**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук  
Департамент программной инженерии

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Заместитель декана по учебно-методической работе  доцент департамента больших данных и информационного поиска факультета компьютерных наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И. Ю. Самоненко «\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 г. | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. В. Шилов «\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 г. |

|  |  |
| --- | --- |
| ***Подп. и дата*** |  |
| ***Инв. № дубл.*** |  |
| ***Взам. инв. №*** |  |
| ***Подп. и дата*** |  |
| ***Инв. № подл*** | RU.17701729.04.01-01 81 01-1-ЛУ |

**WEB ПРИЛОЖЕНИЕ ДЛЯ СОЗДАНИЯ ГЕНЕАЛОГИЧЕСКОГО ДРЕВА**

**Пояснительная записка**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729.04.01-01 81 01-1-ЛУ**

Исполнитель  
студент группы БПИ173  
\_\_\_\_\_\_\_\_\_\_ / Переплетчиков А. И. /  
«\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 г.

**Москва 2020**

УТВЕРЖДЕНRU.17701729.04.01-01 81 01-1-ЛУ

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. № подл | RU.17701729.04.01-01 ПЗ 01-1 |

**WEB ПРИЛОЖЕНИЕ ДЛЯ СОЗДАНИЯ ГЕНЕАЛОГИЧЕСКОГО ДРЕВА**

**Пояснительная записка**

**RU.17701729.04.01-01 81 01-1**

**Листов 17**

**Москва 2020**

**Содержание**

[**1. Введение** 3](#_Toc8332614)

[**1.1. Наименование программы** 3](#_Toc8332615)

[**1.2. Основания для разработки** 3](#_Toc8332616)

[**2. Назначение и область применения** 4](#_Toc8332617)

[**2.1. Назначение разработки** 4](#_Toc8332618)

[**2.1.1. Функциональное назначение** 4](#_Toc8332619)

[**2.1.2. Эксплуатационное назначение** 4](#_Toc8332620)

[**2.2. Область применения** 4](#_Toc8332621)

[**3. Технические характеристики** 5](#_Toc8332622)

[**3.1. Постановка задачи на разработку программы** 5](#_Toc8332623)

[**3.2. Описание и обоснование алгоритма и функционирования программы** 5](#_Toc8332624)

[**3.2.1. Описание алгоритма и функционирования программы** 5](#_Toc8332625)

[**3.2.2. Обоснование выбора алгоритма** 6](#_Toc8332626)

[**3.3. Описание и обоснование метода организации входных и выходных данных** 7](#_Toc8332627)

[**3.3.1. Описание метода организации входных и выходных данных** 7](#_Toc8332628)

[**3.3.2. Обоснование метода организации входных и выходных данных** 7](#_Toc8332629)

[**3.4. Описание и обоснование выбора состава технических и программных средств** 7](#_Toc8332630)

[**3.4.1. Состав технических и программных средств** 7](#_Toc8332631)

[**3.4.2. Обоснование выбора состава технических и программных средств** 8](#_Toc8332632)

[**4. Технико-экономические показатели** 9](#_Toc8332633)

[**4.1. Предполагаемая потребность** 9](#_Toc8332634)

[**4.2. Ориентировочная экономическая эффективность** 9](#_Toc8332635)

[**4.3. Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами** **Ошибка! Закладка не определена.**](#_Toc8332636)

[**Приложение 1. Список используемой литературы** 10](#_Toc8332637)

[**Приложение 2. Описание и функциональное назначение классов** 11](#_Toc8332638)

[**Приложение 3. Описание и функциональное назначение полей, методов и свойств** 12](#_Toc8332639)

# **1. Введение**

## **1.1. Наименование программы**

Наименование программы: «Web приложение для создания генеалогического древа».

## **1.2. Основания для разработки**

Основанием для разработки является приказ декана факультета компьютерных наук Национального исследовательского университета «Высшая школа экономики» № 2.3-02/1004-01 от 10.04.2020 «Об утверждении тем, руководителей курсовых работ студентов образовательной программы Программная инженерия факультета компьютерных наук».

Программа разрабатывается в рамках выполнения курсовой работы по теме «Web приложение для создания генеалогического древа» под руководством доцента департамента больших данных и информационного поиска ФКН И. Ю. Самоненко.

