Table.Free;

Files.Free;

Synchronize(@SuccessfulExecute);

end;

end.

**Приложение 2. GroupFormUnit.pas**

unit GroupFormUnit;

{$mode objfpc}{$H+}

interface

uses

Classes, SysUtils, Forms, Controls, Graphics, Dialogs, StdCtrls, ExtCtrls,

fphttpclient,

AppDatabase, Utils;

type

{ TGroupForm }

TGroupForm = class(TForm)

// OK кнопка

Button1: TButton;

// Кнопка отмены

Button2: TButton;

// Поле имени группы

LabeledEdit1: TLabeledEdit;

// Поле айди репозитория группы

LabeledEdit2: TLabeledEdit;

// Нажатие на ОК кнопку

procedure Button1Click(Sender: TObject);

// Нажатие на кнопку отмены

procedure Button2Click(Sender: TObject);

private

// Айди группы для режима реадктирования

EditId: Integer;

public

constructor Create(TheOwner: TComponent);

constructor Create(TheOwner: TComponent; GroupForEditing: Integer); overload;

// Изменить дизайн формы для редактирования группы

procedure MakeForEdit(GroupId: Integer);

end;

implementation

{$R \*.lfm}

{ TGroupForm }

procedure TGroupForm.Button1Click(Sender: TObject);

var

HTTP: TFPHttpClient;

HttpResult: String;

AppDB: TAppDatabase;

begin

HTTP := CreateGitHubHTTP;

AppDB := TAppDatabase.Create;

//ShowMessage(Format('https://api.github.com/repos/%s', [LabeledEdit2.Text]));

if String(LabeledEdit1.Text).IsEmpty or String(LabeledEdit1.Text).Contains(' ') then

begin

MessageDlg('Имя группы не должно быть пустым или иметь пробелы', mtError,

[mbOK], 0);

ModalResult := mrNone;

exit;

end;

try

HttpResult := HTTP.Get(Format('https://api.github.com/repos/%s', [LabeledEdit2.Text]));

except

if HTTP.ResponseStatusCode = 404 then

begin

MessageDlg('Репозиторий не найден', mtError,

[mbOK], 0);

ModalResult := mrNone;

exit;

end

else

begin

MessageDlg('Ошибка получения данных о репозитории. Возможно репозиторий не существует.', mtError,

[mbOK], 0);

ModalResult := mrNone;

exit;

end;

end;

try

if EditId = -1 then

AppDB.Exec('INSERT INTO groups (name, github\_id) VALUES (:name, :rep\_id);',

[par('name', LabeledEdit1.Text), par('rep\_id', LabeledEdit2.Text)])

else

AppDB.Exec('UPDATE groups SET name=:name, github\_id=:rep\_id WHERE id=:id;',

[par('name', LabeledEdit1.Text), par('rep\_id', LabeledEdit2.Text), par('id', EditId)] );

except

on E: Exception do

begin

MessageDlg('Ошибка при записи в базу данных.'+LineEnding+E.Message, mtError,

[mbOK], 0);

ModalResult := mrNone;

exit;

end;

end;

end;

procedure TGroupForm.Button2Click(Sender: TObject);

begin

end;

constructor TGroupForm.Create(TheOwner: TComponent);

begin

inherited Create(TheOwner);

EditId := -1;

end;

constructor TGroupForm.Create(TheOwner: TComponent; GroupForEditing: Integer);

begin

inherited Create(TheOwner);

MakeForEdit(GroupForEditing);

end;

procedure TGroupForm.MakeForEdit(GroupId: Integer);

var

AppDB: TAppDatabase;

ResultTable: TTable;

begin

AppDB := TAppDatabase.Create;

ResultTable := AppDB.ExecOut('SELECT \* FROM groups WHERE id=:id;', [par('id', GroupId)]);

Self.LabeledEdit1.Text := ResultTable[0].KeyData['name'];

Self.LabeledEdit2.Text := ResultTable[0].KeyData['github\_id'];

Self.Caption := 'Изменение группы';

Self.EditId := GroupId;

end;

end.

