МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ
КАФЕДРА ИНФОРМАЦИОННЫХ СИСТЕМ И ТЕХНОЛОГИЙ

Руководитель работы:
к.ф.-м.н., доцент Е.А. Роганов

Войнов Максим Александрович

# Модернизация эталонного проекта «Видеокаталог» – альбомы и фотографии

Курсовая работа по дисциплине
«Веб-технологии»
3-й курс, 6-й семестр

Москва 2011

## Аннотация

Курсовая работа посвящена модификации эталонного проекта «Видеокаталог», с целью расширения функциональных возможностей. В данной работе описываются внесенные изменения.

# Оглавление

# 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, JavaSvript, Flash, различные СУБД. В данной работе были использованы HTML, СУБД PostgreSQL и платформа *RubyOnRails*.

*RubyOnRails* – 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.

# 2. 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.

# 3. Структура базового проекта

Перед тем как приступать к рассмотрению индивидуального задания 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.

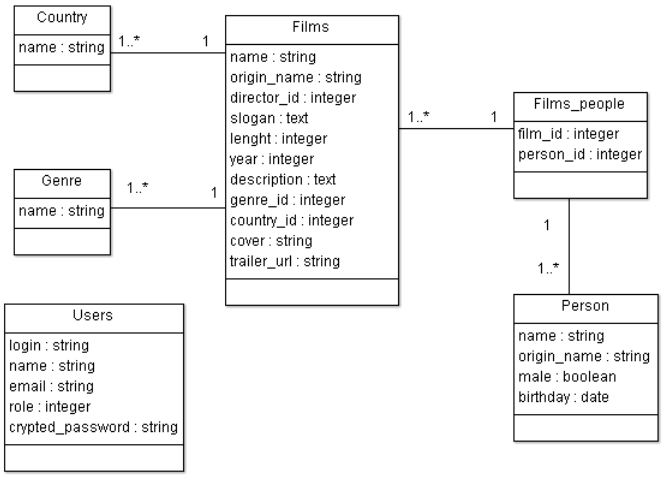


Рис. 1. Диаграмма классов эталонного проекта

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

- Фильм
- Жанр
- Страна
- Персона
- Пользователь

Класс **Фильм** – класс хранилище, которое содержит информацию о фильмах: название, слоган, режиссер, год выпуска и т.д. Он находится в отношении один ко многим с классами **Жанр** и **Страна**, а классом **Персона** в отношении многие ко многим. Класс **Пользователь** не связан отношениями с другими классами, он содержит информацию о пользователях системы: имя, логин, адрес электронной почты, пол, дату рождения и пароль.

# 4. Постановка задачи

Реализовать систему добавления альбомов фотографий персон и кадров из фильма.

С точки зрения конечного пользователя это означает, что в системе должен быть предусмотрен интерфейс для добавления фильму и актеру альбома с фотографиями, отдельный интерфейс для просмотра добавленных фотографий.

# 5. Реализация

## Модель

Для начала определим, какие классы необходимо добавить в систему. Для хранения информации о альбомах добавим новый класс **Album** с атрибутами: имя альбома (string), id альбома (integer), id персоны (integer), id фильма (integer). Однако при этом возникает несколько проблем:

- Поиск по идентификатору происходит намного быстрее, чем поиск на сравнение двух строк.
- Хранить для каждого фильма отдельный альбом в виде строки не оптимально, это лишняя нагрузка на БД.

Избежать всего этого можно, выделив фотографии в отдельный класс. В результате мы получим:

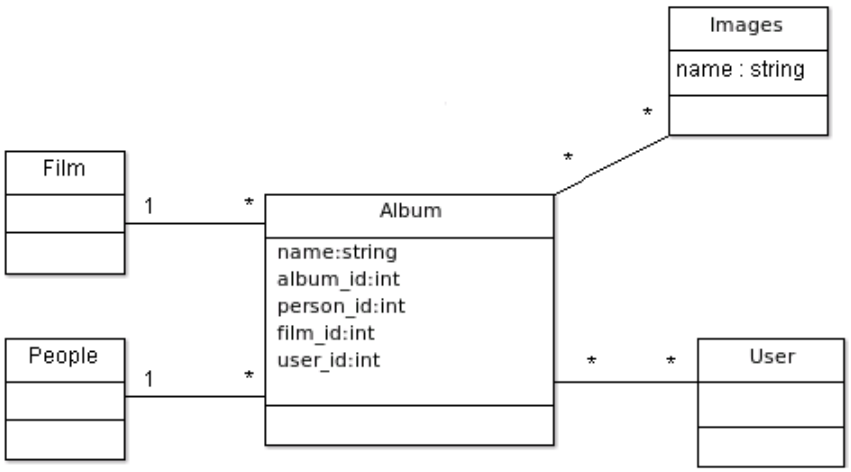


Рис. 2. Диаграмма классов: индивидуальная часть

Из диаграммы видно, что теперь класс **Album** содержит id персон, фильмов и альбомов. Он связан отношением многие ко многим с классом **Image** (фотография),