# **2. Назначение и область применения**

## **2.1. Назначение разработки**

### **2.1.1. Функциональное назначение**

Программа представляет собой web-приложение, предназначенное для создания и визуализации генеалогического дерева пользователя. Программа сохраняет введенную пользователем информацию в облачной генеалогической базе данных, а затем визуализирует ее в виде семейного древа на интернет-странице.

### **2.1.2. Эксплуатационное назначение**

Программа позволяет пользователю, проявляющему интерес к изучению и сохранению информации о своей семье, создать собственное генеалогическое древо, сохраненное в облачной базе данных. Программа позволяет отслеживать жизненный путь семьи, а также, в отличие от таблиц и документов, предоставляет информацию в более удобном для визуального считывания формате дерева.

## **2.2. Область применения**

Программа может применяться в генеалогических исследованиях, а также помочь в популяризации изучения семейной истории. Информация, которую пользователи будут вносить при помощи данного приложения, может послужить ценной для формирования общего единого архива генеалогических баз данных в России и русскоязычных странах.

# **3. Технические характеристики**

## **3.1. Постановка задачи на разработку программы**

Программа должна обеспечивать возможность выполнения следующих функций:

1. регистрация новых пользователей, проверка корректности введенных данных;
2. авторизация пользователей, проверка соответствия данных, введенных при авторизации, с данными одного из зарегистрированных пользователей;
3. визуализация генеалогического древа на странице;
4. создание нового родственника, ввод информации о нем;
5. представление информации о каждом родственнике на отдельной странице;
6. указание родственных связей;
7. редактирование информации об уже имеющемся родственнике;
8. удаление родственника из дерева, проверка допустимости удаления (отсутствие разорванных родственных связей);
9. сохранение информации в облачной базе данных Firebase.

## **3.2. Описание и обоснование алгоритма и функционирования программы**

### **3.2.1. Описание и обоснование общих принципов работы приложения**

### **3.2.1.1. Описание общих принципов работы приложения**

Программа реализует архитектуру одностраничного React-приложения, само приложение при этом не обрабатывает непосредственно бэкенд логику или базы данных. Программа осуществляет непосредственно сборку фронтенда, обращающегося к API Firebase для доступа к облачной базе данных и работы с ней в стиле REST.

### **3.2.1.2. Обоснование выбора общих принципов работы приложения**

В данном проекте для реализации была выбрана исключительно фронтенд-часть. У данного решения есть несколько обоснований:

* Бэкенд логика подобного приложения достаточно примитивна в сравнении с основной и наиболее трудоемкой фронтенд-частью: необходима простая база данных, способная хранить пользователей, родственников и родственные связи, такую БД предоставляет Firebase; в то время, как фронтенд часть подразумевает реализацию рендеринга дерева, механизма добавления новых узлов в дерево, доступ к полям ввода и т.д.
* Облегчение процесса разработки, при использовании Firebase отпадает необходимость повторно реализовывать стандартные механизмы авторизации, доступа к базе данных и прочую бэкенд-логику.

### **3.2.2. Описание способа хранения дерева в базе данных**

Firebase предоставляет облачную СУБД NoSQL класса, удобную для использования в javascript-коде, т.к. позволяет нам напрямую сохранять программные объекты в json-документы.

У каждого пользователя в базе данных есть доступ к двум коллекциям: people и families. Коллекция people хранит в себе документы с информацией о родственниках. Документ в коллекции people имеет следующую структуру:

* string id – уникальный идентификатор документа
* string firstName – имя человека
* string secondName – отчество человека
* string lastName – фамилия человека
* map birthDate – дата рождения человека
  + string day – день рождения
  + string month – месяц рождения
  + string year – год рождения
* map deathDate – дата смерти человека
  + string day – день смерти
  + string month – месяц смерти
  + string year – год смерти
* string sex – пол человека («мужской» или «женский»)
* string nativeCity – место рождения человека
* string photo – url фотографии человека
* string bio – биография человека

Как видно из структуры документа в коллекции people, непосредственно сами люди не имеют никаких родственных связей (например, ссылки на родственников с указанием типа связи родитель/супруг/ребенок). Всю информацию о связях между людьми хранит в себе коллекция families. Документ коллекции families, представляющий одну минимально допустимую семейную ячейку (супруги и их дети), так называемую «малую семью», имеют следующую структуру:

* string id – уникальный идентификатор документа, начинается на ‘\_’
* string husband – id мужа
* string wife – id жены
* string husbandFamily – id семьи родителей по мужской линии
* string wifeFamily – id семьи родителей по женской линии
* array children – массив с id детей, которые могут являться как отдельными людьми (в таком случае указывается id документа из коллекции people), так и самостоятельными семейными ячейками (в таком случае указывается id документа из коллекции families)

|  |
| --- |
|  |
| *Рис. 1. Способ организации семейной ячейки «малой семби» в базе данных* |

### **3.2.3. Описание алгоритма рендеринга дерева на странице**

React-компонент Tree, осуществляющий рендеринг дерева, получает коллекции people и families в качестве пропсов от родительского компонента, осуществляющего непосредственно выгрузку актуальных данных с сервера. Для того, чтобы отрендерить дерево на странице, программе нужно сгенерировать jsx-код, который библиотека React впоследствии превратит в HTML-разметку. Алгоритм рендеринга дерева делится на два этапа: 1) рекурсивный рендеринг отдельной семейной ветви, состоящих из идущих друг за другом связанных «малых семей», и 2) сбор всех ветвей древа в единую «большую семью», включающую всех родственников, связанных близким или шире-дальным родством. Отдельной семейной ветвью будем называть группу малых семей, связанных прямыми родственными связями, при этом:

* Либо не должно быть конфликтующих родительских ветвей, т.е. родительская семья указана лишь для одного из супругов или не указана вовсе.
* Либо, при наличии конфликтующих родительских семей (т.е. когда родители указаны и для мужа, и для жены), предпочтение отдается мужской ветви.

|  |
| --- |
|  |
| *Рис. 2. Пояснение к объединению семейных ветвей (при объединении двух веток результат слияния отходит к мужской ветке)* |

За рендеринг отдельной семейной ветви отвечает метод familyLayout, получающий на вход документ family. В зависимости от условий, в которых находится текущая семья, familyLayout выбирает одну из опций:

1. Если родителей нет ни у одного из супругов, сохраняется глубина, на которую ушла вверх наша ветвь, она пригодится для дальнейшего вертикального ранжирования ветвей;
2. Если родительских семей нет или они уже отрисованы, отрисовываем непосредственно самую семью, и рекурсивно запускаем функцию для детских семей;
3. Если есть лишь одна родительская семья, и она еще не отрисована, запускаем функцию от нее, тем самым помещая ее в текущую ветвь;
4. Если есть обе родительские семьи, и они обе не отрисованы, переходим к семье по мужской линии;
5. Если есть обе родительские семьи, и мужская уже отрисована, помещаем женскую в очередь ветвей для рендеринга и отрисовываем текущую семью в данной ветке;
6. Если есть обе родительские семьи, и женская уже отрисована, помещаем мужскую в очередь ветей для рендеринга, а текущую семью просто не отрисовываем;
7. Если есть обе родительские ветви, и они уже отрисованы, отрисовываем текущую семью только, если в данный момент находимся в мужской ветке.

Для того, чтобы собрать все ветви дерева вместе и отобразить их на странице, используется метод treeLayout:

1. Метод treeLayout объявляет локальный массив renderedFamilies, каждый элемент которого хранит jsx-код, отображающий отдельную ветку дерева.;
2. В массив renderedFamilies помещается результат работы функции familyLayout от первого (нулевого) документа из коллекции families. Функция familyLayout возвращает jsx-код для отдельной ветки семейного древа (т.е. все семьи, связанные воедино родственными связями);
3. Затем, пока глобальный массив familiesToRender, представляющий очередь из id еще не отрисованных семей, не опустеет, вызывается функция familyLayout для каждого из id в очереди. Результат работы функции для каждого id складывается в массив renderedFamilies;
4. Для каждого из элементов массива renderedFamilies высчитывается глубина, на которую ушла вверх данная ветка;
5. Все элементы массива renderedFamilies оборачиваются в блоки div со стилем вертикального отступа (margin-top), пропорционального глубине, на которую ушла ветвь.

