Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

Индивидуальное домашнее задание

по дисциплине «Технологии разработки WEB-приложений»

«Разработка ASP.NET приложения с использованием ADO.NET»

|  |  |  |
| --- | --- | --- |
| Студент |  | Ковальчук А. Д. |
| Группа | АИ-14 |  |
| Руководитель |  | Лаврухина Т. В. |

Липецк – 2018 г.

Цель работы

Разработать ASP.NET приложения с использованием ADO.NET

Задание

Реализовать веб-сайт с эргономичным расположением управляющих элементов, который помимо функций решения задачи темы курсовой работы должен включать в себя:

1. Реализацию подсистем аутентификации и авторизации пользователей, с возможностью распределения ролей пользователей.
2. Использование БД объектов предметной области.
3. Реализацию как минимум одного пользовательского веб-элемента управления, работающего без перезагрузки страницы.
4. Обработчик ошибок как пользователя, так и выполнения, их сохранением в журнал событий.

# Результаты выполнения работы

## Описание предметной области

В качестве предметной области выбран Web-сервис для ведения и блогов и чтения статей.

## Ход выполнения работы

### Подсистема аутентификации и авторизации пользователей

Подсистема аутентификации и авторизации пользователей с возможностью распределения ролей установлена и настроена с помощью ASP.NET Identity.

Использование Identity сильно упростило разработку Web-приложения, позволило легко управлять пользователями и привязанными к ним данными.

Код контроллера, который отвечает за авторизацию и регистрацию представлен ниже:

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 Lab1.Models;

namespace Lab1.Controllers

{

[Authorize]

public class AccountController : Controller

{

ApplicationSignInManager \_signInManager;

ApplicationUserManager \_userManager;

public AccountController()

{

}

public AccountController(ApplicationUserManager userManager, ApplicationSignInManager signInManager)

{

UserManager = userManager;

SignInManager = signInManager;

}

public ApplicationSignInManager SignInManager

{

get

{

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

}

set

{

\_signInManager = value;

}

}

public ApplicationUserManager UserManager

{

get

{

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

}

set

{

\_userManager = value;

}

}

//

// GET: /Account/Login

[AllowAnonymous]

public ActionResult Login(string returnUrl)

{

ViewBag.ReturnUrl = returnUrl;

return View();

}

//

// POST: /Account/Login

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

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

{

if (!ModelState.IsValid)

return View(model);

// Сбои при входе не приводят к блокированию учетной записи

// Чтобы ошибки при вводе пароля инициировали блокирование учетной записи, замените на 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, model.RememberMe });

case SignInStatus.Failure:

default:

ModelState.AddModelError("", "Неудачная попытка входа.");

return View(model);

}

}

//

// GET: /Account/Register

[AllowAnonymous]

public ActionResult Register()

{

return View();

}

//

// POST: /Account/Register

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

var user = new ApplicationUser {

LastName = model.LastName,

FirstName = model.FirstName,

MiddleName = model.MiddleName,

UserName = model.Email,

BirthDay = model.BirthDay,

Email = model.Email,

Sex = model.Sex,

PhoneNumber = model.PhoneNumber

};

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

if (result.Succeeded)

{

await UserManager.AddToRoleAsync(user.Id, "user");

await UserManager.SetLockoutEnabledAsync(user.Id, true);

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

// Дополнительные сведения о том, как включить подтверждение учетной записи и сброс пароля, см. по адресу: http://go.microsoft.com/fwlink/?LinkID=320771

// Отправка сообщения электронной почты с этой ссылкой

// 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, "Подтверждение учетной записи", "Подтвердите вашу учетную запись, щелкнув <a href=\"" + callbackUrl + "\">здесь</a>");

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

}

AddErrors(result);

}

// Появление этого сообщения означает наличие ошибки; повторное отображение формы

return View(model);

}

//

// POST: /Account/LogOff

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult LogOff()

{

AuthenticationManager.SignOut(DefaultAuthenticationTypes.ApplicationCookie);

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

}

#region Вспомогательные приложения

// Используется для защиты от XSRF-атак при добавлении внешних имен входа

const string XsrfKey = "XsrfId";

IAuthenticationManager AuthenticationManager

{

get

{

return HttpContext.GetOwinContext().Authentication;

}

}

void AddErrors(IdentityResult result)

{

foreach (var error in result.Errors)

ModelState.AddModelError("", error);

}

ActionResult RedirectToLocal(string returnUrl)

{

if (Url.IsLocalUrl(returnUrl))

return Redirect(returnUrl);

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

}

#endregion

}

}

Далее представлен код моделей, которые связаны с авторизацией и регистрацией.