**Films** (фильмы) и **Persons** (персоны). Класс **Image** хранит информацию о фотографиях: К какому альбому принадлежит фотография(связь через id альбома), какой тип имеет(фоторафия или скриншот). Теперь, 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 Albums name:string user_id:integer person_id film_id
rails g scaffold Image name:string
```

Внесем изменения в модель **Images:** Здесь указана связь многие ко многим с классом **Album** (belongs_to album). Также добавлены ограничения: id альбома должен быть всегда не пустым. А также добавим изменения в главную модель: **Album:**.

```
class Album < ActiveRecord::Base

  validates_with AlbumValidator


  paginates_per 10
  default_scope order(:name)
  belongs_to :film
  belongs_to :person
  belongs_to :user
  has_many :images, :dependent => :destroy
  has_attached_file :cover, :styles => { :medium => "200x200", :thumb => "100x100" },
                    :convert_options => {:thumb => "-gravity center -extent 100x100"}
  validates :name, :presence => true,
                   :uniqueness => true,
                   :length => { :within => 3..40 }
  validates :user_id, :presence => true

  attr_reader :person_tokens

  attr_reader :film_tokens

  def film_tokens=(ids)
    self.film_id = ids
  end

  def person_tokens=(ids)
    self.person_id = ids
  end
end
```

Вначале указаны связи с классами **Image** (has_many images), **Persons** (belongs_to person), **Films** (belongs_to film) и **User** (belngs_to user). Также добавлены несколько проверок на корректность вводимых пользователем данных: имя альбома не может быть пустым, оно уникально и имеет длину от 3 до 40 символов. Также определена интересная валидация, используемая для того чтобы при создании альбома

нельзя было привязать его сразу к персоне и фильму, либо сразу ни к персоне ни к фильму. Далее определены несколько методов, которые понадобятся в дальнейшем. Для удобного просмотра фотографий сделаем удобную навигацию с помощью плагина fancy$_b ox$, .

В моделях классов **Person** и **Films** также необходимо указать связи с классом **Album**.

```
class Person < ActiveRecord::Base
  has_many :albums, :dependent => :destroy

  ...
end


class Film < ActiveRecord::Base
  has_many :albums

  ...
end
```

## 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 альбомы, не нужно создавать отдельный интерфейс, можно сделать это при редактировании персоны или фильма. Для этого просто добавим ссылку на страничку создания альбома. Рассмотрим, как это будет выглядеть для класса **Album**.

```
= form_for @album, :html=>{:multipart=>true} do |f|
  -if @album.errors.any?
    #error_explanation
      %h2= "#{pluralize(@album.errors.count, "error")} prohibited this album from bei
      %ul
        - @album.errors.full_messages.each do |msg|
          %li= msg
  .field
    = f.label :name, ''
    = f.text_field :name
  .field
    =f.label :person_tokens, "
    %br
    =f.text_field :person_tokens,"data-pre"=>@album.person.to_a.map(&:attributes).to_
  .field
    =f.label :film_tokens, ""
    %br
    =f.text_field :film_tokens,"data-pre"=>@album.film.to_a.map(&:attributes).to_json
  .field
```

```
    = f.label :cover, ""
    = f.file_field :cover
  .actions
    = f.submit ''
```

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() {
  $("#album_film_tokens").tokenInput("/films.json", {
    crossDomain: false,
    prePopulate: $("#album_film_tokens").data("pre"),
    theme: 'facebook',
    hintText: ' ',
    noResultsText: ',
    searchingText: ".."
  });
});

$(function() {
  $("#album_person_tokens").tokenInput("/people.json", {
    crossDomain: false,
    prePopulate: $("#album_person_tokens").data("pre"),
    theme: 'facebook',
    hintText: ' ',
    noResultsText: ',
    searchingText: ".."
  });
});
```

