**Оглавление**

[ВВЕДЕНИЕ 4](#_Toc450909731)

[1 АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПС 5](#_Toc450909732)

[1.1 Сравнение аналогов 5](#_Toc450909733)

[1.2 Постановка задач 5](#_Toc450909734)

[1.3 Выбор модели жизненного цикла 5](#_Toc450909735)

[2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 7](#_Toc450909736)

[2.1 Описание функциональности ПС 7](#_Toc450909737)

[2.2 Спецификация 7](#_Toc450909738)

[3 РАЗРАБОТКА ОСНОВНЫХ КЛАССОВ И МЕТОДОВ 8](#_Toc450909739)

[3.1 Создание графического представления программы 8](#_Toc450909740)

[3.2 Применение стилей в приложении 8](#_Toc450909741)

[3.3 Файл представления \_Layout.cshtml 8](#_Toc450909742)

[3.4 Файл представления Index.cshtml 9](#_Toc450909743)

[3.5 Класс HomeController 10](#_Toc450909744)

[3.6 Метод SelectField 10](#_Toc450909745)

[3.7 Метод EmailList 11](#_Toc450909746)

[3.8 Метод Message 12](#_Toc450909747)

[3.9 Метод SendMessage 12](#_Toc450909748)

[3.10 Метод SelectValues 12](#_Toc450909749)

[3.11 Класс EmailSender 13](#_Toc450909750)

[3.12 Класс SqlRequest 13](#_Toc450909751)

[4 ТЕСТИРОВАНИЕ И АНАЛИЗ РЕЗУЛЬТАТОВ 15](#_Toc450909752)

[4.1 Проверка выборки полей из базы данных 15](#_Toc450909753)

[4.2 Фильтрация адресов 15](#_Toc450909754)

[4.3 Отправка AJAX запроса на сервер по нажатию на чекбокс 16](#_Toc450909755)

[4.4 Анализ результатов 16](#_Toc450909756)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 17](#_Toc450909757)

[5.1 Использование приложения 17](#_Toc450909758)

[ЗАКЛЮЧЕНИЕ 19](#_Toc450909759)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20](#_Toc450909760)

[ПРИЛОЖЕНИЕ А 22](#_Toc450909762)

[ПРИЛОЖЕНИЕ Б. Блок-схема метода генерации sql запроса 27](#_Toc450909763)

# ВВЕДЕНИЕ

С развитием информационных технологий автоматизация приходит во все сферы деятельности человека. Современные достижения в области вычислительной техники, программного обеспечения, математических методов распространяются на управление разными объектами, в том числе и объектами социальной природы. Информационные системы такого типа принципиально не могут быть автоматическими. Объясняется это тем, что органической составной частью в них выступают люди с их неформальным мышлением, чувствами и опытом, являющиеся источниками первичной информации и потребителями результатов ее обработки. Такие системы называются автоматизированными и являются одним из важных направлений применения информационных технологий.

Каждый автоматизируемый процесс, предприятие рассматривается как система с управлением, включающая в себя управляющую систему, объект управления и систему связи. Проблема организации системы связи широко распространена и особенно трудно решаема на крупных предприятиях, имеющих в своем составе большое число подразделений. Для эффективного управления различными видами деятельности необходима организация процессов передачи внешних и внутренних информационных потоков в кратчайшие сроки непосредственным исполнителям (объектам управления); возврата результатов деятельности в управляющую систему. Своевременное и актуальное получение сигналов обратной связи очень важно, так как позволяет прослеживать состояние системы с управлением при воздействии на неё каких-либо факторов, в том числе факторов внешней среды, и в зависимости от этого принимать решения о дальнейших действиях.

Система связи может быть реализована различными методами, технологиями и средствами, в том числе и по электронной почте. В настоящее время разработано большое количество почтовых клиентов, позволяющих осуществлять рассылку файлов и сообщений.

Email рассылка - это процедура массовой рассылки почты на большое число email адресов, не требующая каждый раз создавать новое письмо, вводить адрес и жать кнопку «отправить». Забив сообщение в динамический шаблон один раз, оно «разлетится» всем адресатам.

# 1 АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПС

В данном разделе будет сравнение приложения с аналогами, обзор литературы по теме и постановка задач приложения.

## **1.1 Сравнение аналогов**

Наиболее популярные программы автоматизации настраиваемой рассылки сообщений:

* Customer.io, этот сервис позволяет настроить отправку таргетированных сообщений по списку пользователей на базе их действий с целевым продуктом, делая эти письма более индивидуальными и персонализированными. Кроме того, Customer.io предлагает возможности проведения сегментации, а также функции по разработке разнообразных отчетов и сплит-тестирования, которые необходимо для оптимизации почтового маркетинга;
* ePochta Mailer, программа для рассылки писем обладающая широким функционалом;
* [WMmail](http://livesurf.ru/pochtovye_rassylki/1770-wmmail-lider-runeta.html), программа для рассылки писем;

Эти программы отправляют большое количество сообщений по указанным адресам пользователей, все они предоставляют безопасную отправку сообщений по протоколу SSL, имеют встроенный в себя SMTP-сервер, но они не предоставляют достаточно гибкой настройки адресатов.

