министерство образования и науки РФ
московский государственный индустриальный университет
кафедра информационных систем и технологий

Руководители работы:
доцент, к.т.н. Куприянов Д.Ю.
ассистент Александров А.И.

Войнов Максим Александрович

«Разработка Информационной системы учета успеваемости и посещаемости слушателей ФДО МГИУ»

Курсовая работа по дисциплине
«Проектирование и разработка корпоративных информационных систем»
4-й курс, 7-й семестр

Москва 2011

Аннотация

Курсовая работа посвящена описанию дипломного проекта «Информационная система учета успеваемости и посещаемости слушателей ФДО МГИУ». В данной работе описываются актуальность темы и постановка задачи с описанием её планируемого функционала, обоснование выбора архитектуры планируемой информационной системы и обзор технологий, которые используются для построения аналогичных систем, описание проектирования системы, описание интерфейсов с примерами скриншотов.

Оглавление

1. Введение 4
2. Используемые технологии 5
3. Структура базового проекта 6
4. Разработанные модели 7
5. Пример реализации одного из классов 8
6. Графический интерфейс 13

1. Введение

Последние пять лет ознаменовались фантастическим развитием Интернета и новых способов общения между людьми. На переднем крае этого явления находится *World Wide Web (WWW)*. Ежедневно в этой новой коммуникационной среде открываются тысячи новых сайтов, а потребителям предлагаются новые виды услуг. Вместе с бурным развитием рынка появился огромный спрос на новые технологии и разработчиков, владеющих ими. Комплексная веб разработка сайтов различной тематики и направленности предусматривает создание нового или оптимизацию под нужные характеристики уже готового шаблона сайта, выбор и установку наиболее подходящей системы управления контентом и, при необходимости, заполнение ресурса контентом.Чтобы создать удобный и функциональный web-сайт используют различные технические средства, например HTML, JavaScript, Flash, различные СУБД. В данной работе были использованы HTML, СУБД PostgreSQL и платформа *Ruby on Rails*.
Используя данные технологии можно получить информационную систему учета успеваемости и посещаемости слушателей Факультета Довузовского Образования (ФДО) МГИУ, которая позволяет формировать учебные группы, контролировать успеваемость и посещаемость слушателей ФДО, а также вести отчетность по учебным дисциплинам и группам.
Данную систему можно создать решив следующие задачи:

- реализация интерфейса контроля успеваемости слушателей ФДО. преподаватель будет проставлять баллы в определенной дисциплине, по определенному предмету, в указанную дату;

- реализация интерфейса контроля посещаемости слушателей ФДО. преподаватель будет через специально организованный интерфейс выставлять по определенному предмету, в конкретную дату отметку о том был ли абитуриент на данном занятии, с использованием системы drag&drop;

- реализация интерфейса предоставления отчетов по любой запрошено пользователем информации.

С точки зрения конечного пользователя это означает, что в системе должен быть предусмотрен интерфейс для учета успеваемости слушателей ФДО (преподаватель будет проставлять баллы в определенной дисциплине, по определенному предмету, в указанную дату), также контроль посещаемости слушателей ФДО(будет реализована система учета посещаемости слушателями ФДО, учебных курсов (преподаватель будет через специально организованный интерфейс выставлять по определенному предмету, в конкретную дату отметку о том был ли абитуриент на данном занятии) и интерфейс формирование учебных групп подготовительных курсов с учетом всех выбранных слушателем направлений довузовской подготовки: будет реализован интерфейс организации учебных групп, исходя из списков слушателей и выбранных ими дисциплинами с использованием системы drag&drop.

2. Используемые технологии

Ruby on Rails – это полноценный, многоуровневый фреймворк для построения веб-приложений, использующих базы данных, который основан на архитектуре Модель-Представление-Контроллер (Model-View-Controller, MVC). Динамичный AJAX-интерфейс, обработка запросов и выдача данных в контроллерах, предметная область, отраженная в базе данных, — для всего этого Rails предоставляет однородную среду разработки на Ruby.
Ключевое слово в Ruby on Rails – оптимизация. Главные преимущества данной системы:

- внятные сообщения об ошибках в браузере;
- очень удобное средство scaffold. Это когда на вход подается база данных, а затем одной командой создается как работающая система управления для этой базы данных, так и готовый сайт, отображающий эти данные;
- возможность вывода результата в XML через спец-шаблоны. Это и есть та точка, где Ruby можно связать с другими решениями, например используя Flash.

Основна парадигма фреймворка Ruby on Rails это MVC.
Model-View-Controller (MVC, «Модель - Представление - Контроллер») – шаблон проектирования, в котором модель данных приложения, пользовательский интерфейс и управляющая логика разделены на три отдельных компонента так, что модификация одного из компонентов оказывает минимальное воздействие на остальные. Шаблон MVC позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента:

- Модель (Model). Модель предоставляет данные (обычно для View), а также реагирует на запросы (обычно от контроллера), изменяя своё состояние.
- Представление (View). Отвечает за отображение информации (пользовательский интерфейс).
- Поведение (Controller). Интерпретирует данные, введённые пользователем, и информирует модель и представление о необходимости соответствующей реакции.

Важно отметить, что как представление, так и поведение зависят от модели. Однако модель не зависит ни от представления, ни от поведения. Это одно из ключевых достоинств подобного разделения. Оно позволяет строить модель независимо от визуального представления, а также создавать несколько различных представлений для одной модели. Основной принцип данного фреймворка : «Работа как отдых». Ввиду неоспоримых преимуществ работы с данной системой, именно фреймворк Ruby on Rails был выбран для реализации данного проекта.

3. Структура базового проекта

Рассмотрим структуру базового проекта. Для этого воспользуемся диаграммой классов UML. Диаграммы классов используются при моделировании программных систем (ПС) наиболее часто. Они являются одной из форм статического описания системы с точки зрения ее проектирования, показывая ее структуру. Диаграмма классов не отображает динамическое поведение объектов изображенных на ней классов. На диаграмме классов показываются классы, интерфейсы и отношения между ними.