Затем библиотека React генерирует jsx в HTML-разметку, отображаемую на странице в веб-браузере.

### **3.2.4. Описание организации HTML-разметки дерева**

Каждая семейная ячейка представляет собой div с классом family с двумя вложенными div-ами: parents и children. В div-е parents находятся карточки родителей, в div-е children – карточки детей (если они одни) или еще div-ы с классом family, если у детей есть собственные семьи.

|  |
| --- |
|  |
|  |

*Рис. 3. Организация разметки: красным отмечены div-ы с классом parents, желтым – children. Все это составляет div family.*

## **3.3. Описание и обоснование метода организации входных и выходных данных**

### **3.3.1. Описание метода организации входных и выходных данных**

#### **3.3.1.1. Описание метода организации входных данных**

Ввод данных происходит посредством стандартных html-форм: текст, число, кнопка, переключатели, поле со списком, загрузка файлов. Входные данные упаковывается в JSON документы и помещаются в коллекции NoSQL базы данных.

#### **3.3.1.2. Описание метода организации выходных данных**

Программа выводит семейное дерево в следующем виде:

* Узлы дерева – карточки с информацией о каждом родственнике (фотография, ФИО, место и дата рождения), при клике на карточку открывается страница с расширенной информацией о человеке.
* Ребра дерева – линии, отображающие родственные связи между людьми в узлах; ребра проводятся между двумя супругами и от родителя к ребенку.

Программа должна вносить изменения в БД в соответствии с введенными пользователем данными.

### **3.3.2. Обоснование метода организации входных и выходных данных**

#### **3.3.2.1. Обоснование метода организации входных данных**

Выбранный метод организации входных данных является фактическим стандартом для интернет-страницы и поддерживается всеми современными веб-браузерами.

#### **3.3.2.2. Обоснование метода организации выходных данных**

В генеалогии семейное древо является одним из наиболее простых и наглядных способов представления истории родственных групп семей. Оно дает наглядную инфографику родственных связей, отображает хронологический порядок появления новых членов семьи.

## **3.4. Описание и обоснование выбора состава технических и программных средств**

### **3.4.1. Состав технических и программных средств**

Для нормального функционирования программы требуется компьютер, оснащенный следующими техническими компонентами:

1. процессор не ниже Intel Pentium/Celeron, AMD K6/Athlon/Duron или совместимый с ними с тактовой частотой не ниже 1 ГГц;
2. 512 Мб ОЗУ или более;
3. не менее 10 Мб свободной памяти на жестком диске для хранения кэша приложения;
4. VGA-совместимые видеоадаптер и монитор с разрешением не ниже 1280х800;
5. интернет-канал со скоростью соединения не менее 1 Мбит/сек
6. клавиатура и мышь.

Для нормального функционирования программы требуется компьютер, оснащенный следующими программными компонентами:

1. операционная система Microsoft Windows XP (SP2, SP3) / Vista / 7 / 8 / 8.1 / 10;
2. любой современный десктопный веб-браузер (Google Chrome, Mozilla Firefox, Opera последних версий), поддерживающий технологии HTML5, CSS3 и JS ES6;
3. установленная программная платформа Node.js с пакетным менеджером npm, необходимые для запуска сервера на локальной машине.

### **3.4.2. Обоснование выбора состава технических и программных средств**

Приложение осуществляет сборку фронтенда, для отображения и корректного функционирования которого требуется современный веб-браузер. Все минимальные необходимые технические и программные средства, определенные для корректной работы приложения, обусловлены техническими и программными средствами браузера.

Для запуска локального сервера на компьютере пользователя необходима установленная программная платформа Node.js, включающая пакетный менеджер npm, осуществляющий установку всех необходимых для работы приложения пакетов. Сама Node.js позволяет транслировать Javascript, на котором написана программа, в машинный код.