## **1.2 Постановка задач**

Задача данной работы заключается в следующем: разработать программное средство автоматизации настраиваемой рассылки сообщений. Для этого будет достигнуто выполнение следующих условий:

* удобная для пользования форма настройки фильтрации email адресов;
* возможность написания сообщений для пользователей-адресатов и их отправка по email адресам;
* автоматический разбор таблиц в базе данных.

Отфильтрованные email адреса будут храниться в сессии.

Для работы с базой данный применялась технология Entity Framework 6.

## **1.3 Выбор модели жизненного цикла**

Так как проект по объему относится к типу «небольшой», а также требования будут изменяться по ходу разработки, то подходящие модели жизненного цикла приведены ниже.

Таблица 1.1 – Модели ЖЦ

|  |  |
| --- | --- |
| Вид модели | Качество данной модели |
| Инкрементная | + Добавление функции с каждым инкрементом.  - Возможны ситуации, требующие добавления сразу нескольких взаимосвязанных функций. |
| На основе ранее созданных компонентов | + Снижает время разработки.  - Искажаются требования. |
| Спиральная | +Быстрое получение результата.  +Изменяющиеся требования — не проблема.  - Усложнённая структура разработки. |

Из таблицы видно, что наиболее подходящей моделью жизненного цикла является спиральная модель, так как в ходе проекта ожидается возможное изменение требований.

Отличительной особенностью этой модели является специальное внимание к рискам, влияющим на организацию жизненного цикла. Каждый виток спирали соответствует созданию фрагмента или версии программного обеспечения, на нем уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали.

# 2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

В данном разделе будет проведен анализ требований к ПС.

## **2.1 Описание функциональности ПС**

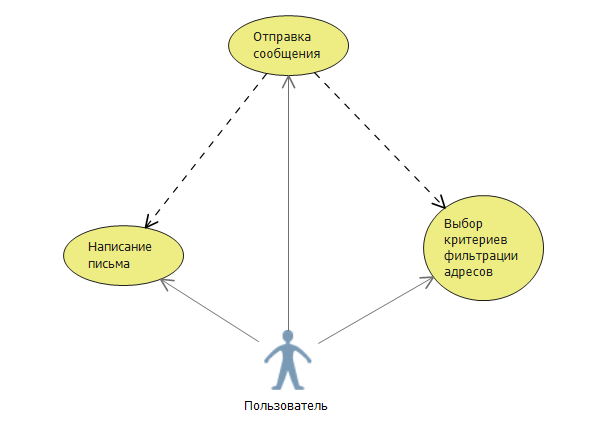


Рисунок 2.1 – UML Use-case -диаграмма взаимодействия пользователя

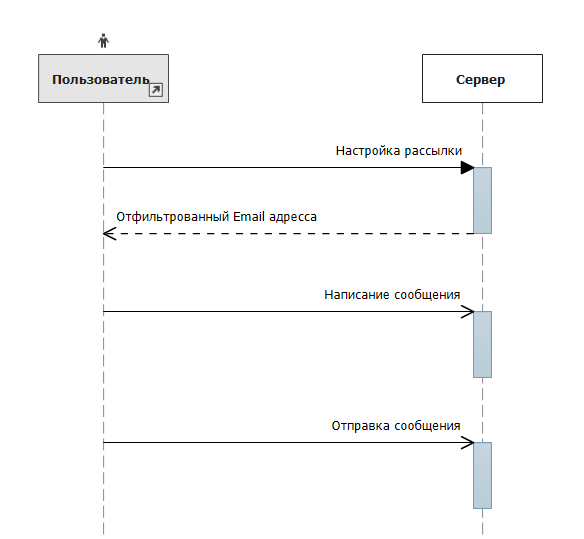


Рисунок 2.2 – UML Sequence -диаграмма связи пользователя с сервером

## **2.3 Спецификация функциональных требований.**

Функциональные требования:

* программное средство должно производить автоматический разбор таблиц в базе данных;
* сообщения должны отправляться асинхронно;
* запросы о настройках рассылки должна производиться без перезагрузки страницы в браузере.

Нефункциональные требования:

* вывод сообщение об ошибке;

# 3 РАЗРАБОТКА ОСНОВНЫХ КЛАССОВ И МЕТОДОВ

Для создания программы были использованы язык программирования С# и среда разработки Microsoft Visual Studio 2015. Использование данной среды позволяет разрабатывать качественные приложения и библиотеки для семейства операционных систем Windows. Целью курсовой работы являлось создать простой в использовании сайт автоматизации настраиваемой рассылки.

## **3.1 Создание графического представления программы**

Поскольку программа будет представлять собой сайт, необходимо сделать интуитивно понятный и привлекательный интерфейс. Для создания такого интерфейса были использованы следующие технологии и языки:

* HTML – язык разметки веб-страницы.
* СSS-код – список инструкций для браузера, – как и где отображать элементы веб-страницы, написанный особым образом.
* JavaScript – это язык сценариев, с его помощью в HTML-документ (страницу сайта) можно встроить небольшие программы, позволяющие упростить работу с HTML-страницей и сделать ее более красивой.
* Razor – движок представлений в ASP.NET.

Представление приложения состоит из мастер-страницы \_Layout.cshtml. Подключаемый контент находится в файле index.cshtml.

