Министерство науки и образования Российской федерации

ФБГОУ ВО «Кубанский государственный технологический университет»
(ФГБОУ ВО «КубГТУ»)

Институт компьютерных систем и информационной безопасности

Кафедра информационных систем и программирования

Направление подготовки/специальность 09.03.04 Программная инженерия
(код и наименование направления подготовки/специальности)

Профиль/специализация Проектирование и разработка программного обеспечения
(наименование профиля/специализации)

КУРСОВОЙ ПРОЕКТ

по дисциплине Технологии разработки программного обеспечения
(наименование дисциплины)

на тему: «Генеалогическое дерево.»
(тема курсовой работы)

Выполнил(-а) студент(-ка) Солдатиков Данил Андреевич курса 2 группы 19-КБ-ПР1
(фамилия, имя, отчество)

Допущена к защите 15.12.2020
(дата)

Руководитель (нормоконтролер) работы доц. 17.12.2020 Попова О.Б.
(должность, подпись, дата)

Защищена 17.12.2020 Оценка хор
(дата)

Члены комиссии: Кушнир Н.В.

Тотухов К.Е.

(должность, подпись, дата, расшифровка подписи)

Краснодар
2020 г.

Институт компьютерных систем и информационной безопасности

Кафедра информационных систем и программирования
Направление подготовки/специальность 09.03.04 Программная инженерия
(код и наименование направления подготовки/специальности)

Профиль/специализация Проектирование и разработка программного обеспечения
(наименование профиля/специализации)

УТВЕРЖДАЮ
Зав. кафедрой доц. Янаева М.В.
«08» 09 2020 г.

ЗАДАНИЕ
на курсовой проект

Студенту Солдатикову Данилу Андреевичу курса 2 группы 19-КБ-ПР1
Тема работы: «Генеалогическое дерево.»
(утверждена указанием директора института №____ от _____20__ г.)

План работы:
1. Изучение предметной области
2. Проектирование интерфейса системы.
3. Создание программного обеспечения

Объем работы:
а) пояснительная записка 36 с.
б) иллюстративная часть _____ лист(-ов)

Рекомендуемая литература:
1. Роберт А. Максимчук. UML для простых смертных.
2. Йордон, Эдвард. Объектно-ориентированный анализ и проектирование систем/ Эдвард Йордон, Карл Аргила. – М.: ЛОРИ, 2014. – 264 с.

Срок выполнения: с «01» 09 по «30» 12 2020 г.

Срок защиты: «21» 12 2020 г.

Дата выдачи задания: «08» 09 2020 г.

Дата сдачи работы на кафедру: «21» 12 2020 г.

Руководитель работы _____ Попова О.Б.
(должность, подпись)

Задание принял студент _____ Солдатиков Д.А.
(подпись)

Реферат

Курсовая работа: 36 страница, 25 рисунков, 10 используемых источников.

ПРОЕКТИРОВАНИЕ, ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ ПОСЛЕДОВАТЕЛЬНОСТИ, ACCES, КЛАСС, ГЕНЕАЛОГИЧЕСКОЕ ДЕРЕВО, МОДЕЛЬ, BPMN, FURPS+, EPC, ГАНТ, IDEF0, ДИАГРАММЫ, DFD.

Объектом исследования является программное обеспечение и создания генеалогического дерева, которое способно, по запросу пользователь, выводить данные из базы данных, выполнять поиск, изменять, дополнять, удалять значения по желанию.

Цель работы состоит в разработке приложения «Генеалогическое дерево» с использованием диаграмм разного вида, в полной мере описывающих приложение и взаимодействие системы на всех этапах работы.

В результате были получены диаграммы, в полной мере описывающие работу генеалогического дерева. К таким диаграммам относятся: диаграмма Ганта, DFD-диаграмма, IDEF0-диаграмма, UML-диаграмма.

Содержание

Введение

Генеалогическое древо – это схема, в которой отображены родственные связи, прямых предков и потомков. В приложении также будет указана дополнительная информация о человеке: братья и сестры, дата рождения, дата смерти и другое.

При разработке приложения был использован язык программирования высокого уровня C#, а также других языков для определённых задач, для обеспечения хранения данный была использована реляционная система управления базами данных Microsoft Access.

Таким образом, данное приложение обеспечивает удобное использование и хранение данных.

Построение диаграмм Ганта было выполнено в приложении LibraProject.

Вся работа выполнена следующим образом:

Солдатиков Д.А. выполняет построение диаграмм Ганта, IDEF1x, IDEF0 и IDEF3 и перерасчёт трудозатрат, а также выполняет построение всех UML диаграмм, а так же реализацию функционала создаваемой программы и её тестирование и описание в руководстве пользователя.

1 Формулировка задачи

Задачей данного курсового проекта является разработка программного обеспечения системы для создания и изменения генеалогических деревьев. Пользователи системы могут осуществлять поиск информации по дереву:

− находить для указанного члена семьи его детей;

− находить для указанного члена семьи его родителей;

− находить для указанной персоны братьев и сестер, если таковые есть;

− получать список всех предков персоны;

− получать список всех потомков персоны;

− получать список всех родственников персоны;

Приложение по нажатию кнопки «БД» предоставляет абсолютно всю информацию, хранящуюся в базе данных. При в по нажатию кнопки «Id» пользователь может либо найти информацию о человеке или удалить все данные о нем в БД, это осуществляется при нажатии кнопок «Найти» и «Удалить». Id пользователя является уникальным, после удаления следующий пользователь получает новый и уникальный идентификатор. При нажатии кнопки «Добавить» открывается новое окно приложения, где необходимо ввести данные о новом пользователи. При нажатии кнопки «Изменить» открывается новое окно приложения, где необходимо ввести Id пользователя, данные которого требуется изменить.

2 Диаграмма Ганта

Диаграмма Ганта — это тип столбчатых диаграмм, который используется для иллюстрации плана, графика работ по какому-либо проекту. Является одним из методов планирования проектов. Находятся за авторством Генри Ганта. Выглядит она следующим образом горизонтальные полосы, расположенные между двумя осями: списком задач по вертикали и датами по горизонтали.

На диаграмме видны не только сами задачи, но и их последовательность и связность между собой.

Ключевым понятием диаграммы Ганта является «веха» — метка значимого момента в ходе выполнения работ, общая граница двух или более задач. Вехи позволяют наглядно отобразить необходимость синхронизации, последовательности в выполнении различных работ. Сдвиг вехи приводит к сдвигу всего проекта. Поэтому диаграмма Ганта не является графиком работ. Кроме того, диаграмма Ганта не отображает значимости или ресурсоемкости работ, не отображает области действия. Для крупных проектов диаграмма Ганта становится чрезмерно тяжеловесной и теряет всякую наглядность.»

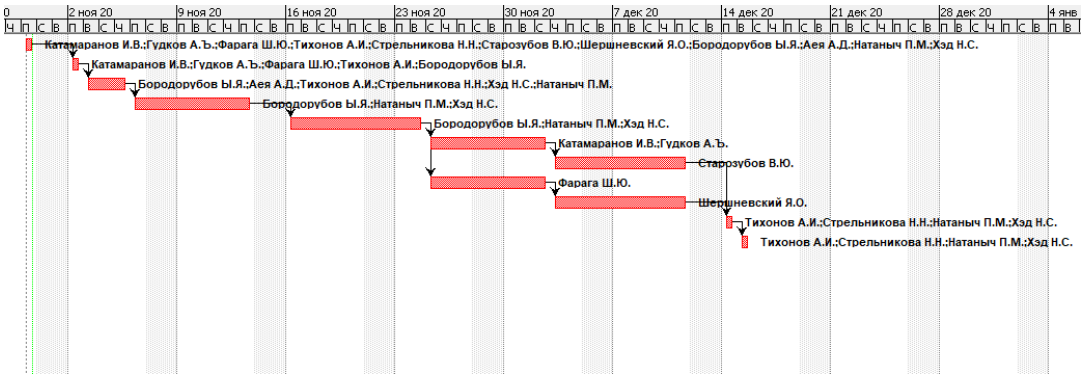


Рисунок 1 - Диаграмма Ганта для распределения работников по этапам проекта «Генеалогическое дерево»

3 Создание модели As-Is в стандарте IDEF0

Чтобы оценить возможности, разрабатываемой системы необходимо построить её базовую модель, которую можно представить в виде диаграммы As-Is.

Диаграмма As-Is – это функциональная модель , перевод «как есть» хорошо описывает суть этой модели, она позволяет узнать, где именно находятся слабые места, в чём будут состоять преимущества и недостатки, протекающих в ней бизнес-процессов относительно конкурентов. Применение данной модели позволит чётко зафиксировать какие информационные объекты принимают участие в жизненном цикле системы, какая информация будет поступать на вход и что будет получаться на выходе. Модель As-Is, строится с использованием нотации IDEF0.

IDEF0 – это графическая нотация, предназначенная для описания бизнес-процессов. Система, описываемая в данной нотации, проходит через декомпозицию или, иными словами, разбиение на взаимосвязанные функции. Для каждой функции существует правило сторон:

– стрелкой слева обозначаются входные данные;

– стрелкой сверху – управление;

– стрелкой справа – выходные данные;

– стрелкой снизу – механизм.

Учитывая всё вышеперечисленное критерии на рисунке 1 была составлена модель As-Is проекта «Генеалогическое дерево».

