Вот пошаговый план реализации модульного приложения на Delphi с указанием ключевых моментов:

### 1. Архитектура приложения

- \*\*MainApp.exe\*\* - основная программа с GUI

- \*\*API модуль\*\* (shared.pas) - общий для всех модулей

- \*\*DLL модули\*\* (поставляются отдельно)

- \*\*Менеджер задач\*\* - для асинхронного выполнения

### 2. Общий API модуль (shared.pas)

```pascal

type

TTaskInfo = record

ID: string;

Name: string;

Description: string;

Params: array of record

Name: string;

ParamType: string; // 'string', 'integer', 'boolean' и т.д.

IsRequired: Boolean;

end;

end;

TTaskExecuteCallback = procedure(const TaskID: string; const Params: array of Variant;

Callback: TProc<string, Integer, string>) of object;

// API функции, которые должны быть в DLL

TGetTasksFunc = function: TArray<TTaskInfo>;

TExecuteTaskFunc = procedure(const TaskID: string; const Params: array of Variant;

Callback: TProc<string, Integer, string>);

```

### 3. Реализация MainApp.exe

\*\*Основная форма:\*\*

```pascal

type

TMainForm = class(TForm)

// Компоненты интерфейса

TaskList: TListView;

RunningTasksList: TListView;

ExecutedTasksList: TListView;

ParamsPanel: TPanel;

ExecuteBtn: TButton;

StopBtn: TButton;

procedure FormCreate(Sender: TObject);

procedure ExecuteBtnClick(Sender: TObject);

procedure StopBtnClick(Sender: TObject);

private

FDLLs: TList<HMODULE>;

FTasks: TDictionary<string, TTaskInfo>;

FActiveTasks: TDictionary<string, TThread>;

procedure LoadDLLs;

procedure ShowTaskParams(const TaskID: string);

procedure TaskCompleted(const TaskID: string; ExitCode: Integer; const Result: string);

end;

```

\*\*Динамическая загрузка DLL:\*\*

```pascal

procedure TMainForm.LoadDLLs;

var

DLLName: string;

hDLL: HMODULE;

GetTasks: TGetTasksFunc;

Task: TTaskInfo;

begin

for DLLName in TDirectory.GetFiles(ExtractFilePath(ParamStr(0)), '\*.dll') do

begin

hDLL := LoadLibrary(PChar(DLLName));

if hDLL <> 0 then

begin

@GetTasks := GetProcAddress(hDLL, 'GetTasks');

if Assigned(GetTasks) then

begin

for Task in GetTasks() do

FTasks.Add(Task.ID, Task);

end;

FDLLs.Add(hDLL);

end;

end;

end;

```

\*\*Асинхронный запуск задач:\*\*

```pascal

procedure TMainForm.ExecuteBtnClick(Sender: TObject);

var

TaskID: string;

hDLL: HMODULE;

ExecuteTask: TExecuteTaskFunc;

Params: array of Variant;

TaskThread: TThread;

begin

TaskID := TaskList.Selected.Caption;

// Получаем параметры из интерфейса

SetLength(Params, 2);

Params[0] := MaskEdit.Text;

Params[1] := DirectoryEdit.Text;

// Создаем поток для выполнения задачи

TaskThread := TThread.CreateAnonymousThread(

procedure

var

hDLL: HMODULE;

ExecuteTask: TExecuteTaskFunc;

begin

hDLL := FDLLs[0]; // Пример - берем первую DLL

@ExecuteTask := GetProcAddress(hDLL, 'ExecuteTask');

if Assigned(ExecuteTask) then

ExecuteTask(TaskID, Params,

procedure(const TaskID: string; ExitCode: Integer; const Result: string)

begin

TThread.Synchronize(nil,

procedure

begin

TaskCompleted(TaskID, ExitCode, Result);

end);

end);

end);

FActiveTasks.Add(TaskID, TaskThread);

TaskThread.Start;

end;