## **3.2 Применение стилей в приложении**

Чтобы интерфейс приложения являлся привлекательным, необходимо использовать CSS-код.

После установки стилей необходимо подключить их в файл \_Layout.cshtml:

*Листинг 3.1 –* Подключение стилей

<link href="~/Content/Styles/LayoutStyle.css" rel="stylesheet" />

<link href="~/Content/Styles/home.css" rel="stylesheet" />

## **3.3 Файл представления \_Layout.cshtml**

Страница \_Layout.cshtml является мастер-страницей и состоит из 2х частей header и bodyContainer.

В части header находится название проекта.

Часть bodyContainer разделяется на 3 части в которой подключается содержимое представления Index.cshtml:

*Листинг 3.2 –* Подключение Index.cshtml

<div class="bodyContainer">

@RenderSection(name: "LeftContainer", required: false)

@RenderSection(name: "CenterContainer", required: false)

@RenderSection(name: "RightContainer", required: false)</div>

## **3.4 Файл представления Index.cshtml**

В файле представления Index.cshtml находится основной доступ к функциональности приложения, такой как: настройка рассылки email адресов, написание и отправка сообщения.

Содержимое страницы разделяется на 3 части. Каждая часть использует частичное представление: левая часть содержит форму, в которой можно выбирать поля из базы данных. Эти поля будут отображаться на центральной части страницы. Центральная часть страницы содержит 2 вкладки. В первой вкладке производится настройка рассылки сообщений, эта форма представляет собой окно, в котором находится список значений полей базы данных и чекбокс для отправки данных на сервер.

Чекбокс имеет следующую сигнатуру:

*Листинг 3.3 –* Сигнатура чекбокса

<input type="checkbox" class="testcheck" onchange="checkboxChange(this)" value="@ViewBag.TableName|@ViewBag.FieldName|@item.Key" @item.Value>

В нём указывается событие onchange, обработка которого производит запрос на сервер и передаёт необходимую информацию для фильтрации email адресов.

*Листинг 3.4 –* Обработка нажатия по чекбоксу

<script>

function checkboxChange(checkbox) {

makeRequest(checkbox.value, checkbox.checked ? "checked" : "unchecked");

}

function makeRequest(value, state) {

var part = value.split("|");

$.ajax({

url: "@Url.Action("EmailList")",

type: "GET",

data: "table=" + part[0] + "&field=" + part[1] + "&value=" + part[2] + "&state=" + state,

success: function (responce) {

$("#emailList").html(responce);

}

});

}

</script>

Во второй вкладке находится форма для написания и отправки сообщения:

*Листинг 3.5 –* Код формы сообщения

@using (Html.BeginForm("SendMessage", "Home", FormMethod.Post))

{

<div class="message-wrap">

<input type="text" name="Title" placeholder="Title" />

<textarea name="Text" placeholder="Text"></textarea>

<input type="submit" value="Отправить" />

</div>

}

В правой части находится список отфильтрованных адресов.

*Листинг 3.6 –* Вывод списка email адресов

@foreach (var email in Model)

{

<p>@email</p>

}

## **3.5 Класс HomeController**

Класс HomeController это основной класс для работы программного средства. В классе присутствуют следующие методы:

* SelectField(string table, string field);
* EmailList();
* Message ();
* SendMessage(Message Message);
* SelectValues(string tableName, string fieldName)

## **3.6 Метод SelectField**

*Листинг 3.7 –* Заполнение статического поля полями из базы данных

public ActionResult SelectField(string table, string field)

{

ViewBag.TableName = table;

ViewBag.FieldName = field;

var list = SelectValues(table, field);

Dictionary<string, string> valueContainer = new Dictionary<string, string>();

foreach(var value in list)

{

if (!fetch.ContainsKey(value))

{

fetch.Add(value, "unchecked");

valueContainer.Add(value, "unchecked");

}

else

{

valueContainer.Add(value, fetch[value]);

}

}

Dictionary<string, Dictionary<string, string>> fieldContainer;

if (!tableContainer.ContainsKey(table))

{

fieldContainer = new Dictionary<string, Dictionary<string, string>>();

fieldContainer.Add(field, fetch);

tableContainer.Add(table, fieldContainer);

}

else if (!tableContainer[table].ContainsKey(field))

{

fieldContainer = new Dictionary<string, Dictionary<string, string>>();

fieldContainer.Add(field, valueContainer);

tableContainer[table] = fieldContainer;

}

else

{

tableContainer[table][field] = valueContainer;

}

return PartialView("\_TableData", valueContainer);

}

В этом методе производится заполнение статической переменной tableContainer данными из базы данных для дальнейшей фильтрации email адресов и отслеживания состояния чекбоксов. В начале метода группируются значения полей, далее собираются имена полей и имена таблиц. В конце в страницу Index.cshtml передаётся частичное представление \_TableData с переданным в неё параметром значений полей выбранной таблицы.

