**СОДЕРЖАНИЕ**

*у*

Изм.

Лист

№ докум.

Подпись

Дата

Лист

3

КПП41015.403.081ПЗ

Разраб.

Масловский

Пров.

*Дроздова*

Реценз.

Н. контр.

Утв.

Сайт «Electronic Scheme»

Лит.

Листов

58

МКП

[Введение 5](#_Toc470464996)

[1 Постановка задачи 7](#_Toc470464997)

[1.1 Формулировка задания 7](#_Toc470464998)

[1.2 Обоснование необходимости автоматизации 8](#_Toc470464999)

[1.3 Определение данных и их представление 9](#_Toc470465000)

[2 Вычислительная система 10](#_Toc470465001)

[2.1 Обоснование выбора языка программирования 10](#_Toc470465002)

[2.2 Обоснование выбора среды разработки 11](#_Toc470465003)

[2.3 Требования к конфигурации программного и аппаратного обеспечения 12](#_Toc470465004)

[3 Проектирование 13](#_Toc470465005)

[3.1 Проектирование интерфейса 13](#_Toc470465006)

[3.2 Инфологическая модель предметной области 13](#_Toc470465007)

[3.3 Логическая модель данных 15](#_Toc470465008)

[3.4 Физическая модель базы данных 18](#_Toc470465009)

[3.5 Описание справочной системы 21](#_Toc470465010)

[4 Описание программы 22](#_Toc470465011)

[4.1 Логическая структура 22](#_Toc470465012)

[4.2 Физическая структура 24](#_Toc470465013)

[4.3 Особенности интерфейса 24](#_Toc470465014)

[5 Испытание и отладка программы 31](#_Toc470465015)

[5.1 Тестовые примеры 31](#_Toc470465016)

[5.2 Анализ полученных результатов 31](#_Toc470465017)

Изм.

Лист

№ докум.

Подпись

Дата

КПП41015.403.081ПЗ

Лист

4

[6 Описание применения программы 34](#_Toc470465018)

[6.1 Назначение программы 34](#_Toc470465019)

[6.2 Условия применения 34](#_Toc470465020)

[Заключение 35](#_Toc470465021)

[Список использованных источников 36](#_Toc470465022)

[Приложение 1 37](#_Toc470465023)

# Введение

В данном курсовом проекте необходимо разработать сайт «Electronic Scheme».

Сайт — это совокупность логически связанных между собой веб-страниц. Обычно сайт в интернете представляет собой массив связанных данных, имеющий уникальный адрес и воспринимаемый пользователем как единое целое.

В настоящее время существует много различных аналогов, например, EasyEDA, Circuit Sims, EveryCircuit и т.п. Данный курсовой проект отличается от других тем, что каждая схема находятся в открытом доступе и любой пользователь может просмотреть и прокомментировать ее. Поэтому данный проект похож больше на такие сайты как Habrahabr или Пикабу, только вместо обычных постов можно добавить электронную схему.

Основной целью выполнения курсового проекта является расширение, углубление знаний в области алгоритмизации и программирования на языке C# и в среде разработки Visual Studio 2015, расширение знаний о ASP .Net MVC и разработке сайтов, формирование навыков научно-исследовательской деятельности.

Задачи выполнения курсового проекта:

* систематизация научных знаний в области алгоритмизации и программирования на языке C#;
* изучение технологии ASP .Net MVC;
* углубление уровня и расширение объема профессионально значимых знаний, умений и навыков работы в среде быстрой разработки Visual Studio;
* формирование умений и навыков самостоятельной организации научно-исследовательской работы;
* овладение современными методами поиска, обработки и использования информации с помощью баз данных.

Данная пояснительная записка содержит следующие разделы:

1. Раздел «Постановка задачи» содержит формулировку задачи с указанием всех функций проекта, определение круга задач, которые должны быть автоматизированы в данном курсовом проекте, описание входных и выходных данных проекта.
2. Раздел «Вычислительная система» содержит обоснование выбора языка программирования и среды разработки проекта, описание операционной системы, в которой была реализована программа. Требования конфигурации программного и аппаратного обеспечения.
3. Раздел «Проектирование» содержит описание внешнего пользовательского интерфейса, структуру меню, алгоритма, функционально независимых подзадач и путей их решения.
4. В разделе «Описание программы» содержится описание всех модулей, функций, входящих в проект, их назначение и взаимодействие, особенности интерфейса программы.
5. В разделе «Отладка и испытание программы» описывается проверка работы программы на различных тестах и анализ полученных результатов.
6. Раздел «Описание и применение программы» содержит сведения о назначении программного средства, области применения, требования к необходимым программным и техническим ресурсам.
7. В «Заключении» описывается краткая формулировка проблемы и пути ее решения, использованные методы и средства, возможность дальнейшей модификации проекта.

# 1 Постановка задачи

## 1.1 Формулировка задания

Необходимо разработать сайт «Electronic Scheme». Реализовать возможность создавать и просматривать электронные схемы с описанием.

В проекте реализовать возможность регистрации и авторизации. Так же должна быть возможность авторизоваться через социальные сети (Вконтакте, Facebook, Tweeter).

После авторизации пользователю присваивается роль: «User» или «Admin».

Неавторизированный пользователь должен иметь возможность только просматривать, осуществлять поиск и фильтрацию контента.

User может делать все что и неавторизированный пользователь, а так же возможность: добавлять, удалять и редактировать свои посты (содержащие разделы: «Название», «Теги», «Категория», «Схема», «Описание»), комментировать и оценивать свои и чужие посты. Каждый user имеет свою личную страницу, где он может поменять свое фото и увидеть все свои посты.

Admin имеет те же возможности, что и user, плюс к этому он может просмотреть список пользователей и лишить прав пользователя, оставив возможность только просматривать контент или дать права администратора. Так же admin может удалять и редактировать любые посты.

На сайте должна быть возможность поиска контента по категориям, тегам или по введенному тексту и возможность фильтрации контента по дате.

## 1.2 Обоснование необходимости автоматизации

Электронная схема — это сочетание отдельных электронных компонентов, таких как резисторы, конденсаторы, индуктивности, диоды, транзисторы и интегральные микросхемы, соединённых между собой.

Электронные схемы могут изображать разными способами, но наиболее распространены следующие:

1. Изображение электронной схемы на бумаге. Плюсы данного подхода – это доступность и простота. Минусы – можно неточно изобразить какой-либо компонент, бумага не надежный источник хранения данных и информацию в таков виде трудно быстро передать в другое место (как минимум нужно отсканировать или сфотографировать).
2. Изображение электронной схемы на компьютерной или мобильной программе. Плюсы данного подходы – это быстрота создания схемы так как основные компоненты уже встроены в программу и все что нужно это принести их на рабочую область. Минусы – данные программы нужно предварительно установить на устройство и большинство таких программ не поддерживают возможность экспорта электронной схемы в разных форматах.
3. Изображение электронной схемы на специализированном сайте. Данный подход лучше предыдущего, так как что бы пользоваться сайтом достаточно иметь две вещи: доступ в интернет и браузер. Что бы передать электронную схему, достаточно скопировать и отослать ссылку на нее. Единственный минус данного подхода — это то что нужен доступ к интернету.

Существует множество аналог данного программного обеспечения, такие как:

* EasyEDA – сайт имеет много возможностей для изображения электронных схем, экспорта и конвертации в различные форматы. Данный сайт создан не для новичков, так как интерфейс будет понятен только опытным инженерам;
* Circuit Sims – сайт не имеет возможности экспорта электронных схем, только их изображения.
* EveryCircuit – хороший аналог с возможностью проверки электронных схем, но он не бесплатный в использовании.

Данный проект отличается от аналогов: интуитивно понятным интерфейсом, возможностью добавлять описания к схеме и просматривать другие схемы. Тем самым можно быстро найти нужную схему или похожую, и основываясь на ней сделать схему для себя – это уменьшит время на проектирования электронной схемы, тем самым с экономит время.

## 1.3 Определение данных и их представление

Форматы входных и выходных данных являются частью этапа проектирования. Входные форматы должны быть разработаны с учетом максимального удобства для пользователя и минимальной возможности ошибок. Порядок переменных и форматы данных, привычные для пользователя, помогут избежать ошибок и облегчат использование программ. В данном проекте входными данными являются: курсор мыши, настройки поиска, данные авторизации, электронные компоненты и текст.

Выходные спецификации могут сильно различаться. Иногда даются четкие инструкции и выходные данные подгоняются под определенный стандарт. Однако часто вообще отсутствуют какие-либо указания. Выходные данные подчас представляют собой: электронная схема и описанием.

# 2 Вычислительная система

## 2.1 Обоснование выбора языка программирования

В качестве языка программирования, выбранного для разрабатываемого проекта выбран язык программирования C#. Преимуществами данного языка являются:

1. C# создавался параллельно с каркасом Framework .Net и в полной мере учитывает все его возможности как FCL, так и CLR.
2. С# является полностью объектно-ориентированным языком, где даже типы, встроенные в язык, представлены классами.
3. C# является мощным объектным языком с возможностями наследования и универсализации.
4. C# является наследником языков C/C++, сохраняя лучшие черты этих популярных языков программирования: общий с этими языками синтаксис, знакомые операторы языка облегчают переход программистов от С++ к C#.
5. Благодаря каркасу Framework .Net, ставшему надстройкой над операционной системой, программисты C# получают те же преимущества работы с виртуальной машиной, что и программисты Java. Эффективность кода даже повышается, поскольку исполнительная среда CLR представляет собой компилятор промежуточного языка, в то время как виртуальная Java-машина является интерпретатором байт-кода.
6. При помощи библиотеки EF Framework, осуществляется быстрый и удобный способ доступа к базе дынных.

Так использовался язык программирования баз данных SQL. SQL – формальный непроцедурный язык программирования, применяемый для создания, модификации и управления данными в произвольной реляционной базе данных, управляемой соответствующей системой управления базами данных (СУБД). SQL является прежде всего информационно-логическим языком, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных. Основные функции SQL: создание в базе данных новой таблицы; добавление в таблицу новых записей; изменение записей; удаление записей; выборка записей из одной или нескольких таблиц.

## 2.2 Обоснование выбора среды разработки

Проект был реализован в среде быстрой разработки Visual Studio 2015.

Visual Studio – линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

Система управления базами данных (СУБД) − совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Все современные СУБД имеют:

* визуальные средства создание таблиц, форм, отчетов;
* доступ к данным серверных СУБД;
* создание приложений, выполняющих функции WEB – браузеров и размещение данных в Internet;
* объектно-ориентированный язык;
* поддержку SQL.

MySQL – свободная реляционная система управления базами данных. MySQL имеет двойное лицензирование. MySQL может распространяться в соответствии с условиями лицензии GPL. Однако по условиям GPL, если какая-либо программа включает исходные коды MySQL, то она тоже должна распространяться по лицензии GPL. Это может расходиться с планами разработчиков, не желающих открывать исходные тексты своих программ. Для таких случаев предусмотрена коммерческая лицензия, которая также обеспечивает качественную сервисную поддержку.

## 2.3 Требования к конфигурации программного и аппаратного обеспечения

Данный проект разработан для ОС Windows 7. Достоинства:

1) Данную ОС устанавливают на большей массе современных ноутбуков, нетбуков и стационарных компьютеров, поэтому намного проще выполнить восстановление windows-системы после вирусов.

2) Высокая стабильность работы.

3) Интерфейс является крайне удобным. Интерфейс пользователя «семерки» в сравнении с Windows XP и Vista отличается яркостью красок и стилями оформления.

5) Перечень настроек Windows 7 весьма богат – начиная от графики и заканчивая правами доступа и настройкой безопасности.

6) Данная ОС совместима с современными распространенными программами.

Минимальные системные требования:

* ОС: Windows 7/8/10;
* процессор: Intel Core i3-3110M 2.40GHz;
* оперативная память: 2 ГБ (рекомендуется 4 ГБ).

# 3 ПРОЕКТИРОВАНИЕ

## 3.1 Проектирование интерфейса

Сайт должен состоять из 5 основных страниц:

1. Страница «Регистрации», содержит поля для ввода электронной почты и пароля, и кнопкой «зарегистрироваться».
2. Страница «Авторизации», содержит поля для ввода электронной почты и пароля, и кнопкой «войти», так же на странице есть 3 кнопки для входа на сайт через социальные сети (Вконтке, Tweeter, Facebook).
3. Главная страница, на которой расположены посты (название, автор, дата и время создания и часть описания). Слева располагаются категории схем, справа теги с текстовым полем для их поиска. Сверху страницы располагается текстовое поле для поиска постов.
4. Страница просмотра поста, в начале располагается электронная схема, над ней заголовок, автор, категория, теги и дата создания. Под электронной схемой описания и рейтинг. В самом низу располагаются комментарии.
5. Страница пользователя, слева-сверху располагается фото пользователя справа от него медальки пользователя, а снизу его посты.

## 3.2 Инфологическая модель предметной области

Электронные схемы сильно распространены инженерии микросхем. Гораздо удобнее нарисовать электронную схему в начале, а после сделать макет по схеме. Схемы можно хранить в базе данных – это обеспечит удобный поиск схем, что упростит последующее создания электронной схемы.

Проанализируем объекты реального мира. Для формирования концептуальной модели необходимо провести идентификацию объектов сущности базы данных.

Необходимо спроектировать базу данных электронных схем-сайт, информация которой будет использоваться для хранения электронных схем пользователей.

При проектировании БД необходимо учитывать следующее:

* к одному посту пользователь может добавить только одну схему;
* к одному посту можно добавить неограниченное количество тегов и комментариев;
* пользователь может просмотреть все свои посты и посты других пользователей;
* предусмотреть механизм ролей («Admin», «User»);
* у каждого поста должна быть категория из фиксированного списка;
* каждый пост имеет положительную и отрицательную отметку;
* каждая электронная схема содержит неограниченное количество компонентов и проводов;
* должна иметься авторизация и регистрация.