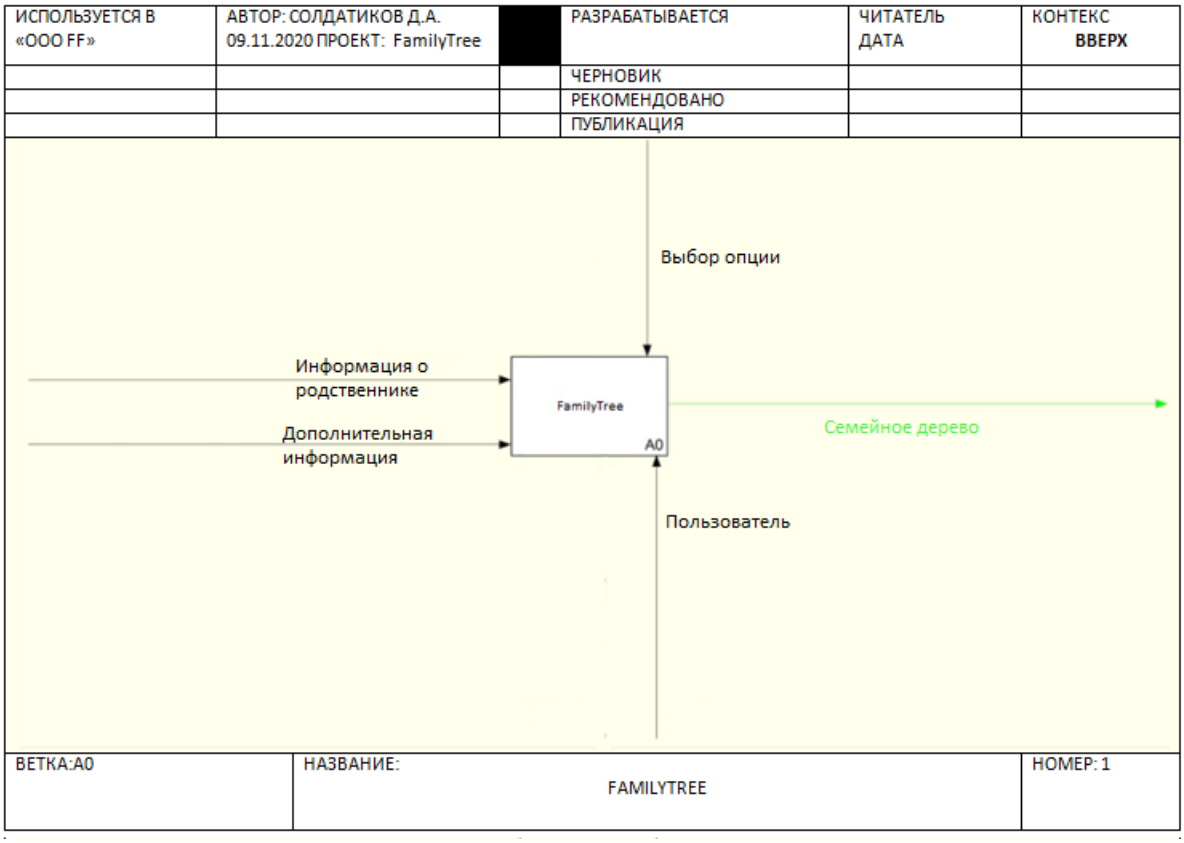


Рисунок 2 – Диаграмма IDEF0 уровня A-0.

Полученная модель системы может быть представлена в более подробном виде путём разбиения на большее количество составных элементов.

На рисунке 3 можно видеть модель кода «Генеалогическое дерево» после декомпозиции.

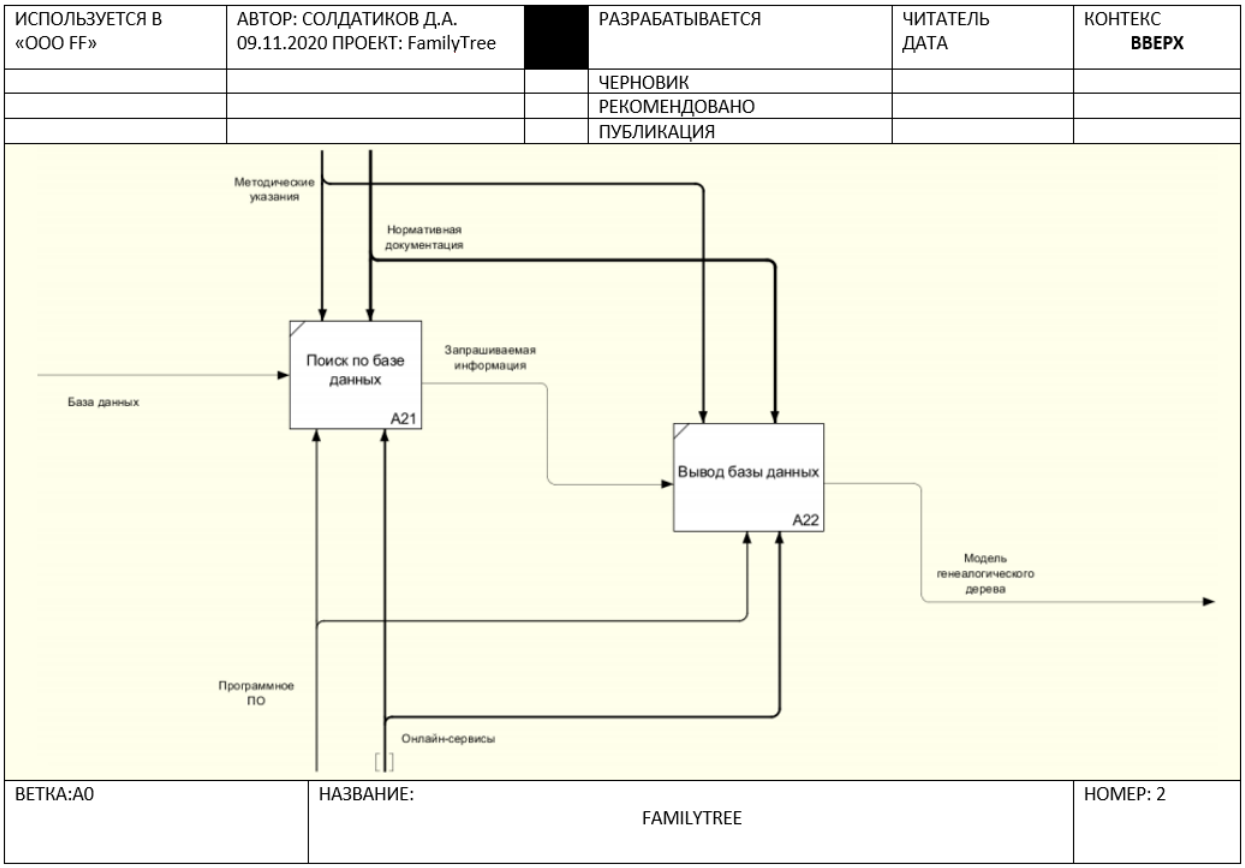
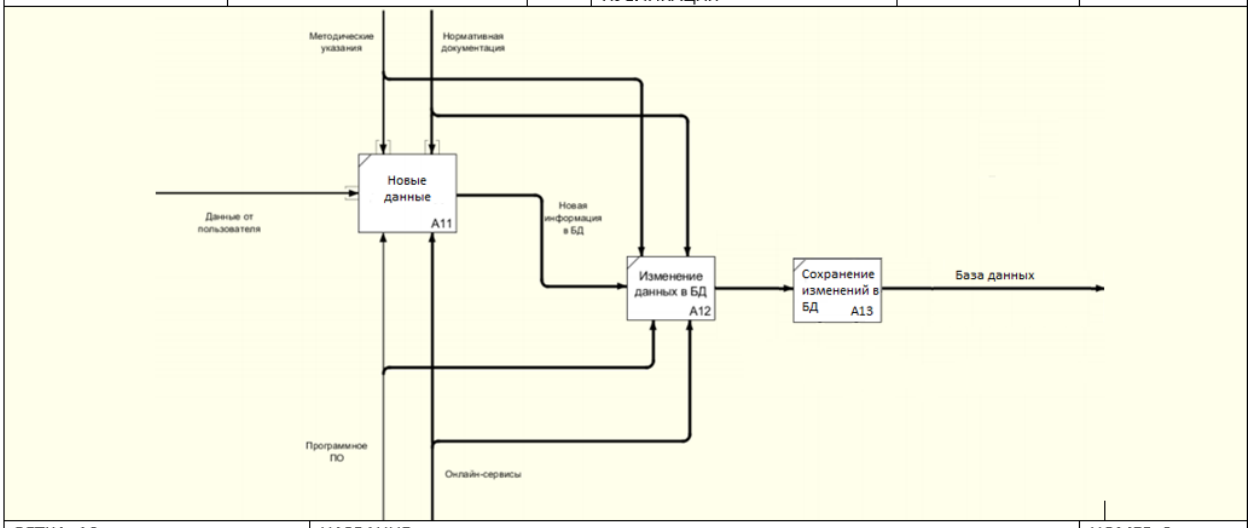


Рисунок 3 – Диаграмма IDEF0 уровня A0.

Для удобного представления работы программы и ее комплектующих данную схему можно рассмотреть более подробно, для этого необходимо разбить на более простые процессы. Данные декомпозиции будут представлены на рисунках 4 и 5.

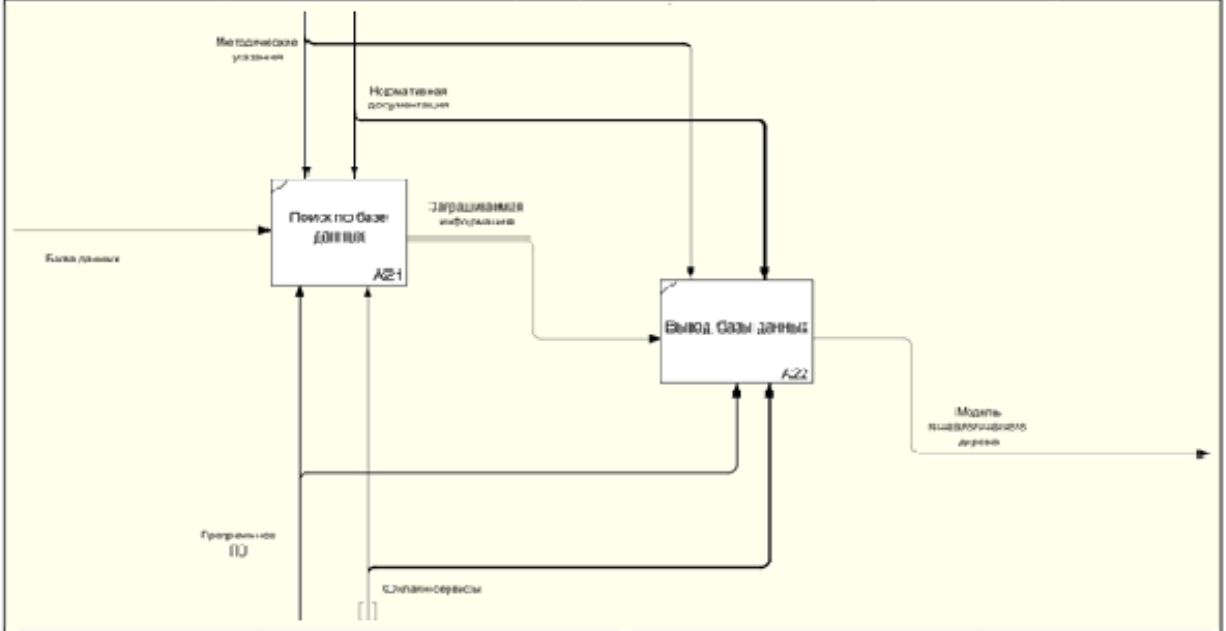
ИСПОЛЬЗУЕТСЯ В «ООО FF»	АВТОР: СОЛДАТИКОВ Д.А. 09.11.2020 ПРОЕКТ: FamilyTree		РАЗРАБАТЫВАЕТСЯ	ЧИТАТЕЛЬ ДАТА	КОНТЕКС **ВВЕРХ**
			ЧЕРНОВИК		
			РЕКОМЕНДОВАНО		
			ПУБЛИКАЦИЯ		



ВЕТКА: A0	НАЗВАНИЕ: FAMILYTREE	НОМЕР: 3

Рисунок 4 – Диаграмма декомпозиции ввода информации.

