МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГТУ»)

#### Факультет информационных технологий и компьютерной безопасности

(факультет)

### Кафедра Систем автоматизированного проектирования и информационных систем

# КУРСОВОЙ ПРОЕКТ

по дисциплине Программирование на платформе .NET Framework

тема Разработка программного обеспечения на платформе .NET Framework

**Расчетно-пояснительная записка**

Разработал студент Д.В. Тюленев

Подпись, дата Инициалы, фамилия

## Руководитель Д.В. Иванов

Подпись, дата Инициалы, фамилия

## Члены комиссии

Подпись, дата Инициалы, фамилия

## 

Подпись, дата Инициалы, фамилия

## Нормоконтролер Д.В. Иванов

Подпись, дата Инициалы, фамилия

Защищена \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата

2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГТУ»)

Кафедра Систем автоматизированного проектирования и информационных систем

ЗАДАНИЕ

на курсовой проект

по дисциплине Программирование на платформе .NET Framework

Тема работы Разработка программного обеспечения на платформе .NET Framework

Студент группы ИСТ-214 Тюленев Данил Вячеславович

Фамилия, имя, отчество

Вариант 3. Определение пути с минимальным числом дуг на основе поиска в ширину.

Технические условия процессор Intel® Core*™* i3-8145U CPU @ 2,10 ГГц,

операционная система Windows 10, ОЗУ 8192 МБ

Содержание и объем проекта (графические работы, расчеты и прочее):

анализ требований к программному обеспечению (10 страниц); разработка математического обеспечения (10 страниц); разработка архитектуры ПО (24 страницы); 29 рисунков, 1 таблица, 1 приложение.

Сроки выполнения этапов анализ требований к программному обеспечению (13.09 – 02.10); разработка математического обеспечения (04.10 – 30.10); разработка архитектуры ПО (01.11 – 13.11); реализация программы (15.11 – 11.12); оформление пояснительной записки (13.12 – 18.12)

Срок защиты курсового проекта 22.12.2022

Руководитель Д.В. Иванов

Подпись, дата Инициалы, фамилия

Задание принял студент 15.09.2022 Д.В. Тюленев

Подпись, дата Инициалы, фамилия

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc24395)

[1 Анализ требований к программному обеспечению 5](#_Toc19707)

[1.1 Общая характеристика языков высокого уровня 5](#_Toc32418)

[1.2 Анализ платформ и средств реализации 8](#_Toc18229)

[1.3 Преимущества платформы .NET Framework 10](#_Toc11218)

[2 Разработка математического обеспечения 15](#_Toc32299)

[2.1 Описание задачи определения пути с минимальным числом дуг на основе поиска в ширину 15](#_Toc5374)

[2.2 Математическая модель определения пути с минимальным числом дуг на основе поиска в ширину 17](#_Toc15263)

[2.3 Алгоритм математического функционирования 21](#_Toc2147)

[3 Разработка архитектуры ПО 25](#_Toc6733)

[3.1 Модульная структура приложения 25](#_Toc3423)

[3.2 Разработка алгоритмов и методов 30](#_Toc17771)

[3.3 Описание диалога с пользователем 37](#_Toc1647)

[3.4 Минимальные системные требования 46](#_Toc28954)

[3.5 Контрольный пример 46](#_Toc11366)

[ЗАКЛЮЧЕНИЕ 49](#_Toc15177)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 50](#_Toc29749)

[ПРИЛОЖЕНИЕ А 51](#_Toc24805)

ВВЕДЕНИЕ

ЭВМ помогает человеку реализовать трудоёмкие работы в различных сферах его деятельности. Универсальность различных научных приложений заключается в многократности их использования с различными входными данными для решения задачи в заданной области. Графы встречаются в сотнях разных задач, и алгоритмы обработки графов очень важны. Существует множество разработанных алгоритмов для решения различных задач из самых разных областей человеческой деятельности. Формулировка задачи описывает, каким требованиям должно удовлетворять решение задачи, а алгоритм, решающий эту задачу, находит объект, этим требованиям удовлетворяющий. В этой работе будет проанализирован алгоритм поиска в ширину в графе. Поиском по заданному аргументу называется алгоритм, определяющий соответствие ключа с заданным аргументом. Алгоритм поиска в ширину может быть использован для просмотра созданного графа, чтобы узнать состав информационных вершин для последующего поиска.

В результате работы алгоритма поиска заданная вершина может быть найдена или может быть отмечено отсутствие её в исходных данных. Если заданная информационная вершина найдена, то происходит вывод об успешном окончании поиска, вывод времени поиска и времени поиска ключа. Данный курсовой проект посвящён разработке программного решения для определения пути с минимальным числом дуг на основе поиска в ширину.

Исходными данными для проектирования оптимального решения будут являться теоретический материал анализа проектных решений, раздела дискретной математики, изучающего графы. В курсовом проекте будет проанализирована поставленная задача на основе теоретического материала, а также программного и аппаратного обеспечения. В результате исследования будет предоставлено приложение для расчёта кратчайшего пути на основе составленного пользователем графа.

1 Анализ требований к программному обеспечению

1.1 Общая характеристика языков высокого уровня

Высокоуровневые языки программирования отличаются большей направленностью на человека, чем на машину, для которой их конструкции будут слишком сложными и ресурсоемкими. Высокоуровневый язык программирования (ЯП)－средство записи компьютерных программ, обеспечивающее высокую скорость и удобство работы. Высокоуровневый язык программирования обеспечивает возможность введения смысловых конструкций, способных коротко описать форматы данных и операции с ними в тех случаях, когда описания на низкоуровневом языке (например, на машинном коде) будут сложными для восприятия и очень длинными. Языки программирования рассматриваемого уровня создавались с целью предотвращения зависимости сути алгоритмов от платформы. В этом случае платформенная независимость обеспечивается перекладыванием связей на инструментальные программы, которые осуществляют перевод текстов с высокоуровневых языков на машинный код. Инструментальные программы выступают своего рода трансляторами.

Высокоуровневые ЯП облегчают выполнение сложных задач программирования и упрощают адаптацию ПО. Использование разнообразных трансляторов и интерпретаторов обеспечивает связь программ, написанных при помощи языков высокого уровня, с различными операционными системами программируемыми устройствами и оборудованием, и, в идеале, не требует модификации исходного кода (текста, написанного на высокоуровневом языке) для любой платформы.

Основным достоинством машинно-независимых языков программирования являются их простота и универсальность. Как следствие, значительно сокращается продолжительность написания кода и отладки. В языках высокого уровня особенности конкретной компьютерной архитектуры не учитываются, поэтому написанная на них программа легко может быть перенесена на другую машину. Очень часто бывает достаточно скомпилировать программу под ту ли иную архитектурную и операционную систему.

Также к преимуществам языков программирования высокого уровня следует отнести такие факты:

* алфавит существенно шире машинного.(содержит 256 символов и позволяет описать любые конструкции);
* для операторов и ключевых слов используются осмысленные слова естественного языка;
* развитые операторы управления и огромный набор средств описания структур данных;
* существует понятие типа данных и поддерживается их широкий набор.

Недостатком высокоуровневых языков программирования в сравнении с низкоуровневыми является большой размер программ. Поэтому существуют сферы, где до сих пор используются ассемблеры. Это разработка компиляторов для языков высокого уровня, драйверы, системный код, микроконтроллеры.

Основной областью применения языков высокого уровня является написание ПО для компьютеров и устройств с большими объёмами памяти. Программы, написанные на языках программирования высокого уровня, более объёмные. Таким образом мы расплачиваемся за удобство и скорость написания кода. Чем более человеко-ориентирован язык, тем больше памяти он занимает. Здесь можно отметить, что программы, написанные на низкоуровневых ЯП, не нуждаются в таких больших объёмах памяти.

Языки программирования высокого уровня подразделяются на следующие категории: процедурно-ориентированные, проблемно-ориентированные, объектно-ориентированные языки.

Процедурно-ориентированные ЯП применяются для записи процедур или алгоритмов обработки информации на каждом определённом круге задач, выполнение кода организовано сверху вниз по иерархическому принципу. Языки данной категории предоставляют программисту возможность определять каждый шаг в процессе решения задачи. Особенность состоит в том, что задачи разбиваются на шаги и решаются шаг за шагом. Используя процедурный язык, программист определяет языковые конструкции для выполнения последовательности алгоритмических шагов. Важным шагом в развитии процедурного программирования стал переход к структурной парадигме. Процедурно-ориентированные языки: Pascal, Basic, C и т.д.

Проблемно-ориентированные языки используются для решения целых классов новых задач, возникших в связи с постоянным расширением области применения вычислительной техники, характеризуются наиболее высоким уровнем абстракции, а алфавитом в таких языках служат термины конкретной научно-технической области. К проблемно-ориентированным языкам относят: LISP, Prolog, SQL и др.

Концепция объектно-ориентированные языки заключается в представлении программы, как совокупности объектов, важно понимание взаимодействия друг с другом объектов, а не логика работы каждого объекта. Объекты создаются на основании класса, определённого в коде и объединяют данные и действия, которые можно выполнять с данными, в одно целое. Использование классов лучше структурирует программу и значительно сокращает её размер. Объектно-ориентированные языки основаны на четырёх принципах. Эти четыре принципа являются общими свойствами, которые определяют их и делают их значительно более эффективными.: инкапсуляция (объединение и сокрытие данных и операции), наследование (использование структуру и функциональность уже определённого класса), полиморфизм (использование объектов с одинаковым интерфейсом в схожей манере). Примерами объектно-ориентированных ЯП являются: С#, C++, Java, Python, Ruby и т.д.

1.2 Анализ платформ и средств реализации

Средства разработки программного обеспечения－совокупность приёмов, методов, методик, а также набор инструментальных программ (компиляторы, прикладные/системные библиотеки и т.д.), используемых разработчиком для создания программного кода программы, отвечающего заданным требованиям.

В зависимости от предметной области и задач, поставленных перед разработчиками, разработка программ может представлять собой достаточно сложный, поэтапный процесс, в котором задействовано большое количество участников и разнообразных средств. Для того, чтобы определить, когда и в каких случаях какие средства применяются, выделяют следующие основные этапы разработки программного обеспечения: проектирование приложения, реализация программного кода, тестирование.

На этапе реализации программного кода выполняется кодирование отдельных компонент программы в соответствии с разработанным техническим проектом. Средства, которые могут быть применены, в значительной степени зависит от того, какие подходы были использованы во

время проектирования и, кроме этого, от степени проработанности технического проекта. Тем не менее, среди средств разработки программного кода необходимо выделить следующие основные виды средств: методы и методики алгоритмирования, языки программирования (C++, Си, Java, C#, PHP и многие другие), средства создания пользовательского интерфейса (MFC, WPF, QT, GTK+ и т.д.), средства управления версиями программного кода (CVS, SVN, VSS, GIT), средства получения исполняемого кода (Visual Studio, GCC и многие другие), средства управления базами данных (Microsoft SQL, FireBird, MySQL и многие другие), отладчики (MS Visual Studio, GDB и т.д.).

Основными задачами тестирования является проверка соответствия функциональности разработанной программы первоначальным требованиям, а также выявление ошибок, которые в явном или неявном виде проявляются во время работы программы. Среди основных работ по тестированию можно выделить следующее: тестирование на отказ и восстановление, функциональное тестирование, тестирование безопасности, тестирование взаимодействия, тестирование процесса установки, тестирование удобства пользования.

Среди основных видов средств, которые могут быть применены для выполнения поставленных работ можно привести следующие:

* средства анализа кода, профилирования (Code Wizard–ParaSoft, Purify–Rational Softawre. Test Coverage – Semantic и т.д.);
* средства для тестирования функциональности (TEST－Parasoft, QACenter ;
* Compuware, Borland SilkTestи т.д.);
* средства для тестирования производительности (QACenter Performance－Compuware и т.д).

Платформы разработки составляют основу инструментальных средств создания программных продуктов. Инструментальные средства разработки программного обеспечения – это программные инструменты, предназначенные для обеспечения полного цикла проектирования программного продукта. К ним относятся IDE (Integrated Development Environment) – это интегрированная, единая среда разработки, которая используется разработчиками для создания различного программного обеспечения. IDE представляет собой комплекс из нескольких инструментов, а именно: текстового редактора, компилятора либо интерпретатора, средств автоматизации сборки и отладчика.

Среда разработки позволяет выполнять следующие функции: писать, просматривать, запускать и отлаживать код внутри одного окна, редактировать код и частично автоматизировать его рефакторинг (улучшение), компилировать код, собирать проект, тестировать написанное (некоторые IDE позволяют эмулировать разные устройства), быстро искать, устанавливать и подключать новые компоненты, управлять проектами, создавать их и редактировать, работать с системами контроля версий, создавать визуальный интерфейс (если в IDE входит визуальный редактор, как в Visual Studio, выполнять другие задачи в зависимости от назначения среды и от языка).

Все это делает разработку удобнее и быстрее, поэтому IDE пользуются большинство программистов. Правда, для отдельных технологий или задач предпочтительнее редакторы кода － они проще и легче.

Среды разработки бывают бесплатными и платными, проприетарными и с открытым кодом, то есть с открытым исходным кодом. Сейчас их существуют десятки － для разных языков программирования, технологий и задач. Широко известными IDE являются: Microsoft Visual Studio, Eclipse, Borland C++Builder и многие другие.

1.3 Преимущества платформы .NET Framework

Платформа .NET Framework－это технология, которая поддерживает создание и выполнение веб-служб и приложений Windows. Первый выпуск .NET Framework состоялся ещё в далёком 2002 году. Тогда платформа была разработана как альтернатива Java. Основное отличие заключалось в том, что платформа официально рассчитывалась на работу с операционными системами семейства Microsoft Windows.

Платформа .NET Framework состоит из общеязыковой среды выполнения (среды CLR) и библиотеки классов .NET Framework. Основой платформы .NET Framework является среда CLR. Среду выполнения можно считать агентом, который управляет кодом во время выполнения и предоставляет основные службы, такие как управление памятью, управление потоками и удалённое взаимодействие. При этом средой накладываются условия строгой типизации и другие виды проверки точности кода, обеспечивающие безопасность и надёжность. Фактически основной задачей среды выполнения является управление кодом. Код, который обращается к среде выполнения, называют управляемым кодом, а код, который не обращается к среде выполнения, называют неуправляемым кодом. Библиотека классов является комплексной объектно-ориентированной коллекцией повторно используемых типов, которые применяются для разработки приложений－приложения командной строки, приложения с графическим интерфейсом (GUI) и приложения ASP.NET (веб-приложений).

Платформа .NET Framework может размещаться неуправляемыми компонентами, которые загружают среду CLR в собственные процессы и запускают выполнение управляемого кода, создавая таким образом программную среду, позволяющую использовать средства как управляемого, так и неуправляемого выполнения. Платформа .NET Framework не только предоставляет несколько базовых сред выполнения, но также поддерживает разработку базовых сред выполнения независимыми производителями.

Платформа .NET позволяет не устанавливать для каждой программы отдельную среду. Таким образом, уменьшается не только время на установку, но и жёсткий диск не забьётся лишними копиями немного отличающихся библиотек. Если говорить о программистах, то .NET Framework позволяет заметно сократить время и ресурсы при разработке, так как с его помощью можно развить одну среду и использовать её для нескольких языков. Разработчики, которые знают разные языки, могут писать общий программный продукт под конкретную .NET платформу. Элементы этого продукта, написанные на разных языках, смогут коммуницировать между собой без каких-либо проблем.

Можно выделить следующие основные черты платформы .NET Framework:

* Поддержка нескольких языков－при компиляции код на любом из этих языков компилируется в сборку на общем языке CIL (Common Intermediate Language), благодаря чему .NET поддерживает несколько языков: наряду с C# это также VB.NET, C++, F#, а также различные диалекты других языков, привязанные к .NET (при определённых условиях мы можем сделать отдельные модули одного приложения на отдельных языках);
* Кроссплатформенность－.NET является переносимой платформой (с некоторыми ограничениями), используя различные технологии на платформе .NET, можно разрабатывать приложения на языке C# для самых разных платформ: Windows, MacOS, Linux, Android, iOS;
* Мощная библиотека классов－.NET представляет единую для всех поддерживаемых языков библиотеку классов;
* Общий исполняющий механизм, разделяемый всеми поддерживающими .NET языками, одним из аспектов такого механизма является наличие чётко определённого набора типов, которые способен опознавать каждый поддерживающий .NET язык;
* Упрощённая модель развёртывания－в отличие от СОМ библиотеки .NET не регистрируются в системном реестре. Более того, платформа .NET позволяет нескольким версиям одной и той же сборки DLL благополучно сосуществовать на одном компьютере.
* Автоматическая сборка мусора－в большинстве случаев не придётся, в отличие от С++, заботиться об освобождении памяти, так как общеязыковая среда CLR сама вызовет сборщик мусора и очистит память.

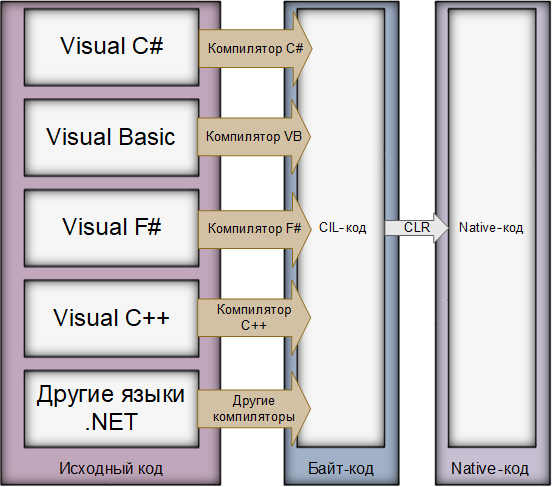


Рисунок 1－Схема работы CLR

Главный механизм CLR физически представлен библиотекой по имени «mscoree.dll». Когда на сборку производится ссылка для её применения, библиотека загружается автоматически и в свою очередь загружает требуемую сборку в память. Исполняющая среда отвечает за решение нескольких задач: определением местоположения сборки и нахождением запрошенного типа в двоичном модуле за счёт чтения содержащихся в нем метаданных, размещает тип в памяти, компилирует ассоциированный код CIL в специфичные для платформы инструкции, производит все необходимые проверки безопасности и, наконец, выполняет нужный код.

Библиотека классов платформы .NET Framework представляет собой коллекцию типов, которые тесно интегрируются со средой CLR. Библиотека классов является объектно-ориентированной. Она предоставляет типы, от которых управляемый код пользователя может наследовать функции. Это не только упрощает работу с типами .NET Framework, но и сокращает время изучения новых средств платформы .NET Framework. Кроме того, компоненты независимых производителей можно легко объединять с классами платформы .NET Framework.

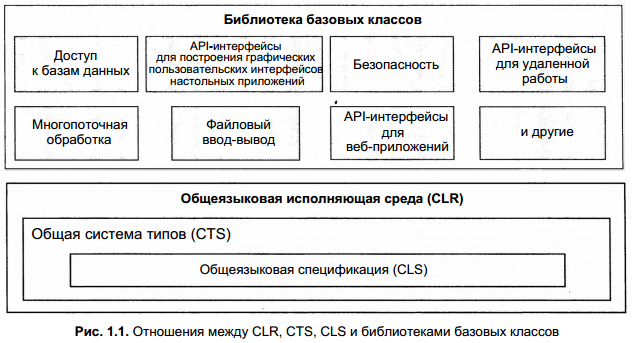


Рисунок 2－Структура платформы .NET Framework

Двоичные модули, называемые сборкой .NET, содержат независимые от платформы инструкции на промежуточном языке (Intermediate Language－CIL) и метаданные типов. Код CIL не компилируется в специфичные для платформы инструкции до тех пор, пока это не станет абсолютно необходимым. Обычно «абсолютная необходимость» наступает тогда, когда на блок инструкций CIL производится ссылка с целью его применения исполняющей средой .NET.

Метаданные детально описывают характеристики каждого типа внутри двоичного модуля, автоматически генерируются компилятором языка .NET. Помимо инструкций CIL и метаданных типов сами сборки также описываются с помощью манифеста, который содержит информацию о текущей версии сборки, сведения о культуре (используемые для локализации строковых и графических ресурсов) и список ссылок на все внешние сборки, которые требуются для надлежащего функционирования.

2 Разработка математического обеспечения

2.1 Описание задачи определения пути с минимальным числом дуг на основе поиска в ширину

Теория графов－важнейший математический инструмент, широко используемый в разных сферах человеческой жизни, так как посредством графов можно описывать разнообразные реальные явления: организацию транспортных систем, сети передачи данных, человеческих взаимоотношений, структуру гена или молекулы. Возможность формального моделирования такого множества разных реальных структур позволяет программисту решать широкий круг прикладных задач. Теория графов стала активно применяться в программировании одновременно с использованием ЭВМ в силу удобного выражения задач обработки информации.

Применение теории о нахождении кратчайшего пути на графах широко распространено и находит новые области развития рассматриваемому алгоритму «поиска в ширину». Использование этого метода особенно актуально в таких сферах, как экономика, логистика, транспорт, туристический бизнес и др. Задача поиска кратчайшего пути встречается при расчётах маршрута между двумя объектами, при управлении телефонным трафиком и компьютерными сетями, а также при разработке городских транспортных маршрутов. Учёными разработано множество способов поиска кратчайшего путь на графе. Например, в алгоритмах перебора вариантов решение разлагается постепенно. При разветвлённой сети маршрутов построить граф и найти кратчайший путь достаточно трудно, именно поэтому применяются параллельное программирование и нейронные сети, которые помогают решать непростые динамические задачи.

«Поиск в ширину» является задачей над графами, которая входит в категорию «обход графа» и решается, когда нужно над вершинами графа выполнить какие-либо операции. Графом в математике называется конечная совокупность точек, называемых вершинами; некоторые из них соединены друг с другом линиями, называемыми рёбрами графа. Дугой или ориентированным ребром называется ребро, имеющее направление.

Графы бывают: неориентированные－в этих графах важно только присутствие связи между вершинами; ориентированные－в этих графах важны присутствие и направление связей между вершинами; взвешенные－в этих графах важны присутствие, направление и вес связей между вершинами. Поиск или обход в ширину графа могут осуществляться над всеми типами графов.

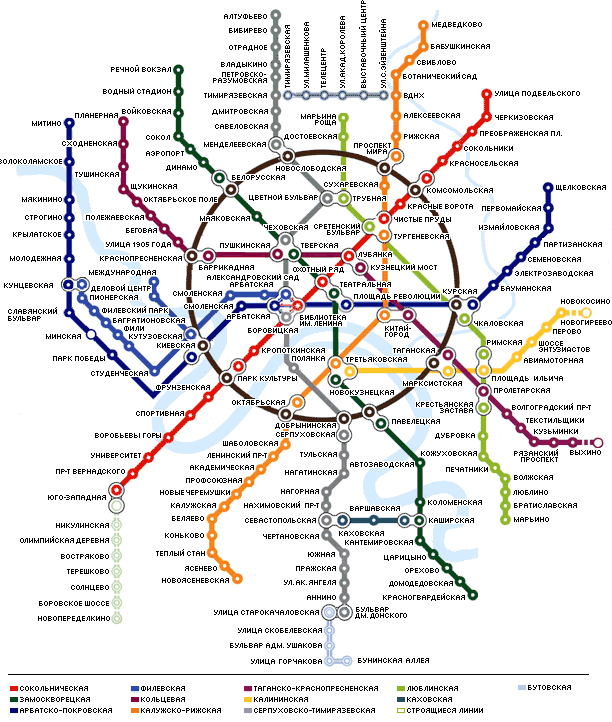


Рисунок 3－Пример задачи о поиске кратчайшего пути

Поиск кратчайшего пути между узлами является самой известной графовой задачей. Системы навигации GPS проводят поиск в графе улиц и перекрёстков для вычисления маршрута. Некоторые из них даже используют данные дорожного движения с целью увеличения веса рёбер, представляющих улицы, где образовался затор. В реальной жизни, графы встречаются на каждом шагу, а мы ведь об этом даже не задумываемся. Например, люди, живущие в больших городах, пользуются метро для следования до места работы или учёбы и каждый день они смотрят на схему метро, а ведь она является тем же самым графом. Станции на схеме－это вершины графа, линии между станциями－это ребра графа－обозначают пути.

2.2 Математическая модель определения пути с минимальным числом дуг на основе поиска в ширину

Предположим, что мы хотим найти кратчайший путь между двумя конкретными вершинами некоторого графа－путь, соединяющий вершины, обладающие тем свойством, что никакой другой путь, соединяющий эти вершины, не содержит меньшее число рёбер. Поиск в ширину (англ. Breadth-First Search, BFS)－один из основных алгоритмов на графах, служит основой многочисленных алгоритмов обработки графов, позволяющий находить все кратчайшие пути от заданной вершины и решать многие другие задачи. Поиск в ширину также называют обходом－так же, как поиск в глубину и все другие обходы, он посещает все вершины графа по одному разу, только в другом порядке: по увеличению расстояния до начальной вершины. Поиск в глубину мало пригоден для решения этой задачи, поскольку предлагаемый им порядок прохождения графа не имеет отношения к поиску кратчайших путей. В отличие от поиска в глубину, поиск в ширину предназначен как раз для достижения этой цели.

Обход－это простой переход от вершины к вершине, для того чтобы разыскать свойства связей между этими вершинами. Задачи для обхода графа в ширину или глубину могут быть следующие: поиск кратчайшего пути между какими-то вершинами; найти в графе «дерево» с наименьшим суммарным весом рёбер; раскрасить вершины графа разными цветами, чтобы у смежных вершин не было одинакового цвета; поиск пути, проходящего кратчайшим путём по всем вершинам графа.

Деревом называется граф, который является связным (между каждой парой вершин есть путь) и не имеет циклов－путь, в первая и последняя вершины совпадают и ребра в течение пути не повторяются. Лесом называют граф, связные компоненты которого являются деревьями. В частности, дерево не может иметь петель и кратных рёбер. Вершину графа, инцидентную (подключённую) только одному его ребру, называют концевой (или висячей) вершиной, а ребро, инцидентное концевой вершине, будем называть концевым ребром графа. Остовным деревом связного графа называется его остовный подграф, являющийся деревом.

На вход алгоритма подаётся невзвешенный граф G = (V, E) и номер выделенной начальной вершины S, где где V－непустое конечное множество вершин, E－конечное множество рёбер, причём каждому ребру e ∈ E сопоставлена неупорядоченная пара вершин, то есть e = (v, w), где v, w ∈ V.

Путём в графе G = (V, E) из вершины v0 в вершину vm (также обозначается (v0, vm)) называется последовательность вершин и рёбер графа G (v0, e1, v1, e2, v2, ... , vm−1, em, vm), в которой ej = (vj−1, vj) ∈ E для каждого j = [1, m], (m >0 ). Для графов без петель и кратных рёбер путь однозначно определяется последовательностью вершин v0, v1, v2, ... , vm−1, vm.

Поиск кратчайшего пути от вершины S к вершине W мы начнём с того, что среди всех вершин, в которые можно перейти по одному ребру из вершины S, мы попытаемся обнаружить вершину W, затем мы проверяем все вершины, в которые мы можем перейти по двум рёбрам, и т.д. Алгоритм поиска в ширину систематически обходит все ребра графа G для «открытия» всех вершин, достижимых из S. В процессе обхода строится дерево поиска в ширину с корнем в начальной вершине, содержащее все достижимые вершины. Заметим, что расстояние (количество рёбер) от корневой вершины до любой вершины этого дерева является кратчайшим. Поиск в ширину имеет такое название, так как в процессе обхода графа помечаются все вершины на расстоянии k, прежде чем начнётся обработка любой вершины на расстоянии k + 1. Данный принцип изображён на рисунке 4. Алгоритм работает как для ориентированных, так и для неориентированных графов.