# **4. Технико-экономические показатели**

## **4.1. Предполагаемая потребность**

На момент разработки на русскоязычном и зарубежном рынках существует несколько компьютерных программ и веб-сайтов для создания и управления генеалогическими базами данных (Gramps, FamilySpace, GenoPro, Geni.com, Родственники и т.д.). Все эти продукты предоставляют широкий функционал для работы с родословной, некоторые из них предоставляют возможность поиска родственников по открытым базам данных. Тем не менее, те из них, что продолжают поддерживаться разработчиками и сообществом, являются сильно усложненными и перегруженными дополнительными возможностями (например, для того, чтобы воспользоваться программой Gramps, обязательно необходимо потратить время на изучение инструкции к ней вплоть до изучения специальных терминов), а те, кто, как и разрабатываемое приложение, предоставляет простой и самый необходимый функционал, уже безнадежно устарели и вряд ли могут удовлетворить потребности современных пользователей (например, программа «Родственники» была выпущена в 2008 году и с тех пор претерпела несильные изменения). Помимо этого, разрабатываемое приложения является бесплатным.

## **4.2. Ориентировочная экономическая эффективность**

Программа предполагает функционирование в виде отдельного бесплатного веб-сайта с полностью бесплатным свободным доступом. Обеспечением работы бэкенда и хранения базы данных занимается компания Firebase, предоставляющая свои услуги (в необходимом для проекта объеме) бесплатно. Иные расчеты экономической эффективности не проводились.

# **Приложение 1. Список используемой литературы**

1. ГОСТ 19.101-77 Виды программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2. ГОСТ 19.301-79 Программа и методика испытаний. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3. ГОСТ 19.401-78 Текст программы. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4. ГОСТ 19.505-79 Руководство оператора. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5. ГОСТ 19.404-79 Пояснительная записка. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7. ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8. React – JavaScript-библиотека для создания пользовательских интерфейсов [Электронный ресурс] // URL: <https://ru.reactjs.org/> (Дата обращения: 11.04.2020, режим доступа: свободный).
9. Documentation | Firebase [Электронный ресурс] // URL: <https://firebase.google.com/docs> (Дата обращения: 11.04.2020, режим доступа: свободный).

# **Приложение 2. Описание и функциональное назначение классов**

Описываются только классы, созданные вручную для данного проекта. Используемые классы сторонних библиотек (Graphviz4Net) не описываются.

*Табл. 1. Описание и функциональное назначение классов*

|  |  |
| --- | --- |
| Класс | Назначение |
| ThisAddIn | Класс, автоматически создаваемый Visual Studio в проекте надстройки VSTO кода. Содержит методы и обработчики событий, использующиеся для доступа к объектной модули Visio. |
| Ribbon | Класс, представляющий ленту с инструментами в верхней панели Microsoft Visio, содержит обработчики нажатий кнопок ленты пользователем (выбрать файл, сохранить, выделить и т.д.). |
| VisioGraph | Класс, наследующий класс DotGraph (при помощи композиции), входящий в библиотеку Graphviz4Net. Объединяет в себе работу с DOT графами (чтение, парсинг, запись) и объектной моделью Visio (фигуры, соединительные линии). Выполняет роль ретранслятора из DOT графа в граф-схему на странице документа Visio. |
| GraphParser | Используется для получения инстанса объекта DotGraph путем парсинга исходной строки, представляющей код на языке DOT. |
| VisioColor | Статический класс, обеспечивающий ретрансляцию цвета из цветовой схемы программы GraphViz, представленный в виде строки-наименования цвета на английском языке, в цвет, доступный в объектной модели Visio, представленный в виде RGB. |

# **Приложение 3. Описание и функциональное назначение полей, методов и свойств**

*Табл. 2. Описание и функциональное назначение полей и свойств класса ThisAddIn*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Модификатор доступа | Тип | Назначение |
| graphs | private | Dictionary<Visio.Page, VisioGraph> | Хранит пары значений типа страница документа Visio – граф, соответствующий данной странице |