ИСПОЛЬЗУЕТСЯ В «ООО FF»	АВТОР: СОЛДАТИКОВ Д.А. 09.11.2020 ПРОЕКТ: FamilyTree		РАЗРАБАТЫВАЕТСЯ	ЧИТАТЕЛЬ ДАТА	КОНТЕКС **ВВЕРХ**
			ЧЕРНОВИК		
			РЕКОМЕНДОВАНО		
			ПУБЛИКАЦИЯ		



ВЕТКА:A0	НАЗВАНИЕ: FAMILYTREE	НОМЕР: 4

Рисунок 5 – Диаграмма декомпозиции вывода информации.

4 Диаграмма потоков данный (DFD)

Диаграмма потоков данных (data flow diagram, DFD) — один из инструментов структурного анализа и проектирования информационных систем, существовавших до широкого распространения UML. DFD – это нотация, предназначенная для моделирования информационный систем с точки зрения хранения, обработки и передачи данных.

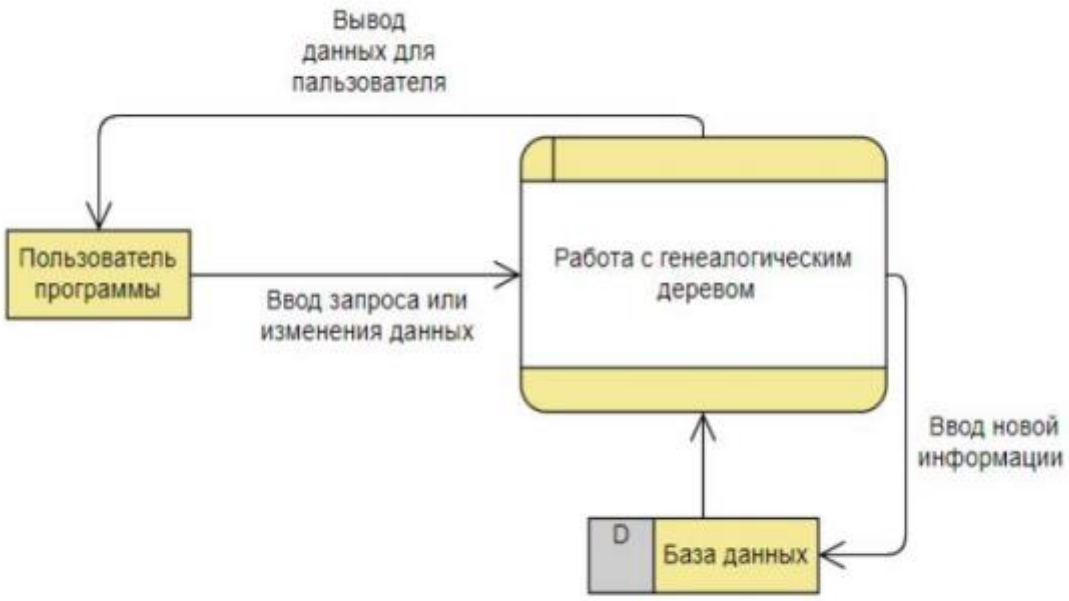


Рисунок 6 – Диаграмма DFD-системы «Генеалогическое дерево».

5 IDF3 Диаграммы

Для описания логики взаимодействия информационных потоков наиболее подходит IDEF3, называемая также workflow diagramming – методологией моделирования, использующая графическое описание информационных потоков, взаимоотношений между процессами обработки информации и объектов, являющихся частью этих процессов.

IDEF3 – это метод, имеющий основной целью дать возможность аналитикам описать ситуацию, когда процессы выполняются в определенной последовательности, а также описать объекты, участвующие совместно в одном процессе. IDEF3 дополняет IDEF0 и содержит все необходимое для построения моделей, которые в дальнейшем могут быть использованы для имитационного анализа.

Основные блоки диаграммы IDEF3:

1. Работы – являются центральными компонентами модели, изображаются прямоугольниками с прямыми углами и имеют имя, обозначающее процесс действия.
2. Связи – показывают взаимоотношение работ.
3. Перекрестки.

Окончание одной работы может служить сигналом к началу нескольких работ, или же одна работа для своего запуска может ожидать окончание нескольких работ. Перекрестки используются для отображения логики взаимодействия стрелок при слиянии и разветвлении или для отображения множества событий, которые могут или должны быть завершены перед началом следующей работы.

Обозначение	Наименование	Смысл в случае слияния стрелок	Смысл в случае разветвления стрелок
	Asynchronous AND	Все предшествующие процессы должны быть завершены	Все следующие процессы должны быть запущены
	Synchronous AND	Все предшествующие процессы завершены одновременно	Все следующие процессы запускаются одновременно
	Asynchronous OR	Один или несколько предшествующих процессов должны быть завершены	Один или несколько следующих процессов должны быть запущены
	Synchronous OR	Один или несколько предшествующих процессов завершены одновременно	Один или несколько следующих процессов запускаются одновременно
	XOR (Exclusive OR)	Только один предшествующий процесс завершен	Только один следующий процесс запускается

Рисунок 7 – Типы перекрестков.

Рисунок 9 – IDF3 диаграмма для проекта «Генеалогическое дерево»

6 IDEF1X Диаграммы

BPMN (IDEF1X предназначена для описания данных. Чаще всего такая методология используется для описания данных в целях последующей автоматизации их обработки с помощью систем управления базами данных.

Основные элементы модели IDEF1X:

1. Сущности – это множество объектов, обладающих общими характеристиками.
2. Атрибуты – характеристики сущности.
3. Отношения – это связи между двумя и более сущностями.

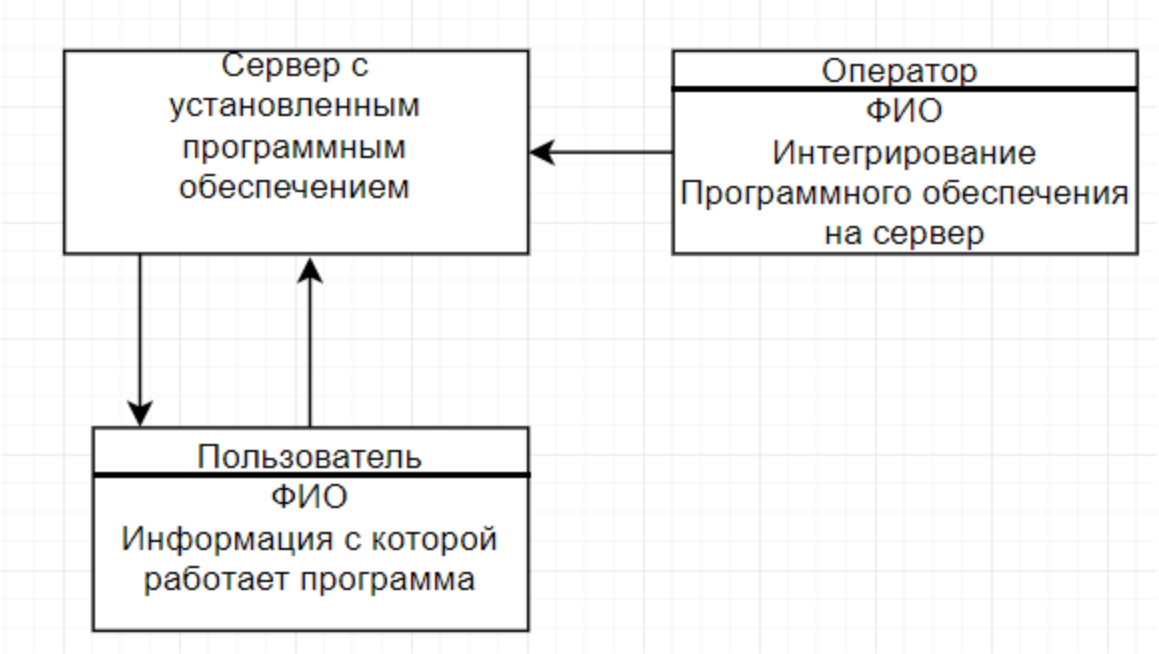


Рисунок 8 – IDEF1X диаграмма для проекта «Генеалогическое дерево»

где X – суммарное количество функциональных точек для каждого бизнес-процесса;

Fi – коэффициенты регулировки сложности.

7 Расчёт трудоёмкости с помощью функциональных точек

Оценка трудоемкости создания программного обеспечения является основной и, пожалуй, главной составляющей в определении сроков реализации программного проекта и его выполнения. Модели и методы оценки трудоемкости используются для разработки бюджета проекта, анализа степени риска, планирования и управления проектом. Анализ функциональных точек — классический метод измерения размера программного продукта с точки зрения пользователей системы. Метод предназначен для оценки на основе логической модели объема программного продукта количеством функционала, востребованного заказчиком и поставляемого разработчиком

Общее количество функциональных точек (FP-оценок) рассчитывается по формуле $FP=X*(0{,}65+0{,}01\sum_{i=1}^{14} Fi)$,

где X – суммарное количество функциональных точек для каждого бизнес-процесса;

Fi – коэффициенты регулировки сложности.

8 UML диаграммы

UML — стандартизированный язык моделирования, язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур. [8]

В UML используются следующие виды диаграмм:

Диаграмма вариантов использования

Понятие варианта использования впервые ввел Ивар Якобсон. В настоящее время вариант использования превратился в основной элемент разработки и планирования проекта. Вариант использования представляет собой последовательность действий, выполняемых системой в ответ на событие, инициируемое некоторым действующим лицом.

Действующее лицо – это роль, которую пользователь играет по отношению к системе. Действующие лица представляют собой роли, а не конкретных людей или наименования работ.

При этом на диаграмме вариантов использования существует три типа связи:

1. Связь коммуникации – это связь между вариантом использования и действующим лицом.
2. Связь включения применяется, когда имеется фрагмент поведения системы, который повторяется больше чем в одном варианте использования.
3. Связь расширения позволяет варианту использования при необходимости использовать функциональные возможности другого.