```

### 4. Пример реализации DLL (FileOperations.dll)

\*\*Реализация функций DLL:\*\*

```pascal

library FileOperations;

uses

SysUtils, Classes, shared;

function GetTasks: TArray<TTaskInfo>; stdcall;

begin

SetLength(Result, 2);

// Задача 1: Поиск файлов

Result[0].ID := 'FileSearch';

Result[0].Name := 'Поиск файлов';

Result[0].Description := 'Поиск файлов по маске в указанной папке';

SetLength(Result[0].Params, 2);

Result[0].Params[0].Name := 'Маска';

Result[0].Params[0].ParamType := 'string';

Result[0].Params[0].IsRequired := True;

Result[0].Params[1].Name := 'Папка';

Result[0].Params[1].ParamType := 'string';

Result[0].Params[1].IsRequired := True;

// Задача 2: Поиск в файле

Result[1].ID := 'FileContentSearch';

Result[1].Name := 'Поиск в файле';

Result[1].Description := 'Поиск последовательности байт в файле';

// ... параметры аналогично

end;

procedure ExecuteTask(const TaskID: string; const Params: array of Variant;

Callback: TProc<string, Integer, string>); stdcall;

begin

if TaskID = 'FileSearch' then

ExecuteFileSearch(Params, Callback)

else if TaskID = 'FileContentSearch' then

ExecuteContentSearch(Params, Callback);

end;

procedure ExecuteFileSearch(const Params: array of Variant;

Callback: TProc<string, Integer, string>);

var

Mask, Folder: string;

Files: TStringList;

begin

Mask := Params[0];

Folder := Params[1];

Files := TStringList.Create;

try

// Реализация поиска файлов

FindFiles(Folder, Mask, Files);

// Возвращаем результат

Callback('FileSearch', 0, Format('Найдено файлов: %d', [Files.Count]));

finally

Files.Free;

end;

end;

exports

GetTasks,

ExecuteTask;

begin

end.

```

### 5. Дополнительные рекомендации

1. \*\*Для отображения прогресса\*\*:

- Добавьте в Callback промежуточные статусы

- Используйте TProgressBar в интерфейсе

2. \*\*Для прерывания задач\*\*:

- Добавьте флаг Cancelled в поток

- Реализуйте StopBtn.OnClick для установки флага

3. \*\*Для логов\*\*:

- Создайте TStringList для каждой задачи

- Сохраняйте туда все сообщения Callback

- Добавьте кнопку "Показать логи" для просмотра

4. \*\*Для параметров задач\*\*:

- Динамически создавайте элементы управления на ParamsPanel

- Используйте TEdit для строк, TCheckBox для boolean и т.д.

5. \*\*Для Shell-команд\*\*:

- Во второй DLL используйте CreateProcess с перенаправлением вывода

- Читайте вывод процесса в отдельном потоке

Это базовая архитектура, которую можно расширять по мере необходимости. Главные преимущества такого подхода - модульность, динамическая загрузка функционала и асинхронное выполнение задач без блокировки интерфейса.

Вот полный код для обеих DLL с реализацией всех указанных функций:

### 1. FileOperations.dll (поиск файлов и содержимого)

```pascal

library FileOperations;

uses

Windows, SysUtils, Classes, Masks, StrUtils, Variants, shared;

type

TSearchThread = class(TThread)

private

FTaskID: string;

FParams: array of Variant;

FCallback: TProc<string, Integer, string>;

procedure DoCallback;

protected

procedure Execute; override;

public

constructor Create(const TaskID: string; const Params: array of Variant;

Callback: TProc<string, Integer, string>);

end;

constructor TSearchThread.Create(const TaskID: string; const Params: array of Variant;

Callback: TProc<string, Integer, string>);

begin

inherited Create(True);

FTaskID := TaskID;

FParams := Params;

FCallback := Callback;

FreeOnTerminate := True;

end;

procedure TSearchThread.DoCallback;

begin

if Assigned(FCallback) then

FCallback(FTaskID, 0, '');

end;

procedure TSearchThread.Execute;

var

Files: TStringList;

SearchResult: string;

i: Integer;

begin

Files := TStringList.Create;

try

if FTaskID = 'FileSearch' then

begin

// Поиск файлов по маске

FindAllFiles(Files, FParams[1], FParams[0], True);

SearchResult := Format('Найдено файлов: %d', [Files.Count]);

// Дополнительно: список файлов

if Length(FParams) > 2 then

if FParams[2] then

SearchResult := SearchResult + sLineBreak + Files.Text;

end

else if FTaskID = 'FileContentSearch' then

begin

// Поиск содержимого в файле

SearchInFile(FParams[1], FParams[0], Files);

SearchResult := Format('Найдено вхождений: %d', [Files.Count]);

// Дополнительно: позиции вхождений

if Length(FParams) > 2 then

if FParams[2] then

SearchResult := SearchResult + sLineBreak + Files.Text;

end;

Synchronize(DoCallback);

FCallback(FTaskID, 0, SearchResult);

finally

Files.Free;

end;

end;

procedure FindAllFiles(var Files: TStringList; const Path, Mask: string; Recursive: Boolean);

var

SR: TSearchRec;

SubMasks: TStringList;

i: Integer;

begin

SubMasks := TStringList.Create;

try

SubMasks.Delimiter := ';';

SubMasks.StrictDelimiter := True;

SubMasks.DelimitedText := Mask;

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

begin

if FindFirst(IncludeTrailingPathDelimiter(Path) + '\*.\*', faAnyFile, SR) = 0 then

begin

repeat

if (SR.Name = '.') or (SR.Name = '..') then Continue;

if (SR.Attr and faDirectory) = faDirectory then

begin

if Recursive then

FindAllFiles(Files, IncludeTrailingPathDelimiter(Path) + SR.Name, SubMasks[i], True);

end

else if MatchesMask(SR.Name, SubMasks[i]) then

Files.Add(IncludeTrailingPathDelimiter(Path) + SR.Name);

until FindNext(SR) <> 0;

FindClose(SR);

end;

end;

finally

SubMasks.Free;

end;

end;

procedure SearchInFile(const FileName, Pattern: string; var Positions: TStringList);

var

Stream: TFileStream;

Buffer: array of Byte;

PatternBytes: TBytes;

i, j: Integer;

Found: Boolean;

begin

Positions.Clear;

PatternBytes := TEncoding.ANSI.GetBytes(Pattern);

Stream := TFileStream.Create(FileName, fmOpenRead or fmShareDenyNone);

try

SetLength(Buffer, Stream.Size);

Stream.ReadBuffer(Buffer[0], Stream.Size);

for i := 0 to Length(Buffer) - Length(PatternBytes) do

begin

Found := True;

for j := 0 to Length(PatternBytes) - 1 do

begin

if Buffer[i + j] <> PatternBytes[j] then

begin

Found := False;

Break;

end;

end;

if Found then

Positions.Add(Format('Вхождение на позиции: %d', [i]));

end;

finally

Stream.Free;

end;

end;

function GetTasks: TArray<TTaskInfo>; stdcall;

begin

SetLength(Result, 2);

// Задача 1: Поиск файлов

Result[0].ID := 'FileSearch';

Result[0].Name := 'Поиск файлов';

Result[0].Description := 'Поиск файлов по маске в указанной папке';

SetLength(Result[0].Params, 3);

Result[0].Params[0].Name := 'Маска';

Result[0].Params[0].ParamType := 'string';

Result[0].Params[0].IsRequired := True;

Result[0].Params[1].Name := 'Папка';

Result[0].Params[1].ParamType := 'string';

Result[0].Params[1].IsRequired := True;

Result[0].Params[2].Name := 'Показывать список файлов';

Result[0].Params[2].ParamType := 'boolean';

Result[0].Params[2].IsRequired := False;

// Задача 2: Поиск в файле

Result[1].ID := 'FileContentSearch';

Result[1].Name := 'Поиск в файле';

Result[1].Description := 'Поиск последовательности байт в файле';

SetLength(Result[1].Params, 3);

Result[1].Params[0].Name := 'Искомая строка';

Result[1].Params[0].ParamType := 'string';

Result[1].Params[0].IsRequired := True;

Result[1].Params[1].Name := 'Файл';

Result[1].Params[1].ParamType := 'string';

Result[1].Params[1].IsRequired := True;

Result[1].Params[2].Name := 'Показывать позиции';

Result[1].Params[2].ParamType := 'boolean';

Result[1].Params[2].IsRequired := False;

end;

procedure ExecuteTask(const TaskID: string; const Params: array of Variant;

Callback: TProc<string, Integer, string>); stdcall;

begin

// Запускаем в отдельном потоке

with TSearchThread.Create(TaskID, Params, Callback) do

Start;

end;

exports

GetTasks,

ExecuteTask;

begin

end.