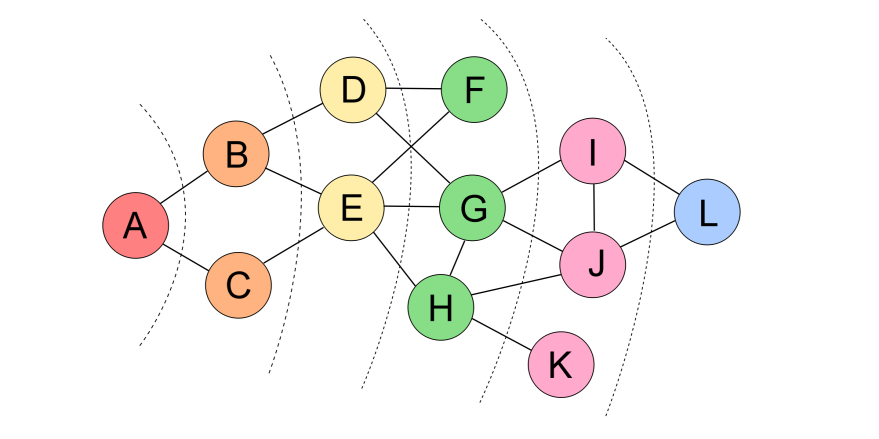


Рисунок 4－Разделение вершин графа на уровни для обхода в ширину

Поиск в ширину работает путём последовательного просмотра отдельных уровней графа, начиная с узла-источника S. Рассмотрим все рёбра (u, v), выходящие из узла u. Если очередной узел v является целевым узлом, то поиск завершается; в противном случае узел v добавляется в список (очередь). После того, как будут проверены все рёбра, выходящие из узла u, из очереди извлекается следующий узел u, и процесс повторяется.

Детальное описание этапов алгоритма обработки в ширину графа G: поместить узел, с которого начинается поиск, в изначально пустую очередь → извлечь из начала очереди узел u и пометить его как развёрнутый → если узел u является целевым узлом, то завершить поиск с результатом «успех», в противном случае, в конец очереди добавляются все преемники узла u, которые ещё не развёрнуты и не находятся в очереди → если очередь пуста, то все узлы связного графа были просмотрены, следовательно, целевой узел недостижим из начального и завершить поиск с результатом «неудача» → вернуться к второму этапу. Примечание: деление вершин на развёрнутые и не развёрнутые необходимо для произвольного графа (так как в нём могут быть циклы). Для дерева эта операция не нужна, так как каждая вершина будет выбрана один раз.

Существует два способа представления графа, в виде списков смежности и в виде матрицы смежности. Оба способа подходят для представления ориентированных и неориентированных графов.

Список смежности－это набор неупорядоченных списков, используемых для представления конечного графа. Каждый неупорядоченный список в списке смежности описывает набор соседей конкретной вершины в графе. Главный недостаток этого способа представления в том, что нет быстрого способа проверить существует ли ребро (e, v). Графический вид списка смежности представлен на рисунке 5.

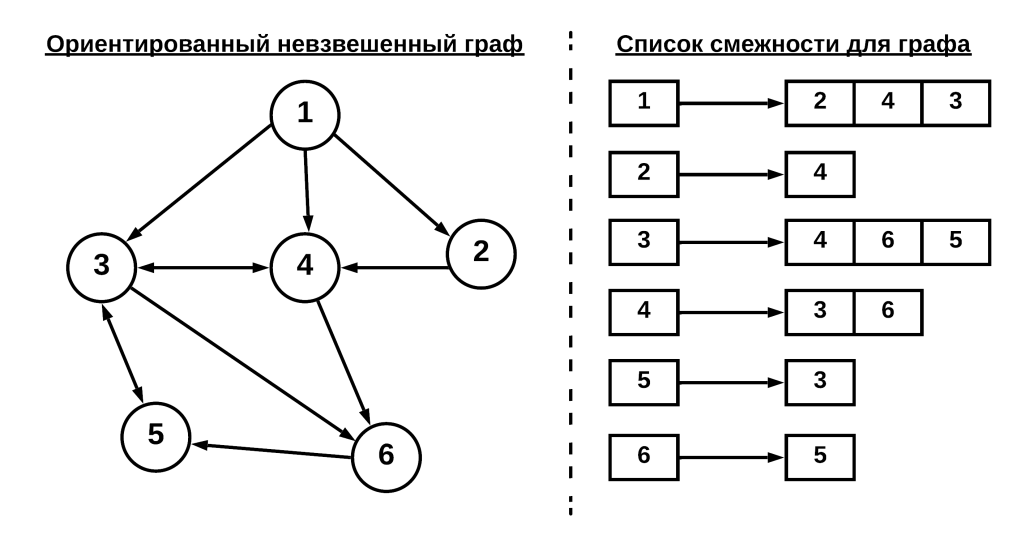


Рисунок 5－Пример работы списка смежности для графа

Матрицей смежности (англ. Adjacency matrix) A=|аi,j| невзвешенного графа G=(V, E) называется матрица A[V×V], в которой аi,j－количество рёбер, соединяющих вершины vi и vj, причём при i=j каждую петлю учитываем дважды, если граф не является ориентированным, и один раз, если граф ориентирован. Имея в наличии лишь матрицу такого типа, несложно построить соответствующий ей граф. Для графического отображения графа необходимо уметь вычислять по матрице смежности количество его вершин, а также обладать знанием следующего правила. Когда из одной вершины в другую проход свободен (имеется ребро), тогда в ячейку заноситься 1, иначе – 0. Графический вид матрицы смежности представлен на рисунке 6.

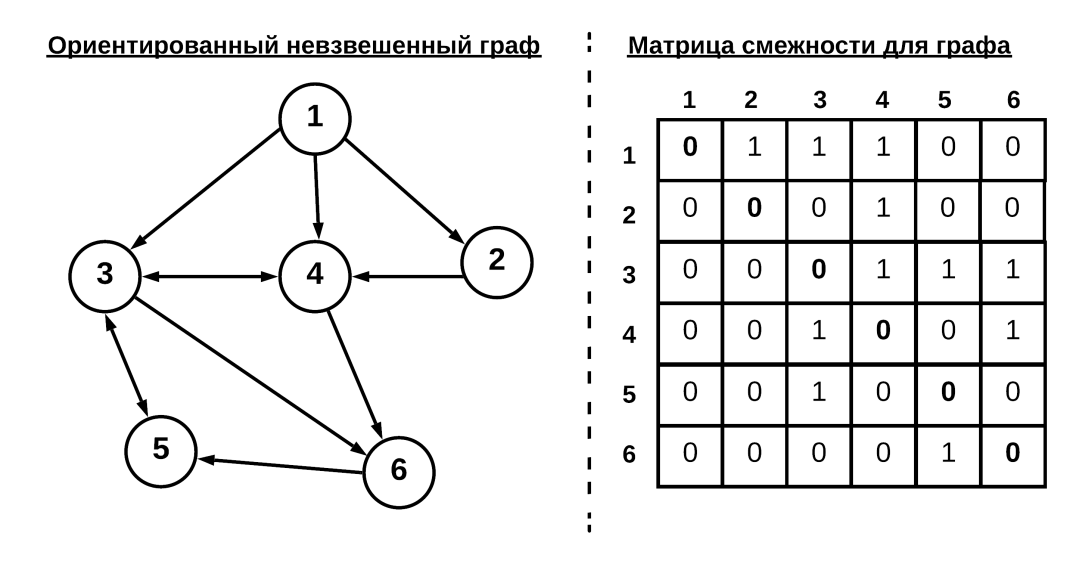


Рисунок 6－Пример работы матрицы смежности для графа

Матрица смежности занимает O(|V|2) памяти. За O(1) можно определить вес ребра или его наличие между любыми двумя вершинами. Для графов без петель и кратных рёбер матрица смежности бинарна (состоит из нулей и единиц).

2.3 Алгоритм математического функционирования

Алгоритм простой и интуитивно понятный. Он проходит по вершинам графа, пока в том не останется непосещённых вершин, и рассчитывает самый короткий путь до целевой вершины. Чтобы показать его работу нагляднее, представим алгоритм пошагово:

* Начало работы. В качестве начальной можно выбрать любую вершину. На момент начала работы алгоритма все вершины помечены как непосещённые－их называют «белыми». Первое, что делает алгоритм,－помечает начальную вершину как посещённую (также используют термины «развёрнутая» или «серая»). Если она и есть целевая, на этом алгоритм завершается. Но чаще всего это не так.
* Поиск соседей. Алгоритм проверяет, какие соседи есть у начальной вершины. Они добавляются в «очередь действий» в том порядке, в каком алгоритм их нашёл, и тоже помечаются как «серые». Это продолжается, пока у начальной вершины не останется «белых» соседей.
* Переход на следующую вершину. Когда алгоритм проходит по всем соседям начальной вершины, он помечает её как полностью обойдённую. Такие вершины ещё называют «чёрными»: алгоритм к ним не возвращается. Затем он переходит к одной из «серых» вершин－соседей начальной. Алгоритм выбирает первую вершину в очереди. Далее действия повторяются: «соседи» вершины, кроме «чёрной», заносятся в очередь. Когда и эта вершина будет пройдена, переход повторится по тому же принципу－первая вершина в очереди. В этом случае ею будет второй сосед начальной вершины－мы помним, что их добавляли в очередь первыми. И только когда соседи начальной вершины в очереди закончатся, алгоритм пойдёт по следующему «уровню» вершин. Так и достигается обход в ширину.
* Конец алгоритма. Если очередь оказалась пустой, это значит, что «белых» и «серых» вершин больше не осталось. Алгоритм завершится. Если при этом целевая вершина не будет достигнута, это значит, что доступа к ней из начальной точки нет. Если целевая вершина достигается раньше, чем алгоритм пройдёт по всему графу, это также может означать его завершение. Алгоритм остановится, потому что задача окажется выполнена: самый короткий путь к целевой вершине будет найден.

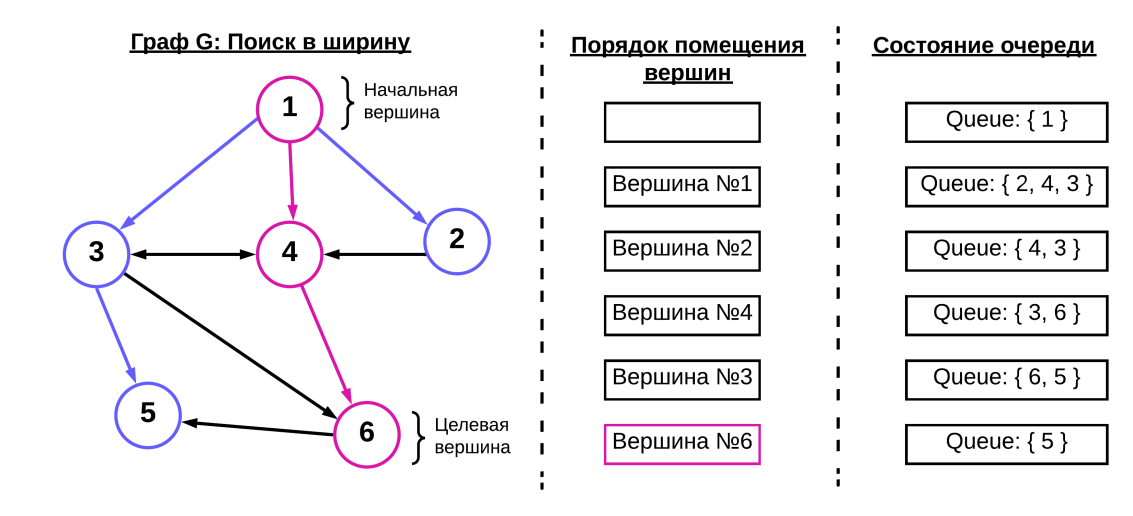


Рисунок 7－Поиск вершины графа G с помощью обхода в ширину

Реализация алгоритма возможна на любом доступном языке программирования. Граф обычно представляется как массив, очередь или другая структура данных. Её элементы－вершины, и в них хранятся сведения о других вершинах, с которыми они соединены. Иногда там напрямую есть ссылки на другие вершины－конкретная реализация зависит от языка программирования и выбранной архитектуры. Алгоритм BFS, реализованный программно, начинает с заданного элемента этой структуры данных－это аналогично начальной вершине. Теперь перейдём к более формальному описанию алгоритма поиска в ширину. Основными объектами－тремя структурами данных, задействованными в программе, будут: список смежности графа G, очередь (Queue), массив посещённых вершин (Visited).

Две первые структуры имеют целочисленный тип данных, последняя－логический. Посещённые вершины, заносятся в массив Visited, что предотвратит зацикливание, а очередь Queue будет хранить задействованные узлы. Напомним, что такое «очередь»－(англ. queue) это структура данных, добавление и удаление элементов в которой происходит путём операций «Push» и «Pop» соответственно, притом первым из очереди удаляется элемент, который был помещён туда первым, то есть в очереди реализуется принцип «первым вошёл, первым вышел» (англ. First-In, First-Out－FIFO); у неё имеется голова (англ. head) и хвост (англ. tail), когда элемент ставится в очередь, он занимает место в её хвосте, а из очереди всегда выводится элемент, который находится в её голове. Рассмотрим разбитый на этапы процесс обхода графа:

1. массив Visited обнуляется, т. е. ни одна вершина графа ещё не была посещена;
2. в качестве стартовой, выбирается некоторая вершина S и помещается в очередь (в массив Queue);
3. вершина S исследуется (помечается как посещённая), и все смежные с ней вершины помещаются в конец очереди, а сама она удаляется;
4. если на данном этапе очередь оказывается пустой, то алгоритм прекращает свою работу;
5. иначе посещается вершина, стоящая в начале очереди, помечается как посещённая, и все её потомки заносятся в конец очереди;
6. пункт 3 выполняется пока это возможно.

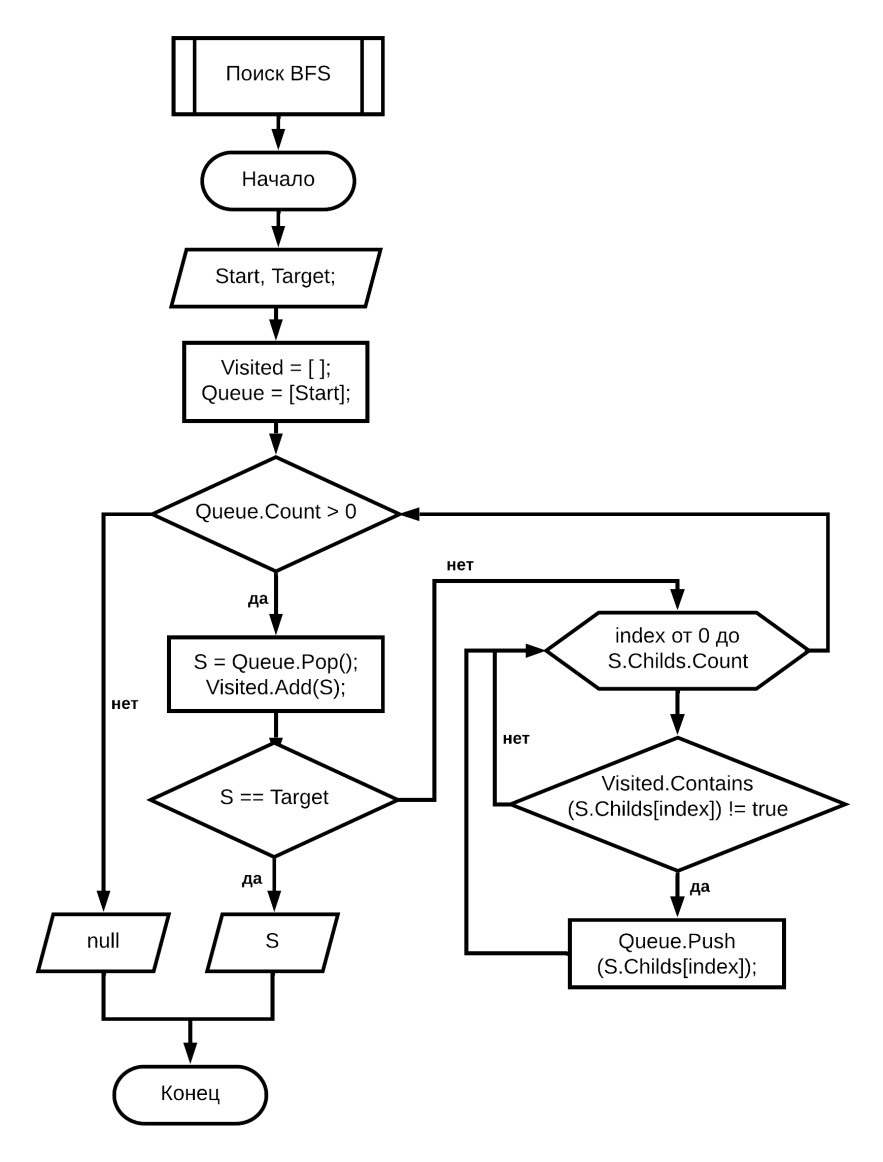


Рисунок 8－Блок-схема алгоритма поиска в ширину

Поиск в ширину, начиная со стартовой вершины, постепенно уходит от неё все дальше и дальше, проходя уровень за уровнем. Получается, что по окончанию работы алгоритма будут найдены все кратчайшие пути из начальной вершины до каждого доступного из неё узла.

3 Разработка архитектуры ПО

3.1 Модульная структура приложения

При разработке решения используется архитектура SOA (сервис-ориентированная архитектура), которая представляет собой способ проектирования распределенной системы, где несколько автономных служб работают совместно, передавая сообщения через границы (либо сетевых машин, либо двух процессов на одной машине) с использованием чётко определённых интерфейсов.. Чтобы было возможно работать с сервисами на платформе .NET Framework, используется программный фреймворк WCF (Windows Communication Foundation).

Проектируемое приложение состоит из трёх основных компонентов: клиентская сборка－отвечает за отображение пользовательской информации, осуществление ввода (составление графов); сборка службы WCF－определяет основные типы бизнес-логики (математические расчёты, управление пользовательскими профилями и проектами); и сборка хоста, которая настраивает и открывает доступ к службам (консольное приложение, Windows служба). Для начала рассмотрим модульную структуру клиентской стороны решения (рисунок 9).

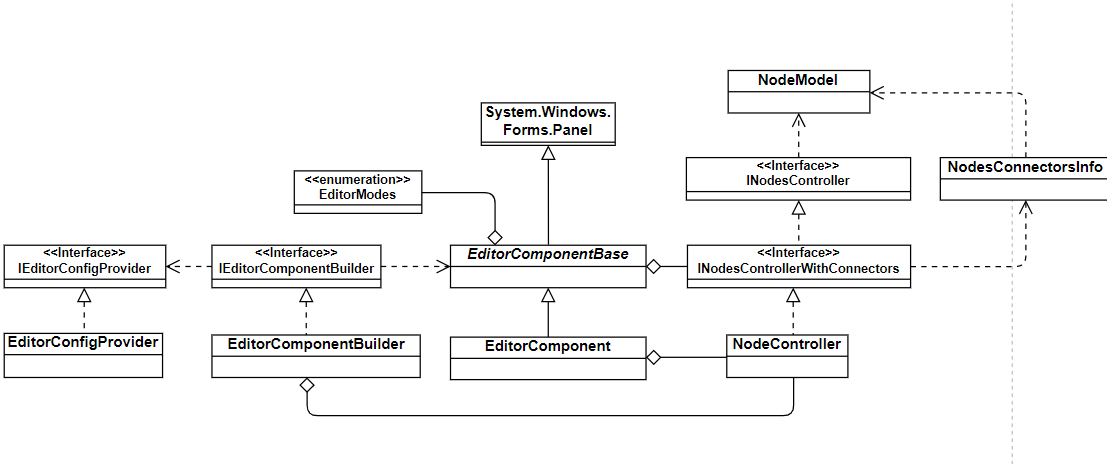


Рисунок 9－UML диаграмма классов клиентской части

Одним из главных типов сборки является абстрактный класс «EditorComponentBase». Он отвечает за определение базовых свойств и операций, доступных всем дочерним объектам-редакторам (отрисовка объектов и обработка взаимодействия с ними), включает набор режимов и событий, которые уведомляют пользователя класса о взаимодействии с рабочим полем или конкретным узлом (вершиной). Также необходимо уточнить, что данный класс наследуется от класса «System.Windows.Forms.Panel», который используется, чтобы сгруппировать коллекции элементов управления, а также включает возможность взаимодействовать с объектом «System.Drawing.Graphics», который предоставляется доступ к интерфейсу рисования.

Помимо вышесказанного, базовый класс редакторов инкапсулирует доступ к объектам интерфейса «INodeControllerWithConnectors», который описывает поведение управления соединениями вершин и созданием ребёр (экземпляров класса «NodesConnectorsInfo», который содержит информацию о связи двух вершин). Данный интерфейс расширяет функциональность интерфейса «INodeController», который описывает способ взаимодействия с базовыми операциями управления узлами: удаление, добавление, проверка столкновений, предоставления списка.

В качестве модели, которая характеризует отдельно взятую вершину, используется класс «NodeModel», который содержит основные характеристики узла: положение в плоскости, его персональный идентификатор и ссылки на подключённые с ним вершины.

Класс реализует интерфейс «INodeControllerWithConnectors» тем самым, создаёт тип, экземляр которого возможно использовать внутри дочерних классов «EditorComponentBase». Данный тип будет предоставлять доступ к управлению загруженными вершинами.

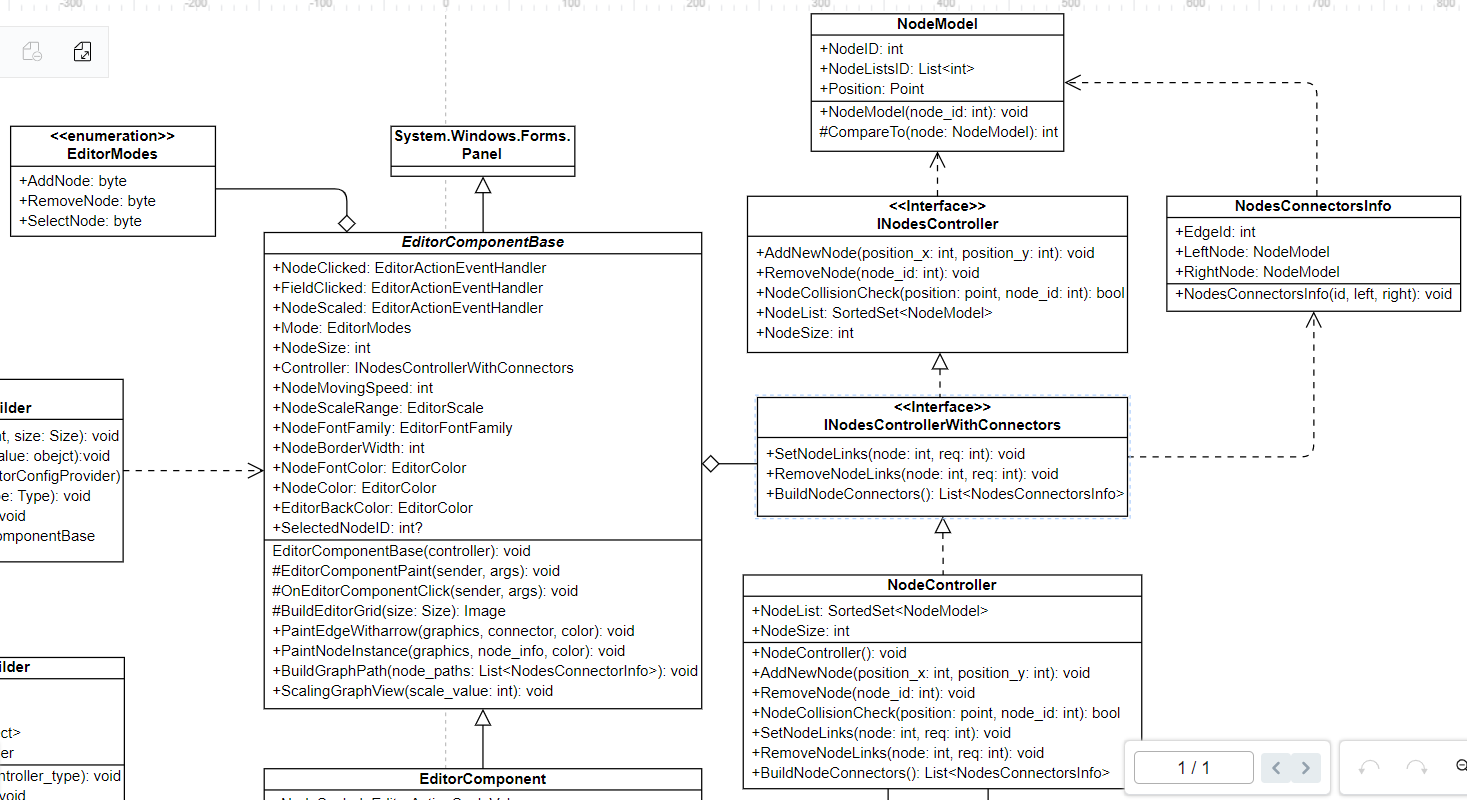


Рисунок 10－Подробное описание структуры и связей классов клиента (часть 1)

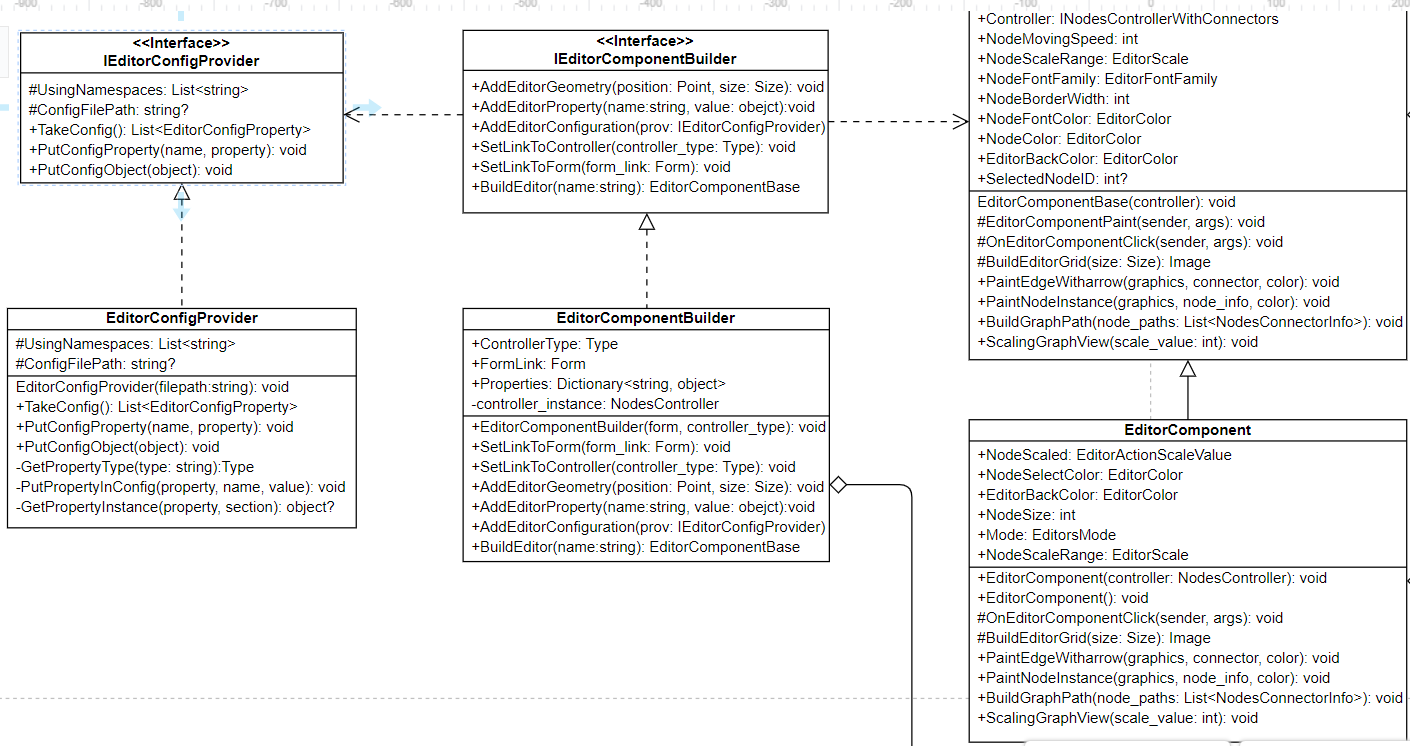


Рисунок 11－Подробное описание структуры и связей классов клиента (часть 2)

Интерфейс «EditorComponentBase» определяет набор абстрактным членов, которые используются для настройки конфигурации и создания экземпляра «EditorComponentBase». Класс «EditorComponentBuilder», реализующий данный интерфейс, используется для создания экземляра конкретного класса «EditorComponent». Один из методов интерфейса принимает объект типа «IEditorConfigProvider», который предоставляет способ взаимодействия с конфигурационным файлом клиентской сборки: чтения и сохранения изменений.