Таким образом база данных содержит следующие сущности как Пользователи, Посты, Теги, Категории, Комментарии, Медальки, Компоненты, Провода, Отметки.

Сущность ПОСТЫ имеет связь с такими сущностями как КАТЕГОРИИ, ПОЛЬЗОВАТЕЛИ, ПРОВОДА, ТЕГИ, КОМПОНЕНТЫ, КОМЕНТАРИИ, ОТМЕТКИ.

Связь между сущностями ПОСТЫ и ПОЛЬЗОВАТЕЛИ, связывает пост с ее автором. Данная связь позволяет связывать с одним пользователем более одного поста, причем каждый пост должен быть связан с каким-либо пользователем. Также данная связь показывает, что каждый пост может принадлежать только одному пользователю.

Связь между сущностями ПОСТЫ и КАТЕГОРИИ, связывает пост с её категорией. Данная связь позволяет связывать с одним постом одну категорию, причем каждый пост должен быть связан с какой-либо категорией.

Связь между сущностями ПОСТЫ и ПРОВОДА (КОМПОНЕНТЫ, КОМЕНТАРИИ, ТЕГИ, ОТМЕТКИ) связывает пост с «проводами» (компонентами, тегами, отметками, комментариями). Данная связь позволяет связывать с одним постом более одного «провода» (компонента, тега, отметкой, комментарием).

Сущность ПОЛЬЗОВАТЕЛИ имеет связь с сущностью МЕДАЛЬКИ, данная связь позволят связать с пользователем более одной медальки. Так же данная связь позволяет пользователю не имеет медалек вообще.

## 3.3 Логическая модель данных

Логическая модель базы данных представлена следующими таблицами:

Таблица 1 − Категории

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| Номер категории | Числовой | Уникален для категории. |
| Название | Текстовый | Название категории. |

Таблица 2 − Провода

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| Номер провода | Числовой | Уникален для провода. |
| Номер поста | Числовой | Номер связанного поста. |
| X1 | Числовой | Координата X первой точки |
| X2 | Числовой | Координата X второй точки |
| Y1 | Числовой | Координата Y первой точки |
| Y2 | Числовой | Координата Y второй точки провода. |

Таблица 3 − Пользователи

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| Номер Пользователя | Числовой | Уникален для каждого пользователя магазина. |
| Имя | Текстовый | Имя пользователя. |
| Email | Текстовый | Email пользователя. |
| Логин | Текстовый | Логин пользователя. |
| Пароль | Текстовый | Пароль пользователя. |

Таблица 4 − Теги

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| Номер тега | Числовой | Уникален для каждого тега. |
| Номер поста | Числовой | Номер связанного поста. |
| Значение | Текстовый | Значение тега. |

Таблица 5 − Комментарии

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| Номер комментария | Числовой | Уникален для каждого комментария. |
| Номер поста | Числовой | Номер связанного поста. |
| Номер пользователя | Числовой | Номер пользователя, написавшего комментарий. |
| Текст | Текстовый | Текст комментария. |
| Дата | Дата | Дата создания комментария. |

Таблица 6 − Медальки

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| Номер медальки | Числовой | Уникален для каждой медальки. |
| Изображение | Текстовый | Ссылка на картинку медальки. |
| Номер пользователя | Числовой | Номер пользователя которому принадлежит медалька. |
| Описание | Текстовый | Описание медальки. |

Таблица 7 − Компоненты

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| Номер компонента | Числовой | Уникален для каждого компонента. |
| Номер поста | Числовой | Номер связанного поста. |
| X | Числовой | Позиция компонента по оси Х. |
| Y | Числовой | Позиция компонента по оси Y. |
| Название | Текстовый | Название компонента. |
| Изображение | Текстовый | Ссылка на картинку компонента. |
| Угол | Числовой | Угол поворота компонента. |
| Ширина | Числовой | Ширина компонента. |
| Высота | Числовой | Высота компонента. |

Таблица 8 − Отметки

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| Номер отметки | Числовой | Уникален для каждого отметки. |
| Номер поста | Числовой | Номер связанного поста. |
| Номер пользователя | Числовой | Номер пользователя. |
| Номер комментария | Числовой | Номер комментария. |
| Положительный | Логический | Положительная ли отметка. |

Таблица 9 − Посты

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| Номер поста | Числовой | Уникален для каждого поста. |
| Номер категории | Числовой | Номер категории |
| Номер пользователя | Числовой | Номер пользователя. |
| Дата | Дата | Дата публикации поста. |
| Заголовок | Текстовый | Заголовок поста. |
| Описание | Текстовый | Текстовое описание поста. |

## 3.4 Физическая модель базы данных

Физическая модель базы данных представлена следующими таблицами:

Таблица 1 − Категории

|  |  |
| --- | --- |
| Поле | Тип данных |
| Номер категории | INT (PRIMARY KEY) |
| Название | varchar |

Таблица 2 − Провода

|  |  |
| --- | --- |
| Поле | Тип данных |
| Номер провода | INT (PRIMARY KEY) |
| Номер поста | INT (PRIMARY KEY) |
| X1 | INT |
| X2 | INT |
| Y1 | INT |
| Y2 | INT |

Таблица 3 − Пользователи

|  |  |
| --- | --- |
| Поле | Тип данных |
| Номер Пользователя | INT (PRIMARY KEY) |
| Имя | varchar |
| Email | varchar |
| Логин | varchar |
| Пароль | varchar |

Таблица 4 − Теги

|  |  |
| --- | --- |
| Поле | Тип данных |
| Номер тега | INT (PRIMARY KEY) |
| Номер поста | INT (PRIMARY KEY) |
| Значение | varchar |

Таблица 5 − Комментарии

|  |  |
| --- | --- |
| Поле | Тип данных |
| Номер комментария | INT (PRIMARY KEY) |
| Номер поста | INT (PRIMARY KEY) |
| Номер пользователя | INT (PRIMARY KEY) |
| Текст | varchar |
| Дата | Date |

Таблица 6 − Медальки

|  |  |
| --- | --- |
| Поле | Тип данных |
| Номер медальки | INT (PRIMARY KEY) |
| Изображение | varchar |
| Номер пользователя | INT (PRIMARY KEY) |
| Описание | varchar |

Таблица 7 − Компоненты

|  |  |
| --- | --- |
| Поле | Тип данных |
| Номер компонента | INT (PRIMARY KEY) |
| Номер поста | INT (PRIMARY KEY) |
| X | INT |
| Y | INT |
| Название | varchar |
| Изображение | varchar |
| Угол | INT |
| Ширина | INT |
| Высота | INT |

Таблица 8 − Отметки

|  |  |
| --- | --- |
| Поле | Тип данных |
| Номер отметки | INT (PRIMARY KEY) |
| Номер поста | INT (PRIMARY KEY) |
| Номер пользователя | INT (PRIMARY KEY) |
| Номер комментария | INT (PRIMARY KEY) |
| Положительный | Boolean |

Таблица 9 − Посты

|  |  |
| --- | --- |
| Поле | Тип данных |
| Номер поста | INT (PRIMARY KEY) |
| Номер категории | INT (PRIMARY KEY) |
| Номер пользователя | INT (PRIMARY KEY) |
| Дата | Date |
| Заголовок | varchar |
| Описание | varchar |

## 

## 3.5 Описание справочной системы

Справочная система предназначена для получения пользователем максимально точной (релевантной) информации по интересующей его/её (и ограниченной базой статей) теме. Обычно выбор статьи происходит по иерархии разделов справки. Справочные системы часто комбинируются с поисковыми, где выборка релевантных статей определяется по заданным ключевым словам или (при полнотекстовом поиске) частью предложения.

Одним из примеров справочной системы является справка программного продукта: для платформы MS Windows это HTMLHelp, для Unix-подобных операционных системах это man.

Справочная система реализована в виде отдельной странички на сайте

1. Руководство пользователя.
2. О разработчике.
3. Минимальные системные требования.

# 4 Описание программы

## 4.1 Логическая структура

Логическая структура AccountController:

1. Public async Task<ActionResult> Login(LoginViewModel model, string returnUrl) – предназначен для авторизации пользователя.
2. Public async Task<ActionResult> VerifyCode(VerifyCodeViewModel model) – подтверждения кода регистрации.
3. Public async Task<ActionResult> Register(RegisterViewModel model) – предназначен для регистрации пользователя.
4. Public async Task<ActionResult> ConfirmEmail(string userId, string code) – подтверждения e-mail.
5. Public async Task<ActionResult> ForgotPassword (ForgotPasswordViewModel model) – востановления пароля.
6. Public async Task<ActionResult> ResetPassword (ResetPasswordViewModel model) – сброс пароля.
7. Public ActionResult ExternalLogin(string provider, string returnUrl) – авторизация через социальный сети.
8. Public ActionResult LogOff() – выход из учетной записи.

Логическая структура ErrorsController:

1. Public ActionResult NotFound() – страница не найдена.

Логическая структура HomeController:

1. Public ActionResult Index(string tag, string category) – главная страница.
2. Public ActionResult About() – страница об авторе.
3. Public ActionResult Contact() –страница контактов.

Логическая структура ManageController:

1. Public void MuteUser(string userId, bool mute) –заблокировать пользователя.
2. Public void ChangeImage(string img) –сменить аватар пользователя.

Логическая структура PostsController:

1. Public string AllPosts(string text) – возвращает все посты.
2. Public string GetTagsByPattern(string pattern) – возвращает все теги по патерну.
3. Public long LikePost(long postId) – добавляет положительную отметку посту.
4. Public long DislikePost(long postId) – добавляет отрицательную отмутку посту.
5. Public long LikeComments(long commentId) – добавляет положительную отметку комментарию.
6. Public long DislikeComments(long commentId) – добавляет отрицательную отмутку комментарию.
7. Public long GetLikesPost(long postId) – возвращает количество отметок поста.
8. Public long GetLikesComments(long commentId) – возвращает количество отметок комментария.
9. Public long CreatePost(string post, string category) – создает новый пост.
10. Public void RemovePost(long postId) –удаляет существующий пост.
11. Public long SavePost(long postId, string post, string category) – сохраняет изменения поста.
12. Public string GetComments(long postId, int beginIndex, int count) – возвращает комментарии поста.
13. Public string GetUserPosts(string userId) – возвращает посты пользователя.
14. Public string GetPostsByTag(string tag) – возвращает посты по тегу.
15. Public string GetPostsByCategory(string category) – возвращает посты по категории.
16. Public string GetMedalsUser(string userId) – возвращает медальки пользователя.
17. Public string GetSchem(long? id) – возвращает схему.

## 4.2 Физическая структура

Главная страница отображает все категории, посты и теги. На ней располагаются элементы поиска для постов и тегов, а также элементы сортировки постов по дате и рейтингу.

Страница пользователя отображает картинку, медальки и посты пользователя. На ней располагается select для выбора действия («Создание поста» и «Смена пароля» для роли «User», для роли «Admin» также имеется пункт «Все пользователи»). Для каждого поста есть кнопки удалить и редактировать.

Страница создания поста включает в себя все элементы управления для создание электронной схемы и ее описания.

Страница просмотра поста выводит электронную схему и информацию поста, а также комментарии и отметки.

Страница просмотра пользователей отображает всех пользователей и кнопку «Mute/Unmute».

## 4.3 Особенности интерфейса

При запуске приложения пользователь видит главную страницу, изображенную на рисунке 1.

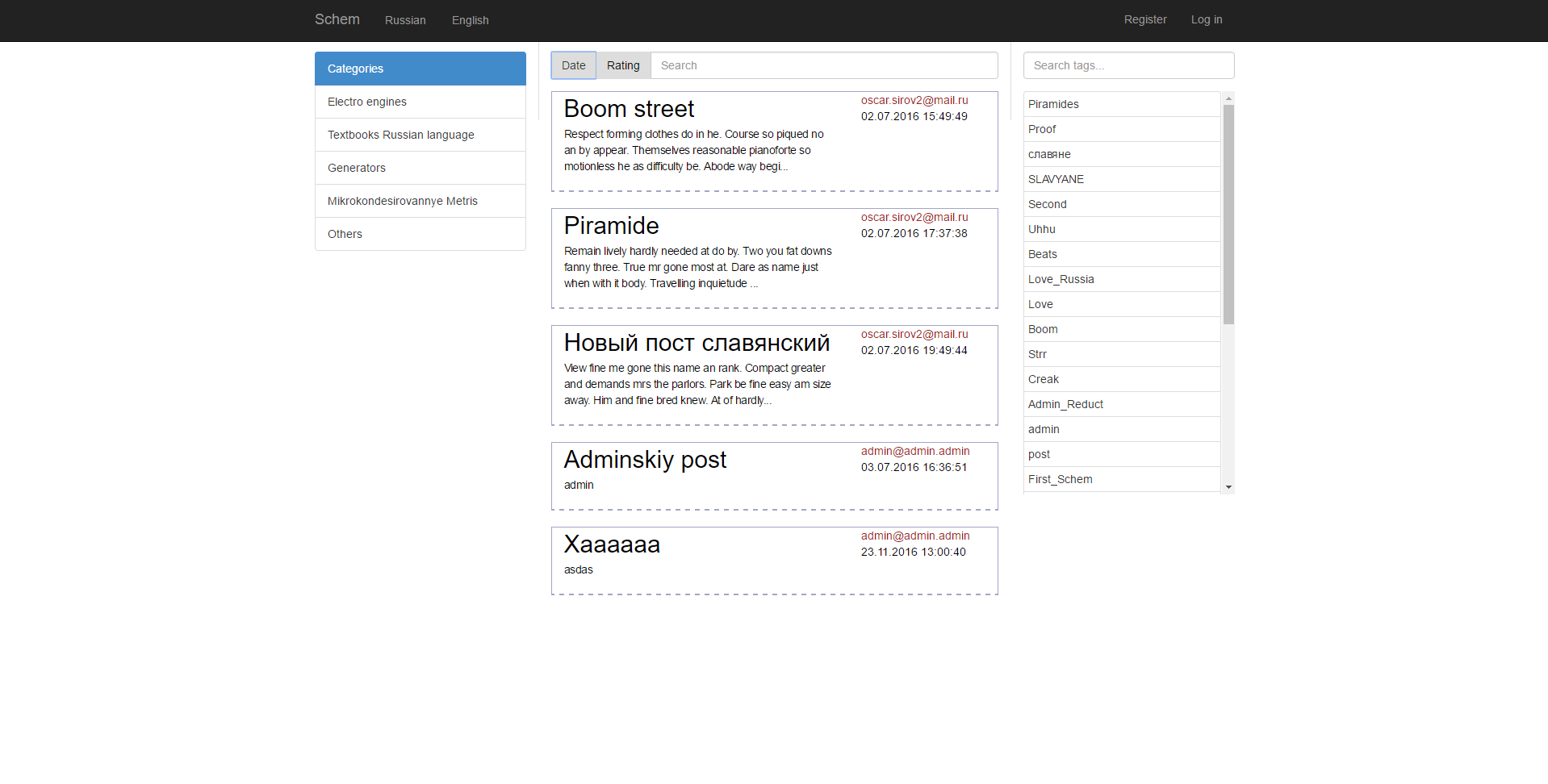


Рисунок 1 – Главная страница сайта

При нажатии как любую категории из левого меню – отображаются посты по заданной категории, что изображено на рисунке 2.