Из данной диаграммы видно, что в системе присутствуют такие классы как:

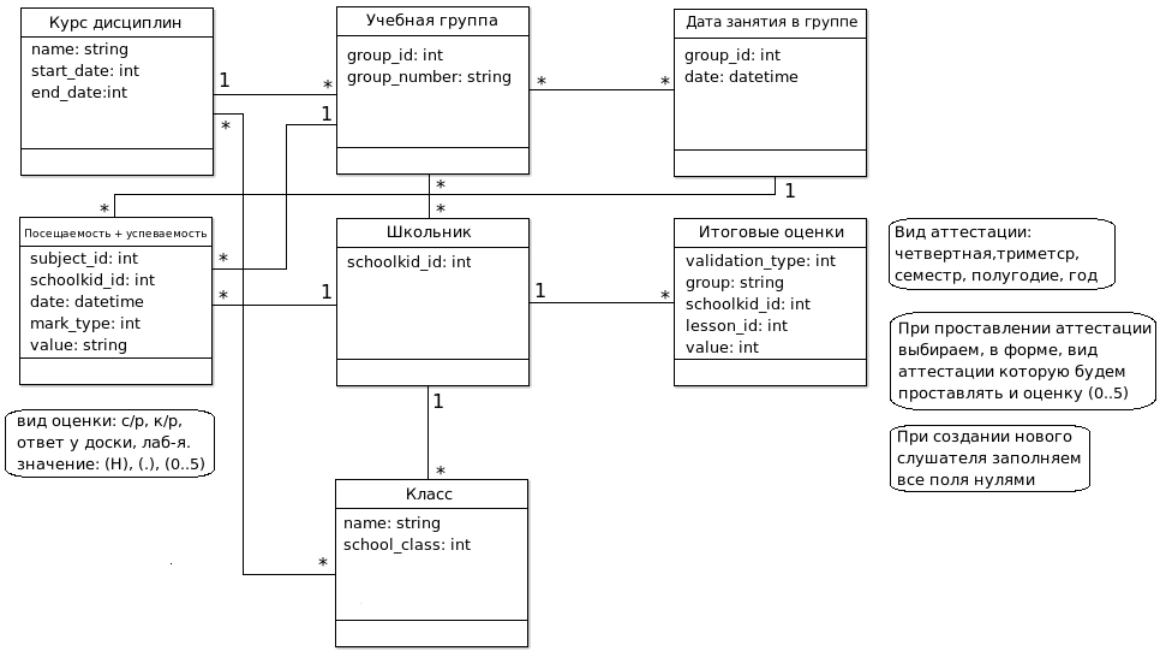

Рис. 1: Диаграмма классов

- курс дисциплин;
- учебная группа;
- школьник;
- посещаемость и успеваемость;
- дата занятия в группе;
- класс;
- итоговые оценки;

Класс **Итоговые оценки** и класс **Посещаемость и успеваемость** – классы хранилища, которые содержат информацию об успеваемости и посещаемости слушателей ФДО: оценки, информацию о посещаемости, и т.д. Класс **Итоговые оценки** находится в отношении один ко многим с классом **Школьник**.А класс **Посещаемость и успеваемость** связан отношением один ко многим, с классами **Школьник**, **Учебная группа** и **Дата занятия в группе**. Он содержит информацию о посещаемости и успеваемости слушателя ФДО на данный момент: Оценки и количество посещенных или пропущенных занятий.

4. Разработанные модели

Для начала определим, какие классы необходимо добавить в систему. Это будут следующие классы:

- **Школьники** - в этом классе будет храниться информация о школьниках. В этом классе следующие атрибуты: first_name, second_name, last_name, birthday, male, addres, telephone, school_group_id, group_id, created_at, updated_at.

- **Дисциплины** - в данном классе хранится список существующих дисциплин, со следующими атрибутами: full_name, short_name, created_at, updated_at.

- **Курсы** - содержит информацию о датах проведения курсов у слушателей ФДО. Имеет следующие атрибуты: start_date, finish_date, discipline_id, created_at, updated_at.

- **Школы** - хранится список школ, из которых поступают абитуриенты. Атрибуты : number, director_name, director_surname, director_pathname, created_at, updated_at.

- **Классы** - данный класс хранит информацию о школьных классах, в которых проходят обучение абитуриенты. Атрибуты : schoolkid_id, group_id, created_at, updated_at.

- **Учебные группы** - хранится информация об учебных группа ВУЗа. Данный класс содержит следующие атрибуты : number, year, school_id, stype, created_at, updated_at.

5. Пример реализации одного из классов

Использование генератора scaffold

мТеперь, после определения классов и их атрибутов, которые будут в нашей системе, можно переходить к реализации.
Так как для каждого класса необходимы модель, контроллер и представления воспользуемся генератором *scaffold*, и сгенерируем все части с его помощью.

```
rails g scaffold cources start_date:datetime finish_date:datetime
discipline_id:integer created_at:datetime updated_at:datetime

rails g scaffold disciplines full_name:string short_name:string
created_at:datetime updated_at:datetime

rails g scaffold groups number:string course_id:integer
year:integer gtype:boolean created_at:datetime updated_at:datetime

rails g scaffold school_groups number:string year:integer
school_id:integer stype:boolean created_at:datetime
updated_at:datetime

rails g scaffold schoolkids first_name:string second_name:string
last_name:string birthday:datetime male:boolean addres:string
telephone:string school_group_id:integer group_id:integer
created_at:datetime updated_at:datetime

rails g scaffold schoolkids_group schoolkid_id: integer
group:integer created_at:datetime updated_at:datetime

rails g scaffold schools number:string director_name:string
director_surname:string director_pathname:string
telephone:string created_at:datetime updated_at:datetime
```