Далее перейдём к рассмотрению части типов сборки службы, которые продемонстрированы в виде диаграммы на рисунке 12.

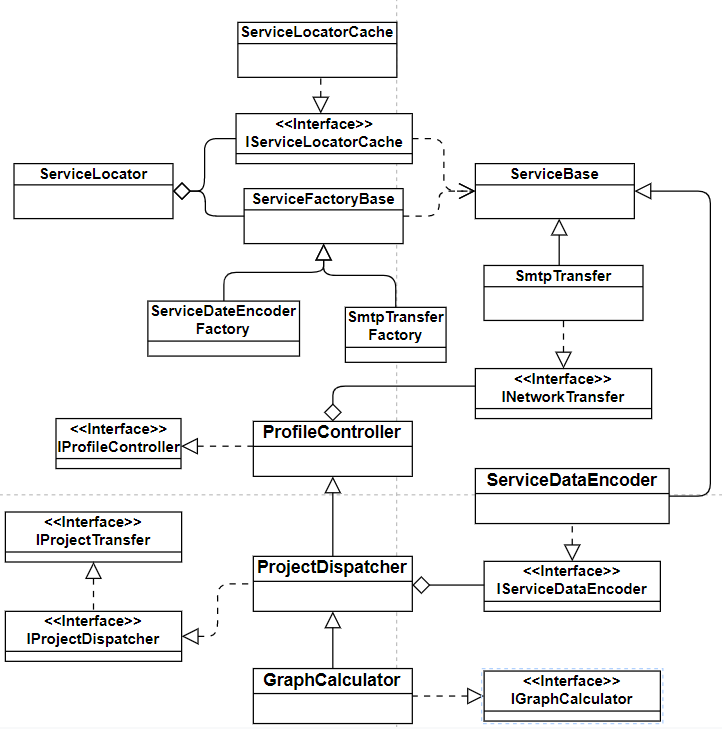


Рисунок 12－UML диаграмма классов сборки службы WCF

Для начала рассмотрим класс «ProfileController», реализующий интерфейс «IProfileController», который определяет контракт службы для взаимодействия с сетевым ресурсом. Интерфейс предоставляет поведение управления пользовательскими профилями: авторизация, регистрация, обновление, получение данных. Внутри класса располагается ссылка на объект совместимого с «INetworkTransfer» интерфейса－«SmtpTransfer», который ответственен за отправку сообщений на устанавливаемые адреса Email, используя для этого протокол «SMTP». От класса наследуется следующий тип службы «ProjectDispatcher», реализующий контракт «IProjectDispatcher». Данная служба используется для администрированием профильных проектов: создание, удаление, обновление. Также стоит заметить что, «IProjectDispatcher» расширяет своё поведение путём наследования от интерфейса «IProjectTransfer», который отвечает за экспорт и импорт отдельный поддерживаемых проектов.

Также служба инкапсулирует взаимодействием с объектом интерфейса «IServiceDataEncoder», который отвечает за безопасность и шифрование сохраняемых данных.

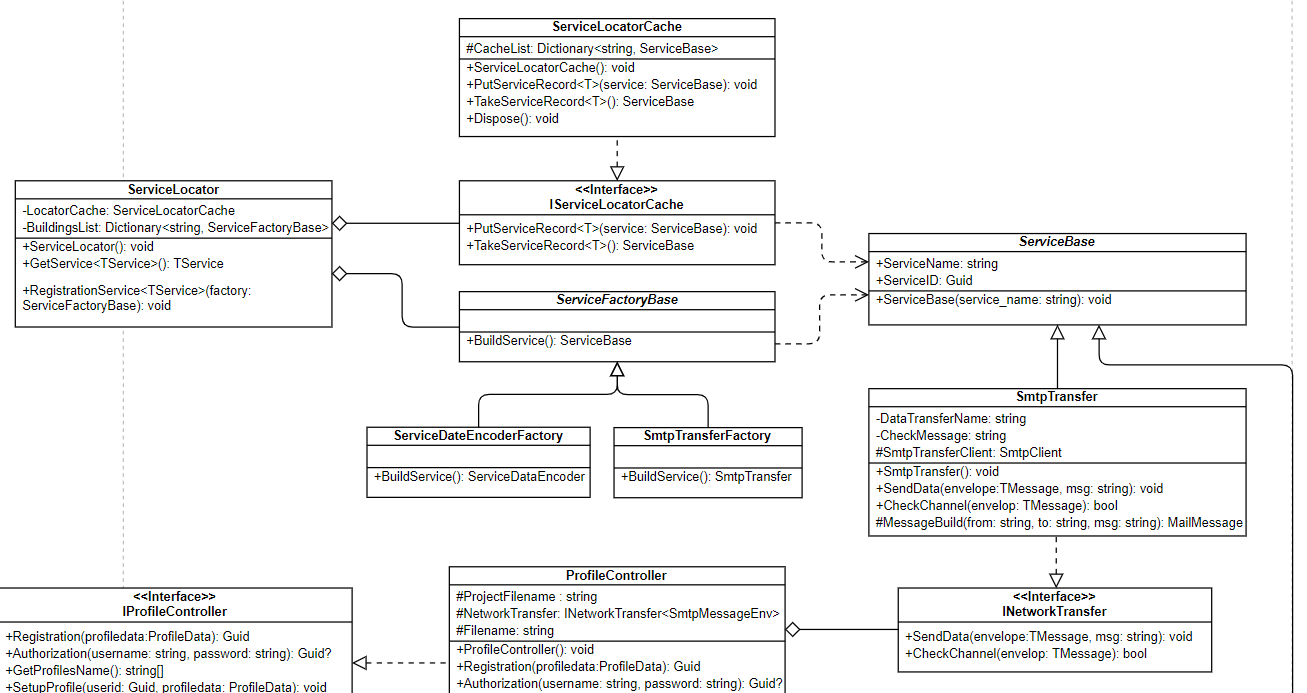


Рисунок 13－Подробное описание структуры и связей классов службы (часть 1)

Также необходимо упомянуть, что для устранения жёсткого связывания зависимостей используется синлтон компонент «ServiceLocator», который отвечает за поиск и предоставление необходимых зависимостей для успешной работы вышеописанных служб. Внутри локатора реализован механизм хранения кэша ранне созданных сервисов, чтобы сократить количество возможных вызовов конструкторов и ненужных выделений в памяти под новые объекты.

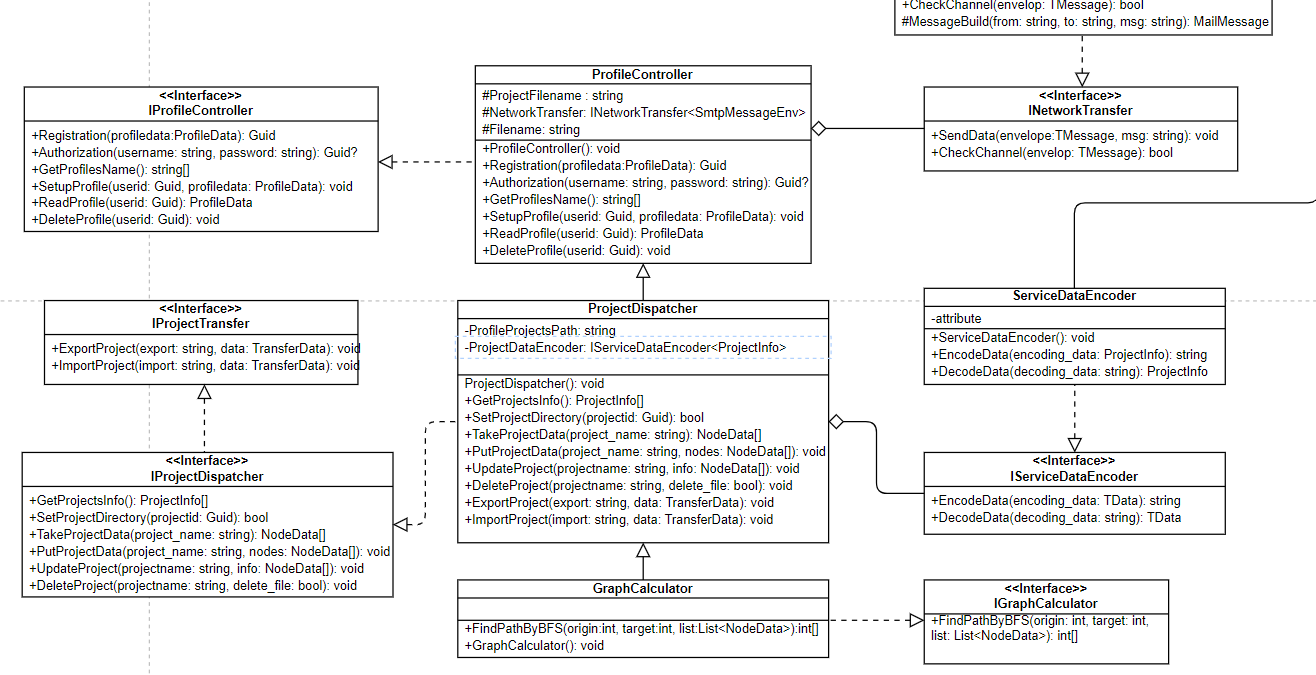


Рисунок 14－Подробное описание структуры и связей классов службы (часть 2)

Финальной службой в иерархии наследования является класс «GraphCalculator», отвечает за математические расчёты кратчайшего пути, используя для этого массив вершин, заданный в виде списка смежности

3.2 Разработка алгоритмов и методов

Первым делом, необходимо реализовать процедуру, которая в процессе выполнения будет обрабатывать нажатие на рабочую область редактора графа. Такой процедурой обработчиком выступает метод «OnEditorComponentClick» класса «EditorComponentBase». В начале выполнения тела метода производится проверка всех установленных (в текущий момент) вершин на поле редактора: производится вычисление столкновения текущей позиции курсора мыши и границ узла, и если такая вершина, которая попадает под указатель, существует, то ссылка на неё сохраняется для будущих вычислений.

Далее производится начальная инициализация локальной переменной, отвечающей за флаг обнуления свойство класса «SelectNodeID», которое хранит данные касательно текущей выбранной вершины редактора. После производится проверка была ли найдена вершина, с которой столкнулся курсор: если найдена, то проводится подключение ссылки на вершину с свойством класса и активации флага для дальнейшего обнуления. После чего происходит оповещение подписчиков события «NodeClicked». В противном случае, если вершина столкновение не была найдена, происходит вызов подписчиков события «FieldClicked». Блок схема описанного метода представлена на рисунке 15.

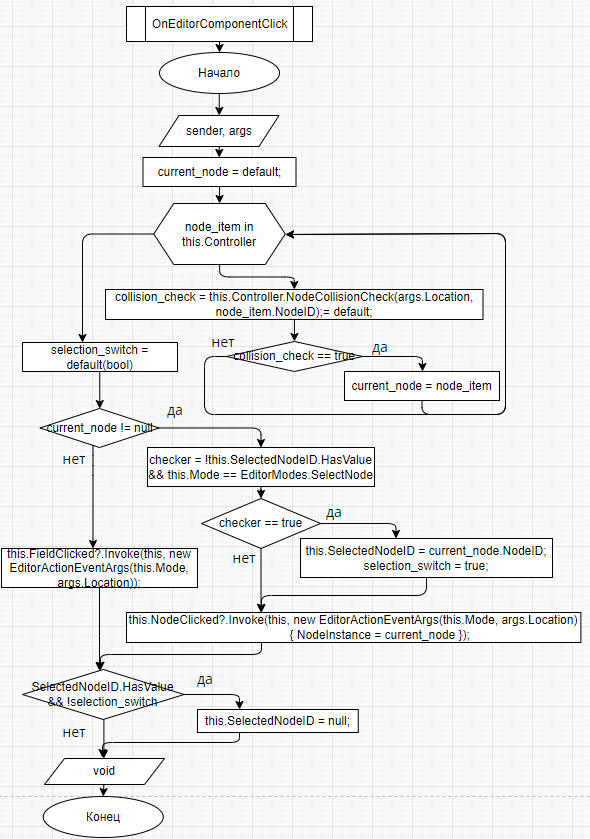


Рисунок 15－Блок схема метода «OnEditorComponentClick»

Далее необходимо предоставить поведение методу «EditorComponentPaint», отвечающий за отображение всех объектов на рабочей области редактора графов. Данная защищённая процедура является целевым методом экземпляра делегата (события) «Paint» класса «Panel», который срабатывает в момент запроса на отрисовку области контейнера (вызов метода «Invalidate»). Самая реализация основывается на вызовах абстрактных методов класса «EditorComponentBase», которые отвечают за вывод тех или иных компонентов системы на экран: сетка поля, узлы (вершины), ориентированные пути и т.д.

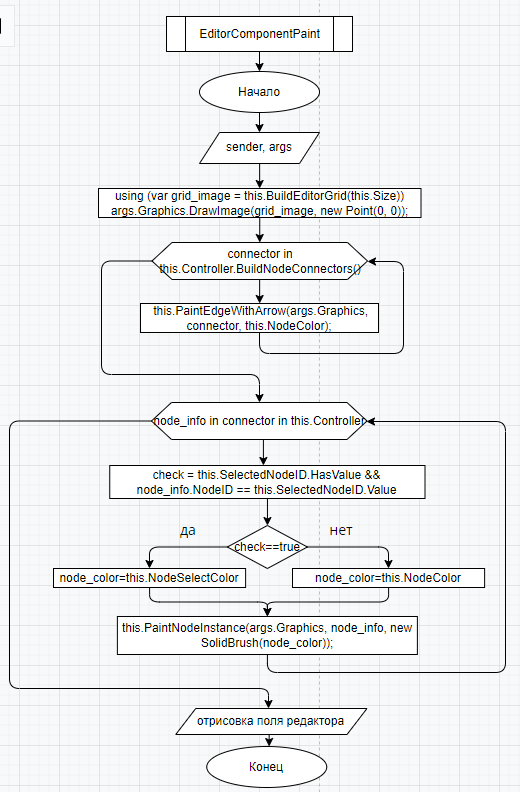


Рисунок 16－Блок схема метода «EditorComponentPaint»

Абстрактный метод «EditorComponentBase.BuildGraphPath» реализует механизм отображения найденного пути в заданном пользователем графе от начальной вершины до целевой. В качестве входного параметра принимает коллекцию объектов класса «NodesConnectorInfo» (модель, отображающая состояние соединения двух узлов). Внутри реализации метода производится получение экземпляра класса «System.Drawing.Graphics», который используется для взаимодействия с графическим интерфейсом панели. После производится отображение всех элементов на поле редактора, поверх которых совершается отрисовка составленных отрезков пути на основе переданного списка дуг графа.

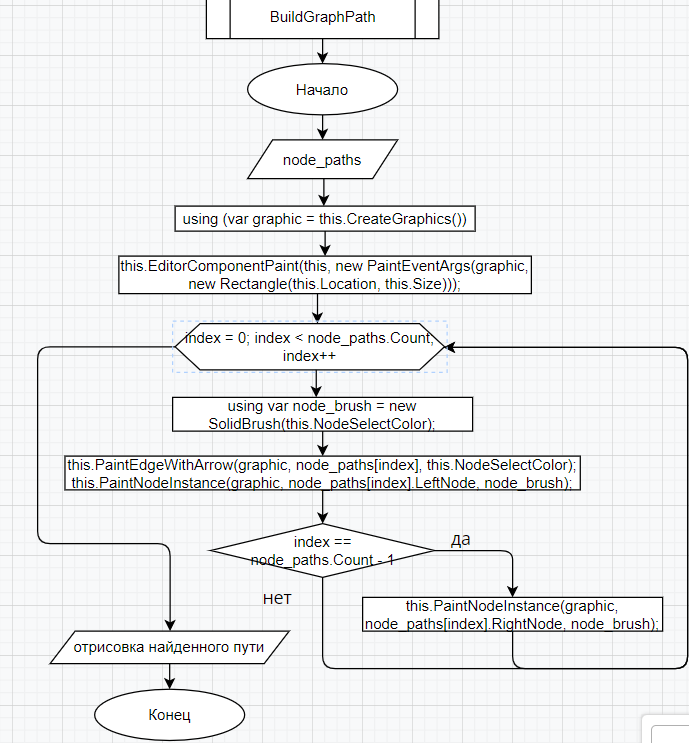


Рисунок 17－Блок схема метода «BuildGraphPath»

Для остальных методов и свойств проекта описание основной информации представлено внутри таблицы 1. В качестве описания используются следующие характеристики: название класса, назначение и заголовок метода.

Таблица 1－Список свойств и методов классов решения

|  |  |  |
| --- | --- | --- |
| Имя класса | Заголовок | Назначение |
| EditorComponentBase | public virtual Nullable<System.Int32> SelectedNodeID {get;set;} | Применяется для взаимодействия с значением идентификатора текущего выбранного узла |
| public virtual EditorComponents.EditorModes Mode { get; set; } | Предоставляет доступ к состоянию режима редактора графа: «AddNode» - добавление вершины, «RemoveNode» - удаление вершины, «SelectNode» - выбор вершины |
| public TNodesController Controller { get; set; } | Открывает доступ к контроллеру вершин редактора, где «TNodesController» реализует интерфейс «INodesControllerWithConnectors» |
| public abstract void ScalingGraphView(int scale\_value) | Позволяет масштабировать отображение поля графа на указанное значение |
| public abstract void PaintEdgeWithArrow(Graphics graphics, NodesConnectorInfo connector, Color color); | Используется для отображения дуги графа. В качестве входных параметров принимает объект GDI, данные состояния дуги, цвет для отображения. |
| public abstract void PaintNodeInstance(Graphics graphic, NodeModel node\_info, Brush node\_brush); | Используется для отображения вершины графа. В качестве входных параметров принимает объект GDI, данные состояния вершины графа и цвет для отображения. |
| protected abstract System.Drawing.Image BuildEditorGrid(Size size); | Применяется для создания запечатанной сетки поля редактора в виде экземпляра типа «Image». Принимает в качестве параметра размер клетки. |
| EditorComponentBuilder | public void SetLinkToForm(Form form\_link) | Выполняет подключение родительской формы приложения, на которую будет создаваться редактор. В качестве параметра принимает ссылку на форму,. |
| public void SetLinkToNodeController(System.Type controller\_type) | Выполняет подключение контроллера узлов, с помощью которого будут выполняться базовые операции редактора. В качестве параметра принимает тип контроллера. |
| public IEditorComponentBuilder<NodesController> AddEditorGeometry(Point position, Size size) | Устанавливает значения геометрической характеристики создаваемого редактора: ширину поля, местоположение поля внутри родительской формы. В качестве параметра принимает имя свойства, значение, и возвращает сборщик с обновлённым состоянием. |

Продолжение Таблицы 1

|  |  |  |
| --- | --- | --- |
| Имя класса | Заголовок | Назначение |
| EditorComponentBuilder | public IEditorComponentBuilder<NodesController> AddEditorConfiguration(IEditorConfigProvider provider) | Позволяет использовать объект интерфейса «IEditorConfigProvider» для настройки параметров редактора и на основе установленных значений конфигуратора. В качестве параметра принимает конфигуратор и возвращает сборщик с обновлённым состоянием. |
| public EditorComponentBase<NodesController> BuildEditor(string name) | Создаёт редактор графа, на основе собранных конфигураций и прикрепляет на указанную форму. В качестве параметра принимает имя будущего редактора и возвращает экземпляр класса «EditorComponentBase». |
| public IEditorComponentBuilder<NodesController> AddEditorProperty(string name, object value) | Используется для загрузки значений свойств для собираемого экземпляра редактора графа. В качестве параметра принимает имя свойства, значение, и возвращает сборщик с обновлённым состоянием. |
| NodesController | public SortedSet<NodeModel> NodesList { get; set; } | Свойство используется для предоставления доступа с отсортированному списку узлов. |
| public NodeModel? this[int node\_id] { get; set; } | Индексаторный метод предназначен для доступа к отдельному элементу вершины по идентификатору. В качестве параметра принимает номер идентификатора, на основе которого будет совершаться поиск. |
| public bool NodeCollisionCheck(Point position, int node\_id) | Производит проверку пересечения установленной вершины с указанной позицией. В качестве параметра принимает позицию курсора пересечения, идентификатор вершины и возвращает значение проверки. |
| public void AddNewNode(int pos\_x, int pos\_y) | Создаёт новый узел по указанным координатам. В качестве входным параметров указывается позиция вершины. |
| public void RemoveNode(int node\_id) | Производит удаление вершины. В качестве входного параметра принимает идентификатор хранящейся вершины. |
| public void SetNodeLinks(int node\_id, int required\_link\_id) | Устанавливает связь (ориентированную) между двумя сохранёнными вершинами. В качестве параметров принимает идентификаторы необходимых узлов. |
| public void RemoveNodeLinks(int node\_id, int required\_links\_id) | Удаляет связь (ориентированную) между двумя вершинами. В качестве параметров принимает идентификаторы необходимых узлов. |
| public List<NodesConnectorInfo> BuildNodeСonnectors() | Выполняет построение графа и возвращает результат в виде списка рёбер типа «NodesConnectorInfo». |

Продолжение Таблицы 1

|  |  |  |
| --- | --- | --- |
| Имя класса | Заголовок | Назначение |
| EditorConfigProvider | public IEditorConfigProvider PutConfigObject(object @object) | Данная перегрузка выполняет настройку параметра редактора, используя для этого готовый экземпляр конфигурации. В качестве входного параметра передаёт конфигурацию и возвращает текущее состояние конфигурации. |
|  | public IEditorConfigProvider PutConfigProperty(string name, object property) | Данная перегрузка выполняет настройку параметра редактора, используя для этого пару ключ-значение В качестве входного параметра принимается название параметра, значение и возвращает текущее состояние конфигурации.. |
|  | List<EditorConfigProperty> TakeConfig() | Производит сбор и вывод всех установленных настроек. Возвращает список составленных свойств. |
| ProfileController | public string[] GetProfilesName() | Позволяет просмотреть список имён зарегистрированных в системе пользователей. Возвращает множество имён учётных записей. |
|  | public System.Guid? Authorization(string username, string password) | Производит авторизацию в систему. В качестве входных параметров принимает имя пользователя и пароль, и возвращает идентификатор найденной учётной записи. |
|  | public System.Guid Registration(ServiceContracts.ProfileData profile\_data) | Производит регистрацию нового аккаунта системы. В качестве параметра принимает экземпляр типа «ProfileData», хранящий всю информацию о контакте пользователя, и возвращает идентификатор, созданной записи. |
|  | public void DeleteProfile(System.Guid userid) | Совершает очистку профиля пользователя из системы. В качестве параметра принимает идентификатор записи пользователя. |
|  | public void SetupProfile(System.Guid userid, ServiceContracts.ProfileData profile\_data) | Используется для настройки параметров учётной записи пользователя. В качестве параметра принимает идентификатор и объект с изменениями конфигурации профиля. |
|  | public ServiceContracts.ProfileData ReadProfile(System.Guid userid) | Используется для чтения данных профиля выбранного пользователя. В качестве входного аргумента принимает идентификатор пользователя и возвращает объект с конфигурацией |
| ProjectDispatcher | public bool SetProjectsDirectory(System.Guid project\_id) | Производит установку активного (в рамках текущей сессии) каталога, который хранит проекты пользователя. В качестве параметра принимает идентификатор профиля и возвращает состояние запроса. |
|  | public void CreateProject(ServiceContracts.ProjectInfo project\_info) | Создаёт новый проект, который прикрепляется к каталогу текущего пользователя редактора. В качестве входных аргументов принимает экземпляр типа «ProjectInfo», который хранит всю необходимую информацию о проекте. |

Продолжение Таблицы 1

|  |  |  |
| --- | --- | --- |
| Имя класса | Заголовок | Назначение |
|  | public void UpdateProject(string project\_name, ServiceContracts.ProjectInfo project\_info) | Обновляет содержимое описания проекта. В качестве параметра принимает имя проекта и новую информацию в виде объекта типа «ProjectInfo». |
|  | public void DeleteProject(string project\_name, bool delete\_file = true) | Совершает удаление выбранного проекта с учётной записи пользователя (при необходимости удаляется и файл с графом). В качестве параметров принимает имя проекта и флаг, который отвечает за удаление файла проекта из каталога пользователя. |
|  | public ServiceContracts.ProjectInfo[] GetProjectsInfo() | Производит вывод всех информации, касающейся всех добавленных проектов пользователя текущей сессии. Возвращает массив объектов «ProjectInfo». |
|  | public void PutProjectData(string project\_name, NodeData[] nodes\_field) | Упаковывает в файл проекта данные, отвечающие за представление графа проекта: позицию, количество вершин и др. В качестве входных аргументов принимает имя проекта и массив типа «NodeData», который хранит информацию о каждой вершине сохраняемого пользовательского графа. |
|  | public NodeData[] TakeProjectData(string project\_name) | Производит извлечение записанных в файл данных о состоянии сохранённого графа. В качестве параметра принимает имя проекта и возвращает массив объектов типа «NodeData» |
|  | public void ExportProject(string export\_entity, TransferData transfer\_data) | Производит экспорт проекта. В качестве параметра указывается имя проекта и экземпляр типа «TransferData», который хранит информацию о получателе проекта (адрес электронной почты и др.) |
|  | public void ImportProject(string import\_entity, TransferData transfer\_data) | Производит импорт выбранного поддерживаемого файла из хранилища компьютера. В качества параметра принимает имя проекта и информацию о файле с добавляемым проектом: полный путь к файлу в системе. |

3.3 Описание диалога с пользователем

При запуске программного решения перед пользователем появляется окно «Авторизации», которое выступает в качестве интерфейса для запуска или создания учётной записи системы. Данная форма разделена на две вкладки: вкладка «Авторизация» и вкладка «Регистрация».

На странице «Авторизации» представлены элементы, которые отвечают за ввод имя профиля пользователя и пароля. По нажатии кнопки «Войти в профиль» производиться проверка на валидность введённых данных и совершается поиск учётной записи внутри базы данных. Если профиль был найден, то пользователю открывается редактор графа с полным возможным функционалом программы. У пользователя есть возможность выбрать конкретный профиль из доступных, который будет использоваться для авторизации, для этого необходимо нажать на указатель выпадающего списка и выбрать нужное имя аккаунта. Также пользователь может использовать ссылку «Пропустить авторизацию» для использования редактора в качестве гостя, но в таком режиме некоторый функционал программы не будет доступен: сохранение и открытие проектов, настройка учётной записи и др.

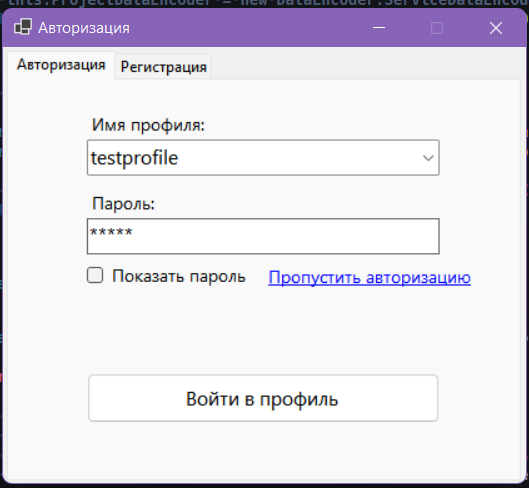


Рисунок 18－Страница авторизации

Вкладка «Регистрация» позволяет создать новый пользовательский профиль. Чтобы зарегистрировать новую учётную запись необходимо предоставить следующие данные: имя пользователя (используется при авторизации), пароль профиля, адрес электронной почты и пароль (для некоторых сервисов необходимо сгенерировать ключ для внешних приложений, который указывается вместо пароля от почты), а также полный путь до пустого каталога, в котором будут располагаться сохраняемые и загружаемые пользователем проекты карт графов. Для каждого текстового поля используется цветная панель, которая уведомляет пользователя о корректности введённых значений. Для ввода значения месторасположения каталога проектов, используется кнопка «Каталога», которая открывает диалоговое окно для выбора необходимой папки. Для регистрации новой учётной записи на основе введённой информации необходимо нажать на кнопку «Регистрация профиля».