```

### 2. ShellCommands.dll (выполнение shell-команд)

```pascal

library ShellCommands;

uses

Windows, SysUtils, Classes, JclShell, Process, Variants, shared;

type

TShellThread = class(TThread)

private

FTaskID: string;

FParams: array of Variant;

FCallback: TProc<string, Integer, string>;

FOutput: TStringList;

procedure DoCallback;

procedure DoProgress;

protected

procedure Execute; override;

public

constructor Create(const TaskID: string; const Params: array of Variant;

Callback: TProc<string, Integer, string>);

destructor Destroy; override;

end;

constructor TShellThread.Create(const TaskID: string; const Params: array of Variant;

Callback: TProc<string, Integer, string>);

begin

inherited Create(True);

FTaskID := TaskID;

FParams := Params;

FCallback := Callback;

FOutput := TStringList.Create;

FreeOnTerminate := True;

end;

destructor TShellThread.Destroy;

begin

FOutput.Free;

inherited;

end;

procedure TShellThread.DoCallback;

begin

if Assigned(FCallback) then

FCallback(FTaskID, 0, FOutput.Text);

end;

procedure TShellThread.DoProgress;

begin

if Assigned(FCallback) then

FCallback(FTaskID, 1, 'Выполнение...');

end;

procedure TShellThread.Execute;

var

CmdLine: string;

ExitCode: Integer;

begin

if FTaskID = 'ShellCommand' then

begin

CmdLine := FParams[0];

// Простое выполнение команды с захватом вывода

ExitCode := ExecuteAndCapture(CmdLine, FOutput);

Synchronize(DoCallback);

FCallback(FTaskID, ExitCode, FOutput.Text);

end

else if FTaskID = 'LongOperation' then

begin

// Длительная операция с промежуточными статусами

for var i := 1 to 10 do

begin

if Terminated then Break;

Sleep(1000);

FOutput.Add(Format('Шаг %d выполнен', [i]));

Synchronize(DoProgress);

end;

if not Terminated then

begin

FOutput.Add('Операция завершена');

Synchronize(DoCallback);

FCallback(FTaskID, 0, FOutput.Text);

end;

end;

end;

function GetTasks: TArray<TTaskInfo>; stdcall;

begin

SetLength(Result, 2);

// Задача 1: Простая команда

Result[0].ID := 'ShellCommand';

Result[0].Name := 'Shell команда';

Result[0].Description := 'Выполнение shell команды';

SetLength(Result[0].Params, 1);

Result[0].Params[0].Name := 'Команда';

Result[0].Params[0].ParamType := 'string';

Result[0].Params[0].IsRequired := True;

// Задача 2: Длительная операция

Result[1].ID := 'LongOperation';

Result[1].Name := 'Длительная операция';

Result[1].Description := 'Имитация длительной операции';

SetLength(Result[1].Params, 0);

end;

procedure ExecuteTask(const TaskID: string; const Params: array of Variant;

Callback: TProc<string, Integer, string>); stdcall;

begin

with TShellThread.Create(TaskID, Params, Callback) do

Start;

end;

exports

GetTasks,

ExecuteTask;

begin

end.

```

### Ключевые особенности реализации:

1. \*\*Асинхронное выполнение\*\*:

- Каждая задача выполняется в отдельном потоке (TThread)

- Используется Synchronize для обновления UI

2. \*\*Гибкие параметры\*\*:

- Поддержка разных типов параметров (string, boolean)

- Возможность добавления опциональных параметров

3. \*\*Обратная связь\*\*:

- Callback для возврата результатов

- Промежуточные статусы выполнения

4. \*\*Безопасность\*\*:

- Проверка параметров

- Обработка ошибок

5. \*\*Дополнительные функции\*\*:

- Поиск по нескольким маскам

- Поиск нескольких подстрок

- Захват вывода командной строки

Для использования этих DLL необходимо:

1. Скомпилировать их как отдельные проекты Delphi

2. Поместить в папку с основным приложением

3. Основное приложение будет автоматически их загружать и отображать доступные задачи