МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

РАБОТА ПРОВЕРЕНА
Рецензент
_____
«___»_____ 2019 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____ 2019 г.

# Разработка веб-приложения для поиска возможных родственников на основе генеалогических деревьев формата GedCom

## ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
## К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____И.Л. Надточий
«___»_____2019 г.

Автор работы,
студент группы КЭ-452
_____А.Е. Горохов
«___»_____2019 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____2019 г.

Челябинск 2019

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____2019 г.

# ЗАДАНИЕ

**на выпускную квалификационную работу бакалавра**

студенту группы КЭ-452
Горохову Артемию Евгеньевичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Разработка веб-приложения для поиска возможных родственников на основе генеалогических деревьев формата GedCom» утверждена приказом по университету от 25 апреля 2019 г. №899

2. **Срок сдачи студентом законченной работы:** 1 июня 2019 г.

3. **Исходные данные к работе:** статьи, книги, техническое задание.
   - Томпсон, Л. Разработка web-приложений на PHP и MySQL / Л. Томпсон, Л. Веллинг; под ред. С.Н. Козлова, науч. ред. Ю.Н. Артеменко. – 2-е изд., испр. – СПб: ООО «ДиаСофтЮП», 2003. – 672 с;
   - Yiiframework community – https://yiiframework.com.ua/ru;
   - Бьюли, А. Изучаем SQL / А. Бьюлен; пер. с англ. Н. Шахотина – СПб: Символ-Плюс, 2007. – 312 с;

- Симдянов, И. Самоучитель PHP 7 / И. Симдянов, М. Кузнецов – СПб: БХВ-Петербург, 2018. – 450 с;

4. **Перечень подлежащих разработке вопросов:**
    - анализ современных технологий для создания веб-приложений;
    - выбор среды и средств реализации приложения для работы с генеалогическим древом;
    - проектирование архитектуры приложения с учётом специфики построения генеалогических древ;
    - составление технического задания к разработке веб-приложения;
    - определение функциональных и нефункциональных требований к приложению;
    - разработка функционала приложения для обеспечения работы с генеалогическим деревьями;
    - создание интуитивно понятного интерфейса пользователя;
    - создание серверной и клиентской частей веб-приложения;
    - создание парсера файлов GedCom для загрузки информации о древе в базу данных приложения;
    - тестирование разработанного программного обеспечения.
5. **Дата выдачи задания:** 1 декабря 2018 г.

Руководитель работы _____/*И.Л. Надточий*/

Студент _____/*А.Е. Горохов* /

КАЛЕНДАРНЫЙ ПЛАН

| Этап | Срок сдачи | Подпись руководителя |
|---|---|---|
| *Введение и обзор литературы* | *15 января 2019 года* | |
| *Разработка инфологической модели* | *2 февраля 2019 года* | |
| *Разработка датологической модели* | *22 февраля 2019 года* | |
| *Проектирование базы данных* | *10 марта 2019 года* | |
| *Разработка серверной части веб-приложения* | *5 апреля 2019 года* | |
| *Разработка внешнего оформления веб-приложения* | *25 апреля 2019 года* | |
| *Тестирование веб-приложения* | *7 мая 2019 года* | |
| *Оформление пояснительной записки* | *25 мая 2019 года* | |
| *Приемка приложения при комиссии высшего учебного заведения* | *1 июня 2019 года* | |

Руководитель работы _____/*И.Л. Надточий*/

Студент _____/*А.Е. Горохов*/

Аннотация

А.Е. Горохов. Разработка веб-приложения для поиска возможных родственников и возможных родственных связей в формате GedCom. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2019, 107 с., 47 ил., библиогр. список – 23 наим.

В рамках выпускной квалификационной работы проведён анализ существующих средств разработки. Предоставлен список существующих языков программирования, был проведён анализ языков, в следствии которого был выбран подходящий.

Организуется разработка алгоритма для поиска возможных родственных связей, предусмотрено взаимодействие веб-приложения с файлами формата GedCom. Доказывается способность представленного веб-приложения к стабильной работе.

Созданное веб-приложение поддерживает аутентификацию пользователей по логину и паролю, позволит составлять генеалогические деревья, просматривать их содержимое, позволит производить поиск людей, существующих в базе данных.

Новизна в плане функционала:

- система поиска информации обо всех людях, находящихся в базе данных веб-приложения;
- система автоматического поиска возможных родственников по всем деревьям, находящимся в базе данных;
- возможность добавления родственника вне древа;
- использование фреймворка Yii2 для создания веб-приложения для поиска родственников;
- разработан новый способ построения генеалогических древ.

# ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

## Актуальность исследования

В наше время практически невозможно представить человека, который не использовал бы сеть интернет на персональном компьютере или мобильном устройстве. Каждый день более половины людей во всём мире посещают миллиарды, а то и больше веб-страниц.

Веб-приложения – это программные продукты, которые разработаны для персональных компьютеров, ноутбуков, смартфонов и других устройств. Зачастую веб-приложения доступны для других пользователей в свободном доступе, использовать их можно при наличии браузера и доступа к сети интернет.

Актуальность данной исследовательской темы заключается в том, что каждому из нас, даже во время технологичного прорыва и нескончаемого потока информации, порой недостаёт знаний о истории своего рода. Ведь нередко изучение своего рода, его истории может помочь понять нам самих себя. Ведь наверняка, каждый из нас хотя бы раз в своей жизни задумывался: «А кем были наши родственники?»

На сегодняшний день, ассортимент веб-приложений для построения деревьев достаточно разнообразен, но функции поиска родственников в них, как правило, не предусмотрена. Данная функция, при её введении, может расширить функционал веб-приложений данного формата и подогреет интерес пользователей к ним.

Проект будет применяться пользователями для составления генеалогических деревьев, а также для поиска возможных родственников, которые уже зарегистрировались в данном приложении. Это даст возможность пользователям завести новые знакомства с родственниками столь далёкими, что вы даже и не знаете их.

В целях дальнейшей формализации проекта были выделены и сформулированы следующие понятия.

Объект исследования – процесс разработки веб-приложения.

Предмет исследования – проблемы и подходы к решению проблем, связанных с разработкой веб-приложения.

Цель исследования – разработать жизнеспособный продукт для поиска возможных родственников и родственных связей.

## Цель и задачи исследования

Целью проекта является разработка веб-приложения для поиска возможных родственников и возможных родственных связей в формате GedCom [10]. Использование формата GedCom позволит пользователям загружать древо в систему, не вводя при этом никаких данных о составе семьи, т.к. все данные уже хранятся в файле.

Для реализации вышеупомянутой цели были поставлены следующие задачи:

- Проанализировать существующие подходы к разработке веб-проектов и выбрать подходящий;
- выбрать необходимый инструментарий разработки;
- обосновать сделанный выбор;
- определить конкретную логику бизнес-процессов;
- разработать жизнеспособный продукт.

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Веб-приложения построены на архитектуре «клиент-сервер». Само веб-приложение находится на сервере и обрабатывает запросы, которые передают ему через Internet многочисленные клиенты. На стороне клиента веб-приложение работает в браузере. Пользовательский интерфейс веб-приложения передается на клиентскую машину в виде страниц на языке HTML (Hypertext Markup Language), где браузер интерпретирует и отображает их. На стороне сервера веб-приложение работает под управлением IIS (Internet Information Services). IIS управляет работой приложения, передает ему клиентские запросы и возвращает клиентам результаты исполнения их запросов. Запросы и результаты их исполнения передаются через интернет по протоколу HTTP. Протокол – это набор правил, регламентирующих взаимодействие двух и более сущностей, которое реализуется через среду, такую, как интернет [1].

Веб-приложения во многом напоминают традиционные веб-сайты, но в отличие от них отображают пользователю динамическое содержимое, генерируемое исполняемым кодом приложения, а не статические страницы, хранящиеся на сервере в готовом виде.

Однако за длительное время развития интернета функционал веб-приложений начинал обрастать всё более новыми «начинками». Например, JavaScript превращал обычные статичные страницы в более сложные интерфейсы и всё это запускалось через браузер, что в одно время считалось по-настоящему чем-то потрясающим. С появлением асинхронных Ajax-запросов, сложность интерфейсов вырастала ещё больше. Возрастающая сложность разработки веб-приложений явилась причиной спроса на различные фронтенд-фреймворки, а забота о безопасности веб-приложений и желание использовать шаблонную разработку – бэкенд-фреймворки.

Одним из самых основных способов веб-разработки является разделение фронтенд и бэкенд частей [1]. Доступ к базе данных и к бизнес-

логике имеет только независимая бэкенд часть приложения, а за вывод информации для пользователя и получение информации от него – фронтенд. В теории обе эти части программы должны быть независимы друг от друга. Разработчик может изменить стек фронтенд составляющей, при этом ничего не изменяя в бэкенде. С таким подходом один сервер может обслуживать несколько разных приложений: веб, мобильное приложение и десктопное.

При создании генеалогических древ, пользователю часто предлагают воспользоваться файлом формата GedCom. Это стандартизованный файл, который используется по всему миру для хранения информации о генеалогическом древе [9]. GedCom был разработан церковью Иисуса Христа Святых последних дней для помощи в генеалогических исследованиях. Формат файла с расширением GedCom – текстовый. Текущая версия стандарта – 5.5. Описывает иерархическую структуру данных (как XML): раздел персон, семей и т.д.

## 1.1 АРХИТЕКТУРА ВЕБ-ПРИЛОЖЕНИЯ

Написанное веб-приложение будет построено по основным правилам построений данного вида приложения, следовательно, его архитектура будет похожа на уже имеющиеся аналогичные проекты.

На рисунке 1.1.1 показан пример отправки заполненных полей объекта person, порядок обработки запроса в веб-приложении и вывод уже записанного объекта на экран.

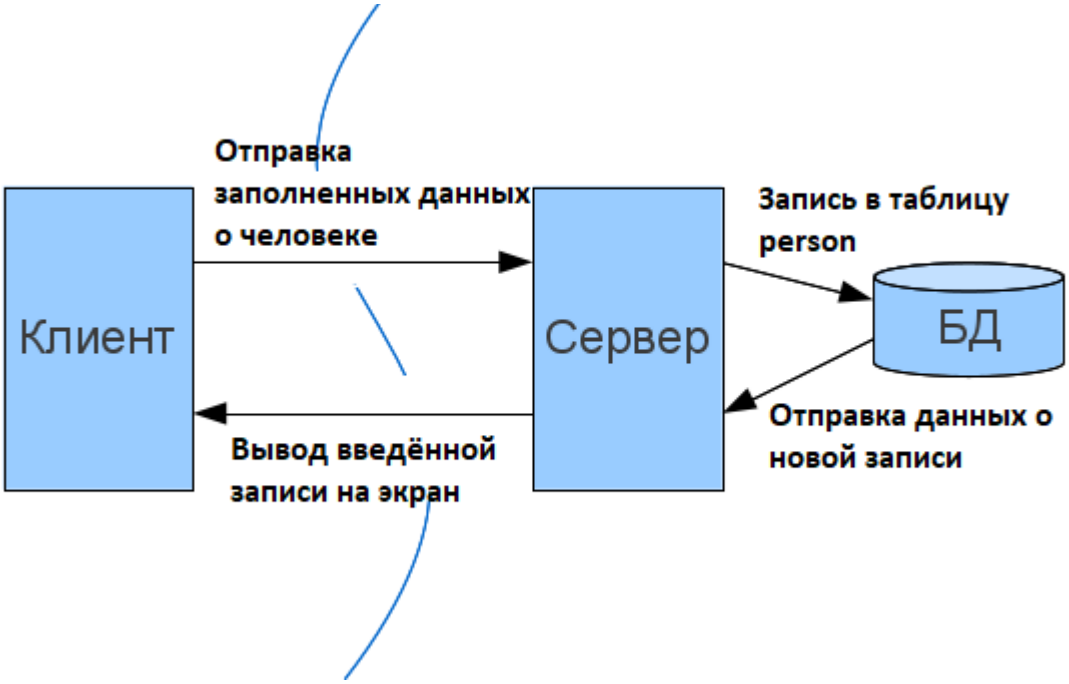


Рисунок 1.1.1 – Архитектура веб-приложения

Весь смысл архитектуры веб-приложения состоит в том, что клиент, взаимодействуя с приложением, посылает запросы на сервер, сервер, в свою очередь, обрабатывает поступающие запросы и, обращаясь к базе данных, «достаёт» необходимые для выполнения запроса данные. После этого, полученные данные сервер отображает на мониторе клиента в браузере, переадресовав на какую-либо страницу приложения.

На рисунке 1.1.2 показана подробная архитектура веб-приложения. Показаны все возможные действия пользователя после авторизации в системе.

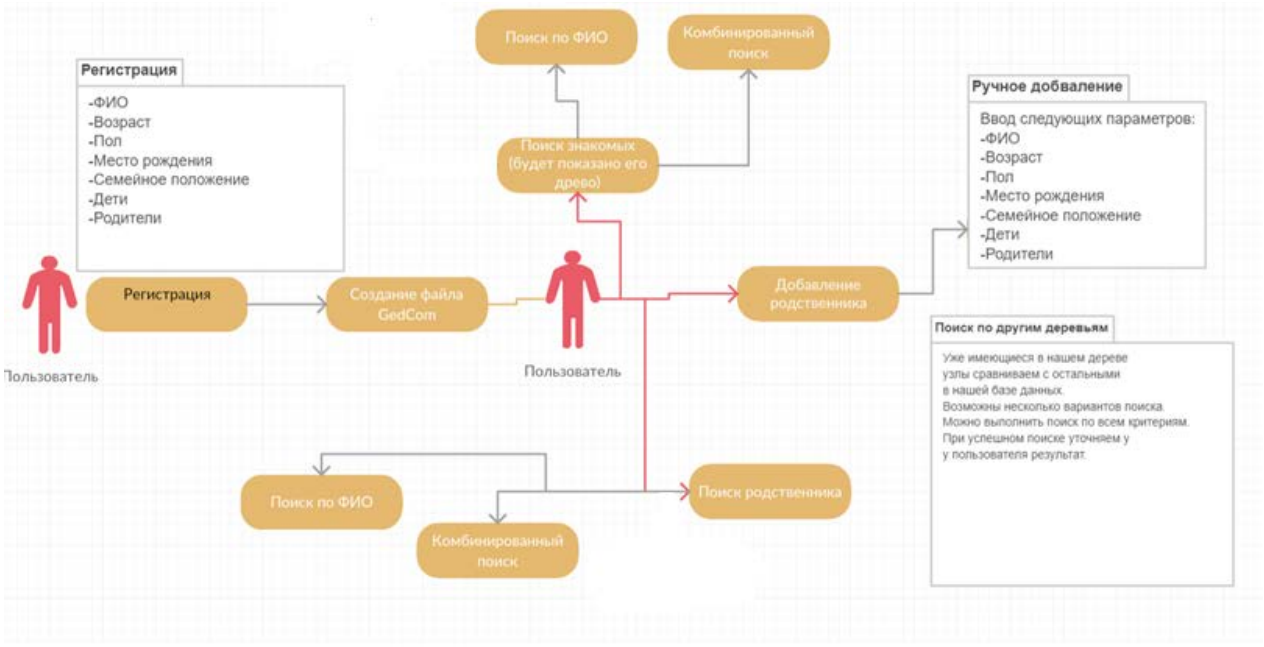


Рисунок 1.1.2 – Архитектура веб-приложения

## 1.2 ОБЗОР АНАЛОГОВ

На сегодняшний день уже имеются разработки веб-приложений для поиска родственников, но не все из них удовлетворяют потребностям пользователей. Даже можно сказать, что многие из созданных вариантов данного веб-приложения порой могут только усложнить жизнь для пользователей, а никак не удовлетворить его потребности.

Для выявления преимуществ и недостатков для анализа были выбраны на рассмотрение два веб-приложения: «MyHeritage.com» и «rodovid.org».

### “MyHeritage.com”

“MyHeritage.com” - веб-приложение для поиска родственников [11]. На рисунке 1.2.1 показана главная страница веб-приложения “MyHeritage”.

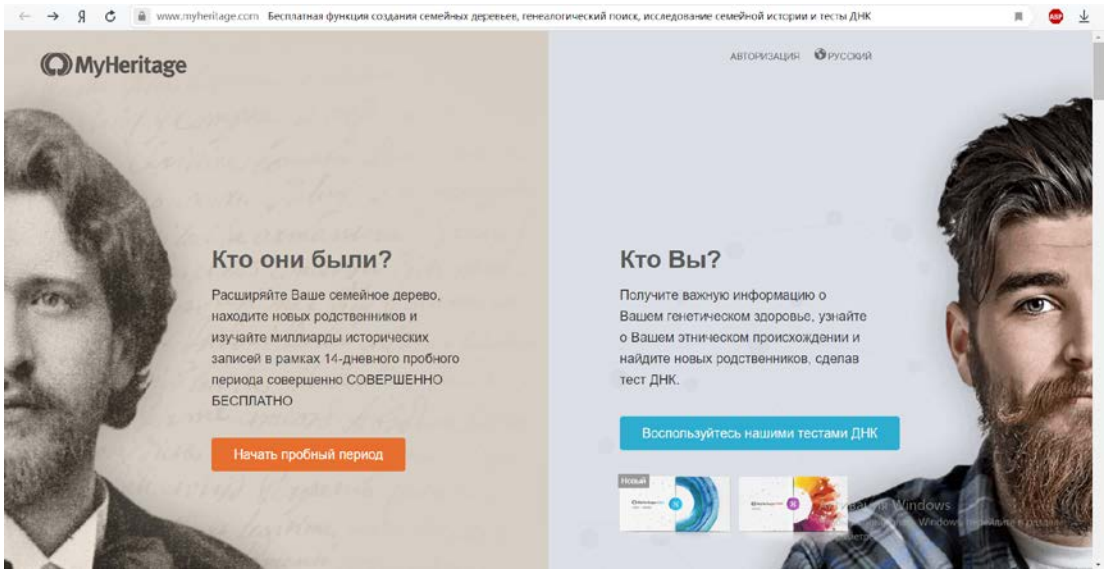


Рисунок 1.2.1 – Главная страница веб-сайта Myheritage.com

Приложение предоставляет возможность заполнить семейное дерево. Приложение является достаточно популярным, при этом являясь коммерческим, заполнить дерево предоставляется бесплатно, но за остальные функции может взыматься плата. Также предлагается заказать ДНК-тест за отдельную плату. Имеется достаточно много положительных отзывов, но практически в каждом из них, указывалось недовольство высокой стоимостью полной версии сайта.

### “rodovid.org”

“rodovid.org” – веб-приложение для создания семейных древ [11]. На рисунке 1.2.2 показана главная страница веб-приложения “Rodovid”.


Рисунок 1.2.2 – Главная страница веб-приложения Rodovid.org

Приложение не является коммерческим, до сих пор получает поддержку от разработчиков. В функционал приложения входит составление древа, просмотр других деревьев, даже присоединение к уже имеющемуся древу другого пользователя ваших записей. Но возможности добавления родственника вне древа, когда пользователь точно знает, что является предком кого-то, но точно не уверен кем были его родители или дети.

### «Древо жизни»

«Древо жизни» – программа для создания генеалогических древ [20]. На рисунке 1.2.3 показан интерфейс приложения «Древо жизни».

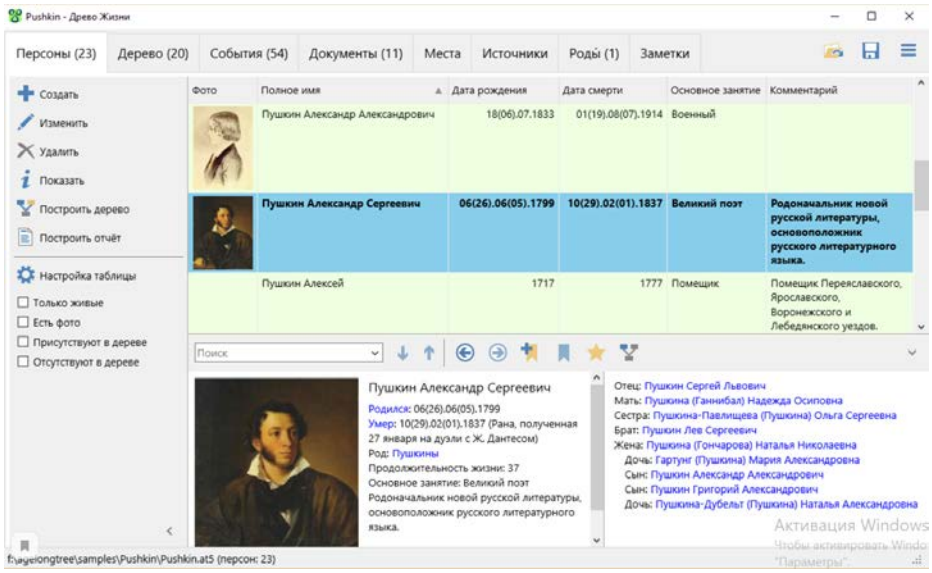


Рисунок 1.2.3 – Интерфейс приложения «Древо жизни»

Приложение является «десктопным». Данный факт можно выделить как существенный минус, т.к. веб-приложения в наше время намного популярнее, поскольку они имеют более высокий уровень мобильности. Часть функций приложения являются платными. Максимальный размер древа в приложении – 40 человек. Дальнейшее заполнение древа и работа с ним будут платными.