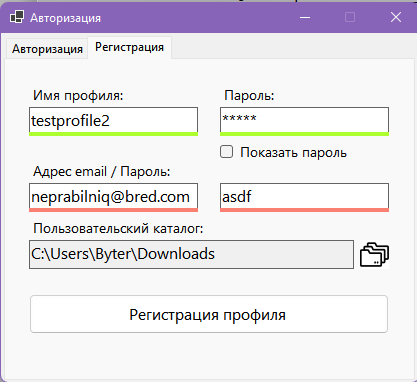


Рисунок 19－Страница регистрации

После того, как процесс авторизации завершён, пользователь может приступить к созданию и/или редактированию графа. В редакторе предусмотренны несколько встроенных режимов работы: режим добавления новой вершины－создаётся новый узел на месте нажатия ЛКМ по полю редактора; режим удаления вершины－производит удаление указанного узла (также удаляют ссылки на данную вершину у других узлов, тем самым производится удаление связанной дуги); режим выбора вершины－производится закрепление указателя редактора на выбранной вершине, после чего пользователь может либо соединить её с другой вершиной, либо переместить её на указанную позицию. Выбор вышеописанных инструментов производится с помощью переключателей, расположенных выше поля редактора. Также в этой секции окна присутствуют дополнительные инструменты редактирования: ползунок масштабирования (позволяет указать масштаб представления графа) и кнопка «Очистить» (очищает поле редактора).

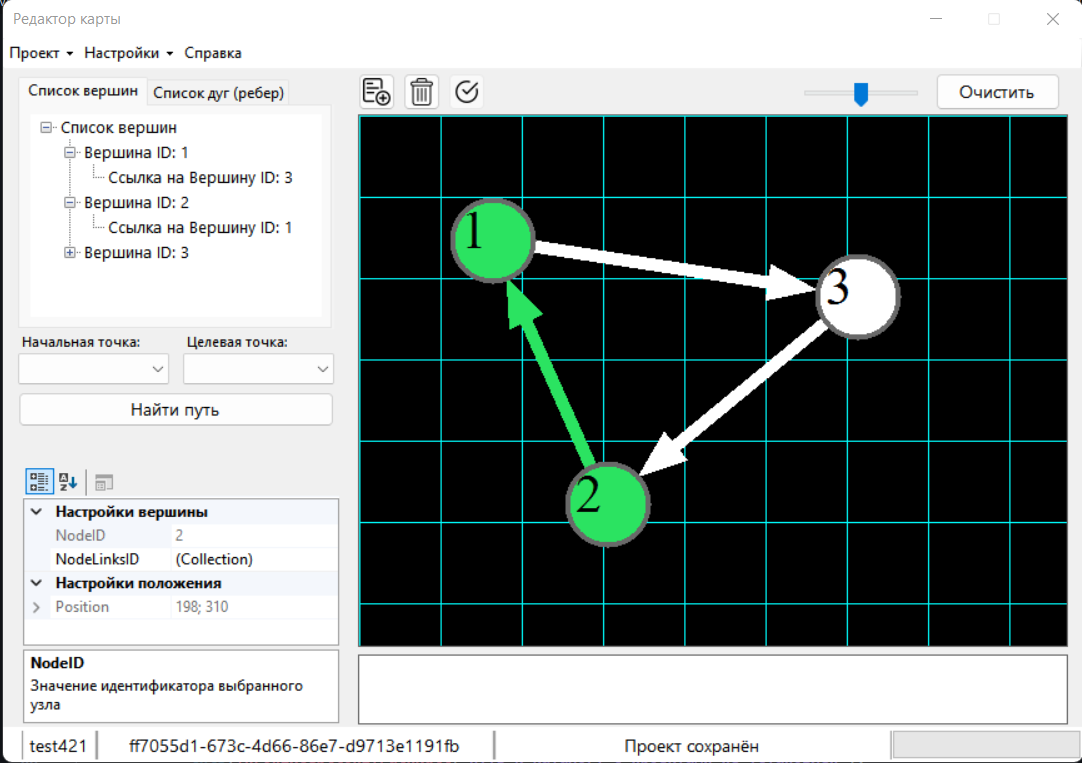


Рисунок 20－Окно «Редактор карты»

На боковой панели формы расположен контейнер, который содержит две вкладки, содержащие описание всех элементов добавленных пользователем в процессе редактирования: вкладка «Список вершин» представляет дерево элементов, состоящее из названия добавленных вершин и подключённых ссылок; вкладка «Список дуг» содержит список определённых соединений узлов (дуг). Для каждого элемента панели можно вызвать контекстное меню, с помощью можно произвести удаление вершины или дуги.

Ниже располагается панель свойств, которая отображает все характеристики выбранной вершины редактора: идентификатор, список ссылок на другие смежные вершины и местоположение элемента на поле редактора.

Нижняя полоса используется для отображения информации о совершенных действиях, идентификаторе авторизованного пользователя и названия загруженного проекта. Верхняя полоса содержит множество выпадающих списков, которые используются для перехода между вспомогательными окнами приложения.

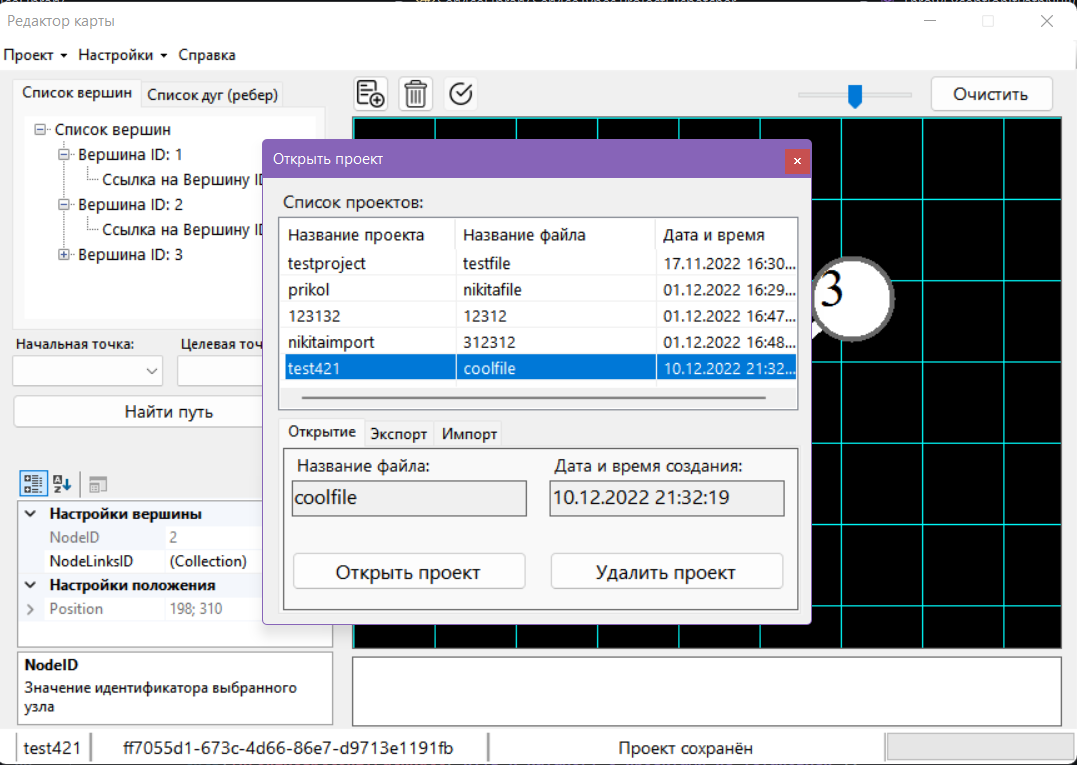


Рисунок 21－Диалоговое окно «Открыть проект»

Чтобы открыть ранне созданный проект необходимо перейти в «Проект/Открыть проект», после чего откроется диалоговое окно в котором будут отображены все проекты сохранённые в каталоге пользователя в виде списка. Также в нижней части окна отображается панель, содержащая 3 вкладки: вкладка «Открытие» используется для извлечение информации о выбранном проекте и загрузки его в текущую пользовательскую сессию (также есть возможность удалить выбранный проект); вкладка «Экспорт» позволяет экспортировать выбранный проект по указанной электронной почте; вкладка «Импорт» используется для регистрации нового проекта на основе указанного файла, который обладает совместимым расширением (graphproj / json).

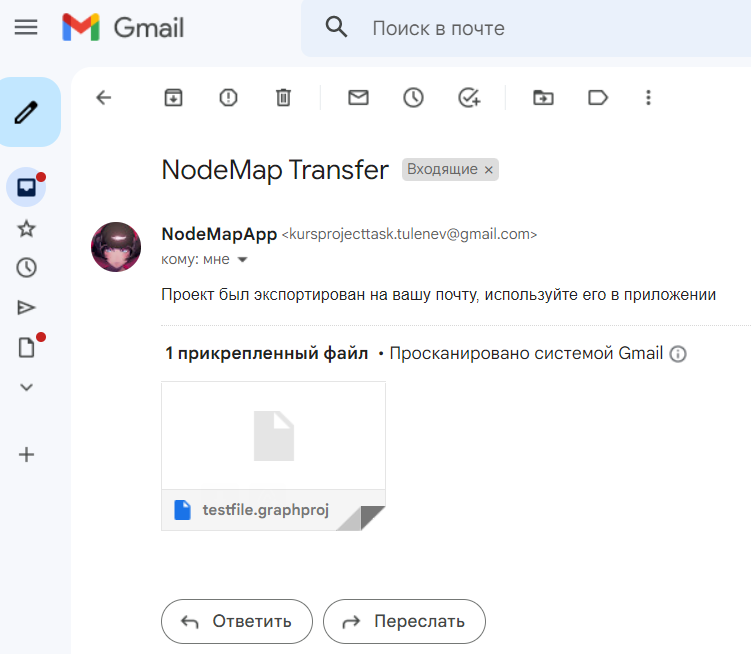


Рисунок 22－Экспорт проекта на электронную почту

Чтобы сохранить разработанный в редакторе граф, используется элемент «Проект/Сохранить проект», который открывает диалоговое окно «Сохранить проект». Внутри используются элементы управления для указания основных данных о сохраняемом проекте: название проекта, название файла, дата и время создания. Для сохранения нового файла используется кнопка «Сохранить проект». Если производится сохранение редактируемого проекта, используется кнопка «Изменить проект».

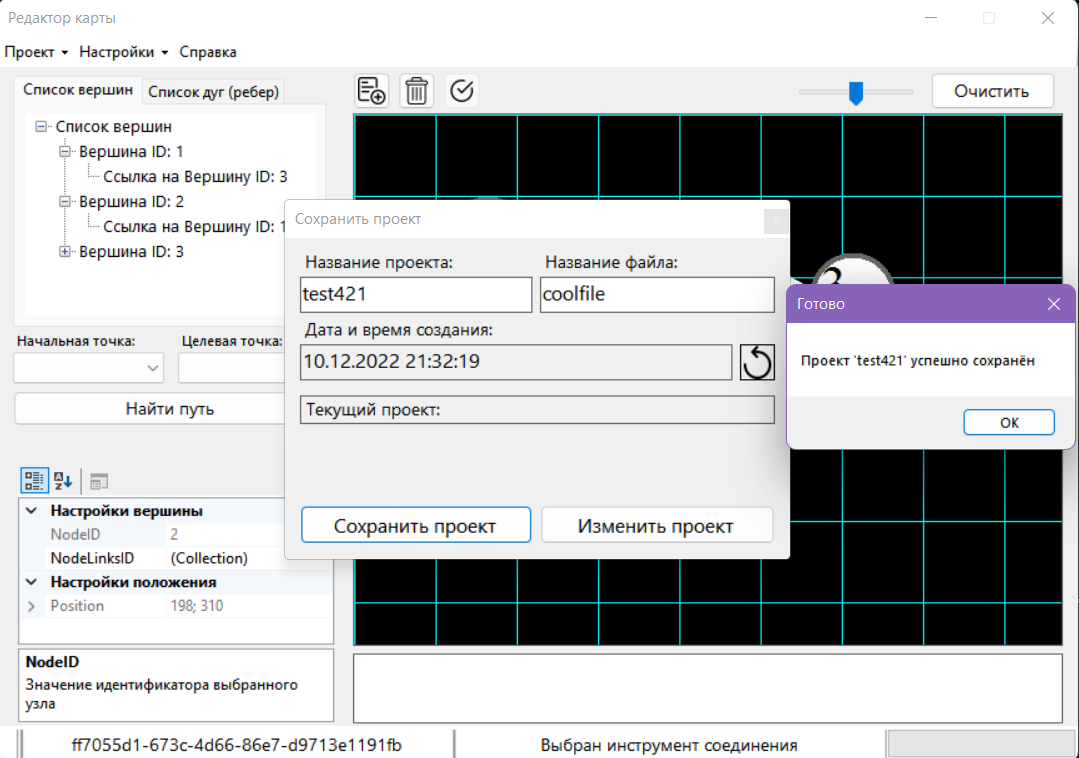


Рисунок 23－Диалоговое окно «Сохранить проект»

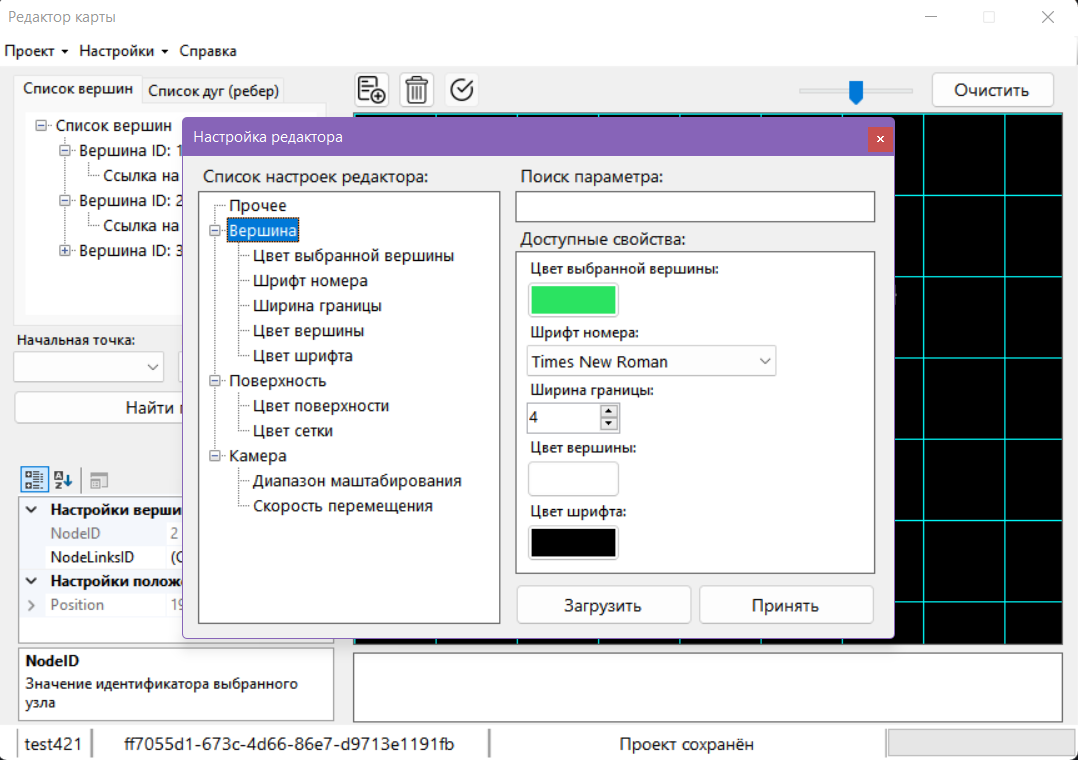


Рисунок 24－Диалоговое окно «Настройка редактора»

Также для настройки поля редактора используется диалоговое окно «Настройка редактора». Для получения доступа к настройка необходимо перейти в «Настройки/Настройки редактора». С помощью боковой панели можно выбрать необходимый раздел с настройками. Внутри панели «Доступные свойства», используя различные элементы управления, производится изменение параметров отображения поля редактора.

Если пользователю будет необходимо настроить параметры учётной записи, то необходимо открыть диалоговое окно «Настройка профиля», используя «Настройки/Настройка профиля». Все доступные настройки показаны на рисунке 24.

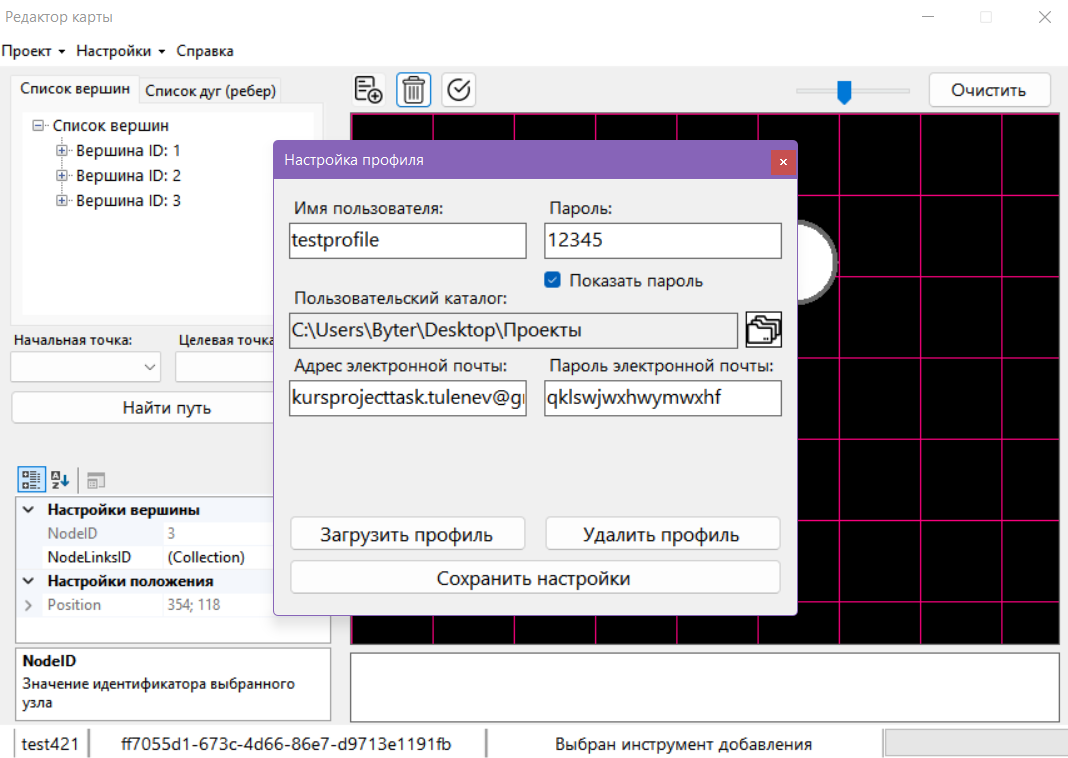


Рисунок 25－Диалоговое окно «Настройка профиля»

Для вычисления кратчайшего пути в установленном графе используется панель в левой части редактора. С помощью выпадающих списков выбираются начальная и целевая вершина пути, после чего необходимо нажать кнопку «Найти путь». Когда процесс расчёта завершиться, на поле редактора будет отображён найденный путь (рисунок 25).

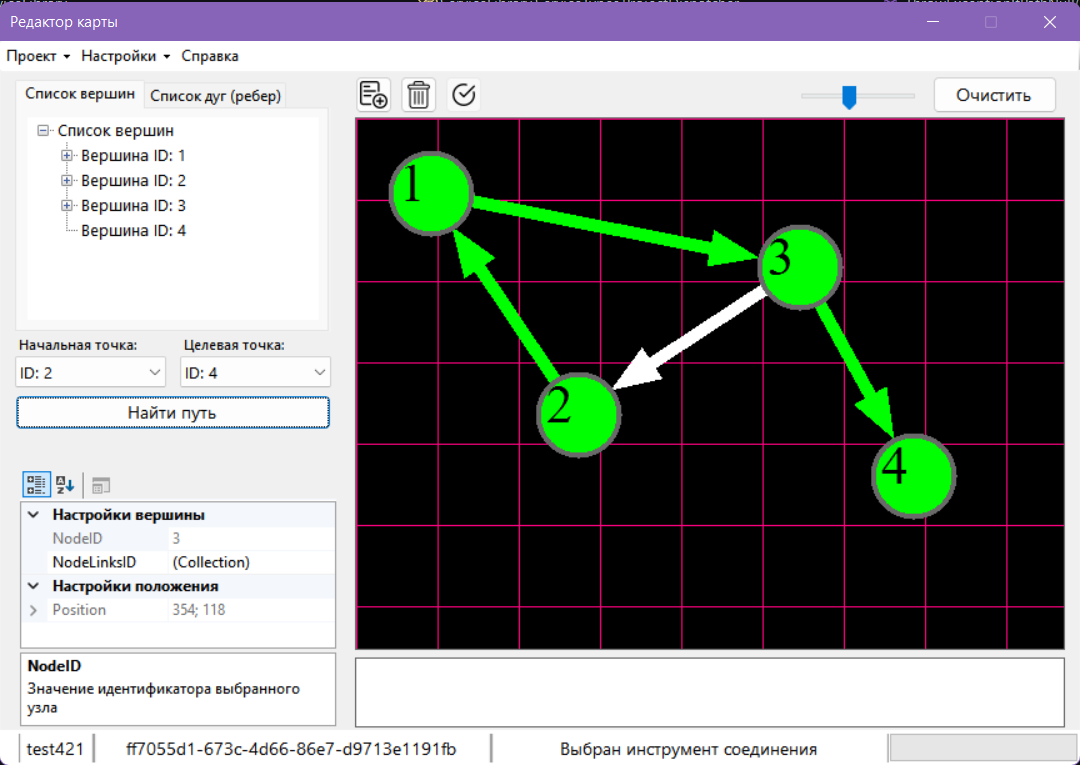


Рисунок 26－Вычисление кратчайшего пути в графе

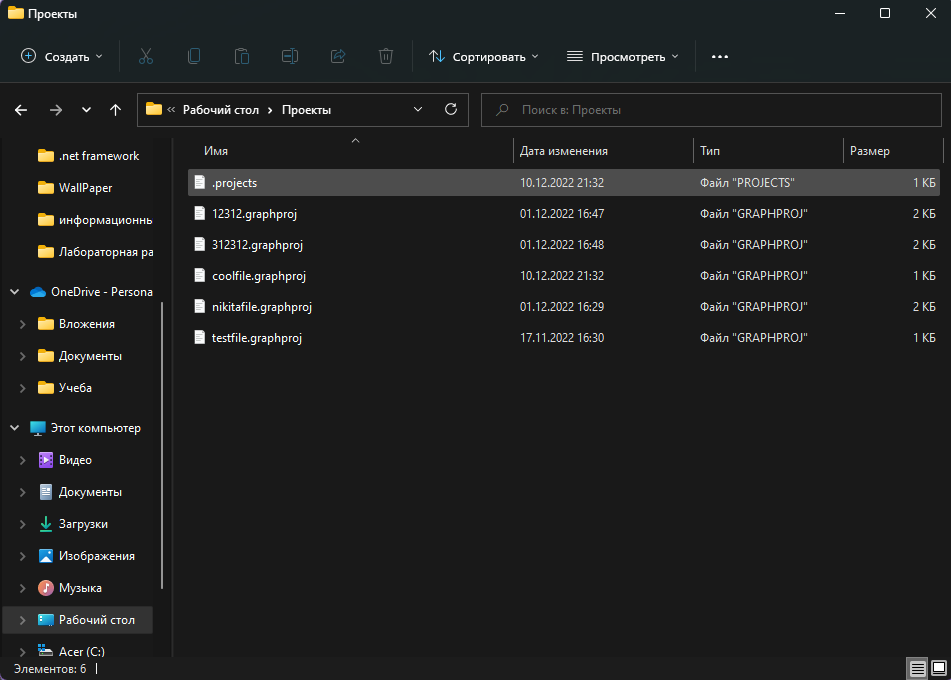


Рисунок 27－Содержание каталога с проектами

На рисунке 26 показано содержание установленного для профиля каталога, который используется для сохранения проектов. Файл «.project» содержит информацию о всех подключаемых к профилю проектах: название проекта, название файла с данными графа, время создания. Файлы с расширением «.graphproj» представляют сохранённые графы.

3.4 Минимальные системные требования

Для корректной работы программы компьютер пользователя должен соответствовать минимальным заданным системным требованием, которые представляют собой следующие параметры: операционная система Windows 7 и выше, процессор 800 МГц, оперативная память 128 Мб, видеокарта 32Мб видеопамяти, памяти на жёстком диске ~16Мб.

3.5 Контрольный пример

Для проверки корректной работы математической составляющей программы, приведём контрольный пример. Для начала нарисуем произвольный граф и попробуем сами найти кратчайшее расстояние между двумя вершинами, придерживаясь всех правил вычисления описанных ранее в работе.

Выберем в качестве начальной вершины №1, а в качестве целевой №6. Первым делом вершина №1 помещается в очередь и сразу же извлекается из неё для просмотра доступных смежных элементов, которые добавляются в очередь: {2, 4, 3}. После чего достаём из контейнера вершину №2 и проводим аналогичные операции, получив в результате обработки состояние очереди текущей итерации: {4, 3}. Произведём вычисления для всех вершин графа, пока не дойдём до целевой вершины №6, изменяя очередь обработки следующим образом: [4] = {3, 5}, [3] = {5, 6}, [5] = {6}, [6] = { }. В процессе поиска целевой вершины, все добавляемые элементы обозначаются дополнительной характеристикой－уровнем обработки. После того, как целевой элемент был найден, производится составление пути до начальной вершины: в путь добавляются все ссылки, достижимые с предыдущего уровня обработки {[6] - [2], [3] - [1], [1] - [0]}. Произведём операцию реверсирования полученного множества и получим результат расчёта: {1, 3, 6}.

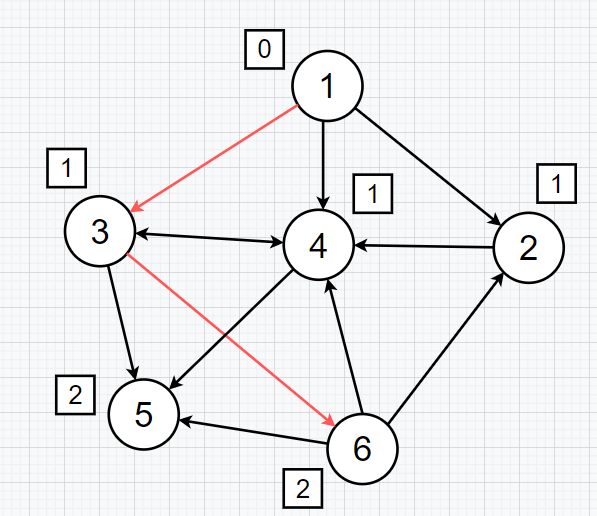


Рисунок 28－Контрольный пример графа

Далее используем программное решение для вычисления представленного графа. Составим граф, используя предоставленные редактором инструменты ввода и вычислим расстояние от вершины №1 до вершины №6.

Программа вывела результат: {1, 3, 6}, который полностью совпадает с полученным нами при ручном вычислении ответом, следовательно, программа функционирует корректно.

