

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет «Вычислительная техника» (наименование)	Кафедра «Системы автоматизированного проектирования» (наименование)
Направление подготовки	09.03.01 «Информатика и вычислительная техника» (код и наименование)
Профиль	«Системы автоматизированного проектирования» (наименование)

БАКАЛАВРСКАЯ РАБОТА

на тему

Разработка информационной системы
поиска учебных заведений по направлению
подготовки

Студент

 (подпись, дата)	Абдураззов Р.Т. (фамилия, инициалы)
--	--

Руководитель

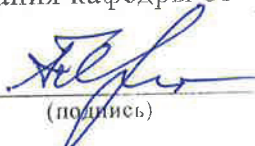
Бур- 1.06.17 (подпись, дата)	Бурмистов М.Н. (фамилия, инициалы)
---------------------------------	---------------------------------------

Нормоконтролёр

проф. каф. САПР ПГУ (должность, место работы)	 (подпись, дата)	Терзетов С.С. (фамилия, инициалы)
--	---	--------------------------------------

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

 (подпись)	Терзетов А.М. (фамилия, инициалы)
---	--------------------------------------

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

_____	_____
(подпись)	(фамилия, инициалы)

Пенза 2017

Реферат

Пояснительная записка: 75 листов, 19 рисунков, 12 источников, 1 приложение.

ИНФОРМАЦИОННАЯ СИСТЕМА ПОИСКА, ПОИСКА ВУЗОВ, WEB-ПРИЛОЖЕНИЕ, RUBY, RUBY ON RAILS, POSTGRESQL, NGINX, HTML5, CSS.

Объектом разработки является информационная система поиска высших учебных заведений, осуществляющих подготовку по определенным направлениям бакалавриата.

Цель работы – разработать информационную систему поиска высших учебных заведений, осуществляющих подготовку по определенным направлениям бакалавриата.

В процессе разработки проводилось изучение принципов построения WEB-приложений, в частности технологии разработки приложений с использованием фреймворка Ruby on Rails.

В результате проведенной работы создан проект, удовлетворяющий поставленной задаче.

Разработанную систему следует применять для поиска информации о высших учебных заведениях.

					ВКР 090301.01 17 81 01			
Изм.	Лист	№ докум.	Подпись	Дата	Разработка информационной системы (см. тит. лист) Пояснительная записка	Лит.	Лист	Листов
Разраб.		Абдразяков Р.Т						
Провер.		Бурукина И.П					3	75
Реценз.						ФВТ, каф. САПР, группа 13ВВ2		
Н. Контр		Бождай А.С						
Утверд.		Бершадский А.						

СОДЕРЖАНИЕ

ПЕРЧЕНЬ СОКРАЩЕНИЙ	7
ВВЕДЕНИЕ	8
1. ПОСТАНОВКА ЗАДАЧИ.....	9
2. ОБЗОР ТЕХНОЛОГИЙ WEB-ПРОГРАММИРОВАНИЯ	10
2.1. Выбор архитектуры приложения.....	10
2.2. Язык программирования Ruby.....	13
2.3. Фреймворк Ruby on Rails	14
2.4. Среда разработки Ruby Mine	15
2.5. SQLite3	15
2.6. PostgreSQL	15
2.7. Slim.....	16
2.8. Scss	17
2.9. Ubuntu.....	17
2.10. Proху-сервер.....	17
3. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.....	18
3.1. Описание метода разработки.....	18
3.2. Описание логической структуры.....	19
3.3. Разработка структуры БД.....	21
3.5. Проектирование контроллеров.....	27
3.6. Реализация загрузки фотографий	28
3.7. Создание панели администратора	30
3.8. Обработка URL.....	32
3.9. Файлы представления	33
3.10. Организация поиска ВУЗов	35

3.11. Разработка дизайна приложения	38
3.11.1. Общий стиль	38
3.11.2. Страница показа информации о ВУЗе.....	38
3.11.3. Список ВУЗов	41
3.11.4. Панель администратора.....	42
3.11.5. Адаптивная верстка	43
3.12. Тестирование	45
3.13. Размещение на сервере.....	45
4. ФУНКЦИОНИРОВАНИЕ ПРОГРАММЫ	47
4.1. Диаграмма использования	47
4.2. Описание схемы работы системы.....	47
4.3. Описание схемы взаимодействия программ.....	48
5. ТЕХНОЛОГИЧЕСКАЯ ИНСТРУКЦИЯ ПО ИСПОЛЬЗОВАНИЮ ПРОГРАММЫ.....	51
5.1. Описание программы	51
5.2. Добавление ВУЗа.....	53
5.3. Добавление направления подготовки.....	54
5.4. Основные ошибки и способы их устранения.....	54
5.4.1. Ошибка 404	54
5.4.2. Ошибка 502	55
5.4.3. Ошибка 500	56
6. ЗАКЛЮЧЕНИЕ	57
7. СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ.....	58
ПРИЛОЖЕНИЕ А. СХЕМЫ.....	60
ПРИЛОЖЕНИЕ Б. ЛИСТИНГИ	
Приложение Б1. Модели	62
Приложение Б2. Контроллеры.....	64
Приложение Б3. База данных	67

Приложение Б4. Панель администратора 69

Приложение Б5. Представление 72

ПЕРЧЕНЬ СОКРАЩЕНИЙ

ВУЗ – высшее учебное заведение;

ROR – (Ruby on Rails) фреймворк для создания web-приложений на языке ruby;

MVC – (Model-View-Controller) модель-представление-контроллер;

ORM – (Object relative mapping) объектно-реляционное отображение;

IDE – (Integrated Development Environment) интегрированная среда разработки;

CRUD – (Create Read Update Delete) создать прочитать обновить удалить;

REST – (Representational State Transfer) архитектурный стиль взаимодействия компонентов распределённого приложения в сети;

СУБД – система управления базой данных

ОС – операционная система;

ПО – программное обеспечение;

ПГУ – Пензенский государственный университет;

ЕСПД – единая система программной документации;

БД – база данных.

					ВКР 090301.01 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		7

ВВЕДЕНИЕ

Система высшего образования объединяет 446 государственных и 358 негосударственных вузов, в которых обучается 4,7 млн. человек. После относительного спада в середине 90-х годов число студентов ежегодно растет. В 2016 г. число студентов на 10 тысяч населения составило 327 человек, что значительно выше, чем соответствующий показатель 1995 г. - 189 студентов на 10 тысяч населения. Вузы России готовят специалистов по более чем 350 специальностям [13].

Перед выпускником школы встает проблема поиска ВУЗа для получения высшего профессионального образования. Выпускник, в основном, знает только о существовании основных российских ВУЗов и самых крупных ВУЗах города. Но многие учебные заведения предлагают высокое качество образования на выгодных условиях, найти информацию о которых достаточно проблематично.

Разрабатываемая система соединит в себе информацию о всех ВУЗах Российской Федерации, что позволит абитуриентам выбрать наиболее подходящее учебное заведение, основываясь на объективных данных, независимом рейтинге, а также пользовательских отзывах.

Таким образом, разработка данного программного продукта способна решить достаточно актуальную проблему.

					<i>ВКР 090301.01 17 81 01</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		8

1. ПОСТАНОВКА ЗАДАЧИ

Задачей дипломного проектирования является написание и отладка информационной системы поиска высших учебных заведений, осуществляющих подготовку по определенным направлениям бакалавриата.

Система должна быть представлена WEB-сервисом, в котором абитуриент может найти ВУЗ и узнать обо всех направлениях подготовки по программе бакалавриата, либо, зная необходимое направление подготовки, найти все учебные заведения, где осуществляется подготовка по данным направлениям.

Пользователи должны иметь возможность оставить отзыв об учебном заведении. Отзывы бывших студентов и работодателей, предоставляющих работу выпускникам данных вузов полезна для абитуриентов.

Система должна иметь административную панель, в которой администратор сможет управлять работой сервиса, добавлять, удалять или изменять информацию об учебных заведениях. Также администратор имеет возможность модерировать отзывы об учебных заведениях.

Система должны иметь удобный пользовательский интерфейс, современный дизайн и адаптивную верстку. Система должны корректно работать на всех системах и браузерах, начиная с Internet Explorer 6.

					<i>ВКР 090301.01 17 81 01</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		9

2. ОБЗОР ТЕХНОЛОГИЙ WEB-ПРОГРАММИРОВАНИЯ

2.1. Выбор архитектуры приложения

Программная система должна быть сетевой и многопользовательской. База данных системы будет располагаться на сервере и доступ к ней должен быть получен с нескольких компьютеров.

Чтобы выполнить данные требования, будет применяться технология «клиент-сервер».

Серверная часть веб-приложения — это программа или скрипт на сервере, обрабатывающая запросы пользователя (точнее, запросы браузера). Чаще всего серверная часть веб-приложения программируется на PHP. При каждом переходе пользователя по ссылке браузер отправляет запрос к серверу. Сервер обрабатывает этот запрос, вызывая некоторый PHP-скрипт, который формирует веб-страницу, описанную языком HTML, и отправляет клиенту по сети. Браузер тут же отображает полученный результат в виде очередной веб-страницы [6].

Клиентская часть веб приложения — это графический интерфейс. Он отображается в браузере. Пользователь взаимодействует с веб-приложением именно через браузер, кликая по ссылкам и кнопкам.

База данных (БД, или система управления базами данных, СУБД) — программное обеспечение на сервере, занимающееся хранением данных и их выдачей в нужный момент. Серверная часть веб-приложения т.е., PHP скрипт, обращается к базе данных, извлекая данные, которые необходимы для формирования страницы, запрошенной пользователем.

«Клиент-сервер» означает такой способ взаимодействия программных компонентов, при котором они образуют единую систему. Как видно из самого названия, существует некий клиентский процесс, требующий определённых ресурсов, а также серверный процесс, который эти ресурсы предоставляет.

При этом совсем необязательно, чтобы они находились на одном и том же компьютере. На практике принято размещать сервер на одном узле локальной сети, а клиенты — на других узлах. На рисунке 1 показана архитектура типа «клиент-сервер».

Клиент управляет пользовательским интерфейсом и логикой приложения, действуя как сложная рабочая станция, на которой выполняются приложения баз данных. Клиент принимает от пользователя запрос, проверяет синтаксис и генерирует запрос к базе данных на языке SQL, который соответствует логике приложения. Затем он передаёт сообщение серверу, ожидает поступления ответа и форматирует полученные данные для представления их пользователю. Сервер принимает и обрабатывает запросы к базе данных, а затем передаёт полученные результаты обратно клиенту.

Такая обработка включает проверку полномочий клиента, обеспечение требований целостности, поддержку системного каталога, а также выполнение запроса и обновление данных. Помимо этого, поддерживается управление параллельностью и восстановлением.

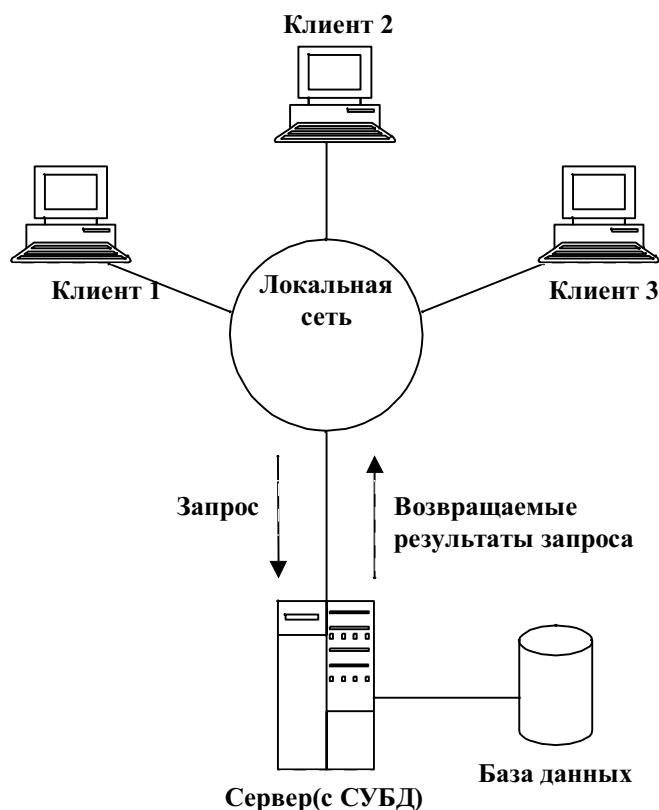


Рисунок 1 – Схема построения архитектуры «клиент/сервер»

Преимущества данного типа архитектуры:

1. Обеспечивается более широкий доступ к существующим базам данных.
2. Повышается общая производительность системы. Поскольку клиенты и сервер находятся на разных компьютерах, их процессоры способны выполнять приложения параллельно. При этом настройка производительности компьютера с сервером упрощается, если на нём выполняется только работа с базой данных.
3. Стоимость аппаратного обеспечения снижается. Достаточно мощный компьютер с большим устройством хранения нужен только серверу – для хранения и управления базой данных.
4. Сокращаются коммуникационные расходы. Приложения выполняют часть операций на клиентских компьютерах и посылают через сеть

только запросы к базе данных, что позволяет существенно сократить объём пересылаемых по сети данных.

5. Повышается уровень непротиворечивости данных. Сервер может самостоятельно управлять проверкой целостности данных, поскольку все ограничения определяются и проверяются только в одном месте. При этом каждому приложению не придётся выполнять собственную проверку.

2.2. Язык программирования Ruby

Ruby (англ. *Ruby* — рубин) — динамический, рефлексивный, интерпретируемый высокоуровневый язык программирования для быстрого и удобного объектно-ориентированного программирования. Язык обладает независимой от операционной системы реализацией многопоточности, строгой динамической типизацией, сборщиком мусора и многими другими возможностями. По особенностям синтаксиса он близок к языкам Perl и Eiffel, по объектно-ориентированному подходу — к Smalltalk. Также некоторые черты языка взяты из Python, Lisp, Dylan и Клу [1].

К достоинствам данного языка программирования можно отнести:

- Имеет лаконичный и простой синтаксис.
- Позволяет обрабатывать исключения в стиле Java, Python, C#.
- Позволяет переопределять операторы, которые на самом деле являются методами.
- Полностью объектно-ориентированный язык программирования. Все данные в Ruby являются объектами. Единственное исключение — управляющие конструкции, которые не являются объектами.
- Не поддерживает множественное наследование, но вместо него может использоваться концепция «примесей», основанная в данном языке на механизме модулей.

					ВКР 090301.01 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		13

- Содержит автоматический сборщик мусора. Он работает для всех объектов Ruby, в том числе для внешних библиотек.
- Поддерживает замыкания с полной привязкой к переменным.
- Поддерживает блоки кода (код заключается в { ... } или do ... end).
- Не требует предварительного объявления переменных. Язык использует простые соглашения для обозначения области видимости.
- В Ruby непосредственно в языке реализованы многие шаблоны проектирования.
- Может динамически загружать расширения, если это позволяет операционная система.

2.3. Фреймворк Ruby on Rails

Для разработки был выбран фреймворк Ruby on Rails.

Ruby on Rails (RoR) — фреймворк, написанный на языке программирования Ruby, реализует архитектурный шаблон Model-View-Controller для веб-приложений, а также обеспечивает их интеграцию с веб-сервером и сервером баз данных. Является открытым программным обеспечением и распространяется под лицензией MIT [2].

Причины выбора:

1. Большая распространенность. Существует большое количество библиотек и готовых решений, позволяющих значительно ускорить процесс разработки. Также на stackoverflow.com можно найти ответ на практически любой вопрос, возникающий при разработке.
2. Основанность на MVC. Данный шаблон проектирования позволяет разграничить объекты, отвечающие за визуальное отображение, обработку web запросов и реализацию модели данных.
3. Поддержка ORM. Каждая таблица в БД может автоматически быть преобразована в класс, а каждая запись в экземпляре класса. Данный

функционал реализует стандартная библиотека ActiveRecord. Также все классы имеют встроенные функции работы с БД

4. Язык программирования Ruby. На данный момент язык считается наиболее объектно-ориентированным. Множество специфичных операторов и синтаксических конструкций позволяют сократить размер кода, а также вероятность появления ошибок в коде.

2.4. Среда разработки Ruby Mine

На сегодняшний день Ruby Mine самая распространенная IDE для разработки с использованием фреймворка ROR. На втором месте по популярности стоит связка текстовый редактор – консоль [3].

Причины выбора данной IDE:

1. Удобный пользовательский интерфейс;
2. Уникальные средства создания и оптимизации приложений;
3. Своевременные обновления, позволяющие IDE не отставать от быстрого развития фреймворка.

2.5. SQLite3

При разработке приложения использовалась СУБД SQLite3. Ruby on Rails работает с ней «из коробки», она быстроедействительна и легко переносится с компьютера на компьютер. Так же фреймворк абстрагируется от используемой базы данных и не имеет значения какую БД ты используешь при разработке, а какая используется в релизе.

2.6. PostgreSQL

Для размещения приложения на сервере было решено выбрать СУБД PostgreSQL. PostgreSQL – объектно-ориентированная СУБД с открытым исходным кодом. Основной причиной выбора послужила высокая

					ВКР 090301.01 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		15

безопасность и защищенность. Данная СУБД широко распространена и Ruby on Rails легко конфигурируется для работы с ней. На «Википедии» также указаны дополнительные возможности выбранной СУБД:

- Соблюдение принципов ACID;
- Соответствие стандартам ANSI SQL-92 и SQL-99;
- Поддержка запросов с OUTER JOIN, UNION, UNION ALL, EXCEPT, INTERSECT и подзапросов;
- Последовательности;
- Контроль целостности;
- Репликация;
- Общие табличные выражения и рекурсивные запросы;
- Аналитические функции;
- Поддержка Юникода (UTF-8);
- Поддержка регулярных выражений в стиле Perl;
- Встроенная поддержка SSL, SELinux и Kerberos;
- Протокол разделяемых блокировок;
- Подгружаемые расширения, поддерживающие SHA1, MD5, XML;
- Расширения для написания сложных выборок, отчётов и т.д. (API открыт);
- Средства для генерации совместимого с другими системами SQL-кода и импорта из других систем;
- Автономные блоки на доступных языках, а не только SQL [11].

2.7. Slim

Для создания HTML кода было решено выбрать шаблонизатор Slim. Разметка, написанная с его использованием, легко читается и понятна программисту, значительно компактнее HTML и быстро транслируется. При показе страницы, Ruby on Rails кеширует страницу, следовательно не будет проблем с производительностью.

					<i>ВКР 090301.01 17 81 01</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		16

2.8. Scss

Scss — это метаязык на основе CSS, предназначенный для увеличения уровня абстракции CSS кода и упрощения файлов каскадных таблиц стилей.

Причиной выбора данного препроцессора стилей послужила сложность создания иерархической структуры при использовании обычного CSS.

2.9. Ubuntu

Сайт было решено размещать на сервере с ОС Ubuntu. Ruby on Rails работают лучше всего под данной операционной системой. Некоторые библиотеки зависят от операционной системы и не работают под Windows. Так же сервера с данной операционной системой значительно дешевле.

2.10. Proxy-сервер

При размещении сайта на сервере было решено использовать два вида серверов: Backend и Frontend. Frontend-сервер получает web-запрос от сети и перенаправляет его на Backend-сервер, который в свою очередь обрабатывает его и возвращает ответ. В качестве Frontend-сервера был выбран Nginx. Отличительными особенностями которого являются легкая конфигурируемость, быстрое действие и, в отличие от Apache, он не создает дополнительных нитей для каждого запроса. Вопросы в выборе Backend-сервера не стояло. Для целей создаваемого приложения подходил любой. Выбран был Python, так как в отличие от остальных поддерживает web-сокеты «из коробки», незначительно быстрее остальных и имеет возможность распараллеливать запросы, а не выполнять поочередно (что важно только для высоконагруженных приложений) [6].

3. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1. Описание метода разработки

Проектирование программного обеспечения выполняется в несколько этапов, что соответствует основным этапам объектно-ориентированного анализа и проектирования программных систем:

- разработка спецификаций;
- проектирование и конструирование программы;
- доказательство правильности проекта (верификация);
- кодирование, т.е. описание программы на конкретном языке программирования;
- отладка и проверка правильности программы;
- доработка и улучшение;
- производство окончательного программного продукта.

Последовательный процесс объектно-ориентированного проектирования приведен на рисунке 2.

Он включает анализ требований к системе со стороны пользователя, анализ системы с точки зрения принципов ее функционирования и основных составных частей, проектирование и непосредственно кодирование с использованием определенного языка программирования.



Рисунок 2 - Процесс объектно-ориентированного анализа и проектирования системы

На первом этапе детально анализировалась основная цель разработки программы, и составлялось подробное описание всех действий, которые она должна выполнять. На данном этапе было необходимо дать ответы на следующие вопросы:

- какими должны быть входные данные;
- какие данные являются корректными, а какие ошибочными;
- кто будет использовать разработанное программное обеспечение, и каким должен быть интерфейс;
- какие ошибки необходимо выявлять, и какие сообщения необходимо выдавать пользователю при наличии ошибок;
- какими должны быть выходные данные;
- какая документация на разрабатываемую программу должна быть подготовлена.

3.2. Описание логической структуры

В качестве языка для создания спецификаций, конструирования, визуализирования и документирования функций данного программного

обеспечения применяется язык UML, который является языком визуального моделирования программного обеспечения, включающий в себя определенную систему условных обозначений (нотацию).

При проведении первого этапа проектирования была построена UML диаграмма вариантов использования.

Диаграммы вариантов использования отображают взаимодействие между вариантами использования, представляющими функции системы, и действующими лицами, представляющими людей или системы, получающие или передающие информацию в данную систему.

Для отражения последовательности передачи сообщений между объектами была построена диаграмма последовательностей действий.

Взаимодействие объектов в системе происходит посредством приема и передачи сообщений объектами-клиентами и обработки этих сообщений объектами-серверами. При этом в разных ситуациях одни и те же объекты могут выступать и в качестве клиентов, и в качестве серверов.

Для представления общей конфигурации и топологии распределенной программной системы была построена диаграмма развертывания.

Она применяется и содержит распределение компонентов по отдельным узлам системы. Кроме того, диаграмма развертывания показывает наличие физических соединений — маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы.

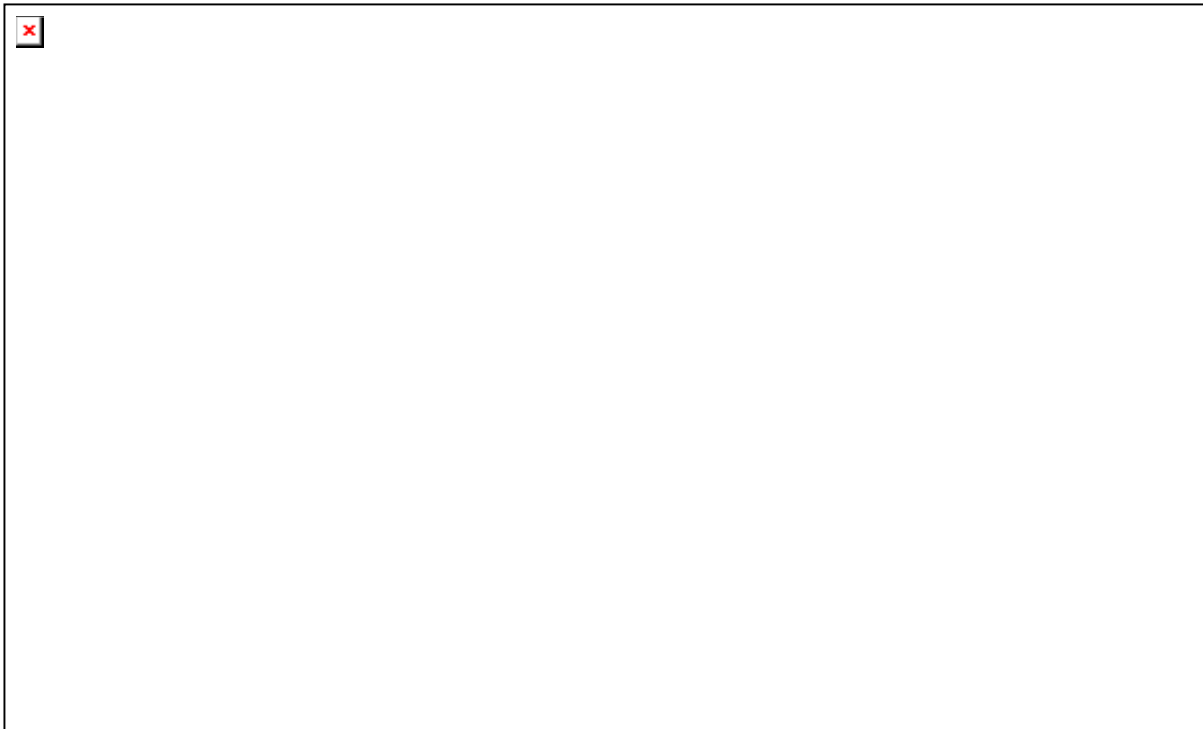


Рисунок 3 – Структурная схема

Структура разрабатываемой системы делится на две ветки: для использования администратором и для использования рядовым пользователем. Основываясь на методах аутентификации и авторизации, происходит разграничение доступа между этими ветками.

3.3. Разработка структуры БД

Структура базы данных формируется при помощи миграций. В миграции указываются изменения, которые необходимо произвести. Миграции можно накатывать и откатывать. По файлам миграции создается файл схемы базы данных [5].

Модель администратора создается в первую очередь. Для удобства именования, назовем ее пользователь. Так как на сервисе не требуется регистрация, все пользователи являются администраторами. Скрипт, представленный ниже создает таблицу в базе данных со всеми необходимыми данными:

```
class DeviseCreateUsers < ActiveRecord::Migration[ 5.0]
  def change
```

```

create_table :users do |t|
  ## Database authenticatable
  t.string :email, null: false, default: ""
  t.string :encrypted_password, null: false, default: ""

  ## Recoverable
  t.string :reset_password_token
  t.datetime :reset_password_sent_at

  ## Rememberable
  t.datetime :remember_created_at

  ## Trackable
  t.integer :sign_in_count, default: 0, null: false
  t.datetime :current_sign_in_at
  t.datetime :last_sign_in_at
  t.string :current_sign_in_ip
  t.string :last_sign_in_ip

  t.string :name

  t.timestamps null: false
end

add_index :users, :email, unique: true
add_index :users, :reset_password_token, unique: true
end
end

```

Для администратора создаются поля с информацией об адресе электронной почты, хешем пароля, датой регистрации, именем и дополнительной служебной информацией, необходимой для восстановления и защиты аккаунта. Так как функционал не требует регистрации пользователей на сайте, таблица называется именно администраторы, а не пользователи. Комментирование на сервисе будет реализовано при помощи дополнительного сервиса, реализующего все, связанное с аутентификацией и авторизацией пользователей. Модерирование комментариев будет проходить также на этом сервисе.

Затем создается таблица с информацией о ВУЗах. Таблица называется University для удобства.

					<i>ВКР 090301.01 17 81 01</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		22

```

class CreateUniversities < ActiveRecord::Migration[ 5.0]
  def change
    create_table :universities do |t|
      t.string      :avatar, null: false
      t.string      :description, null: false
      t.string      :address, null: false
      t.string      :name, null: false
      t.boolean     :state, null: false, default: true
      t.timestamps
    end
  end
end

```

Каждый кортеж в таблице с информацией о ВУЗах состоит из названия, адреса, описания и флага, является ли заведение частным или государственным. Поле описание является текстовым и способно хранить в себе текст значительного размера, в котором может быть размещена любая информация об учебном заведении. Дополнительно к таблице было добавлено поле с путем до аватара учебного заведения.

Далее создается таблица с информацией о направлении подготовки. Назовем ее «Study».

```

class CreateStudies < ActiveRecord::Migration[ 5.0]
  def change
    create_table :studies do |t|
      t.references :category, null: false
      t.string :name, null: false
      t.string :code, null: false
      t.string :description, null: false
      t.timestamps
    end
    create_table :categories do |t|
      t.string :name, null: false
      t.timestamps
    end
  end
end

```

В данной таблице хранится информация о направлениях подготовки. Обязательными полями являются: код специальности, название, описание направления подготовки (аналогично таблице с ВУЗами, поле способно хранить произвольный текст любой длины). Дополнительно была создана

таблица со списком категорий образовательных курсов. Данное решение позволит сгруппировать информацию и упростить ее поиск.

Между учебным заведением и направлением подготовки существует связь многие ко многим, для ее реализации создадим таблицу Relationships.

На эту таблицу будет также наложен класс ORM. Дополнительно к

```
class CreateRelationships < ActiveRecord::Migration[ 5.0]
  def change
    create_table :relationships do |t|
      t.references :university, null: false
      t.references :study, null: false
      t.decimal :cost, precision: 8, scale: 2, null:
false, default: 0
      t.timestamps
    end
    add_index :relationships, [:university_id, :study_id],
unique: true
  end
end
```

В данной таблице помимо ссылок на учебное заведение и направление подготовки, имеется информация о стоимости обучения. Для таблицы добавлены индексы, обеспечивающие быстрый поиск и уникальность записей на уровне базы данных.

Для каждой таблицы скрыто будет добавлен идентификатор, дата создания и последнего изменения записи.

После создания миграций, необходимо выполнить команду «rails db:migrate»

На рисунке ниже представлена физическая модель базы данных. База данных состоит из пяти таблиц.



Рисунок 4 – Концептуальная модель БД

3.4. Разработка моделей данных

К каждой созданной таблице необходимо создать модель, установить ограничения по данным, отношения между моделями.

В разработке данной квалификационной работы используется принцип объектно-реляционного отображения. Он подразумевает соотношение класса с таблицей в базе данных, где запись — это экземпляр класса, а поле — это свойство класса. Для реализации используется стандартная библиотека фреймворка Ruby on Rails – ActiveRecord. Данная библиотека инкапсулирует SQL-запросы и предоставляет их в форме методов и свойств класса. Библиотека имеет встроенные механизмы валидации данных и создания реляционных отношений между классами [12].

```
class User < ApplicationRecord
  devise :database_authenticatable,
         :recoverable, :rememberable, :trackable,
  :validatable
end
```

Данный класс реализует отношение многие ко многим между направлением подготовки и ВУЗом. Ruby on Rails реализует отношение многие ко многим автоматически, инкапсулируя создание дополнительной таблице. В разработке данного приложения таблица создавалась явно для того, чтобы добавить указание цены, по которой ВУЗ осуществляет подготовку специалистов [8].

```
class Relationship < ApplicationRecord
  belongs_to :university, dependment: destroy
  belongs_to :study, dependment: destroy
  validates_presence_of :university, :study
  validates_format_of :cost, :with =>
/\A\d+ (?:\.\d{ 0,2} )?\z/

  def self.min_cost
    d = Relationship.all.minimum(:cost)
    d == nil || d == 0 ? 1 : d
  end

  def self.max_cost
    d = Relationship.all.maximum(:cost)
    d == nil ? 0 : d
  end

end
```

У моделей ВУЗа и направления подготовки нужно сделать ссылку на модель отношения между ними и установить необходимость удаления записи из сводной таблицы, при удалении записи из одной из данных таблиц.

```
require 'file_size_validator'
class University < ApplicationRecord
  validates :avatar, file_size: { maximum: 4096.kilobytes }
  mount_uploader :avatar, AvatarUploader
  validates_presence_of :avatar, :description, :address,
:name, :state
  has_many :relationship
end

class Study < ApplicationRecord
  belongs_to :category
  has_many :relationship
end
```

```

        validates_presence_of :name, :code, :description,
:category
    end

```

3.5. Проектирование контроллеров

Дальше необходимо создать контроллеры для моделей. Контроллеры описывают все действия, которые можно произвести с моделью. В Ruby on Rails используется модель CRUD, но из нее будут реализованы только методы показа элемента и списка элементов, так как вся работа с данными будет производиться из административной панели администратором, а не пользователями. Данное решение позволяет увеличить защищенность системы и ее отказоустойчивость. Помимо того, данное решение исключить вероятность появления ложных сведений, так как перед размещением на сайте, все данные, предоставляемые представителями ВУЗов, будут повторно проверяться.

Ниже представлен контроллер модели University. Для модели Study контроллер аналогичный. Модель Relationship не имеет контроллера, так как она вспомогательная.

```

class UniversitiesController < ApplicationController

  def index
    p = params[:index]
    #запрос к базе данных будет выполнен после выхода из
этого метода, сейчас формируем запрос
    @universities = University.all.page(params[:page])
    if p != nil
      unless p[:query].blank?
        str = '%' + p[:query] + '%'
        puts str
        @universities = @universities.where('lower(name)
LIKE ? OR lower(description) LIKE ?', str, str)
      end
      unless p[:state].blank?
        @universities = @universities.where(state: p[:state]
== '1')
      end
    end
  end

```

```

        unless p[:category].blank?
          @universities =
            @universities.joins(:relationships).where('relationships.study_id = ?', p[:category])
        end
      end

      @universities = @universities.page(params[:page])
      respond_to do |format|
        format.js
        format.html
      end
    end

    def show
      @university = University.find_by_id params[:id]
    end

  end
end

```

Как можно заметить из исходного кода, для поиска ВУЗов реализован полнотекстовый поиск. Полнотекстовый поиск реализован без использования сторонних библиотек. Полнотекстовый поиск осуществляется при помощи ключевого слова «like». До и после слова для поиска добавлен символ вопроса, говорящий системе о том, что до и после ключевого слова может быть любой текст. Также текст в базе данных и поискового запроса приводятся к нижнему регистру. Команда «like» работает медленно на больших данных, но в разрабатываемой системе таких объемов информации не может быть.

3.6. Реализация загрузки фотографий

Для загрузки фотографий ВУЗов использовалась библиотека Carrierwave. Для подключения библиотеки в GemFile необходимо добавить строку:

```
gem 'carrierwave-crop'
```

А затем выполнить команду `bundle install`.

Загрузкой файлов в Ruby on Rails занимаются «Загрузчики». Для реализации собственного, необходимо унаследоваться от базового класса и перегрузить некоторые методы.

Ниже представлен загрузчик фотографий университетов.

```
class AvatarUploader < CarrierWave::Uploader::Base
  # Include RMagick or MiniMagick support:
  # include CarrierWave::RMagick
  include CarrierWave::MiniMagick
  configure do |config|
    config.permissions = 0644
    config.directory_permissions = 0777
    config.root = Rails.root + 'public'
  end
  # Choose what kind of storage to use for this uploader:
  storage :file
  # This is a sensible default for uploaders that are meant to
  be mounted:
  def store_dir

    "uploads/#{model.class.to_s.underscore}/#{mounted_as}/#{model.id}"
  end

  # Provide a default URL as a default if there hasn't been a
  file uploaded:
  def default_url
    version_name.to_s + '.png'
    #{} "/images/fallback/" + [version_name,
"default.png"].compact.join('_')
    #'original_avatar.gif' #rails will look at
'app/assets/images/default_avatar.png'
  end

  version :original

  # Create different versions of your uploaded files:
  version :x_large_avatar do
    process crop: :avatar
    process :resize_to_fill => [500, 500]
  end
  version :large_avatar do
    process crop: :avatar
    process :resize_to_fill => [200, 200]
```

					ВКР 090301.01 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		29

```

end
version :little_avatar do
  process crop: :avatar
  process :resize_to_fill => [ 75, 75]

  # Add a white list of extensions which are allowed to be
  uploaded.
  # For images you might use something like this:
  def extension_white_list
    %w(jpg jpeg gif png)
  end
end
end

```

Представленный выше код создает три версии изображения, подходящих для различных мест отображения. Также добавлен фильтр файлов по их разрешениям – нельзя загружать файлы, имеющие расширение, отличное от общепринятых разрешений изображений.

Для подключения загрузчика к модели необходимо добавить следующие строки в текст модели University:

```

mount_uploader :avatar, AvatarUploader
crop_uploaded :avatar

```

3.7. Создание панели администратора

Для реализации панели администратора использовалась библиотека «activeadmin». Библиотека проста в использовании и для ее настройки необходимо заполнить несколько файлов с описанием.

Для входа на сайт администратору необходимо авторизоваться на сайте. Форма авторизации представлена ниже.

					ВКР 090301.01 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		30

You need to sign in or sign up before continuing.

Skytravel Login

Email*

admin@example.com|

Password*

.....

☐ Remember me

Login

[Forgot your password?](#)

Рисунок 4 – Страница авторизации.

Сразу после аутентификации пользователь перенаправляется к странице управления информацией о ВУЗах.

Все файлы конфигураций состоят из трех необязательных блоков: конфигурирования списка, конфигурирование показа и конфигурирование редактирования. Так же имеются дополнительные блоки, но они не использовались в разработке приложения.

```
ActiveAdmin.register University do
  menu label: 'ВУЗы'
  permit_params :avatar, :description, :address, :name,
:state

  index do
    selectable_column
    column 'Аватар', :avatar do |u|
      image_tag u.avatar.small_avatar.url, height: 100
    end
    column 'Название', :name
    column 'Адрес', :address
    column 'Государственный', :state
    column :created_at
    actions defaults: true
  end
end
```

```

form do |f|
  f.inputs 'ВУЗ' do
    f.input :name, label: 'Имя'
    f.input :avatar, as: :file, label:
'Изображение (аватар)'
    f.input :state, label: 'Государственный'
    f.input :address, label: 'Адресс'
    f.input :description, as: :ckeditor, label: 'Описание'
  end
  f.actions
end

end

```

В данном файле конфигурируется только списка вузов и форма ввода информации. Метод показа подробной информации оставлен без ограничения. В нем для удобства администрирования будет отображена все информация «как есть». На рисунке ниже представлено отображение списка ВУЗов, сконфигурированное данным скриптом.

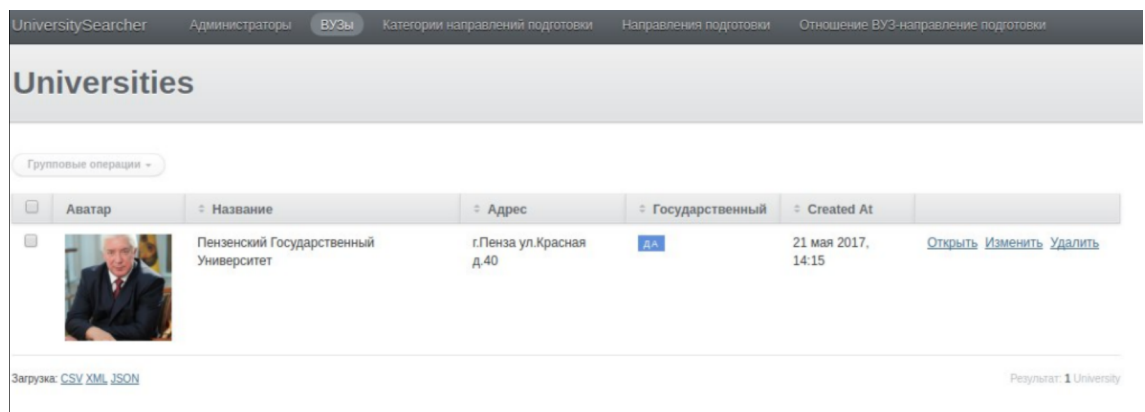


Рисунок 4 –Панель администратора.

3.8. Обработка URL

Для обработки различных URL и перенаправления их методам контроллера в ROR имеется файл «Routes». Для работы с моделями, реализующими методы CRUD, существует метод «resources» с указанием имени класса, для которого необходимо реализовать пути. Данный метод

можно расширить указав, какие методы не реализует контроллер модели, а какие реализует дополнительно.

Метод «root» описывает куда направлять запрос без указания пути. Первым параметром идет имя контроллера, вторым через шарп имя метода.

Методы контроллера, которые не относятся к моделям, описываются при помощи метода «match», где первым параметром идет URL, вторым метод котроллера, аналогично методу «root», тип запроса и псевдоним к данному пути.

```
Rails.application.routes.draw do
  devise_for :users, ActiveAdmin::Devise.config
  mount Ckeditor::Engine => '/ckeditor'
  default_url_options host: 'university_searcher.ru'
  ActiveAdmin.routes(self)
  root 'universities#index'

  resources :universities, only: [:index, :show]
  resources :studies, only: [:index, :show]
  resources :relationships, only: [:index, :show]

end
```

3.9. Файлы представления

После обрабатывания каждого метода контроллера происходит генерация файлов отображения. Контроллеры инициализируют глобальные переменные, которые затем использует представления для генерации представления.

В приложении был создан главный файл «application.html.slim», в котором хранится информация, общая для каждой страницы и отведено место для прорисовки содержимого страниц. Отрисовка происходит на месте функции «yield».

```
doctype html
html lang="ru"
head
  meta charset="utf-8"
  meta http-equiv="X-UA-Compatible" content="IE=edge"
```

					ВКР 090301.01 17 81 01	Лист
						33
Изм.	Лист	№ докум.	Подпись	Дата		

```

        meta name="viewport" content="width=device-width,
initial-scale=1.0"
        title= content_for?(:title) ? yield(:title) : 'Найти
ВУЗ'

        meta name="description"
content="#{ content_for?(:description) ? yield(:description) : '
информационная система поиска высших учебных заведений,
осуществляющих подготовку по определенным направлениям
бакалавриата' } "
        = csrf_meta_tags
        = stylesheet_link_tag 'tasks'
        = javascript_include_tag 'application', 'data-
turbolinks-track' => true
        = stylesheet_link_tag 'application', media: 'all',
'data-turbolinks-track' => true
        = stylesheet_link_tag 'font-awesome'
        = stylesheet_link_tag 'fancybox'
        /! Le HTML5 shim, for IE6-8 support of HTML elements
        /[ if lt IE 9]
        = javascript_include_tag
'//cdnjs.cloudflare.com/ajax/libs/html5shiv/3.7.2/html5shiv.min.
js'

    body
      #flash
        = bootstrap_flash
      =render 'layouts/header'
    main
      .row
        = yield
      =render 'layouts/footer'

```

Для отображения коллекций необходимо вызвать функцию «render» с параметром коллекции элементов. Она автоматически отрисует каждый элемент. Необходимо только создать файл с правилами отображения элемента. Если не указано имя файла, файл должен начинаться с подчеркивания и иметь именем название модели, которой принадлежит файл. Строка «render @universities», приведет к генерации для каждого элемента массива разметки из файла «_university.html.slim»(представлен ниже), в который будет передан параметр с именем в единственном числе. В нашем случае «vacancy».

```

.col.l4.m6.s12.list_tour
.tour
.center-align.image
    =image_tag university.avatar.small_avatar, style:
'width: 100%'
.text
    .destination= truncate(university.name, length: 34)
    .date
        span Адрес:
        b= ' ' + truncate(university.address, length: 34)
    .duration
        span Статус:
        b #{ university.state ? 'Государственный' :
'Частный'}
    =link_to 'Подробнее', university_path(university), class:
'btn'

```

3.10. Организация поиска ВУЗов

Для удобства поиска вакансий было решено автоматизировать поиск. Основными параметрами для поиска являются: ключевое слово для поиска, диапазон стоимости обучения по определенной специальности, ВУЗ и направление подготовки. В разрабатываемом сервисе всего три формы поиска: поиск ВУЗов, поиск направлений подготовки и поиск коммерческих предложений учебных заведений. На рисунке 5 представлена форма поиска высших учебных заведений.

Поисковой запрос
Пензенский

✓ Государственный
Направление подготовки

НАЙТИ

Рисунок 5 – Форма поиска ВУЗа.

На данной форма пользователь может указать ключевое слово для поиска, выбрать тип университета и направление подготовки, по которому он производит обучение.

В форме поиска коммерческих предложений функционал шире – можно также указать диапазон стоимости обучения. Выбор диапазона стоимости обучения дублируется двусторонним «ползунком». Крайние значения ограничиваются максимальным и минимальным значением стоимости года обучения по всем специальностям.

Для валидации данных формы и анимирования ее работы был написан исходный код на язык JavaScript с использованием библиотеки JQuery, представленный ниже. Логика работы формы проста – при генерации формы создаются два скрытых поля, в которых хранятся максимальный и минимальный размеры оклада. Первоначально они получают и устанавливаются как граничные для слайдера. [4]. На изменение положения слайдеров устанавливается слушатель, при изменении которого также изменяется значение полей ввода текстовых значений граничных величин [10].

```
$(document).ready(function(){
```

```

$start_cost = $("#index_start_cost");
$last_cost = $("#index_last_cost");

$slider = $("#slider-range");
min = parseInt($start_cost.val());
max = parseInt($last_cost.val());
$slider.slider({
    range: true,
    min: min,
    max: max,
    values: [ min, max] ,
    slide: function (event, ui) {
        $start_cost.val(ui.values[ 0] );
        $last_cost.val(ui.values[ 1] );
    }
});

$start_cost.on('input keyUp', function(){
    if(+$start_cost.val() > +$last_cost.val()){
        $last_cost.val(+$start_cost.val());
        $slider.slider("values", 1, [ +$start_cost.val()]);
    }
    $slider.slider("values", 0, [ +$start_cost.val()]);
});

$last_cost.on('input keyUp', function () {
    if (+$last_cost.val() < +$start_cost.val()) {
        $start_cost.val(+$last_cost.val());
        $slider.slider("values", 0, [ +$last_cost.val()]);
    }
    $slider.slider("values", 1, [ +$last_cost.val()]);
});
});

```

После нажатия кнопки «Поиск» происходит асинхронный запрос к серверу. Сервер обрабатывает все параметры, создает запрос к базе данных и возвращает результат также асинхронно. Ниже представлен код асинхронной перерисовки пользовательского интерфейса.

```

console.log("<%= @relationships.count %>");
$('#tours_list').html("<%= escape_javascript(render
@relationships) %>");
$('#paginator').html("<%=
escape_javascript(paginate(@relationships, remote: true)) %>");
$('#html, body').animate({ scrollTop: 0 }, 1000);

```

3.11. Разработка дизайна приложения

3.11.1. Общий стиль

Для создания современного дизайна приложения использовалась библиотека «materialize». Даная библиотека содержит в себе набор классов для создания внешнего вида приложения в стиле Material Design, а также библиотеки JavaScript для придания элементам анимации [9].

Ознакомиться с библиотекой можно по адресу: materializecss.com.

Библиотека имеет встроенный набор классов, используя которые, можно изменять внешний вид, не прибегая к помощи дизайнера и написания таблиц стилей.

3.11.2. Страница показа информации о ВУЗе

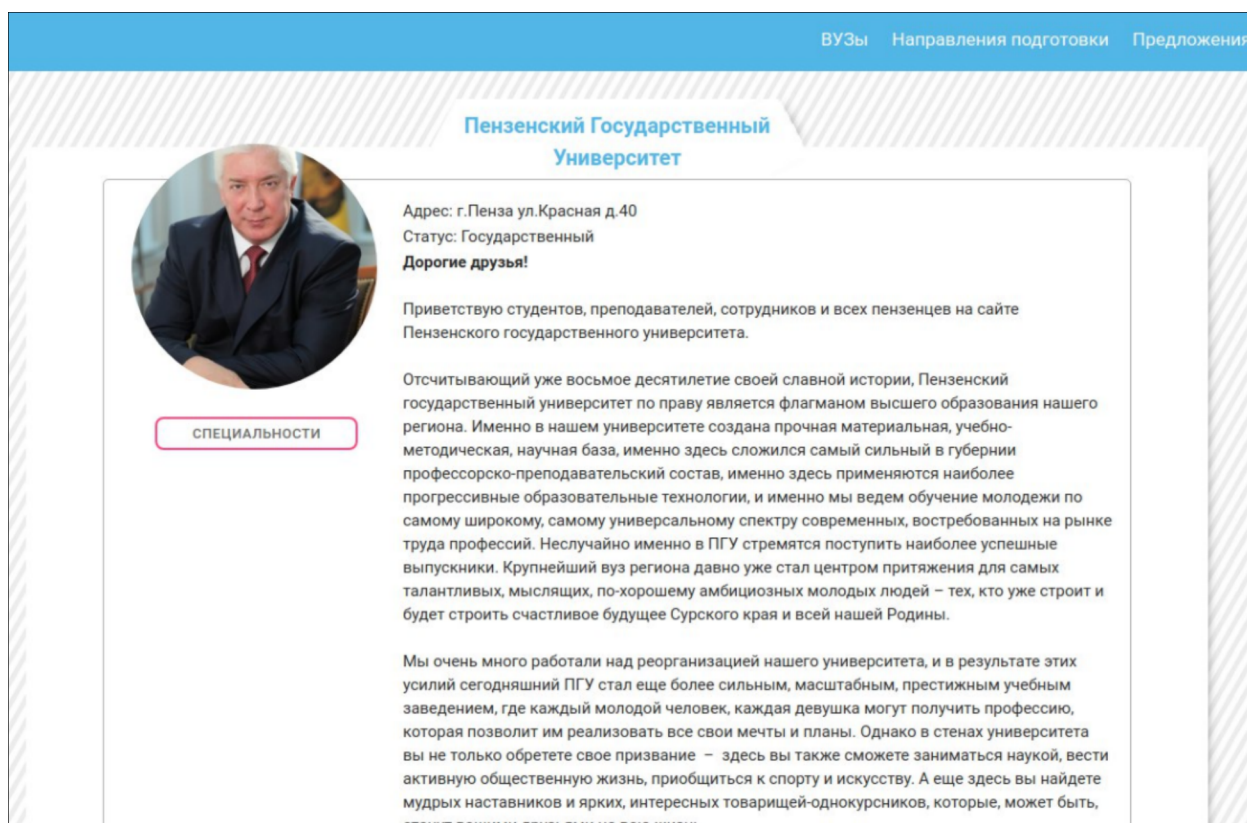


Рисунок 6 – Страница с информацией о ВУЗе.

						Лист
Изм.	Лист	№ докум.	Подпись	Дата	ВКР 090301.01 17 81 01	38

На рисунке 6 изображен снимок экрана страницы с отображением информации о ВУЗе.

Для придания уникальности дизайну разрабатываемого сайта использовались каскадные таблицы стилей третьего стандарта, а также препроцессор SCSS. Ниже представлен код, описывающий дизайн страницы, отображенной на рисунке 4.

```
#main_content {
  padding: 26px 10%;
  #trapeze {
    position: relative;
    left: 50%;
    margin-left: -175px;
    width: 350px;
    text-align: center;
    border-bottom: 45px solid white;
    border-left: 24px solid transparent;
    border-right: 24px solid transparent;
    height: 0;
    .text {
      padding-top: 10px;
      font-size: 20px;
      font-weight: 600;
      color: $blue;
    }
  }
}

#trapeze:before {
  box-shadow: 0 -8px 21px -15px #000;
  content: '';
  height: 64px;
  width: 65px;
  margin-top: 6px;
  margin-left: 104px;
  overflow: hidden;
  position: absolute;
  -webkit-transform: rotate(62deg);
  transform: rotate(62deg);
  z-index: 1;
}

.items {
  box-shadow: 8px 8px 18px -13px #000;
  margin-bottom: 0;
```

```

background-color: white;
padding: 2rem;
.col {
  padding: 8px;
  .tour {
    height: 270px;
    position: relative;
    border: 1px solid #bbbbbb;
    border-radius: 6px;
    padding: 6px;
    .image {
      position: relative;
      img {
        border-radius: 500px;
        margin-top: -36px;
        max-width: 200px;
      }
      .cost {
        position: absolute;
        right: 50%;
        margin-right: -96px;
        bottom: 4px;
      }
    }
    .text {
      padding-bottom: 16px;
    }
    .btn {
      bottom: -12px;
      position: absolute;
      margin-top: -8px;
      padding-left: 0;
      padding-right: 0;
      font-size: 14px;
      box-shadow: 0 0 0;
      border-radius: 6px;
      font-weight: 500;
      text-transform: capitalize;
      height: 26px;
      line-height: 26px;
      width: 134px;
      left: 50%;
      margin-left: -67px;
    }
  }
}

```

```

    }
  }
}

```

3.11.3. Список ВУЗов

Страница авторизации пользователя выполнена так же в стиле Material Design. Пользователь может ввести свои авторизационные данные или зайти через социальные сети. В случае, если у него нет профиля, он может перейти к странице регистрации.

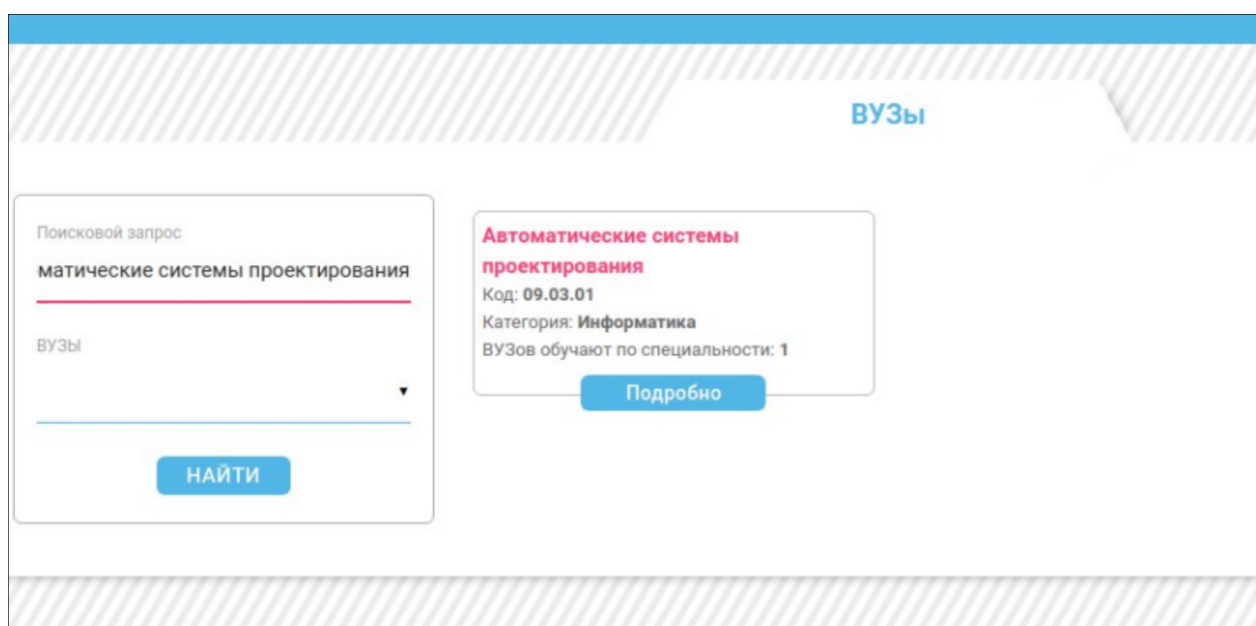


Рисунок 7 – Результат поиска направления подготовки.

Для компоновки элементов на экране использовался прием 12 колонок. Независимо от размера браузера, каждая из этих колонок всегда будет иметь одинаковую ширину.

Основываясь на данной методологии, список ВУЗов занимает 7 колонок, форма поиска оставшиеся 5 колонок. После перехода в мобильную версию каждый блок занимает по 12 колонок, вследствие чего блоки занимают всю ширину и находятся друг под другом.

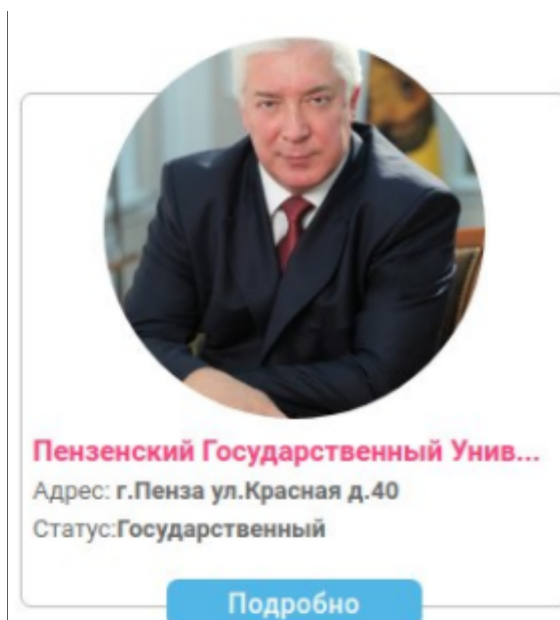


Рисунок 9 – Информация о ВУЗе в мобильной версии.

Подробнее о реализации мобильной версии описано далее

3.11.4. Панель администратора

Модуль панели администратора имеет стандартную верстку. Собственные стили к ней не создавались, все изменения имели только структурный характер и прописывались в конфигурационных файлах.

Все данные на сервис заносятся администратором из панели администрирования. Для занесения информации о новом учебном заведении, необходимо перейти на вкладку «ВУЗы» и нажать кнопку «Создать ВУЗ».

Рисунок 11 – Форма создания нового ВУЗа

Форма, представленная на рисунке 11 создается автоматически. Данные о необходимых полях и типах данных берутся из базы данных. Существует возможность изменения данной формы, но в рамках данной работы этого не потребовалось. В случае ввода некорректных данных, у неверно введенных полей будут выведены читаемые сообщения.

3.11.5. Адаптивная верстка

Разработанное в ходе выполнения дипломного проектирования приложение имеет адаптивную верстку, то есть корректно отображается как на настольном компьютере, так и на мобильном телефоне.

В основе адаптации лежат медиазапросы. Медиазапросы, согласно свойствам экрана, позволяют устанавливать различные стили. Библиотека «Materialize» сделала на основе медиазапросов. Набор классов для больших экранов (префикс «l»), средних экранов (префикс «m») и маленьких (префикс

«S»). Использование данных классов позволяет значительно упростить адаптацию под мобильные телефоны. В вышеупомянутой библиотеке большими экранами считаются экраны с разрешением больше 992 пикселей, средними экранами в промежутке между 600 и 992 пикселями, маленькими с разрешением менее 600 пикселей.

Для целей данного дипломного проекта было недостаточно использования данных классов и были разработаны собственные с использованием медиазапросов.

Представленный ниже исходный код демонстрирует использование медиазапросов. В нем, в зависимости от ширины экрана, устанавливаются различные отступы от краев экрана.

```
@media (max-width: 1992px) {
    .container {
        width: 80% !important;
    }
}

@media (max-width: 992px) {
    .container {
        width: 85% !important;
    }
}

@media (max-width: 600px) {
    .container {
        width: 100% !important;
    }
}
```

Библиотеке «Materialize» позволяет использовать различные типы меню для различных размеров экрана. На экране настольного компьютера меню отображается так же, как на рисунке 4. В мобильной же версии, меню преобразуется к боковому меню. Мобильный тип меню аналогичен стандартному для ОС Android «DrawerLayout».

3.12. Тестирование

Приложение тестировалось двумя различными методами: поведенческим и при помощи Unit-тестов.

Поведенческое тестирование проходило в ручном режиме. В ходе него было выявлено множество ошибок, которые впоследствии были исправлены. В данном виде тестирования моделируется поведение пользователя.

При разработке частично использовался метод TDD (Test Drive Development), согласно которому сначала пишется тест, покрывающий желаемые изменения, затем пишется код, позволяющий пройти тест, затем проводится рефакторинг написанного кода и цикл повторяется.

3.13. Размещение на сервере

Для размещения на сервере необходимо установить Nginx, а также Ruby on Rails, аналогично пунктам 3.2.2 и 3.2.3.

Nginx устанавливается на сервер при помощи двух команд:

```
sudo apt-get update  
sudo apt-get install nginx
```

Затем его необходимо настроить. Настройка прокси-сервера происходит путем соотнесения DNS-адреса, на запросы к которому необходимо отвечать сайту, а также указанию пути да файлов с номером процесса приложения и номером сокета, через который общаются Backend и Frontend сервера.

Для размещения самого приложения воспользовались библиотекой «Capistrano». Для корректной работы библиотеки необходимо подключить все необходимые модули, а также написать скрипт «деплой».

Дополнительно необходимо указать в приложении IP-адреса, по которым будет происходить размещение, а также конфигурацию приложения. Поддерживается указание различных серверов для различных

конфигураций, для того чтобы исключить возможность повреждения данных при тестировании приложения.

4. ФУНКЦИОНИРОВАНИЕ ПРОГРАММЫ

4.1. Диаграмма использования

На рисунке 12 представлена диаграмма использования. На диаграмме представлено два субъекта: администратор и пользователь. Как видно из диаграммы, управление информацией на сервисе занимается администратор, пользователь является лишь потребителем информации. Диаграмма использования подтверждает, что разработанный сервис полностью соответствует техническому заданию.

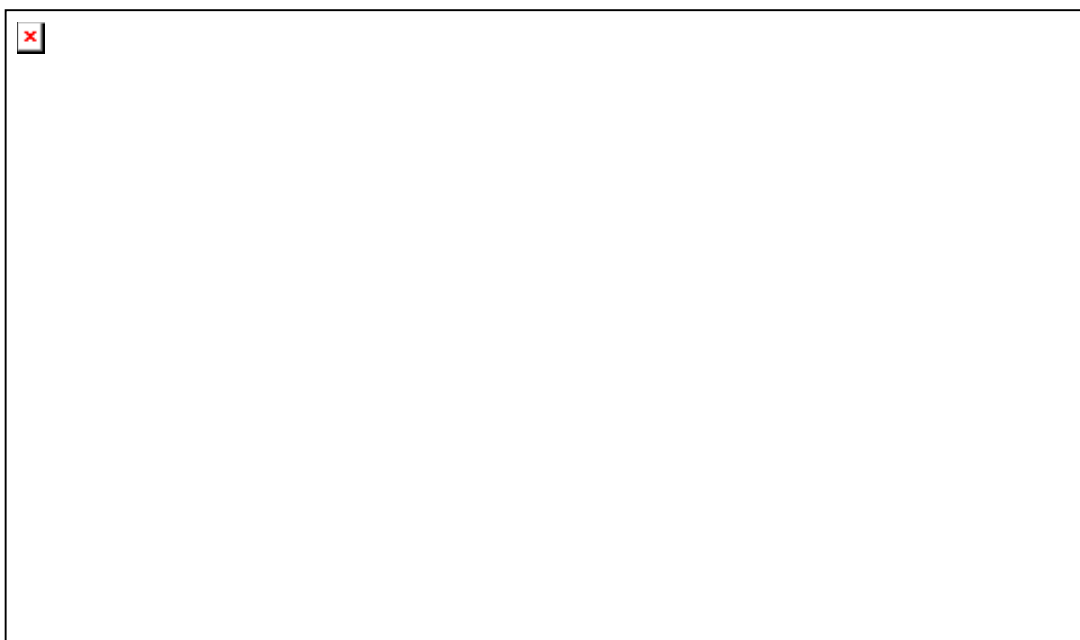


Рисунок 12 –Диаграмма использования

4.2. Описание схемы работы системы

На рисунке 13 представлена схема работы системы. Пользователь отправляет запросы к приложению через сеть интернет, прокси-сервер Nginx получает их и перенаправляет приложению. Приложение в свою очередь получает данные из базы данных, представляет их в нужном виде и передает HTML-страницы обратно через сеть интернет.



Рисунок 13 – Схема работы системы

Данная схема является стандартной для web-приложений. Необязательным элементом на данной схеме является Nginx. Система без него работоспособна, но будет ряд ограничений, самым важным из которых будет невозможность разместить на одном IP-адресе нескольких web-приложений. Если рассматривать схему без привязки к конкретному приложению, схема превращается в дерево, с многими приложениями, связанными с Nginx. Во избежание монопольного использования ресурсов сервера была использована данная схема.

4.3. Описание схемы взаимодействия программ

На рисунке 14 представлена схема взаимодействия программ. Цифрами на данном рисунке обозначена последовательность взаимодействия. Также цифры обозначают разную форма представляемых данных.



Рисунок 14 – Схема взаимодействия программ

Система состоит из 7 компонентов. Рассмотрим подробно работу данной системы (работа системы будет рассмотрена применительно для просмотра списка номеров):

1. Все начинается с того, что пользователь в браузере отправляет запрос на открытие списка номеров. Пользователь передает приложению имя страницы;
2. Запрос приходит в маршрутизатор приложения. Маршрутизатор отвечает за соответствие путей и методов контроллеров. Получив путь до страницы, маршрутизатор определяет контроллер и метод контроллера, который должен обработать данный запрос;
3. Контроллер, получив управление, должен подготовить данные для отображения.
4. Модель отправляет запрос к базе данных.
Полученные от базы данных данные модель преобразует их в объекты класса и массивом возвращает контроллеру;
5. Получив необходимые данные, контроллер начинает подготовку HTML-страницы для отображения, для этого он передает управление представлению;
6. Представление, получив управление имеет в распоряжении глобальные переменные, которые создал контроллер. С их использованием происходит создание HTML-страницы;
7. Подготовив все необходимые данные для отображения, представление возвращает управление контроллеру;
8. Контроллер возвращает браузеру HTML-страницу. Браузер при необходимости напрямую получает файлы стилей и изображения, в обход данной схемы.

Данная схема является стандартной не только для любого Ruby on Rails приложения, но и с небольшими корректировками может подойти для

					<i>ВКР 090301.01 17 81 01</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		49

любого web-приложения, созданного с использованием паттерна проектирования MVC.

5. ТЕХНОЛОГИЧЕСКАЯ ИНСТРУКЦИЯ ПО ИСПОЛЬЗОВАНИЮ ПРОГРАММЫ

5.1. Описание программы

Разработанная система представлена WEB-сервисом, в котором пользователь может найти высшее учебное заведение, осуществляющее подготовку студентов по определенным направлениям бакалавриата.

Система имеет административную панель, в которой администратор размещает информацию о высших учебных заведениях и направлениях подготовки по программам обучения бакалавриата.

В приложении реализован полнотекстовый поиск высших учебных заведений по ключевым словам, что делает ресурс максимально дружелюбным для пользователя.

Система работает без авторизации, что также повышает удобство использования.

Разработанная программа написана на языке Ruby с использованием фреймворка Ruby on Rails. Разработанное приложение реализует паттерн MVC, а также модель CRUD. Архитектурный стиль взаимодействия компонентов – REST.

					<i>ВКР 090301.01 17 81 01</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		51

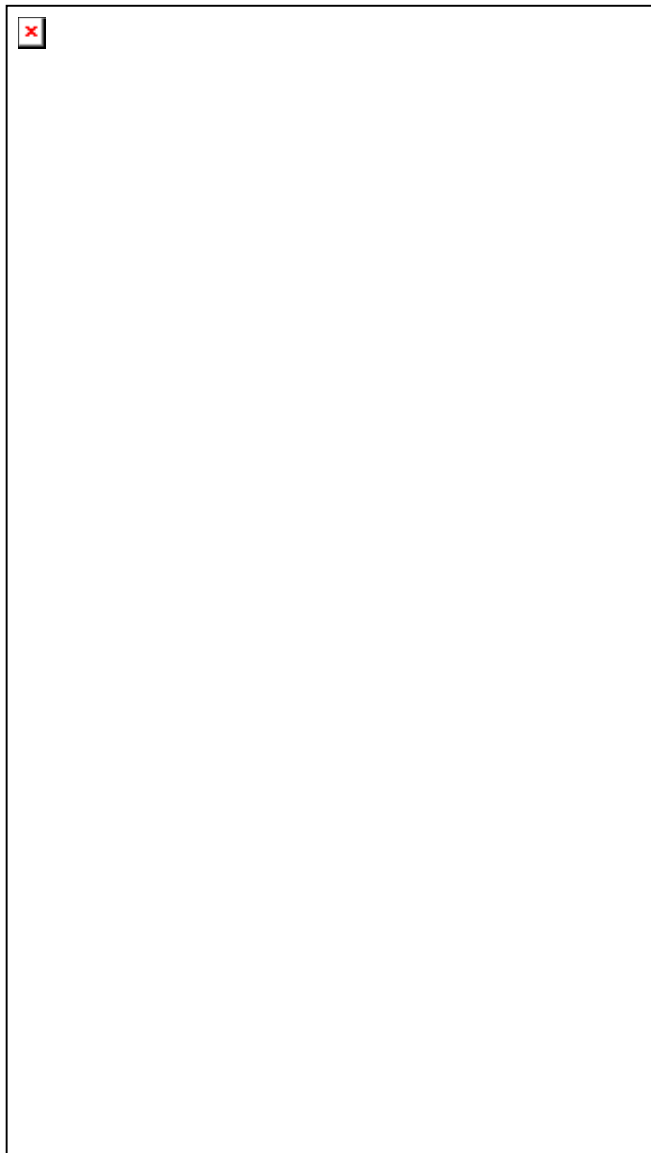


Рисунок 15 – Схема программы

Схема программы максимально проста. Упрощение было введено специально для создания максимально дружественного пользовательского интерфейса. Для поиска ВУЗа пользователю нужно только задать поисковой запрос и, если ВУЗ найден, пользователь имеет доступ ко всей информации о нем на сайте. Если в поисковой выдаче пользователь не нашел то, что ищет, ему нужно создать другой поисковой запрос и повторить попытку, пока искомый результат не будет найден.

5.2. Добавление ВУЗа

Для того, чтобы добавить учебное заведение в систему необходимо связаться с администрацией ресурса и предоставить информацию в любой удобной форме. После перепроверки данных администратор создает новый объект типа «University» в административной панели и заполняет описание. Затем необходимо добавить университету информацию о всех программах обучения с указанием стоимости. Для этого во вкладке «Relationships» необходимо установить отношение между университетом и программами обучения. Данное инженерное решение вызвано тем, что к каждому направлению также необходимо добавить описание стоимости обучения. После этого университет будет размещен на сервисе и пользователи получат доступ к информации о нем. На Рисунке 16 представлена диаграмма последовательности добавления информации о ВУЗе.

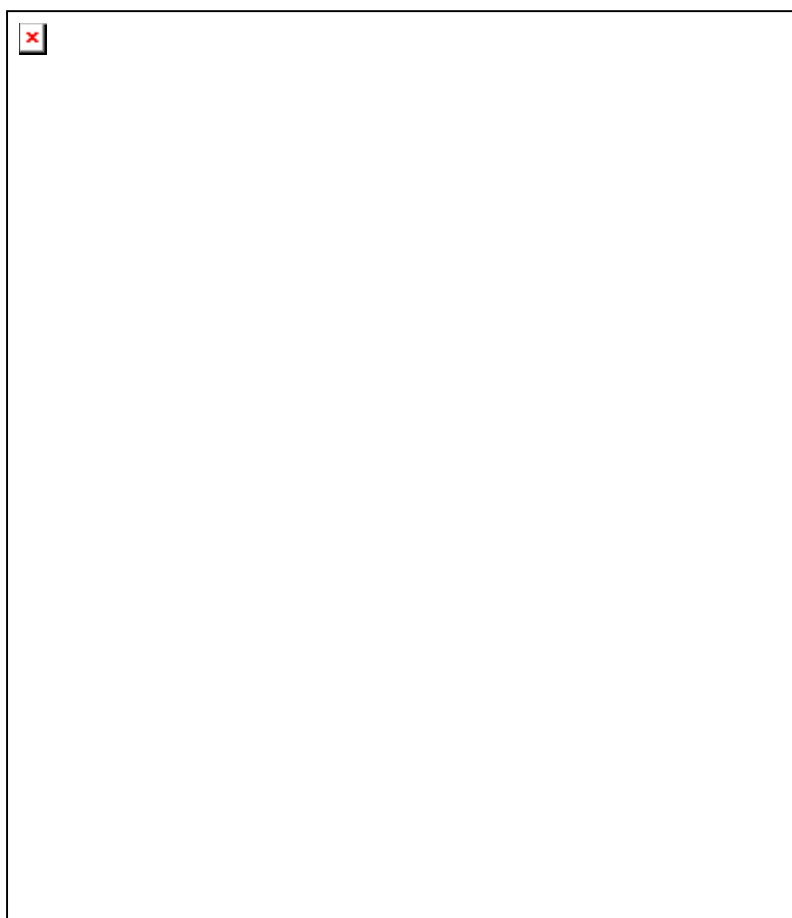


Рисунок 16 – Диаграмма последовательности

5.3. Добавление направления подготовки

Добавление направления подготовки аналогично добавлению ВУЗа, только изначально добавляется направление подготовки, а для нее уже создается отношение с университетом.

После добавления информации на сайт администратор должен удостовериться в корректности добавления данных. После это он должен уведомить представителей ВУЗа о том, что учебное заведение успешно добавлено на сайт. После чего они перепроверят данные и, в случае некорректности введенных данных укажут на ошибки для их исправления.

5.4. Основные ошибки и способы их устранения

В ходе работы программы возможно возникновение ошибок. С наибольшей вероятностью возможно появление ошибок с кодами: 400, 500, 403 и 502. Остальные ошибки крайне маловероятны. Далее рассмотрим каждый тип ошибки и способ исправления подробно.

5.4.1. Ошибка 404

Данная ошибка обозначает, что страница не найдена на сервере. Чаще всего это случается из-за того, что у пользователя неправильная ссылка на ресурс. Или ко времени перехода по ссылке ресурс был удален или перемещен. При возникновении данной ошибки пользователь увидит страницу, изображенную на рисунке 17.

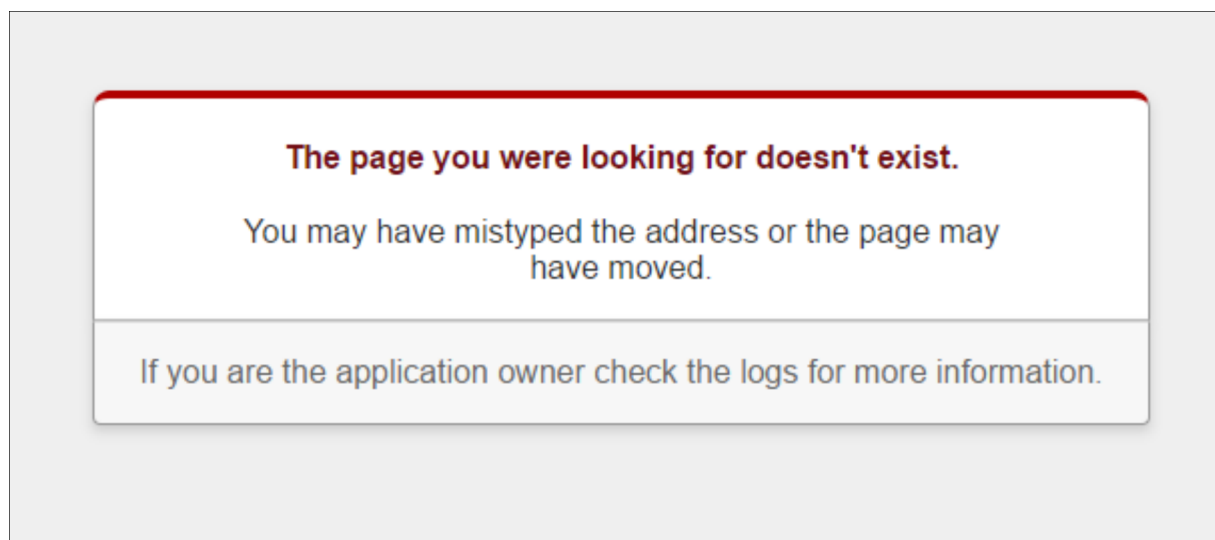


Рисунок 17 – Ошибка 404

В случае возникновения данной ошибки, необходимо проверить правильность написания ссылки, если это не помогло, то необходимо обратиться в службу поддержки.

5.4.2. Ошибка 502

На рисунке 15 представлено изображение ошибки 502. Данная ошибка означает, что Nginx не смог обратиться к приложению. Возникает из-за того, что процесс приложения остановлен или процесс не отвечает на запросы. Для решения данной проблемы необходимо повторно запустить приложение на сервере, либо проверить настройки Nginx, в частности, расположение файла с номером процесса приложения.

Чаще всего такая ошибка возникает в случае, когда сайт «лежит». Если до этого сайт был настроен корректно, то просто необходимо его перезапустить.

502 Bad Gateway

nginx/1.2.6

Рисунок 18 – Ошибка 502

5.4.3. Ошибка 500

Ошибка 500 обозначает ошибку на сервере. Данная ошибка самая распространенная и может возникать по различным причинам. В большинстве случаев она говорит о том, что программист сделал ошибку в приложении. При возникновении ошибки, появится страница, изображенная на рисунке 16.

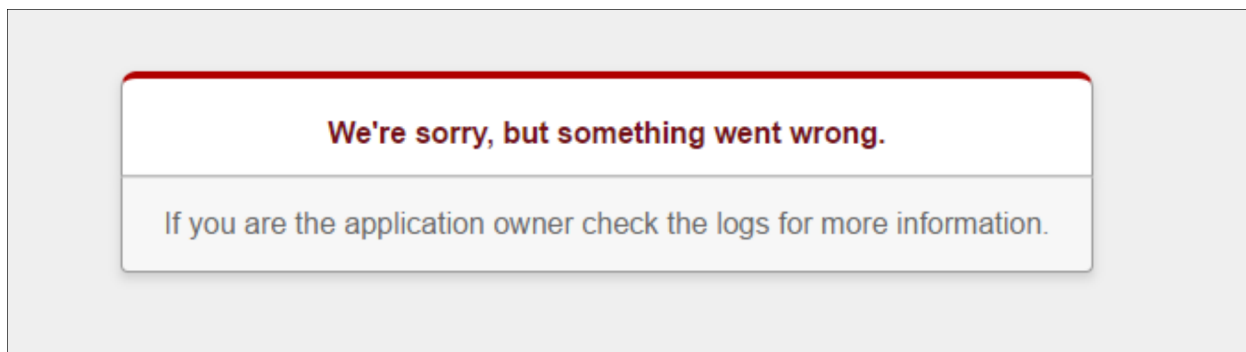


Рисунок 19 – Ошибка 500

Подробное описание причины возникновения ошибки, а также файл, метод и даже строчку можно найти в файле протокола на сервере. Для анализа работы сервера необходимо просматривать файл протокола. В нем можно найти все происходящие ошибки, все запросы, приходящие к приложению.

Данная ошибка требует вмешательства разработчика для устранения. Если проводить параллели с языками программирования, данная ошибка означает необработанное исключение времени исполнения.

6. ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта был разработан сервис, предназначенный для поиска высших учебных заведений, осуществляющих подготовку по определенным направлениям бакалавриата.

Система имеет административную панель, в которой администратор размещает информацию о высших учебных заведениях и направлениях подготовки по программам обучения бакалавриата.

В приложении реализован полнотекстовый поиск высших учебных заведений по ключевым словам, что делает ресурс максимально дружелюбным для пользователя.

Система работает без авторизации, что также повышает удобство использования.

Разработанная программа написана на языке Ruby с использованием фреймворка Ruby on Rails. Разработанное приложение реализует паттерн MVC, а также модель CRUD. Архитектурный стиль взаимодействия компонентов – REST.

В ходе выполнения дипломного проектирования были продемонстрированы навыки, полученные в ходе обучения в ПГУ. Такие как: умение программировать на объектно-ориентированном языке программирования, умение создавать web-приложения, умение проектировать базы-данных, умение тестировать программные продукты, умение писать документацию, соответствующую требованиям ЕСПД.

Работа выполнена в полном объеме в соответствии с ТЗ в установленные сроки.

7. СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Фултон Х., The Ruby Way. – М.: Питер, 2015 – 348 с.
2. Ruby on Rails по-русски [Электронный ресурс]. – Электрон.дан. – URL: <http://rusrails.ru/> (дата обращения 16.04.2017).
3. Ruby on Rails [Электронный ресурс]. – Электрон.дан. – URL: https://ru.wikipedia.org/wiki/Ruby_on_Rails (дата обращения 20.04.2017).
4. Бенедетти Р., Крэнли Р. Изучаем работу с jQuery. – СПб.: Питер, 2012. – 528 с.
5. Коннолли Т., Бегг К., Страчан А. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание.: Пер. с англ. – М.: Издательский дом "Вильямс", 2003. – 1440 с.
6. Д. Румянцев. Сам себе Web-программист. Практикум создания качественного Web-сайта. – М.: ИНФРА-М, 2001.
7. Хартл М, Ruby on Rails Tutorial. – М.: Питер, 2011 – 514 с.
8. Ruby on Rails API [Электронный ресурс]. – Электрон.дан. – URL: <http://api.rubyonrails.org> (дата обращения 2.04.2017).
9. Фримен Э., Робсон Э. Изучаем программирование на JavaScript. – СПб.: Питер, 2015. – 640 с.
10. Закас Н. JavaScript для профессиональных веб-разработчиков / [Пер. с англ. А. Лютича]. – СПб.: Питер, 2015. – 960 с.
11. PostgreSQL [Электронный ресурс]. – Электрон.дан. – URL: <https://ru.wikipedia.org/wiki/PostgreSQL> (дата обращения 20.04.2017).
12. Ruby [Электронный ресурс]. – Электрон.дан. – URL: <https://ru.wikipedia.org/wiki/Ruby> (дата обращения 20.04.2017).
13. Stat Edu [Электронный ресурс]. – Электрон.дан. – URL: <http://stat.edu.ru/stat/vis.shtml> (дата обращения 16.04.2017).

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А. СХЕМЫ

					<i>ВКР 090301.01 17 81 01</i>	<i>Лист</i>
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		60

Схема взаимодействия программы



					ВКР 090301.01 17 81 01	Лист
						61
Изм.	Лист	№ докум.	Подпись	Дата		

ПРИЛОЖЕНИЕ Б. ЛИСТИНГИ

					ВКР 090301.01 17 81 01	Лист
						62
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б1. Модели

Файл – user.rb

```
class User < ApplicationRecord
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable and :omniauthable
  devise :database_authenticatable,
         :recoverable, :rememberable, :trackable,
:validatable
end
```

Файл – study.rb

```
class Study < ApplicationRecord
  belongs_to :category
  has_many :relationship
  validates_presence_of :name, :code, :description,
:category
end
```

Файл – relationship.rb

```
class Relationship < ApplicationRecord
  belongs_to :university
  belongs_to :study
  validates_presence_of :university, :study
  validates_format_of :cost, :with =>
/\A\d+(?:\.\d{ 0,2} )?\z/

  def self.min_cost
    d = Relationship.all.minimum(:cost)
    d == nil || d == 0 ? 1 : d
  end

  def self.max_cost
    d = Relationship.all.maximum(:cost)
    d == nil ? 0 : d
  end

end
```

Файл – university.rb

```
require 'file_size_validator'
```

					ВКР 090301.01 17 81 01	Лист
						63
Изм.	Лист	№ докум.	Подпись	Дата		

```

class University < ApplicationRecord
  validates :avatar, file_size: { maximum: 4096.kilobytes }
  mount_uploader :avatar, AvatarUploader
  validates_presence_of :avatar, :description, :address,
:name, :state
  has_many :relationship
end

```

Файл – category.rb

```

class Category < ApplicationRecord
  validates_presence_of :name
  has_many :study
end

```

Приложение Б2. Контроллеры

Файл – application_controller.rb

```

class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception

end

```

Файл – relationships_controller.rb

```

class RelationshipsController < ApplicationController

  def index
    @max_cost = Relationship.max_cost + 1
    @min_cost = Relationship.min_cost - 1
    p = params[:index]
    #запрос к базе данных будет выполнен после выхода из
этого метода, сейчас формируем запрос
    @relationships = Relationship.all.page(params[:page])
    if p != nil
      last_cost = p[:last_cost]
      if p[:last_cost] == 0 || p[:last_cost] == nil
        last_cost = 2147483646#MAX_INT - 1
      end
      start_cost = p[:start_cost]
      if p[:start_cost] == 0 || p[:start_cost] == nil

```

					ВКР 090301.01 17 81 01	Лист
						64
Изм.	Лист	№ докум.	Подпись	Дата		

```

        start_cost = 0
      end
      unless p[:study].blank?
        str = '%' + p[:query] + '%'
        puts str
        @relationships = @universities.where(study:
p[:study] )
      end
      unless p[:university].blank?
        @relationships = @relationships.where(university:
p[:university] )
      end
      @relationships = @relationships.where(cost:
start_cost..last_cost)
    end

    @relationships = @relationships.page(params[:page] )
    respond_to do |format|
      format.js
      format.html
    end
  end

  def show
    @relationship = Relationship.find_by_id params[:id]
  end

end

```

Файл – studies_controller.rb

```

class StudiesController < ApplicationController

  def index
    p = params[:index]
    #запрос к базе данных будет выполнен после выхода из
этого метода, сейчас формируем запрос
    @studies = Study.all.page(params[:page] )
    if p != nil
      unless p[:query].blank?
        str = '%' + p[:query] + '%'
        puts str
        @studies = @studies.where('lower(name) LIKE ? OR
lower(description) LIKE ? OR lower(code) LIKE ?', str, str, str)

```

					BKP 090301.01 17 81 01	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

```

        end
        unless p[:university].blank?
          @studies = @studies.joins(:relationships)

        .where('relationships.university_id = ?', p[:university])
        end
      end
    end

    @studies = @studies.page(params[:page])
    respond_to do |format|
      format.js
      format.html
    end
  end
end

def show
  @study = Study.find_by_id params[:id]
end

end

```

Файл – universities_controller.rb

```

class UniversitiesController < ApplicationController

  def index
    p = params[:index]
    #запрос к базе данных будет выполнен после выхода из
    этого метода, сейчас формируем запрос
    @universities = University.all.page(params[:page])
    if p != nil
      unless p[:query].blank?
        str = '%' + p[:query] + '%'
        puts str
        @universities = @universities.where('lower(name)
LIKE ? OR lower(description) LIKE ?', str, str)
      end
      unless p[:state].blank?
        @universities = @universities.where(state: p[:state]
== '1')
      end
      unless p[:category].blank?

```

					BKP 090301.01 17 81 01	Лист
						66
Изм.	Лист	№ докум.	Подпись	Дата		

```

        @universities =
@universities.joins(:relationships).where('relationships.study_id = ?', p[:category])
        end
    end

    @universities = @universities.page(params[:page])
    respond_to do |format|
        format.js
        format.html
    end
end

def show
    @university = University.find_by_id params[:id]
end

end

```

Приложение Б3. База данных

```

ActiveRecord::Schema.define(version: 20170521065745) do

  create_table "categories", force: :cascade do |t|
    t.string "name", null: false
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
  end

  create_table "ckeditor_assets", force: :cascade do |t|
    t.string "data_file_name", null: false
    t.string "data_content_type"
    t.integer "data_file_size"
    t.string "type", limit: 30
    t.integer "width"
    t.integer "height"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["type"], name: "index_ckeditor_assets_on_type"
  end

  create_table "relationships", force: :cascade do |t|
    t.integer "university_id",
    null: false
  end

```

					ВКР 090301.01 17 81 01	Лист
						67
Изм.	Лист	№ докум.	Подпись	Дата		

```

        t.integer "study_id",
null: false
        t.decimal "cost", precision: 8, scale: 2,
default: "0.0", null: false
        t.datetime "created_at",
null: false
        t.datetime "updated_at",
null: false
        t.index ["study_id"], name:
"index_relationships_on_study_id"
        t.index ["university_id", "study_id"], name:
"index_relationships_on_university_id_and_study_id", unique:
true
        t.index ["university_id"], name:
"index_relationships_on_university_id"
    end

    create_table "studies", force: :cascade do |t|
        t.integer "category_id", null: false
        t.string "name", null: false
        t.string "code", null: false
        t.string "description", null: false
        t.datetime "created_at", null: false
        t.datetime "updated_at", null: false
        t.index ["category_id"], name:
"index_studies_on_category_id"
    end

    create_table "universities", force: :cascade do |t|
        t.string "avatar", null: false
        t.string "description", null: false
        t.string "address", null: false
        t.string "name", null: false
        t.boolean "state", default: true, null: false
        t.datetime "created_at", null: false
        t.datetime "updated_at", null: false
    end

    create_table "users", force: :cascade do |t|
        t.string "email", default: "", null:
false
        t.string "encrypted_password", default: "", null:
false
        t.string "reset_password_token"
        t.datetime "reset_password_sent_at"
    end

```

					BKP 090301.01 17 81 01	Лист
						68
Изм.	Лист	№ докум.	Подпись	Дата		

```

        t.datetime "remember_created_at"
        t.integer "sign_in_count",          default: 0,  null:
false
        t.datetime "current_sign_in_at"
        t.datetime "last_sign_in_at"
        t.string   "current_sign_in_ip"
        t.string   "last_sign_in_ip"
        t.string   "name"
        t.datetime "created_at",           null:
false
        t.datetime "updated_at",           null:
false
        t.index ["email"], name: "index_users_on_email", unique:
true
        t.index ["reset_password_token"], name:
"index_users_on_reset_password_token", unique: true
      end

    end
  end
end

```

Приложение Б4. Панель администратора

Файл – user.rb

```

ActiveAdmin.register User do
  menu label: 'Администраторы'
  permit_params :email, :password, :password_confirmation

  index do
    selectable_column
    id_column
    column :email
    column :current_sign_in_at
    column :sign_in_count
    column :created_at
    actions
  end

  filter :email
  filter :current_sign_in_at
  filter :sign_in_count
  filter :created_at

  form do |f|

```

					ВКР 090301.01 17 81 01	Лист
						69
Изм.	Лист	№ докум.	Подпись	Дата		

```

    f.inputs 'Admin Details' do
      f.input :email
      f.input :password
      f.input :password_confirmation
    end
    f.actions
  end
end

```

end

Файл – study.rb

```

ActiveAdmin.register Study do
  menu label: 'Направления подготовки'
  permit_params :name, :code, :description, :category_id

```

```

  index do
    selectable_column
    column 'Название', :name
    column 'Код', :code
    column 'Категория', :category
    column :created_at
    actions defaults: true
  end

```

```

  form do |f|
    cat = Category.all.map { |cat| [cat.name, cat.id] }
    f.inputs 'Направление подготовки' do
      f.input :name, label: 'Имя'
      f.input :code, label: 'Код образовательной программы'
      f.input :category_id, label: 'Категория', as: :select,
include_blank: false, collection: cat
      f.input :description, as: :ckeditor, label: 'Описание'
    end
    f.actions
  end
end
end

```

Файл – relationship.rb

```

ActiveAdmin.register Relationship do
  menu label: 'Отношение ВУЗ-направление подготовки'
  permit_params :university_id, :study_id, :cost

```

```

  index do
    selectable_column
    column 'ВУЗ', :university

```

					ВКР 090301.01 17 81 01	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

```

        column 'Направление', :study
        column 'Цена', :cost
        column :created_at
        actions defaults: true
    end

    form do |f|
        u = University.all.map { |cat| [cat.name, cat.id]}
        s = Study.all.map { |cat| [cat.name, cat.id]}
        f.inputs 'Направление подготовки' do
            f.input :study_id, label: 'Направление подготовки',
as: :select, include_blank: false, collection: s
            f.input :university_id, label: 'Университет', as:
:select, include_blank: false, collection: u
            f.input :cost, label: 'Стоимость обучения'
        end
        f.actions
    end

end

end

```

Файл – university.rb

```

ActiveAdmin.register University do
  menu label: 'ВУЗы'
  permit_params :avatar, :description, :address, :name,
:state

  index do
    selectable_column
    column 'Аватар', :avatar do |u|
      image_tag u.avatar.small_avatar.url, height: 100
    end
    column 'Название', :name
    column 'Адрес', :address
    column 'Государственный', :state
    column :created_at
    actions defaults: true
  end

  form do |f|
    f.inputs 'ВУЗ' do
      f.input :name, label: 'Имя'
      f.input :avatar, as: :file, label:
'Изображение (аватар) '

```

					ВКР 090301.01 17 81 01	Лист
						71
Изм.	Лист	№ докум.	Подпись	Дата		

```

        f.input :state, label: 'Государственный'
        f.input :address, label: 'Адресс'
        f.input :description, as: :ckeditor, label: 'Описание'
      end
      f.actions
    end
  end
end

```

end

Файл – category.rb

```

ActiveAdmin.register Category do
  menu label: 'Категории направлений подготовки'
  permit_params :name
end

```

Приложение Б5. Представление

Файл –application.html.slim

```

doctype html
html
  head
    meta charset="utf-8"
    meta http-equiv="X-UA-Compatible" content="IE=edge"
    meta name="viewport" content="width=device-width,
initial-scale=1.0"
    title= content_for?(:title) ? yield(:title) : 'Найти
ВУЗ'
    meta name="description"
content="#{ content_for?(:description) ? yield(:description) :
'информационная система поиска высших учебных заведений,
осуществляющих подготовку по определенным направлениям
бакалавриата'} "
    = csrf_meta_tags
    = stylesheet_link_tag 'font-awesome'
    = stylesheet_link_tag 'application', media: 'all',
'data-turbolinks-track' => 'reload'
    = javascript_include_tag 'application', 'data-
turbolinks-track' => 'reload'
    = favicon_link_tag asset_path('favicon.png'), rel:
'shortcut icon'
  body
    #flash

```

					ВКР 090301.01 17 81 01	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

```

    = bootstrap_flash
  =render 'layouts/header'
main
  = yield
  =render 'layouts/footer'
javascript:
  $(document).ready(function () {
    // девайс сообщение выдает с дочерними элементами
    $alert = $('.alert');
    var text = $alert.first().text();
    $alert.remove();
    console.log(text);
    if (jQuery.trim(text).length > 0) {
      Materialize.toast(text, 5000);
    }
  });

```

Файл – _header.html.slim

```

-landing = params[:controller] == 'static_pages' &&
params[:action] == 'landing'
.navbar-fixed class="#{ 'navbar-landing' if landing} "
  nav
    .nav-wrapper
      ul.main_actions.hide-on-med-and-down.right
        li=link_to 'ВУЗы', universities_path, data: {
turbolinks: false }
        li=link_to 'Направления подготовки', studies_path,
data: { turbolinks: false }
        li=link_to 'Предложения ВУЗов', relationships_path,
data: { turbolinks: false }
        a.button-collapse href="#" data-activates="side-menu"
          i.fa.fa-bars aria-hidden="true"
        .side-nav#side-menu.collection
          =link_to 'ВУЗы', universities_path, data: {
turbolinks: false }, class: 'collection-item'
          =link_to 'Направления подготовки', studies_path,
data: { turbolinks: false }, class: 'collection-item'
          =link_to 'Предложения ВУЗов', relationships_path,
data: { turbolinks: false }, class: 'collection-item'
      javascript:
        $(document).ready(function () {
          $(".button-collapse").sideNav();
          $("#new_signer").validate({
            rules: {

```

```

        signer_email: {
            required: true
        }
    },
    errorElement: 'div',
    errorPlacement: function (error, element) {
        var placement = $(element).data('error');
        $(element).addClass('invalid');
        if (placement) {
            $(placement).append(error)
        } else {
            error.insertAfter(element);
        }
    }
});
});

```

Файл – _footer.html.slim

```

footer.page-footer
  .container
    p © Руслан Абдряязков, 2017

```

Файл –index.html.slim

```

=javascript_include_tag 'nouislider'
=stylesheet_link_tag 'nouislider'
#main_content.tour_index
  #trapeze
    .text ВУЗы
  .items.row
    .about
      .col.l3.m12
        #form=render 'form'
      #orders.col.l9.m12
        #tours_list=render @universities
        #paginator= paginate @universities, remote: true

```

Файл – show.html.slim

```

-title @university.name
#main_content.tour_show
  #trapeze
    .text=@university.name
  .items.row.about_container

```

					ВКР 090301.01 17 81 01	Лист
						74
Изм.	Лист	№ докум.	Подпись	Дата		

```

        .about
            =image_tag @university.avatar.small_avatar, class:
'office'
            .btn_transparent=link_to 'Специальности',
relationships_path(index: { iniversity_id: @university.id}),
class: 'btn'
            .text
                p Адрес: #{ @university.address}
                p Статус: #{ @university.state ? 'Государственный' :
'Частный'}
            =@university.description.html_safe

```

					ВКР 090301.01 17 81 01	Лист
						75
Изм.	Лист	№ докум.	Подпись	Дата		