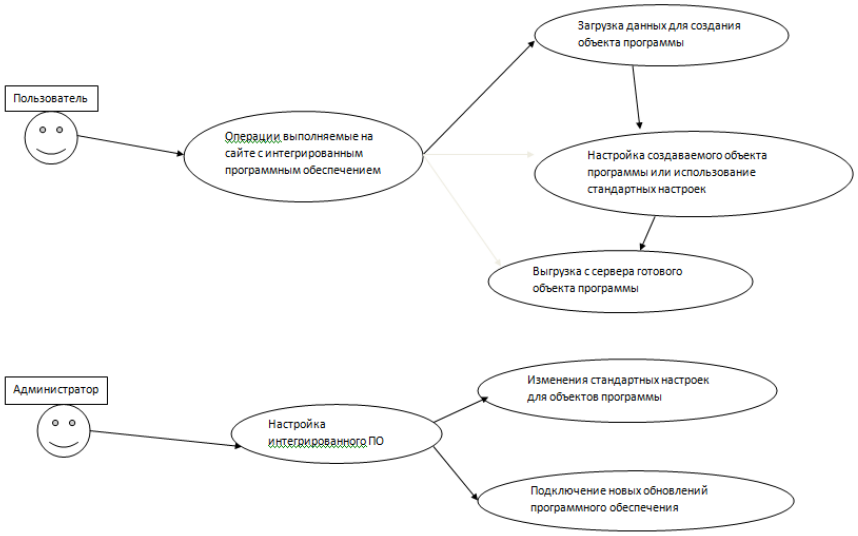


Рисунок 10 – UML-диаграмма системы «Генеалогическое дерево».

Диаграмма последовательности

Диаграмма последовательности отражает поток событий, происходящих в рамках варианта использования. Действующие лица показаны в верхней части диаграммы. Стрелки соответствуют сообщениям, передаваемым между действующим лицом и объектами для выполнения требуемых функций. На диаграмме объект изображается в виде прямоугольника, от которого вниз проведена пунктирная вертикальная линия (линия жизни).[8]

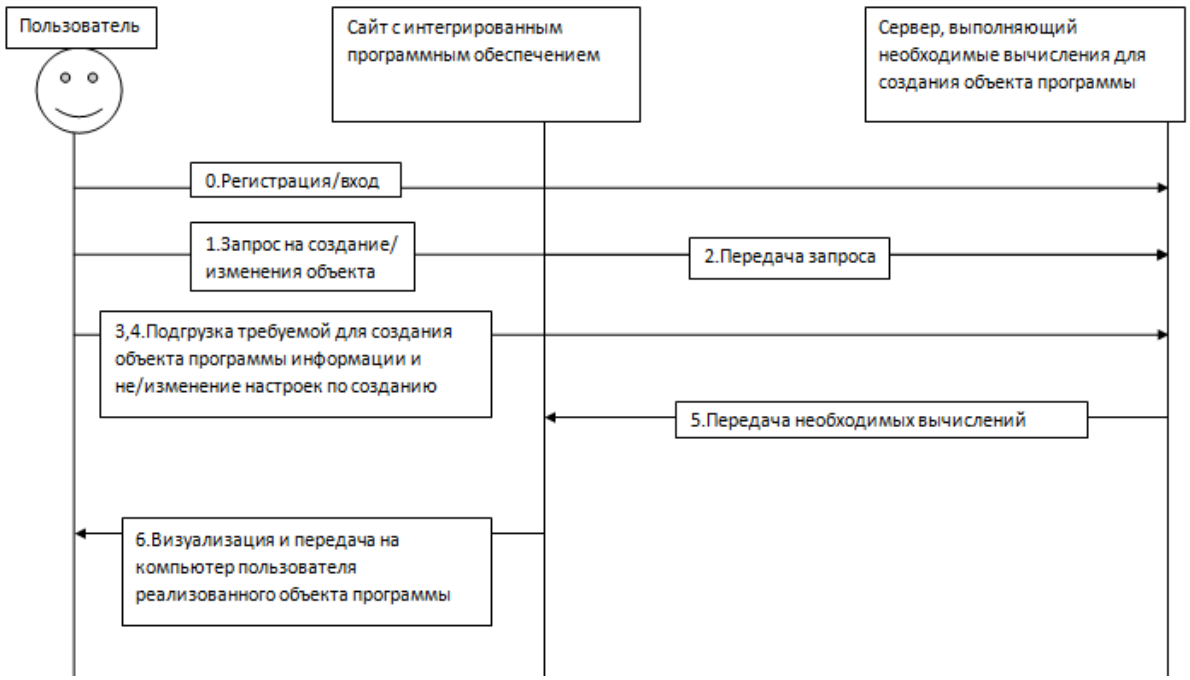


Рисунок 11 – Диаграмма последовательности для Пользователя в проекте «Генеалогическое дерево»

Диаграмма состояний

Диаграммы состояний определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате наступления некоторых событий. На диаграмме имеются два специальных состояния – начальное и конечное. Начальное состояние – это выделенная черная точка, конечное состояние – черная точка в белом круге. На диаграмме может отсутствовать конечное состояние или их может быть сколько угодно, но должно быть одно и только одно начальное состояние.

Диаграмма состояний состоит из:

1. Деятельность – поведение, реализуемое объектом, пока он находится в данном состоянии. Деятельность изображают внутри самого состояния, ей должно предшествовать слово do (делать) и двоеточие.

2. Входное действие – поведение, которое выполняет объект при переходе в данное состояние. Входное действие также показывают внутри состояния, ему предшествует слово entry (вход) и двоеточие.

3. Выходное действие – поведение, которое выполняет объект при выходе из данного состояния. Выходное действие изображают внутри состояния, ему предшествует слово exit (выход) и двоеточие.

4. Событие – то, что вызывает переход из одного состояния в другое. Событие размещают на диаграмме вдоль линии перехода.

5. Ограждающие условия определяют, когда переход может осуществиться, а когда нет. Ограждающие условия задавать необязательно. Ограждающие условия изображают на диаграмме вдоль линии перехода после имени события, заключая их в квадратные скобки

6. Действие – часть перехода. Рисуют вдоль линии перехода после имени события, ему предшествует косая черта. Действие рисуют вдоль линии перехода после имени события, ему предшествует косая черта.

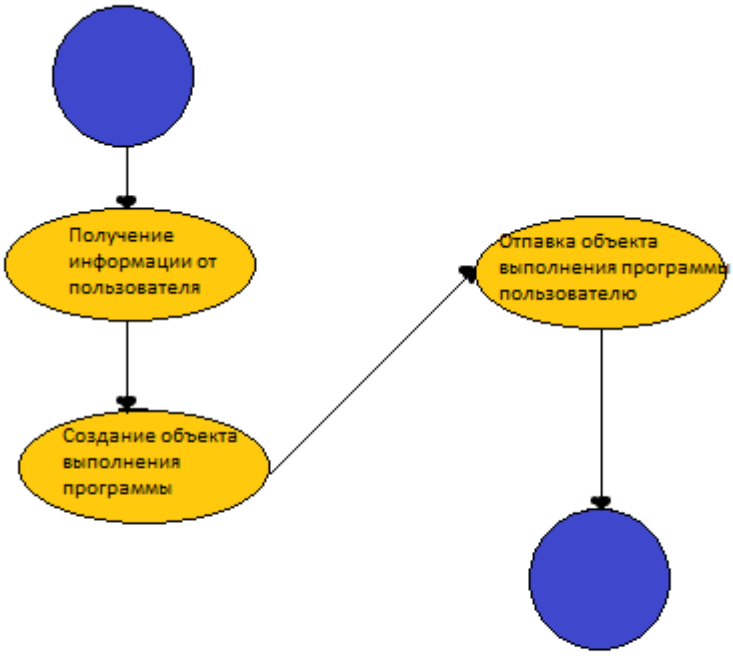


Рисунок 12 – Диаграмма состояний для проекта «Генеалогическое дерево»

Диаграмма компонентов и размещения

Диаграмма компонентов и размещения отражает физические взаимосвязи между программными и аппаратными компонентами системы. Она является хорошим средством для того, чтобы показать маршруты перемещения объектов и компонентов в распределенной системе. Каждый узел на диаграмме размещения представляет собой некоторый тип вычислительного устройства – в большинстве случаев, часть аппаратуры. Эта аппаратура может быть простым устройством или датчиком, а может быть и мэйнфреймом. Диаграмма размещения показывает физическое расположение сети и местонахождение в ней различных компонентов. Диаграмма размещения используется менеджером

проекта, пользователями, архитектором системы и эксплуатационным персоналом, чтобы понять физическое размещение системы и расположение её отдельных подсистем

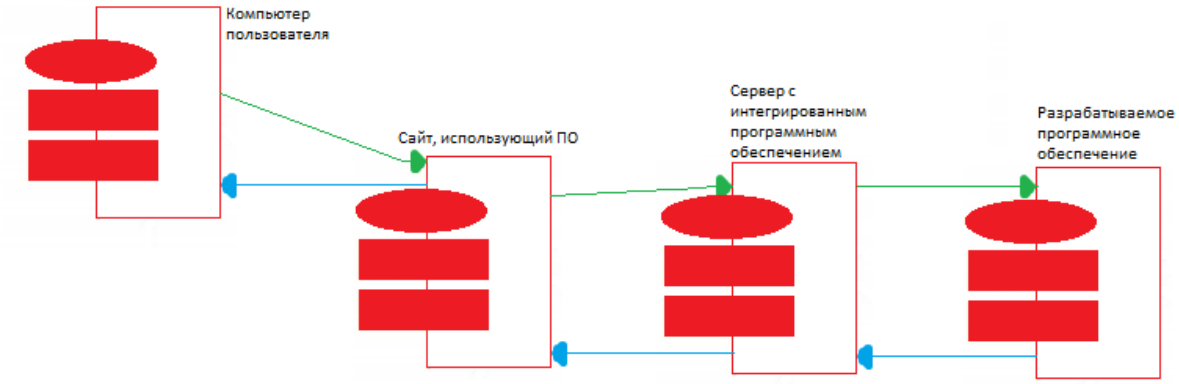


Рисунок 13 – Диаграмма компонентов и размещения для проекта «Генеалогическое дерево»