**Приложение 3. TaskFormUnit.pas**

unit TaskFormUnit;

{$mode objfpc}{$H+}

interface

uses

Classes, SysUtils, Forms, Controls, Graphics, Dialogs, StdCtrls, ExtCtrls,

CheckLst, ColorBox, ShellCtrls, ComCtrls, Buttons, AppDatabase, fpjson;

type

{ TTaskForm }

TTaskForm = class(TForm)

// ОК кнопка

Button1: TButton;

// Кнопка отмены

Button2: TButton;

// Кнопка добавления дочерней ноды имени файла

Button3: TBitBtn;

// Кнопка удаления ноды имени файла

Button4: TBitBtn;

Label1: TLabel;

// Имя таска

LabeledEdit1: TLabeledEdit;

Panel1: TPanel;

// Дерево файлов

TreeView1: TTreeView;

// Нажатие на кнопку сохранения

procedure Button1Click(Sender: TObject);

// Нажатие на кнопку отмены

procedure Button2Click(Sender: TObject);

// Нажатие на кнопку добавления ноды

procedure Button3Click(Sender: TObject);

// Нажатие на кнопку удаления ноды

procedure Button4Click(Sender: TObject);

// Выделение ноды

procedure TreeView1MouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

private

// Айди таска для режима редактирования

EditId: Integer;

// Создание массива путей для каждого файла

function TreeToPathsGenerator(Tree: TTreeNodes): TStringList;

// Создание дерева из массива путей

function TreeFromPathsGenerator(Paths: TStringList): TTreeNodes;

public

constructor Create(TheOwner: TComponent);

constructor Create(TheOwner: TComponent; TaskForEditing: Integer); overload;

// Изменить дизайн формы для режима редактирования

procedure MakeForEdit(TaskId: Integer);

end;

implementation

{$R \*.lfm}

{ TTaskForm }

procedure TTaskForm.Button3Click(Sender: TObject);

var

Filename: String;

begin

Filename := InputBox('Введите имя файла',

'Введите имя файла для проверки', '');

if not Filename.IsEmpty then

TreeView1.Items.AddChild(TreeView1.Selected, Filename);

end;

procedure TTaskForm.Button4Click(Sender: TObject);

begin

if TreeView1.Selected <> nil then

TreeView1.Items.Delete(TreeView1.Selected);

end;

procedure TTaskForm.TreeView1MouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

var

i: Integer;

begin

if not Assigned(TreeView1.GetNodeAt(X, Y)) then

for i := 0 to TreeView1.Items.Count - 1 do

TreeView1.Items[i].Selected := false;

end;

procedure TTaskForm.Button1Click(Sender: TObject);

var

AppDB: TAppDatabase;

FilePath: String;

json: TJSONArray;

begin

if Length(LabeledEdit1.Text) <= 0 then

begin

MessageDlg('Введите имя задания', mtError,

[mbOK], 0);

ModalResult := mrNone;

exit;

end;

if TreeView1.Items.Count <= 0 then

begin

MessageDlg('Задайте структуру задания', mtError,

[mbOK], 0);

ModalResult := mrNone;

exit;

end;

json := TJSONArray.Create;

AppDB := TAppDatabase.Create;

for FilePath in TreeToPathsGenerator(TreeView1.Items) do

json.Add(FilePath.Substring(1));

//ShowMessage(json.AsJSON);

try

if EditId = -1 then

AppDB.Exec('INSERT INTO tasks (name, files) VALUES (:name, :files);', [par('name', LabeledEdit1.Text), par('files', json.AsJSON)] )

else

AppDB.Exec('UPDATE tasks SET name=:name, files=:files WHERE id=:id;',

[par('name', LabeledEdit1.Text), par('files', json.AsJSON), par('id', EditId)] );

except

on E: Exception do

begin

MessageDlg('Ошибка при записи в базу данных.'+LineEnding+E.Message, mtError,

[mbOK], 0);

ModalResult := mrNone;

exit;

end;

end;

end;

procedure TTaskForm.Button2Click(Sender: TObject);

begin

end;

function TTaskForm.TreeToPathsGenerator(Tree: TTreeNodes): TStringList;

var Paths: TStringList;

procedure TreeWalker(Node: TTreeNode; PathString: String);

var

i: Integer;

begin

PathString += '/'+Node.Text;

if Node.HasChildren then

for i:=0 to Node.Count-1 do

TreeWalker(Node.Items[i], PathString)

else

Paths.Add(PathString);

end;

var

Node: TTreeNode;

begin

Paths := TStringList.Create;

for Node in Tree do

begin

if Node.Parent = nil then

TreeWalker(Node, '');

end;

Result := Paths;

end;

function TTaskForm.TreeFromPathsGenerator(Paths: TStringList): TTreeNodes;

var

Path: String;

PathParsed: TStringArray;

i, j: Integer;

Node, ChildNode: TTreeNode;

begin

Result := TreeView1.Items;

for Path in Paths do

begin

PathParsed := Path.Split(['/'], TStringSplitOptions.ExcludeEmpty);