## **3.7 Метод EmailList**

В методе EmailList происходит фильтрация email адресов:

*Листинг 3.8 –* Код метода EmailList

public PartialViewResult EmailList()

{

string table = Request.QueryString["table"];

string field = Request.QueryString["field"];

string value = Request.QueryString["value"];

string state = Request.QueryString["state"];

if (table != null && field != null && value != null && state != null)

{

if (fetch.ContainsKey(value))

{

fetch[value] = state;

tableContainer[table][field][value] = state;

}

}

List<string> emails = new List<string>();

var results = context.Database.SqlQuery<Abiturient>(new SqlRequest(tableContainer).GenerateSqlString());

foreach(dynamic i in results)

{

emails.Add(i.Email);

}

Session["EmailList"] = emails;

return PartialView("\_EmailList", Session["EmailList"]);

}

В методе извлекается информация из GET запроса, который приходит после нажатия по чекбоксу. На основе этой информации меняется состояние tableContainer. По состоянию tableContainer фильтруются email адреса.

## **3.8 Метод Message**

Метод Message подключает частичное представление \_Message.cshtml.

## **3.9 Метод SendMessage**

Метод SendMessage вызывается как отклик на нажатие по кнопке отправки сообщения. В качестве параметра методу передаётся привязанная модель Message в которой содержится информация о состоянии письма. Метод проходится циклом по всем отфильтрованным адресам и вызывает их асинхронную отправку.

## 

## **3.10 Метод SelectValues**

*Листинг 3.9 –* Код метода SelectValue

List<string> mylist = new List<string>();

var list = ((IEnumerable<dynamic>)context.GetType().GetProperty(tableName).GetValue(context)).ToList();

foreach (var item in list)

{

if (!mylist.Contains(item.GetType().GetProperty(fieldName).

GetValue(item).ToString()))

{

mylist.Add(item.GetType().GetProperty(fieldName).GetValue(item).ToString());

}

}

return mylist;

Из кода метода видно, что сначала происходит получение экземпляра таблицы данных с помощью рефлексии, далее в цикле идёт формирование списка значений данных.

## **3.11 Класс EmailSender**

Статический класс EmailSender является классом, в котором определён один публичный метод SendMessage, который отправляет сообщение по переданному email адресу.

*Листинг 3.1 –* Отправка сообщения

public static void SendMessage(string sender, string senderName, string recipient, string subject, string body)

{

MailMessage message = new MailMessage()

{

From = new MailAddress(sender, senderName),

Subject = subject,

Body = body,

IsBodyHtml = true

};

message.To.Add(new MailAddress(recipient));

SmtpClient smtp = new SmtpClient("smtp.gmail.com", 25);

smtp.EnableSsl = true;

smtp.Credentials = new NetworkCredential("sashachechoitko@gmail.com", "4qz1001hp3posa");

smtp.SendCompleted += MailDeliveryComplete;

smtp.SendAsync(message, message);

}

## **3.12 Класс SqlRequest**

В классе SqlRequest генерируется sql запрос к базе данных для выборки email адресов в соответствии настроенным параметрам.

*Листинг 3.10 –* Код генерации строки запроса

public string GenerateSqlString()

{

string request = baseBody;

bool flag = false;

if (tableContainer.Count > 0)

request += " WHERE ";

foreach(var table in tableContainer)

{

flag = true;

foreach (var field in table.Value)

{

foreach (var value in field.Value)

{

if (value.Value == "checked")

{

request += table.Key + "." + field.Key + " = '" + value.Key+"' OR

}

}

request = request.Remove(request.Length - 4, 4);

request += "AND ";

}

request = request.Remove(request.Length - 4, 4);

request += " AND ";

}

if (flag)

{

request = request.Remove(request.Length - 5, 5);

}

return request;

}

# 4 ТЕСТИРОВАНИЕ И АНАЛИЗ РЕЗУЛЬТАТОВ

В этом разделе описана проверка функций, реализованных в программе и анализ результатов в соответствии с ТЗ.

## **4.1 Проверка выборки полей из базы данных**

Для выборки полей из базы данных пользователь должен кликнуть по одной из ссылок в левой части страницы приложения.