### “moederevo”

Сервис «Moederevo» – веб-приложение для создания генеалогических древ. Интерфейс веб-приложения показан на рисунке 1.2.4.

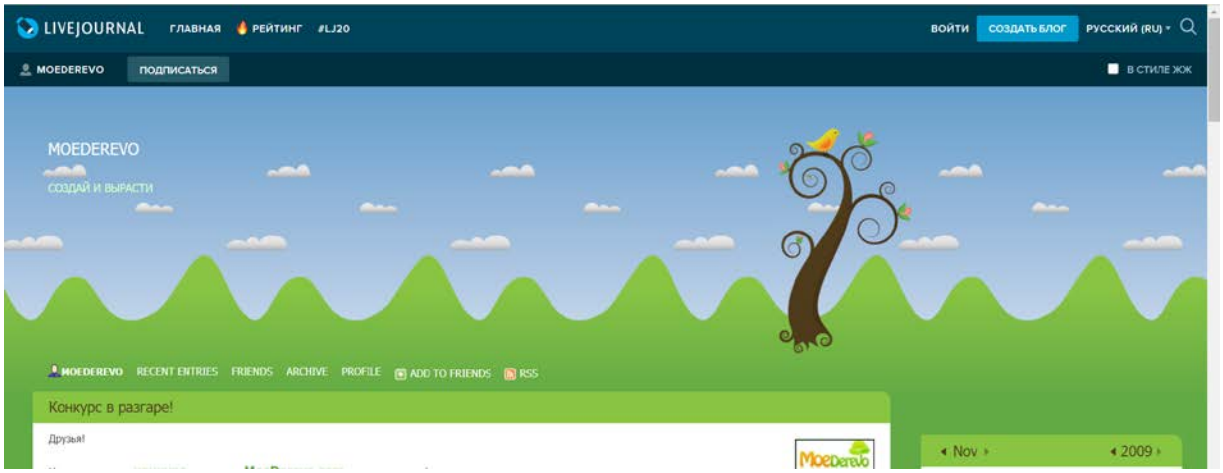


Рисунок 1.2.4 – Интерфейс веб-приложения “moederevo”

Помимо графического редактора для построения деревьев и интуитивно понятного интерфейса, предоставляет возможность печати дерева [21]. Но существенными недостатками являются отсутствие контроля приватности данных и отсутствие возможности импорта и экспорта генеалогических деревьев.

### “GenoPro”

“GenoPro” – приложение для построения генеалогических древ [22]. Интерфейс приложения показан на рисунке 1.2.5.

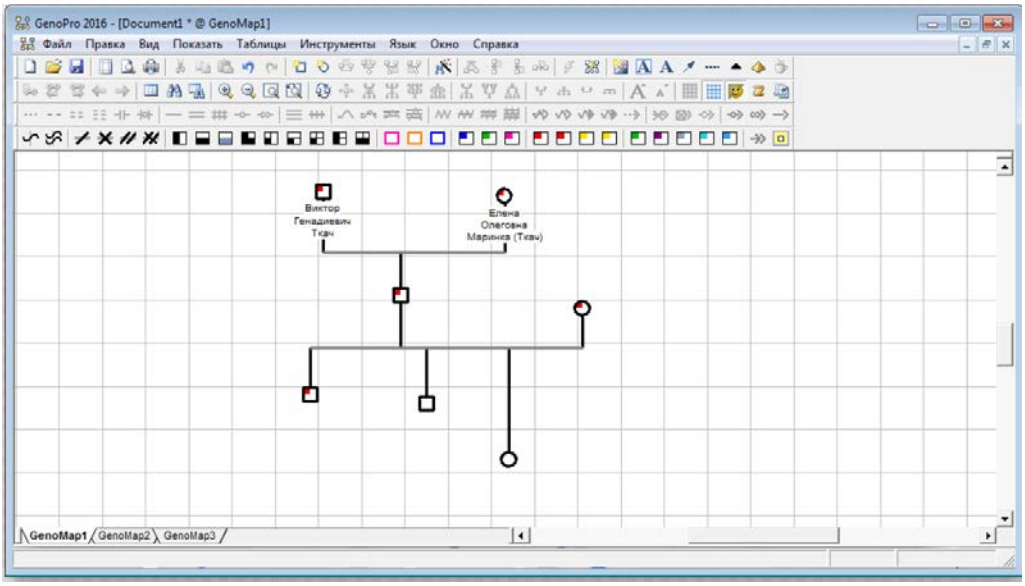


Рисунок 1.2.5 – Интерфейс приложения “GenoPro”

Приложение является многофункциональным. Также как многие рассмотренные раньше аналоги является «десктопным». Также приложение является коммерческим, его стоимость составляет 49$. Однако, многие пользователи отмечали, что заполнение древа неудобно, также нет возможности посмотреть древа других пользователей.

## ИТОГИ АНАЛИЗА

Анализ существующих аналогов в дальнейшем позволит реализовать более доработанную и продуманную систему, которая возьмёт все самое лучшее из существующих разработок, а также доработает те минусы, которые удалось заметить в ходе анализа.

Несмотря на большой выбор среди веб-приложений для построения генеалогических древ, среди всех аналогов можно выделить большое количество недостатков. Ни в одном аналоге не была предусмотрена возможность добавления родственника вне древа, когда человек точно знает о наличии очень далёкого родственника, но не знает, кто ему предшествовал или наоборот, не знает его детей.

Реализация системы поиска возможных родственников разнообразит функционал существующих аналогов, дав пользователям возможность найти своих дальних родственников.

В некоторых из аналогов была реализована функция чтения GedCom файлов, необходимо предусмотреть данную возможность и в создаваемом веб-приложении.

В данный момент более высокой популярностью пользуются веб-приложения, поэтому было принято решение начать разработку в направлении создания веб-приложения.

## 1.3 АНАЛИЗ ОСНОВНЫХ СУЩЕСТВУЮЩИХ ПЛАТФОРМ ДЛЯ СОЗДАНИЯ ПРИЛОЖЕНИЙ

Разработка веб-приложения – это процесс, в результате которого разрабатывается и создаётся приложение, способное работать на устройствах, у которых есть доступ к сети интернет и установлен некоторый браузер.

Для разработки веб-приложений существует огромное количество вариантов различных платформ, каждая из которых имеет свои плюсы и свои минусы. В конце концов каждый разработчик решает сам для себя, какой средой разработки ему пользоваться.

### 1.3.1 Visual Studio

Разрабатывать веб-приложения возможно с помощью известной для каждого программиста IDE – Visual Studio [15]. Первый выпуск Visual Studio датируется 1997 годом.

Microsoft Visual Studio — линейка продуктов компании Microsoft, которая включает в себя интегрированную среду разработки и ряд других средств.

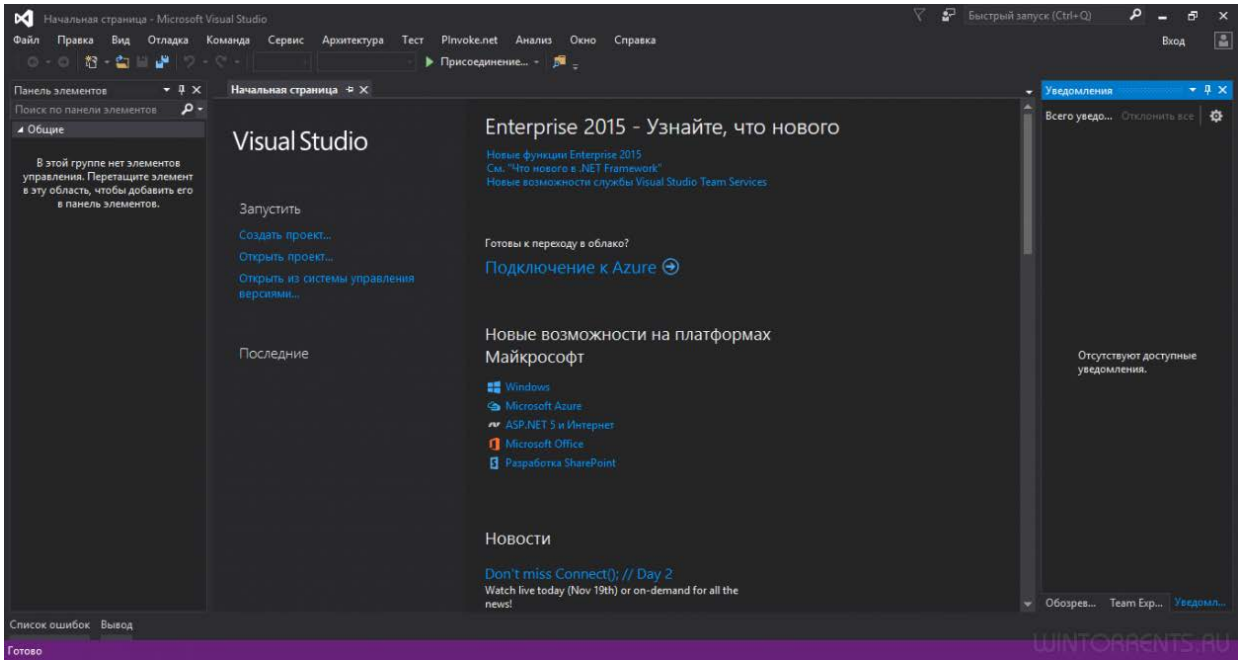


Рисунок 1.3.1.1 – Интерфейс IDE Visual Studio

Достоинства среды Visual Studio:

- добавление сторонних плагинов может позволить серьёзно расширить функциональность среды;
- встроенный веб-сервер;
- более высокая скорость разработки;
- возможность отладки.

Недостатки:

- данная среда создана конкретно для опытных разработчиков;
- невозможность отладчика отслеживать в коде режима ядра.

### 1.3.2 NetBeans

Также стоит выделить одного из гигантов в сфере разработки программный продукт – NetBeans. Это свободная интегрированная среда разработки приложений на языках программирования Java, PHP, Python и др.

Дата создания первой версии данного продукта – 1996 год.

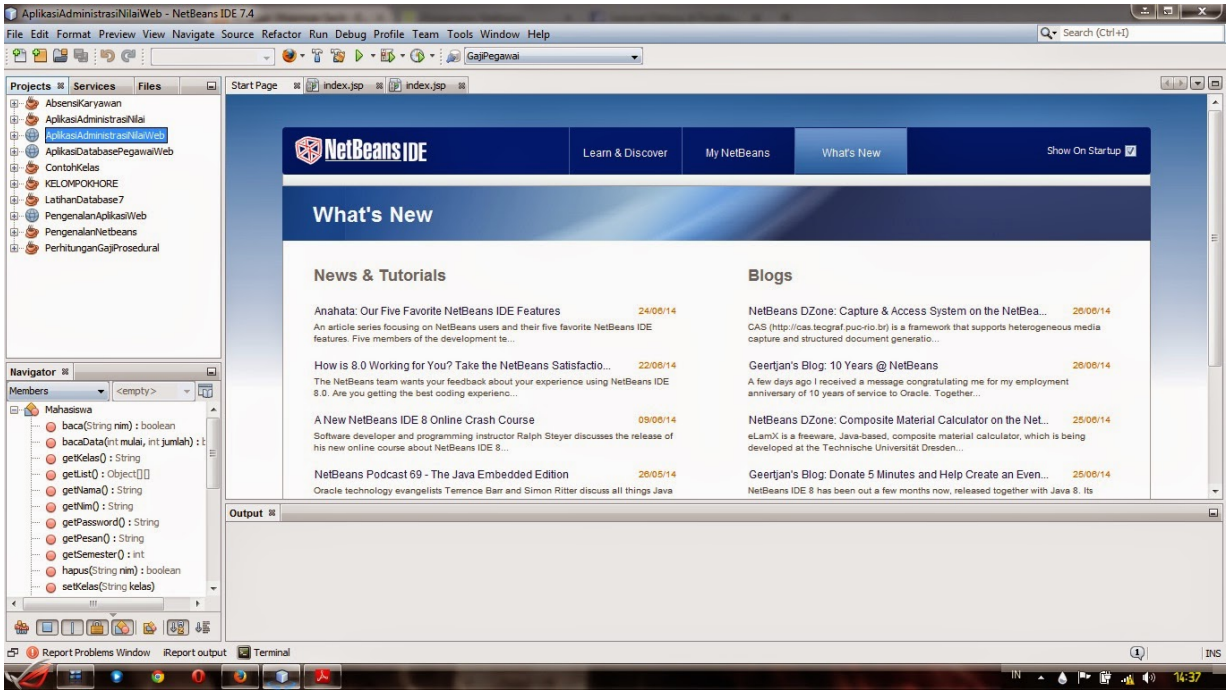


Рисунок 1.3.2.1 – Интерфейс IDE NetBeans

NetBeans является распространённой и очень мощной средой разработки на разных языках, таких как: Java, JavaScript, Python, PHP, C++ и др.

Как главный недостаток данной среды стоит выделить её невысокое быстродействие из-за идеи её концепции «всё в одном». На не самых сильных компьютерах данная IDE может серьёзно «глючить». Некоторые плагины имеют ограничение в функциональности.

### 1.3.3 Eclipse

Eclipse – свободная интегрированная среда разработки модульных кроссплатформенных приложений. Развивается и поддерживается компанией Eclipse Foundation. Первый выпуск этого программного продукта датирован 7 ноября 2001 года [20].

Одно из самых популярных решений для веб-разработчика. Также как и NetBeans, Eclipse является бесплатных программным обеспечением.

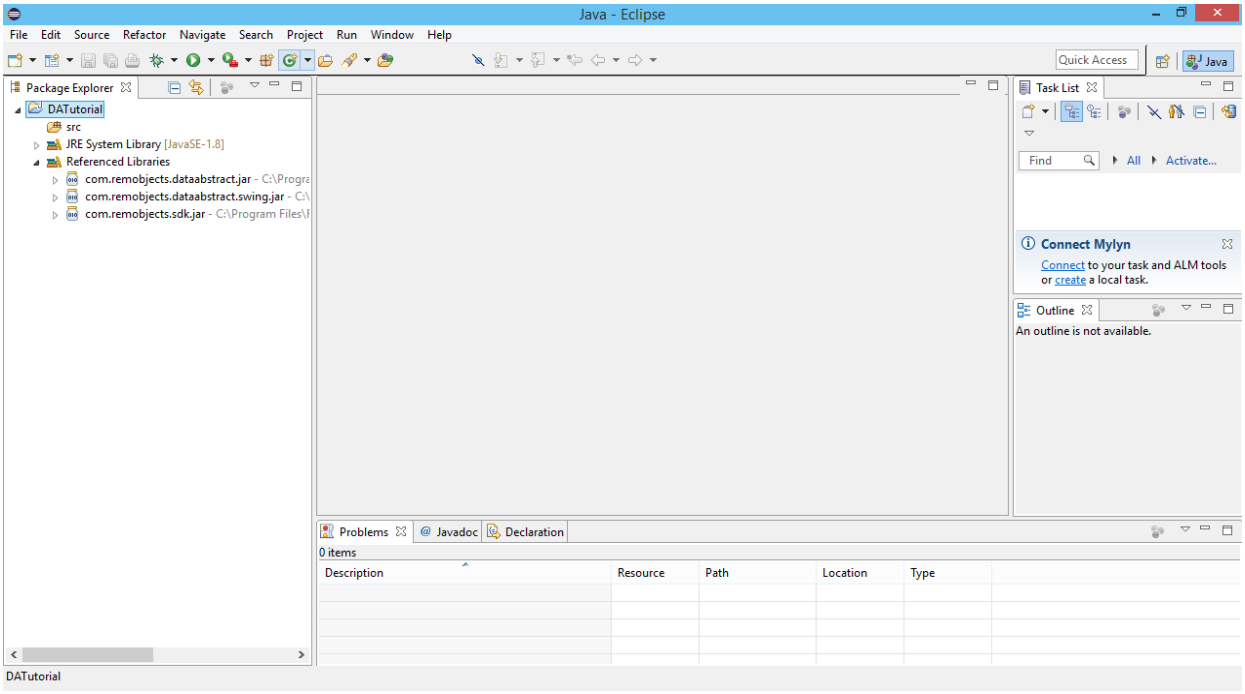


Рисунок 1.3.3.1 – Интерфейс IDE Eclipse

Преимущества среды Eclipse:

- с помощью данной среды можно создать не только веб-приложения, а ещё и продукты мобильного программного обеспечения;
- открытый код, любой желающий может создать расширение для IDE;
- поддержка многих языков программирования.

Недостатки:

- предварительная инсталляция Sun JDK;
- большие требования к компьютеру, в следствии чего происходит процесс очень долгой загрузки приложения;
- высокий порог вхождения;

### 1.3.4 Brackets

Brackets –текстовый редактор для веб-разработчиков, разработанный Adobe Systems. Редактор является кроссплатформенным, что позволяет использовать его на любой операционной системе windows, либо системе семейства Linux. Первая версия данного проекта была разработана позже всех вышеперечисленных. В самой первой версии продукта Brackets был представлен как обычный текстовый редактор. Дата первого выпуска – 2014 год.

Хоть Brackets и позиционируется как текстовый редактор, по факту он всё больше напоминает полноценную IDE [2].

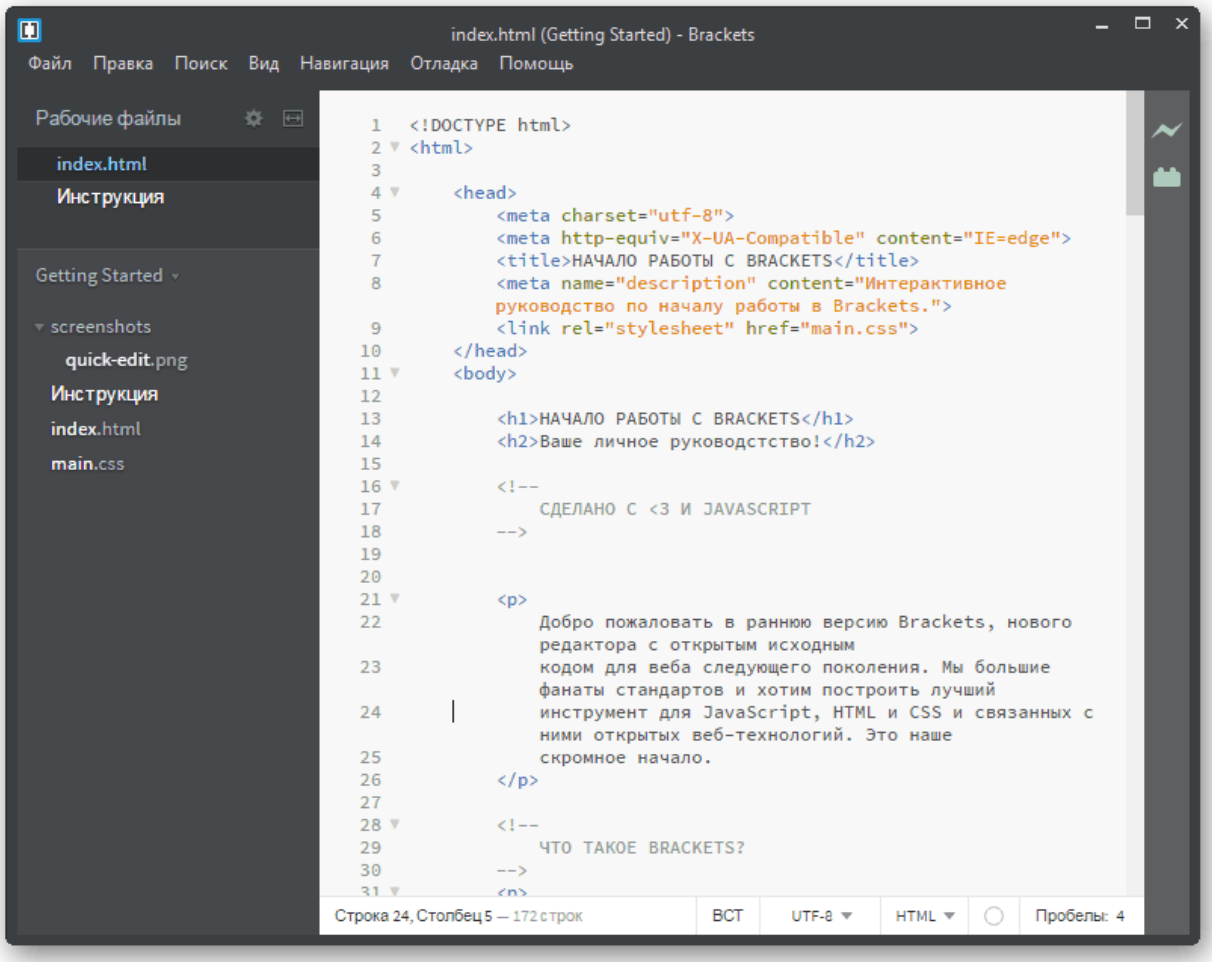


Рисунок 1.3.4.1 – Интерфейс Brackets

Преимущества среды Brackets:

- лицензия на использование данного средства программирования бесплатна;
- большое количество плагинов;
- возможность интерактивного просмотра.