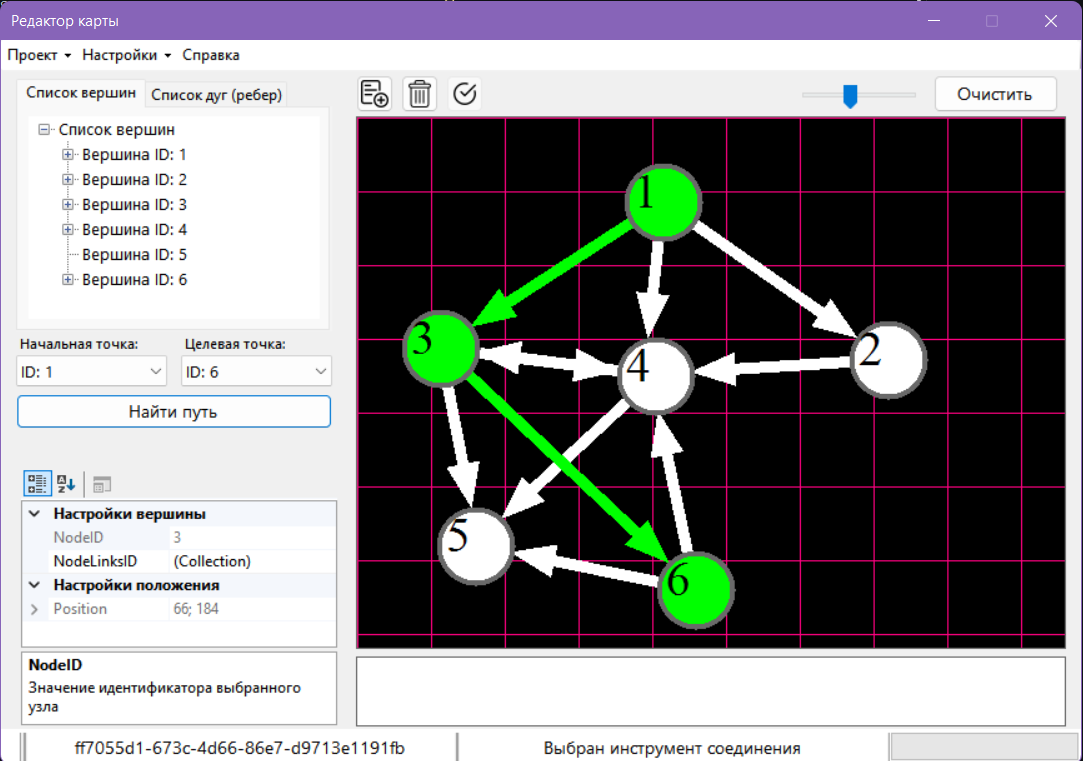


Рисунок 29－Проверка графа в редакторе

ЗАКЛЮЧЕНИЕ

В ходе курсового проектирования на платформе .NET языка C# с использованием среды разработки Visual Studio 2022 была реализована программа для построения графов и определения пути с минимальным числом дуг на основе поиска в ширину.

Разработанная программа предоставляет пользователю систему с удобным интерфейсом и инструментарием для построения графов различной сложности, для выполнения операций, выбранных пользователем, организован контроль ошибок при возникновении.

Все требования к программе выполнены успешно. Реализованы следующие аспекты: иерархия классов, соответствующая и использующая три основных принципа ООП, необходимые для корректной работы, функции. Интуитивно понятный редактор для построения графов. Списки, предоставляющие удобное обращение к элементам, добавление новых элементов, удаление. Были реализованы методы сохранения графов в файл различными способами и дальнейшего открытия файлов сохранений при помощи интерфейса программы. Использование библиотеки Windows Forms, для создания форм пользовательского интерфейса. Математическая составляющая программы также функционирует корректно – проверено при помощи программ-аналогов и ручного расчёта.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

* 1. Документация по языку C# / [Электронный ресурс]－Режим доступа: <https://learn.microsoft.com/ru-ru/dotnet/csharp>
  2. Документация по библиотеке Windows Forms / [Электронный ресурс]－Режим доступа: [https://learn.microsoft.com/ru-ru/dotnet/desktop/ winforms](https://learn.microsoft.com/ru-ru/dotnet/desktop/winforms)
  3. Чарльз Петцольд Программирование для Microsoft Windows на C# - В 2-х томах – Том 1. Переведено с английского – М.: Издательско-торговый дом “Русская Редакция”, 2002. – 576 с.
  4. Шилдт Герберт С# 4.0 полное руководство.:. Переведено с английского – М.: ООО “И.Д. Вильямс”, 2011
  5. Курипта О.В. Основы программирования и алгоритмизации: практикум / О.В.Курипта, О.В. Минакова, Д.К. Проскурин; Воронежский ГАСУ. – Воронеж, 2015. – 132 с.
  6. Н. Кристофидес Теория графов. Алгоритмический подход. – М.: Мир, 1978. – 430 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программы на C#

Файл «EditorComponents/EditorComponent.cs»

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Drawing.Drawing2D;

using System.IO;

using System.Linq;

using System.ServiceModel.Channels;

using System.Text;

using System.Threading.Tasks;

using CSCourseWork.NodesControllers;

using static System.Math;

namespace CSCourseWork.EditorComponents

{

public class EditorComponent : EditorComponents.EditorComponentBase<NodesController>

{

public override event EditorComponents.EditorActionEventHandler? NodeScaled;

private const int DefaultGridDelta = 5, MinimunScaleValue = 10, NodeDefaultSize = 10;

private bool movingbutton\_hold = default(bool);

private Point movingposition\_buffer = default(Point);

public override EditorColor NodeSelectColor { get; set; } = new EditorColor(220, 20, 60);

public override EditorColor EditorBackColor

{

get => new EditorColor(this.BackColor.R, this.BackColor.G, this.BackColor.B);

set => this.BackColor = value;

}

public override int NodeSize

{

protected set {

this.Controller.NodeSize = (value >= node\_scale.Min && value <= node\_scale.Max)

? value : this.Controller.NodeSize;

}

get { return this.Controller.NodeSize; }

}

private EditorModes editor\_mode = default(EditorModes);

public override EditorModes Mode

{

set {

this.editor\_mode = value;

switch (value)

{

case EditorModes.AddNode: this.Cursor = Cursors.Cross; break;

case EditorModes.RemoveNode: this.Cursor = Cursors.No; break;

case EditorModes.SelectNode: this.Cursor = Cursors.Hand; break;

}

}

get => this.editor\_mode;

}

private EditorComponents.EditorScale node\_scale = new(1, 100);

public override EditorComponents.EditorScale NodeScaleRange

{

get { return this.node\_scale; }

set {

if (value.Max - value.Min <= 0)

{ throw new EditorComponentException("Неверные значение масштаба"); }

this.NodeSize = (value.Max - value.Min) / 2 + value.Min;

this.node\_scale = value;

}

}

public EditorComponent(NodesController controller) : base(controller)

{

(this.DoubleBuffered, this.BorderStyle, this.NodeSize) = (true, BorderStyle.FixedSingle, NodeDefaultSize);

this.MouseWheel += new MouseEventHandler(this.EditorComponent\_MouseWheel);

this.MouseMove += new MouseEventHandler(this.EditorComponentMouseMove);

this.NodeClicked += new EditorActionEventHandler(this.EditorComponentNodeClicked);

this.FieldClicked += new EditorActionEventHandler(this.EditorComponentFieldClicked);

this.MouseUp += new MouseEventHandler(delegate (object? sender, MouseEventArgs args)

{ if (args.Button == MouseButtons.Right) this.movingbutton\_hold = false; });

}

public EditorComponent() : this(new NodesController()) { }

protected override void OnEditorComponentClick(object? sender, MouseEventArgs args)

{

if (args.Button == MouseButtons.Right) { this.movingbutton\_hold = true; return; }

base.OnEditorComponentClick(sender, args);

}

public override void BuildGraphPath(List<NodesConnectorInfo> node\_paths)

{

using (var graphic = this.CreateGraphics())

{

this.EditorComponentPaint(this, new PaintEventArgs(graphic, new Rectangle(this.Location, this.Size)));

for (var index = 0; index < node\_paths.Count; index++)

{

using var node\_brush = new SolidBrush(this.NodeSelectColor);

this.PaintEdgeWithArrow(graphic, node\_paths[index], this.NodeSelectColor);

this.PaintNodeInstance(graphic, node\_paths[index].LeftNode, node\_brush);

if (index == node\_paths.Count - 1)

{

this.PaintNodeInstance(graphic, node\_paths[index].RightNode, node\_brush);

}

}

}

}

protected override Image BuildEditorGrid(Size size)

{

var grid\_bitmap = new Bitmap(size.Width, size.Height);

using (var grid\_graphic = Graphics.FromImage(grid\_bitmap))

{

var shift\_x = this.movingposition\_buffer.X \* this.NodeSize / size.Width \* this.NodeMovingSpeed \* 2;

var shift\_y = this.movingposition\_buffer.Y \* this.NodeSize / size.Height \* this.NodeMovingSpeed \* 2;

using var grid\_color = new SolidBrush(this.EditorGridColor);

for (int i = 0; i < size.Width; i += this.NodeSize)

{

grid\_graphic.DrawLine(new Pen(grid\_color), new Point(i + shift\_x, 0), new Point(i + shift\_x, size.Height));

grid\_graphic.DrawLine(new Pen(grid\_color), new Point(0, i + shift\_y), new Point(size.Width, i + shift\_y));

}

}

return grid\_bitmap;

}

public override void PaintEdgeWithArrow(Graphics graphics, NodesConnectorInfo connector, Color color)

{

int delta\_x = connector.RightNode.Position.X - connector.LeftNode.Position.X, // x1 - x0

delta\_y = connector.RightNode.Position.Y - connector.LeftNode.Position.Y; // y1 - y0

double D = Math.Sqrt(Math.Pow(delta\_x, 2) + Math.Pow(delta\_y, 2)), S = D - (this.NodeSize / 2);

var b = delta\_y \* (1 - ((double)this.NodeSize / 2 / D));

var a = Math.Sqrt(Math.Pow(S, 2) - Math.Pow(b, 2));

var x = connector.LeftNode.Position.X + Math.Sign(delta\_x) \* a;

var y = connector.LeftNode.Position.Y + b;

var line\_pen = new Pen(color, this.NodeSize / 6) { CustomEndCap = new AdjustableArrowCap(3, 4, true) };

graphics.DrawLine(line\_pen, connector.LeftNode.Position, new Point((int)x, (int)y));

}

public override void PaintNodeInstance(Graphics graphic, NodeModel node\_info, Brush node\_brush)

{

var node\_position = new Point(node\_info.Position.X - this.NodeSize / 2, node\_info.Position.Y - this.NodeSize / 2);

var node\_geometry = new Rectangle(node\_position, new Size(this.NodeSize, this.NodeSize));

graphic.FillEllipse(node\_brush, node\_geometry);

graphic.DrawEllipse(new Pen(Brushes.DimGray, this.NodeBorderWidth), node\_geometry);

using (var node\_font = new Font(this.NodeFontFamily.FontFamily, this.NodeSize / 2))

{

using var font\_color = new SolidBrush(this.NodeFontColor);

graphic.DrawString(node\_info.NodeID.ToString(), node\_font, font\_color, node\_position);

}

}

public override void ScalingGraphView(int scale\_value)

{

if (scale\_value >= this.node\_scale.Min && scale\_value <= this.node\_scale.Max)

{

foreach (var item in this.Controller)

{

int position\_x = item.Position.X / Controller.NodeSize \* scale\_value,

position\_y = item.Position.Y / Controller.NodeSize \* scale\_value;

item.Position = new Point(position\_x, position\_y);

}

this.Controller.NodeSize = scale\_value;

}

else throw new EditorComponentException("Значение вне диапазона");

this.Invalidate();

}

private void EditorComponent\_MouseWheel(object? sender, MouseEventArgs args)

{

var buffer\_delta = this.Controller.NodeSize + Math.Sign(args.Delta) \* DefaultGridDelta;

try { this.ScalingGraphView(buffer\_delta); } catch (EditorComponentException) { return; }

this.NodeScaled?.Invoke(this, new EditorActionEventArgs(this.Mode, default)

{ NodeScale = buffer\_delta });

}

private void EditorComponentNodeClicked(object? sender, EditorActionEventArgs args)

{

if (args.NodeInstance == null) return;

switch (args.NodeAction)

{

case EditorModes.RemoveNode: this.Controller.RemoveNode(args.NodeInstance.NodeID); break;

case EditorModes.SelectNode when this.SelectedNodeID.HasValue:

this.Controller.SetNodeLinks(this.SelectedNodeID.Value, args.NodeInstance.NodeID); break;

default: return;

}

this.Invalidate();

}

private void EditorComponentMouseMove(object? sender, MouseEventArgs args)

{

if (this.movingbutton\_hold != true) return;

int moving\_direction\_x = Math.Sign(args.X - this.movingposition\_buffer.X),

moving\_direction\_y = Math.Sign(args.Y - this.movingposition\_buffer.Y);

foreach (var node\_info in this.Controller.NodesList)

{

node\_info.Position = new Point(node\_info.Position.X + moving\_direction\_x \* this.NodeMovingSpeed,

node\_info.Position.Y + moving\_direction\_y \* this.NodeMovingSpeed);

}

this.movingposition\_buffer = args.Location;

this.Invalidate();

}

private void EditorComponentFieldClicked(object? sender, EditorActionEventArgs args)

{

if (args.NodeAction == EditorModes.AddNode)

{

try { this.Controller.AddNewNode(args.ActionPosition.X, args.ActionPosition.Y); }

catch (NodesControllerException node\_error)

{

MessageBox.Show($"{node\_error.Message} - Node: {node\_error.Node?.NodeID}", "Ошибка"); return;

}

}

else if (args.NodeAction == EditorModes.SelectNode && this.SelectedNodeID.HasValue)

{

this.Controller[this.SelectedNodeID.Value]!.Position = args.ActionPosition;

}

else return;

this.Invalidate();

}

}

}

Файл «EditorComponents/EditorComponentBase.cs»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms.VisualStyles;

using CSCourseWork.EditorConfiguration;

using CSCourseWork.NodesControllers;

using CSCourseWork.Windows;

namespace CSCourseWork.EditorComponents

{

public enum EditorModes : System.SByte { AddNode, RemoveNode, SelectNode };

[AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = true)]

public sealed class EditorSettingsAttribute : System.Attribute

{

public System.String SettingName { get; private set; } = string.Empty;

public System.String SettingSection { get; set; } = string.Empty;

public EditorSettingsAttribute(string name) : base() => this.SettingName = name;

}

public sealed class EditorActionEventArgs : System.EventArgs

{

public EditorModes NodeAction { get; private set; } = default(EditorModes);

public Point ActionPosition { get; private set; } = default(Point);

public NodeModel? NodeInstance { get; set; } = null;

public System.Int32? NodeScale { get; set; } = null;

public EditorActionEventArgs(EditorModes action, Point position) : base()

{ this.NodeAction = action; this.ActionPosition = position; }

}

public delegate void EditorActionEventHandler(object? sender, EditorActionEventArgs args);

public sealed class EditorComponentException : System.Exception

{ public EditorComponentException(string message) : base(message) { } }

public abstract class EditorComponentBase<TNodesController> : System.Windows.Forms.Panel

where TNodesController : NodesControllers.INodesControllerWithConnectors, new()

{

public virtual event EditorComponents.EditorActionEventHandler? NodeClicked;

public virtual event EditorComponents.EditorActionEventHandler? FieldClicked;

public virtual event EditorComponents.EditorActionEventHandler? NodeScaled;

public virtual EditorComponents.EditorModes Mode { get; set; } = default;

public virtual System.Int32 NodeSize { get; protected set; }

public TNodesController Controller { get; private set; }

[EditorSettingsAttribute("Диапазон маштабирования", SettingSection = "Камера")]

public virtual EditorComponents.EditorScale NodeScaleRange { get; set; } = new(10, 100);

[EditorConfiguration.EditorConfigTargetAttribute]

[EditorSettingsAttribute("Скорость перемещения", SettingSection = "Камера")]

public virtual System.Int32 NodeMovingSpeed { get; set; } = default(int) + 1;

[EditorConfiguration.EditorConfigTargetAttribute]

[EditorSettingsAttribute("Шрифт номера", SettingSection = "Вершина")]

public virtual EditorComponents.EditorFontFamily NodeFontFamily { get; set; } = new();

[EditorConfiguration.EditorConfigTargetAttribute]

[EditorSettingsAttribute("Ширина границы", SettingSection = "Вершина")]

public virtual System.Int32 NodeBorderWidth { get; set; } = default(int);

[EditorSettingsAttribute("Цвет сетки", SettingSection = "Поверхность")]

public virtual EditorComponents.EditorColor EditorGridColor { get; set; } = default(EditorColor);

[EditorSettingsAttribute("Цвет выбранной вершины", SettingSection = "Вершина")]

public virtual EditorComponents.EditorColor NodeSelectColor { get; set; } = default(EditorColor);

[EditorSettingsAttribute("Цвет поверхности", SettingSection = "Поверхность")]

public virtual EditorComponents.EditorColor EditorBackColor { get; set; } = default(EditorColor);

[EditorSettingsAttribute("Цвет вершины", SettingSection = "Вершина")]

public virtual EditorComponents.EditorColor NodeColor { get; set; } = default(EditorColor);

[EditorSettingsAttribute("Цвет шрифта", SettingSection = "Вершина")]

public virtual EditorComponents.EditorColor NodeFontColor { get; set; } = new(255, 255, 255);

private int? selected\_nodeid = default(int?);

public virtual Nullable<System.Int32> SelectedNodeID

{

set => this.selected\_nodeid =

(value > 0 || value <= this.Controller.NodesList.Count) ? value : null;

get => this.selected\_nodeid;

}

public EditorComponentBase(TNodesController controller) : base()

{

this.MouseDown += new MouseEventHandler(this.OnEditorComponentClick);

this.Paint += new PaintEventHandler(this.EditorComponentPaint);

this.Controller = controller;

}

protected void EditorComponentPaint(object? sender, PaintEventArgs args)

{

using (var grid\_image = this.BuildEditorGrid(this.Size)) args.Graphics.DrawImage(grid\_image, new Point(0, 0));

foreach (var connector in this.Controller.BuildNodeСonnectors())

{ this.PaintEdgeWithArrow(args.Graphics, connector, this.NodeColor); }

this.Controller.ToList().ForEach(delegate (NodeModel node\_info)

{

var node\_color = (this.SelectedNodeID.HasValue && node\_info.NodeID == this.SelectedNodeID.Value)

? this.NodeSelectColor : this.NodeColor;

this.PaintNodeInstance(args.Graphics, node\_info, new SolidBrush(node\_color));

});

}

protected virtual void OnEditorComponentClick(object? sender, MouseEventArgs args)

{

NodesControllers.NodeModel? current\_node = default;

foreach (var node\_item in this.Controller)

{

var collision\_check = this.Controller.NodeCollisionCheck(args.Location, node\_item.NodeID);

if (collision\_check) { current\_node = node\_item; break; }

}

var selection\_switch = default(bool);

if (current\_node != null)

{

if (!this.SelectedNodeID.HasValue && this.Mode == EditorModes.SelectNode)

{

this.SelectedNodeID = current\_node.NodeID; selection\_switch = true;

}

this.NodeClicked?.Invoke(this, new EditorActionEventArgs(this.Mode, args.Location)

{ NodeInstance = current\_node });

}

else this.FieldClicked?.Invoke(this, new EditorActionEventArgs(this.Mode, args.Location));

if (this.SelectedNodeID.HasValue && !selection\_switch) this.SelectedNodeID = null;

}

protected abstract System.Drawing.Image BuildEditorGrid(Size size);

public abstract void PaintEdgeWithArrow(Graphics graphics, NodesConnectorInfo connector, Color color);

public abstract void PaintNodeInstance(Graphics graphic, NodeModel node\_info, Brush node\_brush);

public abstract void BuildGraphPath(List<NodesConnectorInfo> node\_paths);

public abstract void ScalingGraphView(int scale\_value);

}

}

Файл «EditorComponents/EditorComponentBuilder.cs»

using System;

using System.CodeDom;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Configuration;

using System.Diagnostics;

using System.Linq;

using System.Reflection;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using CSCourseWork.EditorConfiguration;

using CSCourseWork.NodesControllers;

namespace CSCourseWork.EditorComponents

{

public interface IEditorComponentBuilder<TController> : IEnumerable<EditorConfigProperty>

where TController : NodesControllers.INodesControllerWithConnectors, new()

{

public IEditorComponentBuilder<TController> AddEditorGeometry(Point position, Size size);

public IEditorComponentBuilder<TController> AddEditorProperty(string name, object value);

public IEditorComponentBuilder<TController> AddEditorConfiguration(IEditorConfigProvider provider);

public void SetLinkToNodeController(System.Type controller\_type);

public void SetLinkToForm(Form form\_link);

public EditorComponentBase<TController> BuildEditor(System.String name);

}

public sealed class EditorComponentBuilder : System.Object, IEditorComponentBuilder<NodesController>

{

public System.Type ControllerType { get; private set; }

public Form FormLink { get; private set; }

public Dictionary<string, object> Properties { get; private set; } = new();

private System.Drawing.Point EditorPosition = default(Point);

private System.Drawing.Size EditorSize = default(Size);

private NodesControllers.NodesController? controller\_instance = default(NodesController);

public NodesController? ControllerInstance { set => controller\_instance = value; }

public EditorComponentBuilder(Form form\_link, Type controller\_type) : base()

{

try { this.SetLinkToNodeController(controller\_type); }

catch (EditorComponents.EditorComponentException error)

{

throw new System.Exception("Невозможно создать Builder", error);

}

this.FormLink = form\_link;

}

public void SetLinkToForm(Form form\_link) => this.FormLink = form\_link;

public void SetLinkToNodeController(System.Type controller\_type)

{

if (!typeof(NodesController).IsAssignableFrom(controller\_type))

{

throw new EditorComponentException($"Тип не совместим с {typeof(NodesController).Name}");

}

this.ControllerType = controller\_type;

}

public IEditorComponentBuilder<NodesController> AddEditorGeometry(Point position, Size size)

{ (this.EditorPosition, this.EditorSize) = (position, size); return this; }

public IEditorComponentBuilder<NodesController> AddEditorProperty(string name, object value)

{

if (this.Properties.ContainsKey(name)) { this.Properties[name] = value; }

else { this.Properties.Add(name, value); } return this;

}

public IEditorComponentBuilder<NodesController> AddEditorConfiguration(IEditorConfigProvider provider)

{

try { foreach (var property in provider.TakeConfig()) this.AddEditorProperty(property.Name, property.Value); }

catch (EditorConfiguration.EditorConfigException error) { Console.WriteLine(error.Message); } return this;

}

public EditorComponentBase<NodesController> BuildEditor(string name)

{

if (this.controller\_instance == null)

{ this.controller\_instance = (NodesController)(Activator.CreateInstance(this.ControllerType)!); }

var editor\_instance = new EditorComponent(this.controller\_instance)

{

Location = this.EditorPosition, Size = this.EditorSize, Name = name

};

this.FormLink.Controls.Add(editor\_instance);

foreach (KeyValuePair<string, object> item in this.Properties)

{

var property\_info = editor\_instance.GetType().GetProperty(item.Key);

if (property\_info != null)

{

try { property\_info.SetValue(editor\_instance, item.Value); }

catch (System.Exception) { continue; }

}

}

return editor\_instance;

}

public IEnumerator<EditorConfigProperty> GetEnumerator()

{

foreach (KeyValuePair<string, object> property in this.Properties)

{ yield return new EditorConfigProperty(property.Key, property.Value); }

}

IEnumerator IEnumerable.GetEnumerator() => this.GetEnumerator();

}

}

Файл «EditorComponents/EditorComponentTypes.cs»

using System;

using System.CodeDom;

using System.Collections.Generic;

using System.Linq;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

namespace CSCourseWork.EditorComponents

{

[EditorConfiguration.EditorConfigTypeAttribute]

public struct EditorScale

{

public static System.Int32 MinLimit = 10, MaxLimit = 100;

private System.Int32 min = default, max = default;

[EditorConfiguration.EditorConfigPropertyAttribute("min")]

public System.Int32 Min

{ get => this.min; set => this.min = this.RangeValidate(value) ? value : MinLimit; }

[EditorConfiguration.EditorConfigPropertyAttribute("max")]

public System.Int32 Max

{ get => this.max; set => this.max = this.RangeValidate(value) ? value : MaxLimit; }

private System.Boolean RangeValidate(int value) => (value <= MaxLimit && value >= MinLimit);

public EditorScale(int min, int max) => (this.Min, this.Max) = (min, max);

public EditorScale() : this(10, 100) { }

}

[EditorConfiguration.EditorConfigTypeAttribute]

public struct EditorColor

{

private System.Int32 red = default, green = default, blue = default;

[EditorConfiguration.EditorConfigPropertyAttribute("red")]

public System.Int32 RColorValue

{ get => this.red; set => this.red = this.ColorValidate(value) ? value : 0; }

[EditorConfiguration.EditorConfigPropertyAttribute("green")]

public System.Int32 GColorValue

{ get => this.green; set => this.green = this.ColorValidate(value) ? value : 0; }

[EditorConfiguration.EditorConfigPropertyAttribute("blue")]

public System.Int32 BColorValue

{ get => this.blue; set => this.blue = this.ColorValidate(value) ? value : 0; }

public EditorColor(int r\_color, int g\_color, int b\_color)

{ (this.RColorValue, this.GColorValue, this.BColorValue) = (r\_color, g\_color, b\_color); }

public EditorColor() : this(default(int), default(int), default(int)) { }

private System.Boolean ColorValidate(int value) => (value <= 255 && value >= 0);

public static implicit operator System.Drawing.Color(EditorColor color)

=> Color.FromArgb(color.RColorValue, color.GColorValue, color.BColorValue);

}

[EditorConfiguration.EditorConfigTypeAttribute]

public struct EditorFontFamily

{

[EditorConfiguration.EditorConfigPropertyAttribute("font")]

public System.String FontFamily { get; set; } = string.Empty;

public EditorFontFamily() => this.FontFamily = new("Arial");

}

[EditorConfiguration.EditorConfigTypeAttribute]

public struct EditorTestType

{

[EditorConfiguration.EditorConfigPropertyAttribute("text")]

public System.String Text { get; set; } = string.Empty;

[EditorConfiguration.EditorConfigPropertyAttribute("range")]

public EditorComponents.EditorScale Range { get; set; } = new();

public EditorTestType(string text, int min, int max)

=> (this.Text, this.Range) = (text, new(min, max));

public EditorTestType() : this("", 0, 100) { }

}

}

Файл «EditorConfiguration/EditorConfigProvider.cs»

using System;

using System.CodeDom;

using System.Collections;

using System.Collections.Generic;

using System.Configuration;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Linq;

namespace CSCourseWork.EditorConfiguration

{

//public record class EditorConfigProperty(string Name, object Value, System.Type Type);

public sealed class EditorConfigProperty : System.Object

{

public System.String Name { get; private set; } = String.Empty;

public System.Object Value { get; set; } = default(object)!;

public EditorConfigProperty(string name, object value) : base() => (this.Name, this.Value) = (name, value);

public System.Type Type { get => this.Value!.GetType(); }

}

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Struct, AllowMultiple = false, Inherited = true)]

public sealed class EditorConfigTypeAttribute : System.Attribute { }

[AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = true)]

public sealed class EditorConfigTargetAttribute : System.Attribute { }

[AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = true)]

public sealed class EditorConfigPropertyAttribute : System.Attribute

{

public System.String Name { get; private set; } = string.Empty;

public EditorConfigPropertyAttribute(string name) : base() => this.Name = name;

}

public sealed class EditorConfigException : System.Exception

{

public System.String PropertyName { get; private set; } = string.Empty;

public EditorConfigException(string message, string name) : base(message) => this.PropertyName = name;

}

public interface IEditorConfigProvider : IEnumerable<EditorConfigProperty>

{

public List<System.String> UsingNamespaces { get; }

public System.String? ConfigFilePath { get; set; }

public List<EditorConfiguration.EditorConfigProperty> TakeConfig();

public IEditorConfigProvider PutConfigProperty(System.String name, System.Object property);

public IEditorConfigProvider PutConfigObject(System.Object @object);

protected static Configuration GetConfiguration(string? filepath)

{

if (filepath == null) return ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);

var fileMap = new ExeConfigurationFileMap() { ExeConfigFilename = filepath };

return ConfigurationManager.OpenMappedExeConfiguration(fileMap, ConfigurationUserLevel.None);

}

}

public class EditorConfigProvider : System.Object, IEditorConfigProvider

{

public System.String? ConfigFilePath { get; set; } = null;

public List<System.String> UsingNamespaces { get; private set; } = new();

public EditorConfigProvider(string? filepath) : base() => this.ConfigFilePath = filepath;

public IEditorConfigProvider PutConfigObject(object @object)

{

foreach (var property in @object.GetType().GetProperties())

{

var property\_type = property.PropertyType;

if (property.GetCustomAttribute<EditorConfigTargetAttribute>(true) == null &&

property\_type.GetCustomAttribute<EditorConfigTypeAttribute>(true) == null) continue;

this.PutConfigProperty(property.Name, property.GetValue(@object)!);