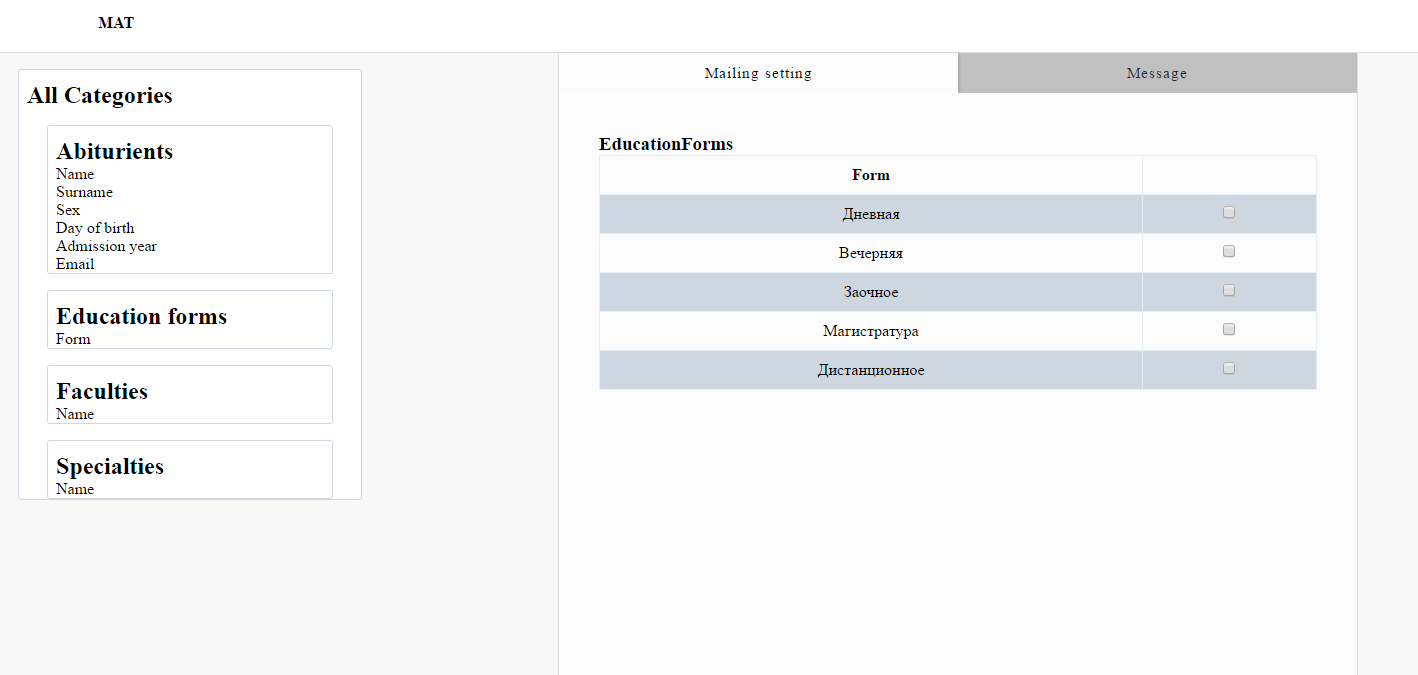


Рисунок 4.1 – Поведение программы в случае нажатия на ссылку Form

## **4.2 Фильтрация адресов**

В результате нажатий на чекбоксы происходит фильтрация данных

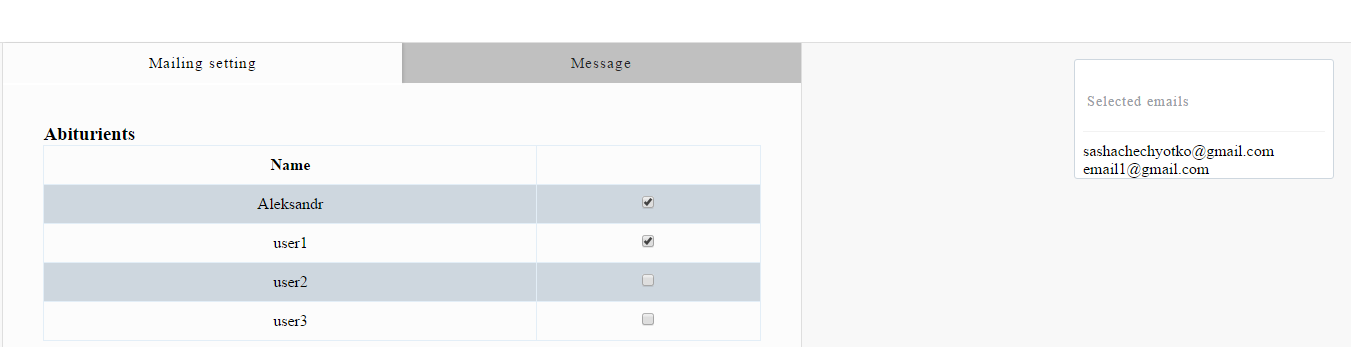


Рисунок 4.4 – Отфильтрованные данные

## 

## **4.3 Отправка AJAX запроса на сервер по нажатию на чекбокс**

URL адрес должен содержать 4 GET параметра. Если подменить данные выводится сообщение об ошибке.

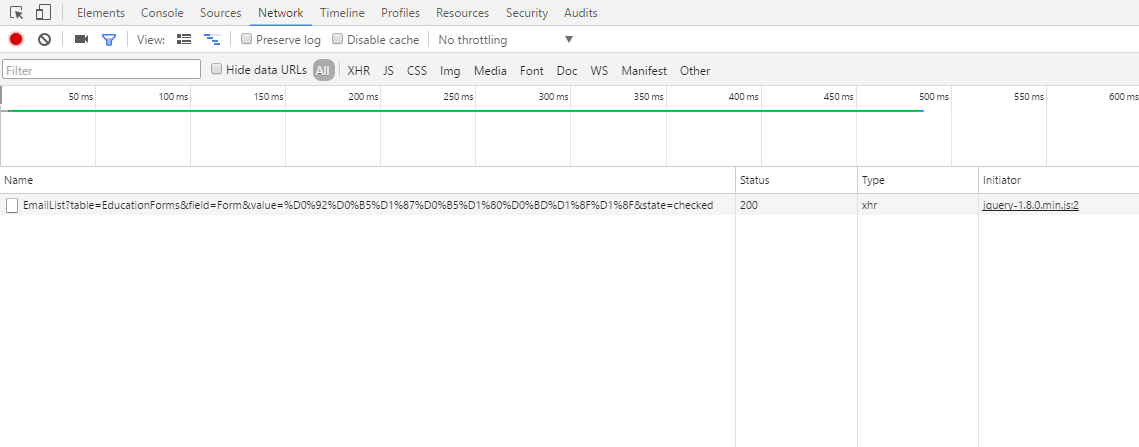


Рисунок 4.9 – Отправляемый запрос при нажатии на чекбокс

## **4.4 Анализ результатов**

В результате запуска и использования созданного приложения были проверены следующие возможности, представленные программой:

* разбор полей базы данных;
* правильность фильтрации email адресов;
* возможность написания сообщения;
* асинхронная отправка email сообщений.

Вывод: данное приложение позволяет быстро настроить адресатов для рассылки сообщений на электронные ящики пользователей.

# 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## **5.1 Использование приложения**

После открытия сайта с приложением, у вас будет возможность настраивать рассылку сообщений. Для этого вам понадобится нажать на ссылку в левой части страницы.

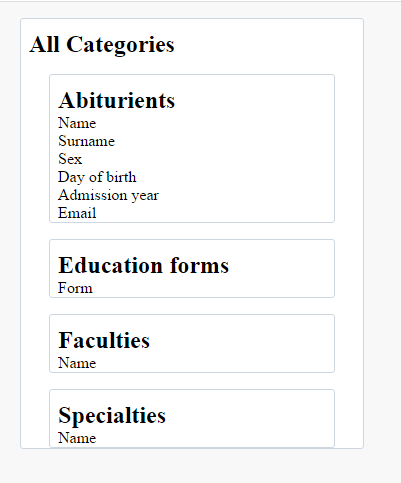


Рисунок 5.1 – Вид формы для выборки полей из базы данных

Для написания сообщения необходимо кликнуть по вкладке Message в центре страницы.

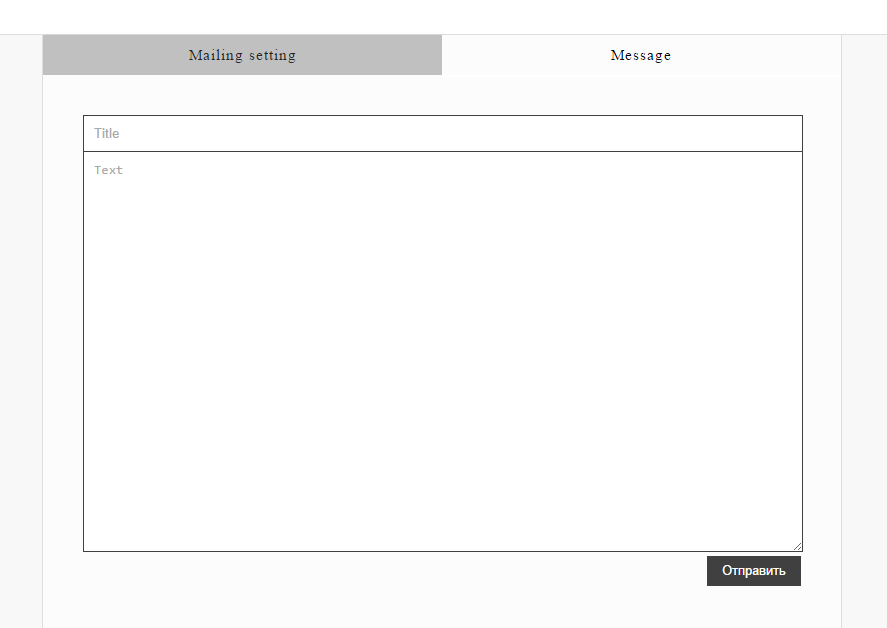


Рисунок 5.2 – Вид формы написания сообщения

Для изменения параметров рассылки сообщений необходимо кликнуть на вкладке Mailing Setting.

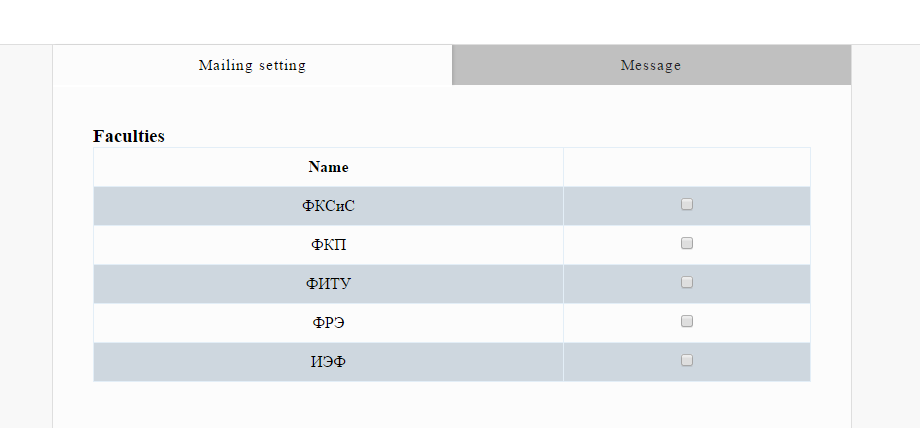


Рисунок 5.3 – Вид настройки рассылки писем

# ЗАКЛЮЧЕНИЕ

В ходе данной работы было создано программное средство автоматизации настраиваемой рассылки сообщений. Больших трудностей в ходе работы замечено не было. Приложение получилось функциональным и удобным в использовании.

Так как реализация некоторых частей алгоритма приложения требовала интересных логических решений, к результату работы можно также отнести развитие и приобретение опыта у автора в целом.

В ходе работы были достигнуты следующие цели:

* Создание комфортного и интуитивно понятного интерфейса с помощью CSS и JavaScript;
* Отправка асинхронных сообщений;
* Создание приложения на ASP.NET MVC 5;
* Применение технологии AJAX;
* Использование Entity Framework 6 для работы с базой данных;
* Использование SQL языка для запросов к базе данных.

Дальнейшая модификация программы подразумевает под собой расширение взаимодействия с базой данных, а также создание более гибкой настройки рассылки сообщений, использование прокси-серверов и сторонних сервисов для обмана спам фильтров почтовых серверов, улучшение программного интерфейса, а также добавление регистрации пользователей.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Р. Мартин. Чистый код. Создание, анализ и рефакторинг [Текст]: учебное пособие; [Питер](https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D1%82%D0%B5%D1%80_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)), 2012г., 256с.

[2] Д. Рихтер. Программирование на платформе Microsoft .NET Framework [Текст]: учебное пособие; Изд-во: Русская Редакция, 2002 г., 486с.

[3] Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. [Приемы объектно-ориентированного проектирования. Паттерны проектирования](https://ru.wikipedia.org/wiki/Design_Patterns). — [Addison-Wesley](https://ru.wikipedia.org/wiki/Addison-Wesley), [«Питер»](https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D1%82%D0%B5%D1%80_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)), 1994. —  395с.

[4] Основы JavaScript [Электронный ресурс] – Электронные данные. – Режим доступа: <http://learn.javascript.ru/first-steps> <https://ru.wikipedia.org/wiki/7-Zip>

[5] Основы jQuery [Электронный ресурс] – Электронные данные. – Режим доступа: <http://www.w3schools.com/jquery/>

[6] Основы HTML [Электронный ресурс] – Электронные данные. – Режим доступа: <http://www.w3schools.com/html/>

[7] Основы CSS [Электронный ресурс] – Электронные данные. – Режим доступа: <http://www.w3schools.com/css/>

[8] Онлайн-книга Изучаем ASP.NET MVC 4

[Электронный ресурс]– Режим доступа: <http://www.metanit.com/sharp/mvc/>

[9] Статья о метапрограммировании [Электронный ресурс]. – Режим доступа: https://habrahabr.ru/post/137446/

[10] Bootstrap свободный набор инструментов для создания сайтов и веб-приложений. [Электронный ресурс] – Электронные данные. – Режим доступа: <http://www.getbootstrap.com/>

[11] Yukihiro Matsumoto, Head First Design Patterns [Текст]: учебное пособие. -  М.:O’REILLY, 2014. - 696с.

[12] Using the CodeDOM [Электронный ресурс]. – Электронные данные. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/y2k85ax6(v=vs.110).aspx.

[13] Design Patterns in c# [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.dofactory.com/net/design-patterns

[14] Синтаксис движка Razor [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.w3schools.com/aspnet/razor\_syntax.asp

[15] ASP.NET | Файл web.config [Электронный ресурс]. –Электронные данные. – Режим доступа: http://professorweb.ru/my/ASP\_NET/base/level4/

[16] Нейгел, К. C# 5.0 и платформа .NET 4.5 для профессионалов[Текст] / К. Нейгел, Б. Ивьен, Д. Глинн, К. Уотсон, М. Скиннер. – М.: Диалектика, 2014.

[17] Объект ViewData [Электронный ресурс]. –Электронные данные. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/ss7fbaez%28v=vs.110%29.aspx

[18] int (справочник по C#) [Электронный ресурс]. – Электронные данные. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/5kzh1b5w.aspx

[19] Основы Ajax. [Электронный ресурс]. – Электронные данные. – Режим доступа: [https://www.web4you.com/developerworks/ru/library/ajax /](https://www.web4you.com/developerworks/ru/library/ajax%20/)

[20] Неизвестный автор, C# Beginner's Course [Текст]: учебное пособие. -  М.:Serfs, 2015. - 188с.

# ПРИЛОЖЕНИЕ А

Исходный код программы

Объектная модель базы данных приложения

public class Abiturient

{

public int Id { get; set; }

[Required]

public string Name { get; set; }

[Required]

public string Surname { get; set; }

[Required]

public string Email { get; set; }

[Required]

public string Sex { get; set; }

[Required]

public DateTime DateOfBirth { get; set; }

[Required]

public int AdmissionYear { get; set; }

[Required]

public int EducationFormId { get; set; }

public virtual EducationForm EducationForm { get; set; }

[Required]

public int FacultyId { get; set; }

public virtual Faculty Faculty { get; set; }

[Required]

public int SpecialtyId { get; set; }

public virtual Specialty Specialty { get; set; }

}

public class EducationForm

{

public int Id { get; set; }

[Required]

public string Form { get; set; }

public virtual ICollection<Abiturient> Abiturients { get; set; }

public EducationForm()

{

Abiturients = new List<Abiturient>();