9 Результаты машинного тестирования программы

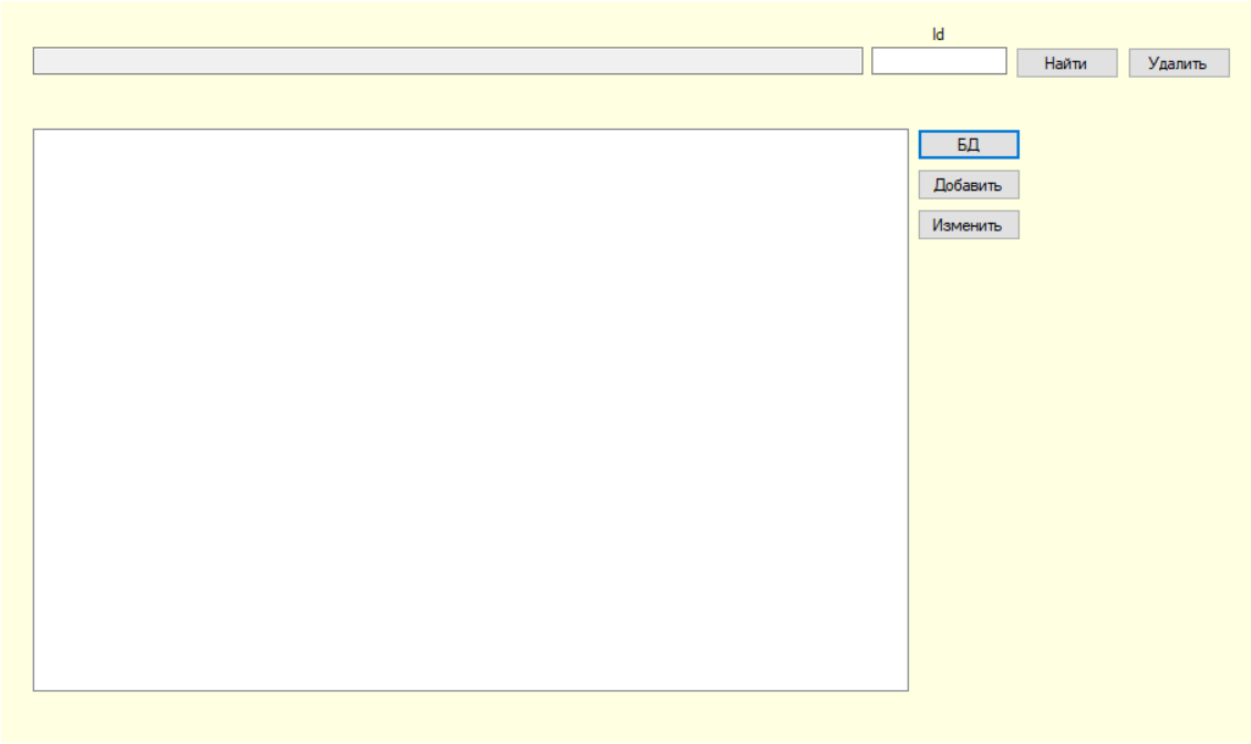


Рисунок 14 – Окно при запуске программы

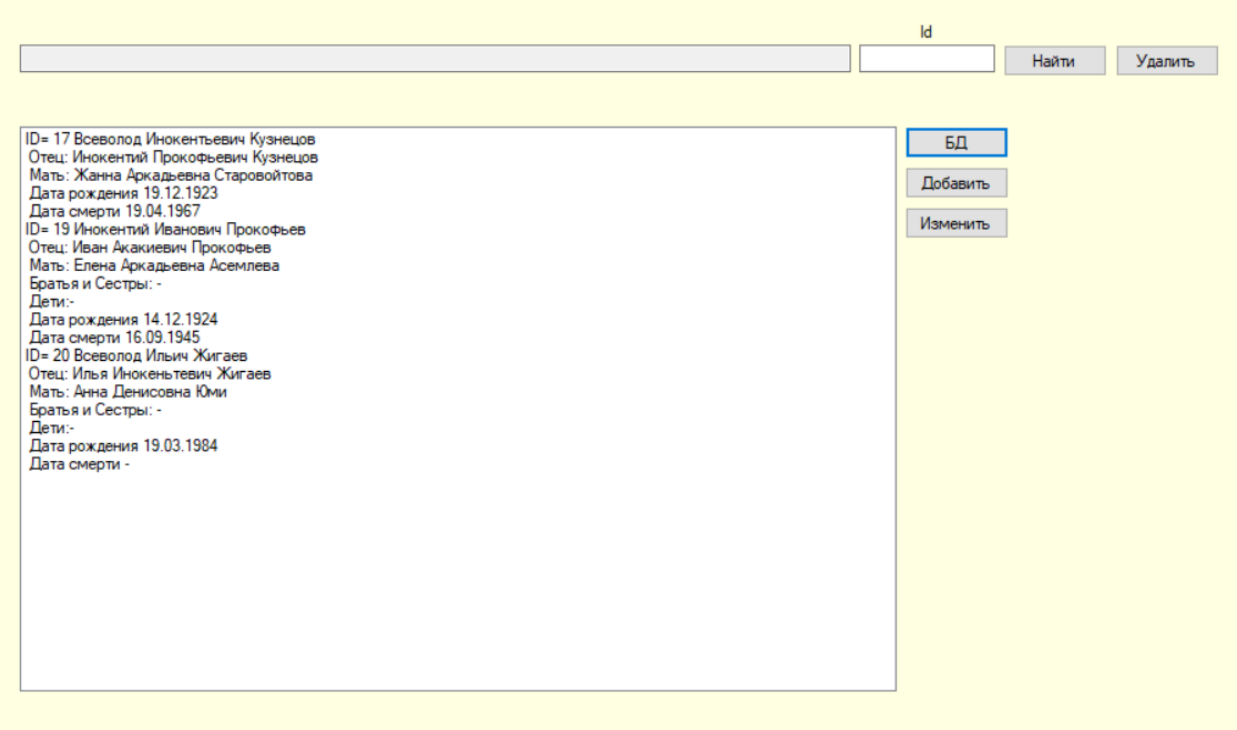


Рисунок 15 – Вывод базы данных.

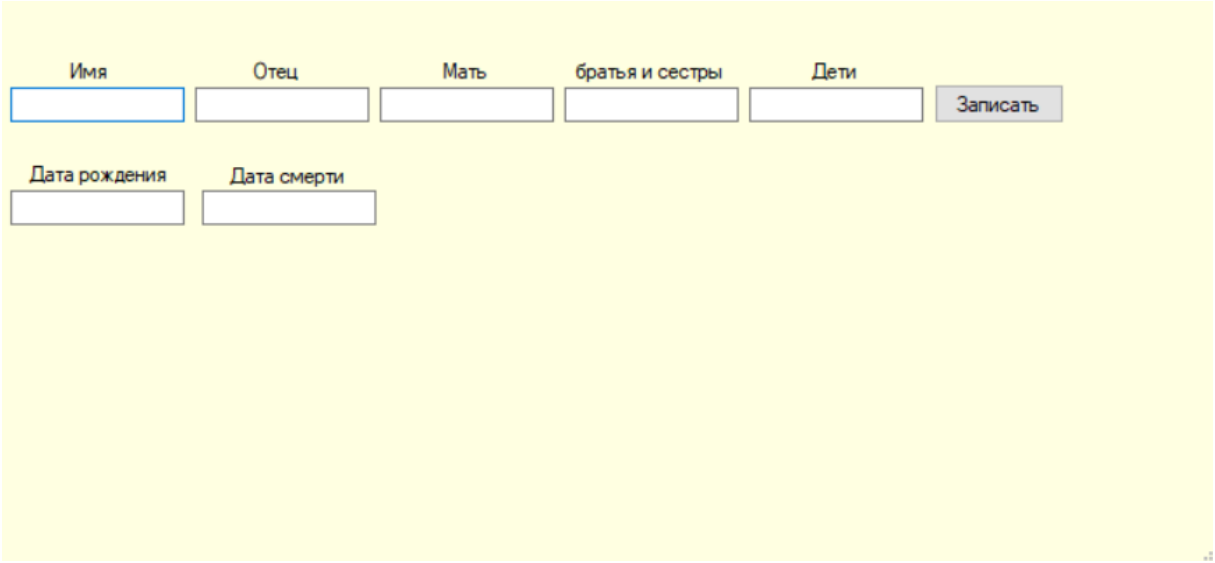


Рисунок 16 – Запрос на добавление.

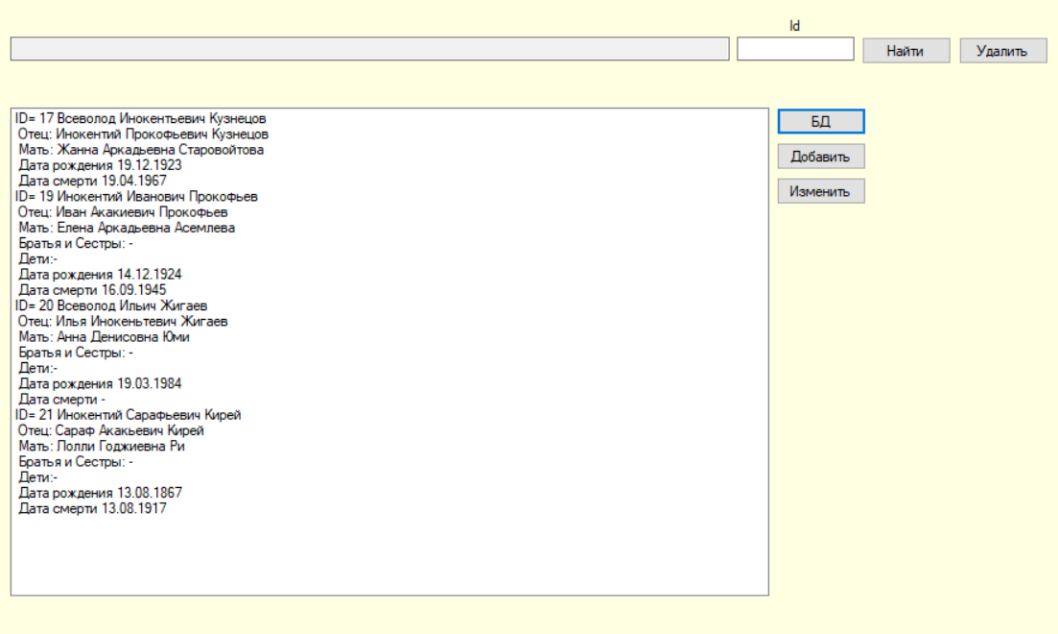


Рисунок 17 – Изменённая база данных.

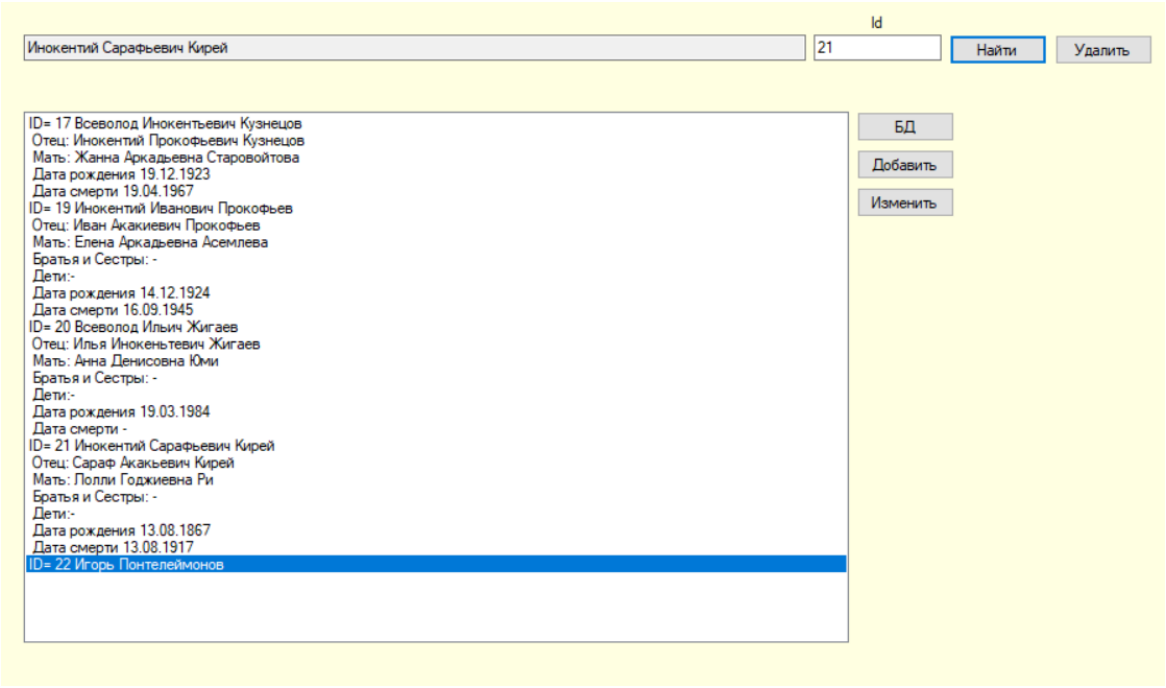


Рисунок 18 – Вывод определённого человека из базы данных с помощью соответствующего ID.