Для корректной работы системы необходимо правильно выстроить отношения. Возьмем к примеру классы «Дисциплина» и класс «Курс». Для того чтобы связать эти классы отношением один ко многим, внесем изменения в модели **Discipline** и **Cource**

```
class Discipline < ActiveRecord::Base
  paginates_per 7
  default_scope order(:full_name)

  has_many :courses, :dependent => :destroy

  validates :full_name, :presence => true,
                        :uniqueness => true,
                        :length => {:in=>3...30}

  validates :short_name, :presence => true,
```

```
                          :uniqueness => true,
                          :length => {:in=>2...30}
end

class Course < ActiveRecord::Base
  paginates_per 30
  default_scope order(:start_date)

  belongs_to :discipline

  validates :start_date, :presence => true
  validates :finish_date, :presence => true
  validates :discipline_id, :presence => true

  attr_reader :discipline_token

  def discipline_token=(id)
    self.discipline_id = id
  end
end
```

Связь один ко многим с классом **Course** (belongs_to discipline). Также добавлены ограничения: Дата начала и конца дисциплины, а также идентификатор должны быть не пустыми.
В модель **Course** вводим метод attr_reader, передаем переменную discipline_token, он делает эту переменную доступной вне этого класса. И метод **discipline_token**, для того чтобы можно было использовать jquery плагин под названием token_input, который позволяет выбирать множество пунктов из предопределенного листа, используя автоподстановку для поиска каждого из элементов.

Контроллер и Представление

Теперь, после внесения всех необходимых изменений в модели, можно переходить к контроллерам. Контроллер интерпретирует данные, введённые пользователем, и информирует модель и представление о необходимости соответствующей реакции.
Для того чтобы пользователь мог добавлять курсы, нужно создавать отдельный интерфейс. Для этого просто добавим ссылку на страничку создания курса.
Рассмотрим, как это будет выглядеть для класса **Course**.

```
= form_for @course do |f|
  -if @course.errors.any?
    .msg.error
      %h2= "При сохранении произошли ошибки"
      %ul
        - @course.errors.full_messages.each do |msg|
          %li= msg

  .field
    = f.label :start_date
```

```
    %br
    = f.text_field :start_date
  .field
    = f.label :finish_date
    %br
    = f.text_field :finish_date
  .field
    =f.label :discipline_id
    %br
    =f.text_field :discipline_token,"data-pre"=>Discipline.where
(:id => @course.discipline_id).map(&:attributes).to_json
  .actions
    = f.submit 'Добавить'
```

Необходимо добавить небольшой скрипт, который будет посылать запрос в фоновом режиме к контролеру и ожидать результатов поиска. При этом контролер может извлекать данные из любого места, как, например, базы данных или жесткого диска. Но результаты поиска должны возвращаться в формате *JSON*.

```
$(function() {
    $("#course_discipline_token").tokenInput("/disciplines.json", {
crossDomain: false,
propertyToSearch: "full_name",
resultsFormatter: function(item){ return "<li>" + item.full_name +
"</li>" },
tokenFormatter: function(item){ return "<li><p>" + item.full_name +
"</p></li>"},
prePopulate: $("#course_discipline_token").data("pre"),
theme: 'facebook',
hintText: 'Введите дисциплину',
noResultsText: 'Не найдено',
searchingText: "Поиск...",
tokenLimit: "1"
    });
});
```

Также необходимо добавить создание, отображение, редактирование, удаление, изменение и сохранение нового курса в контроллере курса.

```
# -*- coding: utf-8 -*-
class CoursesController < ApplicationController
  before_filter :check_admin_user, :except=>['index', 'show']

  def index
    @courses = Course.page(params[:page])
  end

  def show
    @course = Course.find(params[:id])
    @dis = Discipline.where('id = ?', @course.discipline_id).first
```

```
  end

  def new
    @course = Course.new
  end

  def edit
    @course = Course.find(params[:id])
  end

  def create
    @course = Course.new(params[:course])
    if @course.save
      redirect_to @course, :notice => 'Курс дисциплины добавлен.'
    else
      render :action => "new"
    end
  end

  def update
    @course = Course.find(params[:id])
    if @course.update_attributes(params[:course])
      redirect_to @course, :notice => 'Курс сохранен.'
    else
      render :action => "edit"
    end
  end

  def destroy
    @course = Course.find(params[:id])
    @course.destroy
    redirect_to courses_url, :notice => 'Курс удален.'
  end
end
```

Если пользователь, при создании нового курса, не ввел никакой информации, то он не сохранится. Если не удалось сохранить курс (а это возможно только в одном случае – данные, введенные пользователем, не прошли проверку), делается возврат в форму, и отображается сообщение об ошибке пользователю.
Для отображения уже созданных курсов необходимо внести изменения в представления **Index** и **Show**

```
%p
  %b="#{Course.human_attribute_name(:start_date)}:"
  = @course.start_date.strftime("%d-%m-%Y")
  %br
  %b="#{Course.human_attribute_name(:finish_date)}:"
  = @course.finish_date.strftime("%d-%m-%Y")
  %br
-if @current_user && @current_user.admin?
  = link_to 'Редактировать', edit_course_path(@course)
```

```
  \|
= link_to 'Назад', courses_path
```

Представление Show

```
%h1 Курсы

%table
  %thead
    %tr
      %th=Course.human_attribute_name('start_date')
      %th=Course.human_attribute_name('finish_date')
      -if @current_user && @current_user.admin?
        %th
        %th
  %tbody
    -@courses.each do |course|
      %tr
        %td= link_to course.start_date.strftime("%d-%m-%Y"), course
        %td= link_to course.finish_date.strftime("%d-%m-%Y"), course
        -if @current_user && @current_user.admin?
          %td= link_to 'Редактировать', edit_course_path(course)
          %td= link_to 'Удалить', course, :confirm => 'Вы уверены?',
          :method => :delete

=paginate @courses
%br
-if @current_user && @current_user.admin?
  = link_to 'Добавить', new_course_path
```

Представление Index

Аналогичным образом реализуются остальные классы.

6. Графический интерфейс

Приведем пример работы.

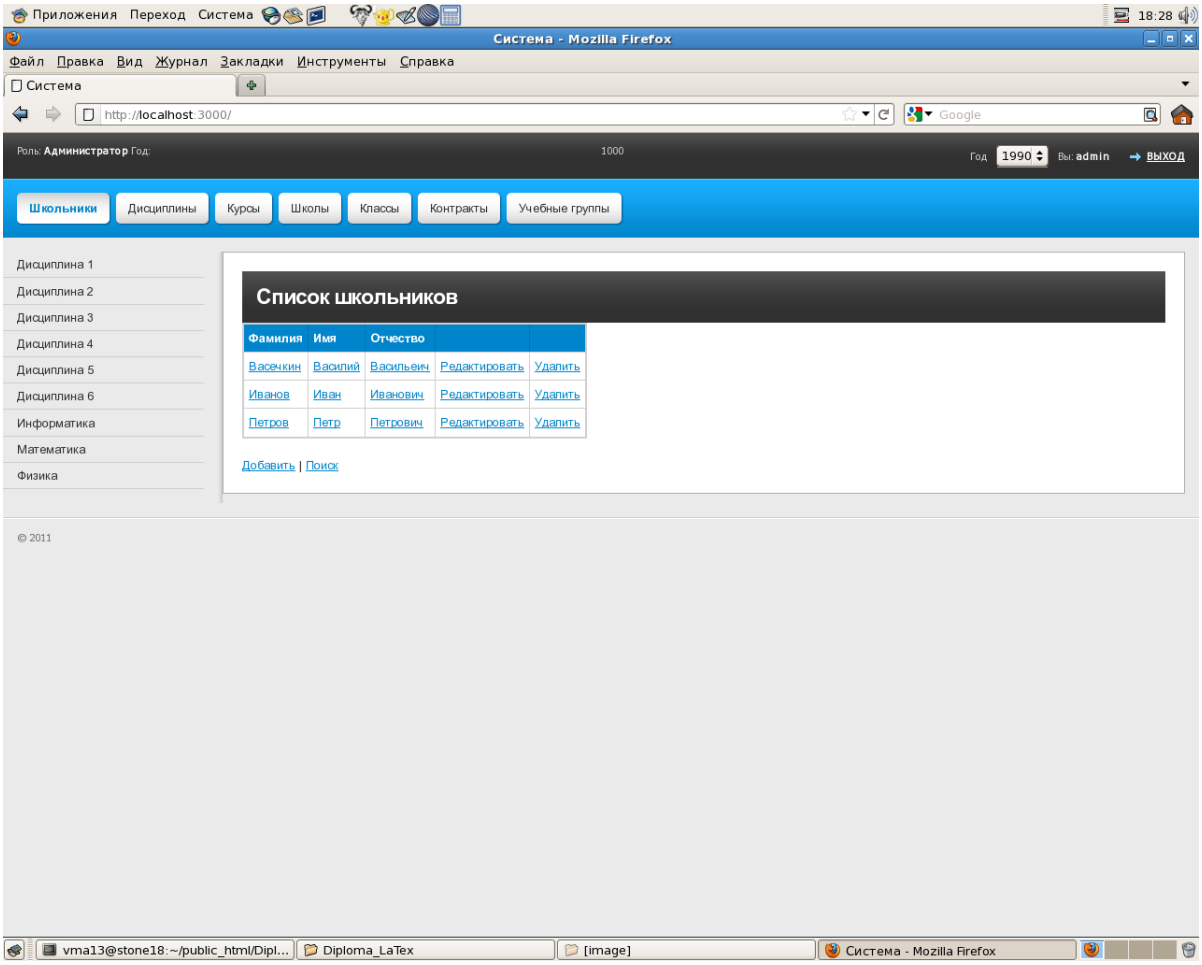

Рис. 2: Домашняя страничка

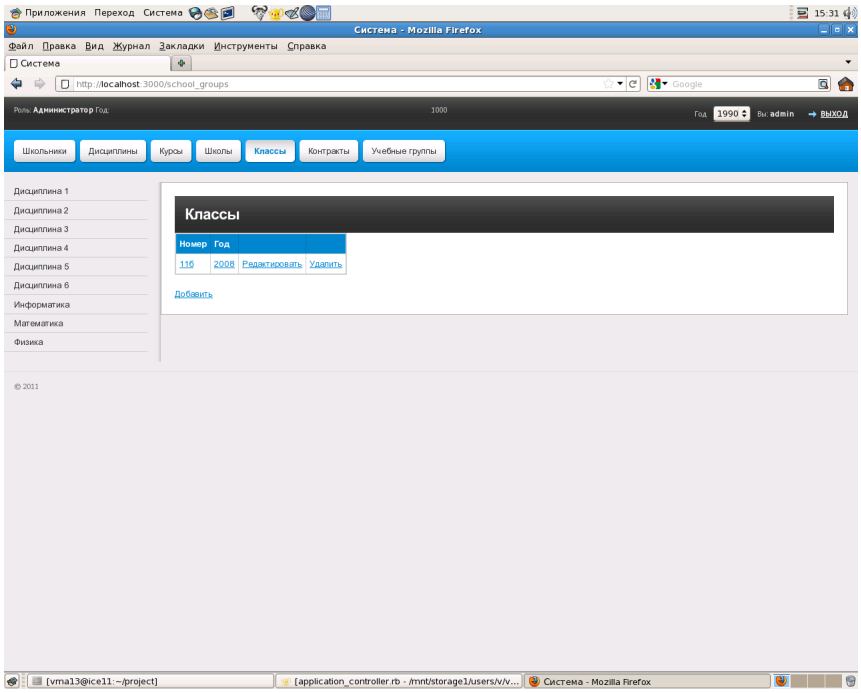

Рис. 3: Таблица школьных классов слушателей ФДО

Далее рассматривается, как будет выглядеть интерфейс просмотра классов, курсов, дисциплин, групп, учеников и школ.

Приведем пример добавления нового учебного класса новый альбом, для этого достаточно перейти по ссылке «Добавить». При нажатии на данную ссылку произойдет

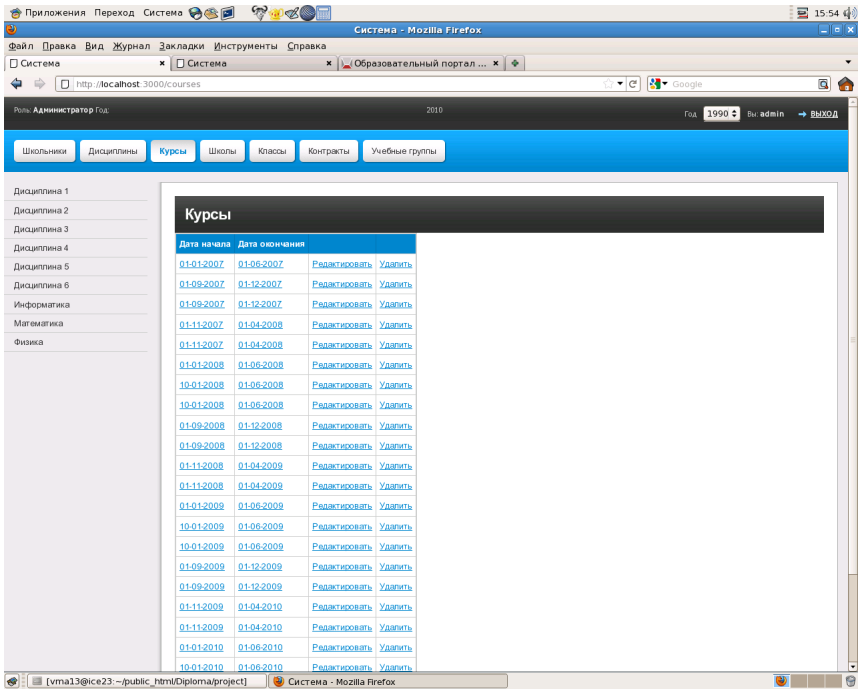

Рис. 4: Таблица курсов слушателей ФДО

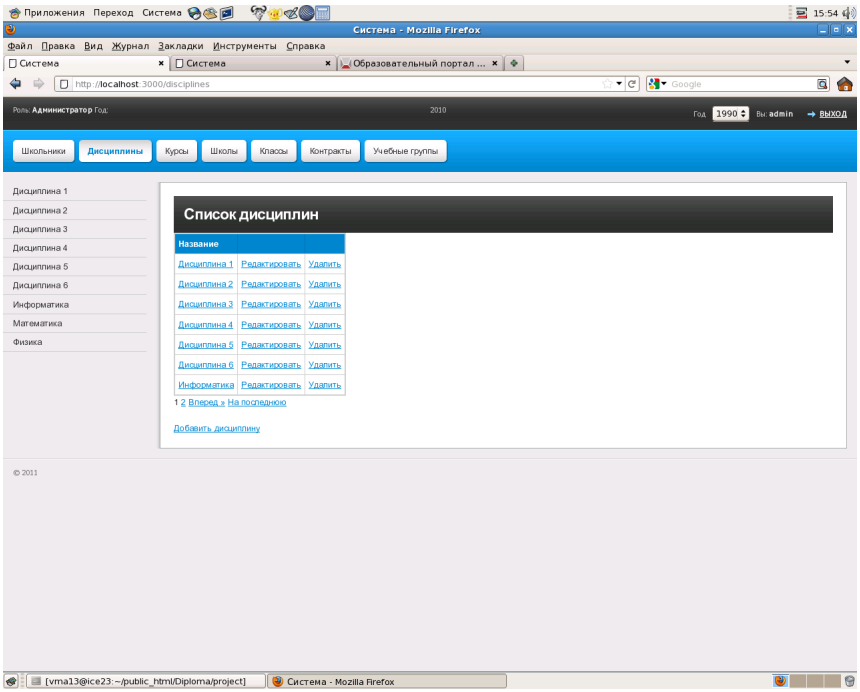

Рис. 5: Таблица дисциплин слушателей ФДО

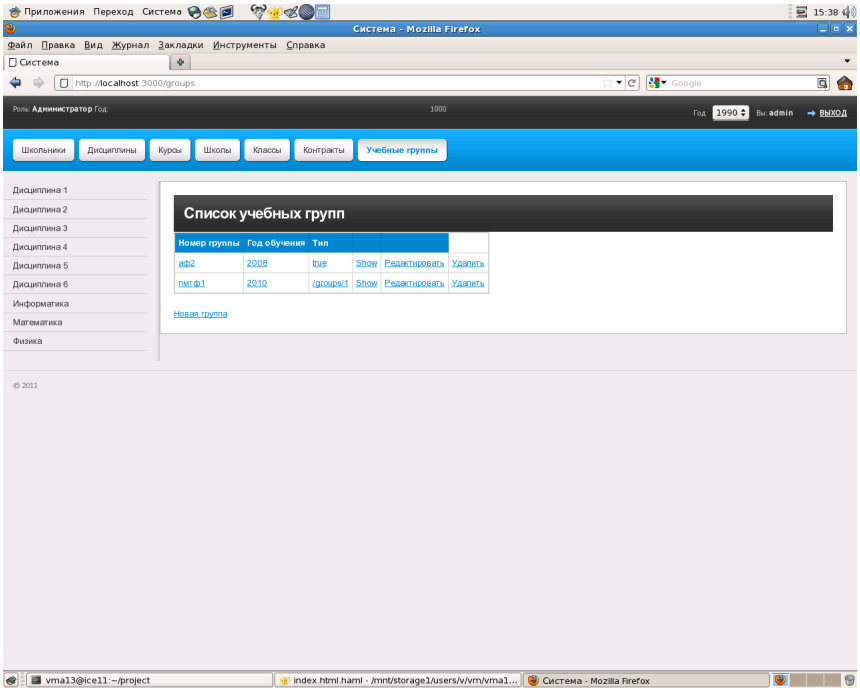

Рис. 6: Таблица групп слушателей ФДО

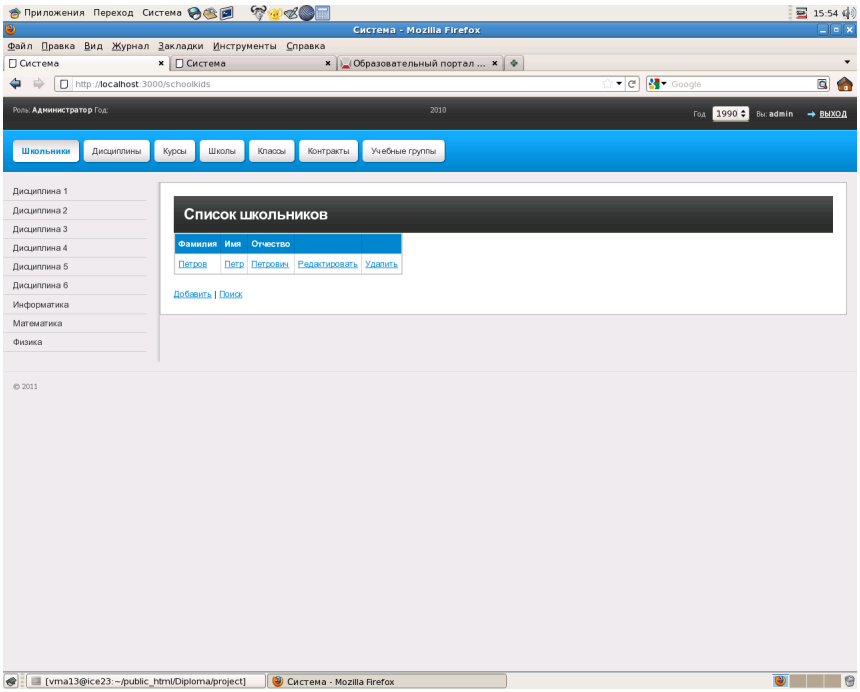

Рис. 7: Таблица учеников слушателей ФДО

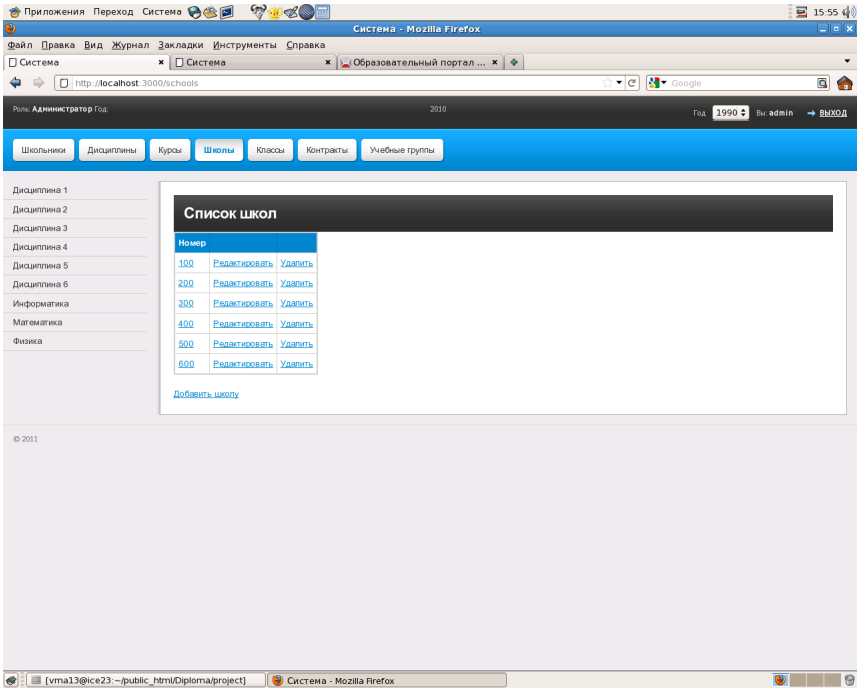

Рис. 8: Таблица школ

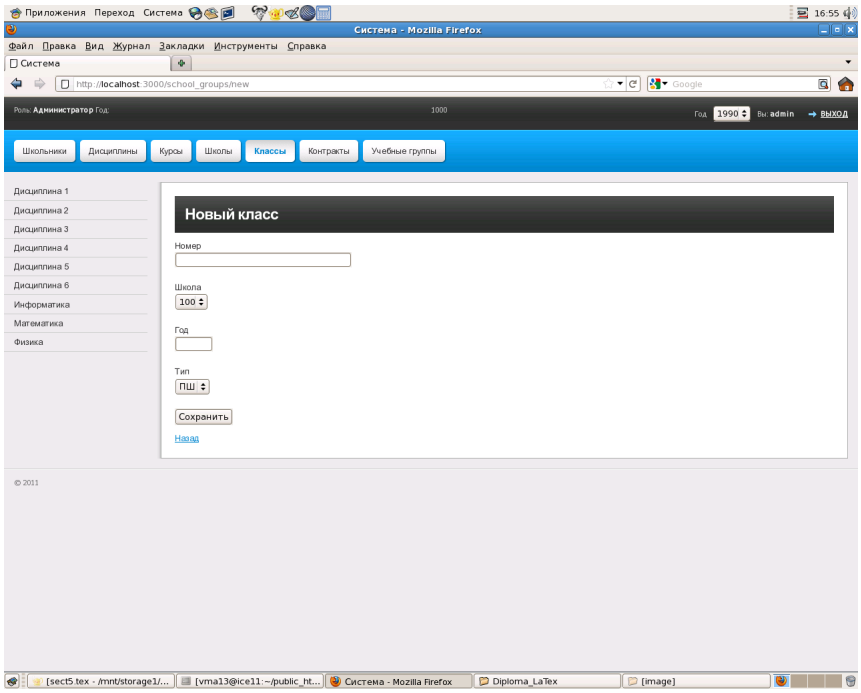

Рис. 9: Добавление учебного класса

перенаправление на страничку создания учебного класса.

Если просто нажать кнопку «Сохранить», то отобразится сообщение об ошибке.

В данном случае продемонстрирован наглядный пример работы валидатора: название класса должно существовать, т.е. не существует класса без названия и это

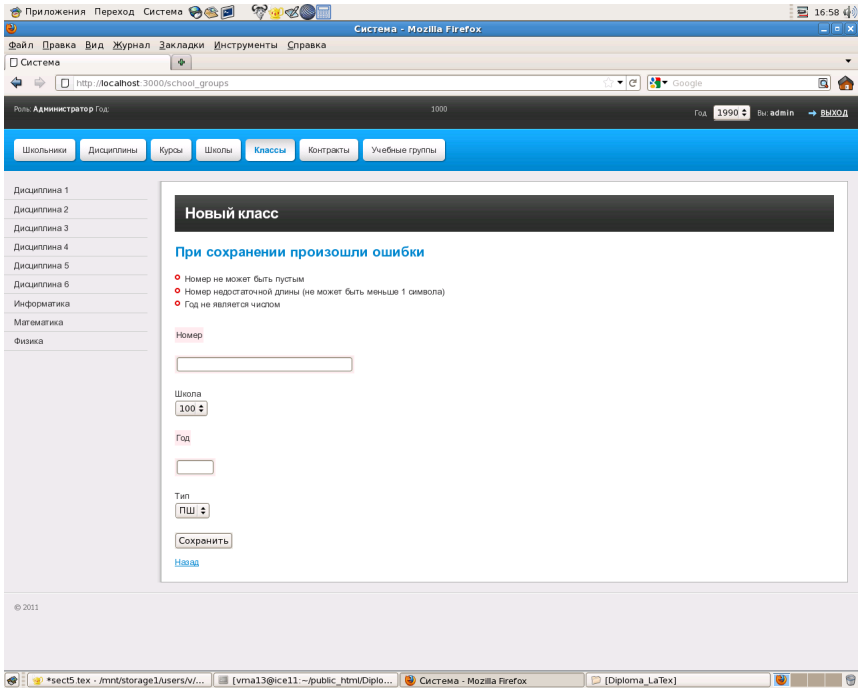

Рис. 10: Ошибка

название должно содержать не менее 1 символа в названии. А также учебный год обязательно должен быть числом.
В остальных классах реализован аналогичный интерфейс, вот иллюстрация его работы, на примере ученика.

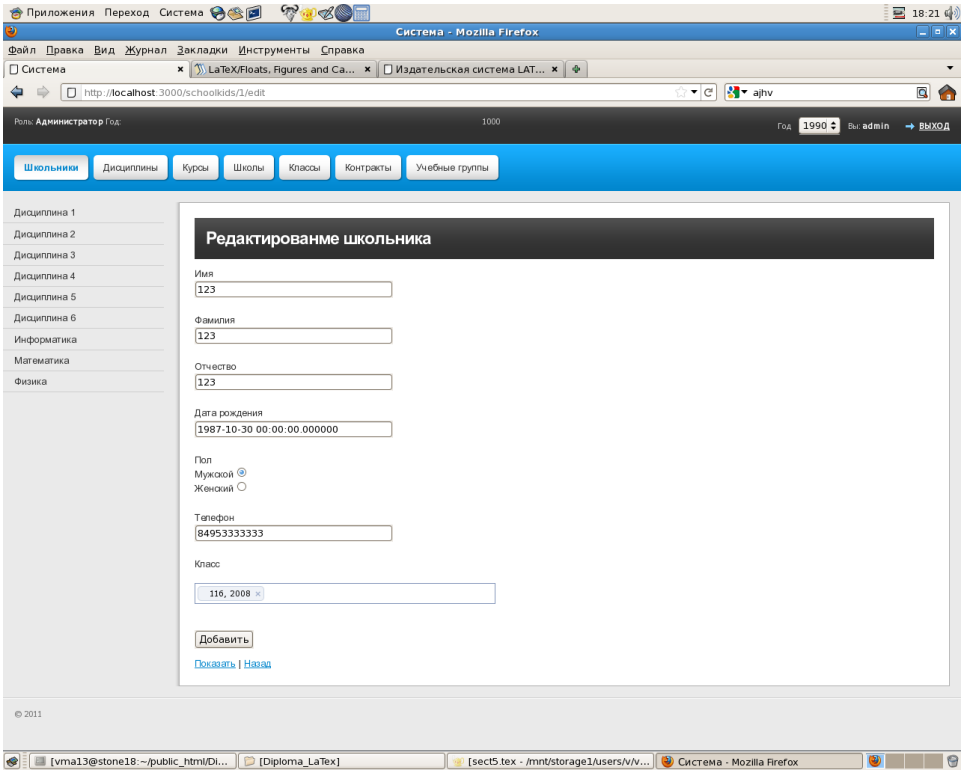

Рис. 11: Редактирование ученика

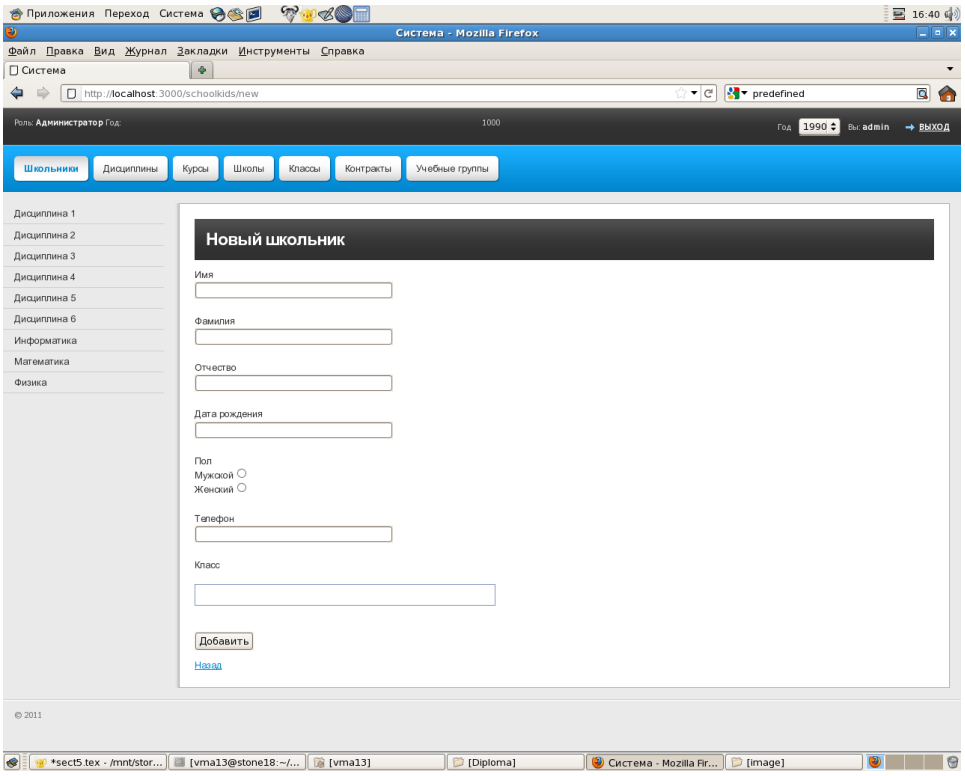

Рис. 12: Создание нового ученика

Список литературы и интернет-ресурсов

[1] С.М. Львовский. *Набор и вёрстка в системе LaTeX, 3-е изд., испр. и доп.* — М., МЦНМО, 2003. Доступны исходные тексты этой книги.

[2] `http://ru.wikipedia.org/wiki/LaTeX` — Википедия (свободная энциклопедия) о системе LaTeX.

[3] `http://www.sbras.ru/win/docs/TeX/LaTex2e/docs_koi.html` — Различная документация по системе LaTeX.

[4] `http://edgeguides.rubyonrails.org/` — Официальный сайт Ruby On Rails.

[5] `http://railscasts.com/` Видео уроки работы с Ruby On Rails.

[6] `http://apidock.com/rails` Электронная документация по Ruby On Rails

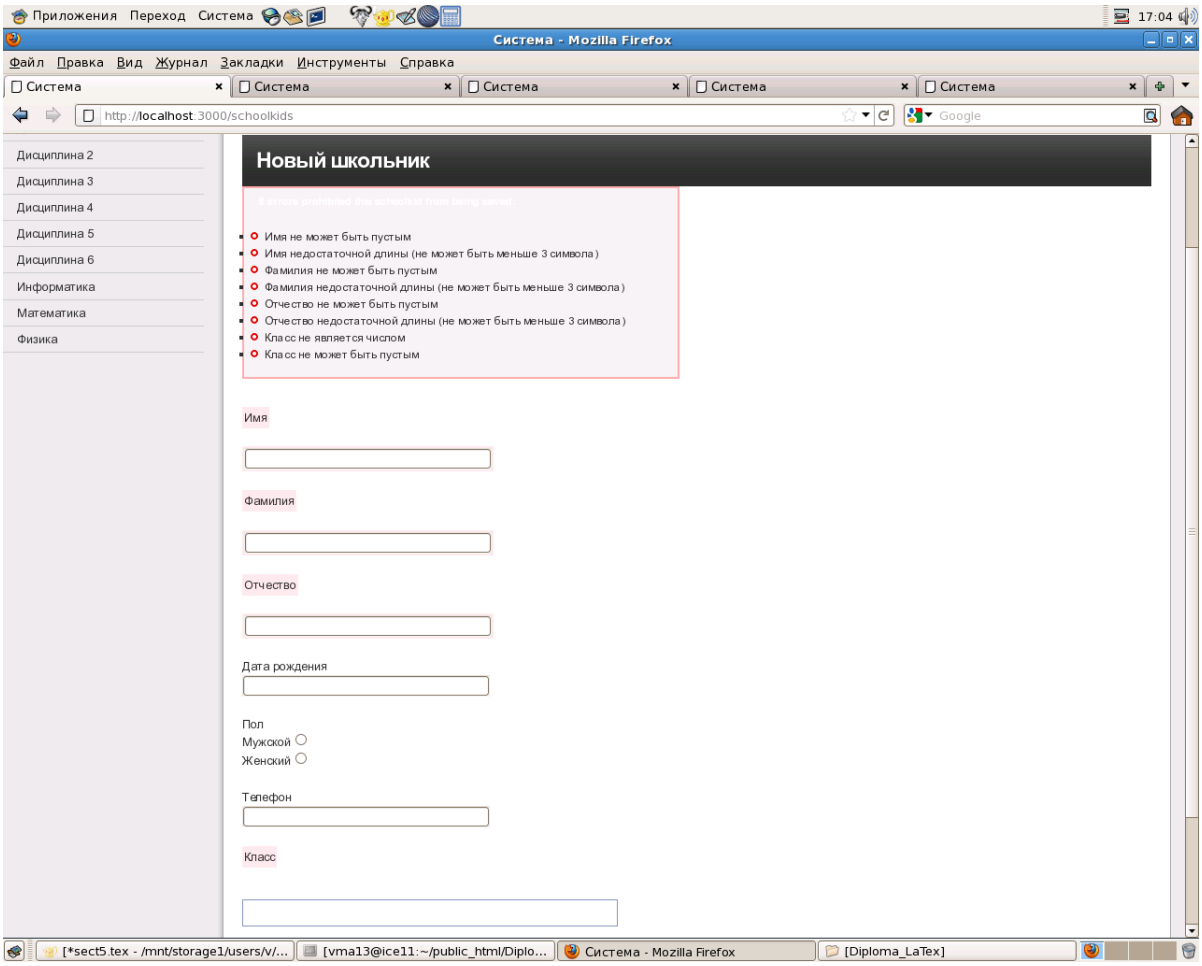

Рис. 13: Ошибки при сохранении ученика