Недостатки:

- слишком «молодой» проект;
- изначально был задуман как текстовой редактор и не является полноценной IDE;
- много плохо реализованных расширений.

### 1.3.5 PHPStorm

Компания JETBrains создала целый набор инструментов для веб-разработчиков, с самыми разными запросами, одним из которых является – PHPStorm[18].

PHPStorm – среда разработки с поддержкой PHP, JS и прочими языками веб-программирования.

PHPStorm разработан на основе платформы IntelliJ IDEA, написанной на Java. Пользователи могут расширить функциональность среды разработки за счет установки плагинов, разработанных для платформы IntelliJ, или написав собственные плагины.

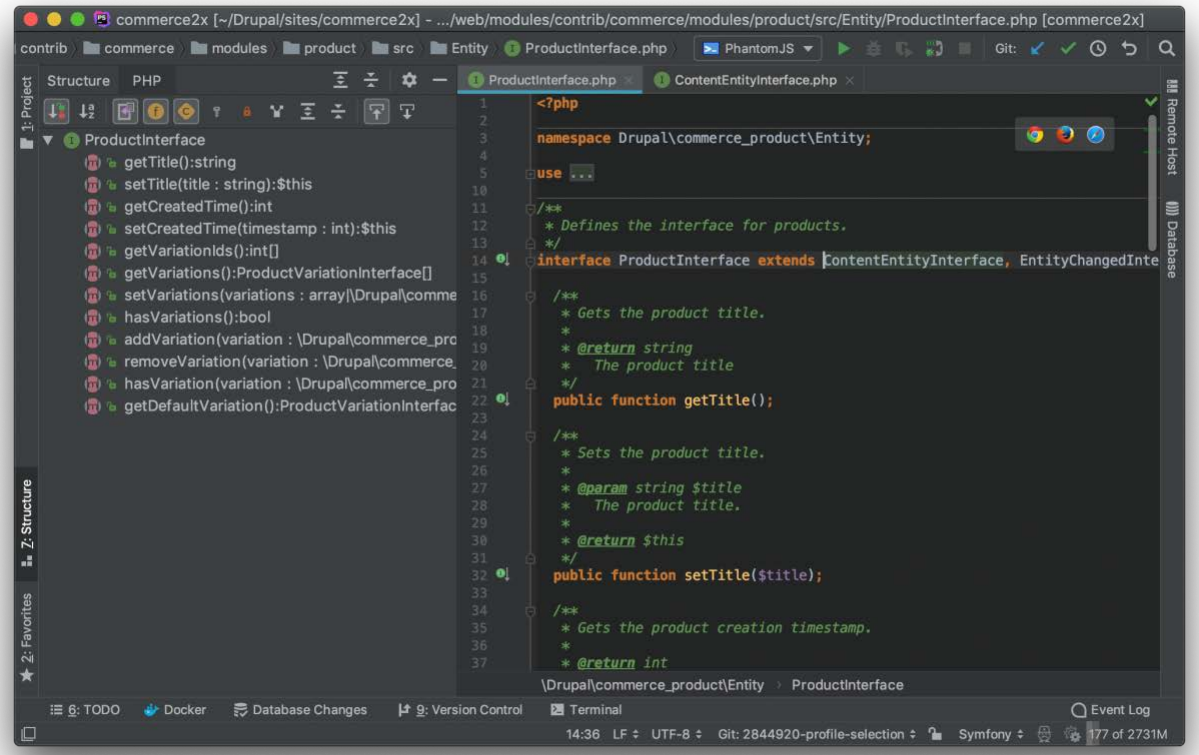


Рисунок 1.3.5.1 – Интерфейс PHPStorm

Преимущества среды PHPStorm:

- для студентов предоставляется лицензия на продукты JetBrains и выдаётся ровно на один календарный год;
- возможна разработка на многих языках программирования;
- широкий выбор возможностей для разработчика;
- автосохранение файлов;
- низкий порог вхождения;
- поиск классов;
- рефакторинг имён классов, файлов и методов во всём проекте и/или поиск их использования.

Недостатки:

- подвисание программы при индексации дополнительных папок к проекту;
- мало плагинов.

### 1.3.6 ВЫВОД ПО ВЫБОРУ ПЛАТФОРМЫ ДЛЯ РАЗРАБОТКИ

В ходе проделанного анализа был сделан вывод, что наиболее подходящей средой разработки является PHPStorm. Данная IDE была разработана специально для разработки веб-приложений. Возможность получить бесплатную лицензию как студент также является плюсом при выборе IDE. В PHPStorm не было выявлено много минусов, а нацеленность н0 разработку веб-приложений стала решающим критерием при выборе среды разработки.

### 1.3.7 ВЫБОР ЯЗЫКА ПРОГРАММИРОВАНИЯ И ФРЕЙМВОРКА

Перед тем как выбрать фреймворк для веб-приложения, следует выбрать язык программирования.

#### 1.3.7.1 C#

C# - это объектно-ориентированный язык программирования. Был разработан в 1998-2001 годах инженерами компании Microsoft как язык разработки приложений для платформы Microsoft.NET Fraamework [13].

C# относится к языкам с Си-подобным синтаксисом, его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов, делегаты, атрибуты , события и многое другое.

Достоинства языка:

- объектно-ориентированный подход;
- высокая скорость разработки;
- высокий уровень читаемости кода;
- наличие большого количества библиотек и шаблонов.

Недостатки языка:

- язык достаточно легко дизассемблируется;
- использование JIT-компиляции;

- не является повсеместно распространённым языком;
- работа приложений, написанных на данном языке только на ОС Windows.

#### 1.3.7.2 Java

Java – сильно типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Mycrosystems [14].

Достоинства языка:

- объектно-ориентированный подход;
- язык высокого уровня с простым синтаксисом;
- является стандартом для корпоративных вычислительных систем;
- безопасность;
- независимость от платформы.

Недостатки языка:

- платное коммерческое использование;
- низкая производительность;
- отсутствие нативного дизайна;
- многословный и сложный код

#### 1.3.7.3 PHP

Php – скриптовой язык общего назначения, интенсивно применяемый для разработки веб-приложений [6]. В настоящее время поддерживается подавляющим большинством хостинг-провайдеров и является одним из лидеров среди языков, применяющихся для создания динамических веб-сайтов. Язык и его интерпретатор разрабатываются группой энтузиастов в рамках проекта с открытым кодом.

Достоинства языка:

- высокая скорость работы (высокая общая производительность ресурсов);
- простота освоения;
- отличная совместимость и переносимость;
- высокая гибкость, ёмкость и функциональность;
- многозадачность.

Недостатки языка:

- не подходит для создания десктопных приложений;
- слабые средства для работы с исключениями;

#### 1.3.7.4 ВЫВОД ПО ВЫБОРУ ЯЗЫКА

Как известно, компилируемые языки программирования выполняют заданные перед ними задачи быстрее чем интерпретируемые и, стоит отметить, имеется достаточное количество уже готовых решений на базе компилируемых языков, таких как: C# и Java [3]. При наличии опыта программирования на C# было бы логично сделать выбор в сторону данного языка и фреймворка ASP.NET, но именно компилируемость языка стала важнейшим фактором при его выборе.

Не имея за спиной достаточное количество опыта по разработке масштабных проектов, хотелось иметь возможность постоянно иметь доступ к проектированию и тестированию небольших частей программы. Именно по этой причине было принято решение отказаться от компилируемых языков программирования.

Ситуация с интерпретируемыми языками в корне отличается. Интерпретируемые языки программирования позволяют разработчику увидеть результат написанных им пары строк кода, всего лишь обновив страницу браузера.

Самым подходящим языком программирования для разработки веб-приложения был выбран PHP. Данный язык имеет достаточно большое

количество фреймворков, язык постоянно обновляется, да и создавался он как раз для создания веб-приложений. Также стоит отметить низкий порог вхождения для начала использования данного языка.

### 1.3.7.5 ВЫБОР ФРЕЙМФОРКА

#### 1.3.7.5.1 Yii2

Рисунок 1.3.7.5.1.1 Логотип фреймворка Yii2

Для фреймворка уже разработано много библиотек, виджетов и расширений как основными, так и сторонними разработчиками. Это все позволяет быстро разрабатывать качественные приложения.

Данный фреймворк призывает к чистому написанию кода, без лишних связанностей, принципу “don’t repeat yourself” направленный на снижение повторения кода, событийно-ориентированное программирование, соглашение по конфигурации, по которому если класс соответствует соглашению наименованию, тогда он не нуждается в дополнительной конфигурации, а так же модульная структура приложения. Фреймворк Yii2 следует архитектурному паттерну – MVC (Model-View-Controller), или Модель-Представление-Контроллер, который обеспечивает разделение

бизнес-логики от логики представления, связанной с графическим пользовательским интерфейсом [6].

В шаблоне Yii2 существует три основных компонента:

Модель - каскад, вокруг которого работает приложение. Модели основаны на реальных предметах, таких как человек, событие, билет или продукт. Модели как правило являются постоянными и хранятся вне приложения, часто в базе данных. Модель – это последний рубеж для защиты и представления данных [6].

Представление - визуальное представление модели с учетом какого-либо контекста. Обычно это результирующая разметка, которую фреймворк передает браузеру, например, HTML страница, представляющая собой отображение информации о билете в браузере. Уровень представления отвечает за создание пользовательского интерфейса, обычно на основе данных в модели. Например, в интернет-магазине будет список продуктов, которые будут отображаться на экране каталога. Этот список будет доступен через модель, но это будет представление, которое обращается к списку из модели и форматирует его для конечного пользователя. Хотя представление может предоставлять пользователю различные способы ввода данных, само представление никогда не обрабатывает поступающие данные. Работа представления закончена после отображения данных [6].

Контроллер - связывающее звено, обеспечивающее связь между представлением и моделью. Контроллер отвечает за обработку входных данных, и принятие решения о том, какие действия должны быть выполнены, например, перенаправление на другую страницу [6].

Также в фреймворк встроена функция Gii, отвечающая за автоматическую генерацию кода, что может несколько облегчить разработку, не тратя время на описание общих блоков в программе.

Сайты, построенные по современным технологиям, имеют блочную структуру. Одни блоки повторяются на всех страницах, другие только на некоторых. Например, в разрабатываемом сайте на всех страницах выводится

навигационная панель в «шапке», наверху страницы, и информация о сайте в «подвале», внизу. Можно было бы отдельно прописывать код навигационной панели для каждой страницы, но в этом случае, если бы нам понадобилось внести изменения в этот блок, то пришлось бы эти изменения вносить в каждом файле, где встречается эта навигационная панель. Было бы проще и правильней с точки зрения архитектуры, вынести повторяющиеся блоки в отдельные файлы или классы. Инструмент, который позволяет использовать модули, или шаблоны, при разработке веб-страниц, называется шаблонизатор.

Yii2 поставляется с уже предусмотренным шаблонизатором, носящим название Bootstrap. В использовании Bootstrap можно выделить сразу ряд плюсов, таких как:

1. Высокая скорость вёрстки – большое количество готовых компонентов даёт возможность не останавливаться на каких-то мелочах.
2. Адаптивность – возможность настраивать размеры блоков сайта в зависимости от ширины устройства, как для компьютера так и для телефона, планшета и т.д.
3. Популярность – из-за которой существует большое количество статей и уроков, а также форумов. Поэтому по любому вопросу, в котором может сомневаться разработчик, можно найти ответ на просторах интернета, найти ответ в книгах или же написать на форум, на котором вероятнее всего возможно получить ответ на свой вопрос.

#### 1.3.7.5.2 LARAVEL

Laravel — это фреймворк для web-приложений с выразительным и элегантным синтаксисом [17]. Он позволит упростить решение основных наболевших задач, таких как аутентификация, маршрутизация, сессии и кэширование.


Рисунок 1.3.7.5.2.1 – Логотип фреймворка Laravel

Laravel — это попытка объединить всё самое лучшее, что есть в других PHP фреймворках. Основные преимущества Laravel:

- большая экосистема с мгновенным разворачиванием своей платформы. Официальный сайт предоставляет множество мануалов и информации для ознакомления;
- подробный набор документации;
- наличие собственного «движка» для шаблонов Blade;
- обширный функционал;
- безопасность баз данных;
- популярность и активное сообщество.

Недостатки:

- Реализация веб-проектов с нуля занимает гораздо больше времени в сравнении с другими доступными фреймворками;
- сложносоставленная для неопытного разработчика документация;
- плохая поддержка со стороны разработчиков;

Данный фреймворк также как и вышеописанный Yii2 построен по паттерну MVC, т.е. архитектурно во многом похож на Yii2.

### 1.3.7.5.3 ВЫВОД ПО ВЫБОРУ ФРЕЙМВОРКА

Стоит отметить, что у фреймворка Laravel есть ряд проблем в готовой версии продукта: очень часто при обновлении фреймворка происходит

нарушение обратной совместимости продукта, в следствии чего разработчик может столкнуться с рядом проблем [16].

Официальная документация фреймворка Laravel написана на английском языке, что также делает неудобным использование данного фреймворка, а у Yii2 присутствует официальная документация на русском языке, что увеличивает мобильность использования данного фреймворка.

Подведя итоги всего вышесказанного, можно прийти к выводу, что выбор Yii2 в качестве фреймворка для разработки веб-приложения поиска родственников является верным.

### 1.3.8 ВЫБОР СЕРВЕРА

В качестве сервера необходим был такой, чтобы имелась возможность в любой момент времени проверить написанный код в браузере без лишней траты на это времени. Также хотелось бы не тратить много времени на установку и настройку сервера.

Apache HTTP Server был разработан Робертом Маккулом в 1995 году, а с 1999 года разрабатывается под управлением Apache Software Foundation — фонда развития программного обеспечения Apache. Так как HTTP сервер это первый и самый популярный проект фонда его обычно называют просто Apache [5]. Веб-север Apache был самым популярным веб-сервером в интернете с 1996 года. Благодаря его популярности у Apache сильная документация и интеграция со сторонним софтом. Он может быть расширен с помощью системы динамически загружаемых модулей и исполнять программы на большом количестве интерпретируемых языков программирования без использования внешнего программного обеспечения, в том числе и ранее выбранном языке – PHP.

Преимущества:

- надёжное программное обеспечение;
- простота настройки;

- доступная поддержка и большое количество информации.

Недостатки:

- проблемы с производительностью;
- большое количество параметров конфигурации.

**Nginx** является программным обеспечением, разработанным для UNIX систем. Предназначен для создания и настройки HTTP-сервера [15]. Nginx используется на значительном количестве веб-серверов по всему миру.

Достоинства:

- шифрование, сжатие, поддержка многосайтовости на IP-адресе;
- простота конфигурации, масштабируемость;
- асинхронная система ввода-вывода.

В итоге, проведя анализ двух выше представленных серверов, можно прийти к выводу, что подходящим сервером для работы приложения является Apache.

### 1.3.9 FRONTEND

#### 1.3.9.1 ВЫБОР СРЕДСТВ РАЗРАБОТКИ

Для разработки клиентской части используются следующие средства:

- язык разметки HTML;
- таблицы каскадных стилей CSS;
- язык программирование JavaScript с библиотекой jQuery;
- фреймворк Bootstrap 4, использующий современные наработки в области CSS и HTML.

#### 1.3.9.2 jQUERY

jQuery – это библиотека JavaScript, основанная на принципе «пиши меньше, делай больше». Это инструмент, который упрощает написание общих задач JavaScript. Кроме того, jQuery обладает кроссбраузерной

совместимостью, а значит, можно быть уверенным в том, что любой современный браузер корректно отобразит вывод программы.

#### 1.3.9.3 BOOTSTRAP 4

Главным элементом Bootstrap является адаптивная сетка. В общем-то, без нее фреймворк утратил бы практически всю свою ценность, потому именно благодаря сетке можно быстро создавать адаптивные шаблоны. При этом можно вообще не быть знакомым с медиа-запросами, они не нужны, потому что фреймворк берет на себя реализацию адаптивности, нужно лишь задать блокам правильные классы.

### 1.3.10 ВЫБОР СУБД

Данные можно хранить в файлах или базах данных (БД). Особенности организации данных в БД по сравнению с файловыми системами:

- БД обеспечивают использование одних и тех же данных в различных приложениях;
- БД сводят к минимуму дублирование данных, прибегая к дублированию только для ускорения доступа к данным или для обеспечения восстановления БД при ее разрушении;
- одна из важных черт БД – независимость данных от особенностей прикладных программ, которые их используют, а также возможность создания этих программ в такой форме, что изменение особенностей хранения, логической структуры или значений данных не требует изменения программ их обработки;
- возможность изменения физических особенностей хранения данных без изменения их логической структуры.

Существует множество систем управления базами данных, которые можно использовать для веб-приложений, например MySQL, Microsoft SQL Server, PostgreSQL, MongoDB.

Недостатки Microsoft SQL Server:

- является тяжеловесной;
- даже при тщательной настройке производительности СУБД может занять все доступные ресурсы;
- есть проблемы с использованием службы интеграции для импорта файлов;

Microsoft SQL Server идеально подходит для крупных организаций, которые уже используют ряд продуктов Microsoft.

Недостатки PostgreSQL:

- неразборчивая документация;
- сложная конфигурация;
- скорость работы может падать во время проведения пакетных операций или выполнения запросов чтения.

PostgreSQL идеально подходит для проектов с ограниченным бюджетом, но квалифицированными специалистами.

Недостатки MongoDB:

- SQL не используется в качестве языка запросов;
- программа установки может занять много времени;

MongoDB подходит для проектов с разнородными данными, которые тяжело поддаются классификации. Для внедрения потребуются высококлассные специалисты.

Преимущества MySQL:

- распространяется бесплатно;
- понятная документация;
- простая установка;
- поддерживает набор пользовательских интерфейсов;
- является распространенной на хостингах;
- хорошая поддержка.

MySQL идеально подходит для проектов, которым требуется надежный инструмент управления базами данных, но бесплатный.

Таким образом, при разработке веб-приложения будет использоваться СУБД MySQL.

## 1.4 ВЫВОД

Исходя из всего вышесказанного, требуется создать веб-приложение, которое будет отличаться от уже имеющихся аналогов, создаваемое приложение не должно потерять в функционале, а совсем наоборот, при разработке необходимо увеличить функционал и сделать приложение куда более функциональным, чем аналоги на данный момент.

В нынешних условиях разработки никак нельзя не воспользоваться уже готовыми решениями – фреймворками, для более быстрой и качественной разработки проекта. Был выбран фреймворк – Yii2.

Необходимо предусмотреть возможность работы приложения с файлами формата GedCom, т.к. это позволит пользователям «безболезненно» перенести генеалогическое древо с других приложений. Это обеспечит более многочисленный прилив пользователей к данному приложению.

# 2 ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ К ВЕБ-ПРИЛОЖЕНИЮ

Для реализации данного веб-приложения необходим следующий набор подсистем:

- HTTP-сервер Apache с веб-интерфейсом в виде набора PHP-скриптов;
- MySQL – база данных, обеспечивающая хранение данных о пользователях и созданных ими деревьях;
- взаимодействие веб-приложения с файлами формата GedCom, преобразование записей в файлах GedCom в данные таблиц базы данных веб-приложения;
- графический интерфейс клиентского приложения;
- графический интерфейс администратора;
- графический интерфейс пользователя.

### Назначение приложения

- Построение генеалогических деревьев.
- изменение генеалогических деревьев;
- поиск родственников и возможных родственных связей;
- загрузка деревьев на сайт в специально предназначенном для этого формате GedCom для удобства заполнения дерева;
- просмотр деревьев других пользователей;
- поиск возможных родственников, расчёт возможности родства.

## 2.1 ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

Основные требования к функционалу системы:

- нахождение всей информация об уже присутствующих в базе данных деревьях и людях в одном приложении;
- возможность построения генеалогических деревьев;

- считывание древа в формате GedCom;
- возможность загружать GedCom файлы;
- включение узла в дерево без связи с каким-либо другим узлом (в случае, когда не точно известны данные о человеке и его связях);
- поиск знакомых людей и просмотр их древ;
- критерии поиска родственников:
  - совпадение фамилии, имени, даты и места рождения;
  - наличие одинаковых связей между узлами в разных деревьях (одинаковые родители, сравниваются только фамилии и имена).

В узле генеалогического древа должна храниться следующая информация:

- идентификатор человека;
- фамилия;
- имя;
- дата рождения;
- место рождения;
- пол;
- жив ли человек;
- идентификаторы родителей;
- идентификатор семьи;
- идентификатор супруга (при наличии);
- фотография.

Поля, содержащие неточную информацию, должны иметь специальную пометку.

Возможно отсутствие информации в любом из полей, кроме идентификатора человека (ключевое поле), для формирования узла должно быть заполнено хотя бы одно поле (при создании всегда автоматически будет сгенерирован идентификатор человека, все остальные поля могут быть пустыми).

## 2.2 НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

### 2.2.1 Требования к пользователям

Каждый пользователь, работающий в системе должен зарегистрироваться в системе и заполнить следующие обязательные поля в регистрационной форме:

- e-mail пользователя;
- пароль для входа в систему;

В системе должны быть выделены следующие типы пользователей:

- Незарегистрированный пользователь;

- зарегистрированный пользователь;

Незарегистрированный пользователь не должен выполнять никаких действий, кроме попытки авторизоваться в системе или зарегистрироваться, а также он будет иметь возможность просмотра главной страницы «визитки» веб-приложения.

Зарегистрированный пользователь – это пользователь, который имеет учетную запись и успешно произвёл авторизацию в системе, либо прошёл регистрацию.

Каждый зарегистрированный в системе пользователь после авторизации может начать работу в системе с создания генеалогического древа.

Для этого пользователь должен ввести название древа и установить разрешение или запрет на показ древа другим пользователям.

Далее пользователь имеет возможность начать заполнять информацию о людях и их связях в древе. Пользователь может заполнить следующие поля:

- имя;
- фамилия;
- дата рождения;
- место рождения;

- пол;
- фотография;
- информация о смерти человека;

Авторизованный пользователь, при наличии уже заполненного у него генеалогического древа, имеет возможность воспользоваться автоматическим поиском родственников по другим деревьям, находящимся в системе. Для этого пользователю достаточно перейти по ссылке на поиск и после него посмотреть результаты поиска.

Также если пользователь имеет заполненный GedCom файл с информацией о своей семье, он может загрузить этот файл в веб-приложение, а приложение уже без участия пользователя совершает парсинг файла и загружает в базу данных информацию о семействе пользователя.

### 2.2.2 Требования к системе безопасности

Система должна обладать следующими средствами обеспечения безопасности:

- пользователь при входе в систему обязан указывать свою почту и пароль;
- пароль должен храниться в зашифрованном виде.

### 2.2.3 Требования к навигации сайта

На каждой странице сайта должны быть ссылки на:

- главную страницу;
- поиск родственников;
- моё древо;
- выход из пользователя.

### 2.2.4 Требования к языковым версиям

Приложение должно быть доступно на русском языке.

# 3 ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ

## 3.1 АРХИТЕКТУРА ПРЕДЛАГАЕМОГО РЕШЕНИЯ

Для разработки данного веб-приложения использовался фреймворк “Yes It Is 2”, который использует паттерн MVC для организации кода. Данная организация позволяет полностью разделить код веб-приложения на 3 составляющих: Представление (View), Контроллер (Controller), Модель (Model). У каждого из составляющих есть свои задачи и обязанности. Логика приложения отделена от клиентской составляющей. На рисунке 3.1 представлена подробная схема паттерна MVC.

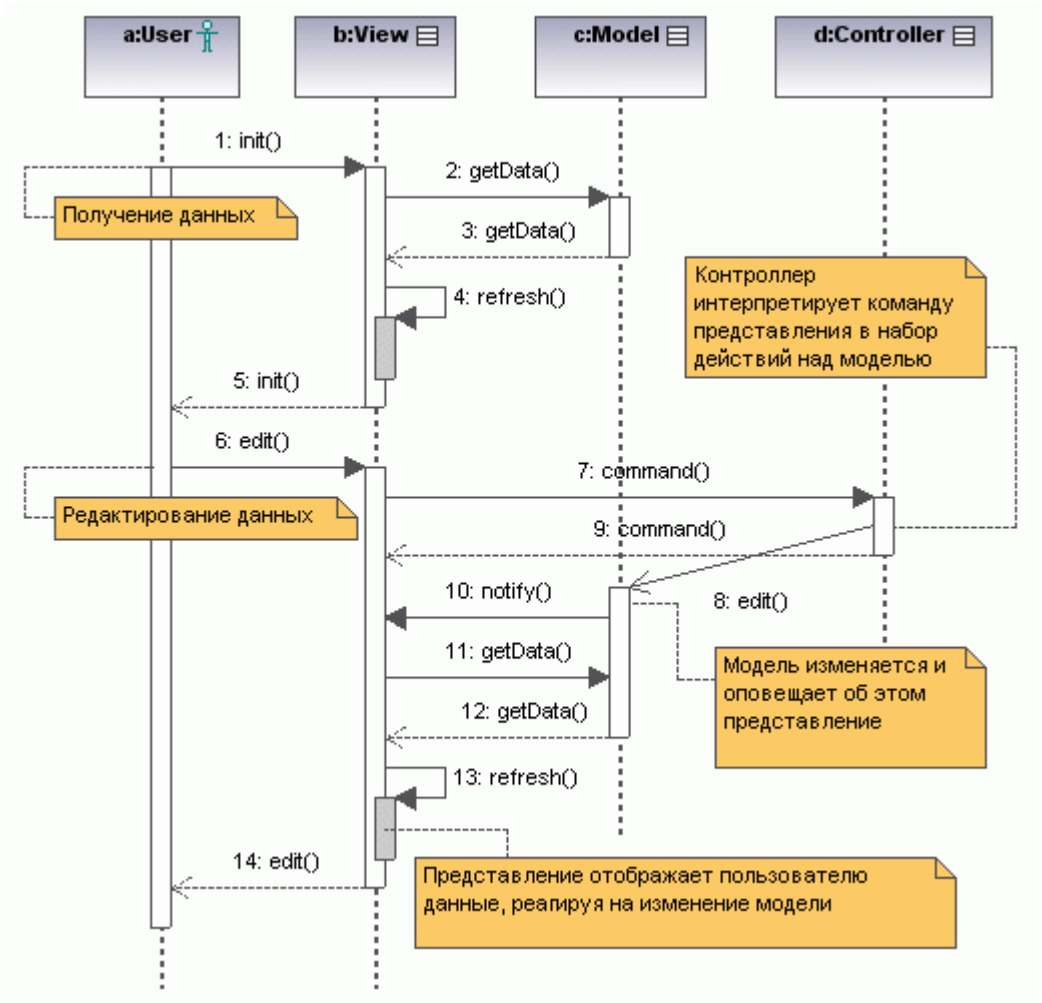


Рисунок 3.1.1 – Подробная схема паттерна MVC

Разработанное веб-приложение работает на фреймворке Yii2, который в свою очередь включает в себя паттерн MVC. Вся работа сводится к обмену и обработке данных между представлениями, моделями и контроллерами.

В таблице 3.1.1 представлены представления приложения «DNA»

Таблица 3.1.1 – Представления веб-приложения «DNA»

| **Контроллер** | **Наименование представления** | **Назначение** |
|---|---|---|
| Site | Index | Главная страница |
| | Login | Отображение формы авторизации |
| | signup | Отображение формы регистрации |
| Family | _form | Форма заполнения данных древа |
| | _search | Отображение поиска древа |
| | create | Отображение страницы создания древа |
| | index | Главная страница |
| | update | Страница редактирования древа |
| | toPerson | Страница перехода от создания древа к его заполнению |
| | view | Страница показа данных древа |
| Person | _form | Форма заполнения данных о человеке |
| | createDad | Форма заполнения данных об отце |

Продолжение таблицы 3.1.1

| **Контроллер** | **Наименование представления** | **Назначение** |
|---|---|---|
| Person | create | Страница создания записи о новом человеке |
| | index | Главная страница |
| | createMom | Форма заполнения данных о матери |
| | view | Страница показа данных о человеке |
| | update | Страница редактирования данных о человеке |
| | _search | Форма для поиска человека |
| | createSpouse | Страница заполнения данных о человеке |
| Search | index | Отображение результатов поиска родственников |

В таблице 3.1.2 представлены контроллеры приложения «DNA»

Таблица 3.1.2 – Контроллеры приложения «DNA»

| **Наименование** | **Назначение** | **Количество action, шт.** | **Взаимодействие с другими компонентами** | |
|---|---|---|---|---|
| | | | **View** | **Model** |
| Site | Работа с пользователями | 4 | • index<br>• signup<br>• login | • Login<br>• Signup<br>• User |

Продолжение таблицы 3.1.2

| **Наименование** | **Назначение** | **Количество action, шт.** | **Взаимодействие с другими компонентами** | |
|---|---|---|---|---|
| | | | **View** | **Model** |
| Family | Работа с древами | 6 | • _form<br>• _search<br>• create<br>• index | • Family<br>• FamilySearch |
| Person | Работа с людьми в древе | 8 | • _search<br>• createDad<br>• create<br>• index<br>• createMum<br>• update<br>• view<br>• createSpouse | • Person<br>• PersonSearch |
| Behaviors | Разделение прав пользователей | 0 | От данного контроллера наследуются все остальные контроллеры | |
| Search | Работа с алгоритмом поиска | 1 | Отображение результатов алгоритма поиска | |

Продолжение таблицы 3.1.2

| **Наименование** | **Назначение** | **Количество action, шт.** | **Взаимодействие с другими компонентами** | |
|---|---|---|---|---|
| | | | **View** | **Model** |
| Parser | Парсинг GedCom файлов | 0 | | • Parser |

В таблице 3.1.3 представлены модели приложения «DNA»

Таблица 3.1.3 – Модели приложения «DNA»

| **Наименование** | **Назначение** |
|---|---|
| Family | Обработка действий с древом |
| FamilySearch | Обработка поиска древа |
| Login | Авторизация пользователя |
| Person | Обработка действий с человеком |
| PersonSearch | Обработка поиска человека |
| Signup | Регистрация пользователя |
| User | Обработка действий с аккаунтом пользователя |
| Search | Обработка действий с алгоритмом поиска родственников |
| Parser | Парсинг GedCom файла |

## 3.2 ОПИСАНИЕ ДАННЫХ

В качестве входных данных в веб-приложении используются данные, которые пользователь вводит самостоятельно в поля, находясь непосредственно в приложении.

Для хранения информации, необходимой для работы приложения, была составлена база данных с помощью СУБД MySQL. Были разработаны и введены поля и таблицы, показанные в таблице 3.2.1.

Таблица 3.2.1 – Состав базы данных веб-приложения

| **Название таблицы** | **Название поля** | **Тип поля** | **Назначение поля** |
|---|---|---|---|
| User | id | int | Идентификатор пользователя |
| | email | varchar | Почта пользователя |
| | password | varchar | Личный пароль пользователя (хранится в зашифрованном виде) |
| Family | ID_family | int | Идентификатор семьи (древа) |
| | name | varchar | Название древа |
| | ID_user | int | Идентификатор пользователя, которому принадлежит древо |
| | access | bool | Разрешение на просмотр древа |
| | gedcom | varchar | Поле содержит GedCom файл |
| Person | ID_person | int | Идентификатор человека |

Продолжение таблицы 3.2.1

| **Название таблицы** | **Название поля** | **Тип поля** | **Описание поля** |
|---|---|---|---|
| Person | first_name | varchar | Имя человека |
| | last_name | varchar | Фамилия человека |
| | date_of_birth | date | Дата рождения |
| | place_of_birth | varchar | Место рождения |
| | live | Boolean | Жив ли человек |
| | photo | text | Фотография человека |
| | gender | Boolean | Пол |
| | ID_family | int | Идентификатор древа, к которому принадлежит человек |
| | ID_father | int | Идентификатор отца |
| | ID_mother | int | Идентификатор матери |
| | ID_spouse | int | Идентификатор супруга |
| | ID_ged | varchar | Идентификатор человека в GedCom файле |
| | FAMS_ged | varchar | Идентификатор семьи, где человек является супругом |

Продолжение таблицы 3.2.1

| **Наименование** | **Название поля** | **Тип поля** | **Описание поля** |
|---|---|---|---|
| Person | FAMC_ged | varchar | Идентификатор семьи, где человек является ребёнком |
| Search | id_search | int | Идентификатор поиска между деревьями |
| | id_family_searching | int | Идентификатор древа, от которого пришел запрос на добавление |
| | id_family_found | int | Идентификатор найденного древа |
| | value | int | Возможность родства в процентах |

На рисунке 3.2.1 показана схема базы данных веб-приложения.

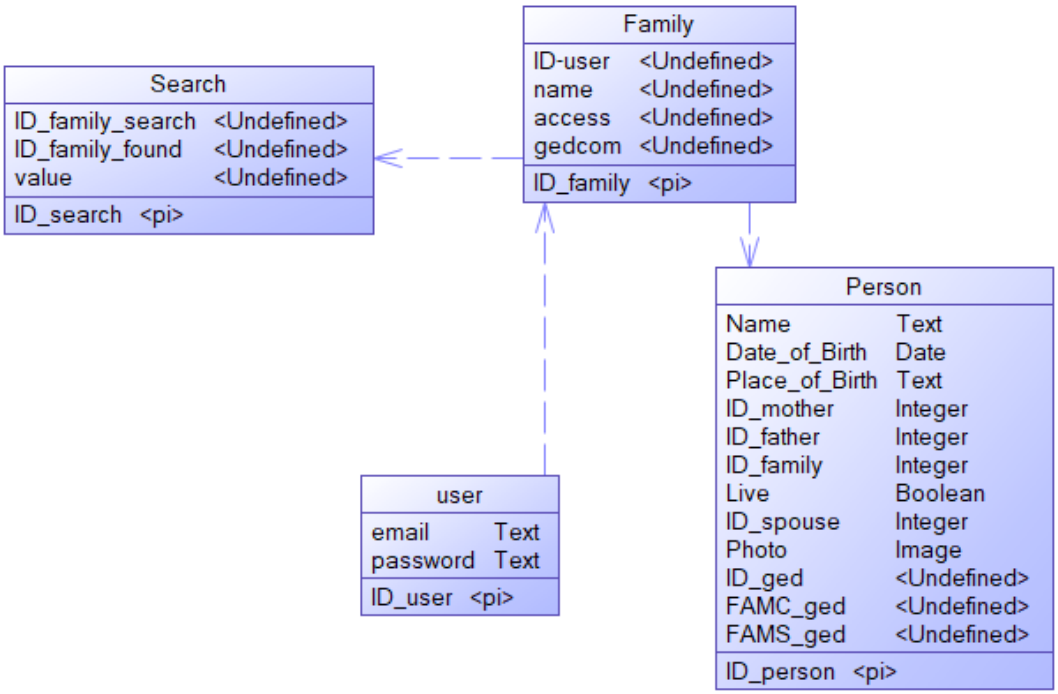


Рисунок 3.2.1 – Схема базы данных

## 3.3 АЛГОРИТМ ПОИСКА ВОЗМОЖНЫХ РОДСТВЕННИКОВ

Для возможности глобального поиска родственников и возможных родственных связей было необходимо разработать специальный алгоритм.

Данный алгоритм работает следующим образом – последовательно проверяется соответствие каждого члена каждого древа с каждым человеком в древе пользователя, в ходе этой проверки происходит расчёт процентовки для каждой связи и идёт подсчёт количества этих связей. Когда проверка закончена, число насчитанных процентов родства делятся на количество таких подсчётов. По итогам последнего расчёта выводится результат расчтёта как возможность родства между двумя древами.

Блок-схема алгоритма поиска родственников показана на рисунке 3.3.1.

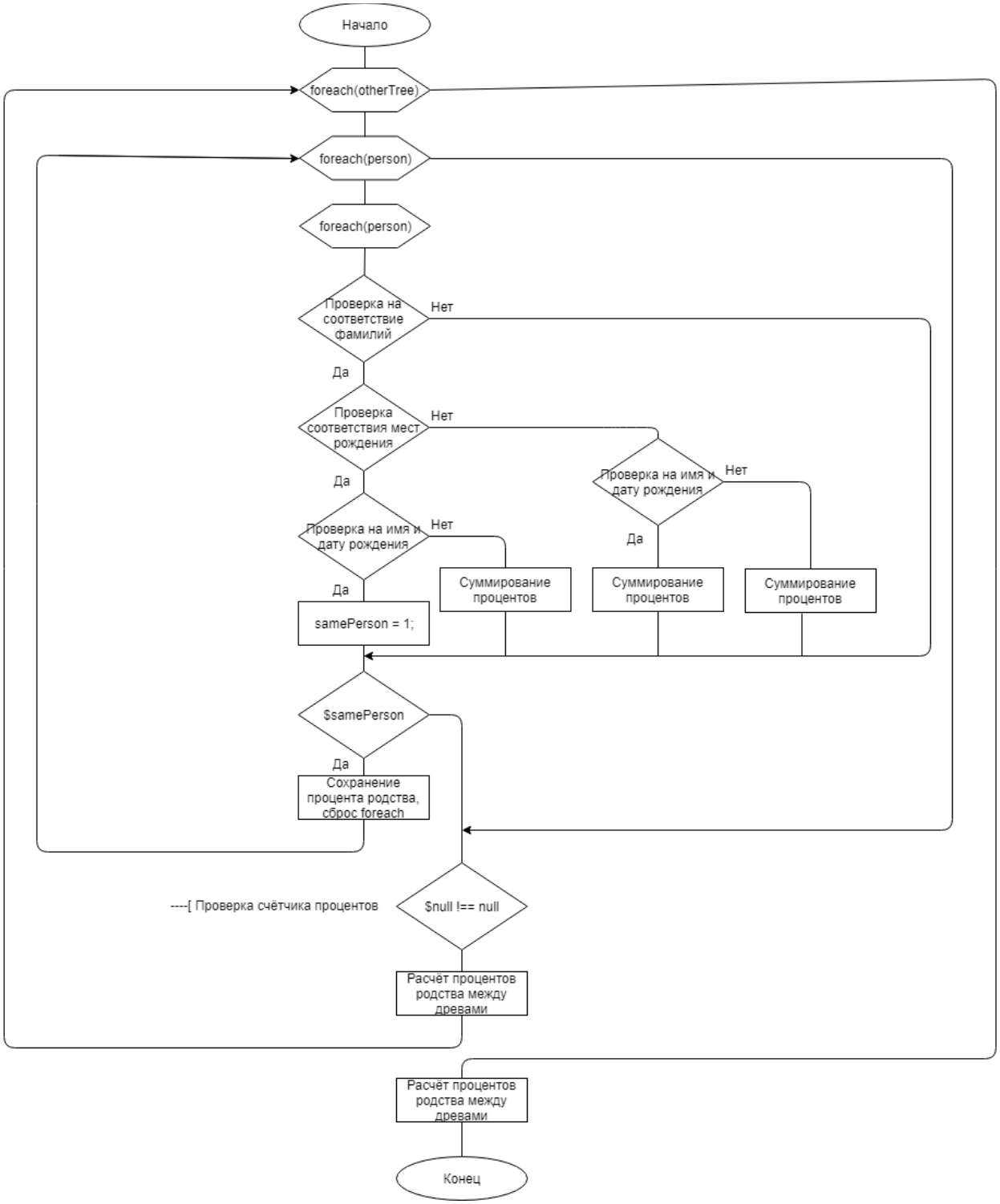


Рисунок 3.3.1 – Алгоритм поиска родственников

Для вычисления алгоритма также была продумана процентовка возможности родства на основе изучения всех возможных вариантов.

В таблице 3.3.1 показаны все варианты процентовки результатов поиска

Таблица 3.3.1 – Процентовка результатов поиска родственников

| **Количество процентов, %** | **Причина выставления процента родства** |
|---|---|
| 15 | Совпадение фамилий |
| 70 | Совпадение фамилий и места рождения |
| 99 | Совпадение людей в двух разных деревьях |

Данная процентовка была получена из статей учёных [23]. Формула расчёта была составлена с уклоном на те же труды.

## 3.4 АЛГОРИТМ СЧИТЫВАНИЯ ФАЙЛОВ GEDCOM

Для удобства перехода пользователей с других приложений необходимо облегчить данный процесс.

Для того, чтобы пользователю не нужно было своими руками вводить информацию о каждом человеке в древе, было принято решение реализовать поддержку чтения GedCom файлов – всемирного стандарта для работы с генеалогическими древами.

На рисунке 3.4.1 продемонстрирован пример содержания GedCom файла.


Рисунок 3.4.1 – Содержание GedCom файла

Разработанный парсер поочерёдно анализирует каждую строку GedCom файла, собирая информацию о человеке. Когда информация о человеке в файле заканчивается в программе создаётся объект с параметрами из файла и этот самый объект записывается в базу данных. Так происходит считывание информации о людях.

Когда происходит считывание информации о связях людей, также происходит считывание информации из файла и последующая запись данных через объекты в базу данных.

# 4 РЕАЛИЗАЦИЯ

## 4.1 РЕАЛИЗАЦИЯ ИНТЕРФЕЙСОВ

Для того, чтобы пользователю начать полноценно взаимодействовать с приложением, ему необходимо авторизоваться на сайте, пройдя для этого процедуру регистрации (при первом использовании веб-приложения или при желании завести новый аккаунт) и авторизации.

На рисунках 4.1.1, 4.1.2 представлена реализация регистрации и авторизации на сайте.

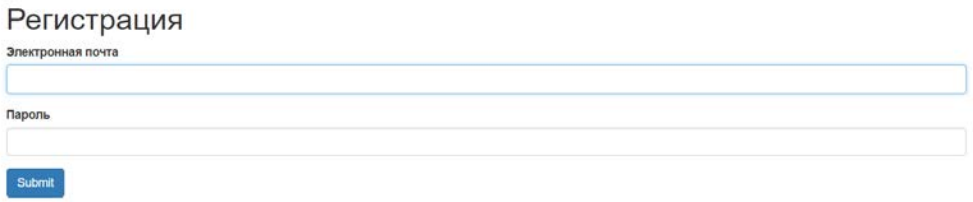


Рисунок 4.1.1 – Экранная форма регистрации


Рисунок 4.1.2 – Экранная форма авторизации

Исходный код контроллера для данных функций приведён в приложении А.

Записи новых пользователей в базу данных после регистрации создаются в таблице user, с помощью модели User, с использованием следующих функций:

- actionSignup() – в контроллере SiteController. Данный экшн вызывается, при запросе пользователя перейти к форме регистрации.
- rules() – в модели Signup. Используется для установки правил для форм ввода, валидация происходит при отправке данных.

- signup() – в модели Signup. Используется для передачи данных в объект для записи в базу данных.
- setPassword() – в модели User. Используется для зашифровки введённого пользователем пароля.

При авторизации используются следующие функции:

- actionLogin() – в контроллере SiteController. Данный экшн вызывается, при запросе пользователя перейти к форме авторизации.
- validatePassword() – в модели Login. Используется для нахождения в базе данных пользователя с такими же данными почты и пароля, что ввёл пользователь при авторизации.
- getUser() – в модели Login. Используется для получения информации из базы данных.

Также после авторизации в приложении, пользователю будет дана возможность выйти из своего профиля (см. рисунок 4.1.3).

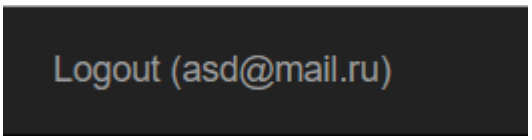


Рисунок 4.1.3 – Кнопка для выхода из профиля

При запросе пользователя выхода из профиля используется функция:

- actionLogout() – в контроллере SiteController.

Далее пользователю будет дана возможность создать своё древо, заполнив лишь одно поле – Название древа (см. рисунок 4.1.4).


Рисунок 4.1.4 – Создание древа

При создании древа используются следующие функции:

- actioncCreate() – в контроллере FamilyController. Данный экшн вызывается при переходе пользователя к странице создания древа.
- getUser() – в модели Family для привязки таблицы Family к таблице User через идентификатор пользователя.

В приложении И указан контроллер Family.

На рисунке 4.1.5 показано древо пользователя с возможностью перейти в него.

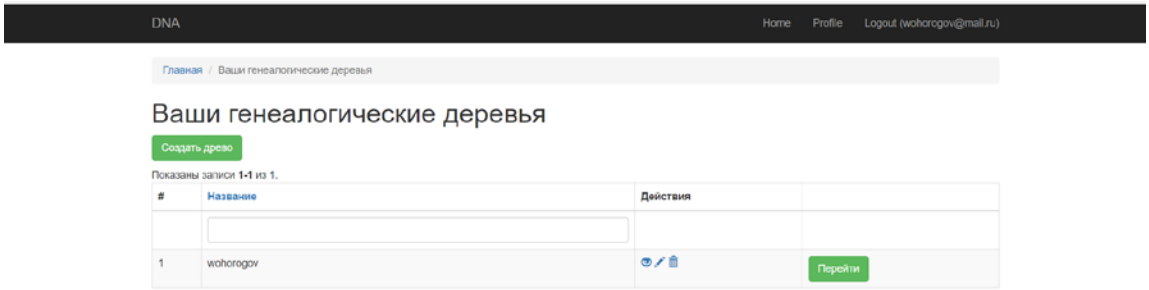


Рисунок 4.1.5 – Список генеалогических древ с возможностью перехода в него

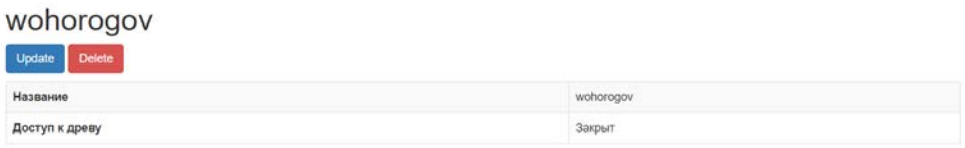


Рисунок 4.1.6 – Показ информации о древе

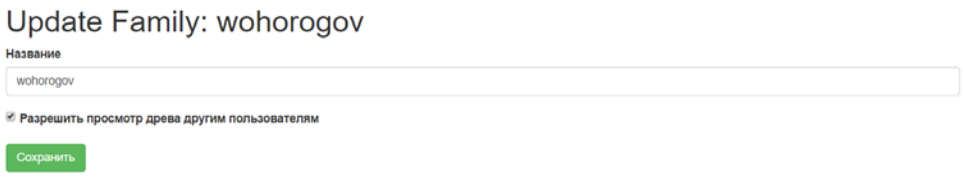


Рисунок 4.1.7 – Редактирование информации о древе

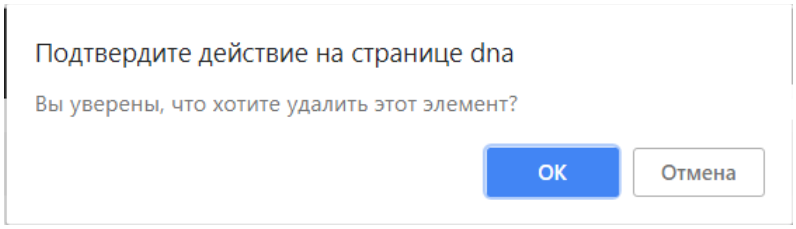


Рисунок 4.1.8 – Удаление древа

После перехода в состав древа пользователь имеет возможность увидеть всех родственников, уже находящихся в древе (см. рисунок 4.1.6).

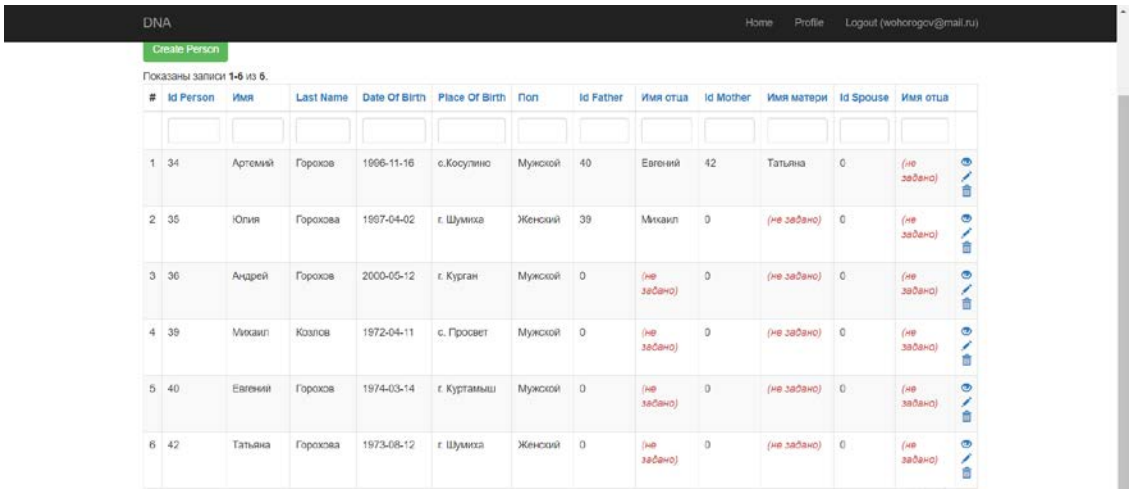


Рисунок 4.1.9 – Отображение людей, находящихся в древе

В приложении Е находится листинг контроллера Person.

При запросе увидеть графическое представление генеалогического древа пользователь увидит древо, показанное на рисунке 4.1.10.

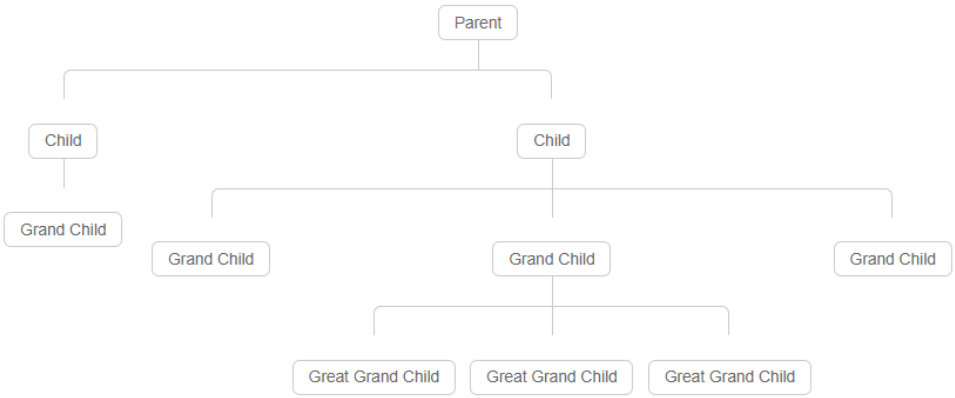


Рисунок 4.1.10 – Графическое представление генеалогического древа

Далее пользователь может начать заполнение древа, заполнив все формы, показанные на рисунке 4.1.11.

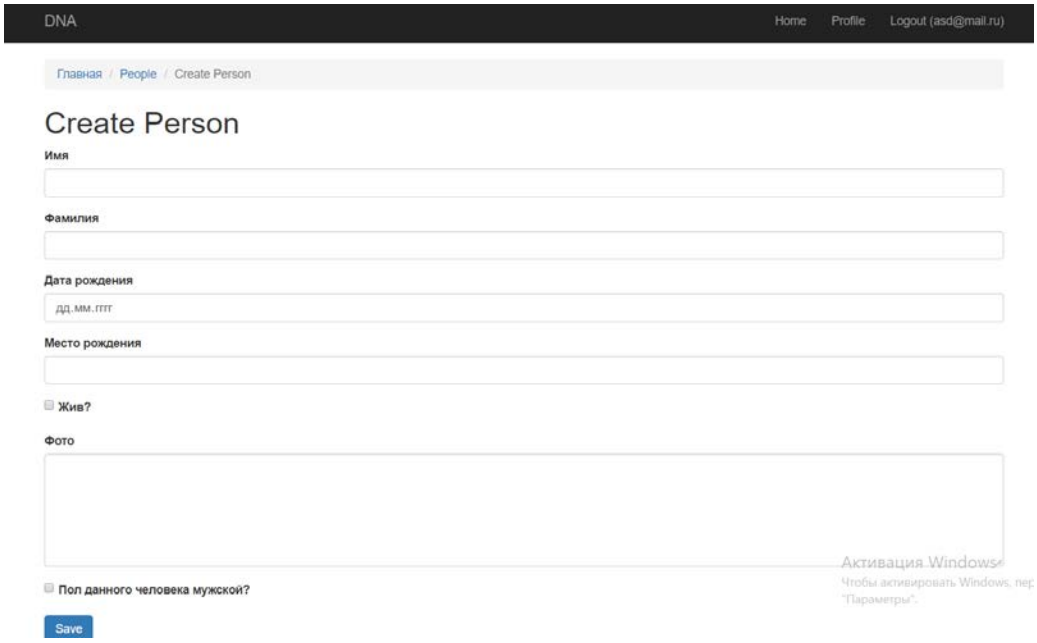


Рисунок 4.1.11 – Заполнение данных о человеке

При заполнении данных о пользователе используются следующие функции:

- actionCreate() – в контроллере PersonController. Данный экшн вызывается при запросе пользователя к заполнению информации о человеке.
- createPerson() – в модели Person. Используется для передачи данных в объект, с последующей записью этого объекта в базу данных
- getFamily() – в модели Person. Используется для привязки человека к древу через идентификатор ID_family.

На рисунке 4.1.12 показан результат записи данных о человеке.

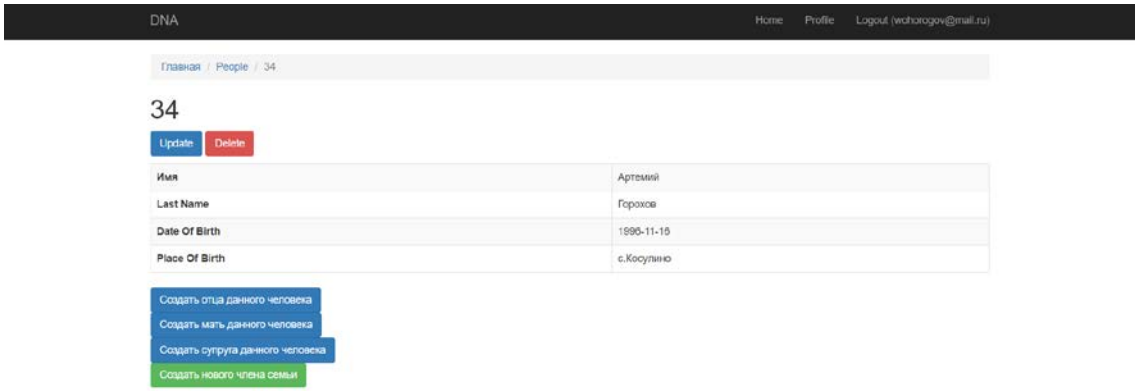


Рисунок 4.1.12 – Заполненный профиль пользователя

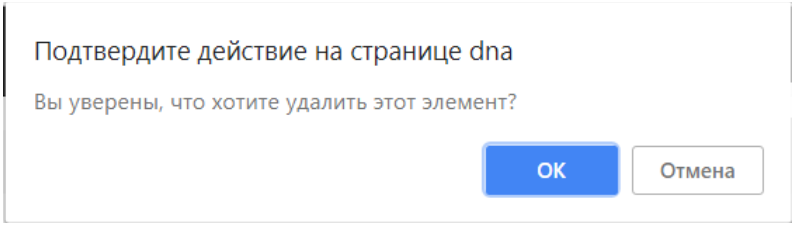


Рисунок 4.1.13 – Удаление человека из древа

При желании пользователя создать какого-либо близкого пользователя для уже имеющегося человека, пользователь будет переадресован на страницу создания записи о человеке, а добавление строки информации об отце в его профиле будет воспроизведено автоматически.

В случае, когда древо уже будет заполнено очень большим количеством записей, а нужно будет посмотреть запись о каком-то конкретном человеке, пользователь может воспользоваться поиском, введя в него какие-то данные, пользователь тутже получит необходимого для него человека (см. рисунок 4.1.14).

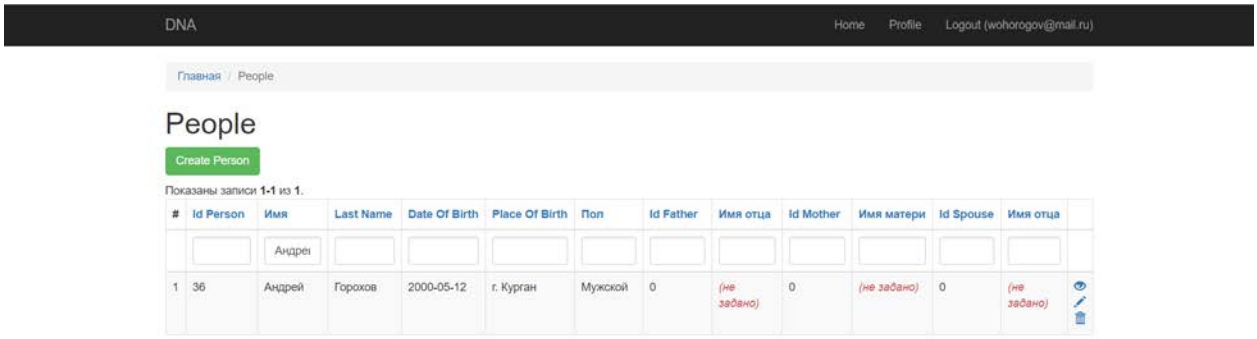


Рисунок 4.1.14 – Найденный пользователь

На рисунке 4.1.15 показана форма для изменения данных о человеке в древе.

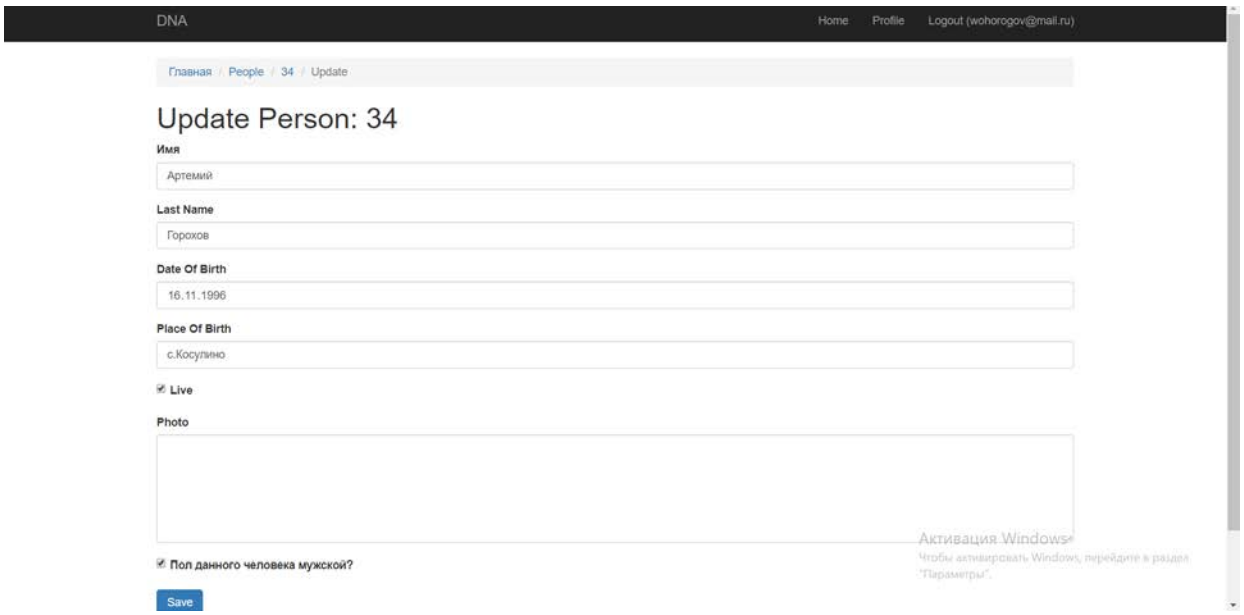


Рисунок 4.1.15 – Редактирование данных о человеке в древе

Основным преимуществом данного веб-приложения является возможность автоматического поиска в базе данных родственников или возможных родственников, с высчитыванием возможности родства в процентах. На рисунке 4.1.16 показан результат поиска родственников по системе для древа wohorogov.

Searches

Create Search

Показаны записи **1-2** из **2**.

| # | Ваше древо | Найденное древо | Возможность родства, % |
|---|---|---|---|
| 1 | wohorogov | asdsad | 99 |
| 2 | wohorogov | lollipop | 26 |

Рисунок 4.1.16 – Результат выполнения поиска родственников

По запросу пользователь может увидеть графическое представление семейного древа (см. рисунок 4.1.17).

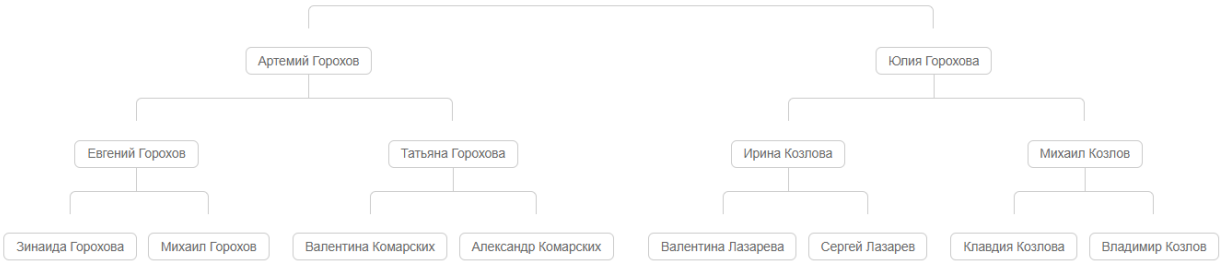


Рисунок 4.1.17 – Заполненное генеалогическое древо

Также в приложении имеется возможность загрузки GedCom файлов для автоматического заполнения генеалогических древ. На следующих рисунках продемонстрирована работа с данными файлами. В приложении С указана модель для обработки GedCom файлов.

```
Файл  Правка  Формат  Вид  Справка
0 @I2@ INDI
1 NAME Charles Phillip
1 SEX M
1 BIRT
2 DATE 10 JAN 1836
2 PLAC Cuba
1 DEAT
1 FAMS @F3@
1 FAMC @F2@
0 @I3@ INDI
1 NAME Caroline Lake
1 SEX F
1 BIRT
2 DATE 12 DEC 1839
2 PLAC Milwaukee
1 DEAT
1 FAMS @F3@
1 FAMC @F21@
0 @I1@ INDI
1 NAME Robert Ingalls
1 SEX M
1 BIRT
2 DATE 15 FEB 1850
2 PLAC Cuba
1 DEAT
1 FAMS @F5@
1 FAMC @F3@
0 @F3@ FAM
1 HUSB @I2@
1 WIFE @I3@
1 MARR
2 DATE 01 FEB 1860
2 PLAC Concord, Jefferson, WI
1 CHIL @I1@
1 CHIL @I42@
1 CHIL @I44@
1 CHIL @I45@
1 CHIL @I47@
```

Рисунок 4.1.18 – Содержание GedCom файла


Рисунок 4.1.19 – Загрузка GedCom файла

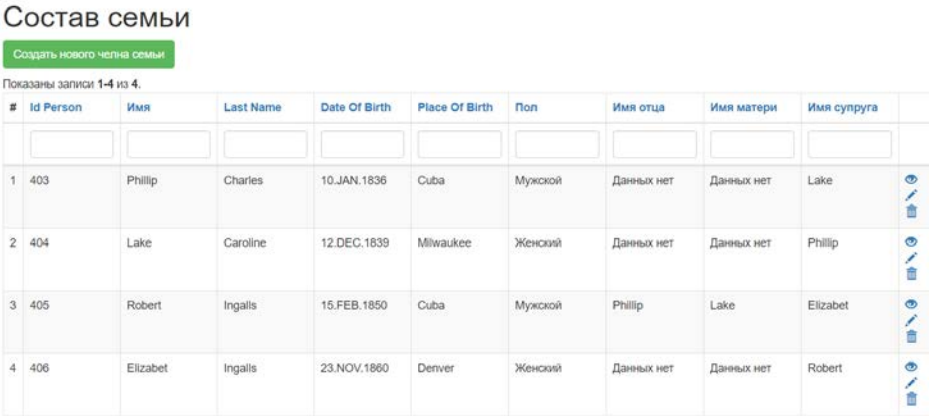


Рисунок 4.1.20 – Обработанные данные из файла GedCom после загрузки в приложение

# 5 ТЕСТИРОВАНИЕ

## 5.1 ПРОВЕДЕНИЕ ПРОЦЕДУРЫ ТЕСТИРОВАНИЯ

Так как разработка проекта велась исключительно в ходе дипломного проектирования на данный момент было проведено только альфа-тестирование. Ниже приведён перечень модульных тестов и показана работа программного обеспечения при выполнении этих тестов.

Тестирование регистрации пользователей.

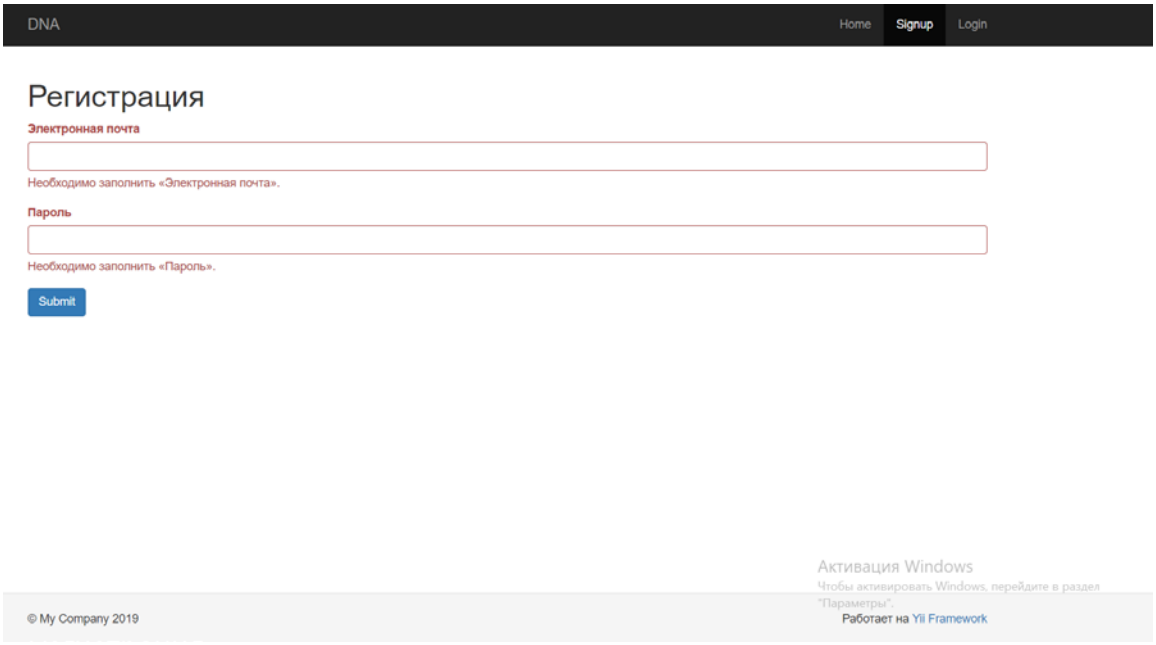


Рисунок 5.1.1 – Проверка данных, введённых пользователем при регистрации, не заполненные обязательные поля

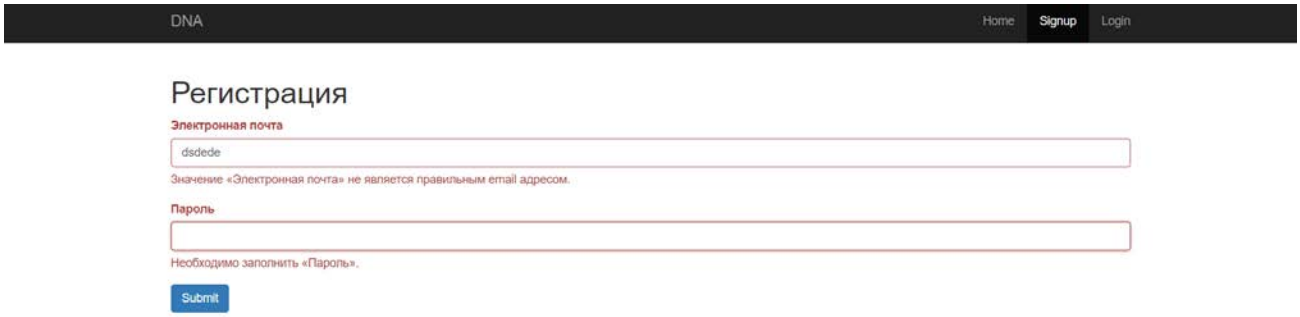


Рисунок 5.1.2 – Проверка данных, введённых пользователем, при регистрации, некорректный ввод почты

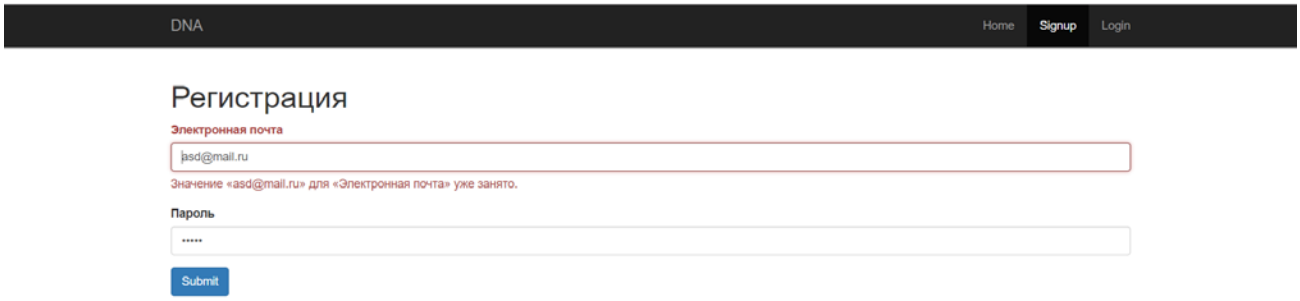


Рисунок 5.1.3 – Проверка данных, введённых пользователем при регистрации, попытка регистрации с уже существующей почтой в базе данных

Тестирование авторизации пользователя.

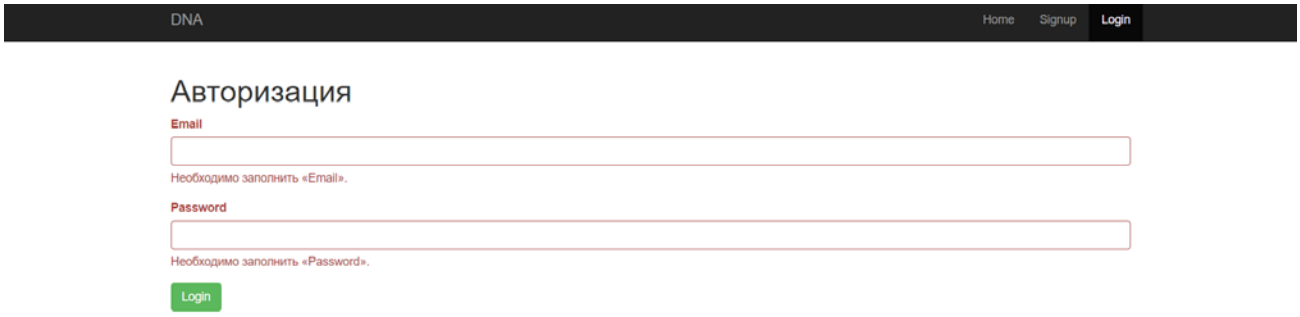


Рисунок 5.1.4 – Проверка данных, введённых пользователем при авторизации, не введены никакие данные


Рисунок 5.1.5 – Проверка данных, введённых пользователем при авторизации, введена некорректная форма для почты

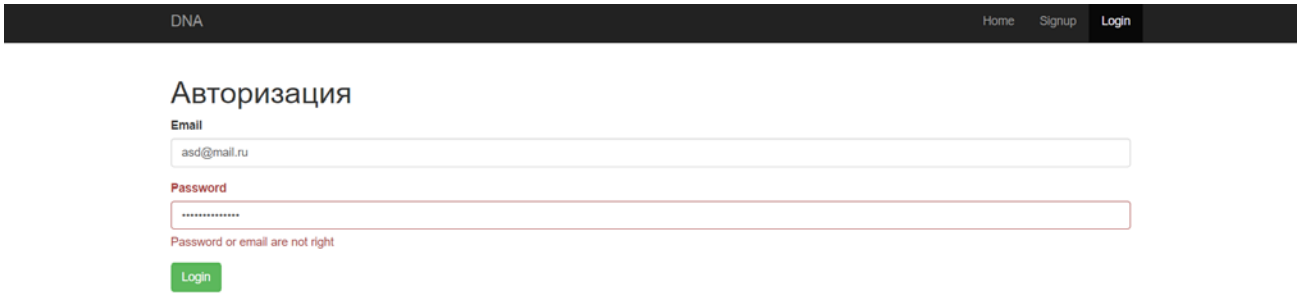


Рисунок 5.1.6 – Проверка данных, введённых пользователем при авторизации, неправильно введены почта или пароль

Тестирование поиска человека

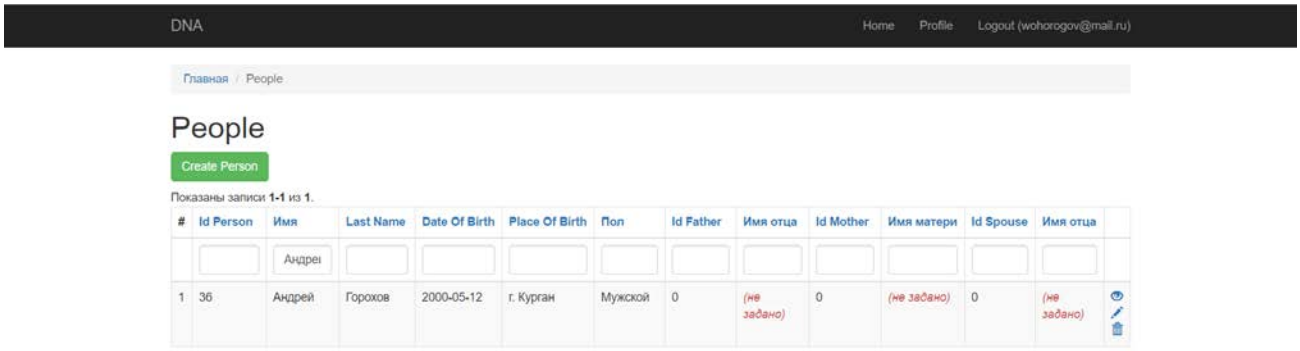


Рисунок 5.1.7 – Проверка данных, введённых пользователем при поиске родственника, удачный пример поиска


Рисунок 5.1.8 - Проверка данных, введённых пользователем при поиске родственника, неудачный пример поиска

Кроме возможности поработать со своим генеалогическим древом пользователю предоставляется возможность воспользоваться алгоритмом автоматического поиска своих возможных родственников, которые уже существуют в базе данных приложения. На рисунке 5.1.9 продемонстрирован результат тестирования автоматического поиска возможных родственников.

Показаны записи **1-2** из **2**.

| # | Ваше древо | Найденное древо | Возможность родства, % |
|---|---|---|---|
| 1 | wohorogov | asdsad | 99 |
| 2 | wohorogov | lollipop | 26 |

Рисунок 5.1.9 – Результаты автоматического поиска родственников

# 6 ЗАКЛЮЧЕНИЕ

В ходе дипломного проектирования было выполнено следующее:

- Проведён анализ предметной области;
- проведён анализ рынка родственных проектов;
- проанализированы и выбраны средства для разработки веб-приложения;
- определены функциональные и нефункциональные требования к приложению;
- спроектирована архитектура веб-приложения;
- разработана база данных;
- разработано и протестировано программное обеспечение.

В веб-приложении реализованы следующие функции:

- Создание генеалогического древа;
- работа с данными генеалогического древа;
- установление родственных связей в генеалогическом древе;
- разработан поиск людей в генеалогическом древе;
- возможен поиск всех древ и пользователей, загруженных в приложение;
- доступен автоматический поиск в системе на наличие возможных родственников;
- разработан алгоритм парсинга GedCom файлов;
- разработан алгоритм поиска возможных родственников;
- был разработан алгоритм расчёта вероятности родства между людьми в разных древах.

В дальнейшем планируется разработка системы администрирования и поиска родственников по фотографии.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК


1 Лекция 1. Введение в web-программирование. – https://studfiles.net/preview/4328850/page:2/. Дата обращения: 16.01.2019.

2 Компилируемые и некомпилируемые языки программирования. – http://itmentor.by/articles/kompiliruemye-i-interpretiruemye-yazyki-programmirovaniya. Дата обращения: 15.01.2019.

3 Что представляет из себя Yii2. – http://ksetrin.com/blog/post/futures-yii2-framework. (Дата обращения: 10.01.2019.

4 Хоккинс, С. Администрирование веб-сервера apache и руководство по электронной коммерции: пер. с англ. Н.В. Воронина, Издательский дом «Вильяме» – Москва. 2001. – 336 с.

5 Томпсон, Л. Разработка web-приложений на PHP и MySQL / Л.Томпсон, Л.Веллинг; под ред. С.Н. Козлова, науч. ред. Ю.Н. Артеменко. – 2-е изд., испр. – СПб: ООО «ДиаСофтЮП», 2003. – 672 с.

6 Yii framework – https://yiiframework.com.ua/ru. Дата обращения: 25.02.2019.

7 Бьюли, А. Изучаем SQL / А. Бьюлен; пер. с англ. Н. Шахотина – СПб: Символ-Плюс, 2007. – 312с.

8 Симдянов, И. Самоучитель PHP 7 / И. Симдянов – СПб: БХВ-Петербург, 2007. – 450с.

9 Family and Church History Department GEDCOM XML Specification Release 6.0 – The Church of Jesus Christ of Latter-day Saints, 2002, – 55с.

10 Функция создания семейных деревьев, генеалогический поиск. – https://www.myheritage.com/?lang=RU. Дата обращения: 10.01.2019.

11 Rodovid, a free multilingual family tree portal. – https://www.rodovid.org. Дата обращения: 10.01.2019.

12 Стилмэн, Э. Изучаем PHP / Э. Стилмэн; 3-е изд. – СПб.: Питер. 2014 – 816с.

13 Шилдт, Г. Java 8. Руководство для начинающих / Г. Шилдт; 6-е изд.: перевод с англ. – Москва: Вильямс. 2015 – 720 с.

14 Айвалиотис, Д. Администрирование сервера NGINX – Москва: ДМК, 2013 – 288 с.

15 Visual Studio Documentation – https://docs.microsoft.com/ru-ru/visualstudio/?view=vs-2019. Дата обращения: 12.01.2019.

16 Laravel Framework. – https://fokit.ru/frejmvork-laravel/. Дата обращения: 12.01.2019.

17 Laravel – лидер среди PHP фрейморков, одобренный разработчиками. – https://webformyself.com/laravel-lider-sredi-php-frejmvorkov-odobrennyj-razrabotchikami/. Дата обращения: 12.01.2019.

18 Windows – JetBrains PhpStorm. – https://web-zones.ru/resources/jetbrains-phpstorm.1945/. Дата обращения: 12.01.2019.

19 Eclipse – Wikipedia. – https://ru.wikipedia.org/wiki/Eclipse. Дата обращения: 10.01.2019.

20 Древо жизни – программа для построения генеалогических деревьев. – https://genery.com/ru/. Дата обращения: 10.01.2019.

21 Moederevo. – https://moederevo.livejournal.com. Дата обращения: 10.01.2019.

22 Genopro – Genealogy Software. – https://www.genopro.com. Дата обращения:10.01.2019.

23 Оценка результатов ДНК-анализа для решения вопросов идентификации личности. – https://www.forens-med.ru/book.php?id=3425. Дата обращения: 22.03.2019.

# ПРИЛОЖЕНИЕ А

Листинг А.1 – исходный код контроллера SiteController

```
<?php
namespace app\controllers;

use app\models\Profile;
use Yii;
use yii\base\Model;
use yii\web\Controller;
use app\models\Signup;
use app\models\Login;

class SiteController extends BehaviorsController
{
    public function actionIndex()
    {
        return $this->render('index');
    }
    public function actionLogout()
    {
        if (!Yii::$app->user->isGuest)
        {
            Yii::$app->user->logout();
            return $this->redirect(['login']);
        }
    }
    public function actionSignup()
    {
        $model = new Signup();
        if(Yii::$app->request->post())
        {
            $model->attributes=Yii::$app->request->post('Signup');
            if($model->validate() && $model->signup())
            {
                return $this->redirect('login');//, ['model'=> new $model]);
            }
        }
        return $this->render('signup',['model'=> $model]);
    }
    public function actionLogin()
    {
        if (!Yii::$app->user->isGuest) {
            return $this->goHome();
        }
        $login_model = new Login();
        if (Yii::$app->request->post('Login'))
        {
            $login_model->attributes = Yii::$app->request->post('Login');
            if ($login_model->validate())
            {
                Yii::$app->user->login($login_model->getUser());
                return $this->redirect('..\family\index');
            }
        }
        return $this->render('login', ['login_model' => $login_model]);
    }
}
```

# ПРИЛОЖЕНИЕ Б

Листинг Б.2 – Исходный код модели User

```
<?php

namespace app\models;
use yii\db\ActiveRecord;
use yii\web\IdentityInterface;
class User extends ActiveRecord implements IdentityInterface
{

    public function setPassword($password)
    {
        $this->password = sha1($password);
    }

    public function validatePassword($password)
    {
        return $this->password === sha1($password);
    }

    public static function findIdentity($id)
    {
        return self::findOne($id);
    }

    public function getId()
    {
        return $this->id;
    }

    public static function findIdentityByAccessToken($token, $type = null)
    {

    }

    public function getAuthKey()
    {

    }

    public function validateAuthKey($authKey)
    {

    }
}
```

# ПРИЛОЖЕНИЕ В

Листинг В.1 – Исходный код модели Login

```
<?php

namespace app\models;

use yii\base\Model;

class Login extends Model
{
    public $email;
    public $password;

    public function rules()
    {
        return [
            [['email','password'],'required'],
            ['email','email'],
            ['password','validatePassword']
        ];
    }

    public function validatePassword($attribute, $params)
    {
        if (!$this->hasErrors())
        {
            $user = $this->getUser();

            if (!$user || !$user->validatePassword($this->password))
            {
                $this->addError($attribute,'Password or email are not right');
            }
        }
    }

    public function getUser()
    {
        return User::findOne(['email'=>$this->email]);
    }
}
```

# ПРИЛОЖЕНИЕ Г

Листинг Г.1 – исходный код модели Signup

```
<?php

namespace app\models;

use yii\base\Model;

class Signup extends Model
{
    public $email;
    public $password;

    public function rules()
    {

        return [
            [['email', 'password'], 'required'],
            ['email', 'email'],
            ['email', 'unique', 'targetClass' => 'app\models\User'],
            ['password', 'string', 'min' => 2, 'max' => 10]
        ];
    }

    public function attributeLabels()
    {
        return [
            'email' => 'Электронная почта',
            'password' => 'Пароль'
        ]; // TODO: Change the autogenerated stub
    }

    public function signup()
    {
        $user = new User();
        $user->email = $this->email;
        $user->setPassword($this->password);
        return $user->save();
    }
}
```

# ПРИЛОЖЕНИЕ Д

Листинг Д.1 – Листинг контролллера BehaviorsController

```
<?php

namespace app\controllers;

use app\models\Family;
use Yii;
use yii\db\Exception;
use yii\web\Controller;
use yii\filters\AccessControl;

class BehaviorsController extends Controller
{
    public function behaviors()
    {
        return [
            'access' => [
                'class' => AccessControl::className(),
                'denyCallback' => function($rule, $action){
                    throw new \Exception('Нет доступа.');
                },
                'rules' => [
                    [
                        'allow' => true,
                        'controllers' => ['site'],
                        'actions' => ['signup', 'login'],
                        'verbs' => ['GET', 'POST'],
                        'roles' => ['?']
                    ],
                    [
                        'allow' => true,
                        'actions' => ['index']
                    ],
                    [
                        'allow' => true,
                        'controllers' => ['site'],
                        'actions' => ['logout'],
                        'verbs' => ['POST', 'GET'],
                        'roles' => ['@']
                    ],
                    [
                        'allow' => true,
                        'controllers' => ['family', 'person', 'relation'],
                        'actions' => [
                            'index',
                            'view',
                            'create',
                            'update',
                            'delete',
                            'toPerson',
                            'list',
                            'createdad',
                            'createmom',
                            'createspouse',
                            'allperson',
                            'zadr'
```

Окончание приложения Д

```
                        ],
                        'verbs' => ['POST', 'GET'],
                        'roles' => ['@']
                    ]
                ]
            ]
        ];
    }
}
```

# ПРИЛОЖЕНИЕ Е

Листинг Е.1 – Исходный код контроллера PersonController

```
<?php

namespace app\controllers;

use app\models\PersonSearch;
use app\models\Search;
use Yii;
use app\models\Person;
use app\models\Family;
use yii\base\Model;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
use app\models\ImageUpLoad;
use yii\web\UploadedFile;


class PersonController extends BehaviorsController
{
    /**
     * {@inheritdoc}
     */

    /**
     * Lists all Person models.
     * @return mixed
     */


    public function actionIndex()
    {
        $searchModel = new PersonSearch();
        $dataProvider = $searchModel->search(Yii::$app->request->queryParams);
//        $searchModel->searchPossibleRelatives();

        return $this->render('index', [
            'searchModel' => $searchModel,
            'dataProvider' => $dataProvider,
        ]);
    }

    public function actionAllperson()
    {
        $searchModel = new PersonSearch();
        $dataProvider       =       $searchModel->searchAll(Yii::$app->request-
>queryParams);

        return $this->render('index', [
            'searchModel' => $searchModel,
            'dataProvider' => $dataProvider,
        ]);
    }

    /**
     * Displays a single Person model.
```

Продолжение приложения Е

```
     * @param integer $id
     * @return mixed
     * @throws NotFoundHttpException if the model cannot be found
     */

    public function actionView($id)
    {
        return $this->render('view', [
            'model' => $this->findModel($id),
        ]);
    }

    /**
     * Creates a new Person model.
     * If creation is successful, the browser will be redirected to the 'view'
page.
     * @return mixed
     */

    public function actionCreate()
    {
        $image = new ImageUpLoad();
        $model = new Person();
        $file = UploadedFile::getInstance($image, 'image'); //


        if ($model->load(Yii::$app->request->post()))
        {
            if ($model->saveImage($image->uploadFile($file, $model->photo)))
                if ($model->createPerson())
                {
                    return $this->redirect(['person/index']);
                }
//              var_dump('hello');
//            var_dump($_POST['$model']); die();
            return $this->render('view', ['model' => $model]);
        }

        return $this->render('create', [
            'model' => $model,
            'image' => $image,
        ]);
    }

    public function actionCreatedad($id)
    {
        $model = new Person();
        $person = Person::findOne($id);

        if ($model->load(Yii::$app->request->post()))
        {
            $person->ID_father = $model->createPersonRel();
            $person->save();
            return $this->render('view', ['model' => $model]);
        }

        return $this->render('createdad', [
            'model' => $model,
            'id' => $id
```

Продолжение приложения Е

```
        ]);
    }

    public function actionCreatemom($id)
    {
        $model = new Person();
        $person = Person::findOne($id);

        if ($model->load(Yii::$app->request->post()))
        {
            $person->ID_mother = $model->createPersonRel();
            $person->save();
            return $this->render('view', ['model' => $model]);
        }

        return $this->render('createmom', [
            'model' => $model,
            'id' => $id
        ]);
    }


    public function actionCreatespouse($id)
    {
        $model = new Person();
        $person = Person::findOne($id);

        if ($model->load(Yii::$app->request->post()))
        {
            $person->ID_spouse = $model->createPersonRel();
            $person->save();
            return $this->render('view', ['model' => $model]);
        }

        return $this->render('createspouse', [
            'model' => $model,
            'id' => $id
        ]);
    }


    /**
     * Updates an existing Person model.
     * If update is successful, the browser will be redirected to the 'view'
page.
     * @param integer $id
     * @return mixed
     * @throws NotFoundHttpException if the model cannot be found
     */

    public function actionUpdate($id)
    {
        $model = $this->findModel($id);
        $image = new UploadedFile();

        if ($model->load(Yii::$app->request->post()) && $model->save()) {
            return $this->redirect(['view', 'id' => $model->ID_person]);
        }
```

Окончание приложения Е

```
            return $this->render('update', [
                'model' => $model,
                'image' => $image,
            ]);
        }
        /**
         * Deletes an existing Person model.
         * If deletion is successful, the browser will be redirected to the
'index' page.
         * @param integer $id
         * @return mixed
         * @throws NotFoundHttpException if the model cannot be found
         */
        public function actionDelete($id)
        {
            $this->findModel($id)->delete();

            return $this->redirect(['index']);
        }
        /**
         * Finds the Person model based on its primary key value.
         * If the model is not found, a 404 HTTP exception will be thrown.
         * @param integer $id
         * @return Person the loaded model
         * @throws NotFoundHttpException if the model cannot be found
         */

        protected function findModel($id)
        {
            if (($model = Person::findOne($id)) !== null) {
                return $model;
            }

            throw new NotFoundHttpException('The requested page does not exist.');
        }
    }
```

# ПРИЛОЖЕНИЕ Ж

Листинг Ж.1 – Исходный код модели Person

```
<?php
/**
 * Created by PhpStorm.
 * User: artem
 * Date: 15.04.2019
 * Time: 23:12
 */

namespace app\models;

use yii\base\Model;
use yii\db\ActiveRecord;
use Yii;

/**
 * This is the model class for table "person".
 *
 * @property int $ID_person
 * @property string $first_name
 * @property string $last_name
 * @property string $date_of_birth
 * @property string $place_of_birth
 * @property int $live
 * @property string $photo
 * @property int $gender
 * @property int $ID_family
 * @property int $ID_mother
 * @property int ID_spouse
 * @property int ID_father
 *
 *
 * @property Family $family
 * @property Relation[] $relations
 * @property Relation[] $relations0
 */

class Person extends ActiveRecord
{
    /**
     * {@inheritdoc}
     */
    public static function tableName()
    {
        return 'Person';
    }

    /**
     * {@inheritdoc}
     */

    public function rules()
    {
        return [
            [['date_of_birth'], 'safe'],
            [['live', 'gender', 'ID_family'], 'integer'],
```

Продолжение приложения Ж

```
                [['photo'], 'string'],
                [['first_name', 'last_name', 'place_of_birth'], 'string', 'max' =>
255],
                [['ID_family'], 'exist', 'skipOnError' => true, 'targetClass' =>
Family::className(), 'targetAttribute' => ['ID_family' => 'ID_family']],
            ];
        }

        /**
         * {@inheritdoc}
         */

        public function attributeLabels()
        {
            return [
                'ID_person' => 'Id Person',
                'first_name' => 'Имя',
                'last_name' => 'Last Name',
                'date_of_birth' => 'Date Of Birth',
                'place_of_birth' => 'Place Of Birth',
                'live' => 'Live',
                'photo' => 'Photo',
                'gender' => 'Gender',
                'ID_family' => 'Id Family',
            ];
        }


        /**
         * @return \yii\db\ActiveQuery
         */

        public function createPerson()
        {
            $user = User::findOne(\Yii::$app->user->id);
            $model = new Person();
            $model->first_name = $this->first_name;
            $model->last_name = $this->last_name;
            $model->place_of_birth = $this->place_of_birth;
            $model->date_of_birth = $this->date_of_birth;
            $model->live = $this->live;
            $model->photo = $this->photo;
            $model->gender = $this->gender;
            $family = \app\models\Family::findOne(['ID_user' => \Yii::$app->user-
>id]);

            $model->ID_family = $family->ID_family;
            return ($model->save()) ? true : false;
        }

        public function createPersonRel()
        {
            $model = new Person();
            $model->first_name = $this->first_name;
            $model->last_name = $this->last_name;
            $model->place_of_birth = $this->place_of_birth;
            $model->date_of_birth = $this->date_of_birth;
            $model->live = $this->live;
            $model->photo = $this->photo;
```

Окончание приложения Ж

```
            $model->gender = $this->gender;
            $family = \app\models\Family::findOne(['ID_user' => \Yii::$app->user->id]);

            $model->ID_family = $family->ID_family;
            if ($model->save())
            {
                return $model->ID_person;
            }

            return null;
        }

        public function saveImage($filename)
        {
         //   $user = User::findOne(Yii::$app->user->id);
            $family = Family::findOne(['ID_user' => Yii::$app->user->id]);
            $person = ($person = Person::findOne(['ID_family' => $family->ID_family])) ? $person : new Person();
            if ($filename != null)
                $person->photo = $filename;
            return $person->save() ? true : false;
        }

        public function getImage()
        {
            $family = Family::findOne(['ID_user' => Yii::$app->user->id]);
            $person = ($person = Person::findOne(['ID_family' => $family->ID_family])) ? $person : new Person();
            if ($person->photo && file_exists('uploads/' . $person->photo)) {
                return '/web/uploads/' . $person->photo;
            } else {
                $this->saveImage($this->photo = null);
            }
        }

        public function deleteImage()
        {
            $imageUpLoadModel = new ImageUpLoad();
            $imageUpLoadModel->deleteCurrentImage($this->avatar);
        }


        public function getFamily()
        {
            return $this->hasOne(Family::className(), ['ID_family' => 'ID_family']);
        }
    }
```

# ПРИЛОЖЕНИЕ 3

Листинг 3.1 – Исходный код модели PersonSearch

```
<?php

namespace app\models;

use yii\base\Model;
use yii\data\ActiveDataProvider;
use app\models\Person;
use app\models\Family;
use app\models\Search;

/**
 *   PersonSearch   represents   the   model   behind   the   search   form   of
`app\models\Person`.
 */
class PersonSearch extends Person
{
    /**
     * {@inheritdoc}
     */
    public function rules()
    {
        return [
            [['ID_person',   'live',   'gender',   'ID_family',   'ID_father',
'ID_mother', 'ID_spouse'], 'integer'],
            [['first_name',  'last_name',  'date_of_birth',  'place_of_birth',
'photo'], 'safe'],
        ];
    }

    /**
     * {@inheritdoc}
     */
    public function scenarios()
    {
        // bypass scenarios() implementation in the parent class
        return Model::scenarios();
    }

    /**
     * Creates data provider instance with search query applied
     *
     * @param array $params
     *
     * @return ActiveDataProvider
     */
    public function search($params)
    {
//        $family = new \app\models\Family();
        $family  =  \app\models\Family::findOne(['ID_user' => \Yii::$app->user-
>id]);
        $query   =   Person::find()->with(['family'])->where(['ID_family'   =>
$family->ID_family]);
        // add conditions that should always apply here
        $dataProvider = new ActiveDataProvider([
            'query' => $query,
```

```
            ]);

            $this->load($params);

            if (!$this->validate()) {
                // uncomment the following line if you do not want to return any
records when validation fails
                // $query->where('0=1');
                return $dataProvider;
            }

            $family = new Family();

            // grid filtering conditions
            $query->andFilterWhere([
                'ID_person' => $this->ID_person,
                'date_of_birth' => $this->date_of_birth,
                'live' => $this->live,
                'gender' => $this->gender,
                'ID_family' => $this->ID_family,
                'ID_father' => $this->ID_father,
                'ID_mother' => $this->ID_mother,
                'ID_spouse' => $this->ID_spouse,
            ]);

            $query->andFilterWhere(['like', 'first_name', $this->first_name])
                ->andFilterWhere(['like', 'last_name', $this->last_name])
                ->andFilterWhere(['like',          'place_of_birth',          $this-
>place_of_birth])
                ->andFilterWhere(['like', 'photo', $this->photo]);

            $query->andFilterWhere([

      //            $family->ID_user == \Yii::$app->user->id,
      //            'ID_family' => \Yii::$app->user->id,
            ]);

            return $dataProvider;
        }

        public function percentDistribution($sum, $num, $otherFamily, $family)
        {
            Search::deleteAll('ID_family_searching  =  :id  &&  ID_family_found  =
:id1',
                [':id' => $family->ID_family, ':id1' => $otherFamily->ID_family]);
            $search = new Search();
            $search->ID_family_searching = $family->ID_family;
            $search->ID_family_found = $otherFamily->ID_family;
            $value = $sum / $num;
            $search->value = $value;
            return ($search->save()) ? true : false;
        }

        public function samePerson($otherFamily, $family)
        {
            Search::deleteAll('ID_family_searching  =  :id  &&  ID_family_found  =
:id1',
                [':id' => $family->ID_family, ':id1' => $otherFamily->ID_family]);
```

```
            $search = new Search();
            $search->ID_family_searching = $family->ID_family;
            $search->ID_family_found = $otherFamily->ID_family;
```

Продолжение приложения 3

```
            $search->value = 99;
            return ($search->save()) ? true : false;
        }

        public function searchPossibleRelatives()
        {
            $model = new PersonSearch();
            $sum = 0;
            $num = 0;
            $sameCity = 0;
            $family = \app\models\Family::findOne(['ID_user' => \Yii::$app->user-
>id]);
            $persons = Person::find()->with('family')->where(['ID_family' =>
$family->ID_family])->all();
            $otherFamilies = Family::find()->where('ID_family != :id', [':id'
=>$family->ID_family])->all();

            foreach ($otherFamilies as $otherFamily)
            {
                $sum = 0;
                $num = 0;
                $sameperson = 0;

                foreach ($persons as $person)
                {
                    $otherPersons = Person::find()->where([ 'ID_family' =>
$otherFamily->ID_family])->all();
                    foreach ($otherPersons as $otherPerson)
                    {
                        if ($person->last_name == $otherPerson->last_name ||
$person->last_name == $otherPerson->last_name.'a' ||
                            $person->last_name.'a' == $otherPerson->last_name)
                        {
                            if ($person->place_of_birth == $otherPerson-
>place_of_birth)
                            {
                                if($person->first_name == $otherPerson->first_name
&&
                                $person->date_of_birth == $otherPerson-
>date_of_birth)
                                {
                                    $sameperson = 1;
                                }
                                else
                                {
                                    $sum = $sum + 70;
                                    $num++;
                                }
                            }
                            else
                            {
                                    $sum = $sum + 15;
                                    $num++;
                            }
                        }
                        if ($sameperson)
```

```
                        {
                            $num = 0;
                            $sum = 0;
```

Окончание приложения 3

```
                            if ($model->samePerson($otherFamily, $family))
                                break 2;
                        }
                    }
                }
                if ($num != 0) {
                    if   ($model->percentDistribution($sum,   $num,   $otherFamily,
$family)) {
                        $sum = 0;
                        $num = 0;
                    }
                }
            }

            $results = Search::find()->all();
            return $results;
        }

        public function searchAll($params)
        {
            $query = Person::find();


            // add conditions that should always apply here

            $dataProvider = new ActiveDataProvider([
                'query' => $query,
            ]);

            $this->load($params);

            if (!$this->validate()) {
                // uncomment the following line if you do not want to return any
records when validation fails
                // $query->where('0=1');
                return $dataProvider;
            }

            // grid filtering conditions
            $query->andFilterWhere([
                'ID_person' => $this->ID_person,
                'date_of_birth' => $this->date_of_birth,
                'live' => $this->live,
                'gender' => $this->gender,
                'ID_family' => $this->ID_family,
                'ID_father' => $this->ID_father,
                'ID_mother' => $this->ID_mother,
                'ID_spouse' => $this->ID_spouse,
            ]);
            $query->andFilterWhere(['like', 'first_name', $this->first_name])
                ->andFilterWhere(['like', 'last_name', $this->last_name])
                ->andFilterWhere(['like',         'place_of_birth',         $this-
>place_of_birth])
                ->andFilterWhere(['like', 'photo', $this->photo]);
            return $dataProvider;
```

```
    }
}
```

# ПРИЛОЖЕНИЕ И

Листинг И.1 – Исходный код контроллера FamilyController

```
<?php

namespace app\controllers;

use app\models\Person;
use Yii;
use app\models\Family;
use app\models\FamilySearch;
use app\models\NewFamilyForm;
use app\models\User;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;

/**
 * FamilyController implements the CRUD actions for Family model.
 */
class FamilyController extends BehaviorsController
{
    /**
     * {@inheritdoc}
     */

    /**
     * Lists all Family models.
     * @return mixed
     */
    public function actionIndex()
    {
        $searchModel = new FamilySearch();
        $dataProvider = $searchModel->search(Yii::$app->request->queryParams);

        return $this->render('index', [
            'searchModel' => $searchModel,
            'dataProvider' => $dataProvider,
        ]);
    }

    /**
     * Displays a single Family model.
     * @param integer $id
     * @return mixed
     * @throws NotFoundHttpException if the model cannot be found
     */
    public function actionView($id)
    {
        return $this->render('view', [
            'model' => $this->findModel($id),
        ]);
    }

    /**
     * Creates a new Family model.
     * If creation is successful, the browser will be redirected to the 'view'
page.
```

Продолжение приложения И

```
     * @return mixed
     */
    public function actionCreate()
    {
        $family = new Family();
        $model = new Family();

        if     ($family->load(Yii::$app->request->post())     &&     ($family-
>CreateFamily())) {
             return   $this->redirect(['index']);   //,   'id'   =>   $model-
>ID_family]);
        }

        return $this->render('create', [
            'model' => $family,
        ]);
    }

    public function actionToPerson()
    {
        return $this->render('toPerson');
    }

    /**
     * Updates an existing Family model.
     * If update is successful, the browser will be redirected to the 'view'
page.
     * @param integer $id
     * @return mixed
     * @throws NotFoundHttpException if the model cannot be found
     */
    public function actionUpdate($id)
    {
        $model = $this->findModel($id);

        if ($model->load(Yii::$app->request->post()) && $model->save()) {
            return $this->redirect(['view', 'id' => $model->ID_family]);
        }

        return $this->render('update', [
            'model' => $model,
        ]);
    }

    /**
     * Deletes an existing Family model.
     *  If  deletion  is  successful,  the  browser  will  be  redirected  to  the
'index' page.
     * @param integer $id
     * @return mixed
     * @throws NotFoundHttpException if the model cannot be found
     */
    public function actionDelete($id)
    {
        $this->findModel($id)->delete();
        return $this->redirect(['index']);
    }
```

Окончание приложения И

```
    /**
     * Finds the Family model based on its primary key value.
     * If the model is not found, a 404 HTTP exception will be thrown.
     * @param integer $id
     * @return Family the loaded model
     * @throws NotFoundHttpException if the model cannot be found
     */
    protected function findModel($id)
    {
        if (($model = Family::findOne($id)) !== null) {
            return $model;
        }

        throw new NotFoundHttpException('The requested page does not exist.');
    }
}
```

# ПРИЛОЖЕНИЕ К

Листинг К.1 – Исходный код модели Family

```
<?php

namespace app\models;

use Yii;

/**
 * This is the model class for table "family".
 *
 * @property int $ID_family
 * @property string $name
 * @property int $ID_user
 * @property int $access
 *
 * @property User $user
 * @property Person[] $people
 * @property Search[] $searchesSearch
 * @property Search[] $searchesFound
 */
class Family extends \yii\db\ActiveRecord
{
    /**
     * {@inheritdoc}
     */
    public static function tableName()
    {
        return 'family';
    }

    /**
     * {@inheritdoc}
     */
    public function rules()
    {
        return [
            [['ID_user', 'access'], 'integer'],
            [['access'], 'required'],
            [['name'], 'string', 'max' => 255],
            [['ID_user'], 'exist', 'skipOnError' => true, 'targetClass' =>
User::className(), 'targetAttribute' => ['ID_user' => 'id']],
        ];
    }

    /**
     * {@inheritdoc}
     */
    public function attributeLabels()
    {
        return [
            'ID_family' => 'Id Family',
            'name' => 'Название',
            'ID_user' => 'Id User',
            'access' => 'Access',
        ];
    }
```

Окончание приложения К

```
            public function createFamily()
            {
                $user = user::findOne(Yii::$app->user->id);
                $family = new Family();
                $family->name = $this->name;
                $family->link('user', $user);
                return $family->save() ? true : false;
            }

            public function isCreator()
            {
                $family = Family::findOne(['ID_family' => $this->ID_family, 'ID_user'
=> Yii::$app->user->id]);
                if (empty($family))
                {
                    $family = Family::findOne(['ID_user' => Yii::$app->user->id]);
                }

                return (!empty($family)) ? true : false;
            }

            /**
             * @return \yii\db\ActiveQuery
             */
            public function getUser()
            {
                return $this->hasOne(User::className(), ['id' => 'ID_user']);
            }

            /**
             * @return \yii\db\ActiveQuery
             */
            public function getPersons()
            {
                return     $this->hasMany(Person::className(),     ['ID_family'     =>
'ID_family']);
            }

            public function getSearchesSearch()
            {
                return  $this->hasMany(Search::className(),  ['ID_family_searching'  =>
'ID_family']);
            }

            public function getSearchesFound()
            {
                return   $this->hasMany(Search::className(),   ['ID_family_found'   =>
'ID_family']);
            }
        }
```

# ПРИЛОЖЕНИЕ Л

Листинг Л.1 – Исходный код модели FamilySearch

```
<?php

namespace app\models;

use yii\base\Model;
use yii\data\ActiveDataProvider;
use app\models\Family;

/**
 *  FamilySearch  represents  the  model  behind  the  search  form  of
`app\models\Family`.
 */
class FamilySearch extends Family
{
    /**
     * {@inheritdoc}
     */
    public function rules()
    {
        return [
            [['ID_family', 'ID_user'], 'integer'],
            [['name'], 'safe'],
        ];
    }

    /**
     * {@inheritdoc}
     */
    public function scenarios()
    {
        // bypass scenarios() implementation in the parent class
        return Model::scenarios();
    }

    /**
     * Creates data provider instance with search query applied
     *
     * @param array $params
     *
     * @return ActiveDataProvider
     */
    public function search($params)
    {
        $query = Family::find();

        // add conditions that should always apply here

        $dataProvider = new ActiveDataProvider([
            'query' => $query,
        ]);

        $this->load($params);
        if (!$this->validate()) {
            // uncomment the following line if you do not want to return any
records when validation fails
```

Окончание приложения К

```
            // $query->where('0=1');
            return $dataProvider;
        }

        // grid filtering conditions
        $query->andFilterWhere([
            'ID_family' => $this->ID_family,
            'ID_user' => \Yii::$app->user->id,
        ]);

        $query->andFilterWhere(['like', 'name', $this->name]);

        return $dataProvider;
    }
}
```

# ПРИЛОЖЕНИЕ М

Листинг М.1 – Исходный код контроллера SearchController

```
<?php

namespace app\controllers;

use Yii;
use app\models\Search;
use app\models\SearchSearch;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;

/**
 * SearchController implements the CRUD actions for Search model.
 */
class SearchController extends Controller
{
    /**
     * {@inheritdoc}
     */
    public function behaviors()
    {
        return [
            'verbs' => [
                'class' => VerbFilter::className(),
                'actions' => [
                    'delete' => ['POST'],
                ],
            ],
        ];
    }

    /**
     * Lists all Search models.
     * @return mixed
     */
    public function actionIndex()
    {
        $searchModel = new SearchSearch();
        $dataProvider = $searchModel->search(Yii::$app->request->queryParams);

        return $this->render('index', [
            'searchModel' => $searchModel,
            'dataProvider' => $dataProvider,
        ]);
    }

    /**
     * Displays a single Search model.
     * @param integer $id
     * @return mixed
     * @throws NotFoundHttpException if the model cannot be found
     */
    public function actionView($id)
    {
        return $this->render('view', [
```

```
            'model' => $this->findModel($id),
        ]);
    }

    /**
     * Creates a new Search model.
     * If creation is successful, the browser will be redirected to the 'view'
page.
     * @return mixed
     */
    public function actionCreate()
    {
        $model = new Search();

        if ($model->load(Yii::$app->request->post()) && $model->save()) {
            return $this->redirect(['view', 'id' => $model->ID_search]);
        }

        return $this->render('create', [
            'model' => $model,
        ]);
    }

    /**
     * Updates an existing Search model.
     * If update is successful, the browser will be redirected to the 'view'
page.
     * @param integer $id
     * @return mixed
     * @throws NotFoundHttpException if the model cannot be found
     */
    public function actionUpdate($id)
    {
        $model = $this->findModel($id);

        if ($model->load(Yii::$app->request->post()) && $model->save()) {
            return $this->redirect(['view', 'id' => $model->ID_search]);
        }

        return $this->render('update', [
            'model' => $model,
        ]);
    }

    /**
     * Deletes an existing Search model.
     *  If  deletion  is  successful,  the  browser  will  be  redirected  to  the
'index' page.
     * @param integer $id
     * @return mixed
     * @throws NotFoundHttpException if the model cannot be found
     */
    public function actionDelete($id)
    {
        $this->findModel($id)->delete();
        return $this->redirect(['index']);
    }

    /**
```

Окончание приложения М

```
     * Finds the Search model based on its primary key value.
     * If the model is not found, a 404 HTTP exception will be thrown.
     * @param integer $id
     * @return Search the loaded model
     * @throws NotFoundHttpException if the model cannot be found
     */
    protected function findModel($id)
    {
        if (($model = Search::findOne($id)) !== null) {
            return $model;
        }

        throw new NotFoundHttpException('The requested page does not exist.');
    }
}
```

# ПРИЛОЖЕНИЕ Н

Листинг Н.1 – Исходный код модели Search

```
<?php

namespace app\models;

use Yii;

/**
 * This is the model class for table "search".
 *
 * @property int $ID_search
 * @property int $ID_family_searching
 * @property int $ID_family_found
 * @property int $value
 *
 * @property Family $familySearching
 * @property Family $familyFound
 */
class Search extends \yii\db\ActiveRecord
{
    /**
     * {@inheritdoc}
     */
    public static function tableName()
    {
        return 'search';
    }

    /**
     * {@inheritdoc}
     */
    public function rules()
    {
        return [
            [['ID_family_searching', 'ID_family_found', 'value'], 'required'],
            [['ID_family_searching', 'ID_family_found', 'value'], 'integer'],
            [['ID_family_searching'], 'exist', 'skipOnError' => true,
'targetClass' => Family::className(), 'targetAttribute' => ['ID_family_searching' =>
'ID_family']],
            [['ID_family_found'], 'exist', 'skipOnError' => true,
'targetClass' => Family::className(), 'targetAttribute' => ['ID_family_found' =>
'ID_family']],
        ];
    }

    /**
     * {@inheritdoc}
     */
    public function attributeLabels()
    {
        return [
            'ID_search' => 'Id Search',
            'ID_family_searching' => 'Id Family Searching',
            'ID_family_found' => 'Id Family Found',
            'value' => 'Value',
        ];
```

```
        }
```

Окончание приложения Н

```
        /**
         * @return \yii\db\ActiveQuery
         */
        public function getFamilySearching()
        {
            return     $this->hasOne(Family::className(),     ['ID_family'     =>
'ID_family_searching']);
        }

        /**
         * @return \yii\db\ActiveQuery
         */
        public function getFamilyFound()
        {
            return     $this->hasOne(Family::className(),     ['ID_family'     =>
'ID_family_found']);
        }
    }
```

# ПРИЛОЖЕНИЕ О

Листинг О.1 – Исходный код модели SearchSearch

```
<?php

namespace app\models;

use yii\base\Model;
use yii\data\ActiveDataProvider;
use app\models\Search;

/**
 *  SearchSearch  represents  the  model  behind  the  search  form  of
`app\models\Search`.
 */
class SearchSearch extends Search
{
    /**
     * {@inheritdoc}
     */
    public function rules()
    {
        return [
            [['ID_search', 'ID_family_searching', 'ID_family_found', 'value'],
'integer'],
        ];
    }

    /**
     * {@inheritdoc}
     */
    public function scenarios()
    {
        // bypass scenarios() implementation in the parent class
        return Model::scenarios();
    }

    /**
     * Creates data provider instance with search query applied
     *
     * @param array $params
     *
     * @return ActiveDataProvider
     */
    public function search($params)
    {
        $family  =  Family::find()->where(['ID_user'  =>  \Yii::$app->user->id])-
>one();
        $query  =  Search::find()->where(['ID_family_searching'  =>  $family-
>ID_family]);

        // add conditions that should always apply here

        $dataProvider = new ActiveDataProvider([
            'query' => $query,
        ]);
        $this->load($params);
```

Окончание приложения О

```
            if (!$this->validate()) {
                // uncomment the following line if you do not want to return any
records when validation fails
                // $query->where('0=1');
                return $dataProvider;
            }

            // grid filtering conditions
            $query->andFilterWhere([
                'ID_search' => $this->ID_search,
                'ID_family_searching' => $this->ID_family_searching,
                'ID_family_found' => $this->ID_family_found,
                'value' => $this->value,
            ]);

            return $dataProvider;
        }
    }
```

# ПРИЛОЖЕНИЕ П

Листинг П.1 – Исходный код модели ImageUpLoad

```
<?php
/**
 * Created by PhpStorm.
 * User: artem
 * Date: 27.05.2019
 * Time: 22:53
 */

namespace app\models;


use yii\base\Model;
use Yii;
use yii\web\UploadedFile;

class ImageUpLoad extends Model
{
    /**
     * @var UploadedFile
     */
    public $image;

    public function rules()
    {
        return [
//            [['image'], 'required'],
            [['image'], 'file', 'extensions' => 'jpg,png']
        ];
    }

    public function attributeLabels()
    {
        return [
            'image' => 'Фото',
        ];
    }

    public function uploadFile($file, $currentImage)
    {
        if (($file != null) && ($this->validate())) {
            $this->image = $file;
            $this->deleteCurrentImage($currentImage);
            return $this->saveImage();
        }
    }

    private function getFolder()
    {
        return Yii::getAlias('@web') . 'uploads/';
    }

    private function generateFileName()
    {
        return strtolower(md5(uniqid($this->image->baseName)) . '.' . $this-
>image->extension);
```

Окончание приложения П

```
        }

        public function deleteCurrentImage($currentImage)
        {
            if ($this->fileExists($currentImage)) {
                unlink($this->getFolder() . $currentImage);
            }
        }

        private function fileExists($currentImage)
        {
            if (!empty($currentImage) && $currentImage != null)
                return file_exists($this->getFolder() . $currentImage);
        }

        private function saveImage()
        {
            $filename = $this->generateFileName();
            if ($this->image != null) {
      //            var_dump($this->getFolder()); var_dump($filename);  die();
                $this->image->saveAs('@web/uploads/trtr.jpg');//$this-
>getFolder().$filename);
                return $filename;
            }
        }
    }
```

# ПРИЛОЖЕНИЕ Р

Листинг Р.1 – Исходный код контроллера ParserController

```
<?php

namespace app\controllers;


use app\models\Family;
use app\models\Parser;
use yii\web\Controller;

class ParserController extends Controller
{
    public function actionParser()
    {
        $parser = new Parser();
        $family = Family::findOne(['ID_user' => \Yii::$app->user->id]);
        if ($family->gedcom != null)
        {
            $parser->parse($family->gedcom);
        }

        return $this->redirect(['person/index']);
    }
}
```

# ПРИЛОЖЕНИЕ С

Листинг С.1 – Исходный код модели Parser

```
<?php
/**
 * Created by PhpStorm.
 * User: artem
 * Date: 05.06.2019
 * Time: 1:48
 */

namespace app\models;

use phpDocumentor\Reflection\DocBlock\Tags\Param;
use yii\base\Model;
use Yii;
use app\models\Person;
use app\models\Family;

class Parser extends Model
{
    public function parse($file)
    {
        $first_name = '';
        $last_name = '';
        $date_of_birth = '';
        $place_of_birth = '';
        $live = 1;
        $gender = 0; $ID_ged = ''; $FAMC_ged = ''; $FAMS_ged = '';

        $family = Family::findOne(['ID_user' => Yii::$app->user->id]);
        $fh = fopen("uploads/" . $file, "r");
        $birt = 0;
        $fam = '';

        if ($fh) {
            while (($line = fgets($fh)) !== false) {
                if (!empty($line))
                {
                    $params = explode(' ', $line);
//                    for ($i = 0; $i < 3; $i++)
//                    {
//                        var_dump($params[$i]);
//                    }
                      switch ($params[2]) {
                          case 'INDI' . "\r\n": {

                              $ID_ged = $params[1] . "\r\n" ;
                              break;
                          }

                          case 'FAM' . "\r\n":{
                              $fam = $params[1];
                              break;
                          }
                      }
                      switch ($params[1]) {
                          case 'NAME': {
```

Продолжение приложения С

```
                                    $last_name = $params [2];
                                    if ($params[3] != null)
                                    {
                                        $first_name = $params[3];
                                    }

                                    break;
                                }
                                case 'SEX': {
                                    if ($params[2] == 'M' . "\r\n")
                                    {
                                        $gender = 1;
                                    }
                                    else if ($params[2] == 'F' . "\r\n")
                                    {
                                        $gender = 0;
                                    }
                                    break;
                                }
                                case 'BIRT' . "\r\n": {
                                    $birt = 1;
                                    break;
                                }
                                case 'DATE': {
                                    if ($birt)
                                    {
                                        $date_of_birth  =  $params[2]  .  '.'  .
$params[3]. '.' . $params[4];
                                    }
                                    break;
                                }
                                case 'PLAC': {
                                    if ($birt) {
                                        $place_of_birth = $params[2];
                                    }
                                    $birt = 0;
                                }
                                case 'DEAT': {
                                    $live = 0;
                                    break;
                                }
                                case 'FAMC': {
                                    $FAMC_ged = $params[2];
                                    if                  ($full                  =
Parser::createPersonGedcom($first_name, $last_name, $date_of_birth, $place_of_birth,
$live, $gender,
                                        $ID_ged, $FAMC_ged, $FAMS_ged)){
                                        break;
                                    }
                                    else {
                                        var_dump('ERRORTUT'); die();
                                    }
                                    break;
                                }
                                case 'FAMS': {
                                    $FAMS_ged = $params[2];
                                    break;
                                }
                                case 'HUSB': {
```

Продолжение приложения С

```
                                    $husband = Person::find()->where(['ID_ged' =>
$params[2]])->one();

                                    break;
                                }
                                case 'WIFE': {
                                    $wife  =  Person::find()->where(['ID_ged'  =>
$params[2]])->one();
      //                                          var_dump($husband->ID_person);
var_dump($wife->ID_person);
                                    $wife->ID_spouse = $husband->ID_person;
                                    $husband->ID_spouse = $wife->ID_person;
                                    $wife->save();
                                    $husband->save();
                                    break;
                                }
                                case 'CHIL': {
                                    $child  =  Person::find()->where(['ID_ged'  =>
$params[2]])->one();
      //                                    var_dump($child->ID_person); die();
                                    $father          =          Person::find()-
>where(['FAMS_ged' => $fam . "\r\n"])->andWhere(['gender' => 1])->one();
                                    $mother          =          Person::find()-
>where(['FAMS_ged' => $fam . "\r\n"])->andWhere(['gender' => 0])->one();

                                    if ($child->ID_person !== null)
                                    {
                                        $child->ID_father      =      $father-
>ID_person;
                                        $child->ID_mother      =      $mother-
>ID_person;
                                    }
                                    else break;


                                    if ($child->save() ? true : false)
                                    {
                                        break;
                                    }
                                    else {
                                        var_dump('ERROR'); die();
                                    }
                                }

                            }

                        }
                    }
                }
            }

            public      function      createPersonGedcom($first_name,      $last_name,
$date_of_birth, $place_of_birth, $live, $gender, $ID_ged, $FAMC_ged, $FAMS_ged)
            {
                $family = Family::findOne(['ID_user' => Yii::$app->user->id]);
                $person = new \app\models\Person();
                $person->first_name = $first_name;
                $person->last_name = $last_name;
                $person->date_of_birth = $date_of_birth;
```

```
        $person->place_of_birth = $place_of_birth;
        $person->live = $live;
```

Окончание приложения С

```
        $person->gender = $gender;
        $person->ID_ged = $ID_ged;
        $person->FAMC_ged = $FAMC_ged;
        $person->FAMS_ged = $FAMS_ged;
        $person->ID_family = $family->ID_family;
        return $person->save() ? true : false;
    }
}
```