Также необходимо добавить создание, отображение, редактирование, удаление, изменени и сохранение нового альбома в контроллере альбома.

```
def index
    @albums = Album.where(:user_id => @current_user.id).page(params[:page])
    respond_to do |format|
      format.html # index.html.erb
      format.xml  { render :xml => @albums }
    end

  end

  def show

    @album = Album.includes(:images).find(params[:id])

    respond_to do |format|
```

```
    format.html # show.html.erb
    format.xml  { render :xml => @album }
  end

end

# GET /albums/new
# GET /albums/new.xml
def new

  @album = Album.new

  respond_to do |format|
    format.html # new.html.erb
    format.xml  { render :xml => @album }
  end

end

# GET /albums/1/edit
def edit
  @album = Album.find(params[:id])
  if @current_user.id != @album.user_id and !@current_user.admin?
    render :text=>"  ", :layout => 'application'
    return
  end
end

# POST /albums
# POST /albums.xml
def create
  @album = Album.new(params[:album])
  @album.user_id = @current_user.id
  if @album.save
    redirect_to(@album, :notice => '')
  else
    render :action => "new"
  end
end

# PUT /albums/1
# PUT /albums/1.xml
def update
  @album = Album.find(params[:id])
  if @album.update_attributes(params[:album])
    redirect_to(@album, :notice => ' .')
  else
    render :action => "edit"
  end
end
```

```
  # DELETE /albums/1
  # DELETE /albums/1.xml
  def destroy
    @album = Album.find(params[:id])
    if !@current_user.admin? and @current_user.id != @album.user_id
      render :text=>"  ", :layout => 'application'
      return
    end
    @album.destroy
    respond_to do |format|
      format.html { redirect_to(album_url) }
      format.xml  { head :ok }
    end
  end
```

Вначале находится персона, у которой *id* равен *params[:id]*, этот параметр передается от формы при ее редактировании. После этого создаётся новый альбом и сохраняется. Если пользователь не ввел никакой информации, то альбом не сохраняется. Если не удалось сохранить альбом (а это возможно только в одном случае – данные, введенные пользователем, не прошли проверку), делается возврат в форму, и отображается сообщение об ошибке пользователю.

Для того чтобы сделать возможным просмотр всех альбомов у персоны или фильма, нужно внести изменения в представление персоны или фильма, а именно в метод *show*.

```
              ...
%h1
-@person.albums.each do |album|
  =render 'albums/album', :album=>album
              ...
```

Аналогичным образом реализуется просмотр альбома у фильма.

```
                ...

    %h1 Альбом Фильма
    -@film.albums.each do |album|
      =render 'albums/album', :album=>album

  =link_to "Новый альбом", new_album_path
  %br
  \|
  =link_to "Редактировать",edit_film_path(@path)
  \|
  =link_to "Удалить",@film,:confirm => "Вы уверены?",:method => :delete
```

# 6. Приложение

## Пример работы

Приведем пример работы тех изменений, которые мы внесли в эталонный проект.