}

}

public class Faculty

{

public int Id { get; set; }

[Required]

public string Name { get; set; }

public virtual ICollection<Abiturient> Abiturients { get; set; }

public Faculty()

{

Abiturients = new List<Abiturient>();

}

}

public class Specialty

{

public int Id { get; set; }

[Required]

public string Name { get; set; }

public virtual ICollection<Abiturient> Abiturients { get; set; }

public Specialty()

{

Abiturients = new List<Abiturient>();

}

}

Класс HomeController

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using MAT.TableModels;

using MAT.Models;

using MAT.DAL;

using MAT.Services;

namespace MAT.Controllers

{

public class HomeController : Controller

{

private MATContext context = new MATContext();

private static Dictionary<string, string> fetch = new Dictionary<string, string>();

private static Dictionary<string, Dictionary<string, Dictionary<string,string>>> tableContainer = new Dictionary<string, Dictionary<string, Dictionary<string, string>>>();

public ActionResult Index()

{

return View();

}

public ActionResult SelectField(string table, string field)

{

ViewBag.TableName = table;

ViewBag.FieldName = field;

var list = SelectValues(table, field);

Dictionary<string, string> valueContainer = new Dictionary<string, string>();

foreach(var value in list)

{

if (!fetch.ContainsKey(value))

{

fetch.Add(value, "unchecked");

valueContainer.Add(value, "unchecked");

}

else

{

valueContainer.Add(value, fetch[value]);

}

}

Dictionary<string, Dictionary<string, string>> fieldContainer;

if (!tableContainer.ContainsKey(table))

{

fieldContainer = new Dictionary<string, Dictionary<string, string>>();

fieldContainer.Add(field, valueContainer);

tableContainer.Add(table, fieldContainer);

}

else if (!tableContainer[table].ContainsKey(field))

{

tableContainer[table].Add(field, valueContainer);

}

else

{

tableContainer[table][field] = valueContainer;

}

return PartialView("\_TableData", valueContainer);

}

public PartialViewResult EmailList()

{

string table = Request.QueryString["table"];

string field = Request.QueryString["field"];

string value = Request.QueryString["value"];

string state = Request.QueryString["state"];

if (table != null && field != null && value != null && state != null)

{

if (fetch.ContainsKey(value))

{

fetch[value] = state;

tableContainer[table][field][value] = state;

}

}

List<string> emails = new List<string>();

var results = context.Database.SqlQuery<Abiturient>(new SqlRequest(tableContainer).GenerateSqlString());

foreach(dynamic i in results)

{

emails.Add(i.Email);

}

Session["EmailList"] = emails;

return PartialView("\_EmailList", Session["EmailList"]);

}

public PartialViewResult Message()

{

return PartialView("\_Message");

}

public ActionResult SendMessage()

{

return View(new Message());

}

[HttpPost]

public ActionResult SendMessage(Message Message)

{

ICollection<string> emails = (ICollection<string>)Session["EmailList"];

foreach (string recipient in emails)

{

try

{

EmailSender.SendMessage("sashachechoitko@gmail.com", "Sasha Chechoitko", recipient, Message.Title, Message.Text);

}

catch (Exception ex)

{

ViewBag.Error = ex.Message;

}

}

return RedirectToAction("Index");

}

private List<string> SelectValues(string tableName, string fieldName)

{

List<string> mylist = new List<string>();

var list = ((IEnumerable<dynamic>)context.GetType().GetProperty(tableName).GetValue(context)).ToList();

foreach (var item in list)

{

if (!mylist.Contains(item.GetType().GetProperty(fieldName).GetValue(item).ToString()))

{

mylist.Add(item.GetType().GetProperty(fieldName).GetValue(item).ToString());

}

}

return mylist;

}

}

}

Класс SqlRequest

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace MAT.Models

{

public class SqlRequest

{

private string baseBody = "SELECT \* FROM Abiturients JOIN EducationForms ON EducationForms.Id = Abiturients.EducationFormId JOIN Faculties ON Abiturients.FacultyId = Faculties.Id JOIN Specialties ON Specialties.Id = Abiturients.SpecialtyId";

private Dictionary<string, Dictionary<string, Dictionary<string, string>>> tableContainer = new Dictionary<string, Dictionary<string, Dictionary<string, string>>>();

public SqlRequest(Dictionary<string, Dictionary<string, Dictionary<string, string>>> tableContainer)

{

this.tableContainer = tableContainer;

}

public string GenerateSqlString()

{

string request = baseBody;

bool flag = false;

if (tableContainer.Count > 0)

request += " WHERE ";

foreach(var table in tableContainer)

{

flag = true;

foreach (var field in table.Value)

{

foreach (var value in field.Value)

{

if (value.Value == "checked")

{

request += table.Key + "." + field.Key + " = '" + value.Key+"' OR ";

}

}

request = request.Remove(request.Length - 4, 4);

request += "AND ";

}

request = request.Remove(request.Length - 4, 4);

request += " AND ";

}

if (flag)

{

request = request.Remove(request.Length - 5, 5);

}

return request;

}

public void ParseRequest(Dictionary<string, string> fetch)

}}

# ПРИЛОЖЕНИЕ Б. Блок-схема метода генерации sql запроса