Node := Result.FindNodeWithText(PathParsed[0]);

if (Node = nil) or (Node.Parent <> nil) then

Node := Result.AddChild(nil, PathParsed[0]);

for i := 1 to Length(PathParsed)-1 do

begin

ChildNode := Node.FindNode(PathParsed[i]);

if ChildNode = nil then

ChildNode := Result.AddChild(Node, PathParsed[i]);

Node := ChildNode;

end;

end;

end;

constructor TTaskForm.Create(TheOwner: TComponent);

begin

inherited Create(TheOwner);

EditId := -1;

end;

constructor TTaskForm.Create(TheOwner: TComponent; TaskForEditing: Integer);

begin

inherited Create(TheOwner);

MakeForEdit(TaskForEditing);

end;

procedure TTaskForm.MakeForEdit(TaskId: Integer);

var

AppDB: TAppDatabase;

ResultTable: TTable;

Paths: TStringList;

Enum: TJSONEnum;

begin

AppDB := TAppDatabase.Create;

ResultTable := AppDB.ExecOut('SELECT \* FROM tasks WHERE id=:id;', [par('id', TaskId)]);

Self.LabeledEdit1.Text := ResultTable[0].KeyData['name'];

Paths := TStringList.Create;

for Enum in TJSONArray(GetJSON(ResultTable[0].KeyData['files'])) do

Paths.Add(Enum.Value.AsString);

Self.TreeFromPathsGenerator(Paths);

Self.Caption := 'Изменение задания';

Self.EditId := TaskId;

end;

end.

**Приложение 4. AppDatabase.pas**

unit AppDatabase;

interface

uses

fpjson, sqldb, db, SQLite3Conn, sysutils, Generics.Collections, fgl, Variants, SQLite3;

type

// SQL параметр вы виде пары имя-значение

TParamPair = specialize TPair<String, Variant, String>;

// Динамический массив параметров

TParamsArray = Array of TParamPair;

// Строка таблицы

TRow = specialize TFPGMap<String, Variant>;

// Таблица

TTable = specialize TList<TRow>;

// Класс базы данных приложения

TAppDatabase = class (TInterfacedObject)

// Объект соединения с базой данных

Connection: TSQLite3Connection;

// Объект транзакции

Transaction: TSQLTransaction;

// Объект запроса

Query: TSQLQuery;

// Выполнить SQL запрос с параметрами без возврата результата

procedure Exec(AQuery: String; AParams: TParamsArray);

// Выполнить SQL запрос без параметров и без возврата результата

procedure Exec(AQuery: String);

// Выполнить SQL запрос с параметрами и с возвратом результата

function ExecOut(AQuery: String; AParams: TParamsArray): TTable;

// Выполнить SQL запрос без параметров с возвратом результата

function ExecOut(AQuery: String): TTable;

constructor Create;

destructor Free;

end;

// Функция для упрощенного создания объекта TParamPair

function par(AName: String; AValue: Variant): TParamPair;

implementation

// ===== TDatabase =====

procedure TAppDatabase.Exec(AQuery: String; AParams: TParamsArray);

var

Param: TParamPair;

begin

Transaction.StartTransaction;

Query.SQL.Text := AQuery;

if Length(AParams) > 0 then

for Param in AParams do

Query.ParamByName(Param.Key).Value := Param.Value;

Query.Prepare;

try

Query.ExecSQL;

finally

Transaction.Commit;

Query.Clear;

Transaction.EndTransaction;

end;

end;

procedure TAppDatabase.Exec(AQuery: String);

begin

Exec(AQuery, nil);

end;

function TAppDatabase.ExecOut(AQuery: String; AParams: TParamsArray): TTable;

var

Param: TParamPair;

Row: TRow;

Field: TField;

begin

Transaction.StartTransaction;

Query.SQL.Text := AQuery;

if Length(AParams) > 0 then

for Param in AParams do

Query.ParamByName(Param.Key).Value := Param.Value;

Query.Prepare;

try

Query.Open;

Result := TTable.Create;

while not Query.EOF do

begin

Row := TRow.Create;

for Field in Query.Fields do

Row.Add(Field.FieldName, Field.Value);

Result.Add(Row);

Query.Next;

end;

finally

Query.Close;

Transaction.Commit;

Query.Clear;

Transaction.EndTransaction;

end;

end;

function TAppDatabase.ExecOut(AQuery: String): TTable;

begin

Result := ExecOut(AQuery, nil);

end;

constructor TAppDatabase.Create;

begin

//if not FileExists('./application.db') then

//begin

// FileCreate('./application.db');