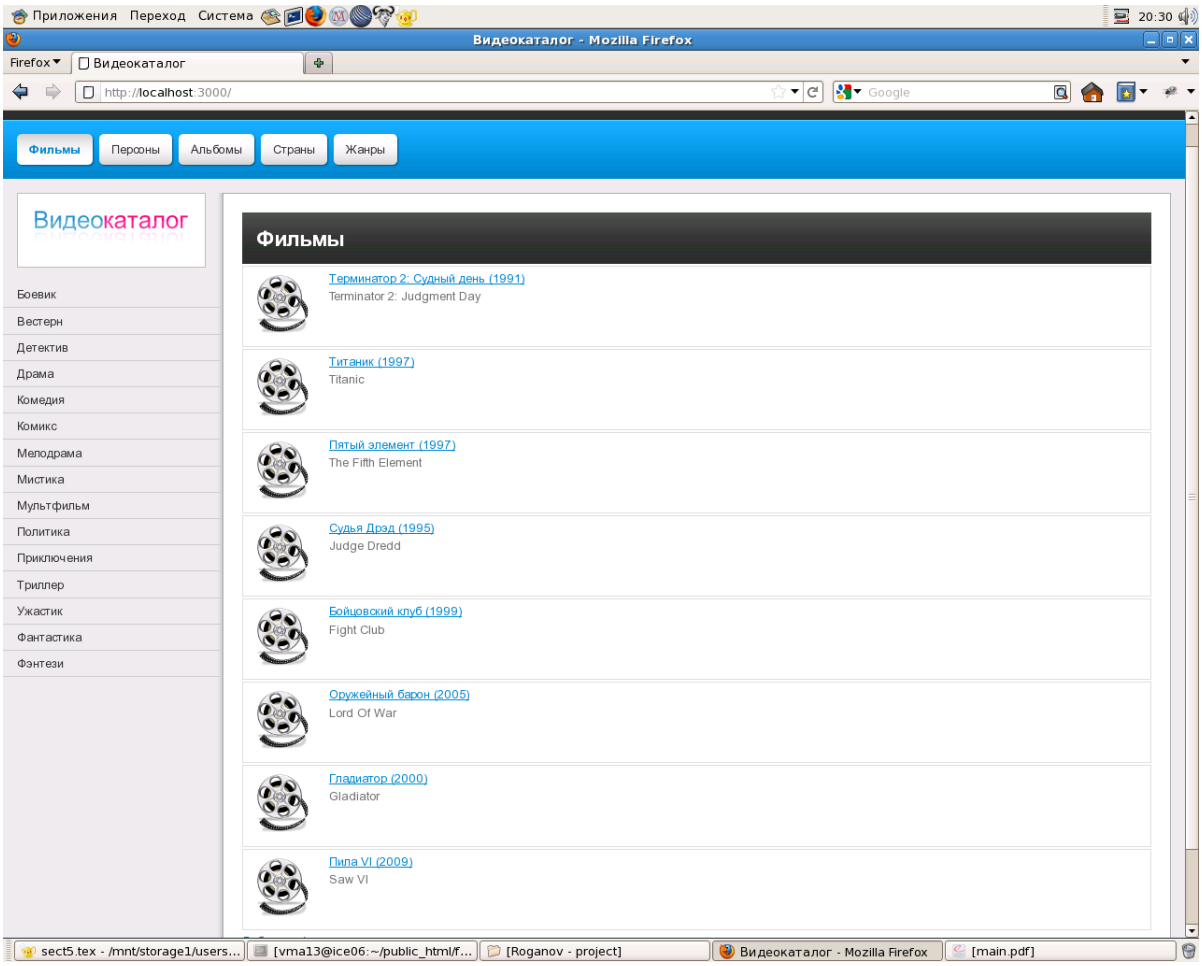


Рис. 3. Домашняя страничка

Далее рассматривается, как будет выглядеть интерфейс просмотра альбома у конкретной персоны и фильма.