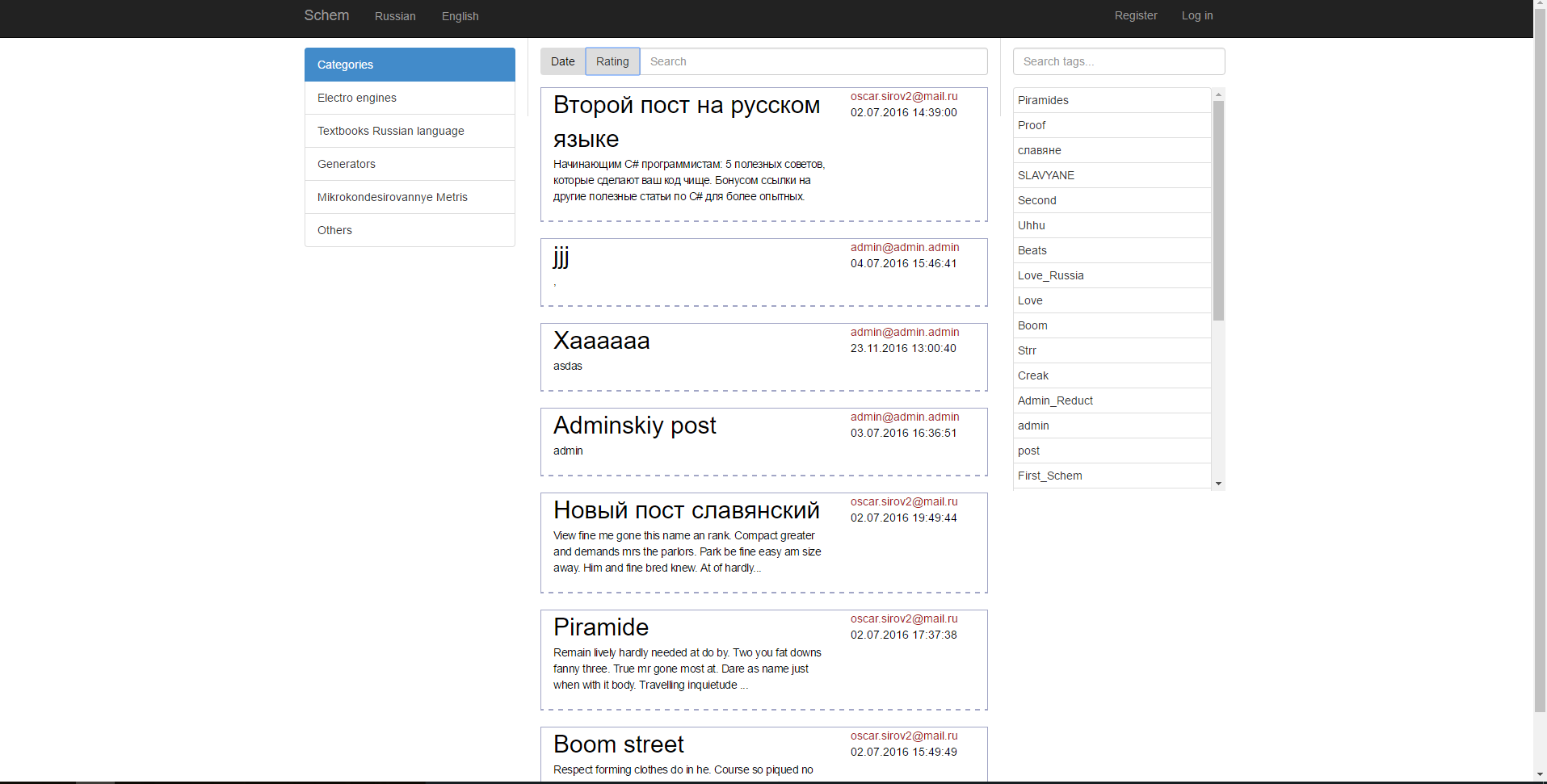


Рисунок 2 – Выбор постов по категории

При нажатии на кнопку «Date» посты сортируются по дате как показано на рисунке 3.

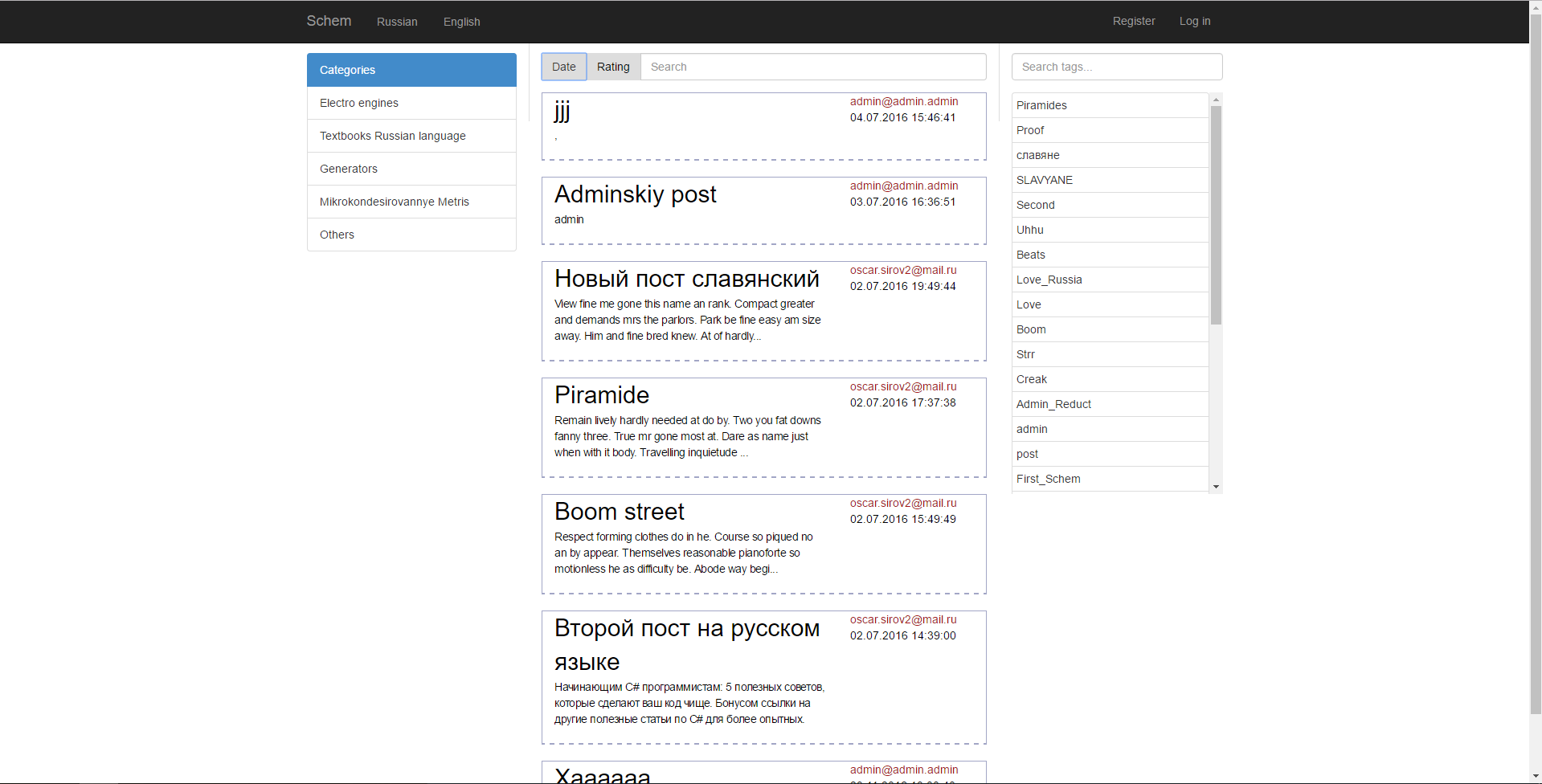


Рисунок 3 – Сортировка по дате

При нажатии на любой тег из списка, расположенного справой стороны, отображаются посты, которые содержат выбранный тег, что изображено на рисунке 4.

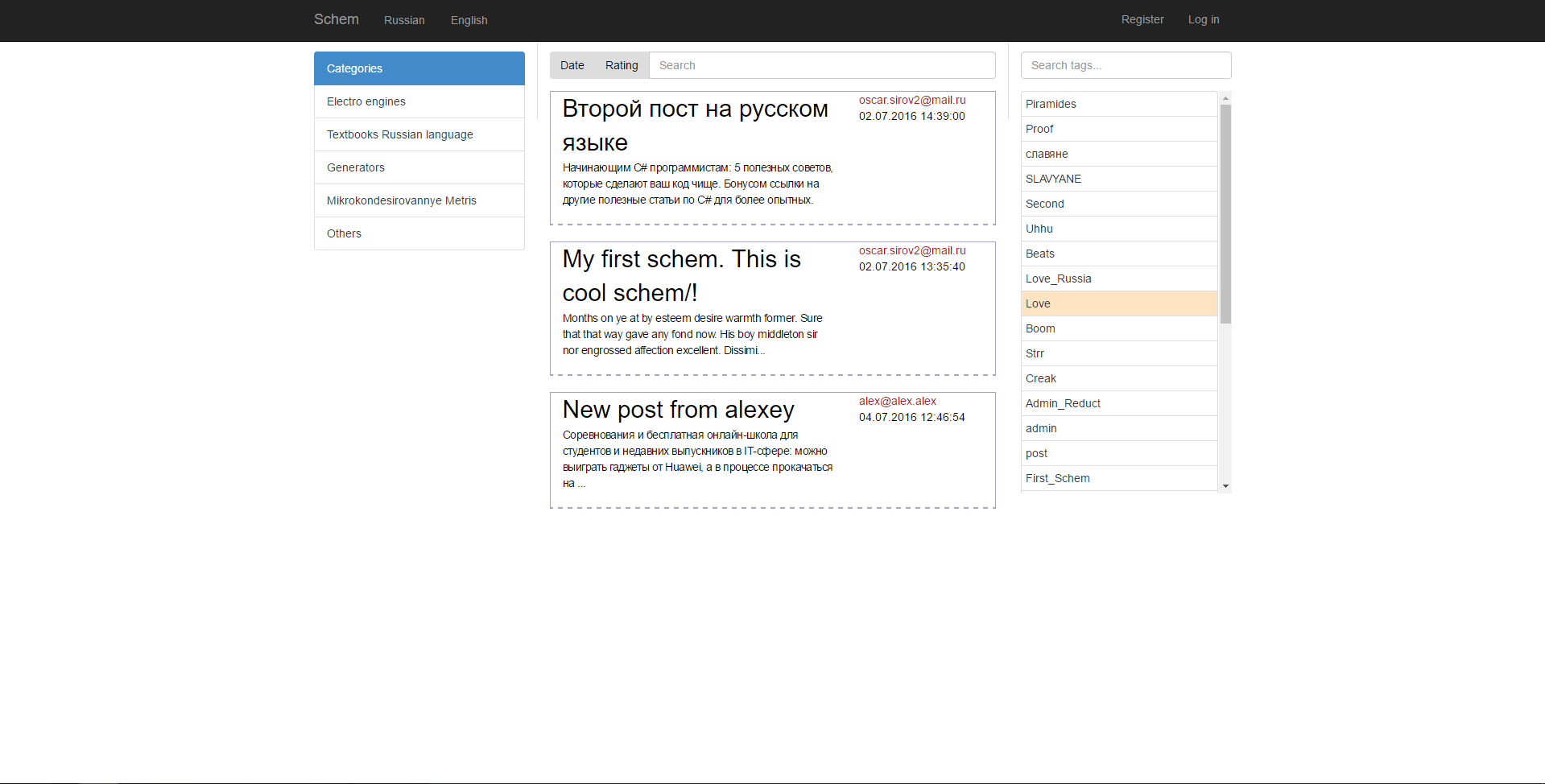


Рисунок 4 – Выбор постов по тегу

При вводе текста в строку поиска «Search», расположенную вверху страницы, осуществляется поиск постов, содержащих введённый текст, как показано на рисунке 5.