//end;

Connection := TSQLite3Connection.Create(nil);

Transaction := TSQLTransaction.Create(nil);

Query := TSQLQuery.Create(nil);

Connection.DatabaseName := './application.db';

Connection.Transaction := Transaction;

Query.DataBase := Connection;

Connection.Open;

Exec('CREATE TABLE IF NOT EXISTS "groups" ("name" TEXT NOT NULL UNIQUE,"github\_id" TEXT NOT NULL UNIQUE,"id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE)');

Exec('CREATE TABLE IF NOT EXISTS "tasks" ("id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,"name" TEXT NOT NULL,"files" TEXT NOT NULL)');

end;

destructor TAppDatabase.Free;

begin

Transaction.EndTransaction;

Connection.Connected := False;

Connection.Free;

Transaction.Free;

Query.Free;

end;

function par(AName: String; AValue: Variant): TParamPair;

begin

Result.Key := AName;

Result.Value := AValue;

end;

end.

**Приложение 5. SettingFormUnit.pas**

unit SettingsFormUnit;

{$mode objfpc}{$H+}

interface

uses

Classes, SysUtils, Forms, Controls, Graphics, Dialogs, ExtCtrls, StdCtrls;

type

{ TSettingsForm }

TSettingsForm = class(TForm)

// Кнопка сохранения

Button1: TButton;

// Кнопка отмены

Button2: TButton;

// Поле с именем пользователя GitHub

LabeledEdit1: TLabeledEdit;

// Поле с токеном пользователя GitHub

LabeledEdit2: TLabeledEdit;

// Нажатие на кнопку сохранения

procedure Button1Click(Sender: TObject);

// Нажатие на кнопку отмены

procedure Button2Click(Sender: TObject);

// Создания формы

procedure FormCreate(Sender: TObject);

private

// Сохранение настроек в конфиг

procedure LoadConfig;

// Загрузка настроек из конфига

procedure SaveConfig;

public

end;

implementation

{$R \*.lfm}

uses

jsonConf;

{ TSettingsForm }

procedure TSettingsForm.Button2Click(Sender: TObject);

begin

Self.Close;

end;

procedure TSettingsForm.FormCreate(Sender: TObject);

begin

LoadConfig;

end;

procedure TSettingsForm.Button1Click(Sender: TObject);

begin

SaveConfig;

Self.Close;

end;

procedure TSettingsForm.LoadConfig;

var

c: TJSONConfig;

begin

c := TJSONConfig.Create(nil);

c.Filename := Application.Location+'/config.json';

LabeledEdit1.Text := c.GetValue('/github/login', '');

LabeledEdit2.Text := c.GetValue('/github/token', '');

c.Free;

end;

procedure TSettingsForm.SaveConfig;

var

c: TJSONConfig;

begin

c := TJSONConfig.Create(nil);

c.Filename := Application.Location+'/config.json';

c.SetValue('/github/login', LabeledEdit1.Text);

c.SetValue('/github/token', LabeledEdit2.Text);

c.Free;

end;

end.

**Приложение 6. Utils.pas**

unit Utils;

{$mode objfpc}{$H+}

interface

uses

Classes, SysUtils, Dialogs, fpjson, fphttpclient, Forms;

type

// Класс инкапсуляции объекта Integer

TIntObj = class

private

FI: Integer;

public

property I: Integer Read FI;

constructor Create(IValue: Integer);

end;

// Создание и подготовка HttpClient для работы с GitHub API

function CreateGitHubHTTP: TFPHttpClient;

// Функция быстрого открытия модального окна

procedure OpenDialog(FormClass: TFormClass);

implementation

uses

opensslsockets, jsonConf;

{ TIntObj }

constructor TIntObj.Create(IValue: Integer);

begin

Inherited Create;

FI := IValue;

end;

function CreateGitHubHTTP: TFPHttpClient;

var

c: TJSONConfig;

begin

c := TJSONConfig.Create(nil);

Result := TFPHTTPClient.Create(nil);

c.Filename := Application.Location+'/config.json';

Result.AllowRedirect := True;

Result.AddHeader('User-Agent', 'Mozilla/5.0 (compatible; fpweb)');

Result.AddHeader('Accept', 'application/vnd.github.v3+json');

Result.UserName := c.GetValue('/github/login', '');

Result.Password := c.GetValue('/github/token', '');

end;

procedure OpenDialog(FormClass: TFormClass);

var

ModalForm: TForm;

begin

ModalForm := FormClass.Create(nil);

ModalForm.ShowModal;

ModalForm.Free;

end;

end.