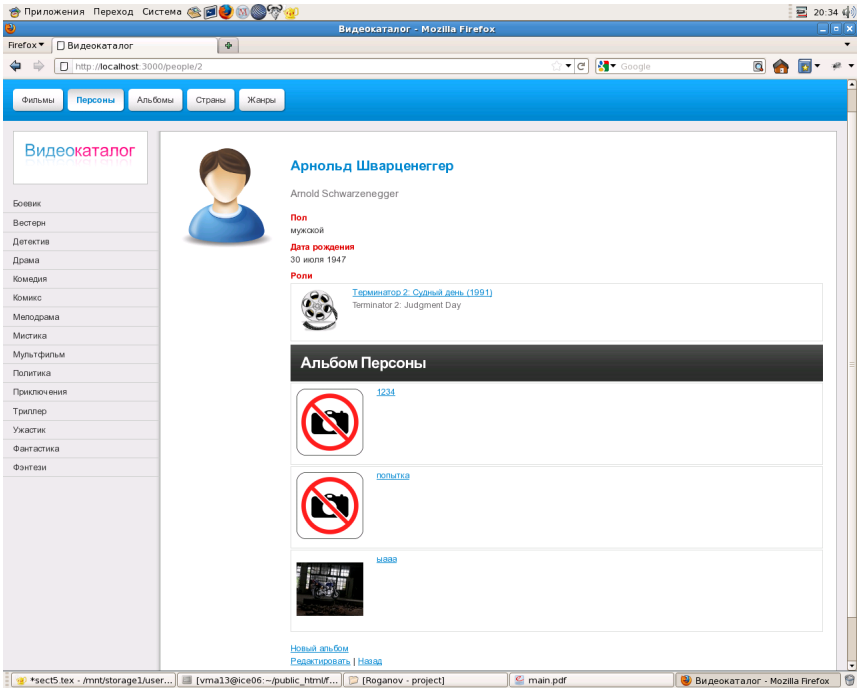


Рис. 4. Альбом у персоны

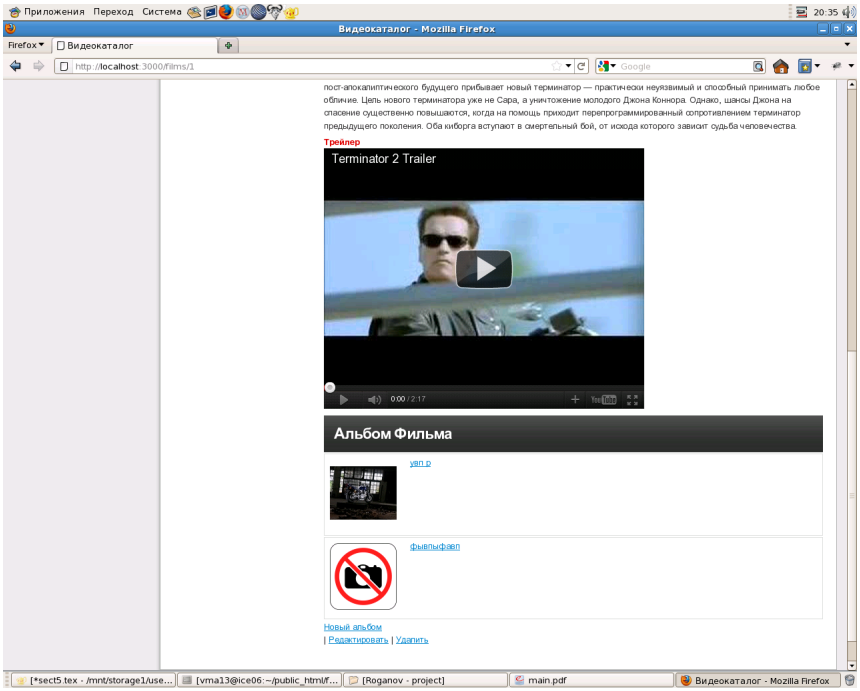


Рис. 5. Альбом у фильма

Для того чтобы добавить новый альбом, достаточно перейти по ссылке "Новый альбом". При нажатии произойдет перенаправление на страничку создания альбома.