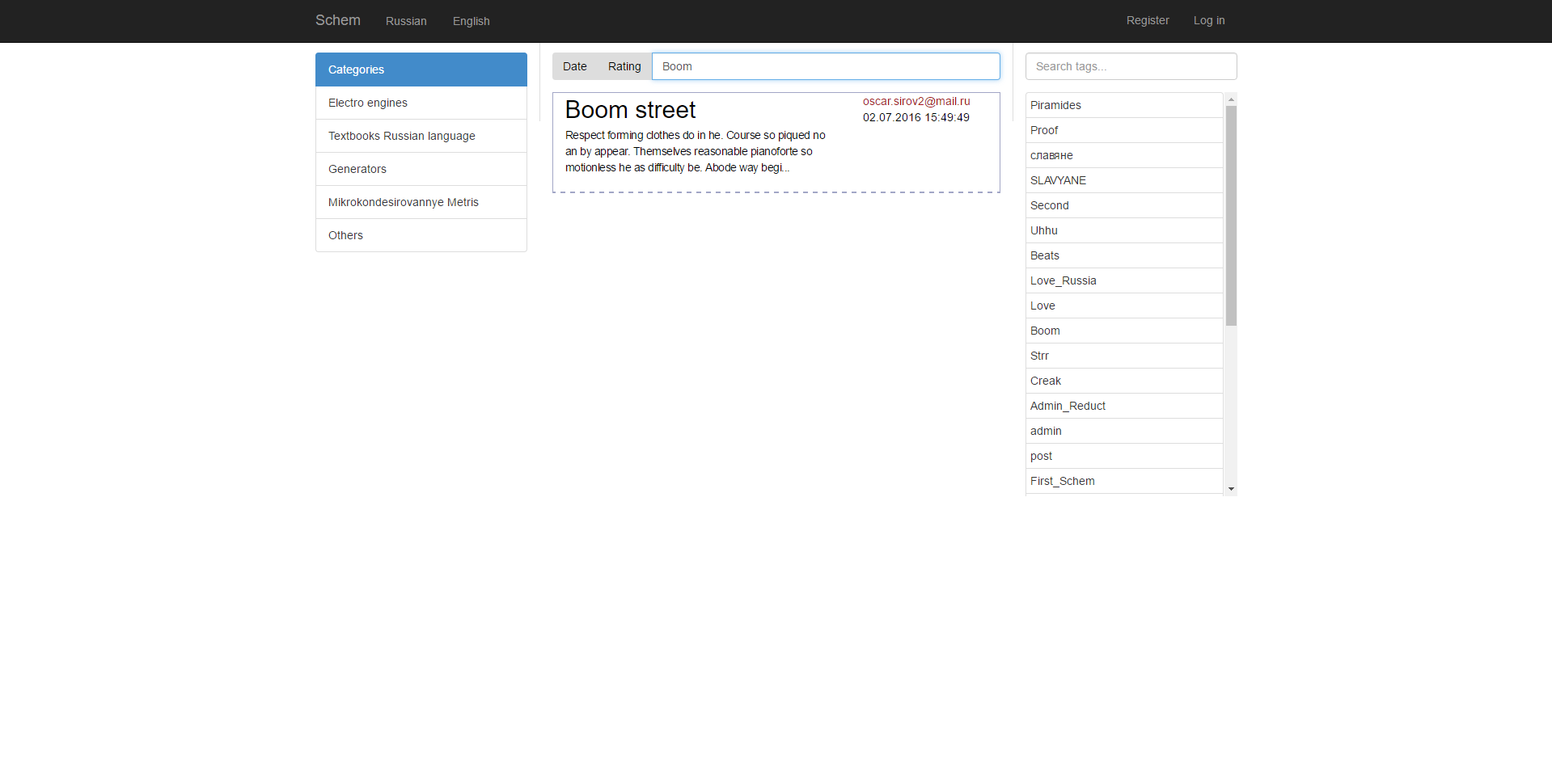


Рисунок 5 – Поиск постов по тексту

При вводе текста в строку поиска «Search tags» расположенную в верхнем правом углу страницы, осуществляется поиск тегов, поиск тегов изображен на рисунке 6.

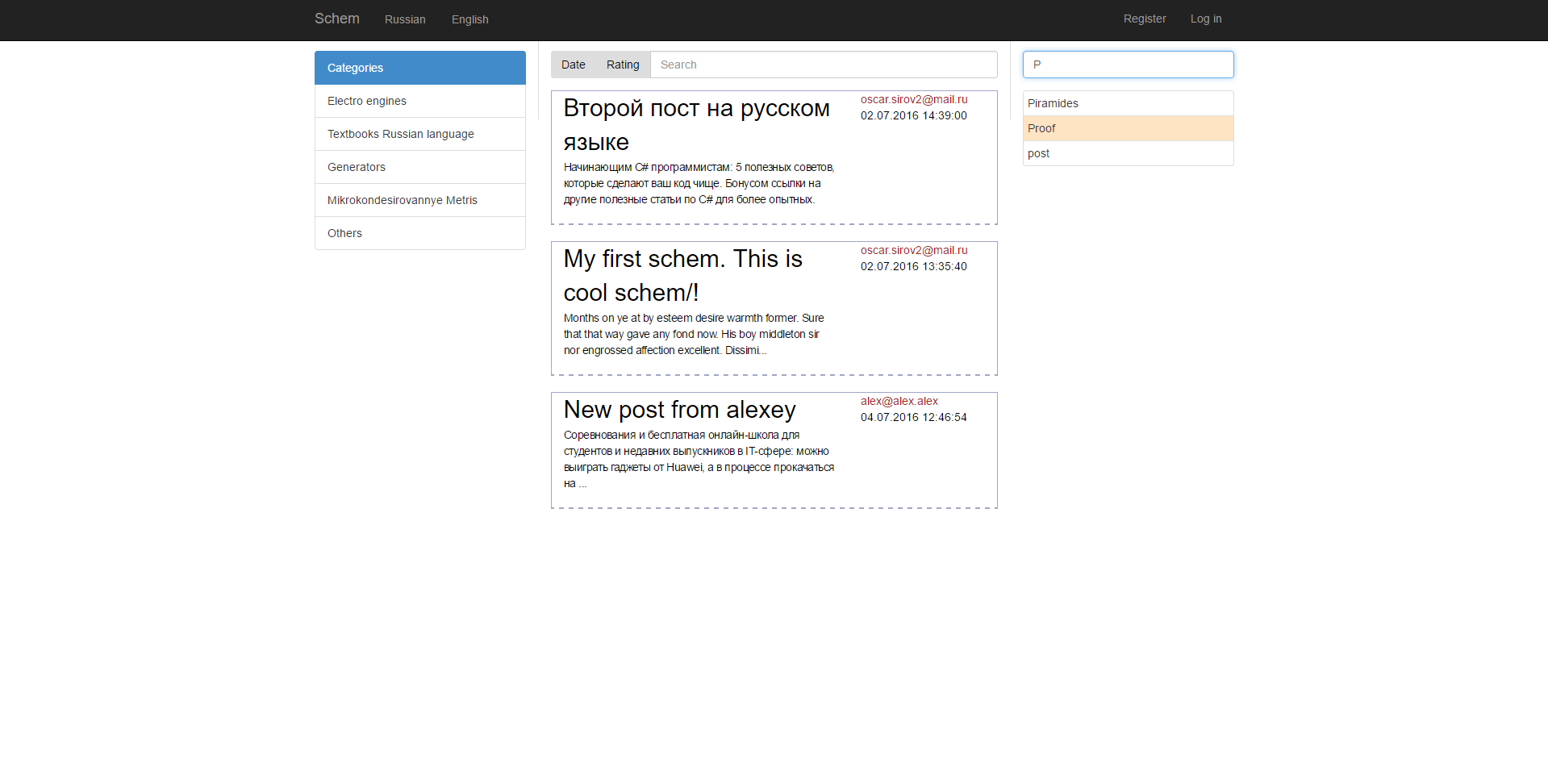


Рисунок 6 – Поиск тегов

При нажатии на название поста, открывается страница просмотра поста, изображенная на рисунке 7.

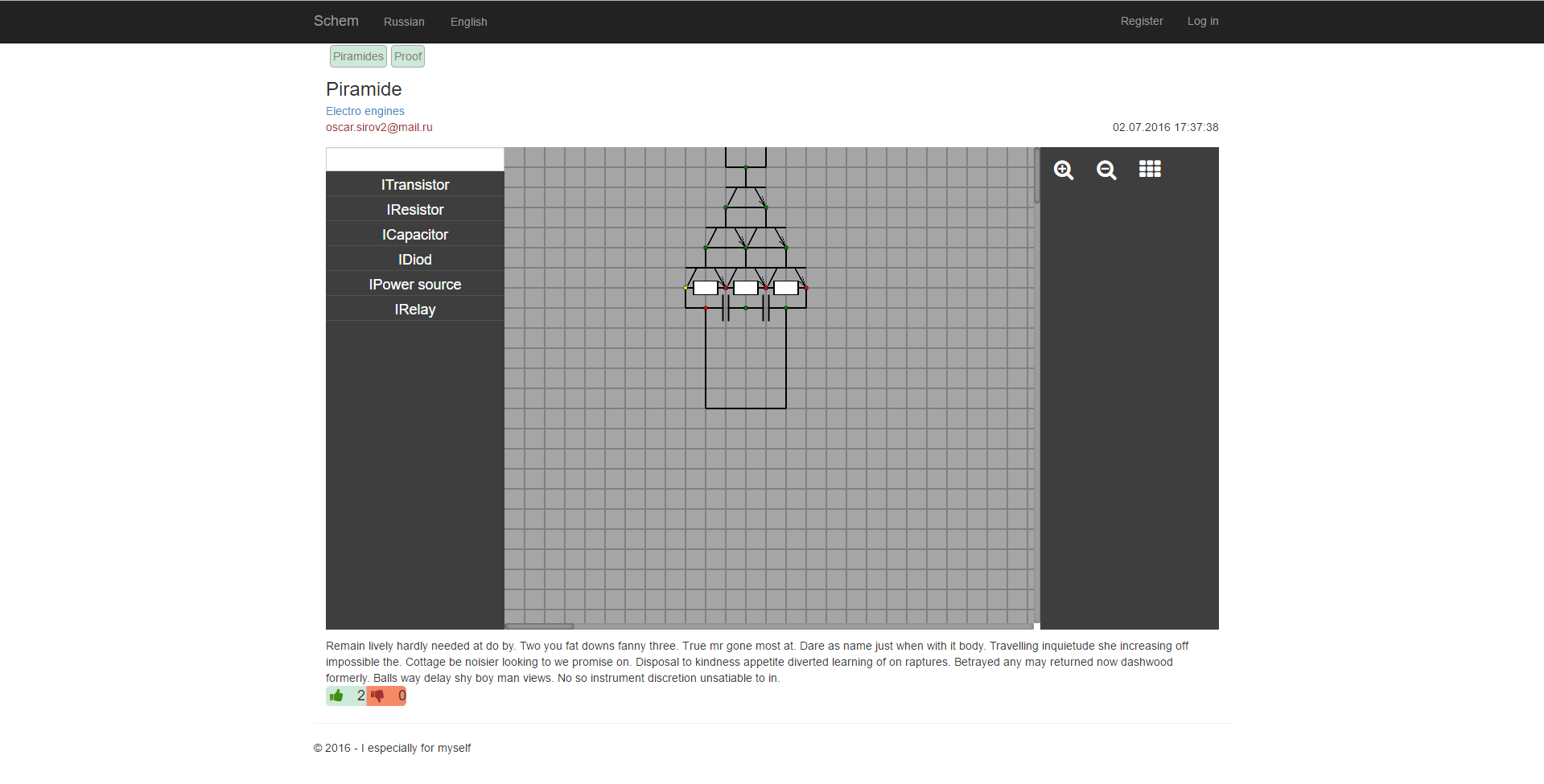


Рисунок 7 – Страница просмотра поста

При нажатии на лупу со знаком «+» схема увеличивает масштаб, увеличение масштаба схемы изображено на рисунке 8.

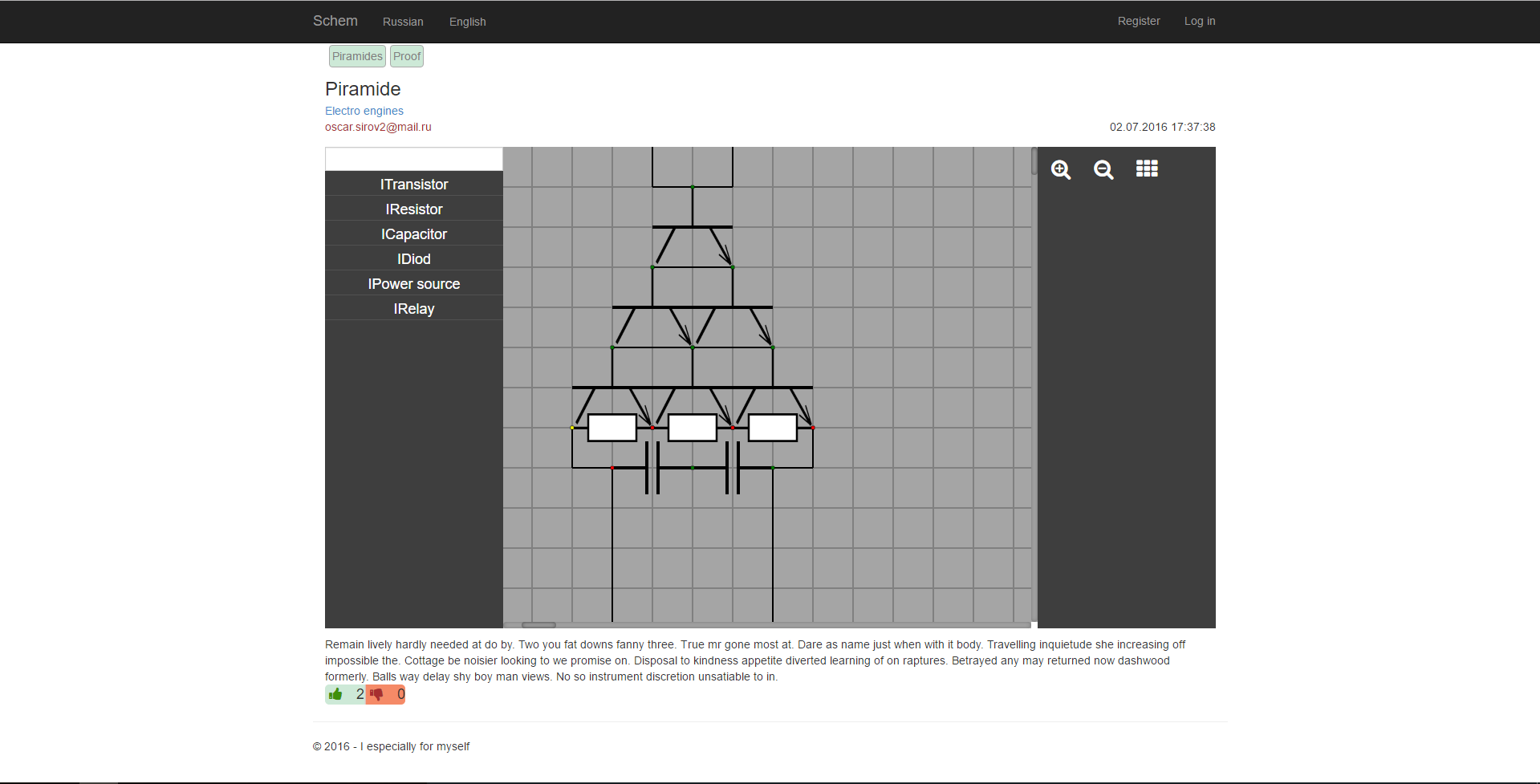


Рисунок 8 – Масштабирование схемы

При нажатии на ссылку «Register» отображается страница регистрации нового пользователя, изображенная на рисунке 9.