*Табл. 3. Описание и функциональное назначение методов класса ThisAddIn*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Модификатор доступа | Тип | Аргументы | Назначение |
| ShowGraph | public | void | string | Метод, отображающий граф на странице документа Visio |
| ExportGraph | public | void | string | Метод, экспортирующий граф со страницы документа в DOT файл |
| Layout | public | void | - | Производит перепланировку графа |
| Invert | public | void | - | Инвертирует выбранные ребра |
| Select | public | void | int | Выделяет вершины графа в зависимости от ключа |
| RemovePageIfError | public | void | - | Если в ходе импорта графа возникла ошибка, данный метод удаляет страницу, соответствующую ему |
| ErrorMessage | public | void | - | Выводит сообщение об ошибке в новое окно |
| DeleteGraph | private | void | Visio.Page | Удаляет граф из словаря, если была удалена данная страница |
| AddEdge | private | void | Visio.Connects | Данный метод вызывается при срабатывании события «Добавить соединение» и добавляет ребро в граф |
| DeleteShape | private | void | Visio.Shape | Данный метод вызывается при срабатывании события «Удалить фигуру» и удаляет ребро или вершину из графа |
| DeleteEdge | private | void | Visio.Connects | Данный метод вызывается при срабатывании события «Удалить соединение» и удаляет ребро из графа |
| ChangeText | private | void | Visio.Shape | Данный метод вызывается при срабатывании события «Изменить текст» |
| AddShape | private | void | Visio.Shape | Данный метод вызывается при срабатывании события «Добавить фигуру» и добавляет вершину в граф |

*Табл. 4. Описание и функциональное назначение методов класса Ribbon*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Модификатор доступа | Тип | Аргументы | Назначение |
| openFileButton\_Click | private | void | object, RibbonControl | Обработчик нажатия на кнопку «Выбрать файл» |
| exportGraphButton\_Click | private | void | object, RibbonControl | Обработчик нажатия на кнопку «Сохранить изменения» |
| selectAllNodesButton\_Click | private | void | object, RibbonControl | Обработчик нажатия на кнопку «Выделить все вершины» |
| selectConnectedNodeButton\_Click | private | void | object, RibbonControl | Обработчик нажатия на кнопку «Выделить соединенные вершины» |
| selectNonConnectedNodesButton\_Click | private | void | object, RibbonControl | Обработчик нажатия на кнопку «Выделить несоединенные вершины» |
| selectAllEdges\_Click | private | void | object, RibbonControl | Обработчик нажатия на кнопку «Выделить все ребра» |
| invertButton\_Click | private | void | object, RibbonControl | Обработчик нажатия на кнопку «Инвертировать ребро» |
| layoutButton\_Click | private | void | object, RibbonControl | Обработчик нажатия на кнопку «Планировка» |

*Табл. 5. Описание и функциональное назначение методов класса GraphParser*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Модификатор доступа | Тип | Аргументы | Назначение |
| ParseGraphData | public | DotGraph<string> | string | Возвращает объект типа DotGraph путем парсинга кода, записанного на языке DOT, при помощи ANTLR парсера, входящего в библиотеку Graphviz4Net |

*Табл. 6. Описание и функциональное назначение методов класса VisioColor*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Модификатор доступа | Тип | Аргументы | Назначение |
| ColorToRGB | public | string | string | Visio использует только цвета, представленные в виде RGB схемы, DOT – в виде строкового наименования цвета на английском языке; Данная функция предназначена для перевода цвета из одной схемы в другую |

*Табл. 7. Описание и функциональное назначение полей и свойств класса VisioGraph*