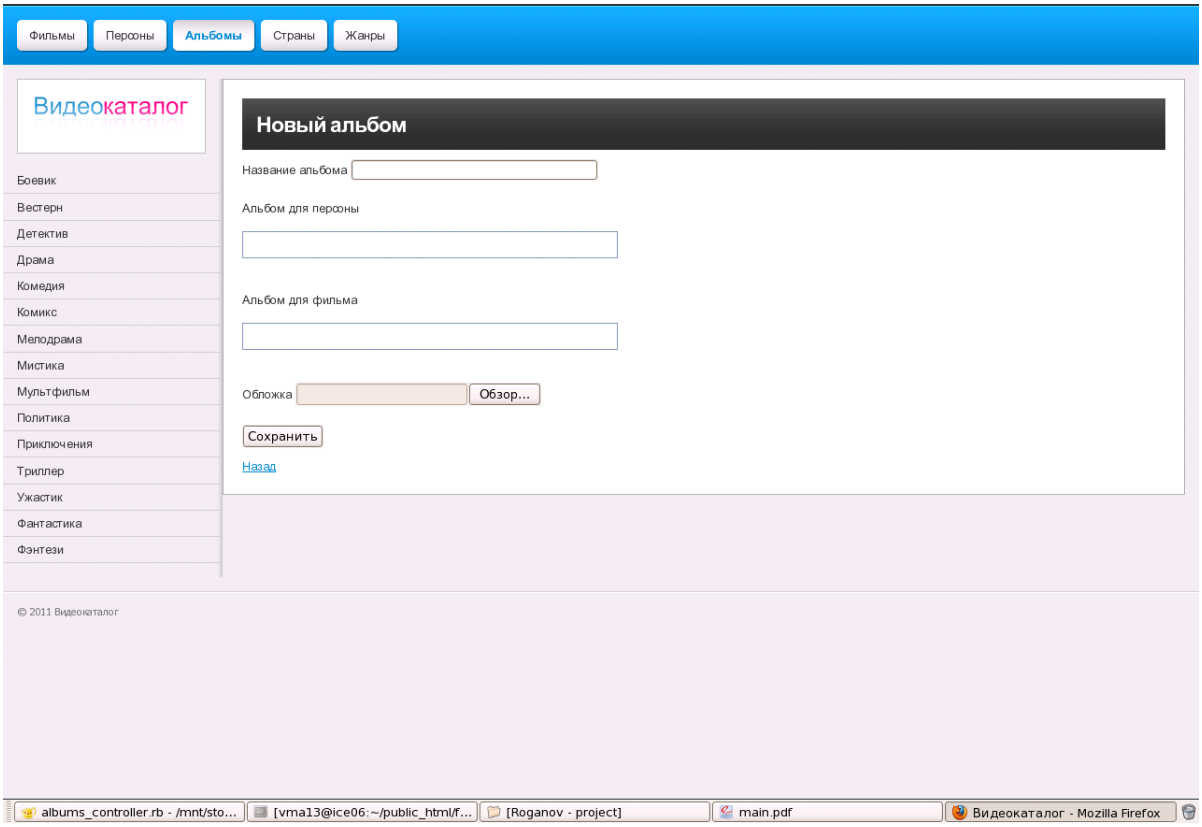


Рис. 6. Добавление Альбома

Если просто нажать кнопку "Сохранить"то отобразится сообщение об ошибке. Альбом должен быть закреплен, либо за персоной, либо за фильмом и никак иначе.

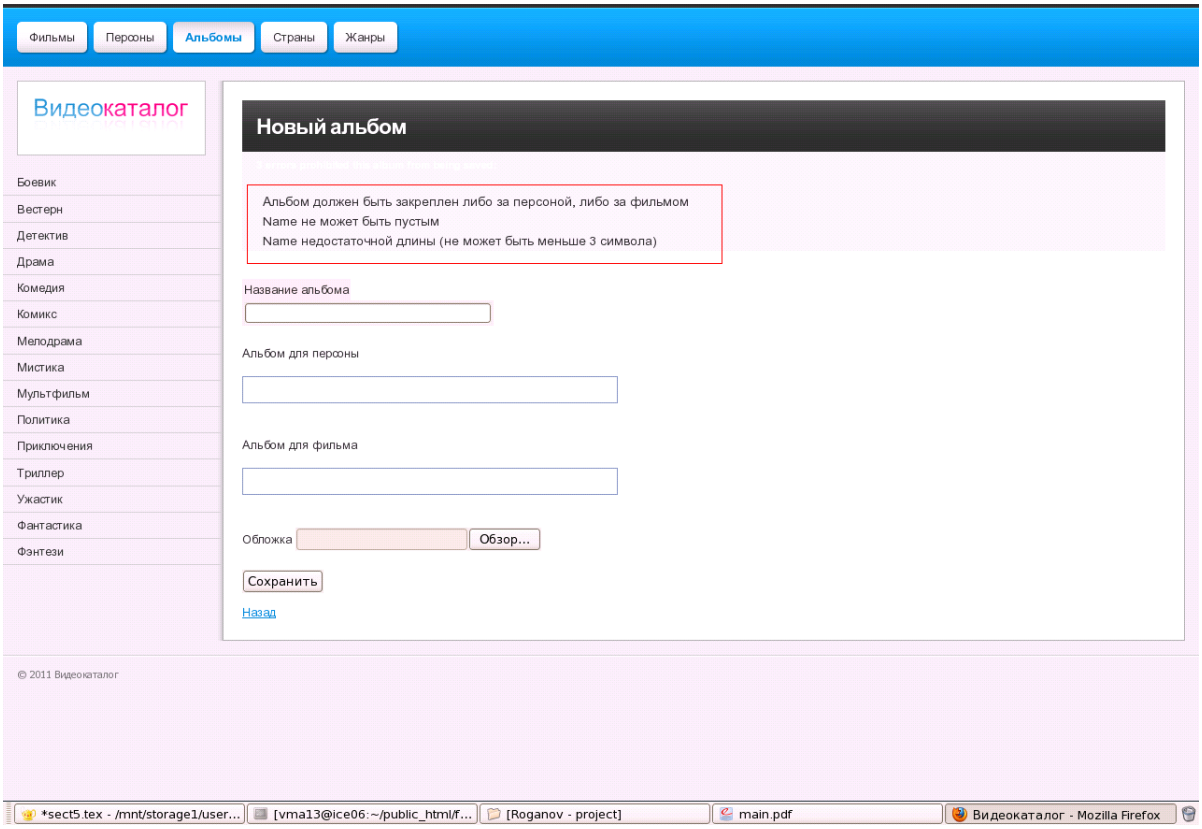


Рис. 7. Ошибка

Также продемонстрирован наглядный пример работы валидатора: название альбома должно существовать и должно содержать от 3 до 40 символов в названии

# 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