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace Lab1.Models

{

public class LoginViewModel

{

[Required]

[Display(Name = "Адрес электронной почты")]

[EmailAddress]

public string Email { get; set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Пароль")]

public string Password { get; set; }

[Display(Name = "Запомнить меня")]

public bool RememberMe { get; set; }

}

public class RegisterViewModel

{

[Required]

[Display(Name = "Фамилия")]

public string LastName { get; set; }

[Required]

[Display(Name = "Имя")]

public string FirstName { get; set; }

[Display(Name = "Отчество")]

public string MiddleName { get; set; }

[DataType(DataType.Date)]

[Display(Name = "Дата рождения")]

public DateTime BirthDay { get; set; }

[Required]

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

[Required]

[Display(Name = "Пол")]

public string Sex { get; set; }

[Phone]

[Display(Name = "Телефон")]

public string PhoneNumber { get; set; }

[Required]

[StringLength(100, ErrorMessage = "Значение {0} должно содержать не менее {2} символов.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "Пароль")]

public string Password { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Подтверждение пароля")]

[Compare("Password", ErrorMessage = "Пароль и его подтверждение не совпадают.")]

public string ConfirmPassword { get; set; }

}

}

### База данных

ASP.NET позволяет взаимодействовать с БД, через созданные модели. БД генерируется, когда к ней поступает первый запрос, если она не была уже создана.

Далее представлен код моделей, которые формируют БД и позволяют делать к ней запросы в рамках синтаксиса C#.

Article

public class Article

{

public int ID { get; set; }

[Required]

[Display(Name = "Категория")]

public int CategoryID { get; set; }

[Required]

public string AuthorID { get; set; }

[Required]

public DateTime DT { get; set; }

[Required]

[Display(Name = "Тема")]

public string Subject { get; set; }

[Required]

[Display(Name = "Текст статьи")]

public string Text { get; set; }

}

Category

public class Category

{

public int ID { get; set; }

public string Name { get; set; }

}

Favorites

public class Favorites

{

public int ID { get; set; }

public string UserID {get; set;}

public int ArticleID { get; set; }

}

### Веб-элементы, работающие без перезагрузки страницы

Встроенные в ASP.NET Хелперы позволяют делать простые проверки данных без перезагрузки проверки. Для этого нужно в модели прописать необходимые параметры проверки, а затем использовать во View эту модель с необходимыми хелперами. Тогда, например, при регистрации пользователь будет оповещен о том, что не все поля были заполнены или пароль не соответствует требованиям.

Ниже представлен код части контроллера, которая отвечает за добавление статьи.

//Метод отображения страницы создания статьи

[Authorize]

public ActionResult Create(int? CategoryID)

{

try

{

ViewBag.CategoryForLooking = db.Categorys.ToList();

return View();

}

catch (Exception ex)

{

logger.Debug(DateTime.Now.ToString("dd.MM.yyyy HH:mm:ss") + " in /Blogs/Create : " + ex.Message);

return RedirectToAction("Error", "Shared");

}

}

//Метод создания статьи

[HttpPost]

[Authorize]

public ActionResult Create(Article art)

{

try

{

if (ModelState.IsValid)

{

db.Articles.Add(art);

db.SaveChanges();

return RedirectToAction("Index", "Blogs", new { art.CategoryID, message = MessageStatus.ArticleCreated });

}

ViewBag.CategoryForLooking = db.Categorys.ToList();

return View(art);

}

catch (Exception ex)

{

logger.Debug(DateTime.Now.ToString("dd.MM.yyyy HH:mm:ss") + " in /Blogs/Create POST : " + ex.Message);

return RedirectToAction("Error", "Shared");

}

}

Далее представлен код представления создания статьи.

@using Lab1.Models

@using Microsoft.AspNet.Identity

@model Article

@{

ViewBag.Page2 = "nav-item active";

ViewBag.Hidden = "hidden = \"hidden\"";

}

<h2 class="display-4 mb-3">Создание статьи</h2>

@if (Request.IsAuthenticated)

{

<div class="alert alert-primary col-12" role="alert">

@Html.ValidationSummary("", new { @class = "text-danger mb-2" })

@using (Html.BeginForm("Create", "Blogs", FormMethod.Post))

{

<div class="form-row">

<div class="form-group col-md-3 cd-6">

@Html.LabelFor(m => m.CategoryID)

@Html.DropDownListFor(m => m.CategoryID, new SelectList(ViewBag.CategoryForLooking, "ID", "Name"), "--Выбрать--", new { @class = "form-control" })

</div>

<div class="form-group col-md-9 cd-6">

@Html.LabelFor(m => m.Subject)

