

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

Кафедра: «Информатика и вычислительная техника»

## КУРСОВАЯ РАБОТА

по дисциплине «Проектирование и разработка корпоративных  
информационных систем»

на тему «Разработка темпоральной информационной системы учета  
контингента студентов»

Группа	<u>121131</u>	
Студент	_____	Умурзакова А.С.
Оценка работы	_____	
Преподаватель	_____	Лукьянова Н.В.

МОСКВА 2016

## **Аннотация**

Работа посвящена созданию системы для темпоральной информационной системы учета контингента 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

1. Введение.....	4
2. Литературный обзор ....	5
3. Проектирование системы.....	10
4. Обзор используемых технологий.....	12
5. Ход работы.....	15
6. Описание пользовательских интерфейсов .....	20
7. Список литературы.....	27
8. Приложение.....	28

## Введение

Мы живем в интенсивно развивающемся 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 защиты 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 фреймвор Ruby on Rails.

Фреймворм 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 программирования 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.[1]

**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 системы с 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 коммерческими 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 операционными 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 ВУЗ-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.

## 2.2 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 принципы **Ruby on Rails**:

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

### 3.2 JAVASCRIPT

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

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

Например, при соответствующих настройках браузеры способны разрешать сценариям считывать файлы, в которых могут содержаться важные данные, например, 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** - библиотека 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.

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

1. Адрес (фактический, регистрация, прописка);
2. Группа в которой студент учится;
3. Направление;
4. Специализация;
5. Паспортные данные.

Так же для удобства пользователя у студента будет фотография (аватар), который так же можно будет изменять.

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):

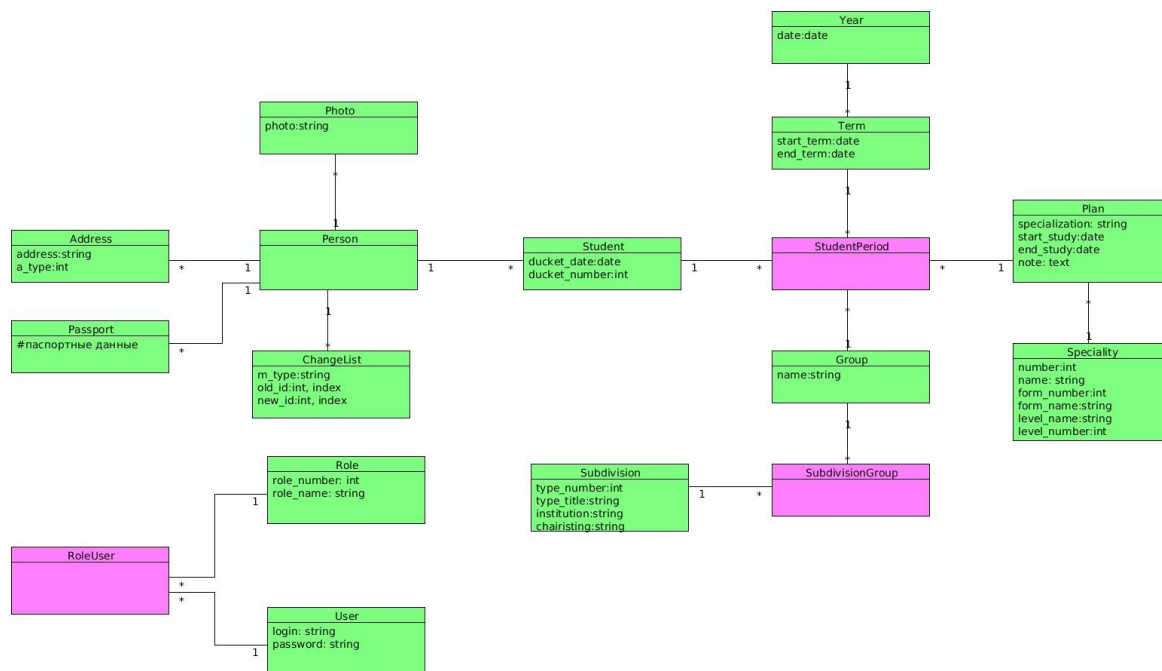


Рис. 1 Диаграмма

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 персоны, для упрощенного выбора студента при поиске.

**Address** — объект хранящих 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 номер студенческого, который неизменный, 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 обучения. С каждым новым учебным отрезком студент переходит в новую группу.

**Subdivision** — подразделения группы.

**Plan** — план обучения.

**Specialization** — Специализация. Одна специализация может содержать различное количество планов работы, обучения.

**Year** — годовой период обучения. К примеру 2015/2016 год. Содержит дату начала и окончания обучения.

**Term** — учебный семестр, часть учебного года. Так же содержит дату начала и окончания.

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, CoffeeScript, 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 выше сказано, 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 привелегий, для облегченного восприятия информацией. К примеру возможно создать роль администратора системы с привелегией удаления/редактирования и создания, к примеру, 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 возможность влиять на данные о пользователях на уровне контроллеров, но и так же не будет отображаться сама панель доступа. Но 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 привелегий.

Примерное взаимодействие с приложением отображено на рисунке 2.

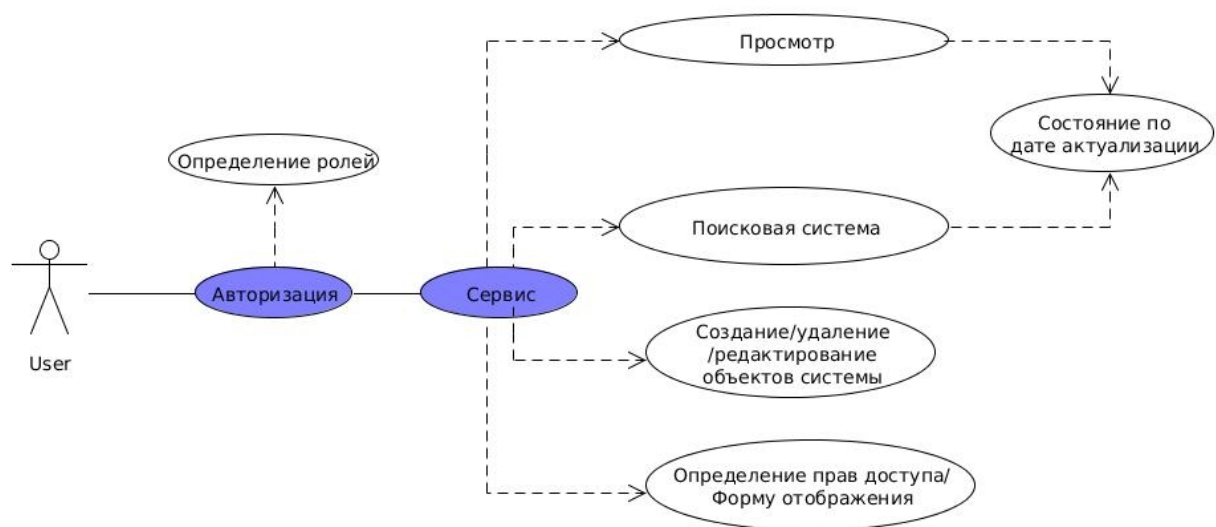


Рис. 2 Диаграмма использования

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 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. 3), далее результат (Рис. 4).

The screenshot shows a web application interface for creating a student record. The header is blue with the text 'КОНТИНГЕНТ' and a user profile 'Alina'. A left sidebar contains navigation links: 'Списки групп', 'Поиск', and 'Студенты'. The main area is titled 'Создать студента' and contains a form with the following fields:

- Загрузить фото:** A button 'Выберите файл' and a filename 'y/SIOghw-Xk.jpg'.
- Имя:** Анна
- Фамилия:** Себеряк
- Отчество:** Владимировна
- Дата рождения:** 11/23/1992
- Пол:** Женский
- Город рождения:** Г.Москва
- Номер студенческого:** 9875
- Дата выдачи студенческого:** 09/01/2012
- Паспорт Серия:** 4513
- Паспорт номер:** 018720
- Код подразделения:** 770-066
- Кем выдан:** Отделом уфмс россии по гор. Москве по району Можайский
- Дата выдачи:** 02/16/2013
- Место прописки:** г.Москва, ул. Беловежская, дом 10, кв 13
- Место регистрации:** г.Москва, ул. Голубинская, дом 28, кв 256
- Фактическое место проживания:** г.Москва, ул. Голубинская, дом 28, кв 256

At the bottom of the form is a blue button labeled 'Create Student'.

Рис. 3 Создание студента

Списки групп

Поиск

Студенты

Общая (на 2012-09-04) информация



Редактировать

Имя:	Анна
Фамилия:	Себеряк
Отчество:	Владимировна
Дата рождения:	1992-11-23
Место рождения:	г.Москва
Пол:	Женский
Номер студенческого:	9875
Когда выдан:	2012-09-01
Серия паспорта:	4513
Номер паспорта:	018720
Кем выдан:	Отделом уфмс россии по гор. Москве по району Можайский
Код подразделения:	770-066
Дата выдачи:	2013-02-16
Место прописки:	г.Москва, ул. Беловежская, дом 10, кв 13
Место регистрации:	г.Москва, ул. Голубинская, дом 28, кв 256
Место проживания (фактическое):	г.Москва, ул. Голубинская, дом 28, кв 256

Рис. 4 Первоначальная форма отображения

Темпоральность этой системы проявляется только при развитии (внесением изменений) . Со временем студенты меняют место адреса, паспорта и т. п. (Рис. 5)

КОНТИНГЕНТ Alina

Списки групп  
Поиск  
Студенты

### Редактирование студента

Заполните поля

Загрузить фото  
Выберите файл | Файл не выбран

Имя	Фамилия	Отчество
Анна	Иванова	Владимировна
Дата рождения	Пол	Город рождения
11/23/1992	Женский	Г.Москва
Номер студенческого	Дата выдачи студенческого	
9875	09/01/2012	
Паспорт Серия	Паспорт номер	Код подразделения
4715	132846	770-066
Кем выдан	Дата выдачи	
Отделом уфмс россии по гор. Москве по району Можайский	02/16/2013	
Место прописки		
г.Москва, ул. Беловежская, дом 10, кв 13		
Место регистрации		
г.Москва, ул. Митинский проезд, дом 7, кв 3		
Фактическое место проживания		
г.Москва, ул. Митинский проезд, дом 7, кв 3		

Рис. 5 Редактирование данных студента


Номер студенческого остается неизменным на протяжении всего периода обучения студента, потому он закрыт от изменений. Даже при потери студенческого 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.

Списки групп

Поиск

Студенты

Общая (на 2016-03-28) информация



Редактировать

Имя:	Анна
Фамилия:	Иванова
Отчество:	Владимировна
Дата рождения:	1992-11-23
Место рождения:	г.Москва
Пол:	Женский
Номер студенческого:	9875
Когда выдан:	2012-09-01
Серия паспорта:	4715
Номер паспорта:	132846
Кем выдан:	Отделом УФМС России по гор. Москве по району Можайский
Код подразделения:	770-066
Дата выдачи:	2013-02-16

Место прописки:	г.Москва, ул. Беловежская, дом 10, кв 13
Место регистрации:	г.Москва, ул. Митинский проезд, дом 7, кв 3
Место проживания (фактическое):	г.Москва, ул. Митинский проезд, дом 7, кв 3

2014-12-04 06:46:32 UTC

Рис. 6 Студент с течением времени

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. 6).

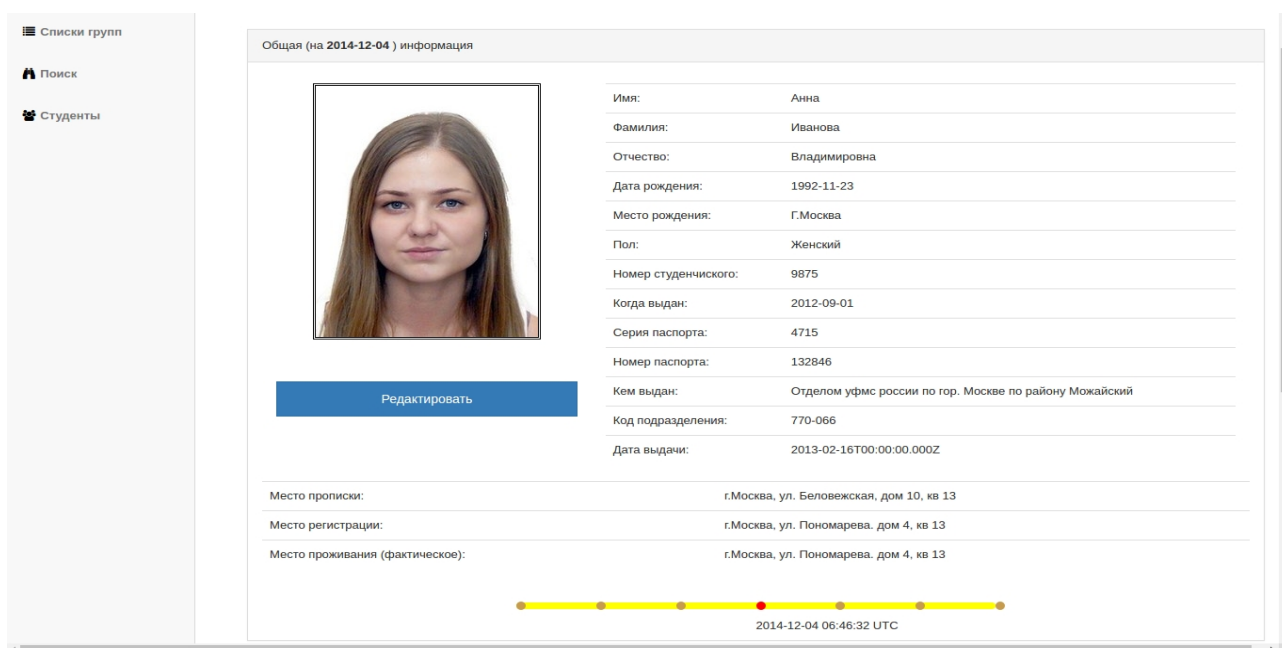



Рис. 6 Просмотр состояния студента по дате актуализации

Для быстрого ориентирования показывается список истории изменений с постраничной навигацией, а так же сортировкой по дате от последний до первых (Рис. 7).





Серия паспорта: 4715

Номер паспорта: 132846

Кем выдан: Отделом уфмс россии по гор. Москве по району Можайский

Код подразделения: 770-066


Дата выдачи: 2013-02-16

Редактировать

Место прописки: г.Москва, ул. Беловежская, дом 10, кв 13

Место регистрации: г.Москва, ул. Митинский проезд, дом 7, кв 3

Место проживания (фактическое): г.Москва, ул. Митинский проезд, дом 7, кв 3



История изменений с момента создания до текущей даты актуализации (6)

Изменен	Что было изменено	Дата изменения
Фотография	Изменена: с yjSIOghw-Xk.jpg на Rj3PFzoo4O4.jpg	2016-03-28
Адрес фактического проживания	Изменен: с г.Москва, ул. Пономарева, дом 4, кв 13 на г.Москва, ул. Митинский проезд, дом 7, кв 3	2014-12-04
Адрес регистрации	Изменен: с г.Москва, ул. Пономарева, дом 4, кв 13 на г.Москва, ул. Митинский проезд, дом 7, кв 3	2014-12-04
Паспорт	Изменена серия паспарта: с 4513 на 4715 Изменен номер паспорта: с 018720 на 132846 Изменена фамилия: с Себеряк на Иванова	2014-12-04
Адрес фактического проживания	Изменен: с г.Москва, ул. Голубинская, дом 28, кв 256 на г.Москва, ул. Пономарева, дом 4, кв 13	2012-09-04

Рис. 7 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.

## 6. Заключение

Результатом работы является собственная система темпоральной информационной системы учета контингента 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;
- Построено 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](http://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](http://railscasts.com) - скринкасты о Ruby in Rails, советы разработчикам, обзор новых гемов.
- 5) [github.com](http://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](http://htmlbook.ru) – справочная информация по HTML, CSS.
- 12) [www.youtube.com](http://www.youtube.com) - популярный мультимедийный сервис, предоставляющий доступ к огромной коллекции видеоматериалов.
- 13) [www.google.com](http://www.google.com) – одна из крупнейших поисковых систем.
- 14) [www.cyberforum.ru](http://www.cyberforum.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.docket_date =
process_date(student_params[:docket_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

# Студенческий 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
```