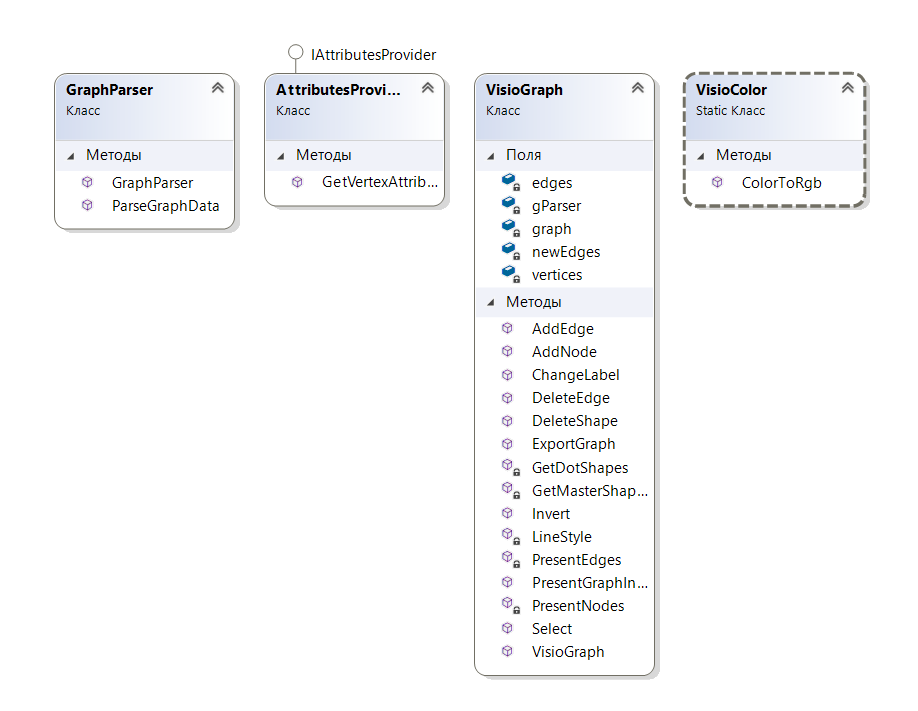
|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Модификатор доступа | Тип | Назначение |
| gParser | private | GraphParser | Объект-парсер, используемый для создания объекта типа DotGraph из кода данного графа, записанного на языке Dot |
| graph | private | DotGraph<string> | Граф (объект библиотеки Graphviz4Net) |
| vertices | private | Dictionary <DotVertex<string>, Visio.Shape> | Хранит пары типа вершина графа – фигура на странице, ей соответствующая |
| edges | private | Dictionary <DotEdge<string>, Visio.Shape> | Хранит пары типа ребро графа – фигура на странице, ему соответствующая |
| newEdges | private | Dictionary <Visio.Shape, List<Visio.Shape>> | При добавлении нового соединения хранит тут новые ребра, не имеющие одного из концов |

*Табл. 8. Описание и функциональное назначение методов класса VisioGraph*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Модификатор доступа | Тип | Аргументы | Назначение |
| PresentGraphInVisio | public | void | Visio.Documents, Visio.Page | Отрисовывает граф на странице документа Visio |
| PresentNodes | private | void | Visio.Documents, Visio.Page | Отрисовывает вершины графа на странице документа Visio |
| PresentEdges | private | void | Visio.Documents, Visio.Page | Отрисовывает ребра графа на странице документа Visio |
| Invert | public | void | Visio.Window | Инвертирует выбранное ребро |
| Select | public | void | int, Visio.Window | В зависимости от ключа, передаваемого в качестве целочисленного параметра, выделяет все, соединенные или несоединенные вершины |
| AddNode | public | void | Visio.Shape | При добавлении фигуры на страницу добавляет вершину в объект graph |
| ChangeLabel | public | void | Visio.Shape | При изменении текста фигуры меняет соответствующий атрибут у вершины или ребра графа |
| AddEdge | public | void | Visio.Connects | При добавлении нового соединения добавляет ребро в объект graph |
| DeleteEdge | public | void | Visio.Connects | При удалении соединения двух фигур удаляет соответствующее ребро графа |
| DeleteShape | public | void | Visio.Shape | При удалении фигуры удаляет вершину или ребро у графа |
| GetDotShapes | private | Dictionary<string, string> | - | Каждой строке названия фигуры из набора Visio сопоставляет название фигуры из языка DOT |
| DotFromVsd | private | List <string> | List<string> | Переводит схемы со страницы Visio в понятный языку DOT формат, используется для декомпозиции методов, использующих перевод из одной системы в другую |
| GetMasterShapes | private | Dictionary<string, Visio.Master> | Visio.Documents | Каждой строке названия фигуры из языка DOT сопоставляет Master-фигуру из доступных в Visio |
| LineStyle | private | string | string | Для одного из стилей линий языка DOT возвращает числовое представление атрибута фигуры в Visio |
| ExportGraph | public | void | string | Записывает объект graph в файл по указанному адресу |

# **Приложение 4. Диаграмма классов библиотеки GraphLibrary**

*Рис. 1. Диаграмма классов библиотеки GraphLibrary*



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Изм. | Номера листов | | | | Всего листов (страниц) в документе | № документа | Входя- щий № сопро- водит. докум. и дата | Под- пись | Дата |
| изменен- ных | заменен- ных | новых | аннулиро- ванных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |