МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**
**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ МАШИНОСТРОИТЕЛЬНЫЙ УНИВЕРСИТЕТ (МАМИ)**

# Выпускная квалификационная работа бакалавра

по направлению 230100 «Информационные системы и технологии»

по дисциплине «Проектирование и разработка»

на тему «Разработка *т*емпоральной информационной системы учета контингента студентов»

Группа 121131

Студент _____ А.С. Умурзакова
Руководитель _____ Радыгин. В.Ю.

ДОПУСКАЕТСЯ К ЗАЩИТЕ

Зав. каф. 36, доцент, к.ф.-м.н. _____ /Белова И.М./

МОСКВА 2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**
**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ МАШИНОСТРОИТЕЛЬНЫЙ УНИВЕРСИТЕТ (МАМИ)**

**Кафедра информационных технологий**

Зав. Кафедрой 36
_____/Белова И.М. /
«_____»_____20__г.

## Задание

на выпускную работу по направлению «Информационные системы и технологии»

1. Тема работы «Разработка темпоральной информационной системы учета контингента студентов»
2. Сроки начала работы
3. Руководитель дипломной работы:  Радыгин Виктор Юрьевич
4. Задание дипломной работы:
    1. Исходные данные.
    2. Цель работы — разработка темпоральной информационной системы учета контингента студентов.
    3. Содержание дипломной работы:
        1. Введение.
        2. Литературный обзор.
        3. Структура системы.
        4. Пользовательский интерфейс.
    4. Используемые технологии: PostgreSQL, JavaScript, JQuery, HTML5, CSS3, Ruby, Ruby on Rails.
    5. Практическая реализация: данная работа может найти применение в различных учебных заведениях».
    6. Графическая часть:
        1. Диаграмма использования.
        2. Диаграмма классов.
        3. Реализация интерфейса.

Руководитель _____ (  )

Дипломник  _____ (  )

**Аннотация**

Работа посвящена созданию системы для темпоральной информационной системы учета контенгента студентов  реализованной на фреймворке Ruby on Rails.

Требуется реализовать следующие подзадачи:

- Web-интерфейс предоставляющий информацию за период времени;
- Разделенная ролевая система;
- Система поиска студентов по различным параметрам с учетом темпоральности;
- Система введения истории изменений студентов (смена паспорта, группы и т.д.) ;
- Гибкая ролевая система управления доступом.

WEB приложение должно обладать упрощенной навигацией состояний студента за различные периоды обучения, а так же систе.

Работа содержит ___  страниц, __  иллюстраций, __ приложений.

**Ключевые слова: Темпаральная система учета студентов,  Ruby, Ruby on Rails, HAML, UML, PostgreSQL.**

## Содержание

# Введение

Мы живем в интенсивно развивающимся мире в котором информация является неотъемлемой частью жизни человека. Количество данных колоссально. Оно постоянно растет и нуждается в обработке. Сейчас большая часть персональных данных людей храниться в цифровом виде. Такие структуры как ЖКХ, банковские отделения  и многие другие тратят колоссальные средства для хранения, обработки и зашиты данного типа информации. Их системы позволяют отслеживать такие вещи как кредитная история, операции с ценными операциями и многое другое. Системы позволяющие отслеживать историю развития объектов называются темпоральными. Но как быть менее обеспеченным структурам, которым необходимо отслеживать изменения персональных данных? Одной из таких структур является московский государственный машиностроительный университет. Одной из потребностей данной структуры является возможность просмотра изменения данных за определенные периоды. Это могут быть как оценки, персональные данные, например изменения паспорта, так и возможность просмотра времени смены групп студента.

Именно реализация данной системы входит в цели дипломной работы. Она должна не только соответствовать поставленным требованиям функционала, но и быть многопользовательской и удобной в использовании.

Платформой для реализации поставленной задачи был выбран выбран фреймворк Ruby on Rails.

Фреймворк Ruby on Rails активно используется для создания крупных приложений, где требуется быстрая и удобная разработка. Данная технология базируется на языке высокого уровня Ruby.

Для работы с Ruby on Rails необходимы навыки работы с CSS, JavaScript, понимание принципов использования AJAX, и конечно же умение работы на объектно — ориентированном языке программировании Ruby.

Ruby on Rails базируется на MVC (model-view-controller) - схема использования нескольких паттернов для разделения приложения на три отдельных сущности, для того что уменьшить связь между компонентами. Это позволяет при изменении одного из компонента уменьшить воздействие на другие.

В связи с разработкой на данной технологии использовался редактор — RubyMine от JetBrains. Он позволяет умелому разработчику быстро переключаться между файлами крупного проекта, обладает автодополнением, что ускоряет разработку, подсветку синтаксиса, удобное переключением между вызовом функции и ее реализации по сигналу с горячих клавиш. Данный IDE обладает многими положительными моментами, которые так необходимы для разработки.

После того как были разобраны средства разработки, пришло время ознакомиться с самим списком и целям задач.

# 1. Литературный обзор
## *1.* Постановка задачи

Необходимо реализовать темпоральную систему учета студентов. Данная система предназначена для централизации данных об студентах, а так же возможности просмотра их истории изменений, состояний в прошлом. Так же должна присутствовать система поиска студентов, для быстрой навигации среди данных, быть гибкой в сопровождении и открытой для изменений. Данная система должна представлять собой Web-ориентированное приложение с WEB интерфейсом предоставляющий возможность просмотра истории, добавления студентов, редактированием их параметров через браузер в интерактивном режиме. Так же должна быть реализована гибкая ролевая система. Более подробно:

**Темпоральность** (от англ. *tempora*– временные особенности) – временная сущность явлений, порожденная динамикой их особенного движения, в отличие от тех временных характеристик, которые определяются отношением движения данного явления к историческим.

**LTL**(логика линейного времени) – позиционирование логики линейного времени с конечным прошлым и бесконечным будущем (время изоморфно). Модели данной логики представляют собой последовательности состояний на временной шкале.

**ГРС**(гибкая ролевая система) позволяет задавать пользователю системы набор ролей, каждая которая содержит набор правил. Для данной системы это подразумевает возможность просмотра различной информации, различным отображением и правами доступа.

Так же уточним значение такого понятия как „Дата актуализации“:

**Дата актуализации** - Действие по значению гл. актуализировать (актуализовать); перевод чего-либо из состояния потенциального, не соответствующего современным условиям, в состояние реальное, актуальное, соответствующее современным условиям; превращение чего-либо в нечто важное, насущное, актуальное. В данной работе это подразумевает запрос (на просмотр)состояния объекта в прошлом.

На данном этапе изложены значения ключевых слов. Далее в работе по мере возникновения новых понятий им будут даваться определения.

## 1.2 Обзор существующих решений

Существуют различные темпоральные системыс возможностью отслеживания изменений данных о человеке, но не подготовленных под работу с особенностями различных ВВУЗ-ов, а те что созданы внутри самих учреждений, зависимы от внутреннего устройства самого учебного учреждения либо являются коммерчискими продуктами. К примеру:

**БИТ.ВУЗ** - оригинальная конфигурация на платформе "1С:Предприятие 8.2". Продукты, входящие в линейку "БИТ.ВУЗ" (БИТ.ВУЗ.Приемная комиссия, БИТ.ВУЗ. Учебная часть, БИТ.ВУЗ. Подготовительные курсы, БИТ.ВУЗ.Учет нагрузки преподавателей), являются подключаемыми модулями единой системы. Данная система не может использоваться повсеместно так как часть университетов работает под операцинными системами семейства Unix и 1С не поддерживается. Причем важно отметить, что 1С коммерческий продукт с высокой обслуживаемой стоимостью.

**Система «Студен»** - продукт от компании Тауруна, содержит в себе такие возможности как практичный набор инструментов для ведения базы данных учащихся и учета их динамики, мониторинга успеваемости в семестре и результатов экзаменационной сессии. Первичные данные могут использоваться для создания комбинированных отчетов по сложным условиям, в том числе — для подготовки сводных ведомостей по итогам года и вкладышей к дипломам. Содержит в себе большое количество различных возможностей, но реализована под операционные системы семейства Windows.

**Bitcop Security** - Современная и удобная система учета рабочего времени. Предназначена для мониторинга работы офисных сотрудников в компаниях среднего и крупного бизнеса с развитой филиальной сетью. Система позволяет добиться существенного повышения эффективности работы персонала всех уровней. Внутри этого продукта нет ничего связанного со студентами или ВВУЗ-ами, но этот пример приведен за как один из основоположников разработки дизайна данной работы.

# 2. Проектирование системы

## 2.1 Общие сведения

Оценив недостатки существующих систем, было принято решение о создании новой системы основными преимуществами которой должны стать:

- Централизованное хранение данных;
- Поиск студентов, групп с учетом даты актуализации;
- Просмотр списка истории изменений;
- Проста в освоении;
- Гибкая ролевая система.

Использование Web-интерфейса обеспечит кроссплатформенность использование данного приложения, который позволит работать пользователям с данными в интерактивном режиме.

### 2.1.1 Теория

Дадим более подробное определение темпоряльности.

Темпоральность (от англ. *tempora* – временные особенности) – временная сущность явлений, порожденная динамикой их особенного движения, в отличие от тех временных характеристик, которые определяются отношением движения данного явления к историческим, астрономическим, биологическим, физическим и другим временным координатам. В современной философской культуре понятие темпоральности вошло через экзистенциалистскую традицию, в которой темпоральность человеческого бытия противопоставляется вещи, отчужденному, бескачественному, навязчивому, подавляющему времени. В феноменологически ориентированной социологии, а также в психологии и культурологии понятие темпоральности широко используется для описания таких динамических объектов, как личность, социальная группа, класс, общество, ценность. Идея анализа взаимодвижущихся социальных явлений через сопоставление их темпоральности легла в основу методологии темпорального анализа.

Соответственно под темпоральной системой подразумевается система в которой наблюдаемый объект с течением времени изменяется.

В разработанной системе эти изменеия можно не только наблюдать, но и просматривать состояние объекта, в нашем случае – студента, в прошлом. Т.е. с навигацией состояний по времени.

## *2.2* Требования к системе

Перед началом проектирования и реализации были уточнены и предъявлены следующие требования:

- Централизованное хранение и обработка данных;
- Гибкая система разделения ролей с возможностью авторизации;
- Информация должна представляться в наглядном виде;
- Интуитивно-понятный интерфейс;
- Возможность просмотра истории изменений студента;
- Обработка запросов на состояние объекта по дате актуализации;

# 3. Обзор используемых технологий

## *3.1* Ruby on Rails

**Ruby on Rails** - фреймворк, написанный на языке программирования Ruby. Ruby on Rails предоставляет архитектурный образец Model-View-Controller (модель-представление-контроллер) для веб-приложений, а также обеспечивает их интеграцию с веб-сервером и сервером базы данных. Данный фреймворк содержит в себе достаточно различных пакетов, которые нужны каждому разработчику для создания нового проекта, что позволяет при меньших усилиях достигать решения поставленной задачи в короткие сроки.

Некоторые принцыпы **Ruby on Rails**:

- **MVC** (Model-View-Controller);
- **REST** - шаблон для веб приложений;
- **DRY** – “Don’t Repeat Yourself” принцып разработки нацеленный на снижение повторения информации различного рода;
- **Convention Over Configuration** - используются соглашения по конфигурации, типичные для большинства приложений;

## *3.2 JAVASCRIPT*

**Javascript** – это интерпритируемый язык программирования, с помощью которого веб-страницам придается интерактивность. С его помощью создаются приложения, которые включаются в **HTML**-код (например, анимированые эффекты, анкеты или формы регистрации).

Сценарии **JavaScript** выполняются на компьютере пользователя и поэтому представляют некоторую несанкционированным опасность, доступом к связанную с возможным конфиденциальной информации.

Например, при соответствующих настройках браузеры способны разрешать сценариям считывать файлы, в которых могут содержаться важные данные, такие как пароли доступа. Поэтому в браузерах предусмотрена возможность отключения выполнения сценариев **JavaScript**. Это следует учитывать при разработке web-страницы с использованием **JavaScript**.

Если предполагается использовать один и тот же сценарий на многих web-страницах, удобно разместить его в отдельном файле и затем сослаться на этот файл. Это целесообразно сделать даже в том случае, если код будет использован только на одной странице. Например, если сценарий слишком большой и громоздкий, то выделение его в отдельный файл облегчает восприятие и отладку кода web-страницы.

## *3.3 Jquery*

**JQuery** - библиотекria JavaScript , фокусирующаяся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими.

## 3.4 PostgreSQL

**PostgreSQL** – унифицированный сервер баз данных, имеющий единый движок – storage engine. Постгрес использует клиент-серверную модель.Разработанная в университете Беркли.

## 3.5 UML

**UML** ( англ Unified Modeling Language - Унифицированный язык моделирования ) - язык графического описания для объектного моделирования в области разработки программного обеспечения . UML является языком широкого профиля , это - открытый стандарт , использующий графические

обозначения для создания абстрактной модели системы , называемой UML - моделью . UML был создан для определения , визуализации , проектирования и документирования , в основном , программных систем . UML не является языком программирования , но на основании UML - моделей возможна генерация кода .

# 4. Ход работы

## *4.1* Реализация

Первоначально необходимо представить как будет выглядеть конечный продукт. Что в нем должно присутствовать, и как с ним будут взаимодействовать пользователи.

Так как система подразумевает возможность наблюдения развития студента, далее объект, с течением времени, то необходимо сделать гибкую систему, которая позволит прослеживать изменение различных атрибутов, а так же позволять пользователю переключаться между состояниями объекта по временной.

В данной дипломной работе объектом для наблюдения служат студенты ФГОУ ВПО «МГИУ» (МАМИ). Для начала рассмотрим какие атрибуты могут меняться со временем:

1. Адрес (фактический, регистрация, прописка);
2. Фотография (аватар) студента;
3. Периоды обучения;
4. Группа в которой студент учится;
5. Подразделение;
6. Направление;
7. Специальность;
8. Паспортные данные.

Система должна иметь возможность поиска как по новым данным, так и возможность поиска студента по старым данным, в зависимости от даты актуализации. Возможность редактирования, просмотр данных о студенте в прошлом. В соответствии с требованиями была предложена следующая архитектура базы данных и связи объектов (рис. 1):

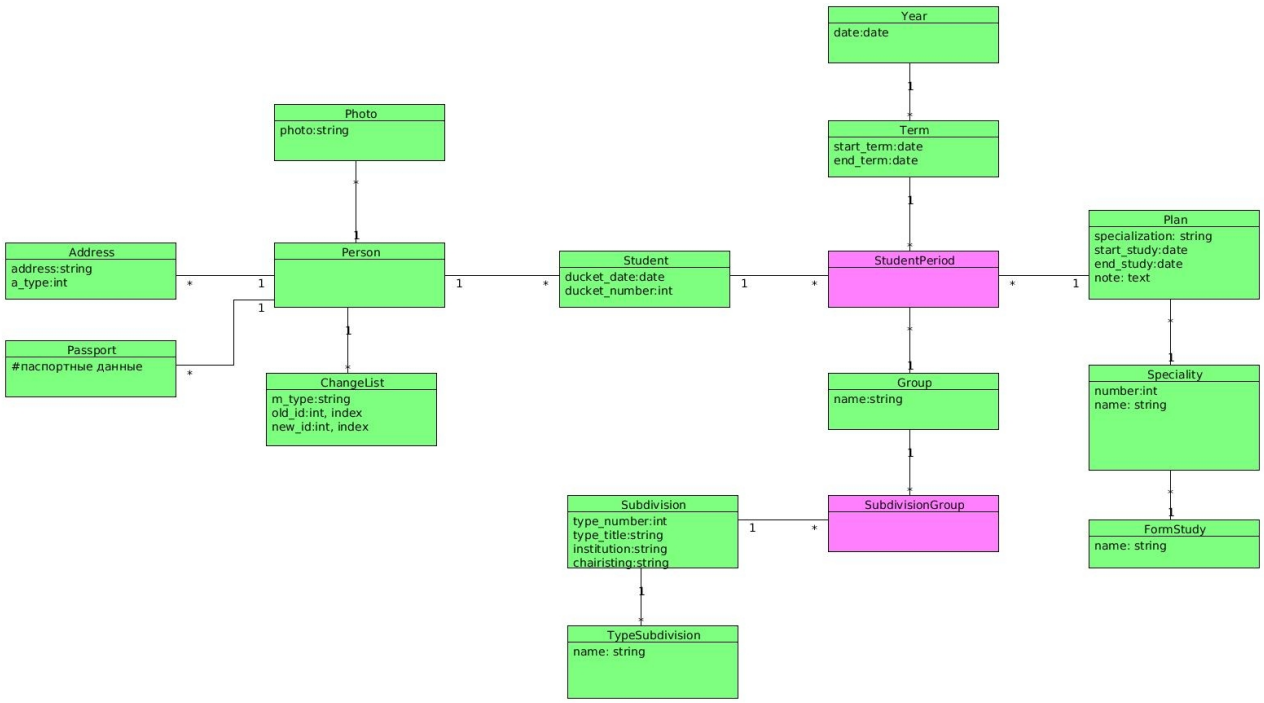



*Рис. 1 Диаграмма*

Необходимо подметит, что в системе хранятся в основном две категории информации, которые описывают студентов в системе:

1. Данные, описывающие студента как человека — паспортные данные;
2. Данные, описывающие студента непосредственно как студента.

Соответственно в системе имеются два «опорных» объекта: Student, Person, которые в совокупности делают то, что описано выше. Каждая из этих моделей отвечает за взаимосвязность информации по категориям.

Далее рассмотрим предложенную модель ближе:

**Person** — одна из основных моделей системы, которая объединяет все атрибуты описывающая человека, как личность. Исходя из того что студент есть человек, архитектура подразумевает два важных объекта таких как персона и студент.

**Photo** — хранит фотографии (аватар) персоны, для упрощенного выбора студента при поиске.

**Address** — объект, хранящий адрес человека, может быть трех типов: пропиской, регистрацией, а так же местом фактического проживания. Важно заметить, что регистрация и фактическое место проживания могут изменяться относительно часто по сравнению с пропиской, потому было решено убрать прямую зависимость между паспортом и адресом.

**Passport** — Модель, хранящая всю информацию паспортных данных такую как: пол, дата рождения, серию, номер, дата выдачи  и другое.

**ChangeList** — Список изменений произведенных с данной персоной. В ней хранится информация что и когда было изменено. Далее благодаря этой таблице предоставляется возможность выбора состояния объекта в прошлом.

**Student** — Модель данных, которая хранит неизменный номер студенческого, потому вести историю его изменений не требуется, но и так же связывает атрибуты характеризующие студента такие как: год поступления, кафедра, группа, подразделение, план обучения, специализация.

**Group —** группа, в которой обучается студент.

**Subdivision —** подразделения группы.

**Plan —** план обучения.

**Specialization —** Специальность. Одна специальность  может содержать различное количество планов работы, обучения.

**Year** — учебный год. Содержит даты начала и окончания обучения. К примеру 2015/2016 год.

**Term** — учебный период, часть учебного года. Так же содержит даты начала и окончания. Причем необходимо заметить, что значение хранится в виде даты, а для пользователя данная информация отображается в удобном виде типа «весна/осень».

Исходя из определения темпоральной системы, основная задача при реализации - это возможность изменения объекта со временем без потери его истории развития для реализации сервиса навигации состояния объекта в определенные промежутки времени. Соответственно данное утверждение обязывает нас не удалять при изменении атрибутов объекта его предыдущих значений, а также реализовать доступ к этим данным по дате актуализации. Для этого будет создан **ListChange**, который хранит в себе данные о том, где хранятся предыдущие записи значений аттрибутов.

После того как принято предложенное решение, перейдем к созданию самой системы. Как было уже сказано, разработка будет производиться на платформе Ruby on Rails. Для этого необходимо обладать навыками программирования на языках программирования Ruby, JavaScript, CoffeScript, а так же владеть языком разметки гипертекста HTML, и уметь создавать каскадные таблицы стилей CSS. Так же при реализации дипломной работы была использована библиотека адаптивной верстки Bootstrap 3, что позволило создать адаптированное приложение под различные форматы экранов, а так же браузеров.

Для решения данной задачи необходимо было определиться с входными данными и конечным результатом. Как было вышесказанно, входные данные — это информация о студентах, соответственно необходимо представить как пользователь будет работать с конечным продуктом. Каким образом необходимо ему получать данные. Так как в системе подразумевается наличие гибкой ролевой системы, было решено создать возможность отображения информации в разных форматах для отображения пользователям с разными уровнями привилегий, для облегченного восприятия информацией. К примеру,

возможность создать роль администратора системы с привилегией удаления/редактирования и создания, пользователей с доступным ему меню навигацией прямо из интерактивного режима взаимодействия с системой. Так же возможно создать роль секретаря, у которого не только не будет возможности влиять на данные о пользователях на уровне контроллеров, но и так же не будет отображаться сама панель доступа. Также возможно создать пользователя с несколькими ролями, допустим тем же «администратором» и «секретаря», с возможностью переключения режимов работы с системой. Тогда мы получим пользователя в системе, обладающим выбором среди множества привилегий.

Примерное взаимодействие с приложением отображено на рисунке 2.

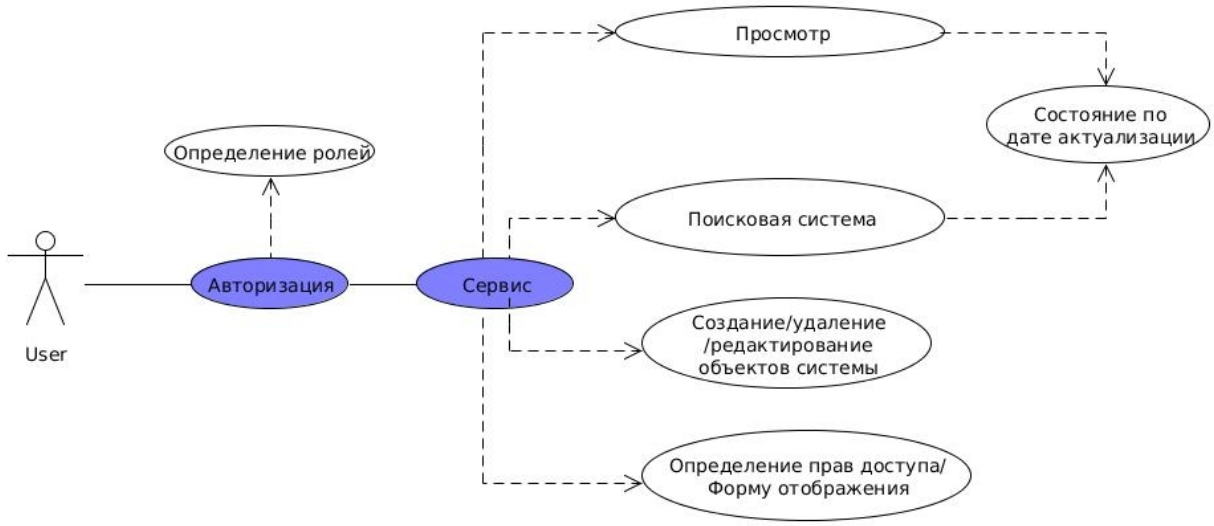



Рис. 2 Диаграмма использования

Пользователь сможет производит поиск, а так же просматривать состояния студентов по дате актуализации.

# 5. Описание пользовательских интерфейсов

В процессе работы была создана темпоральная система учета контингента студентов. В ее возможности входит поиск по дате актуализации, просмотр истории изменений, просмотр состояния студента в различные моменты времени, а так же введена в работу гибкая ролевая система.

## 5.1 Группы, подразделения

Согласно диаграмме классов (рис. 1) студент находиться в группе, которая находиться в подразделении. Для создания студента, необходимо оздать соответствующие записи. Далее показан интерфейс для данного действия (рис. 3).

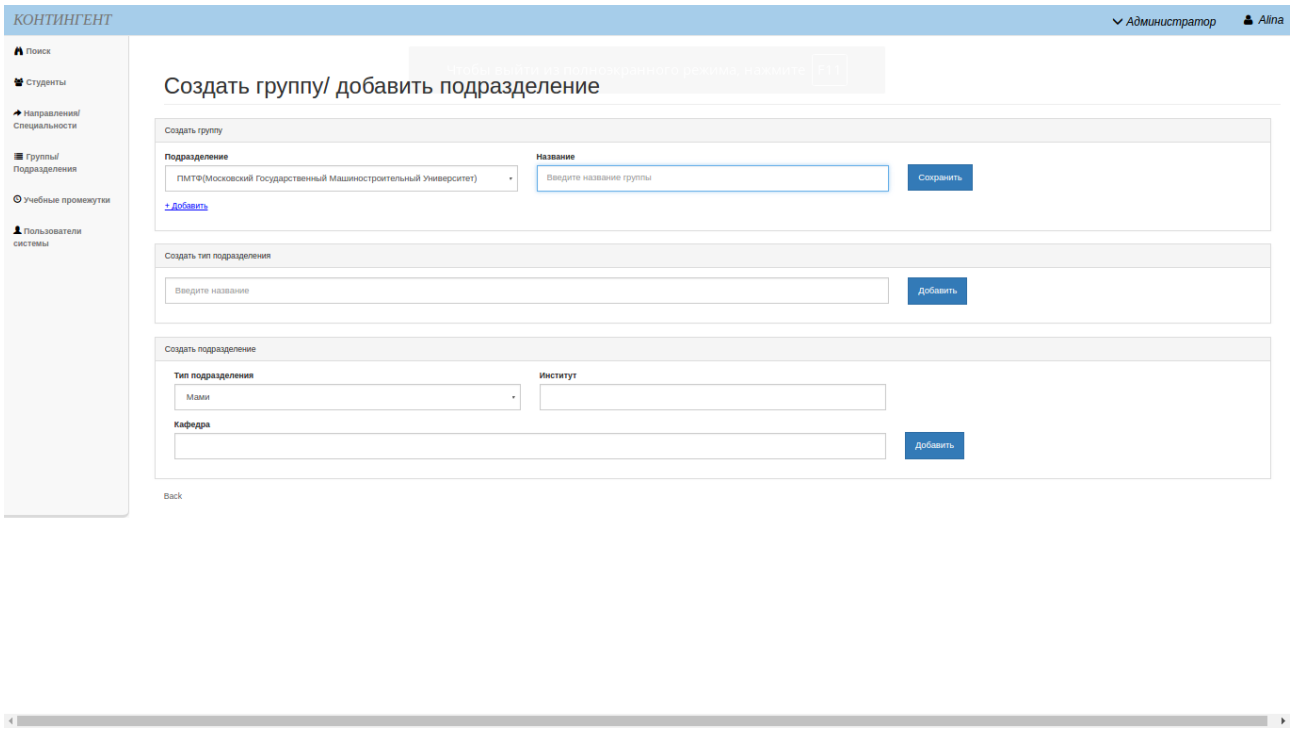



Рисунок 3.

Данный интерфейс позволяет динамически, без перезагрузски страницы создавать типы подразделений, подразделение, а так же группу в которой обучаются студенты. Динамика достигается использованием **JavaScript** и

технологией **Ajax**. Благодаря этой технологии на стороне клиента считываются данные с полей, формируется POST запрос и отправляется на сервер, где он, запрос, проходит обработку и в случает успеха создает запись, о чем уведомляет сторону клиента. Клиент в свою же очередь динамически подгружает новый контент.

**POST** — один из типов запросов, поддерживаемых HTTP протоколов, предназначенный для общения с сервером при необходимости обработки данных.  Он часто применяется для обработки загружаемых данных, а так же заполеных форм.

## 5.2 Специальности, план обучения

Согласно архитектуре, студент имеет специальность, а так же направление, в котором заложен план обучения по специальности. Необходимо заметить, что одна и та же специальность может иметь множество планов обучения. Интерфейс создания подразделения и плана обучения предоставлен ниже.

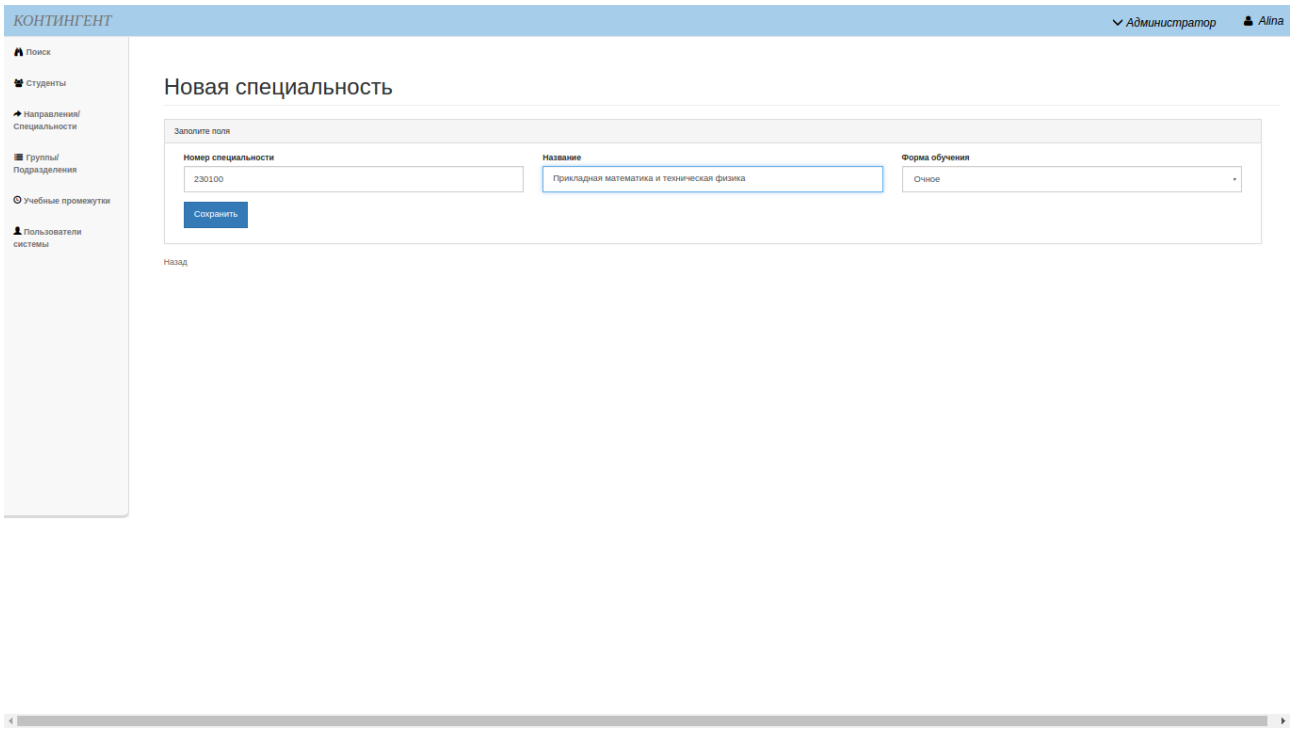



Рис. 4 Создание специальности

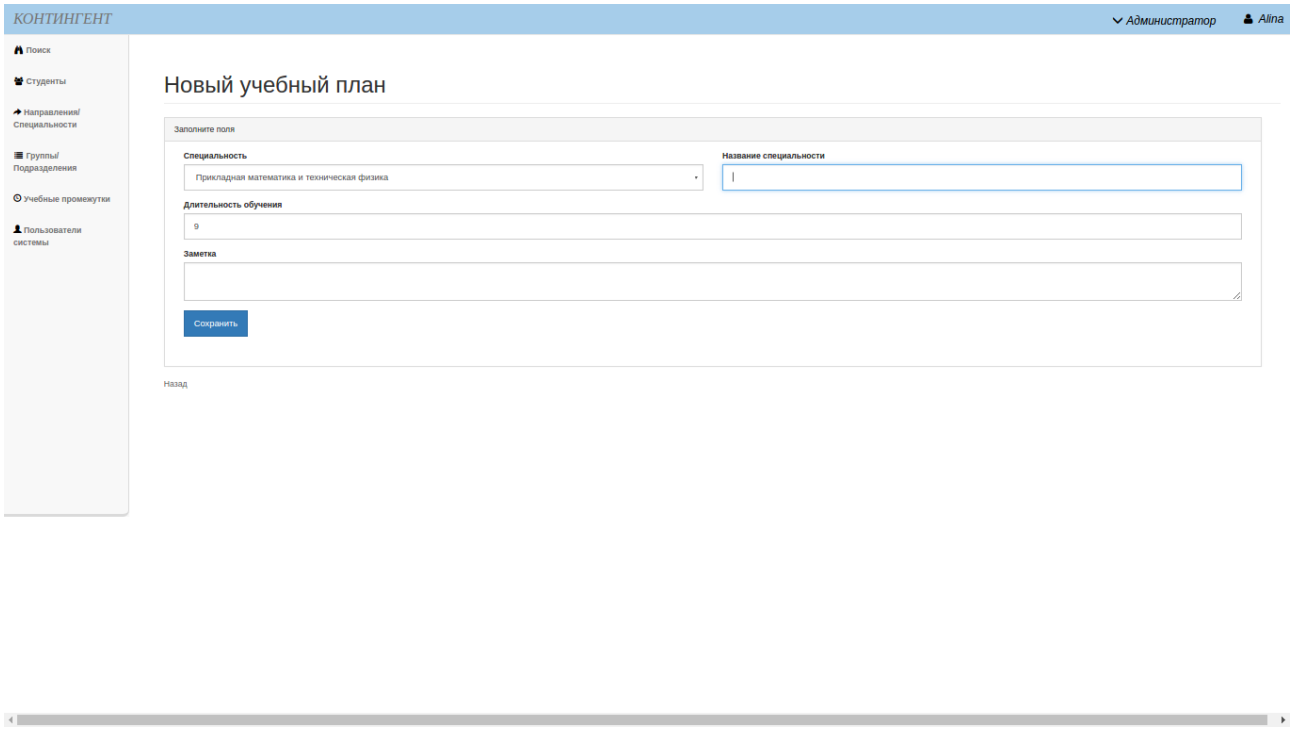



Рис 5. Создание плана обучения

План обучения имеет абстрактное значени, и не содержит полного описания плана обучения, так как цель системы — обработка информации о студентах, а не поддержка планирования обучения студентов как таковых.

Интефрейса плана и специальности разделен по категориям — это сделано для более удобной работы с готовыми списками данных.

## 5.3 Создание, редактирование и просмотр данных о студенте

Далее предлагается ознакомиться с результатами работы программы. По мере прочтения и просмотра результатов будут даваться краткие пояснения. Первая картинка отображает создание студента (Рис. 6), далее результат (Рис. 7).

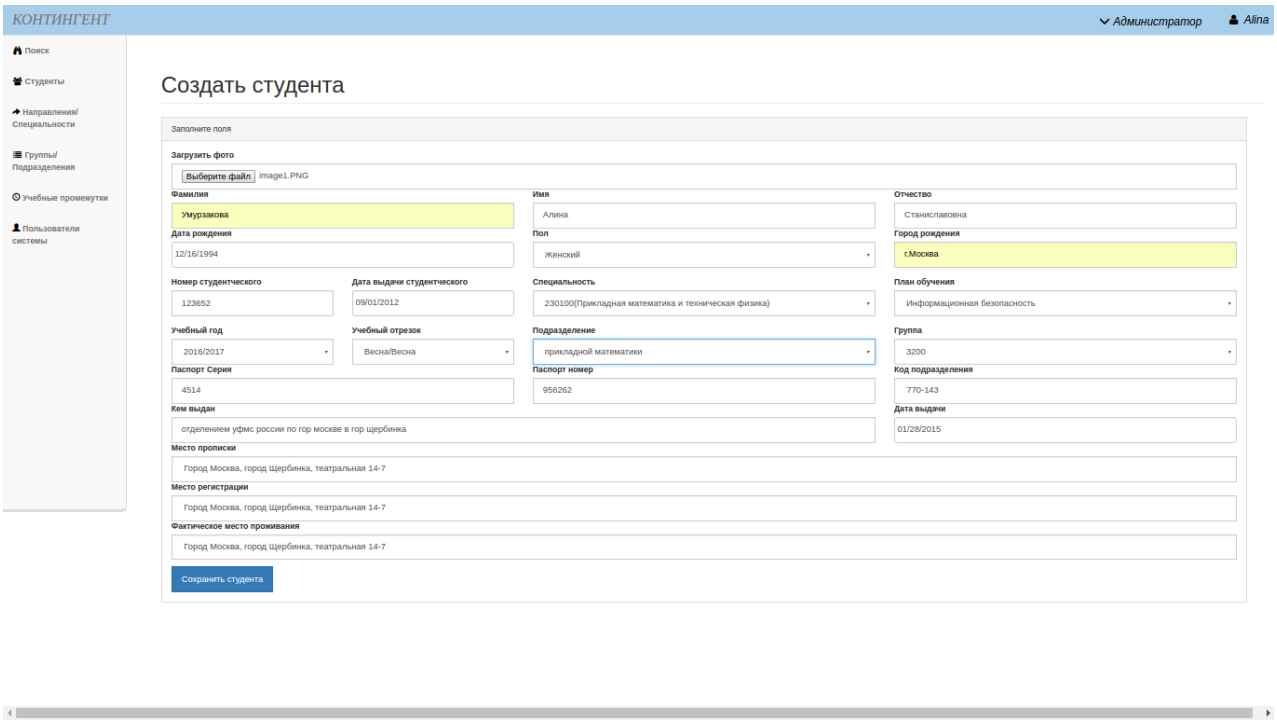


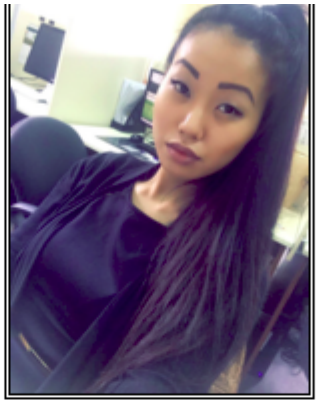

Рис. 6 Создание студента

Направления/ Специальности

Группы/ Подразделения

Учебные промежутки

Пользователи системы



Редактировать

| | |
|---|---|
| Имя: | Алина |
| Фамилия: | Умурзакова |
| Отчество: | Станислав |
| Дата рождения: | 1994-12-16 |
| Место рождения: | г.Москва |
| Пол: | Женский |
| Номер студенческого: | 123652 |
| Группа | 3200 |
| Подразделение: | туртуру |
| Специальность: | Прикладная |
| Направление: | Информаци |
| Учебный семестр | Весна/Весн |
| Учебный год | 2016/2017 |
| Когда выдан: | 2012-09-01 |

| | |
|---|---|
| Серия паспорта: | 4514 |
| Номер паспорта: | 956262 |
| Кем выдан: | отделением уфмс россии по гор москве в гор щербинка |
| Код подразделения: | 770-143 |
| Дата выдачи: | 2015-01-28 |
| Место прописки: | Город Москва, город Щербинка, театральная 14-7 |
| Место регистрации: | Город Москва, город Щербинка, театральная 14-7 |
| Место проживания (фактическое): | Город Москва, город Щербинка, театральная 14-7 |

*История изменений с момента создания до текущей даты актуализации (0)*

Список пуст



Рис. 7Первоначальная  форма отображения

Темпоральность этой системы проявляется только при развитии (внесением изменений) . Со временем студенты меняют место адреса, паспорта и т. п. (Рис. 8)

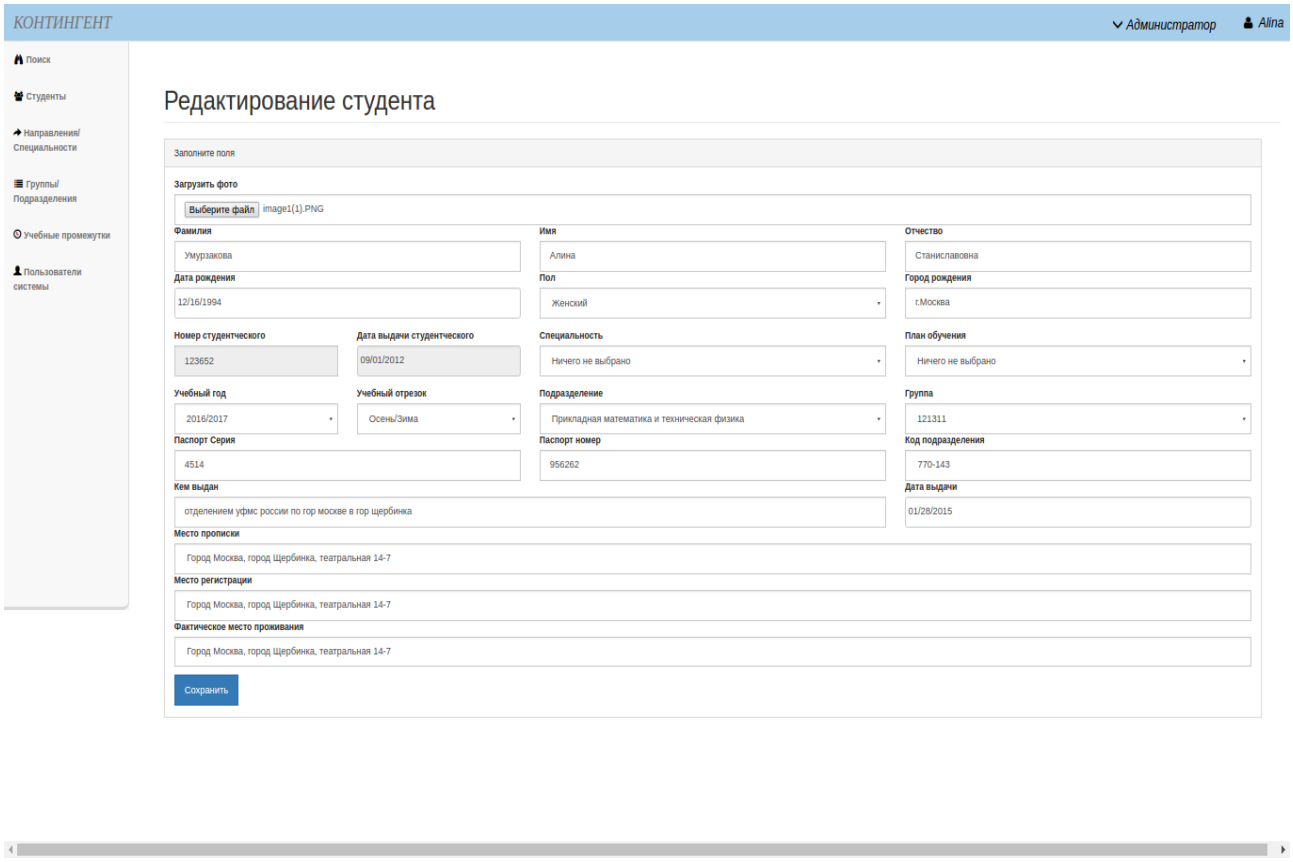



Рис. 8 Редактирование данных студента

Номер студентчиского остается неизменным на протяжении всего периода обучения студента, потому он закрыт от изменений. Даже при потери студентческого рассчитывается на то что, его номер не измениться. Соответственно дату так же можно не менять, так как с точки зрения системы ничего кроме даты выдачи в этом случае не измениться, соответственно лишняя работа.

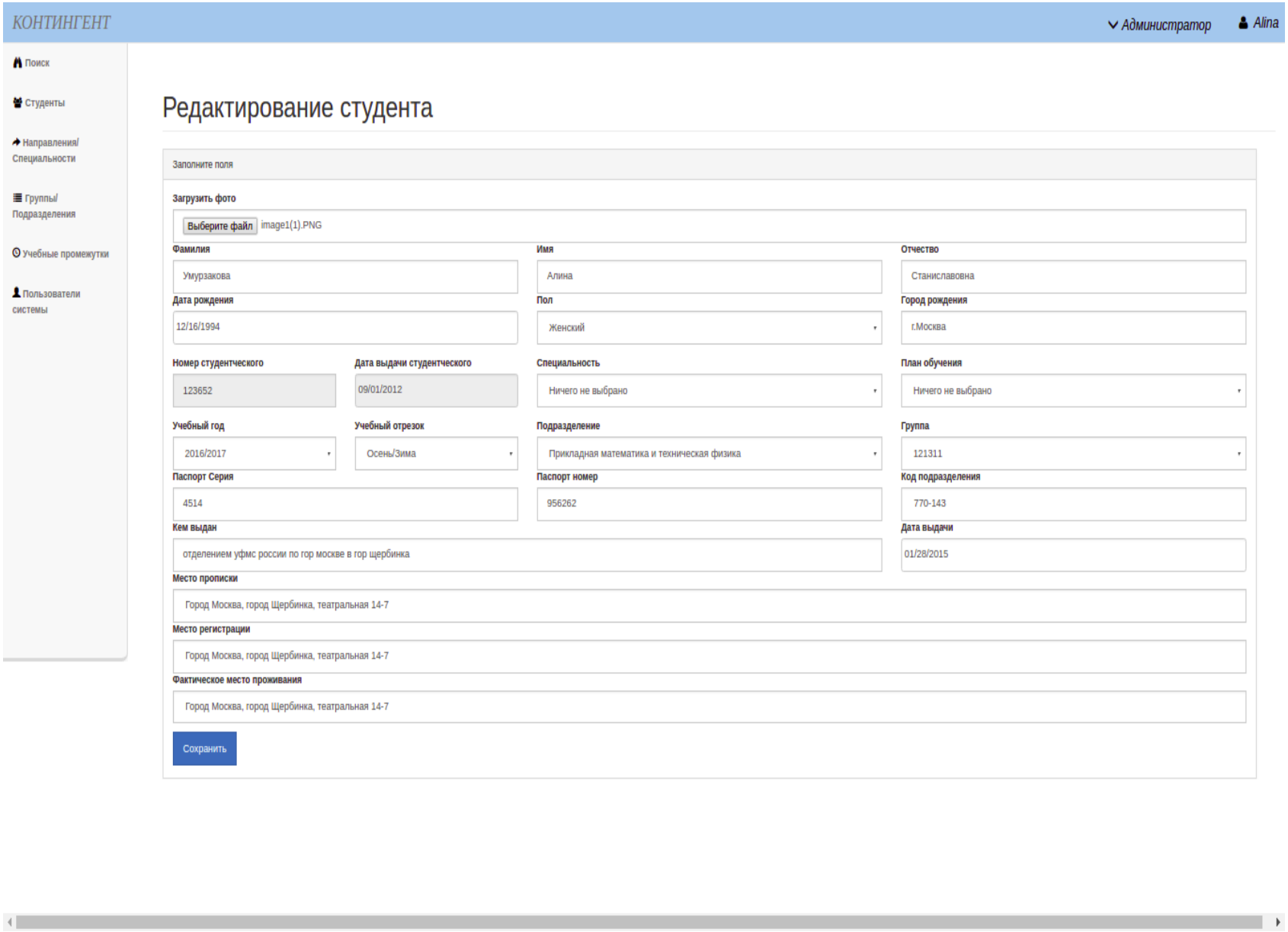



Рис. 9 Студент с течением временины

При изменении данных о студентах, у пользователя появляется возможность просмотра этого объекта по дате актуализации. Для удобства шкала времени была представлена в виде полосы с отмеченными датами изменений на ней. Благодаря этим отметкам осуществляется переход представления от текущего состояния студента к состоянию указанному временем. Данная работа выполняется благодаря технологии AJAX, входящую в состав Jquery (Рис. 10).

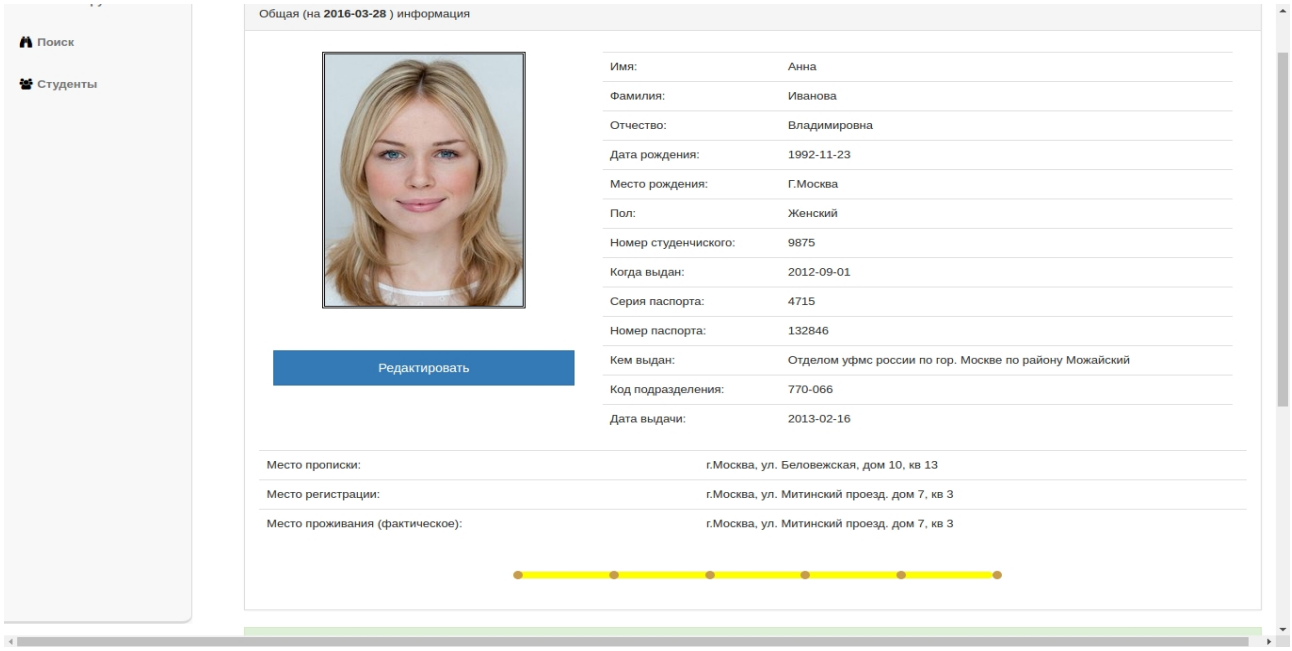



Рис. 10 Просмотр состояния студента по дате актуализации

Для быстрого ориентирования показывается список истории изменений с постраничной навигацией, а так же сортировкой по дате от последний до первых (Рис. 11).

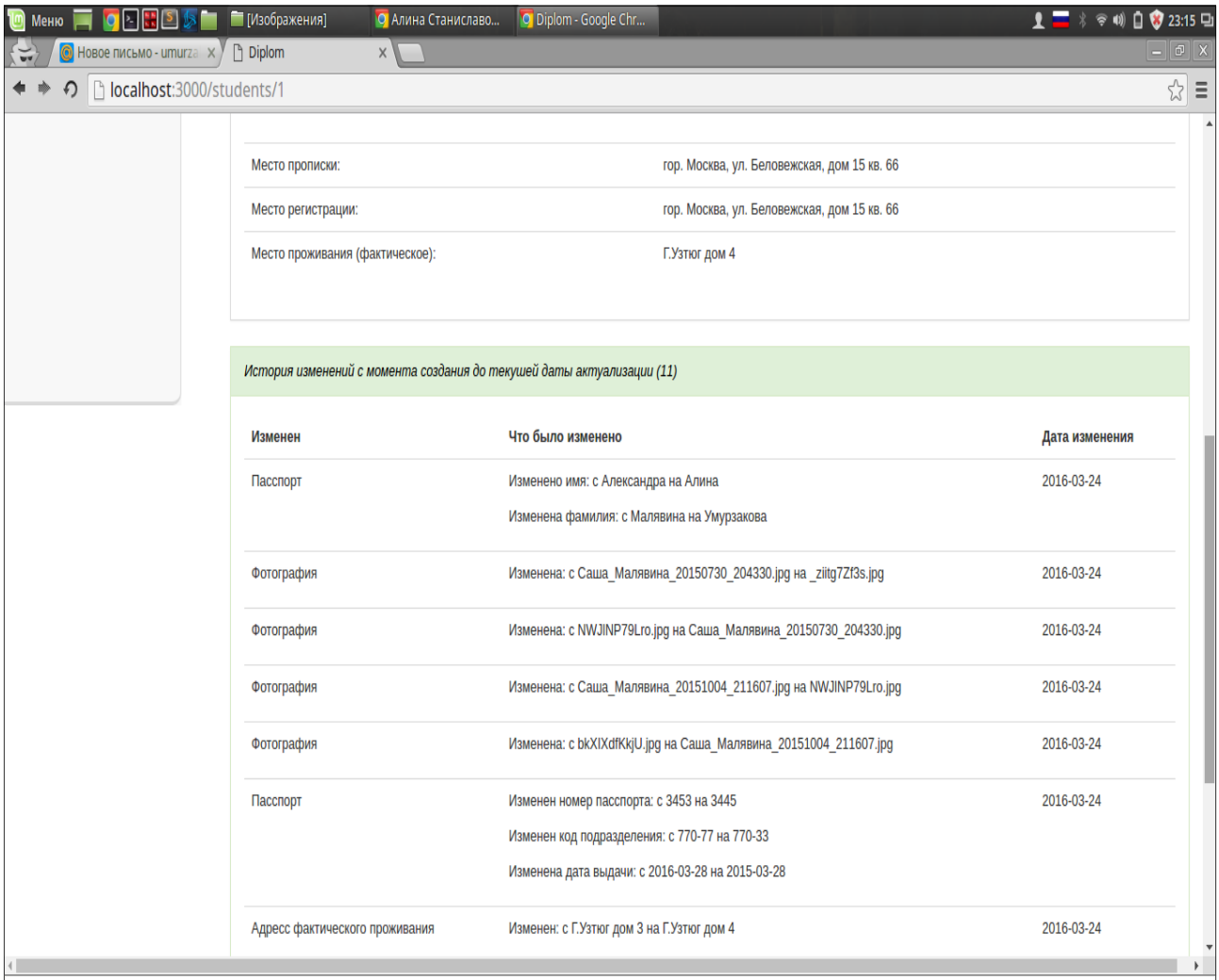



Рис. 11 История изменений

История изменений показывает не только когда было сделано изменение, но и показывает предыдущее значение/состояние данного поля/аттрибута/объекта, что положительно сказывается на восприятие информации.

## 5.4 Ролевая сисетема

Одной из задач было создание гибкой ролевой системы. Значение которой инкапсулировать действия пользователей по привилегиям доступа к сервису. А так же иметь возможность создания отдельных ролей с набором тех прав, которые необходимы тому или иному сотруднику, работающему с данной системой. Для этого была разработана следующая архитектура зависимостей (рис. 12):

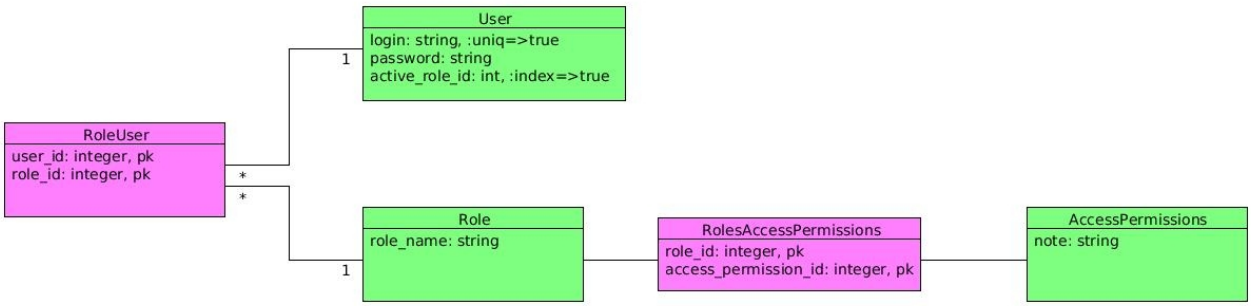



Рис. 12 Архитектура ролевой системы

**User** - Данные о пользователе системы. Содержит логин, пароль в зашифронном виде, а так же время последнего входа в систему.

**AccessPermissions** - Набор привелегий для доступа к системе. Статичны, задаются при развертовании проекта.

**Role** – Роль пользователя в системе. Они связаны с набором привелегий в системе. Динамическое создание и редактирование.

Далее приведены скриншоты данное интерфейса с последующими коментариями.

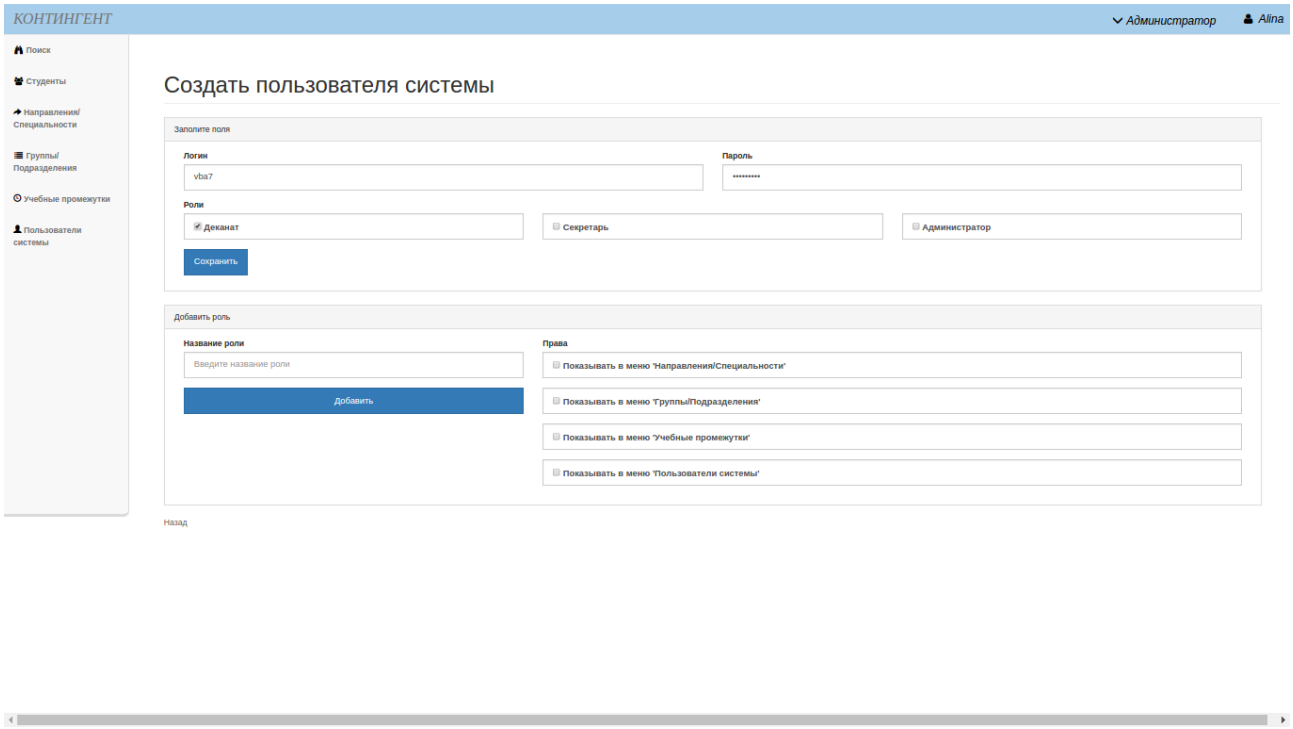



Рис. 13 Интерфейс создания пользователя

Данный интерфейс поддерживает асинхронное создание ролей. Выбирается набор привелегий и записывается название роли. После нажатия на кнопку добавления происходит обмен данными с сервером и при успешном завершении в форме создания пользователя появляется новая роль с вожность выбора.

При создании пользователя выбирается группа ролей, а так же указывается логин и пароль.

После создания пользователя в общем списке показаны роли и права которыми обладает пользователь и время последней авторизации в системе (Рис. 14).

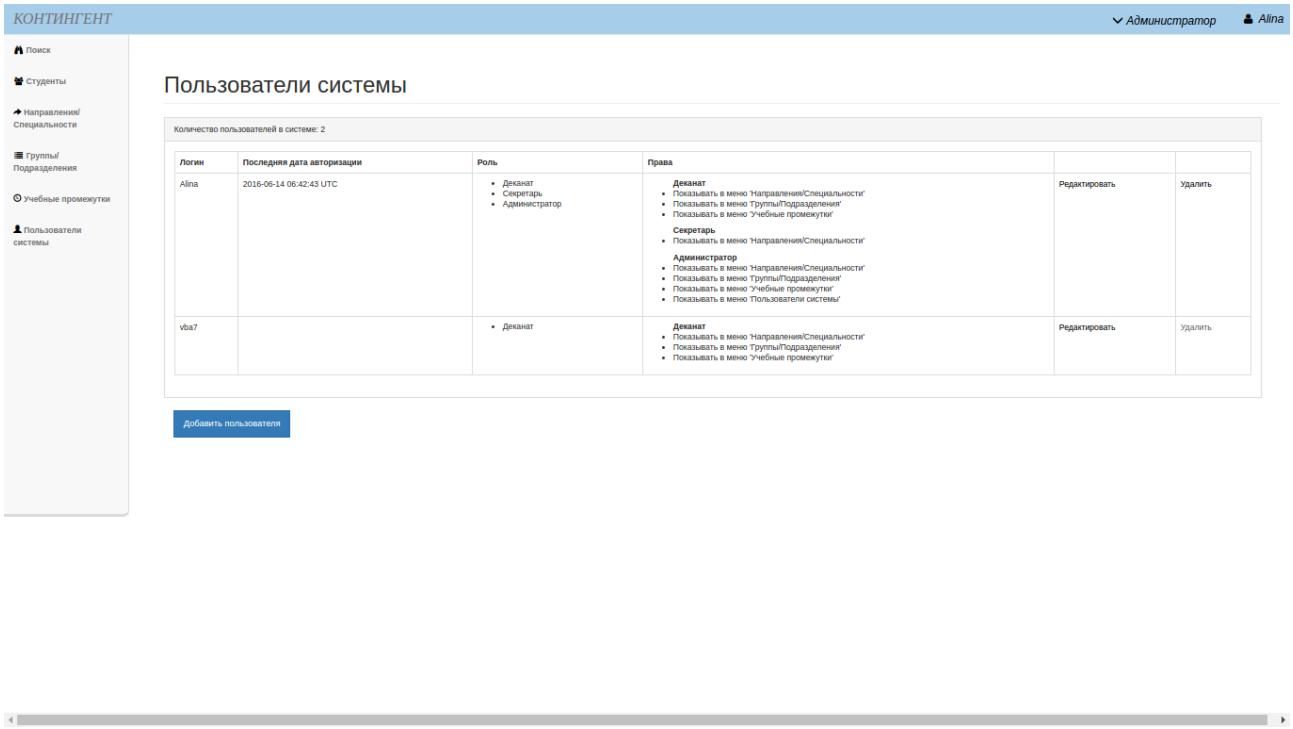



Рис. 14 Список пользователей системы

На рисунке предоставлен список пользователей с возможностю редактиврония, а так же удаления из системы. Между ролями можно переключаться прямо в интерактивном режиме (Рис. 15).

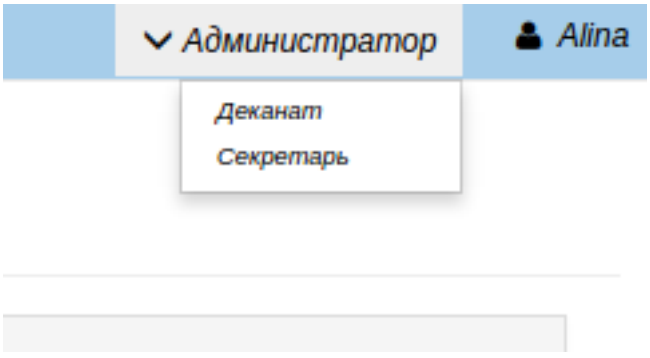



Рис. 15. Переключение ролей

# 6. Заключение

Результатом работы является собственная система темпоральной информационной системы учета контенгента студентов для образовательного портала. Реализована система поиска по темпоральным данным, с учетом изменения студентов в прошлом. Поддерживается гибкая ролевая схема, права которой получает пользователь при авторизации в системе. Причём, регистрацию новых пользователей может производить только администратор. Разделение прав пользователей, объединение прав доступа ролей. Возможность создания, поиска и просмотра данных о студентах по дате актуализации. Имеется стартовая страница для входа в систему с формами для ввода собственного логина и пароля. Пароль хранится в зашифрованном виде и не может быть получен удалённо злоумышленником в момент его ввода пользователем — тем самым обеспечивается надёжная защита аккаунтов пользователей от взлома и несанкционированного проникновения.

В процессе решения данного задания, были решены такие подзадачи как:

- Реализована архитектура хранимых данных;
- Построенно Web-приложение;
- Реализован нативный интерфейс;
- Благодаря применению в проекте шаблонов проектирования, сложная часть работы приложения делается незаметно от пользователя, тем самым облегчая его взаимодействие с системой;
- Реализована темпоральность для работы с объектом.

# 7. Список литературы и интернет-ресурсов


1) ru.wikipedia.org - справочная информация об используемых программах и дистрибутивах.
2) Томас Д., Хэнссон Д. Х., Гибкая разработка веб-приложений в среде Rails. - Санкт-Петербург: Питер, 2007. - 720 с.
3) Rails. Сборник рецептов. 1-е издание. - Санкт-Петербург: Питер, 2007. - 256 с.
4) railscasts.com - скринкасты о Ruby in Rails, советы разработчикам, обзор новых гемов.
5) github.com - крупнейший веб-сервис для хостинга IT-проектов и их разработки.
6) Эрик Фримен, Элизабет Фримен - Паттерны проектирования (Head First Design Patterns).
7) Саймон Ригс, Ханну Кросинг - Администрирование PostgreSQL 9. Книга рецептов.
8) Антон Шевчук – Jquery учебник для начинающих.
9) Карпов Ю.Г. - Темпоральные логики для спецификации свойств программных и аппаратных систем.
10) Дэвид Макфарланд - Большая книга CSS3.
11) htmlbook.ru – справочная информация по HTML, CSS.
12) www.youtube.com - популярный мультимедийный сервис, предостовляющий доступ к огромной коллекции видеоматериалов.
13) www.google.com – одна из крупнейших поисковых систем.
14) www.cyber**forum**.ru - форум начинающих и профессиональных программистов, системных администраторов, администраторов баз данных.

# 8. Приложение

## *8.1 Создание студента*

```
 def create
   begin
    @person = save_person(params)
    unless params[:propiska].nil?
     save_address(1, params[:propiska], @person)
    end
    unless params[:registration].nil?
     save_address(2, params[:registration], @person)
    end
    unless params[:fackt].nil?
     save_address(3, params[:fackt], @person)
    end
    unless params[:photo].nil?
     attr_photo =  {"photo"=>params[:photo]}
     @photo = Photo.new(attr_photo)
     @photo.person = @person
     @photo.save
     @delete_objects << @photo
    end
    save_passport(params, @person)
    @student = Student.new(student_params)
    @student.ducket_date = process_date(student_params[:ducket_date])
    @student.person = @person
unless @student.save
respond_to do |format|
        format.html{redirect_to @student, @student.errors}
      end
    else
      respond_to do |format|
       format.html{redirect_to @student, notice: 'Студент успешно создан'}
```

```
    end
  end
rescue ActiveRecord::StatementInvalid
  err = delete_objects()
  redirect_to '/students/new?err='+err.zip.to_s
end

End
```

*8.2 Редактирование студента*

```
def update
   old_id = @student.person.photos.last.id
   unless params[:photo].nil?
     attr_photo =  {'photo'=>params[:photo]}
     @photo = Photo.new(attr_photo)
     @photo.person = @student.person
     @photo.save
     @delete_objects << @photo
     save_change(@photo.model_name.name, @student.person.id, old_id, @photo.id)
   end

   passport = new_passport(params, @student.person)
   old_id = @student.person.passports.last.id
   unless (@student.person.passports.last == passport)
     passport.save
     @delete_objects << passport
     save_change(passport.model_name.name, @student.person.id, old_id, passport.id)
   end

   unless params[:propiska] == @p_address.address
     old_id = @p_address.id
     @delete_objects << save_address_with_change(1, params[:propiska], @student.person,
old_id)
   end
   unless params[:registration] == @r_address.address
     old_id = @r_address.id
     @delete_objects << save_address_with_change(2, params[:registration], @student.person,
old_id)
   end
   unless params[:fackt] == @f_address.address
```

```
    old_id = @f_address.id
    @delete_objects << save_address_with_change(3, params[:fackt], @student.person, old_id)
  end
  # Студенчиский будет неизменным
  respond_to do |format|
    format.html{redirect_to @student, notice: 'Студент отредактирован успешно'}
  end

end
```

### 8.3 Дополнительные функции

#### 8.3.1 Сохранение изменений

```
def save_change(name, person_id, old_id, new_id)
  c = nil
 begin
  c = ChangeList.create({m_type: name, person_id: person_id,  old_id: old_id, new_id: new_id})
 rescue Exception
  print "При сохранении изменений произошли ошибки #{name} #{person_id} #{old_id} #{new_id}"
 end
 return c
End
```