}

return this;

}

public IEditorConfigProvider PutConfigProperty(string name, object property)

{

var config = IEditorConfigProvider.GetConfiguration(this.ConfigFilePath);

var section = (EditorConfigSection)config.GetSection("editor.settings");

if (section == null) throw new EditorConfigException("Секция не найдена", string.Empty);

var property\_namespace = property.GetType().Namespace!;

var config\_has\_namespace = default(bool);

foreach (EditorNamespacesCollection.EditorNamespace config\_namespace in section.UsingNamespaces)

{

if (config\_namespace.Value == property\_namespace) { config\_has\_namespace = true; break; }

}

if (config\_has\_namespace == default(bool))

{

section.UsingNamespaces.AddNamespace(new EditorNamespacesCollection.EditorNamespace()

{ Value = property\_namespace });

}

var config\_has\_property = default(bool);

foreach (EditorConfiguration.EditorProperty config\_property in section.EditorProperties)

{

if (config\_property.Name == name) { config\_has\_property = true; break; }

}

if (config\_has\_property)

{

section.EditorProperties[name].Type = property.GetType().Name;

var changeable\_property = section.EditorProperties[name];

this.PutPropertyInConfig(ref changeable\_property, name, property);

} else

{

var new\_property = new EditorConfiguration.EditorProperty();

this.PutPropertyInConfig(ref new\_property, name, property);

section.EditorProperties.AddProperty(new\_property);

}

config.Save(ConfigurationSaveMode.Full, true); return this;

}

public List<EditorConfigProperty> TakeConfig()

{

var config = IEditorConfigProvider.GetConfiguration(this.ConfigFilePath);

var section = (EditorConfigSection)config.GetSection("editor.settings");

if (section == null) throw new EditorConfigException("Секция не найдена", string.Empty);

var result\_list = new List<EditorConfigProperty>();

this.UsingNamespaces.Clear();

foreach (EditorNamespacesCollection.EditorNamespace @namespace in section.UsingNamespaces)

{ this.UsingNamespaces.Add(@namespace.Value); }

foreach (EditorConfiguration.EditorProperty property in section.EditorProperties)

{

var property\_instance = this.GetPropertyInstance(property, section);

if (property\_instance == null) continue;

result\_list.Add(new EditorConfigProperty(property.Name, property\_instance));

}

return result\_list;

}

private System.Type? GetPropertyType(string type\_string)

{

Type? property\_type = default;

try { property\_type = Type.GetType(type\_string, true); }

catch (System.Exception)

{

for (int index = 0; index < this.UsingNamespaces.Count; index++)

{

var selected\_namespace = this.UsingNamespaces[index];

try { property\_type = Type.GetType($"{selected\_namespace}.{type\_string}", true); break; }

catch { continue; }

}

}

return property\_type;

}

private void PutPropertyInConfig(ref EditorProperty config\_property, string name, object property)

{

var property\_type = property.GetType();

config\_property.Building.ClearParams();

if (property\_type.IsPrimitive || property\_type == typeof(string))

{

config\_property.Building.Value = property.ToString()!; return;

}

else if (property\_type.GetCustomAttribute<EditorConfigTypeAttribute>(true) == null)

{ throw new EditorConfigException("Невозможно обработать свойство", name); }

foreach (var property\_item in property\_type.GetProperties())

{

var property\_item\_type = property\_item.PropertyType;

var property\_item\_attribute = property\_item

.GetCustomAttribute<EditorConfigPropertyAttribute>(true);

if (property\_item\_attribute == null) continue;

EditorPropertyBuilding.PropertyParams @param = default!;

if (property\_item\_type.IsPrimitive || property\_item\_type == typeof(string))

{

@param = new EditorPropertyBuilding.PropertyParams() { Name = property\_item\_attribute.Name,

Value = property\_item.GetValue(property)?.ToString()!, Type = property\_item\_type.Name };

}

else if (property\_item\_type.GetCustomAttribute<EditorConfigTypeAttribute>(true) != null)

{

var reference\_name = Guid.NewGuid();

this.PutConfigProperty(reference\_name.ToString(), property\_item.GetValue(property)!);

@param = new EditorPropertyBuilding.PropertyParams() { Name = property\_item\_attribute.Name,

Type = property\_item\_type.Name, Reference = reference\_name.ToString() };

}

else continue;

config\_property.Building.AddParam(@param);

}

}

private object? GetPropertyInstance(EditorProperty property, EditorConfigSection section)

{

var result\_property\_type = this.GetPropertyType(property.Type);

if (result\_property\_type == null) return null;

if (property.Building.Value != String.Empty)

{

return Convert.ChangeType(property.Building.Value, result\_property\_type);

}

object? property\_instance = Activator.CreateInstance(result\_property\_type, new object[] { });

if (property\_instance == null) return null;

foreach (EditorPropertyBuilding.PropertyParams param in property.Building)

{

var param\_type = GetPropertyType(param.Type);

if (param\_type == null) continue;

object? param\_value = default(object);

if (!param\_type.IsPrimitive && param\_type != typeof(string))

{

if (param\_type.GetCustomAttribute<EditorConfigTypeAttribute>(true) == null

|| param.Reference == string.Empty) continue;

param\_value = this.GetPropertyInstance(section.EditorProperties[param.Reference], section);

}

else param\_value = Convert.ChangeType(param.Value, param\_type);

PropertyInfo? property\_iteminfo = default;

foreach (var property\_item in result\_property\_type.GetProperties())

{

var property\_attribute = property\_item.GetCustomAttribute<EditorConfigPropertyAttribute>(true);

if (property\_attribute != null && property\_attribute.Name == param.Name)

{

property\_iteminfo = property\_item; break;

}

}

if (property\_iteminfo != null) property\_iteminfo.SetValue(property\_instance, param\_value);

}

return property\_instance;

}

public IEnumerator<EditorConfigProperty> GetEnumerator() => this.TakeConfig().GetEnumerator();

IEnumerator IEnumerable.GetEnumerator() => this.GetEnumerator();

}

}

Файл «EditorConfiguration/EditorConfigSection.cs»

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CSCourseWork.EditorConfiguration

{

public sealed class EditorConfigSection : ConfigurationSection

{

[ConfigurationProperty("namespaces", IsRequired = true)]

public EditorNamespacesCollection UsingNamespaces => (EditorNamespacesCollection)base["namespaces"];

[ConfigurationProperty("properties", IsRequired = true)]

public EditorPropertiesCollection EditorProperties => (EditorPropertiesCollection)base["properties"];

}

[ConfigurationCollection(typeof(EditorNamespace), AddItemName = "item")]

public sealed class EditorNamespacesCollection : ConfigurationElementCollection

{

public sealed class EditorNamespace : ConfigurationElement

{

[ConfigurationProperty("value", IsKey = true, IsRequired = true)]

public string Value { get => (string)base["value"]; set => base["value"] = value; }

}

public void AddNamespace(EditorNamespace @namespace) => base.BaseAdd(@namespace);

protected override ConfigurationElement CreateNewElement() => new EditorNamespace();

protected override object GetElementKey(ConfigurationElement element)

{ return ((EditorNamespace)element).Value; }

}

[ConfigurationCollection(typeof(EditorProperty), AddItemName = "property")]

public sealed class EditorPropertiesCollection : ConfigurationElementCollection

{

protected override ConfigurationElement CreateNewElement() => new EditorProperty();

protected override object GetElementKey(ConfigurationElement element)

{ return ((EditorProperty)element).Name; }

public void AddProperty(EditorProperty property) => base.BaseAdd(property);

public void RemoveProperty(string property\_name) => base.BaseRemove(property\_name);

public new EditorProperty this[string Name] => (EditorProperty)this.BaseGet(Name);

}

public sealed class EditorProperty : ConfigurationElement

{

[ConfigurationProperty("name", DefaultValue = default(string), IsKey = true, IsRequired = true)]

public string Name { get => (string)base["name"]; set => base["name"] = value; }

[ConfigurationProperty("type", DefaultValue = default(string), IsRequired = true)]

public string Type { get => (string)base["type"]; set => base["type"] = value; }

[ConfigurationProperty("building", IsRequired = true)]

public EditorPropertyBuilding Building

{

get { return (EditorPropertyBuilding)base["building"]; }

set { base["building"] = value; }

}

}

[ConfigurationCollection(typeof(EditorPropertyBuilding.PropertyParams))]

public sealed class EditorPropertyBuilding : ConfigurationElementCollection

{

[ConfigurationProperty("value", DefaultValue = default(string))]

public string Value { get => (string)base["value"]; set => base["value"] = value; }

public sealed class PropertyParams : ConfigurationElement

{

[ConfigurationProperty("name", IsRequired = true, IsKey = true)]

public string Name { get => (string)base["name"]; set => base["name"] = value; }

[ConfigurationProperty("value", DefaultValue = default(string))]

public string Value { get => (string)base["value"]; set => base["value"] = value; }

[ConfigurationProperty("type", IsRequired = true)]

public string Type { get => (string)base["type"]; set => base["type"] = value; }

[ConfigurationProperty("ref", DefaultValue = default(string))]

public string Reference { get => (string)base["ref"]; set => base["ref"] = value; }

}

public new PropertyParams this[string Name] => (PropertyParams)this.BaseGet(Name);

public void AddParam(PropertyParams @param) => base.BaseAdd(@param);

public void ClearParams() => base.BaseClear();

protected override ConfigurationElement CreateNewElement() => new PropertyParams();

protected override object GetElementKey(ConfigurationElement element)

{ return ((PropertyParams)element).Name; }

}

}

Файл «NodesControllers/INodesController.cs»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CSCourseWork.NodesControllers

{

public sealed class NodesControllerException : System.Exception

{

public NodesControllers.NodeModel? Node { get; private set; } = default;

public NodesControllerException(string message, NodeModel node) : base(message) => this.Node = node;

}

public interface INodesController : IEnumerable<NodesControllers.NodeModel>, IDisposable

{

public SortedSet<NodesControllers.NodeModel> NodesList { get; set; }

public NodesControllers.NodeModel? this[System.Int32 node\_id] { get; }

public System.Int32 NodeSize { get; set; }

public void AddNewNode(System.Int32 position\_x, System.Int32 position\_y);

public void RemoveNode(System.Int32 node\_id);

public bool NodeCollisionCheck(Point position, System.Int32 node\_id);

}

public interface INodesControllerWithConnectors : INodesController

{

public void SetNodeLinks(System.Int32 node\_id, System.Int32 required\_links\_id);

public void RemoveNodeLinks(System.Int32 node\_id, System.Int32 required\_links\_id);

public List<NodesControllers.NodesConnectorInfo> BuildNodeСonnectors();

}

}

Файл «NodesControllers/NodeModel.cs»

using System;

using System.Collections.Generic;

using System.ComponentModel;

namespace CSCourseWork.NodesControllers

{

public record NodesConnectorInfo(int EdgeId, NodeModel LeftNode, NodeModel RightNode) : System.Object;

public class NodesComparer : IComparer<NodeModel>

{

int IComparer<NodeModel>.Compare(NodeModel? x, NodeModel? y)

{

if (x == null || y == null) throw new NullReferenceException("Argument X | Y is null");

return x.NodeID.CompareTo(y.NodeID);

}

}

[System.SerializableAttribute]

public sealed class NodeModel : object, IComparable<NodeModel>, ICloneable

{

[CategoryAttribute("Настройки вершины"), ReadOnlyAttribute(true)]

[DescriptionAttribute("Значение идентификатора выбранного узла")]

public System.Int32 NodeID { get; set; } = default;

[CategoryAttribute("Настройки вершины"), ReadOnlyAttribute(true)]

[DescriptionAttribute("Коллекция значений привязанных вершин")]

[TypeConverterAttribute(typeof(List<int>))]

public List<System.Int32> NodeLinksID { get; set; } = new();

[CategoryAttribute("Настройки положения"), ReadOnlyAttribute(true)]

[DescriptionAttribute("Текущее местоположение вершины")]

public System.Drawing.Point Position { get; set; } = new(0, 0);

public NodeModel(int node\_id) : base() => this.NodeID = node\_id;

int IComparable<NodeModel>.CompareTo(NodeModel? node\_other)

{

if (node\_other == null) throw new NodesControllerException("Comparable node is null", this);

return this.NodeID.CompareTo(node\_other.NodeID);

}

public object Clone() => this.MemberwiseClone();

}

}

Файл «NodesControllers/NodesController.cs»

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using static System.Windows.Forms.LinkLabel;

namespace CSCourseWork.NodesControllers

{

public class NodesController : System.Object, INodesControllerWithConnectors

{

public SortedSet<NodeModel> NodesList { get; set; }

public int NodeSize { get; set; } = default;

public NodesController() : base() => this.NodesList = new SortedSet<NodeModel>(new NodesComparer());

public NodeModel? this[int node\_id]

{

get {

if (node\_id <= 0 || node\_id > this.NodesList.Count) return null;

return this.NodesList.ElementAt(node\_id - 1);

}

}

public bool NodeCollisionCheck(Point position, int node\_id)

{

NodeModel? select\_node = default;

try {

select\_node = this.NodesList.Where(delegate(NodeModel node\_info)

{ return node\_info.NodeID == node\_id; }).ToList()[0];

}

catch (Exception error) { MessageBox.Show(error.Message, "Ошибка"); return false; }

var node\_position = select\_node.Position;

double delta\_x = node\_position.X - position.X, delta\_y = node\_position.Y - position.Y;

bool check = Math.Sqrt(Math.Pow(delta\_x, 2) + Math.Pow(delta\_y, 2)) < NodeSize;

if (check == true) return true;

return false;

}

public void AddNewNode(int pos\_x, int pos\_y)

{

this.NodesList.ToList().ForEach(delegate (NodesControllers.NodeModel node\_info)

{

if (this.NodeCollisionCheck(new Point(pos\_x, pos\_y), node\_info.NodeID))

{ throw new NodesControllerException("Произошло наложение вершин", node\_info); }

});

var node\_builded = new NodeModel(this.NodesList.Count + 1) { Position = new Point(pos\_x, pos\_y) };

this.NodesList.Add(node\_builded);

}

public void RemoveNode(int node\_id)

{

foreach (NodesControllers.NodeModel node\_info in this.NodesList)

{

if (node\_info.NodeLinksID.Contains(node\_id)) this.RemoveNodeLinks(node\_info.NodeID, node\_id);

for (int id = 0; id < node\_info.NodeLinksID.Count; id++)

{

if (node\_info.NodeLinksID[id] >= node\_id) node\_info.NodeLinksID[id]--;

}

}

this.NodesList.RemoveWhere((node\_info) => node\_info.NodeID == node\_id);

try { for (int id = node\_id; id <= this.NodesList.Count; id++) this[id]!.NodeID--; }

catch (Exception error) { MessageBox.Show(error.Message, "Ошибка"); }

}

public void SetNodeLinks(int node\_id, int required\_link\_id)

{

NodesControllers.NodeModel? selectednode\_info = this[node\_id];

if (selectednode\_info != null && node\_id != required\_link\_id)

{

if (LinkCheck(selectednode\_info, required\_link\_id)) selectednode\_info?.NodeLinksID.Add(required\_link\_id);

}

bool LinkCheck(NodesControllers.NodeModel node\_info, int required\_id)

{ return node\_info.NodeLinksID.Where((id) => id == required\_id).ToList().Count == 0; }

}

public void RemoveNodeLinks(int node\_id, int required\_links\_id)

{

try {

this[node\_id]?.NodeLinksID.RemoveAll((id) => id == required\_links\_id);

this[required\_links\_id]?.NodeLinksID.RemoveAll((id) => id == node\_id);

}

catch (System.Exception error) { MessageBox.Show(error.Message, "Ошибка"); }

}

public List<NodesConnectorInfo> BuildNodeСonnectors()

{

var result\_list = new List<NodesConnectorInfo>();

var edge\_id = default(int);

for (int node\_id = 1; node\_id <= this.NodesList.Count; node\_id++)

{

var node\_links = this.NodesList.Where((node\_info) => node\_info.NodeID != node\_id);

node\_links.ToList().ForEach(delegate (NodeModel link)

{

foreach (var item in result\_list)

{

if (this[node\_id]!.NodeLinksID == item.RightNode.NodeLinksID &&

item.LeftNode.NodeLinksID == link.NodeLinksID) return;

}

if (link.NodeLinksID.Contains(node\_id))

result\_list.Add(new NodesConnectorInfo(edge\_id++, link, this[node\_id]!));

});

}

return result\_list;

}

public IEnumerator<NodeModel> GetEnumerator()

{ foreach (NodeModel item in this.NodesList) yield return item; }

IEnumerator IEnumerable.GetEnumerator() => this.GetEnumerator();

void IDisposable.Dispose() => this.NodesList.Clear();

}

}

Файл «Windows/AuthForm.cs»

using GraphServiceReference;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.ServiceModel;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace CSCourseWork.Windows

{

public class CheckerTokenSourse : System.Object

{

private Dictionary<string, CheckerToken> TokensList { get; set; } = new();

public CheckerTokenSourse() : base() { }

public sealed class CheckerToken : System.Object

{

public System.Boolean TokenValue { get; set; } = default!;

public System.Guid TokenID { get; set; } = default!;

}

public CheckerToken TakeToken(string name)

{

if (this.TokensList.ContainsKey(name)) return this.TokensList[name];

var token\_instance = new CheckerToken() { TokenID = Guid.NewGuid(), TokenValue = default };

this.TokensList.Add(name, token\_instance); return token\_instance;

}

public System.Boolean CheckAllToken()

{

foreach (KeyValuePair<string, CheckerToken> item in this.TokensList)

{

if (item.Value.TokenValue == default(bool)) return false;

}

return true;

}

}

internal partial class AuthForm : Form

{

protected virtual System.Int32 MinCharacter { get; private set; } = 5;

protected CheckerTokenSourse CheckerSourse { get; private set; } = new();

public AuthForm() : base()

{

this.InitializeComponent(); this.InitializeTextboxToken();

this.filepath\_button.FlatAppearance.BorderSize = default(int);

this.auth\_checkbox.CheckedChanged += new EventHandler(CheckboxCheckedChanged);

this.reg\_checkbox.CheckedChanged += new EventHandler(CheckboxCheckedChanged);

this.filepath\_button.Click += new EventHandler(FilepathButtonClick);

this.login\_button.Click += new EventHandler(LoginButtonClick);

this.register\_button.Click += new EventHandler(RegisterButtonClick);

this.reg\_emailkey\_textbox.TextChanged += new EventHandler(Reg\_emailkey\_textbox\_TextChanged);

this.reg\_password\_textbox.TextChanged += new EventHandler(Reg\_password\_textbox\_TextChanged);

this.reg\_email\_textbox.TextChanged += new EventHandler(Reg\_email\_textbox\_TextChanged);

this.reg\_name\_textbox.TextChanged += new EventHandler(Reg\_name\_textbox\_TextChanged);

this.skip\_linklabel.Click += delegate(object? sender, EventArgs args)

{

var client\_form = new ClientForm(null);

client\_form.FormClosed += (object? sender, FormClosedEventArgs args) => this.Show();

client\_form.Show(); this.Hide();

};

this.Load += new EventHandler(AuthForm\_Load);

}

private void InitializeTextboxToken()

{

this.CheckerSourse.TakeToken(this.reg\_password\_textbox.Name);

this.CheckerSourse.TakeToken(this.reg\_name\_textbox.Name);

this.CheckerSourse.TakeToken(this.reg\_email\_textbox.Name);

this.CheckerSourse.TakeToken(this.reg\_emailkey\_textbox.Name);

}

private void FormItemChecker(TextBox textbox, Panel colorpanel, bool expression)

{

var token = this.CheckerSourse.TakeToken(textbox.Name);

colorpanel.BackColor = (token.TokenValue = expression) ? Color.GreenYellow : Color.Salmon;

}

private void Reg\_password\_textbox\_TextChanged(object? sender, EventArgs args)

{

var password = this.reg\_password\_textbox.Text;

this.FormItemChecker(this.reg\_password\_textbox, this.reg\_password\_panel,

password.Length >= MinCharacter);

}

private void Reg\_name\_textbox\_TextChanged(object? sender, EventArgs args)

{

var username = this.reg\_name\_textbox.Text;

this.FormItemChecker(this.reg\_name\_textbox, this.reg\_name\_panel, username.Length >= MinCharacter);

}

private void Reg\_email\_textbox\_TextChanged(object? sender, EventArgs args)

{

var emailname = this.reg\_email\_textbox.Text;

this.FormItemChecker(this.reg\_email\_textbox, this.reg\_email\_panel,

Regex.IsMatch(emailname, @"^[\w.]+@(?:gmail|mail).(?:ru|com)$"));

}

private void Reg\_emailkey\_textbox\_TextChanged(object? sender, EventArgs args)

{

var emailkey = this.reg\_emailkey\_textbox.Text;

this.FormItemChecker(this.reg\_emailkey\_textbox, this.reg\_emailkey\_panel,

emailkey.Length >= MinCharacter);

}

private void AuthForm\_Load(object? sender, EventArgs args)

{

using (var controller = new GraphServiceReference.ProfileControllerClient())

{

this.auth\_name\_combobox.Items.Clear();

try { foreach (var item in controller.GetProfilesName()) this.auth\_name\_combobox.Items.Add(item); }

catch (FaultException<GraphServiceReference.ProfileControllerException> error)

{

MessageBox.Show(error.Detail.Message, "Ошибка"); return;

}

catch (CommunicationException error) { MessageBox.Show(error.Message, "Ошибка"); return; }

}

}

private void CheckboxCheckedChanged(object? sender, EventArgs args)

{

var checkbox\_instance = sender as CheckBox;

if (checkbox\_instance == null) return;

switch (checkbox\_instance.Name)

{

case "auth\_checkbox":

this.auth\_password\_textbox.PasswordChar

= (this.auth\_password\_textbox.PasswordChar == '\*') ? default : '\*'; break;

case "reg\_checkbox":

this.reg\_password\_textbox.PasswordChar

= (this.reg\_password\_textbox.PasswordChar == '\*') ? default : '\*'; break;

default: MessageBox.Show("Выбранный флаг не найден", "Ошибка"); return;

}

}

private void RegisterButtonClick(object? sender, EventArgs args)

{

if (!this.CheckerSourse.CheckAllToken())

{ MessageBox.Show("Неверный формат текстовых полей", "Ошибка"); return; }

System.Guid? profile\_id = default!;

using (var register = new GraphServiceReference.ProfileControllerClient())

{

try {

profile\_id = register.Registration(new ProfileData()

{

EmailName = this.reg\_email\_textbox.Text,

Password = this.reg\_password\_textbox.Text,

UserName = this.reg\_name\_textbox.Text,

EmailKey = this.reg\_emailkey\_textbox.Text,

ProjectsPath = this.reg\_filepath\_textbox.Text

});

}

catch (FaultException<GraphServiceReference.ProfileControllerException> error)

{ MessageBox.Show(error.Detail.Message, "Ошибка"); return; }

catch (CommunicationException error) { MessageBox.Show(error.Message, "Ошибка"); return; }

}

MessageBox.Show("Учётная запись была создана", "Готово");

this.auth\_name\_combobox.Items.Add(this.reg\_name\_textbox.Text);

var client\_form = new ClientForm(profile\_id.Value);

client\_form.FormClosed += (sender, args) => this.Show();

client\_form.Show(); this.Hide();

}

private void LoginButtonClick(object? sender, EventArgs args)

{

string username = this.auth\_name\_combobox.Text, password = this.auth\_password\_textbox.Text;

System.Guid? profile\_id = default!;

using (var authorize = new GraphServiceReference.ProfileControllerClient())

{

try { profile\_id = authorize.Authorization(username, password); }

catch (FaultException<GraphServiceReference.ProfileControllerException> error)

{

MessageBox.Show(error.Detail.Message, "Ошибка"); return;

}

catch (CommunicationException error) { MessageBox.Show(error.Message, "Ошибка"); return; }

}

if (profile\_id == null) { MessageBox.Show("Невозвожно зайти в профиль", "Ошибка"); return; }

var client\_form = new ClientForm(profile\_id.Value);

client\_form.FormClosed += (sender, args) => this.Show();

client\_form.Show(); this.Hide();

}

private void FilepathButtonClick(object? sender, EventArgs args)

{

using (var filedialog = new FolderBrowserDialog())

{

var result = filedialog.ShowDialog();

if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(filedialog.SelectedPath))

{

this.reg\_filepath\_textbox.Text = filedialog.SelectedPath;

}

}

}

}

Файл «Windows/ClientForm.cs»

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Configuration;

using System.Windows.Forms;

using System.Net.Http.Headers;

using System.ComponentModel;

using System.Diagnostics;

using System.ServiceModel;

using CSCourseWork.EditorComponents;

using CSCourseWork.NodesControllers;

using CSCourseWork.Connected\_Services.GraphServiceReference;

using CSCourseWork.Windows;

using CSCourseWork.EditorConfiguration;

namespace CSCourseWork

{

internal partial class ClientForm : Form

{

private EditorComponentBase<NodesController> EditorInstance { get; set; } = default!;

private System.Guid? profile\_id = default;

private System.Guid? ProfileID

{

set { this.profileid\_toolstrip\_label.Text = (this.profile\_id = value).ToString(); }

get { return this.profile\_id; }

}

private System.String? project\_name = default;

private System.String? ProjectName

{

set { this.projectname\_toolstrip\_label.Text = (this.project\_name = value); }

get { return this.project\_name; }

}

public ClientForm(System.Guid? profile\_id) : base()

{

this.InitializeComponent(); this.InstallEditorComponent("editor\_panel");

this.ProfileID = profile\_id;

this.addop\_button.Click += new EventHandler(AddOperationButtonClick);

this.deleteop\_button.Click += new EventHandler(DeleteOperationButtonClick);

this.selectop\_button.Click += new EventHandler(SelectOperationButtonClick);

this.findpath\_button.Click += new EventHandler(FindPathButtonClick);

this.reset\_button.Click += new EventHandler(ResetButtonClick);

this.edges\_listview.MouseClick += new MouseEventHandler(EdgesListViewMouseClick);

this.nodes\_treeview.AfterSelect += new TreeViewEventHandler(NodesTreeViewAfterSelect);

this.nodes\_treeview.MouseClick += new MouseEventHandler(NodesTreeViewMouseClick);

this.app\_propertygrid.PropertyValueChanged += AppPropertyGridPropertyValueChanged;

this.editor\_trackbar.ValueChanged += EditorTrackbarValueChanged;

this.editorconf\_toolstrip\_menuitem.Click += delegate (object? sender, EventArgs args)

{

var settings = new EditorSettings<NodesController>(this.EditorInstance!,

new EditorConfigProvider(null));

if (settings.ShowDialog() == DialogResult.OK) this.InstallEditorComponent("editor\_panel");

};

this.info\_toolstrip\_button.Click += delegate (object? sender, EventArgs args)

{

MessageBox.Show("БИСТ-214 Тюленев Данил; " +

"Тема: Определение пути с минимальным числом дуг на основе поиска в ширину.");

};

this.accountconf\_toolstrip\_menuitem.Click += new EventHandler(AccountConfigurationClick);

this.open\_toolstrip\_menuitem.Click += new EventHandler(OpenProjectClick);

this.save\_toolstrip\_menuitem.Click += new EventHandler(SaveProjectClick);

this.AddOperationButtonClick(this.addop\_button, EventArgs.Empty);

}

private void LoggerPrintMessage(string message) => this.status\_toolstrip\_label.Text = message;

private void EditProjectClick(object? sender, EventArgs args)

{

if (!this.ProfileID.HasValue) { MessageBox.Show("Необходимо авторизироваться", "Ошибка"); return; }

}

private void SaveProjectClick(object? sender, EventArgs args)

{

if (!this.ProfileID.HasValue) { MessageBox.Show("Необходимо авторизироваться", "Ошибка"); return; }

var project\_save = new ProjectSave(this.ProfileID.Value, this.EditorInstance.Controller)

{ ProjectName = this.ProjectName };

if (project\_save.ShowDialog() == DialogResult.Cancel) return;

this.ProjectName = project\_save.ProjectName;

this.LoggerPrintMessage("Проект сохранён");

}

private void OpenProjectClick(object? sender, EventArgs args)

{

if (!this.ProfileID.HasValue) { MessageBox.Show("Необходимо авторизироваться", "Ошибка"); return; }

var project\_open = new ProjectOpen(this.ProfileID.Value) { ProjectName = this.ProjectName };

if (project\_open.ShowDialog() == DialogResult.OK)

{

this.EditorInstance.Controller.NodesList = project\_open.NodeList!.ConvertToClientData();

this.EditorInstance.Invalidate();

this.NodeInfoListUpdate(); this.LoggerPrintMessage("Проект загружен");

}

this.ProjectName = project\_open.ProjectName;

}

private void AccountConfigurationClick(object? sender, EventArgs e)

{

if (!this.ProfileID.HasValue) { MessageBox.Show("Необходимо авторизироваться", "Ошибка"); return; }

var profile\_settings = new ProfileSettings(this.ProfileID.Value);

if(profile\_settings.ShowDialog() == DialogResult.Abort) this.Close();

this.LoggerPrintMessage("Данные учёной записи изменены");

}

private void InstallEditorComponent(string editor\_name)

{

this.pointer\_panel.Hide(); this.Controls.RemoveByKey(editor\_name);

var editor\_builder = new EditorComponentBuilder(this, typeof(NodesController));

editor\_builder.AddEditorGeometry(this.pointer\_panel.Location, this.pointer\_panel.Size)

.AddEditorConfiguration(new EditorConfigProvider(null));

if (this.EditorInstance != null) editor\_builder.ControllerInstance = this.EditorInstance.Controller;

this.LoggerPrintMessage("Настройки редактора загружены");

this.EditorInstance = editor\_builder.BuildEditor(editor\_name);

this.EditorInstance.NodeScaled += new EditorActionEventHandler(EditorInstanceNodeScaled);

this.EditorInstance.FieldClicked += new EditorActionEventHandler(EditorComponentFieldClicked);

this.EditorInstance.NodeClicked += new EditorActionEventHandler(EditorComponentNodeSelected);

}

private void EditorTrackbarValueChanged(object? sender, EventArgs args)

{

var min\_scale = this.EditorInstance.NodeScaleRange.Min;

var max\_scale = this.EditorInstance.NodeScaleRange.Max;

var adapted\_value = this.editor\_trackbar.Value \* Math.Abs(max\_scale - min\_scale) / 100 + min\_scale;

try { this.EditorInstance.ScalingGraphView(adapted\_value); }

catch (EditorComponentException error) { MessageBox.Show(error.Message, "Ошибка"); }

}

private void EditorInstanceNodeScaled(object? sender, EditorActionEventArgs args)

{

if (args.NodeScale is null) return;

var min\_scale = this.EditorInstance.NodeScaleRange.Min;

var max\_scale = this.EditorInstance.NodeScaleRange.Max;

var adapted\_value = (double)Math.Abs(args.NodeScale.Value - min\_scale) / (max\_scale - min\_scale) \* 100;

this.editor\_trackbar.Value = ((int)adapted\_value <= 0) ? 1 : (int)adapted\_value;

}

private void AppPropertyGridPropertyValueChanged(object? s, PropertyValueChangedEventArgs args)

=> (this.EditorInstance as Panel)?.Invalidate();

private void BuildContextMenu(Action<ToolStripItemClickedEventArgs> delete\_action, Point position, string[] items)

{

var context\_menu = new ContextMenuStrip() { AutoSize = true };

foreach (var item\_text in items)

{

context\_menu.Items.Add(new ToolStripButton(item\_text, Resourses.DeleteIcon) { Width = 100 });

}

context\_menu.ItemClicked += delegate (object? sender, ToolStripItemClickedEventArgs args)

{

delete\_action(args);

};

context\_menu.Show(this, position);

}

private void EdgesListViewMouseClick(object? sender, MouseEventArgs args)

{

if (this.edges\_listview.SelectedItems.Count <= 0) return;

var selected\_edge = this.edges\_listview.SelectedIndices[0];

var edge\_info = this.EditorInstance.Controller.BuildNodeСonnectors()[selected\_edge];

this.EditorInstance.BuildGraphPath(new List<NodesConnectorInfo>() { edge\_info });

if (args.Button == MouseButtons.Right)

{

this.BuildContextMenu(delegate (ToolStripItemClickedEventArgs args)

{

if (args.ClickedItem.Text != "Удалить дугу") return;

this.EditorInstance.Controller.RemoveNodeLinks(edge\_info.LeftNode.NodeID, edge\_info.RightNode.NodeID);

(this.EditorInstance as Panel)?.Invalidate();

this.NodeInfoListUpdate();

}, args.Location, new string[] { "Удалить дугу" });

}

}

private void NodesTreeViewAfterSelect(object? sender, TreeViewEventArgs args)

{

if (args.Node == null) return;

switch (args.Node.Level)

{

case 1:

this.EditorInstance.SelectedNodeID = args.Node.Index + 1;

this.app\_propertygrid.SelectedObject = this.EditorInstance.Controller[args.Node.Index + 1];

break;

case 2:

var parent\_node\_info = this.EditorInstance.Controller[args.Node.Parent.Index + 1]!;

this.EditorInstance.SelectedNodeID = parent\_node\_info.NodeLinksID[args.Node.Index];

break;

default: this.app\_propertygrid.SelectedObject = this.EditorInstance.SelectedNodeID = null; break;

}

(this.EditorInstance as Panel)?.Invalidate();

}

private void NodesTreeViewMouseClick(object? sender, MouseEventArgs args)

{

var treeview\_node = this.nodes\_treeview.SelectedNode;

if (treeview\_node != null && treeview\_node.Level != 1) return;

if (args.Button == MouseButtons.Right)

{

this.BuildContextMenu(delegate (ToolStripItemClickedEventArgs args)

{

if (args.ClickedItem.Text != "Удалить вершину") return;

this.EditorInstance.Controller.RemoveNode(treeview\_node!.Index + 1);

(this.EditorInstance as Panel)?.Invalidate();

this.NodeInfoListUpdate();

}, args.Location, new string[] { "Удалить вершину" });

}

}

private void ResetButtonClick(object? sender, EventArgs args)

{

var controller = this.EditorInstance.Controller;

while (controller.NodesList.Count > 0) controller.RemoveNode(1);

this.LoggerPrintMessage("Поверхность редактора очищена");

(this.EditorInstance as Panel)?.Invalidate();

this.NodeInfoListUpdate();

}

private void FindPathButtonClick(object? sender, EventArgs args)

{

int nodeid\_origin = default, nodeid\_target = default;

try

{

nodeid\_origin = int.Parse(this.nodeorigin\_combobox.Text.Split(' ')[1]);

nodeid\_target = int.Parse(this.nodetarget\_combobox.Text.Split(' ')[1]);

}

catch (System.Exception error) { this.LoggerPrintMessage(error.Message); return; }

try {

using var client = new GraphServiceReference.GraphCalculatorClient();

var service\_inputdata = this.EditorInstance.Controller.ConvertToServiceData();

var graph\_path = client.FindPathByBFS(nodeid\_origin, nodeid\_target, service\_inputdata);

if (graph\_path.Length > 0)

{ this.EditorInstance.BuildGraphPath(this.EditorInstance.Controller.ConvertToPath(graph\_path)); }

}

catch (FaultException<GraphServiceReference.GraphCalculatorException> error)

{

MessageBox.Show(error.Detail.Message, "Ошибка");

}

catch (CommunicationException error) { MessageBox.Show(error.Message, "Ошибка"); }

}

private void SelectOperationButtonClick(object? sender, EventArgs args)

{

this.EditorInstance.Mode = EditorModes.SelectNode;

this.LoggerPrintMessage("Выбран инструмент соединения");

this.addop\_button.Enabled = this.deleteop\_button.Enabled = true;

(sender as Button)!.Enabled = false;

}

private void DeleteOperationButtonClick(object? sender, EventArgs args)

{

this.EditorInstance.Mode = EditorModes.RemoveNode;

this.LoggerPrintMessage("Выбран инструмент удаления");

this.addop\_button.Enabled = this.selectop\_button.Enabled = true;

(sender as Button)!.Enabled = false;

}

private void AddOperationButtonClick(object? sender, EventArgs args)

{

this.EditorInstance.Mode = EditorModes.AddNode;

this.LoggerPrintMessage("Выбран инструмент добавления");

this.selectop\_button.Enabled = this.deleteop\_button.Enabled = true;

(sender as Button)!.Enabled = false;

}

private void NodeInfoListUpdate()

{

var treeview\_root = this.nodes\_treeview.Nodes[0];

treeview\_root.Nodes.Clear();

this.nodeorigin\_combobox.Items.Clear();

this.nodetarget\_combobox.Items.Clear();

for (var index = 0; index < this.EditorInstance.Controller.NodesList.Count; index++)

{

var node\_info = this.EditorInstance.Controller[index + 1]!;

treeview\_root.Nodes.Add($"Вершина ID: {node\_info.NodeID}");

this.nodeorigin\_combobox.Items.Add($"ID: {node\_info.NodeID}");

this.nodetarget\_combobox.Items.Add($"ID: {node\_info.NodeID}");

foreach (var link\_id in node\_info.NodeLinksID)

{

treeview\_root.Nodes[index].Nodes.Add($"Ссылка на Вершину ID: {link\_id}");

}

}

treeview\_root.Expand();

var edges\_list = this.EditorInstance.Controller.BuildNodeСonnectors();

this.edges\_listview.Items.Clear();

foreach (var edge\_info in edges\_list)

{

var row\_info = new string[] { (edge\_info.EdgeId + 1).ToString(),

$"Узел1 ID: {edge\_info.LeftNode.NodeID}",

$"Узел2 ID: {edge\_info.RightNode.NodeID}"

};

this.edges\_listview.Items.Add(new ListViewItem(row\_info));

}

}

private void EditorComponentFieldClicked(object? sender, EditorActionEventArgs args) => this.NodeInfoListUpdate();

private void EditorComponentNodeSelected(object? sender, EditorActionEventArgs args)

{

var treeview\_rootnode = this.nodes\_treeview.Nodes[0];

var selected\_node = this.EditorInstance.SelectedNodeID;

this.NodeInfoListUpdate();

if (selected\_node.HasValue)

{

this.nodes\_treeview.SelectedNode = treeview\_rootnode.Nodes[selected\_node.Value - 1];

this.nodes\_treeview.Select();

}

}

}

}

Файл «ServiceWorker.cs»

using ServiceLibrary.DataEncoder;

using ServiceLibrary.DataTransfer;

using ServiceLibrary.ServiceLocatorTool;

using System;

using ServiceLibrary;

using ServiceLibrary.ServiceContracts;

using System.Runtime.Serialization;

using System.Runtime.Serialization.Json;

using Microsoft.Extensions.Hosting;

using System.Threading.Tasks;

using System.Threading;

using Microsoft.Extensions.Logging;

using System.Net;

namespace HostWorker

{

public class ServiceWorker : BackgroundService

{

private ILogger<HostWorker.ServiceWorker> ServiceLogger { get; set; } = default;

public ServiceWorker(ILogger<HostWorker.ServiceWorker> service\_logger)

{

this.ServiceLogger = service\_logger;

ServiceLocator.RegistrationService<INetworkTransfer<SmptMessageEnvelope>>(new SmtpTransferFactory());

ServiceLocator.RegistrationService<IServiceDataEncoder<ProjectInfo>>(new ServiceDataEncoderFactory());

}

protected override async Task ExecuteAsync(CancellationToken stoppingToken)

{

using (var servicehost\_instance = new System.ServiceModel.ServiceHost(typeof(ServiceLibrary.GraphService)))

{

servicehost\_instance.Open();

this.ServiceLogger.LogInformation($"Сервис был запущен [{DateTime.UtcNow}]");

foreach (var address in servicehost\_instance.BaseAddresses) this.ServiceLogger.LogInformation(address.ToString());

while (stoppingToken.IsCancellationRequested != true) { await Task.Delay(200, stoppingToken); }

}

await Task.CompletedTask;

}

}

}

Файл «DataSerialization.cs»

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Linq;

using System.Runtime.Serialization;

namespace TransferDataPackage.DataSerializations

{

public sealed class VertexInfo : System.Object

{

public System.Int32 ID { get; set; } = default;

public System.Int32 PositionX { get; set; } = default;

public System.Int32 PositionY { get; set; } = default;

}

public sealed class EdgeInfo : System.Object

{

public System.Int32 RightNodeID { get; set; } = default;

public System.Int32 LeftNodeID { get; set; } = default;

}

public class TransferDataInteraction<TData> : System.Object

{

protected List<TData> TransferData { get; private set; } = default;

public TransferDataInteraction(List<TData> transfer\_data) => this.TransferData = transfer\_data;

}

public class TransferDataInstaller<TData> : TransferDataInteraction<TData>

{

public TransferDataInstaller(List<TData> transfer\_data) : base(transfer\_data) { }

public void InstallData(TData transfer\_data) => this.TransferData.Add(transfer\_data);

}

public class TransferDataExtractor<TData> : TransferDataInteraction<TData>, IEnumerable<TData>

{

public TransferDataExtractor(List<TData> data) : base(data) { }

public IEnumerable<TData> ExtractData()

{ foreach (TData transfer\_data in base.TransferData) yield return transfer\_data; }

public IEnumerator<TData> GetEnumerator() => base.TransferData.GetEnumerator();

IEnumerator IEnumerable.GetEnumerator() => this.GetEnumerator();

}

[SerializableAttribute, DataContractAttribute(Name = "TransferData", Namespace = "http://coolapp.com")]

public class DataSerializationBase : System.Object

{

[DataMemberAttribute(Name = "VertexList", Order = 0)]

private List<VertexInfo> VertexList { get; set; } = new List<VertexInfo>();

[DataMemberAttribute(Name = "EdgeList", Order = 1)]

private List<EdgeInfo> EdgeList { set; get; } = new List<EdgeInfo>();

public DataSerializationBase() : base() { }

public DataSerializationBase StateInstaller()

{

var vertexs\_installer = new TransferDataInstaller<VertexInfo>(this.VertexList);

var edges\_installer = new TransferDataInstaller<EdgeInfo>(this.EdgeList);

this.SetContextData(vertexs\_installer, edges\_installer);

return new DataSerializationBase { EdgeList = this.EdgeList, VertexList = this.VertexList };

}

public TData StateExtraction<TData>() where TData : DataSerializationBase

{

var vertexs\_extractor = new TransferDataExtractor<VertexInfo>(this.VertexList);

var edges\_extractor = new TransferDataExtractor<EdgeInfo>(this.EdgeList);

var data\_serialization = (TData)Activator.CreateInstance(typeof(TData));

data\_serialization.GetContextData(vertexs\_extractor, edges\_extractor);

return data\_serialization;

}

protected virtual void SetContextData(TransferDataInstaller<VertexInfo> vertexs,

TransferDataInstaller<EdgeInfo> edges)

{ }

protected virtual void GetContextData(TransferDataExtractor<VertexInfo> vertexs,

TransferDataExtractor<EdgeInfo> edges)

{ }

}

}

Файл «ServiceContracts/IGraphCalculator.cs»

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.Text;

using System.Xml.Serialization;

namespace ServiceLibrary.ServiceContracts

{

[ServiceContractAttribute(Name = "GraphCalculator")]

public interface IGraphCalculator

{

[OperationContractAttribute, FaultContractAttribute(typeof(GraphCalculatorException))]

System.Int32[] FindPathByBFS(int origin\_id, int target\_id, List<NodeData> node\_list);

}

[DataContractAttribute, SerializableAttribute]

public sealed class GraphCalculatorException : System.Object

{

[DataMemberAttribute]

public System.String Message { get; private set; } = default;

public GraphCalculatorException(string message) : base() => this.Message = message;

}

[DataContractAttribute, SerializableAttribute]

public sealed class NodeData : System.Object

{

[DataMemberAttribute, XmlArrayItemAttribute(typeof(int))]

public System.Int32[] NodeLinksID { get; set; } = new System.Int32[0];

[DataMemberAttribute, XmlArrayItemAttribute(typeof(int))]

public System.Int32[] NodeInboxsID { get; set; } = new System.Int32[0];

[DataMemberAttribute, XmlAttribute]

public System.Int32 NodeID { get; set; } = default(System.Int32);

[XmlIgnoreAttribute]

public System.Int32 NodePathLevel { get; set; } = default(System.Int32);

[DataMemberAttribute, XmlAttribute]

public System.Double PositionX { get; set; } = default;

[DataMemberAttribute, XmlAttribute]

public System.Double PositionY { get; set; } = default;

}

}

Файл «ServiceContracts/IProfileController.cs»

using System;

using System.Collections.Generic;

using System.Dynamic;

using System.Linq;

using System.Runtime.Remoting.Messaging;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Linq;

namespace ServiceLibrary.ServiceContracts

{

[ServiceContractAttribute(Name = "ProfileController")]

public interface IProfileController

{

[FaultContractAttribute(typeof(ProfileControllerException))]

[OperationContractAttribute]

System.Guid Registration(ServiceContracts.ProfileData profiledata);

[OperationContractAttribute]

System.Guid? Authorization(System.String username, System.String password);

[OperationContractAttribute]

System.String[] GetProfilesName();

[FaultContractAttribute(typeof(ProfileControllerException))]

[OperationContractAttribute]

void SetupProfile(System.Guid userid, ServiceContracts.ProfileData profile\_data);

[OperationContractAttribute]

ServiceContracts.ProfileData ReadProfile(System.Guid userid);

[FaultContractAttribute(typeof(ProfileControllerException))]

[OperationContractAttribute]

void DeleteProfile(System.Guid userid);

}

[DataContractAttribute]

public enum ProfileControllerAction : System.SByte

{

[EnumMemberAttribute] None, [EnumMemberAttribute] Authorization, [EnumMemberAttribute] Registration,

[EnumMemberAttribute] Setup, [EnumMemberAttribute] Delete

}

[DataContractAttribute, SerializableAttribute]

public sealed class ProfileControllerException : System.Object

{

[DataMemberAttribute]

public ServiceContracts.ProfileControllerAction Action { get; private set; } = default;

[DataMemberAttribute]

public System.String Message { get; private set; } = default;

public ProfileControllerException(string message, ProfileControllerAction action): base()

=> (this.Message, this.Action) = (message, action);

public ProfileControllerException(string message) : this(message, ProfileControllerAction.None) { }

}

[System.AttributeUsage(AttributeTargets.Property, Inherited = true)]

public sealed class ProfilePropertyAttribute : System.Attribute

{

public System.String PropertyName { get; private set; } = default;

public ProfilePropertyAttribute(string name) : base() => this.PropertyName = name;

}

[DataContractAttribute, SerializableAttribute]

public sealed class ProfileData : System.Object

{

[DataMemberAttribute]

public System.String UserName { get; set; } = default(string);

[DataMemberAttribute, ProfilePropertyAttribute("email-name")]

public System.String EmailName { get; set; } = default(string);

[DataMemberAttribute, ProfilePropertyAttribute("email-key")]

public System.String EmailKey { get; set; } = default(string);

[DataMemberAttribute, ProfilePropertyAttribute("project-path")]

public System.String ProjectsPath { get; set; } = default(string);

[DataMemberAttribute, ProfilePropertyAttribute("password")]

public System.String Password { get; set; } = default(string);

}

}

Файл «ServiceContracts/IProjectDispatcher.cs»

using Microsoft.SqlServer.Server;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Linq;

namespace ServiceLibrary.ServiceContracts

{

[ServiceContractAttribute(Name = "ProjectTransfer")]

public interface IProjectTransfer

{

[OperationContractAttribute, FaultContractAttribute(typeof(ProjectDispatcherException))]

void ExportProject(System.String export\_entity, TransferData transfer\_data);

[OperationContractAttribute, FaultContractAttribute(typeof(ProjectDispatcherException))]

void ImportProject(System.String import\_entity, TransferData transfer\_data);

}

[ServiceContractAttribute(Name = "ProjectDispatcher")]

public interface IProjectDispatcher : IEnumerable<ServiceContracts.ProjectInfo>, IProjectTransfer

{

[FaultContractAttribute(typeof(ServiceContracts.ProjectDispatcherException))]

[OperationContractAttribute]

ServiceContracts.ProjectInfo[] GetProjectsInfo();

[OperationContractAttribute]

System.Boolean SetProjectsDirectory(System.Guid project\_id);

[FaultContractAttribute(typeof(ServiceContracts.ProjectDispatcherException))]

[OperationContractAttribute]

ServiceContracts.NodeData[] TakeProjectData(System.String project\_name);

[FaultContractAttribute(typeof(ServiceContracts.ProjectDispatcherException))]

[OperationContractAttribute]

void PutProjectData(System.String project\_name, ServiceContracts.NodeData[] nodes\_field);

[FaultContractAttribute(typeof(ServiceContracts.ProjectDispatcherException))]

[OperationContractAttribute]

void CreateProject(ServiceContracts.ProjectInfo project\_info);

[FaultContractAttribute(typeof(ServiceContracts.ProjectDispatcherException))]

[OperationContractAttribute]

void UpdateProject(System.String project\_name, ServiceContracts.ProjectInfo project\_info);

[FaultContractAttribute(typeof(ServiceContracts.ProjectDispatcherException))]

[OperationContractAttribute]

void DeleteProject(System.String project\_name, System.Boolean delete\_file);

}

[DataContractAttribute, SerializableAttribute]

public class TransferData : System.Object

{

[DataMemberAttribute]

public System.String ToPath { get; set; } = default(string);

[DataMemberAttribute]

public System.String FromPath { get; set; } = default(string);

}

[DataContractAttribute, SerializableAttribute]

public sealed class ProjectDispatcherException : System.Object

{

[DataMemberAttribute]

public System.String Message { get; private set; } = default(string);

[DataMemberAttribute]

public System.String ProjectName { get; private set; } = default(string);

public ProjectDispatcherException(string message) : base() => this.Message = message;

}

[DataContractAttribute, SerializableAttribute]

public sealed class ProjectInfo : System.Object

{

[DataMemberAttribute]

public System.String ProjectName { get; set; } = default;

[DataMemberAttribute]

public System.String FileName { get; set; } = default;

[DataMemberAttribute]

public System.DateTime CreateTime { get; set; } = default;

public override System.Boolean Equals(System.Object other\_object)

{

if(other\_object is ServiceContracts.ProjectInfo other\_info)

{

return this.ProjectName == other\_info.ProjectName

|| this.FileName == other\_info.FileName;

}

return false;

}

public override System.Int32 GetHashCode() => base.GetHashCode();

}

}

Файл «ServiceTypes/GraphCalculator.cs»

using System;

using System.Collections.Generic;

using System.Linq;

using System.ServiceModel;

using System.Text;

using System.Threading.Tasks;

using ServiceLibrary.ServiceContracts;

namespace ServiceLibrary.ServiceTypes

{

[ServiceBehaviorAttribute(InstanceContextMode = InstanceContextMode.PerSession)]

public class GraphCalculator : ServiceTypes.ProjectDispatcher, ServiceContracts.IGraphCalculator

{

public int[] FindPathByBFS(int origin\_id, int target\_id, List<NodeData> node\_list)

{

if (origin\_id == target\_id)

{

var exception\_instance = new GraphCalculatorException("Совпадение начального и целевого узла");

throw new FaultException<GraphCalculatorException>(exception\_instance);

}

var node\_queue = new Queue<ServiceContracts.NodeData>(new NodeData[] { node\_list[origin\_id - 1] });

var node\_visited = new List<int>() { origin\_id };

while (node\_queue.Count > 0)

{

var current\_node = node\_queue.Dequeue();

if (current\_node.NodeID == target\_id)

{

var result = new int[current\_node.NodePathLevel + 1];

for (var index = 0; index < current\_node.NodeInboxsID.Length; index++)

{

var inbox = current\_node.NodeInboxsID[index];

if (node\_list[inbox - 1].NodePathLevel == current\_node.NodePathLevel - 1)

{

result[current\_node.NodePathLevel] = current\_node.NodeID;

current\_node = node\_list[inbox - 1];

index = -1; continue;

}

}

result[0] = origin\_id; return result;

}

foreach (var link\_id in current\_node.NodeLinksID)

{

if (node\_visited.Contains(link\_id)) continue;

node\_list[link\_id - 1].NodePathLevel = current\_node.NodePathLevel + 1;

node\_queue.Enqueue(node\_list[link\_id - 1]);

node\_visited.Add(link\_id);

}

}

return new int[] { };

}

}

}

Файл «ServiceTypes/ProfileController.cs»

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Reflection;

using System.Runtime.InteropServices.ComTypes;

using System.Runtime.Remoting.Messaging;

using System.ServiceModel;

using System.Text;

using System.Threading.Tasks;

using System.Xml;

using System.Xml.Linq;

using ServiceLibrary.DataEncoder;

using ServiceLibrary.DataTransfer;

using ServiceLibrary.ServiceContracts;

using ServiceLibrary.ServiceLocatorTool;

namespace ServiceLibrary.ServiceTypes

{

[ServiceBehaviorAttribute(InstanceContextMode = InstanceContextMode.PerSession)]

public class ProfileController : ServiceContracts.IProfileController

{

public static System.String ProjectFilename { get => ".projects"; }

protected INetworkTransfer<SmptMessageEnvelope> NetforkTransfer { get; private set; } = default;

protected virtual System.String FileName { get => ".profiles"; }

public ProfileController() : base()

{

//this.NetforkTransfer = new SmtpTransfer();

this.NetforkTransfer = ServiceLocator.GetService<INetworkTransfer<SmptMessageEnvelope>>();

}

private void SetProfileDocumentIfNotFound(string directory)

{

if (!File.Exists($"{directory}\\{this.FileName}"))

{

using (var writer = new StreamWriter(File.Create($"{directory}\\{this.FileName}")))

{ writer.WriteLine("<?xml version=\"1.0\" encoding=\"utf-8\"?><profiles></profiles>"); }

}

}

protected static XmlDocument GetProfileDocument(string filepath)

{

var exception\_instance = new ProfileControllerException("Файл профилей не найден");

if (!File.Exists(filepath))

{

throw new FaultException<ProfileControllerException>(exception\_instance);

}

var profile\_document = new XmlDocument();

using (var file\_reader = File.OpenRead(filepath))

{

var xml\_reader = XmlReader.Create(file\_reader, new XmlReaderSettings() { IgnoreWhitespace = true });

try { profile\_document.Load(xml\_reader); }

catch { new FaultException<ProfileControllerException>(exception\_instance); }

}

return profile\_document;

}

protected static void CommitProfileDocument(XmlDocument xml\_document, string filepath)

{

using (var writer = File.Open(filepath, FileMode.Create)) { xml\_document.Save(XmlWriter.Create(writer)); }

}

private void ThrowExceptionIfPathNotEmpty(System.String profile\_path)

{

var filesystem\_entries = default(string[]);

try { filesystem\_entries = Directory.GetFileSystemEntries(profile\_path); }

catch (System.Exception error) { throw new FaultException(error.Message); }

if (filesystem\_entries.Length != 0)

{

var error = new ProfileControllerException("Каталог не является пустым", ProfileControllerAction.Registration);

throw new FaultException<ProfileControllerException>(error);

}

}

private void ThrowExceptionIfEmailBad(string email\_name, string email\_key)

{

var emailcheck\_message = new DataTransfer.SmptMessageEnvelope()

{

SendingEmail = email\_name, EmailCredentials = email\_key

};

if (!this.NetforkTransfer.CheckChannel(emailcheck\_message))

{

var exception\_instance = new ProfileControllerException("Email не найден");

throw new FaultException<ProfileControllerException>(exception\_instance);

}

}

public string[] GetProfilesName()

{

var profile\_path = $"{Directory.GetCurrentDirectory()}\\{this.FileName}";

var result\_list = new List<string>();

XmlDocument xml\_document = default;

try { xml\_document = ProfileController.GetProfileDocument(profile\_path); } catch { return new string[] { }; }

foreach (XmlNode profile\_node in xml\_document.DocumentElement.ChildNodes)

{

if(profile\_node.Attributes["name"] != null) result\_list.Add(profile\_node.Attributes["name"].Value);

}

return result\_list.ToArray();

}

public System.Guid? Authorization(string username, string password)

{

var profile\_path = $"{Directory.GetCurrentDirectory()}\\{this.FileName}";

XmlDocument xml\_document = default;

try { xml\_document = ProfileController.GetProfileDocument(profile\_path); } catch { return null; }

XmlNode searching\_node = default;

foreach (XmlNode node in xml\_document.DocumentElement.ChildNodes)

{

if (node.Attributes["name"] != null && node.Attributes["id"] != null

&& node.Attributes["name"].Value == username) { searching\_node = node; break; }

}

if (searching\_node == null) return null;

for(int index = 0; index < searching\_node.ChildNodes.Count; index++)

{

var profile\_data = searching\_node.ChildNodes[index];

if (profile\_data.Name == "password" && profile\_data.Attributes["value"].Value == password

&& profile\_data.Attributes["value"] != null)

{

if (Guid.TryParse(searching\_node.Attributes["id"].Value, out var guid\_result))

{ return guid\_result; } else break;

}

}

return null;

}

public System.Guid Registration(ServiceContracts.ProfileData profile\_data)

{

foreach (var property in profile\_data.GetType().GetProperties())

{

if (property.GetValue(profile\_data) == null)

{ throw new FaultException<ProfileControllerException>(new ProfileControllerException("Данные пустые")); }

}

this.ThrowExceptionIfPathNotEmpty(profile\_data.ProjectsPath);

this.ThrowExceptionIfEmailBad(profile\_data.EmailName, profile\_data.EmailKey);

this.SetProfileDocumentIfNotFound(Directory.GetCurrentDirectory());

var xml\_document = ProfileController.GetProfileDocument($"{Directory.GetCurrentDirectory()}\\{FileName}");

foreach (XmlNode profile in xml\_document.DocumentElement.ChildNodes)

{

if (profile.Attributes["name"] != null && profile.Attributes["name"].Value == profile\_data.UserName)

{ throw new FaultException<ProfileControllerException>(new ProfileControllerException("Профиль уже создан")); }

}

var profile\_id = Guid.NewGuid();

var xml\_profile = xml\_document.CreateElement("profile");

xml\_profile.SetAttribute("name", profile\_data.UserName);

xml\_profile.SetAttribute("id", profile\_id.ToString());

foreach (var property in profile\_data.GetType().GetProperties())

{

var property\_attribute = property.GetCustomAttribute<ProfilePropertyAttribute>();

if (property\_attribute != null)

{

var profile\_property = xml\_document.CreateElement(property\_attribute.PropertyName);

profile\_property.SetAttribute("value", property.GetValue(profile\_data).ToString());

xml\_profile.AppendChild(profile\_property);

}

}

xml\_document.DocumentElement.AppendChild(xml\_profile);

using (var projects\_file = File.Create($"{profile\_data.ProjectsPath}\\{ProjectFilename}")) { }

ProfileController.CommitProfileDocument(xml\_document, $"{Directory.GetCurrentDirectory()}\\{FileName}");

return profile\_id;

}

public void DeleteProfile(System.Guid userid)

{

var xml\_document = ProfileController.GetProfileDocument($"{Directory.GetCurrentDirectory()}\\{FileName}");

XmlNode searching\_node = default;

foreach (XmlNode node in xml\_document.DocumentElement.ChildNodes)

{

if (node.Attributes["id"] != null && node.Attributes["id"].Value == userid.ToString())

{ searching\_node = node; break; }

}

if(searching\_node != null)

{

xml\_document.DocumentElement.RemoveChild(searching\_node);

ProfileController.CommitProfileDocument(xml\_document, $"{Directory.GetCurrentDirectory()}\\{FileName}");

}

}

public void SetupProfile(System.Guid userid, ServiceContracts.ProfileData profile\_data)

{

var exception\_obj = new ProfileControllerException("Профиль не найден", ProfileControllerAction.Setup);

var previous\_profiledata = this.ReadProfile(userid);

if (previous\_profiledata == null) { throw new FaultException<ProfileControllerException>(exception\_obj); }

var xml\_document = ProfileController.GetProfileDocument($"{Directory.GetCurrentDirectory()}\\{FileName}");

XmlNode searching\_node = default;

foreach (XmlNode node in xml\_document.DocumentElement.ChildNodes)

{

if (node.Attributes["id"] != null && node.Attributes["name"] != null

&& node.Attributes["id"].Value == userid.ToString()) { searching\_node = node; break; }

}

if (searching\_node == null) throw new FaultException<ProfileControllerException>(exception\_obj);

if (profile\_data.ProjectsPath != null && profile\_data.ProjectsPath != previous\_profiledata.ProjectsPath)

{

this.ThrowExceptionIfPathNotEmpty(profile\_data.ProjectsPath);

var project\_dir = new DirectoryInfo(previous\_profiledata.ProjectsPath);

foreach (var file in project\_dir.GetFiles()) { file.CopyTo($"{profile\_data.ProjectsPath}\\{file.Name}", true); }

}

if (profile\_data.UserName != null) searching\_node.Attributes["name"].Value = profile\_data.UserName;

foreach (var profile\_property in profile\_data.GetType().GetProperties())

{

var property\_attribute = profile\_property.GetCustomAttribute<ProfilePropertyAttribute>();

if (property\_attribute == null) continue;

var founded\_node = searching\_node.ChildNodes.OfType<XmlElement>()

.Where((XmlElement element) => property\_attribute.PropertyName == element.Name).ToArray()[0];

var property\_instance = profile\_property.GetValue(profile\_data);

if (founded\_node == null)

{

founded\_node = xml\_document.CreateElement(property\_attribute.PropertyName);

founded\_node.SetAttribute("value", property\_instance.ToString());

searching\_node.AppendChild(founded\_node);

}

founded\_node.SetAttribute("value", property\_instance.ToString());

}

ProfileController.CommitProfileDocument(xml\_document, $"{Directory.GetCurrentDirectory()}\\{FileName}");

}

public ServiceContracts.ProfileData ReadProfile(System.Guid userid)

{

var profile\_path = $"{Directory.GetCurrentDirectory()}\\{this.FileName}";

XmlDocument xml\_document = default;

try { xml\_document = ProfileController.GetProfileDocument(profile\_path); } catch { return null; }

XmlNode searching\_node = default;

foreach (XmlNode node in xml\_document.DocumentElement.ChildNodes)

{

if (node.Attributes["id"] != null && node.Attributes["name"] != null

&& node.Attributes["id"].Value == userid.ToString()) { searching\_node = node; break; }

}

if (searching\_node == null) return null;

var result = new ServiceContracts.ProfileData() { UserName = searching\_node.Attributes["name"].Value };

foreach (var profile\_property in result.GetType().GetProperties())

{

var property\_attribute = profile\_property.GetCustomAttribute<ProfilePropertyAttribute>();

if (property\_attribute == null) continue;

foreach (XmlNode property\_node in searching\_node.ChildNodes)

{

if (property\_node.Name != property\_attribute.PropertyName

|| property\_node.Attributes["value"] == null) continue;

profile\_property.SetValue(result, property\_node.Attributes["value"].Value);

}

}

return result;

}

}

}

Файл «ServiceTypes/ProjectDispatcher.cs»

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.ServiceModel;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using ServiceLibrary.ServiceContracts;

using System.ServiceModel.Channels;

using System.Xml.Serialization;

using ServiceLibrary.DataSerializations;

using ServiceLibrary.DataTransfer;

using System.Runtime.Serialization;

using TransferDataPackage.DataSerializations;

using System.Runtime.Serialization.Json;

using System.Text.Json;

using ServiceLibrary.DataEncoder;

using ServiceLibrary.ServiceLocatorTool;

namespace ServiceLibrary.ServiceTypes

{

[ServiceBehaviorAttribute(InstanceContextMode = InstanceContextMode.PerSession)]

public class ProjectDispatcher : ProfileController, ServiceContracts.IProjectDispatcher

{

protected System.String ProfileProjectsPath { get; set; } = default;

protected DataEncoder.IServiceDataEncoder<ProjectInfo> ProjectDataEncoder { get; set; } = default;

protected virtual System.String ProjectFileExtension { get => "graphproj"; }

protected virtual System.String ImportFileExtension { get => "json"; }

public ProjectDispatcher() : base()

{

// this.ProjectDataEncoder = new DataEncoder.ServiceDataEncoder();

this.ProjectDataEncoder = ServiceLocator.GetService<IServiceDataEncoder<ProjectInfo>>();

}

public bool SetProjectsDirectory(System.Guid project\_id)

{

var directory\_path = this.ReadProfile(project\_id)?.ProjectsPath;

var project\_path = $"{directory\_path}\\{ProfileController.ProjectFilename}";

if (!File.Exists(project\_path)) { File.Create(project\_path).Close(); return false; }

this.ProfileProjectsPath = directory\_path; return true;

}

private void ThrowExceptionIfPathNull(string message)

{

if(this.ProfileProjectsPath == null)

{ throw new FaultException<ProjectDispatcherException>(new ProjectDispatcherException(message)); }

}

private void ThrowExceptionIfProjectInfoNull(ServiceContracts.ProjectInfo project\_info)

{

foreach (var item in project\_info.GetType().GetProperties())

{

if (item.GetValue(project\_info) != null) continue;

var exception\_instance = new ProjectDispatcherException($"Пустое значение свойства: {item.Name}");

throw new FaultException<ProjectDispatcherException>(exception\_instance);

}

}

public void CreateProject(ServiceContracts.ProjectInfo project\_info)

{

this.ThrowExceptionIfPathNull("Путь к каталогу с проектами не установлен");

this.ThrowExceptionIfProjectInfoNull(project\_info);

foreach (ServiceContracts.ProjectInfo project\_data in this.GetProjectsInfo())

{

if (project\_data.Equals(project\_info))

{ throw new FaultException<ProjectDispatcherException>(new ProjectDispatcherException("Проект уже создан")); }

}

using (var writer = File.AppendText($"{this.ProfileProjectsPath}\\{ProfileController.ProjectFilename}"))

{ writer.WriteLine(this.ProjectDataEncoder.EncodeData(project\_info)); }

}

public void UpdateProject(string project\_name, ServiceContracts.ProjectInfo project\_info)

{

this.ThrowExceptionIfPathNull("Путь к каталогу с проектами не установлен");

this.ThrowExceptionIfProjectInfoNull(project\_info);

ServiceContracts.ProjectInfo found\_project = default;

foreach (ServiceContracts.ProjectInfo project\_data in this.GetProjectsInfo())

{

if (project\_data.ProjectName == project\_name) { found\_project = project\_data; }

}

if (found\_project == null)

{ throw new FaultException<ProjectDispatcherException>(new ProjectDispatcherException("Проект не найден")); }

if (File.Exists($"{this.ProfileProjectsPath}\\{found\_project.FileName}.{this.ProjectFileExtension}"))

{

string GetFullFilename(string filename)

=> $"{this.ProfileProjectsPath}\\{filename}.{this.ProjectFileExtension}";

File.Move(GetFullFilename(found\_project.FileName), GetFullFilename(project\_info.FileName));

}

this.DeleteProject(found\_project.ProjectName, found\_project.FileName != project\_info.FileName);

using (var writer = File.AppendText($"{this.ProfileProjectsPath}\\{ProfileController.ProjectFilename}"))

{ writer.WriteLine(this.ProjectDataEncoder.EncodeData(project\_info)); }

}

public void DeleteProject(string project\_name, bool delete\_file = true)

{

this.ThrowExceptionIfPathNull("Путь к каталогу с проектами не установлен");

var projects\_list = this.GetProjectsInfo();

using (var writer = File.CreateText($"{this.ProfileProjectsPath}\\{ProfileController.ProjectFilename}"))

{

foreach (ServiceContracts.ProjectInfo data in projects\_list)

{

if (data.ProjectName != project\_name) writer.WriteLine(this.ProjectDataEncoder.EncodeData(data));

else if (delete\_file != default(bool))

{ File.Delete($"{this.ProfileProjectsPath}\\{data.FileName}.{this.ProjectFileExtension}"); }

}

}

}

public ServiceContracts.ProjectInfo[] GetProjectsInfo()

{

this.ThrowExceptionIfPathNull("Путь к каталогу с проектами не установлен");

var projects\_list = new List<ServiceContracts.ProjectInfo>();

using (var reader = File.OpenText($"{this.ProfileProjectsPath}\\{ProfileController.ProjectFilename}"))

{

var reader\_line = string.Empty;

while ((reader\_line = reader.ReadLine()) != null)

{

projects\_list.Add(this.ProjectDataEncoder.DecodeData(reader\_line));

}

}

return projects\_list.ToArray();

}

public void PutProjectData(string project\_name, NodeData[] nodes\_field)

{

this.ThrowExceptionIfPathNull("Путь к каталогу с проектами не установлен");

var project\_info = this.GetProjectsInfo().Where((ProjectInfo info) => info.ProjectName == project\_name).ToArray();

if(project\_info.Length <= 0)

{ throw new FaultException<ProjectDispatcherException>(new ProjectDispatcherException("Проект не найден")); }

var xml\_formatter = new DataContractSerializer(typeof(TransferDataPackage.DataSerializations.DataSerializationBase));

using (var writer = File.Create($"{this.ProfileProjectsPath}\\{project\_info[0].FileName}.{this.ProjectFileExtension}"))

{

var transfer\_data = new DataSerializationAdapter(new NodesCollection(nodes\_field.ToList()));

xml\_formatter.WriteObject(writer, transfer\_data.StateInstaller());

}

}

public NodeData[] TakeProjectData(string project\_name)

{

this.ThrowExceptionIfPathNull("Путь к каталогу с проектами не установлен");

var project\_info = this.GetProjectsInfo().Where((ProjectInfo info) => info.ProjectName == project\_name).ToArray();

if (project\_info.Length <= 0) return null;

var project\_path = $"{this.ProfileProjectsPath}\\{project\_info[0].FileName}.{this.ProjectFileExtension}";

if (!File.Exists(project\_path))

{

this.DeleteProject(project\_name);

throw new FaultException<ProjectDispatcherException>(new ProjectDispatcherException("Файл проекта не найден"));

}

var xml\_formatter = new DataContractSerializer(typeof(TransferDataPackage.DataSerializations.DataSerializationBase));

DataSerializations.NodesCollection nodes\_collection = default;

using (var reader = File.OpenRead(project\_path))

{

var data\_deserialized = ((DataSerializationBase)xml\_formatter.ReadObject(reader))

.StateExtraction<DataSerializationAdapter>();

nodes\_collection = data\_deserialized.NodesList;

}

return nodes\_collection.NodesData.ToArray();

}

public void ExportProject(string export\_entity, TransferData transfer\_data)

{

this.ThrowExceptionIfPathNull("Путь к каталогу с проектами не установлен");

var project\_filename = default(string);

foreach (var project in this.GetProjectsInfo())

{

if (project.ProjectName == export\_entity) project\_filename = project.FileName;

}

if (project\_filename == null)

{ throw new FaultException<ProjectDispatcherException>(new ProjectDispatcherException("Проект не найден")); }

var profile\_info = this.ReadProfile(Guid.Parse(transfer\_data.FromPath));

var entity\_path = $"{this.ProfileProjectsPath}\\{project\_filename}.{this.ProjectFileExtension}";

try {

NetforkTransfer.SendData(new SmptMessageEnvelope()

{

SendingEmail = profile\_info.EmailName, EmailCredentials = profile\_info.EmailKey,

ReceivingEmail = transfer\_data.ToPath, AttachmentObject = entity\_path

},

transfer\_message: "Проект был экспортирован на вашу почту, используйте его в приложении");

}

catch (System.Exception error)

{

var exception\_instance = new ProjectDispatcherException(error.Message);

throw new FaultException<ProjectDispatcherException>(exception\_instance);

}

}

public void ImportProject(string import\_entity, TransferData transfer\_data)

{

this.ThrowExceptionIfPathNull("Путь к каталогу с проектами не установлен");

if (!File.Exists(transfer\_data.FromPath))

{ throw new FaultException<ProjectDispatcherException>(new ProjectDispatcherException("Файл импорта не найден")); }

DataSerializations.NodesCollection node\_data = default;

var file\_info = new FileInfo(transfer\_data.FromPath);

using (var reader = file\_info.OpenRead())

{

var data\_contact = typeof(TransferDataPackage.DataSerializations.DataSerializationBase);

XmlObjectSerializer data\_serializer = default;

if ($".{this.ImportFileExtension}" == file\_info.Extension) data\_serializer = new DataContractJsonSerializer(data\_contact);

else if ($".{this.ProjectFileExtension}" == file\_info.Extension) data\_serializer = new DataContractSerializer(data\_contact);

else {

var exception\_instance = new ProjectDispatcherException("Расширение не поддерживается");

throw new FaultException<ProjectDispatcherException>(exception\_instance);

}

var data\_serialization = ((DataSerializationBase)data\_serializer.ReadObject(reader))

.StateExtraction<DataSerializationAdapter>();

node\_data = data\_serialization.NodesList;

}

this.CreateProject(new ProjectInfo() { CreateTime = DateTime.Now, FileName = transfer\_data.ToPath,

ProjectName = import\_entity });

this.PutProjectData(import\_entity, node\_data.ToArray());

}

public IEnumerator<ProjectInfo> GetEnumerator()

{ foreach (var project in this.GetProjectsInfo()) yield return project; }

IEnumerator IEnumerable.GetEnumerator() => this.GetEnumerator();

}

}

Файл «ServiceLocatorTool/ServiceLocator.cs»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Runtime.InteropServices.ComTypes;

using System.Security.Policy;

using System.Text;

using System.Threading.Tasks;

namespace ServiceLibrary.ServiceLocatorTool

{

public interface IServiceLocatorCache : System.IDisposable

{

void PutServiceRecord<TService>(ServiceBase service) where TService : IServiceContract;

ServiceBase TakeServiceRecord<TService>() where TService : IServiceContract;

}

public class ServiceLocatorCache : System.Object, ServiceLocatorTool.IServiceLocatorCache

{

protected Dictionary<string, ServiceBase> CacheList { get; set; } = default;

public ServiceLocatorCache() : base() => this.CacheList = new Dictionary<string, ServiceBase>();

public void PutServiceRecord<TService>(ServiceBase service) where TService : IServiceContract

{

if (this.CacheList.ContainsKey(typeof(TService).FullName)) throw new ServiceLocatorException("Кеш не найден");

this.CacheList.Add(typeof(TService).FullName, service);

}

public ServiceBase TakeServiceRecord<TService>() where TService : IServiceContract

{

if (!this.CacheList.ContainsKey(typeof(TService).FullName)) return null;

return this.CacheList[typeof(TService).FullName];

}

public void Dispose() => this.CacheList.GetEnumerator().Dispose();

}

public sealed class ServiceLocatorException : System.Exception

{

public System.String ServiceName { get; private set; } = default;

public ServiceLocatorException(string message) : this(message, string.Empty) { }

public ServiceLocatorException(string message, string name) : base(message)

=> this.ServiceName = name;

}

public interface IServiceContract { System.Type FactoryType { get; } }

public abstract class ServiceBase : System.Object

{

public System.String ServiceName { get; private set; } = default;

public System.Guid ServiceID { get; private set; } = default;

protected ServiceBase(System.String service\_name) : base()

{ this.ServiceName = service\_name; this.ServiceID = System.Guid.NewGuid(); }

}

public abstract class ServiceFactoryBase : System.Object

{

public abstract ServiceLocatorTool.ServiceBase BuildService();

}

public static class ServiceLocator : System.Object

{

private static ServiceLocatorCache LocatorCache { get; set; } = default;

private static Dictionary<string, ServiceFactoryBase> BuildingsList { get; set; } = default;

static ServiceLocator()

{

ServiceLocator.BuildingsList = new Dictionary<string, ServiceFactoryBase>();

ServiceLocator.LocatorCache = new ServiceLocatorCache();

}

public static TService GetService<TService>() where TService: IServiceContract

{

var service\_cache = ServiceLocator.LocatorCache.TakeServiceRecord<TService>();

if (service\_cache != null && service\_cache is TService service) return service;

if (!ServiceLocator.BuildingsList.ContainsKey(typeof(TService).FullName))

{

throw new ServiceLocatorTool.ServiceLocatorException("Сервис не найден");

}

var service\_instance = ServiceLocator.BuildingsList[typeof(TService).FullName].BuildService();

ServiceLocator.LocatorCache.PutServiceRecord<TService>(service\_instance);

if (service\_instance is TService result) return result;

throw new ServiceLocatorTool.ServiceLocatorException("Несовместимый интерфейс");

}

public static void RegistrationService<TService>(ServiceFactoryBase factory)

where TService : IServiceContract

{

if (ServiceLocator.BuildingsList.ContainsKey(typeof(TService).FullName))

{ throw new ServiceLocatorTool.ServiceLocatorException("Сервис уже зарегистрирован"); }

ServiceLocator.BuildingsList.Add(typeof(TService).FullName, factory);

}

}

}

Файл «DataTransfer/NetworkTransfer.cs»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Mail;

using System.Net;

using System.Text;

using System.Threading.Tasks;

using System.Diagnostics;

using System.Security.Cryptography;

using ServiceLibrary.ServiceLocatorTool;

namespace ServiceLibrary.DataTransfer

{

public interface INetworkTransfer<TMessageEnvelope> : System.IDisposable, ServiceLocatorTool.IServiceContract

where TMessageEnvelope : class

{

void SendData(TMessageEnvelope envelope, System.String transfer\_message);

System.Boolean CheckChannel(TMessageEnvelope envelope);

}

public sealed class SmptMessageEnvelope : System.Object

{

public System.String SendingEmail { get; set; } = default(string);

public System.String EmailCredentials { get; set; } = default(string);

public System.String ReceivingEmail { get; set; } = default(string);

public System.String AttachmentObject { get; set; } = default(string);

}

public sealed class SmtpTransferFactory : ServiceLocatorTool.ServiceFactoryBase

{

public override ServiceLocatorTool.ServiceBase BuildService() => new SmtpTransfer();

}

public class SmtpTransfer : ServiceLocatorTool.ServiceBase, DataTransfer.INetworkTransfer<SmptMessageEnvelope>

{

protected virtual System.String DataTransferName { get; } = "NodeMapApp";

protected virtual System.String CheckMessage { get; } = "Подтвердение почты";

protected SmtpClient SmptTransferClient { get; private set; } = default;

public System.Type FactoryType { get => typeof(SmtpTransferFactory); }

public SmtpTransfer() : base(typeof(SmtpTransfer).Name)

{ this.SmptTransferClient = new SmtpClient("smtp.gmail.com", 587) { EnableSsl = true}; }

protected virtual MailMessage MessageBuild(string sendfrom, string sendto, string message)

{

return new MailMessage(new MailAddress(sendfrom, this.DataTransferName), new MailAddress(sendto))

{ Body = message, Subject = "NodeMap Transfer" };

}

public System.Boolean CheckChannel(SmptMessageEnvelope envelope)

{

using (var smtp\_message = this.MessageBuild(envelope.SendingEmail, envelope.SendingEmail, this.CheckMessage))

{

this.SmptTransferClient.Credentials = new NetworkCredential(envelope.SendingEmail, envelope.EmailCredentials);

try { this.SmptTransferClient.Send(smtp\_message); } catch (System.Exception) { return false; } return true;

}

}

public void SendData(SmptMessageEnvelope envelope, System.String transfer\_message)

{

using (var smtp\_message = this.MessageBuild(envelope.SendingEmail, envelope.ReceivingEmail, transfer\_message))

{

smtp\_message.Attachments.Add(new Attachment(envelope.AttachmentObject));

this.SmptTransferClient.Credentials = new NetworkCredential(envelope.SendingEmail, envelope.EmailCredentials);

this.SmptTransferClient.Send(smtp\_message);

}

}

public void Dispose() => this.SmptTransferClient.Dispose();

}

}

Файл «DataSerializations/NodesCollection.cs»

using System;

using System.CodeDom;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel.Configuration;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Serialization;

using ServiceLibrary.ServiceContracts;

using TransferDataPackage.DataSerializations;

namespace ServiceLibrary.DataSerializations

{

[type: System.SerializableAttribute, DataContractAttribute]

public class NodesCollection : System.Object, IEnumerable<ServiceContracts.NodeData>

{

[DataMemberAttribute] public List<ServiceContracts.NodeData> NodesData { get; set; }

public NodesCollection(List<NodeData> datalist) : base() => this.NodesData = datalist;

public NodesCollection() : this(new List<ServiceContracts.NodeData>()) { }

public IEnumerator<ServiceContracts.NodeData> GetEnumerator() => this.NodesData.GetEnumerator();

IEnumerator IEnumerable.GetEnumerator() => this.GetEnumerator();

}

[SerializableAttribute]

public sealed class DataSerializationAdapter : DataSerializationBase

{

[NonSerializedAttribute] private DataSerializations.NodesCollection nodes\_list = default;

public DataSerializations.NodesCollection NodesList { get => this.nodes\_list; }

public DataSerializationAdapter() : this(new NodesCollection()) { }

public DataSerializationAdapter(NodesCollection nodes\_list) : base()

{ this.nodes\_list = nodes\_list; }

protected override void SetContextData(TransferDataInstaller<VertexInfo> vertexs,

TransferDataInstaller<EdgeInfo> edges)

{

this.nodes\_list.NodesData.ForEach(delegate (NodeData node\_info)

{

vertexs.InstallData(new VertexInfo { ID = node\_info.NodeID,

PositionX = (int)node\_info.PositionX, PositionY = (int)node\_info.PositionY,

});

foreach (var links in node\_info.NodeLinksID)

{ edges.InstallData(new EdgeInfo { LeftNodeID = node\_info.NodeID, RightNodeID = links }); }

});

}

protected override void GetContextData(TransferDataExtractor<VertexInfo> vertexs,

TransferDataExtractor<EdgeInfo> edges)

{

this.nodes\_list = new DataSerializations.NodesCollection();

Console.WriteLine("GET CONTEXT DATA");

foreach (VertexInfo vertex in vertexs.ExtractData())

{

Console.WriteLine($"{vertex.ID}, {vertex.PositionX}, {vertex.PositionY}");

var node\_data = new NodeData() {

NodeID = vertex.ID, PositionX = vertex.PositionX, PositionY = vertex.PositionY,

};

IList<int> node\_lisks = new List<int>(), node\_inboxs = new List<int>();

foreach (EdgeInfo edge in edges.ExtractData())

{

if (edge.LeftNodeID == node\_data.NodeID) node\_lisks.Add(edge.RightNodeID);

if (edge.RightNodeID == node\_data.NodeID) node\_inboxs.Add(edge.LeftNodeID);

}

node\_data.NodeLinksID = node\_lisks.ToArray(); node\_data.NodeInboxsID = node\_inboxs.ToArray();

this.nodes\_list.NodesData.Add(node\_data);

}

}

}

}

Файл «DataEncoder/EncodingData.cs»

using ServiceLibrary.ServiceLocatorTool;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

namespace ServiceLibrary.DataEncoder

{

public interface IServiceDataEncoder<TServiceData> : IServiceContract where TServiceData : class

{

System.String EncodeData(TServiceData encoding\_data);

TServiceData DecodeData(System.String decoding\_data);

}

public sealed class ServiceDataEncoderFactory : ServiceLocatorTool.ServiceFactoryBase

{

public override ServiceLocatorTool.ServiceBase BuildService() => new ServiceDataEncoder();

}

public sealed class ServiceDataEncoder : ServiceLocatorTool.ServiceBase, IServiceDataEncoder<ServiceContracts.ProjectInfo>

{

public System.Type FactoryType { get => typeof(ServiceDataEncoderFactory); }

public ServiceDataEncoder() : base(typeof(ServiceDataEncoder).Name) { }

public string EncodeData(ServiceContracts.ProjectInfo encoding\_data)

{

var encoding\_text = $"{encoding\_data.ProjectName};{encoding\_data.FileName};"

$"{encoding\_data.CreateTime}";

var encoded\_data = string.Empty;

foreach (var data in Encoding.UTF8.GetBytes(encoding\_text)) encoded\_data += string.Format("{0:X} ", data);

return encoded\_data;

}

public ServiceContracts.ProjectInfo DecodeData(string decoding\_data)

{

var filtered\_data = decoding\_data.Split(' ').Where((string data) => data != String.Empty);

var encoded\_data = filtered\_data.Select((string @string)

=> byte.Parse(@string, System.Globalization.NumberStyles.HexNumber)).ToArray();

var properties\_strings = Encoding.UTF8.GetString(encoded\_data).Split(';');

return new ServiceContracts.ProjectInfo()

{

ProjectName = properties\_strings[0], FileName = properties\_strings[1],

CreateTime = DateTime.Parse(properties\_strings[2])

};

}

}

}