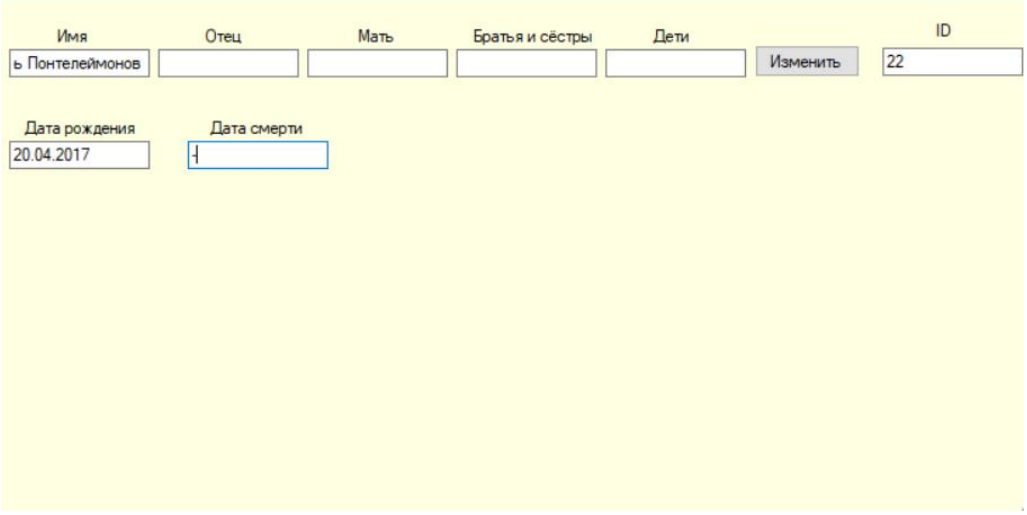


Рисунок 19 – Изменение записи с соответствующим ID.

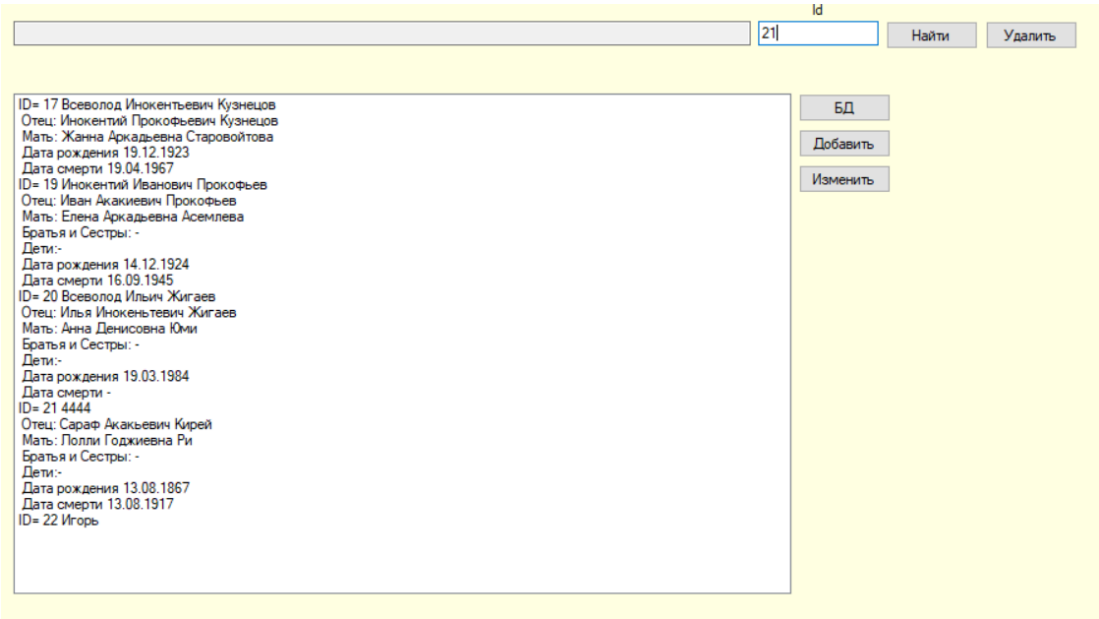


Рисунок 20 – Удаление записи, соответствующей определённому ID из базы данных.

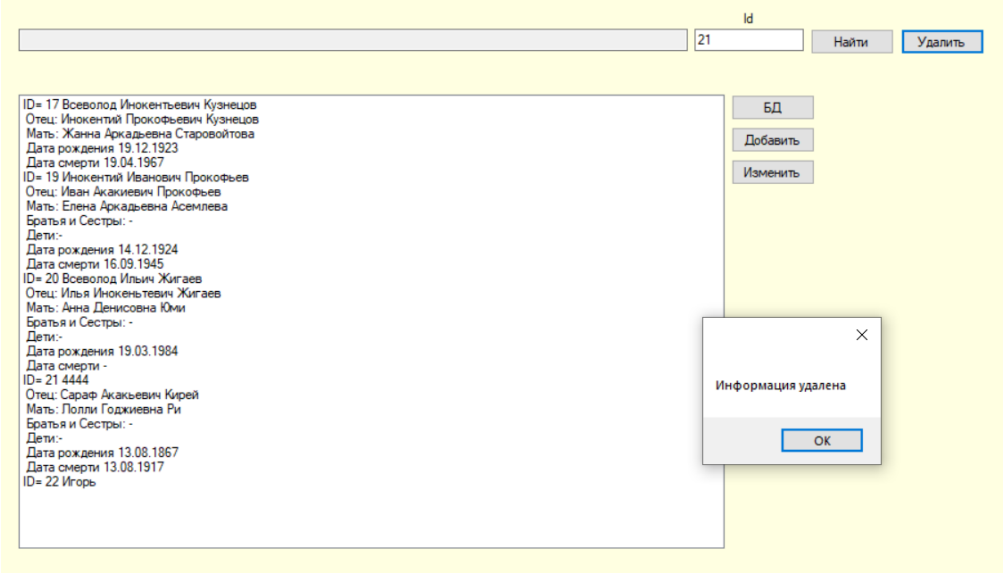


Рисунок 21 – Подтверждение удаления записи с соответствующем ID.

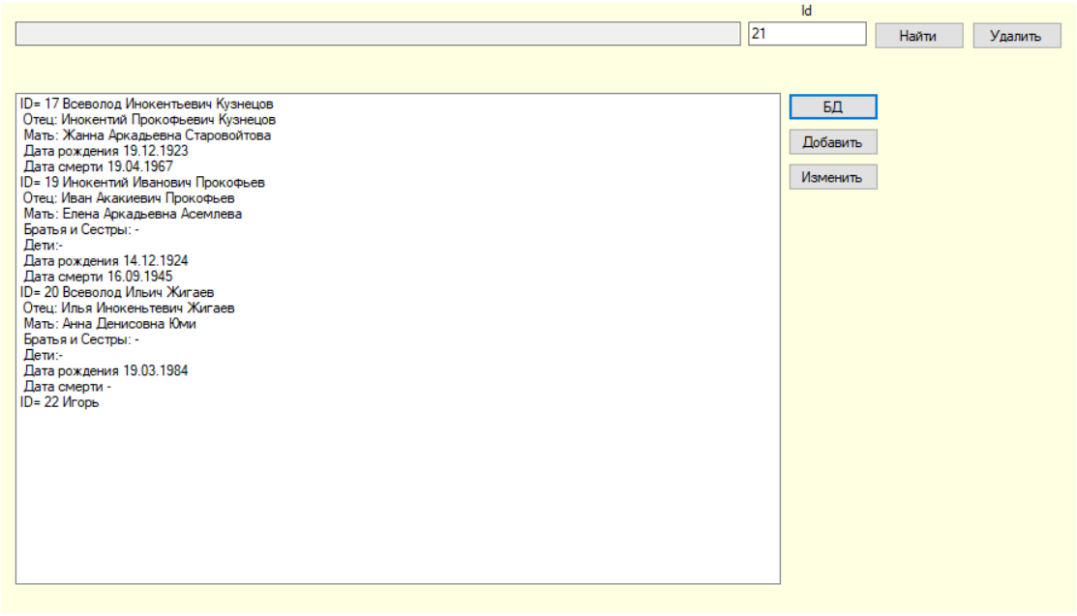


Рисунок 22 – База данных без соответствующей удалённой записи.

10 Системные требования

Таблица 1 – Системные требования программы

Процессор	2.5 ГГц
Оперативная память	150 Мб
Монитор	1920 x 1080
Свободное место на носителе	15 Мб

Устройства взаимодействия с пользователем	Клавиатура и мышь
Программное обеспечение	Windows 10

11 Руководство пользователя

Запуск программы можно осуществить следующим способом: программа поставляется на CD диске с установочной конфигурации.