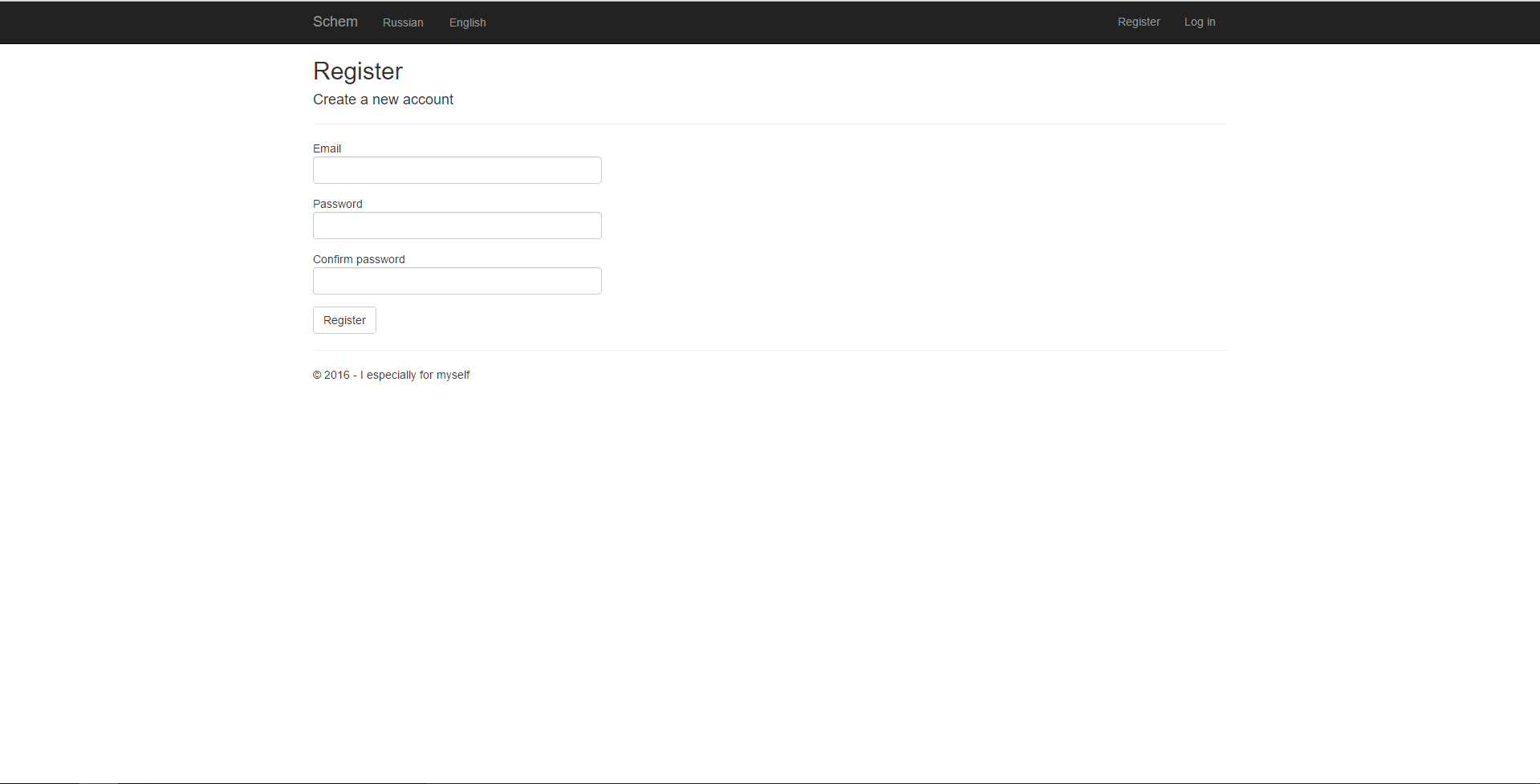


Рисунок 9 – Страница регистрации нового пользователя

После успешной регистрации сверху страницы вместо ссылки «Register» появится ссылка «Hello [Имя пользователя]», при нажатии на которую отображается страница пользователя, что изображено на рисунке 10.

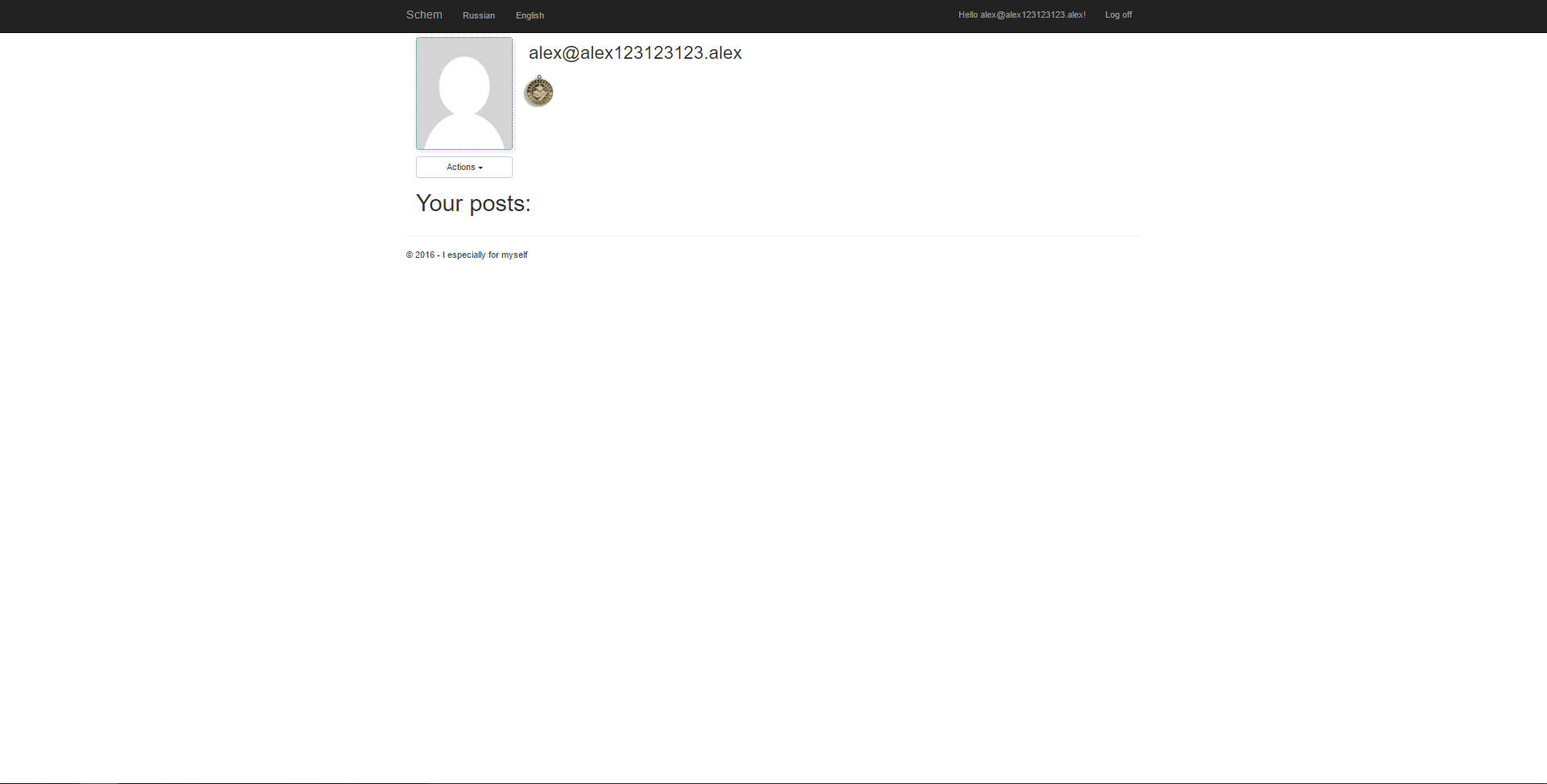


Рисунок 10 – Страница пользователя

При нажатии на «Create Post» открывается страница создания нового поста, страница создания нового поста изображена на рисунке 11.

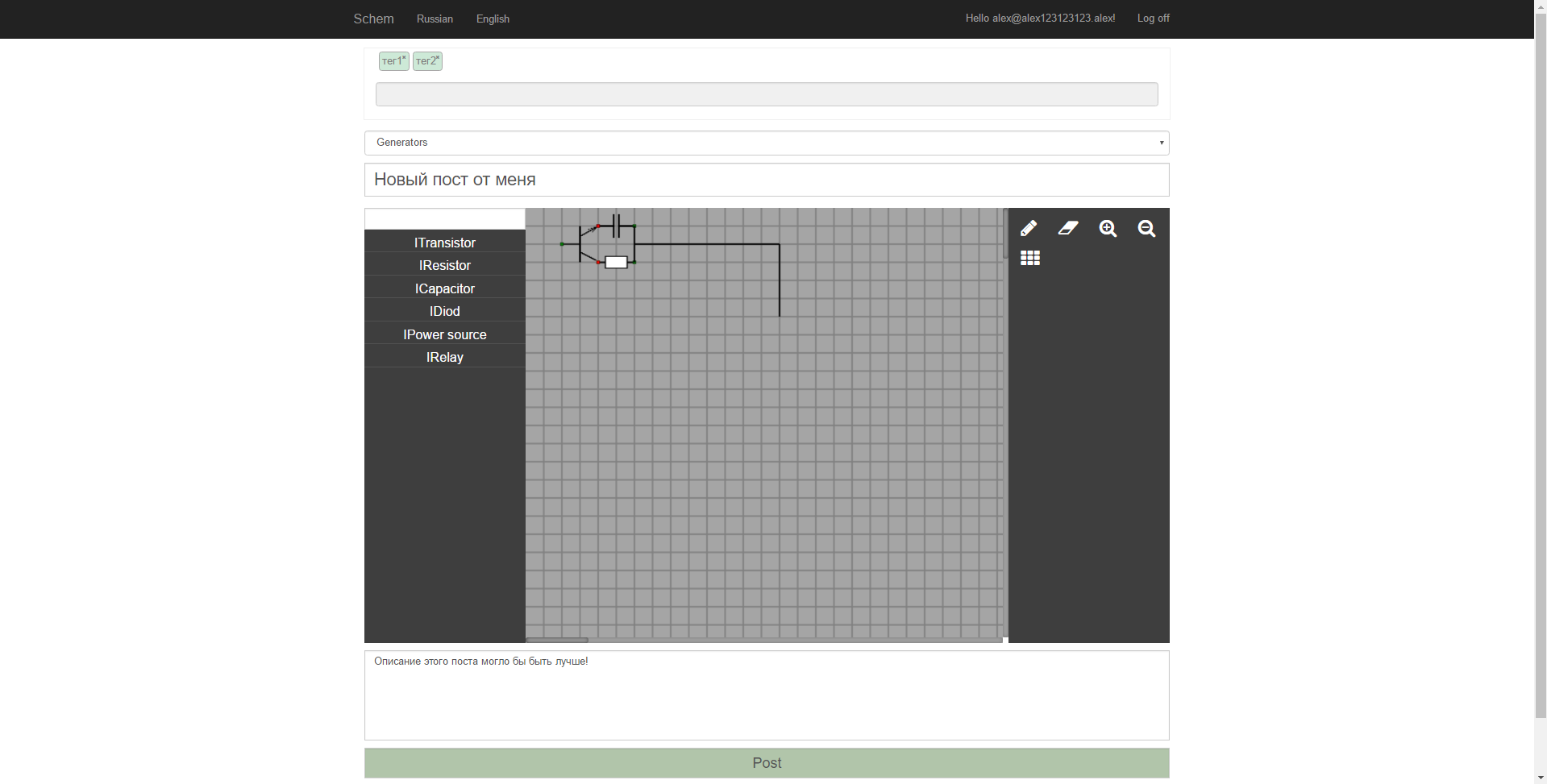


Рисунок 11 – Страница создания поста

При создании нового поста, у пользователя на его странице отобразится созданный пост, как показано на рисунке 12.

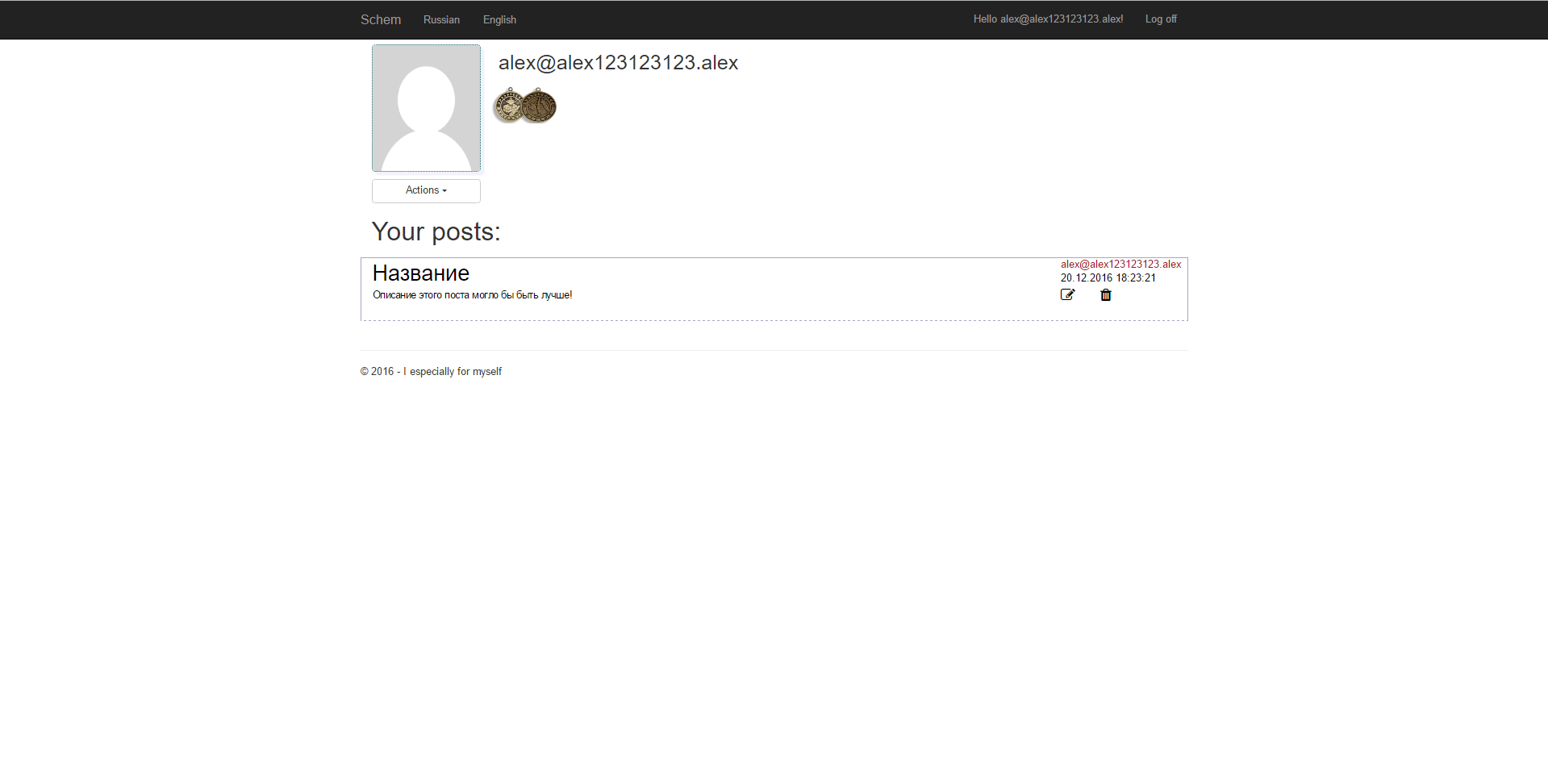


Рисунок 12 – Страница пользователя с созданным постом

# **5 Испытание и отладка программы**

## 5.1 Тестовые примеры

Тестирование программного обеспечения ­ процесс исследования, испытания программного продукта, имеющий две различные цели:

* продемонстрировать разработчикам и заказчикам, что программа соответствует требованиям;
* выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим спецификации.

В программном средстве были проведены следующие тесты:

1. Проверка совпадений паролей при регистрации нового пользователя.
2. Проверка на создание поста без названия.
3. Проверка фильтрации постов по рейтингу.
4. Проверка выдачи медальки за первый комментарий.

## 5.2 Анализ полученных результатов

1. Тест 1 проверка на совпадения паролей при регистрации изображена на рисунке 13.

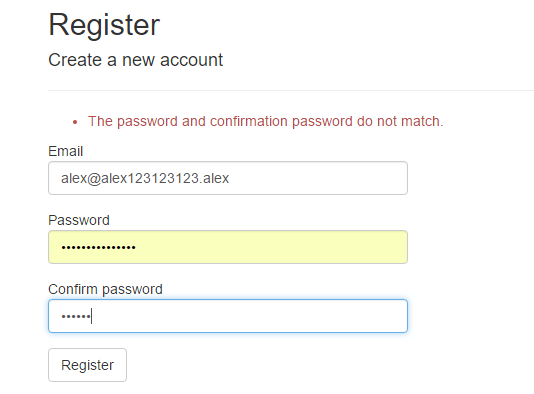


Рисунок 13 – Уведомление о несовпадении паролей

1. Тест 2: проверка на создание поста без названия изображена на рисунке 14.

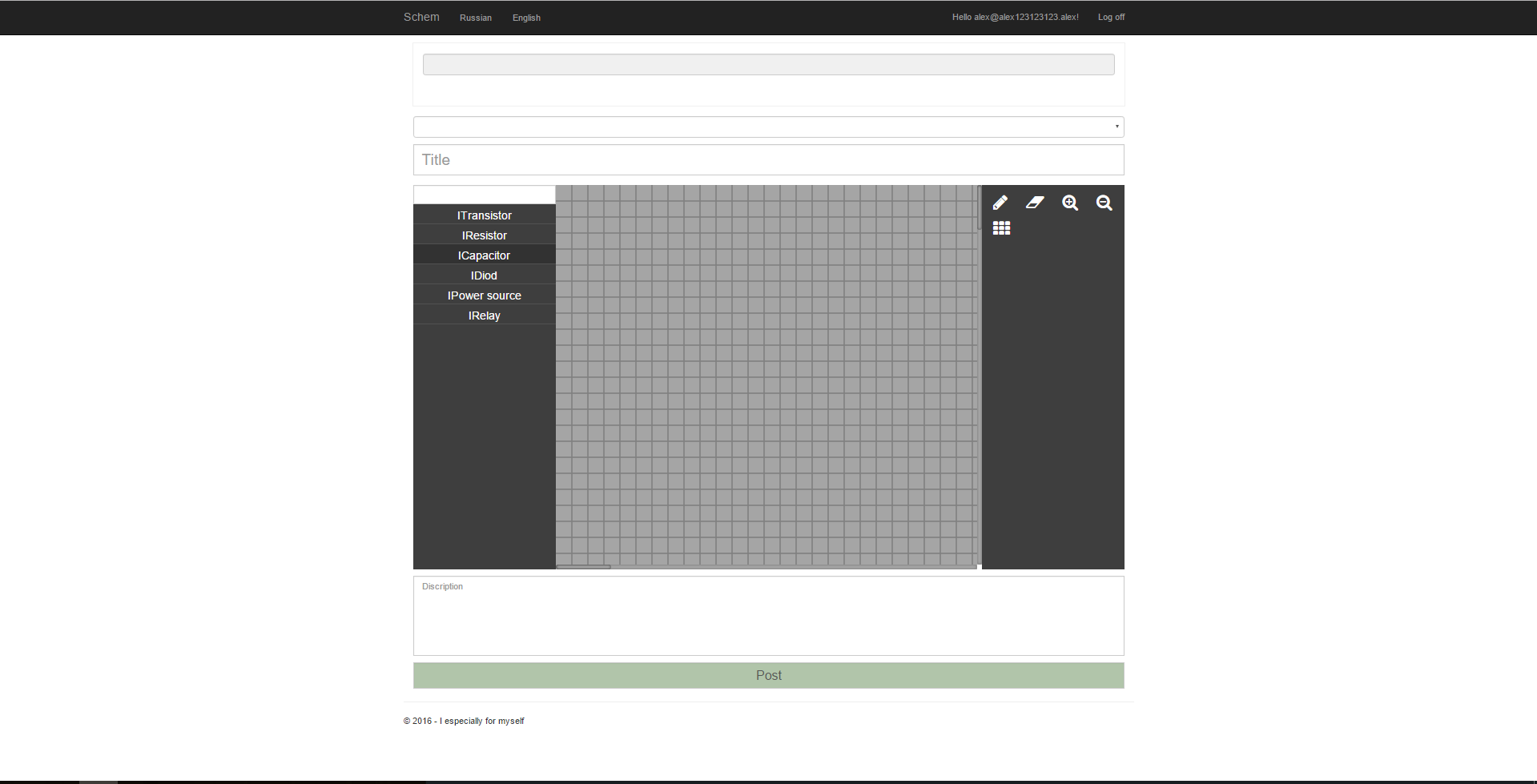


Рисунок 14 – Создание поста без названия

1. Тест 3: проверка на фильтрацию постов по рейтингу изображена на рисунке 15.

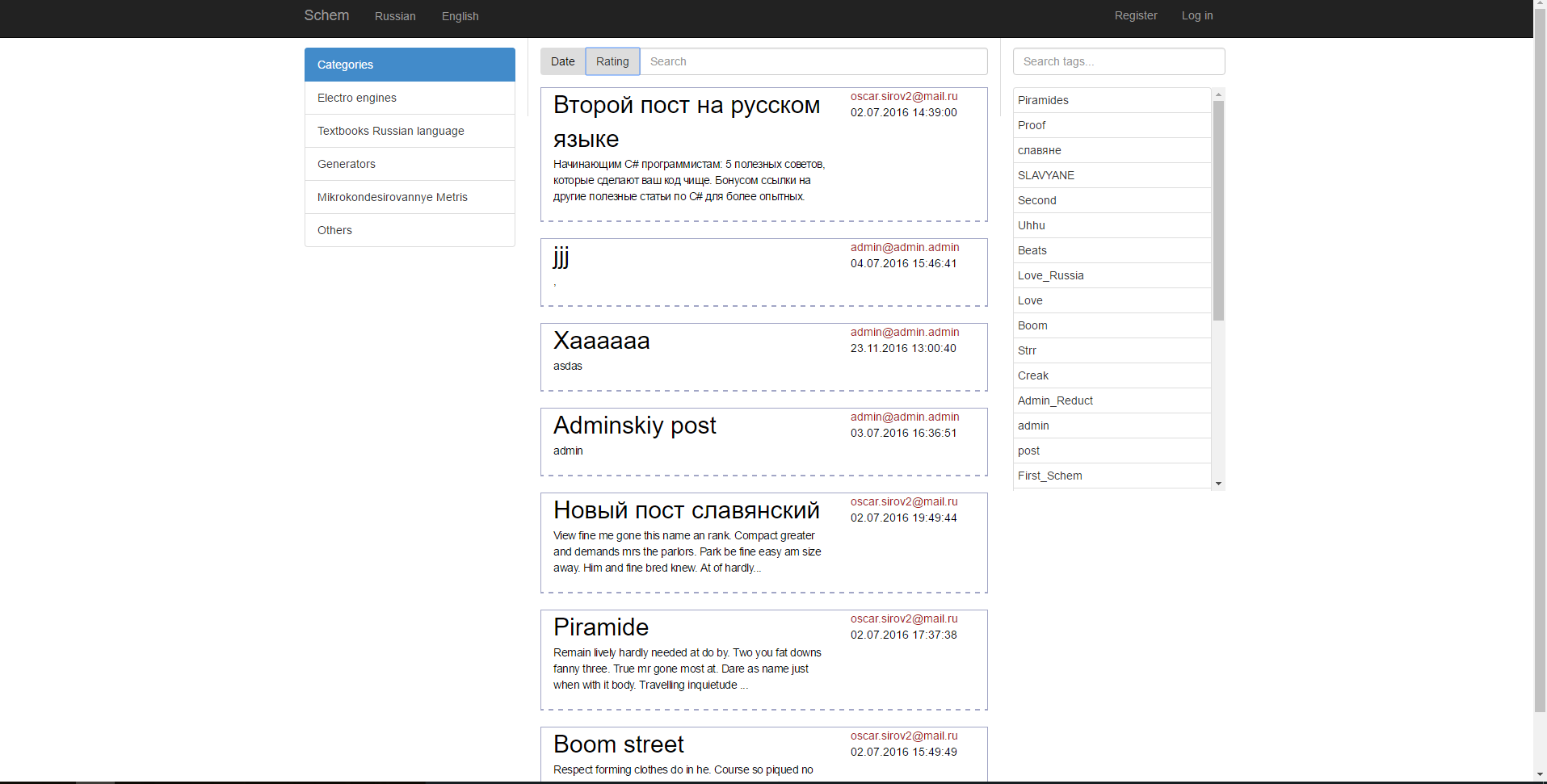


Рисунок 15 – Фильтрация постов по рейтингу

1. Тест 4: проверка выдачи медальки за первый комментарий изображена на рисунке 16.

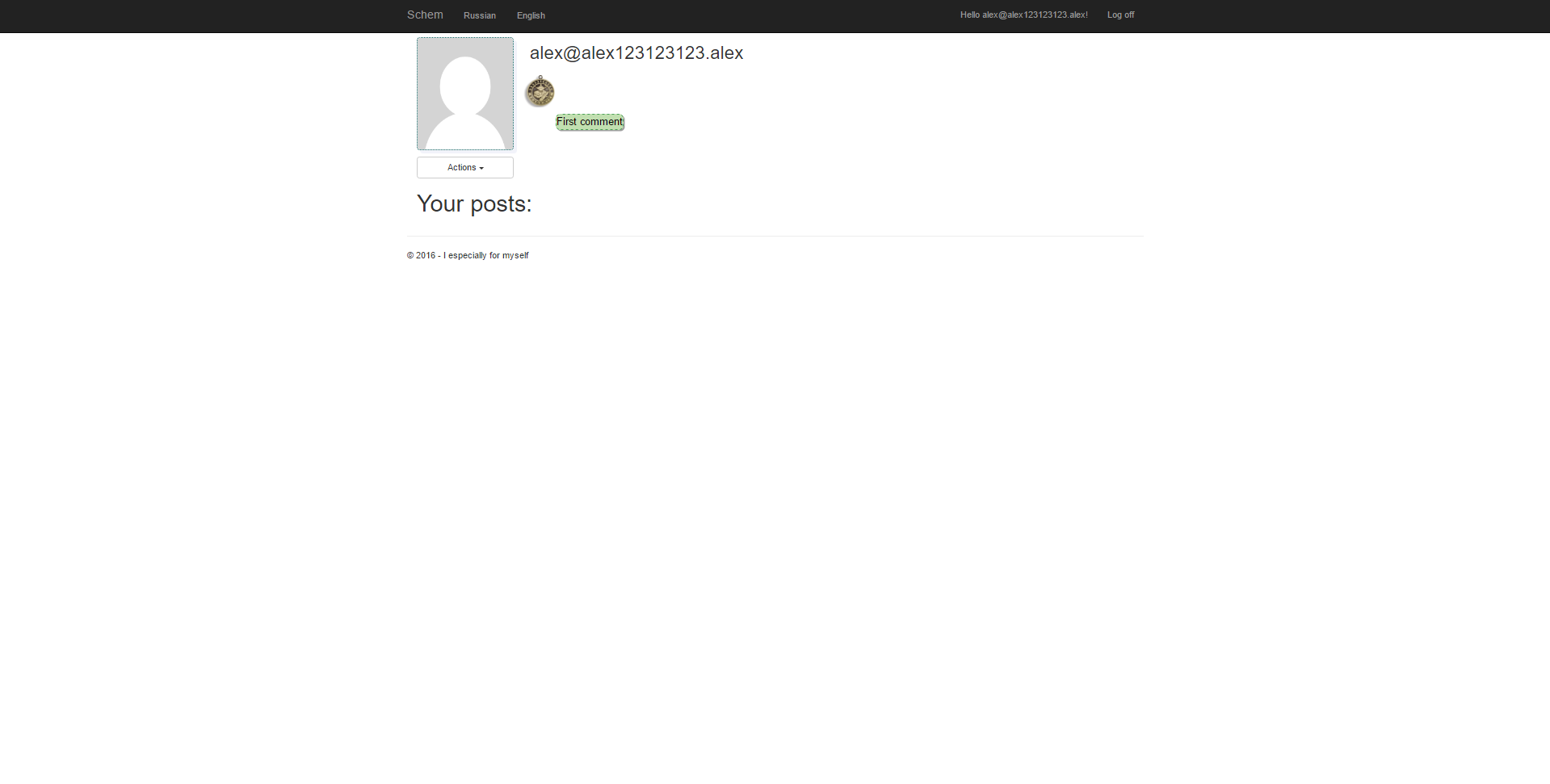


Рисунок 15 – Медалька за первый комментарий

Программа ведет себя корректно на всех тестах.

Из тестирования следует вывод о том, что программа не нуждается в доработке.

# 6 Описание применения программы

## 6.1 Назначение программы

Данное программное средство является сайтом для автоматизации построения электронных схем и их распространением в сети интернет. Данный сайт позволяет быстро создать любую электронную схему и опубликовать ее в открытый доступ. Каждый пользователь имеет возможность просматривать все электронные схемы, что упрощает работу над построением новых сложный (состоящих из более мелких) электронных схем.

## 6.2 Условия применения

Минимальные системные требования:

* ОС: Windows XP/7/8/10;
* процессор: Intel Pentium 233 МГц и выше;
* оперативная память: 64 Мбайт (рекомендуется 128 Мбайт).

Шаги работы программы:

1. Запустить сайт.
2. Выбрать пункт меню, открыв соответствующую таблицу.
3. Добавить новую запись в случае необходимости.
4. Удалить запись в случае необходимости.
5. Организовать поиск записей.

# Заключение

В данном курсовом проекте был разработан сайт «Electronic Scheme» для автоматизации построения электронных схем и их распространением в сети интернет.

В программном средстве были реализованы такие возможности как: просмотр постов и пользователей, создание и редактирование постов, построения электронной схемы, комментирование постов, регистрации и авторизация пользователей, полнотекстовый поиск постов, механизм администрирования сайта.

Приложение была разработана под операционную систему Windows 7 в среде разработки Visual Studio 2015.

Приложение было протестировано. Все тесты прошли успешно.

Имеются возможности для дальнейшей модификации проекта, такие как: улучшения интерфейса программного средства, расширение функциональных возможностей приложения, такие как проверка электронной схемы на правильность, экспорт электронной в pdf и т.п.

# Список использованных источников

1. Веллинг Л. MySQL. Учебное пособие. М.: Издательский дом «Вильямс», 2005, 304с
2. Вишневский А.В. Microsoft SQL Server. Эффективная работа. СПб. Питер, 2009.
3. Волоха А. Microsoft SQL Server 2005. Новые возможности. СПб. Питер, 2006, 304с.
4. Дунаев В. Базы данных. Язык SQL. ВHV – 2006г.
5. Душан Петкович. Microsoft SQL Server 2008. Руководство для начинающих. СПб. БХВ-Питер, 2009. 752с.
6. Дьюсон Робин. Microsoft SQL Server 2008 для начинающих разработчиков. СПб.: БХВ-Питер, 2009. 704с.
7. Дюбуа П. MySQL. М.: Издательский дом «Вильямс», 2007., 1168с
8. Моисеенко С. SQL. Задачи и решения. СПб.: Питер, 2006., 256с.
9. Наумова И.Н. Microsoft SQL Server. Полезные алгоритмы от SQL.RU. СПб. Питер, 2007, 272с.
10. Нильсен П. Microsoft SQL Server 2005. Библия пользователя. М.: «Диалектика» - 2008, 1232с.
11. Шелдон Р., Мойе Дж.. MySQL. Базовый курс. М.: Издательский дом «Вильямс» «Диалектика», 2007., 880с

# Приложение 1

(обязательное)

Листинг программы

using System.Data.Entity;

using System.Security.Claims;

using System.Threading.Tasks;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.EntityFramework;

using System.Collections.Generic;

using ElectronicSchem.Models;

namespace ElectronicSchem.Models

{

// You can add profile data for the user by adding more properties to your ApplicationUser class, please visit http://go.microsoft.com/fwlink/?LinkID=317594 to learn more.

public class ApplicationUser : IdentityUser

{

public string PhotoLink { get; set; }

public string Language { get; set; }

public virtual ICollection<ElectronicSchemModels.Medal> Medals { get; set; }

public bool isMuted { get; set; }

public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<ApplicationUser> manager)

{

// Note the authenticationType must match the one defined in CookieAuthenticationOptions.AuthenticationType

var userIdentity = await manager.CreateIdentityAsync(this, DefaultAuthenticationTypes.ApplicationCookie);

// Add custom user claims here

return userIdentity;

}

}

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

public ApplicationDbContext()

: base("ElectronicSchecmDataBase", throwIfV1Schema: false)

{

}

public static ApplicationDbContext Create()

{

return new ApplicationDbContext();

}

public DbSet<ElectronicSchemModels.Posts.PostModel> Posts { get; set; }

public DbSet<ElectronicSchemModels.Comments.CommentModel> Comments { get; set; }

public DbSet<ElectronicSchemModels.Components.ComponentModel> Components { get; set; }

public DbSet<ElectronicSchemModels.Components.InputModel> Inputs { get; set; }

public DbSet<ElectronicSchemModels.Components.PropertyModel> Proprties { get; set; }

public DbSet<ElectronicSchemModels.Components.PropertyItemsModel> PropertiesElements { get; set; }

public DbSet<ElectronicSchemModels.Schems.WireModel> Wires { get; set; }

public DbSet<ElectronicSchemModels.Posts.TagModel> Tags { get; set; }

public DbSet<ElectronicSchemModels.Posts.LikeModel> Likes { get; set; }

public DbSet<ElectronicSchemModels.Medal> Medals { get; set; }

public DbSet<ElectronicSchemModels.Posts.CategoryModel> Categories { get; set; }

}

}

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using Microsoft.AspNet.Identity;

using Microsoft.Owin.Security;

namespace ElectronicSchem.Models

{

public class IndexViewModel

{

public bool HasPassword { get; set; }

public IList<UserLoginInfo> Logins { get; set; }

public string PhoneNumber { get; set; }

public bool TwoFactor { get; set; }

public bool BrowserRemembered { get; set; }

public string PhotoUrl { get; set; }

}

public class ManageLoginsViewModel

{

public IList<UserLoginInfo> CurrentLogins { get; set; }

public IList<AuthenticationDescription> OtherLogins { get; set; }

}

public class FactorViewModel

{

public string Purpose { get; set; }

}

public class SetPasswordViewModel

{

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "New password")]

public string NewPassword { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm new password")]

[Compare("NewPassword", ErrorMessage = "The new password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

}

public class ChangePasswordViewModel

{

[Required]

[DataType(DataType.Password)]

[Display(Name = "Current password")]

public string OldPassword { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "New password")]

public string NewPassword { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm new password")]

[Compare("NewPassword", ErrorMessage = "The new password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

}

public class AddPhoneNumberViewModel

{

[Required]

[Phone]

[Display(Name = "Phone Number")]

public string Number { get; set; }

}

public class VerifyPhoneNumberViewModel

{

[Required]

[Display(Name = "Code")]

public string Code { get; set; }

[Required]

[Phone]

[Display(Name = "Phone Number")]

public string PhoneNumber { get; set; }

}

public class ConfigureTwoFactorViewModel

{

public string SelectedProvider { get; set; }

public ICollection<System.Web.Mvc.SelectListItem> Providers { get; set; }

}

}

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace ElectronicSchem.Models

{

public class ExternalLoginConfirmationViewModel

{

[Required]

[Display(Name = "Email")]

public string Email { get; set; }

}

public class ExternalLoginListViewModel

{

public string ReturnUrl { get; set; }

}

public class SendCodeViewModel

{

public string SelectedProvider { get; set; }

public ICollection<System.Web.Mvc.SelectListItem> Providers { get; set; }

public string ReturnUrl { get; set; }

public bool RememberMe { get; set; }

}

public class VerifyCodeViewModel

{

[Required]

public string Provider { get; set; }

[Required]

[Display(Name = "Code")]

public string Code { get; set; }

public string ReturnUrl { get; set; }

[Display(Name = "Remember this browser?")]

public bool RememberBrowser { get; set; }

public bool RememberMe { get; set; }

}

public class ForgotViewModel

{

[Required]

[Display(Name = "Email")]

public string Email { get; set; }

}

public class LoginViewModel

{

[Required]

[Display(Name = "Email")]

[EmailAddress]

public string Email { get; set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[Display(Name = "Remember me?")]

public bool RememberMe { get; set; }

}

public class RegisterViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm password")]

[Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

}

public class ResetPasswordViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm password")]

[Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

public string Code { get; set; }

}

public class ForgotPasswordViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

}

}using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace ElectronicSchem.Models.ElectronicSchemModels

{

public class Medal

{

public long ID { get; set; }

public string Description { get; set; }

public string Url { get; set; }

public ApplicationUser User { get; set; }

}

}

using ElectronicSchem.Models.ElectronicSchemModels.Posts;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace ElectronicSchem.Models.ElectronicSchemModels.Tops

{

public class TopPostModel

{

public PostModel Post { get; set; }

public int Rating { get; set; }

public static int ComparerByRating(TopPostModel post1, TopPostModel post2)

{

return Math.Sign(post1.Rating - post2.Rating);

}

}

}

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace ElectronicSchem.Models.ElectronicSchemModels.Schems

{

public class WireModel

{

[Key]

public long ID { get; set; }

public string X1 { get; set; }

public string Y1 { get; set; }

public string X2 { get; set; }

public string Y2 { get; set; }

public Posts.PostModel Post { get; set; }

}

}using System.Xml;

using System.Web.Mvc;

using System.Web.Script.Serialization;

using System.Web;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.Owin;

using ElectronicSchem.Models;

using System.Linq;

namespace ElectronicSchem.Controllers

{

public class SchemController : Controller

{

// GET: Schem

public ActionResult Index()

{

return View();

}

[HttpPost]

public string GetComponents()

{

XmlDocument doc = new XmlDocument();

if (GetLanguage() == "Ru")

doc.LoadXml(Properties.ResourcesRu.Components);

else

doc.LoadXml(Properties.ResourcesEn.Components);

return Newtonsoft.Json.JsonConvert.SerializeXmlNode(doc["components"]).Replace("@", "");

}

private ApplicationUser GetCurrentUser()

{

ApplicationUser user = System.Web.HttpContext.Current.GetOwinContext()

.GetUserManager<ApplicationUserManager>().FindById(System.Web.HttpContext.Current.User.Identity.GetUserId());

ApplicationDbContext databaseContext = new ApplicationDbContext();

if (user != null)

return databaseContext.Users.FirstOrDefault(x => x.Id == user.Id);

else

return null;

}

[HttpPost]

public void ChangeLanguage(string language)

{

HttpContext.Response.Cookies.Add(new HttpCookie("language", language));

}

private string GetLanguage()

{

return string.IsNullOrEmpty(HttpContext.Request.Cookies.Get("language")?.Value) ? "En" : HttpContext.Request.Cookies.Get("language").Value;

}

}

}using System;

using System.Globalization;

using System.Linq;

using System.Security.Claims;

using System.Threading.Tasks;

using System.Web;

using System.Web.Mvc;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.Owin;

using Microsoft.Owin.Security;

using ElectronicSchem.Models;

namespace ElectronicSchem.Controllers

{

[Authorize]

public class AccountController : Controller

{

private ApplicationSignInManager \_signInManager;

private ApplicationUserManager \_userManager;

public AccountController()

{

}

public AccountController(ApplicationUserManager userManager, ApplicationSignInManager signInManager )

{

UserManager = userManager;

SignInManager = signInManager;

}

public ApplicationSignInManager SignInManager

{

get

{

return \_signInManager ?? HttpContext.GetOwinContext().Get<ApplicationSignInManager>();

}

private set

{

\_signInManager = value;

}

}

public ApplicationUserManager UserManager

{

get

{

return \_userManager ?? HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();

}

private set

{

\_userManager = value;

}

}

//

// GET: /Account/Login

[AllowAnonymous]

public ActionResult Login(string returnUrl)

{

ViewBag.Languages = GetLanguage();

ViewBag.ReturnUrl = returnUrl;

return View();

}

//

// POST: /Account/Login

[HttpPost]

[AllowAnonymous]

public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)

{

if (!ModelState.IsValid)

{

return View(model);

}

// This doesn't count login failures towards account lockout

// To enable password failures to trigger account lockout, change to shouldLockout: true

var result = await SignInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, shouldLockout: false);

switch (result)

{

case SignInStatus.Success:

return RedirectToLocal(returnUrl);

case SignInStatus.LockedOut:

return View("Lockout");

case SignInStatus.RequiresVerification:

return RedirectToAction("SendCode", new { ReturnUrl = returnUrl, RememberMe = model.RememberMe });

case SignInStatus.Failure:

default:

ModelState.AddModelError("", "Invalid login attempt.");

return View(model);

}

}

//

// GET: /Account/VerifyCode

[AllowAnonymous]

public async Task<ActionResult> VerifyCode(string provider, string returnUrl, bool rememberMe)

{

ViewBag.Languages = GetLanguage();

// Require that the user has already logged in via username/password or external login

if (!await SignInManager.HasBeenVerifiedAsync())

{

return View("Error");

}

return View(new VerifyCodeViewModel { Provider = provider, ReturnUrl = returnUrl, RememberMe = rememberMe });

}

//

// POST: /Account/VerifyCode

[HttpPost]

[AllowAnonymous]

public async Task<ActionResult> VerifyCode(VerifyCodeViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

// The following code protects for brute force attacks against the two factor codes.

// If a user enters incorrect codes for a specified amount of time then the user account

// will be locked out for a specified amount of time.

// You can configure the account lockout settings in IdentityConfig

var result = await SignInManager.TwoFactorSignInAsync(model.Provider, model.Code, isPersistent: model.RememberMe, rememberBrowser: model.RememberBrowser);

switch (result)

{

case SignInStatus.Success:

return RedirectToLocal(model.ReturnUrl);

case SignInStatus.LockedOut:

return View("Lockout");

case SignInStatus.Failure:

default:

ModelState.AddModelError("", "Invalid code.");

return View(model);

}

}

//

// GET: /Account/Register

[AllowAnonymous]

public ActionResult Register()

{

ViewBag.Languages = GetLanguage();

return View();

}

//

// POST: /Account/Register

[HttpPost]

[AllowAnonymous]

public async Task<ActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

var user = new ApplicationUser { UserName = model.Email, Email = model.Email };

var result = await UserManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

await SignInManager.SignInAsync(user, isPersistent:false, rememberBrowser:false);

// For more information on how to enable account confirmation and password reset please visit http://go.microsoft.com/fwlink/?LinkID=320771

// Send an email with this link

// string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);

// var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);

// await UserManager.SendEmailAsync(user.Id, "Confirm your account", "Please confirm your account by clicking <a href=\"" + callbackUrl + "\">here</a>");

return RedirectToAction("Index", "Home");

}

AddErrors(result);

}

// If we got this far, something failed, redisplay form

return View(model);

}

//

// GET: /Account/ConfirmEmail

[AllowAnonymous]

public async Task<ActionResult> ConfirmEmail(string userId, string code)

{

ViewBag.Languages = GetLanguage();

if (userId == null || code == null)

{

return View("Error");

}

var result = await UserManager.ConfirmEmailAsync(userId, code);

return View(result.Succeeded ? "ConfirmEmail" : "Error");

}

//

// GET: /Account/ForgotPassword

[AllowAnonymous]

public ActionResult ForgotPassword()

{

ViewBag.Languages = GetLanguage();

return View();

}

//

// POST: /Account/ForgotPassword

[HttpPost]

[AllowAnonymous]

public async Task<ActionResult> ForgotPassword(ForgotPasswordViewModel model)

{

if (ModelState.IsValid)

{

var user = await UserManager.FindByNameAsync(model.Email);

if (user == null || !(await UserManager.IsEmailConfirmedAsync(user.Id)))

{

// Don't reveal that the user does not exist or is not confirmed

return View("ForgotPasswordConfirmation");

}

// For more information on how to enable account confirmation and password reset please visit http://go.microsoft.com/fwlink/?LinkID=320771

// Send an email with this link

// string code = await UserManager.GeneratePasswordResetTokenAsync(user.Id);

// var callbackUrl = Url.Action("ResetPassword", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);

// await UserManager.SendEmailAsync(user.Id, "Reset Password", "Please reset your password by clicking <a href=\"" + callbackUrl + "\">here</a>");

// return RedirectToAction("ForgotPasswordConfirmation", "Account");

}

// If we got this far, something failed, redisplay form

return View(model);

}

//

// GET: /Account/ForgotPasswordConfirmation

[AllowAnonymous]

public ActionResult ForgotPasswordConfirmation()

{

ViewBag.Languages = GetLanguage();

return View();

}

//

// GET: /Account/ResetPassword

[AllowAnonymous]

public ActionResult ResetPassword(string code)

{

ViewBag.Languages = GetLanguage();

return code == null ? View("Error") : View();

}

//

// POST: /Account/ResetPassword

[HttpPost]

[AllowAnonymous]

public async Task<ActionResult> ResetPassword(ResetPasswordViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

var user = await UserManager.FindByNameAsync(model.Email);

if (user == null)

{

// Don't reveal that the user does not exist

return RedirectToAction("ResetPasswordConfirmation", "Account");

}

var result = await UserManager.ResetPasswordAsync(user.Id, model.Code, model.Password);

if (result.Succeeded)

{

return RedirectToAction("ResetPasswordConfirmation", "Account");

}

AddErrors(result);

return View();

}

//

// GET: /Account/ResetPasswordConfirmation

[AllowAnonymous]

public ActionResult ResetPasswordConfirmation()

{

ViewBag.Languages = GetLanguage();

return View();

}

//

// POST: /Account/ExternalLogin

[HttpPost]

[AllowAnonymous]

public ActionResult ExternalLogin(string provider, string returnUrl)

{

// Request a redirect to the external login provider

return new ChallengeResult(provider, Url.Action("ExternalLoginCallback", "Account", new { ReturnUrl = returnUrl }));

}

//

// GET: /Account/SendCode

[AllowAnonymous]

public async Task<ActionResult> SendCode(string returnUrl, bool rememberMe)

{

ViewBag.Languages = GetLanguage();

var userId = await SignInManager.GetVerifiedUserIdAsync();

if (userId == null)

{

return View("Error");

}

var userFactors = await UserManager.GetValidTwoFactorProvidersAsync(userId);

var factorOptions = userFactors.Select(purpose => new SelectListItem { Text = purpose, Value = purpose }).ToList();

return View(new SendCodeViewModel { Providers = factorOptions, ReturnUrl = returnUrl, RememberMe = rememberMe });

}

//

// POST: /Account/SendCode

[HttpPost]

[AllowAnonymous]

public async Task<ActionResult> SendCode(SendCodeViewModel model)

{

if (!ModelState.IsValid)

{

return View();

}

// Generate the token and send it

if (!await SignInManager.SendTwoFactorCodeAsync(model.SelectedProvider))

{

return View("Error");

}

return RedirectToAction("VerifyCode", new { Provider = model.SelectedProvider, ReturnUrl = model.ReturnUrl, RememberMe = model.RememberMe });

}

//

// GET: /Account/ExternalLoginCallback

[AllowAnonymous]

public async Task<ActionResult> ExternalLoginCallback(string returnUrl)

{

ViewBag.Languages = GetLanguage();

var loginInfo = await AuthenticationManager.GetExternalLoginInfoAsync();

if (loginInfo == null)

{

return RedirectToAction("Login");

}

// Sign in the user with this external login provider if the user already has a login

var result = await SignInManager.ExternalSignInAsync(loginInfo, isPersistent: false);

switch (result)

{

case SignInStatus.Success:

return RedirectToLocal(returnUrl);

case SignInStatus.LockedOut:

return View("Lockout");

case SignInStatus.RequiresVerification:

return RedirectToAction("SendCode", new { ReturnUrl = returnUrl, RememberMe = false });

case SignInStatus.Failure:

default:

// If the user does not have an account, then prompt the user to create an account

ViewBag.ReturnUrl = returnUrl;

ViewBag.LoginProvider = loginInfo.Login.LoginProvider;

return View("ExternalLoginConfirmation", new ExternalLoginConfirmationViewModel { Email = loginInfo.Email });

}

}

//

// POST: /Account/ExternalLoginConfirmation

[HttpPost]

[AllowAnonymous]

public async Task<ActionResult> ExternalLoginConfirmation(ExternalLoginConfirmationViewModel model, string returnUrl)

{

if (User.Identity.IsAuthenticated)

{

return RedirectToAction("Index", "Manage");

}

if (ModelState.IsValid)

{

// Get the information about the user from the external login provider

var info = await AuthenticationManager.GetExternalLoginInfoAsync();

if (info == null)

{

return View("ExternalLoginFailure");

}

var user = new ApplicationUser { UserName = model.Email, Email = model.Email };

var result = await UserManager.CreateAsync(user);

if (result.Succeeded)

{

result = await UserManager.AddLoginAsync(user.Id, info.Login);

if (result.Succeeded)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

return RedirectToLocal(returnUrl);

}

}

AddErrors(result);

}

ViewBag.ReturnUrl = returnUrl;

return View(model);

}

//

// POST: /Account/LogOff

[HttpPost]

public ActionResult LogOff()

{

AuthenticationManager.SignOut(DefaultAuthenticationTypes.ApplicationCookie);

return RedirectToAction("Index", "Home");

}

//

// GET: /Account/ExternalLoginFailure

[AllowAnonymous]

public ActionResult ExternalLoginFailure()

{

ViewBag.Languages = GetLanguage();

return View();

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

if (\_userManager != null)

{

\_userManager.Dispose();

\_userManager = null;

}

if (\_signInManager != null)

{

\_signInManager.Dispose();

\_signInManager = null;

}

}

base.Dispose(disposing);

}

private string GetLanguage()

{

return string.IsNullOrEmpty(HttpContext.Request.Cookies.Get("language")?.Value) ? "En" : HttpContext.Request.Cookies.Get("language").Value;

}

#region Helpers

// Used for XSRF protection when adding external logins

private const string XsrfKey = "XsrfId";

private IAuthenticationManager AuthenticationManager

{

get

{

return HttpContext.GetOwinContext().Authentication;

}

}

private void AddErrors(IdentityResult result)

{

foreach (var error in result.Errors)

{

ModelState.AddModelError("", error);

}

}

private ActionResult RedirectToLocal(string returnUrl)

{

if (Url.IsLocalUrl(returnUrl))

{

return Redirect(returnUrl);

}

return RedirectToAction("Index", "Home");

}

internal class ChallengeResult : HttpUnauthorizedResult

{

public ChallengeResult(string provider, string redirectUri)

: this(provider, redirectUri, null)

{

}

public ChallengeResult(string provider, string redirectUri, string userId)

{

LoginProvider = provider;

RedirectUri = redirectUri;

UserId = userId;

}

public string LoginProvider { get; set; }

public string RedirectUri { get; set; }

public string UserId { get; set; }

public override void ExecuteResult(ControllerContext context)

{

var properties = new AuthenticationProperties { RedirectUri = RedirectUri };

if (UserId != null)

{

properties.Dictionary[XsrfKey] = UserId;

}

context.HttpContext.GetOwinContext().Authentication.Challenge(properties, LoginProvider);

}

}

#endregion

}

}using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace ElectronicSchem.Controllers

{

public class ErrorsController : Controller

{

// GET: Errors

public ActionResult NotFound()

{

ViewBag.Languages = GetLanguage();

return View();

}

public ActionResult Error()

{

ViewBag.Languages = GetLanguage();

return View();

}

private string GetLanguage()

{

return string.IsNullOrEmpty(HttpContext.Request.Cookies.Get("language")?.Value) ? "En" : HttpContext.Request.Cookies.Get("language").Value;

}

}

}using System;

using System.Linq;

using System.Threading.Tasks;

using System.Web;

using System.Web.Mvc;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.Owin;

using Microsoft.Owin.Security;

using ElectronicSchem.Models;

using System.IO;

using CloudinaryDotNet;

using CloudinaryDotNet.Actions;

namespace ElectronicSchem.Controllers

{

[Authorize]

public class ManageController : Controller

{

private ApplicationSignInManager \_signInManager;

private ApplicationUserManager \_userManager;

private Account account = new Account(

"maslovskialexey@mail.ru",

"344197536716973",

"A73t\_G\_1M7mahpdkQ6YLQqJCo\_U");

public ManageController()

{

}

[HttpPost]

public void MuteUser(string userId, bool mute)

{

var db = new ApplicationDbContext();

db.Users.FirstOrDefault(x => x.Id == userId).isMuted = mute;

db.SaveChanges();

}

[HttpPost]

public void ChangeImage(string img)

{

var db = new ApplicationDbContext();

var userId = GetCurrentUser().Id;

db.Users.FirstOrDefault(x => x.Id == userId).PhotoLink = img;

db.SaveChanges();

}

public ManageController(ApplicationUserManager userManager, ApplicationSignInManager signInManager)

{

UserManager = userManager;

SignInManager = signInManager;

}

public ApplicationSignInManager SignInManager

{

get

{

return \_signInManager ?? HttpContext.GetOwinContext().Get<ApplicationSignInManager>();

}

private set

{

\_signInManager = value;

}

}

public ApplicationUserManager UserManager

{

get

{

return \_userManager ?? HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();

}

private set