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 контенгента студентов*»*  Группа 121131 | | |
| Студент | \_\_\_\_\_\_\_\_\_\_\_\_\_ | А.С. Умурзакова |
| Руководитель | \_\_\_\_\_\_\_\_\_\_\_\_\_ | Радыгин. В.Ю. |
| ДОПУСКАЕТСЯ К ЗАЩИТЕ | | |
| Зав. каф. 36, доцент, к.ф.-м.н. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | /Роганов Е.А./ |

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

\_\_\_\_\_\_\_\_\_\_\_\_\_/Роганов Е.А. /

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г.

**Задание**

на выпускную работу по направлению «Информационные системы и технологии»

1. Тема работы «Разработкатемпоральной информационной системы учета контенгента студентов»
2. Сроки начала работы
3. Руководитель дипломной работы: Радыгин Виктор Юрьевич
4. Задание дипломной работы:
   1. Исходные данные.
   2. Цель работы — разработкатемпоральной информационной системы учета контенгента 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 40 страниц, 13 иллюстраций, 1 приложение.

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

**Содержание**

1. Введение………………………………………………………………..4-5
2. Цели работы……………………………………………………..………6
3. Начало работы……………………………………………………..….....7
4. Ход работы………………………………………………………….……8
5. Вывод…………………………………………………….……………….9
6. Список литературы……………………………………….….…………..10
7. Приложение…………………………………………………………….11

**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 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 c 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.

# Литературный обзор

## 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:

**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.

## 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.

## 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.

# 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);
* **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;
* **REST** - шаблон для веб приложений;

## 3.2 JAVASCRIPT

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

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,

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** - библиотекriа JavaScpt, 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 - свободная объектно-реляционная система управления базами данных (СУБД).

## 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 выглядить конечный продукт. Что в нем должно присутствовать, и как с ним будут взаимодействовать пользователи.

Так как система подразумевает возможность наблюдения развития объекта с течением времени, то необходимо сделать гибкую систему, которая позволит прослеживать изменение различных атрибутов.

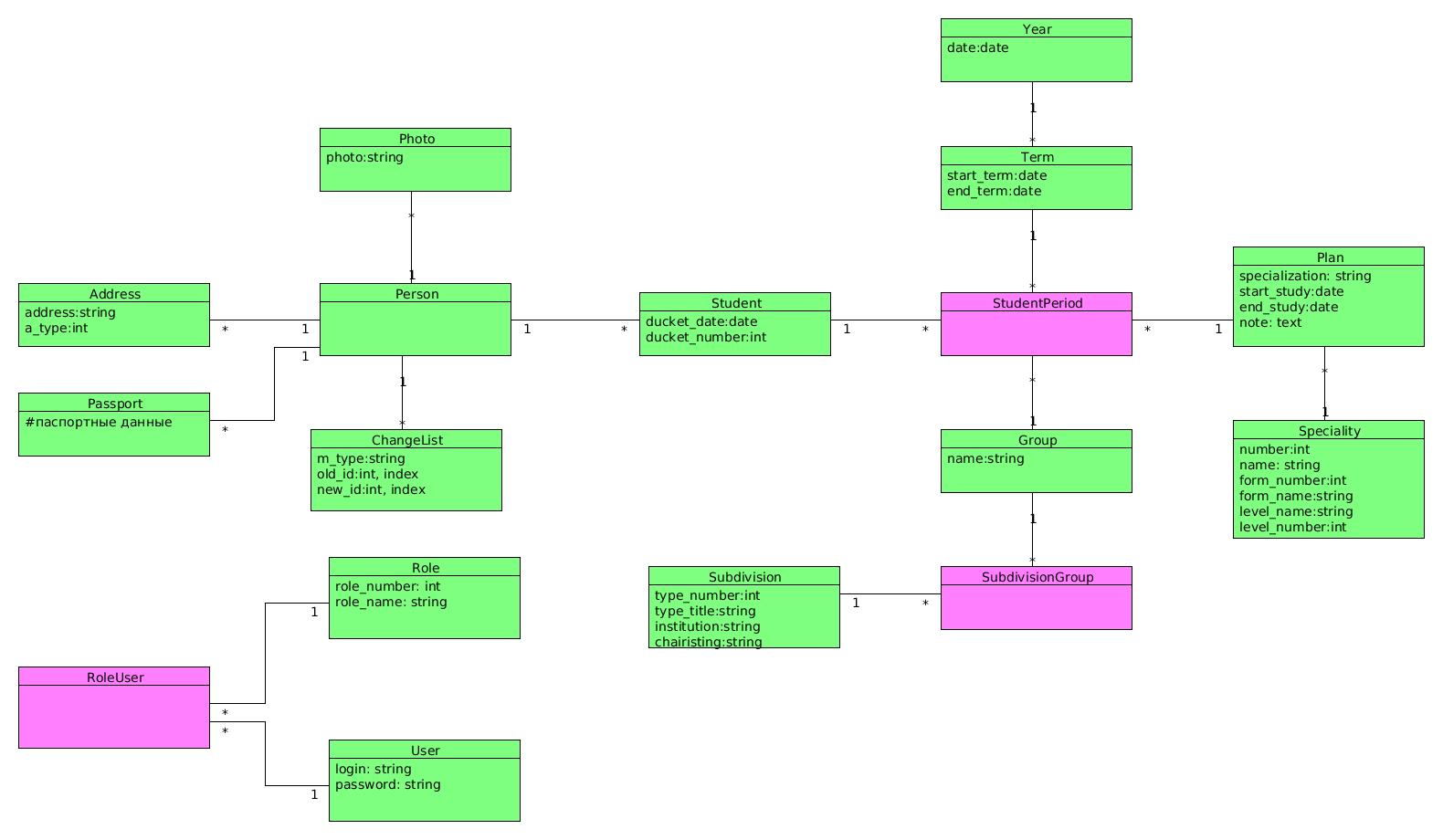
В данной дипломной работе объектом для наблюдения служат студенты ФГОУ ВПО «МГМУ» (МАМИ). Для начала рассмотрим какие атрибуты могут меняться со временем:

1. Адрес (фактический, регистрация, прописка);
2. Группа в которой студет 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. Возможность редактирования, просмотр данных о студенте в прошлом. В соответствии с требованиями была предложена следующая архитектура базы данных и связи объектов (рис. 1):

Рисунок 1

Рисунок 1

После того как принято предложеное решение, перейдем к созданию самой системы. Как было уже сказано, разрабтка будет производиться на платформе Ruby on Rails. Для этого необходимо обладать навыками программирования на языке программирования Ruby, JavaScript, CoffeScript и т.п. Перейдем к созданию самого приложения. Так же в приложении будет использоваться библиотека для адаптивного представления Bootstrap 3 версии.

Разработка web приложений на фреймворке Ruby on Rails очень прост:

1. Создаем новый проект коммандой: rails new Diplom;
2. Заходим в созданную папку и генерируем миграции;
3. Редактируем по необходимости полученный результат;
4. Создаем модели данных с которыми будем работать;
5. Задаем валидации.
6. Редактируем представления и контроллеры, обрабатывающие запросы от пользователей.

**Вывод**

Результатом работы является собственная система темпоральной информационной системы учета контенгента студентов для образовательного портала. Причём, регистрацию новых пользователей может производить только администратор системы. Разделение прав пользователей. Возможность создания, поиска и просмотра данных о студентах. 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.

**Список литературы и интернет-ресурсов.**

* 1. ru.wikipedia.org - справочная информация об используемых программах и дистрибутивах.

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

# Приложение

## .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

## .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

# .3 Дополнительные функции

# .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