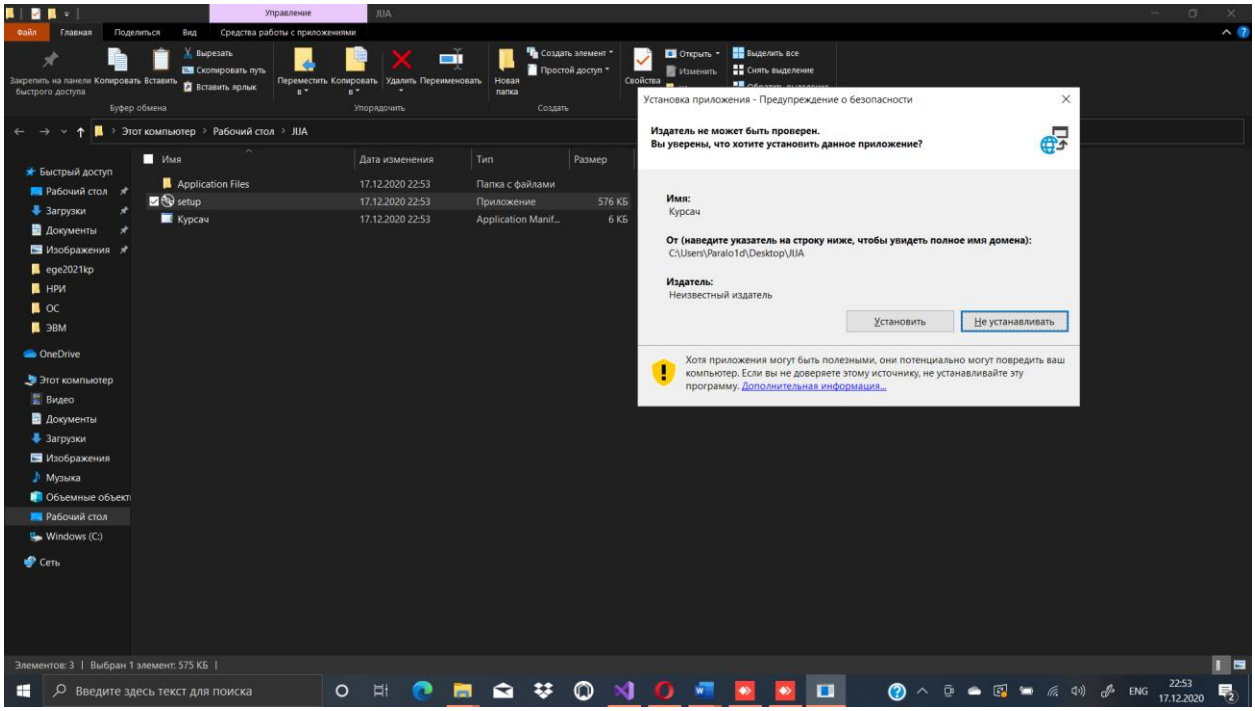


Рисунок 23 – Открытый образ установочного диска

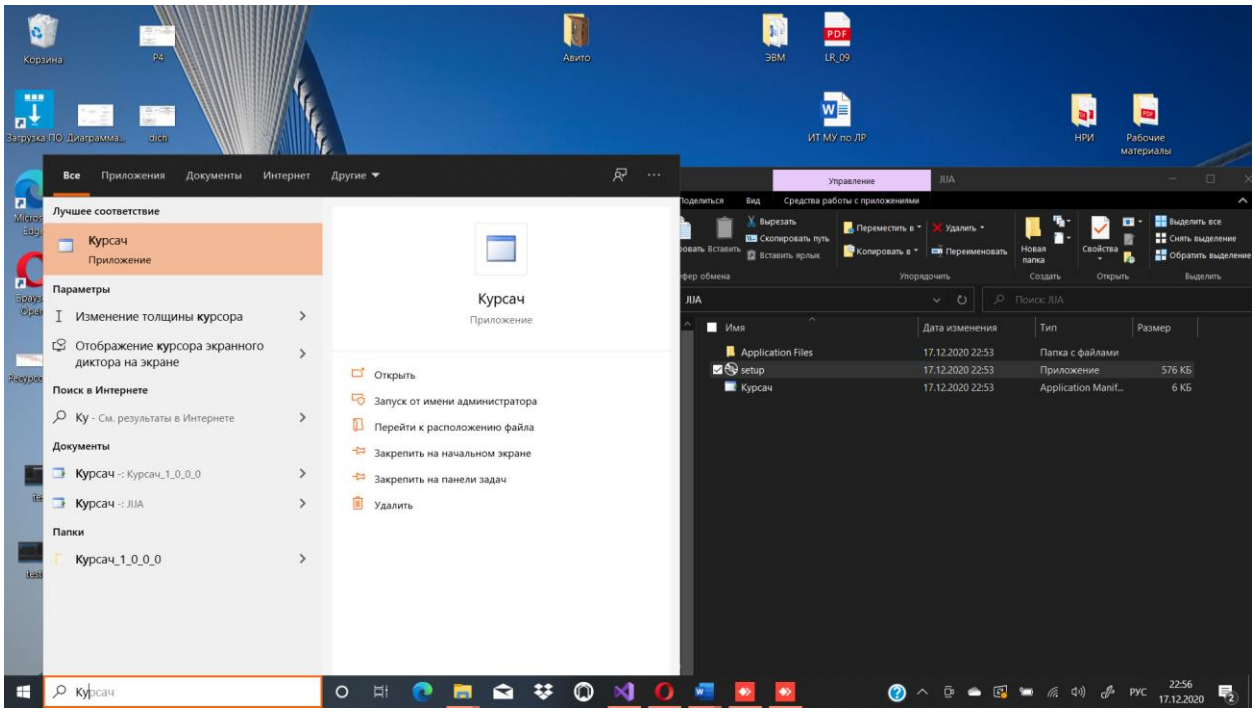


Рисунок 24 – Ярлык приложения в меню «Пуск»

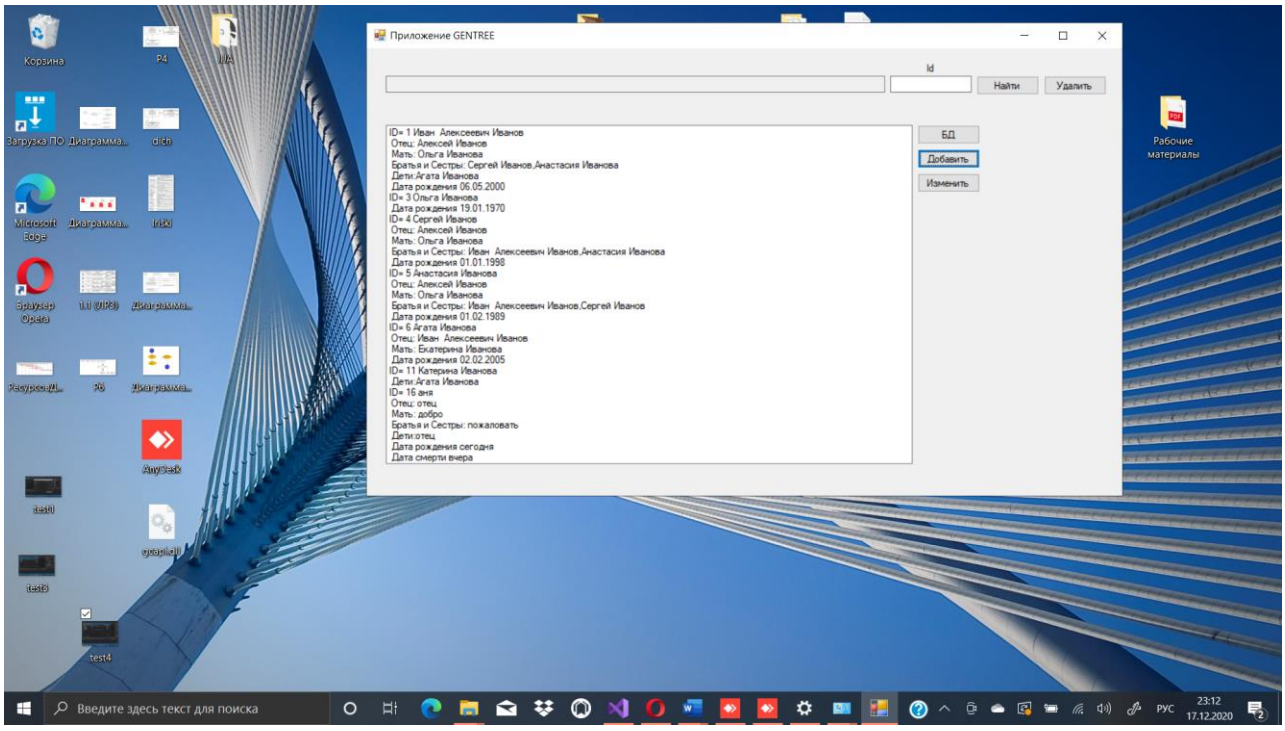


Рисунок 25 – Запущенное приложение с открытой базой данных.

Если необходимо получить все данные из БД, то нужно нажать кнопку «БД». При вводе в окно с надписью Id можно получить данные о человека при нажатии «Найти», или удалить пользователя при нажатии «Удалить». Если нажать кнопку «Добавить», то открывается новое окно, где необходимо ввести

данные о новом человеке, id присваивается автоматически. Если нажать кнопку «Изменить», то открывается новое окно, где необходимо ввести id и новые данные человека.

Заключение

В результате выполнения данного курсового проекта была спроектирована система «Генеалогическое дерево» на языке высокого уровня C#, с использованием других языков программирования, позволяющая наглядно продемонстрировать работу всех её компонентов. Полученные навыки работы с базами данных могут пригодиться в будущем. Созданные диаграммы обладают простой и понятной, структурой, позволяющей понять каждый аспект системы с различных сторон.

Код программы был отлажен вручную. Были получены важные знания и практические навыки как в области использования объектно-ориентированных языков программирования в целом, так и в области построения диаграмм проектирования, отображающих поведение различных организационных структур.

Список использованных источников

1. Ларман, Крэг. Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку / КрэгЛарман. - Москва: Гостехиздат, 2017. - 736 с.

2. Роберт А. Максимчук. UML для простых смертных / Роберт А. Максимчук, Эрик Дж. Нейбург. - Москва: СИНТЕГ, 2014. - 272 с.

3. Йордон, Эдвард. Объектно-ориентированный анализ и проектирование систем / Эдвард Йордон , Карл Аргила. - М.: ЛОРИ, 2014. - 264 с.

4. SoloLearn – C# Tutorial. [Электронный ресурс]: - Режим доступа: https://www.sololearn.com/Course/CSharp/ (Дата обращения 13.03.2020).

5. Википедия. [Электронный ресурс]: - Режим доступа: https://ru.wikipedia.org (Дата обращения 17.09.2019).

6. GitHub – yarajtf/intercom. [Электронный ресурс]: - Режим доступа: https://github.com/yarajtf/intercom (Дата обращения 06.05.2020).

7. Comindware – Нотация BPMN 2.0 [Электронный ресурс]: - Режим доступа: https://comindware.com/ru/blog-нотация-bpmn-2-0-элементы-и-описание/ (Дата обращения 28.02.2020)

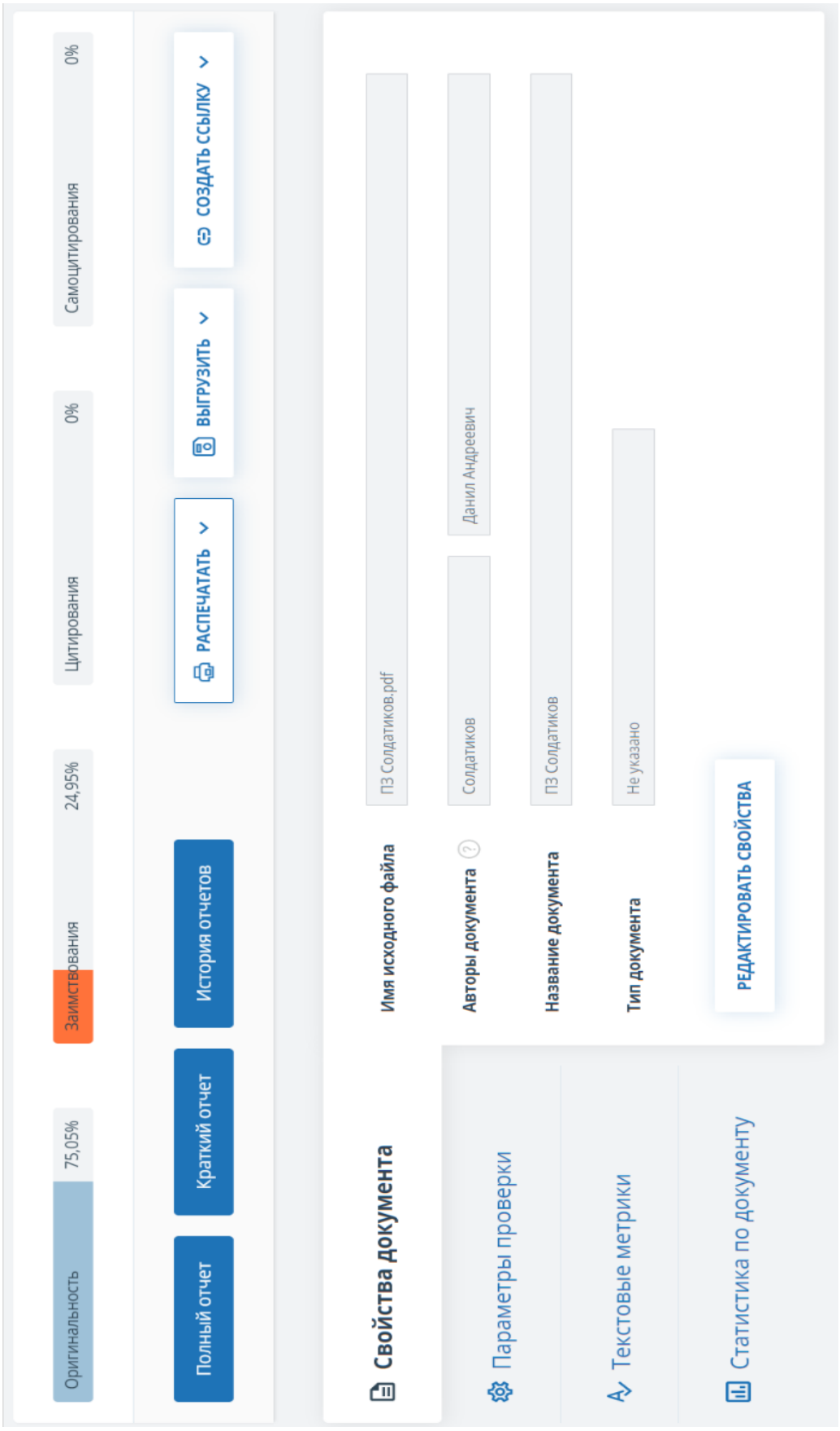
8. SysAna– Требования к системе: классификация FURPS+ [Электронный ресурс]: - Режим доступа: https://sysana.wordpress.com/2010/09/16/furps/ (Дата обращения 03.03.2020)

9. Роберт Максимчук, Нейбург: UML для простых смертных. - М.: ЛОРИ, 2008. - 268 с.

10.

10. Йордон, Эдвард. Объектно-ориентированный анализ и проектирование систем/ Эдвард Йордон, Карл Аргила. – М.: ЛОРИ, 2014 – 264 с .

Приложение А – Проверка на антиплагиат



Приложение Б – Листинг программы

Таблица 2 – Характеристика переменных к заданию

Имя переменной	Смысл переменной	Назначение переменной	Ограничения
textBox1.Text	Значение на панели	Исходная	Целочисленная, строковая
textBox2.Text	Значение на панели	Исходная	Строковая
textBox3.Text	Значение на панели	Исходная	Строковая
textBox4.Text	Значение на панели	Исходная	Строковая
textBox5.Text	Значение на панели	Исходная	Строковая
textBox6.Text	Значение на панели	Исходная	Целочисленная
textBox7.Text	Значение на панели	Исходная	Целочисленная
T	Проверка истинности	Исходная	Булевая
str	Начало командной сторки	Исходная	Строковая

Код программы:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.OleDb;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Курсач
{
    public partial class Form1 : Form
    {

        public static string connectString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=kursovoymdb.mdb;";


        private OleDbConnection myConnection;




        public Form1()
        {
            InitializeComponent();

            myConnection = new OleDbConnection(connectString);

            myConnection.Open();

        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            myConnection.Close();
        }

        private void Button1_Click(object sender, EventArgs e)
        {
            string query = "SELECT id, Name, F, M, BS, CH, Data, dedData FROM BD ORDER BY
id";


            OleDbCommand command = new OleDbCommand(query, myConnection);


            OleDbDataReader reader = command.ExecuteReader();


            listBox1.Items.Clear();

            while (reader.Read())
            {
                listBox1.Items.Add("ID= " + reader[0].ToString() + " " +
reader[1].ToString());
```

```
                if (reader[2].ToString() !="") listBox1.Items.Add(" Отец: " +
reader[2].ToString());
                if (reader[3].ToString() !="") listBox1.Items.Add(" Мать: " +
reader[3].ToString());
                if (reader[4].ToString() !="") listBox1.Items.Add(" Братья и Сестры: " +
reader[4].ToString());
                if (reader[5].ToString() !="") listBox1.Items.Add(" Дети:" +
reader[5].ToString());
                if (reader[6].ToString() !="") listBox1.Items.Add(" Дата рождения " +
reader[6].ToString());
                if (reader[7].ToString() !="") listBox1.Items.Add(" Дата смерти " +
reader[7].ToString());

            }


            reader.Close();
        }

        private void TextBox1_TextChanged(object sender, EventArgs e)
        {

        }

        private void Button2_Click(object sender, EventArgs e)
        {

            string query = "SELECT Name FROM BD WHERE id = " + textBox1.Text;


            OleDbCommand command = new OleDbCommand(query, myConnection);


            textBox2.Text = command.ExecuteScalar().ToString();


        }

        private void Button1_Click_1(object sender, EventArgs e)
        {
            Form2 form = new Form2();
            this.Hide();
            form.ShowDialog();
            this.Show();
        }

        private void Button3_Click(object sender, EventArgs e)
        {
            string query = "DELETE FROM BD WHERE id =" + textBox1.Text;
            OleDbCommand command = new OleDbCommand(query, myConnection);
            command.ExecuteNonQuery();
            MessageBox.Show("Информация удалена");
        }

        private void Button4_Click(object sender, EventArgs e)
        {
            Form3 form = new Form3();
            this.Hide();
            form.ShowDialog();
            this.Show();
        }

        private void Button5_Click(object sender, EventArgs e)
        {

        }
```

```
        private void Label1_Click(object sender, EventArgs e)
        {

        }
    }
}

public partial class Form2 : Form
    {
        public static string connectString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=kursovoymdb.mdb;";

        private OleDbConnection myConnection;
        public Form2()
        {
            InitializeComponent();
            myConnection = new OleDbConnection(connectString);

            myConnection.Open();
        }

        private void Form2_Load(object sender, EventArgs e)
        {

        }

        private void TextBox1_TextChanged(object sender, EventArgs e)
        {

        }

        private void Button1_Click(object sender, EventArgs e)
        {
            string query = "INSERT INTO BD (Name, F, M, BS, CH, Data, dedData) VALUES ('" +
textBox1.Text + "', '" + textBox2.Text + "', '" + textBox3.Text + "','" + textBox4.Text +
"','" + textBox5.Text + "','" + textBox6.Text + "','" + textBox7.Text + "')";
            OleDbCommand command = new OleDbCommand(query, myConnection);
            command.ExecuteNonQuery();
            this.Close();
        }

        private void Label1_Click(object sender, EventArgs e)
        {

        }

        private void TextBox6_TextChanged(object sender, EventArgs e)
        {

        }
    }
}


public partial class Form3 : Form
    {
        public static string connectString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=kursovoymdb.mdb;";
        //public static string connectString = "Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=kursovoymdb.mdb;";

        private OleDbConnection myConnection;
        public Form3()
        {
            InitializeComponent();

            myConnection = new OleDbConnection(connectString);
```

```
            myConnection.Open();
        }

        private void Button1_Click(object sender, EventArgs e)
        {
            if (textBox1.Text != "" && textBox2.Text != "" && textBox3.Text != "" &&
textBox4.Text != "" && textBox5.Text != "") return;
            bool T = false;
            string str = "UPDATE BD SET ";
            if(textBox1.Text!="")
            {
                str += "Name = '" + textBox1.Text + "'";
                T = true;
            }
            if (textBox2.Text != "")
            {

                str += T ? ", " : " ";
                str += "F = '" + textBox2.Text + "' ";
                T = true;
            }

            if (textBox3.Text != "")
            {
                str += T ? "," : "";
                str += "M = '" + textBox3.Text + "' ";
                T = true;

            }
            if (textBox4.Text != "")
            {
                str += T ? "," : "";
                str += "BS = '" + textBox4.Text + "' ";
                T = true;
            }
            if (textBox5.Text != "")
            {
                str += T ? "," : "";
                str += "CH = '" + textBox5.Text + "' ";
                T = true;
            }
            if (textBox5.Text != "")
            {
                str += T ? "," : "";
                str += "Data = '" + textBox7.Text + "' ";
                T = true;
            }
            if (textBox5.Text != "")
            {
                str += T ? "," : "";
                str += "dedData = '" + textBox8.Text + "' ";

            }

            str += "WHERE id = " + textBox6.Text;
            OleDbCommand command = new OleDbCommand(str, myConnection);
            command.ExecuteNonQuery();
            this.Close();
        }

        private void Form3_Load(object sender, EventArgs e)
        {

        }

        private void TextBox6_TextChanged(object sender, EventArgs e)
```

```
    {

    }

    private void TextBox1_TextChanged(object sender, EventArgs e)
    {

    }

    private void TextBox7_TextChanged(object sender, EventArgs e)
    {

    }

    private void Label8_Click(object sender, EventArgs e)
    {

    }

    private void TextBox8_TextChanged(object sender, EventArgs e)
    {

    }
}
```