@Html.TextBoxFor(m => m.Subject, new { @class = "form-control", @placeholder = "Введите тему статьи" })

</div>

</div>

<div class="form-row">

<div class="form-group mx-2">

@Html.LabelFor(m => m.Text)

@Html.TextAreaFor(m => m.Text, 6, 150, new { @class = "form-control", @placeholder = "Введите или скопируйте сюда текст статьи" })

</div>

</div>

@Html.Hidden("AuthorID", User.Identity.GetUserId())

@Html.Hidden("DT", DateTime.Now)

<input type="submit" class="btn btn-primary mx-2" value="Сохранить" />

}

</div>

}

else

{

<div class="alert alert-danger col-12" role="alert">

Для создания статьи необходимо <a href="/Account/Login">авторизоваться</a>

</div>

}

### Обработчик ошибок

Обрабатывать ошибки в ASP.NET можно сторонней библиотекой NLog. Она является достаточно распространенной. После установки и настройки данной библиотеки, логирование можно производить в разные файлы и всего одной строчкой кода.

Далее представлен метод, в котором происходит логирование ошибок пользователя и выполнения.

[Authorize]

public ActionResult Favorites(int? page, MessageStatus? message)

{

try

{

string CurrentUserID = User.Identity.GetUserId();

if (CurrentUserID == null) return RedirectToAction("Index", "Blogs", new { MessageStatus.UserForFavoritesNotFound });

//Определитель сообщения

ViewBag.Message =

message == MessageStatus.ArticleDeletedFavorites ? "Статья успешно удалена из вашего избранного!"

: message == MessageStatus.DeletedArticleNotFoundFavorites ? "У вас в Избранном нет статьи, которую вы пытаетесь удалить."

: message == MessageStatus.ArticleDeletedFromFaavoritesNotFound ? "Не найдена статья для удаления ее из Избранного."

: message == MessageStatus.ArticleDeleted ? "Статья удалена!"

: message == MessageStatus.DeletedArticleNotFound ? "Статья, которую вы пытаетесь удалить, не существует."

: message == MessageStatus.EditedArticleNotFound ? "Статья, которую вы пытались отредактировать, не существует."

: "";

//Определитель цвета оповещения: 1 - красный, 0 - синий

switch(message)

{

case MessageStatus.DeletedArticleNotFoundFavorites:

ViewBag.MessageCode = 1;

logger.Info(DateTime.Now.ToString("dd.MM.yyyy HH:mm:ss") + " in /Blogs/Favorites : У вас в Избранном нет статьи, которую вы пытаетесь удалить.");

break;

case MessageStatus.ArticleDeletedFromFaavoritesNotFound:

ViewBag.MessageCode = 1;

logger.Info(DateTime.Now.ToString("dd.MM.yyyy HH:mm:ss") + " in /Blogs/Favorites : Не найдена статья для удаления ее из Избранного.");

break;

case MessageStatus.DeletedArticleNotFound:

ViewBag.MessageCode = 1;

logger.Info(DateTime.Now.ToString("dd.MM.yyyy HH:mm:ss") + " in /Blogs/Favorites : Статья, которую вы пытаетесь удалить, не существует.");

break;

case MessageStatus.EditedArticleNotFound:

ViewBag.MessageCode = 1;

logger.Info(DateTime.Now.ToString("dd.MM.yyyy HH:mm:ss") + " in /Blogs/Favorites : Статья, которую вы пытались отредактировать, не существует.");

break;

default:

ViewBag.MessageCode = 0;

break;

}

ViewBag.Categorys = db.Categorys;

ViewBag.Articles = db.Articles;

ViewBag.Authors = db.Users;

List<Favorites> favorites = db.Favorites.Where(x => x.UserID == CurrentUserID).ToList();

favorites.Reverse();

ViewBag.CountPage = (int)Math.Ceiling((double)favorites.Count / CAFP);

ViewBag.FirstPageDiasbled = "";

ViewBag.LastPageDiasbled = "";

// Если статей меньше CAFP в Избранном, то выводим все

if (page == null)

{

// Если больше CAFP, но пользователь не указывал какую страницу открыть, то

// открываем превую

if (ViewBag.CountPage > 1)

{

favorites = favorites.GetRange(0, CAFP);

ViewBag.CurrentPage = 1; //В этом случае текущей страницей будет первая

ViewBag.FirstPageDiasbled = "disabled"; //Выключаем возврат на предыдующую страницу

}

}

else

{

// Если статей больше CAFP в Избранном, то выводим CAFP статей для указанной страницы

//Если на последней странице нет CAFP статей

if (CAFP \* (int)page >= favorites.Count)

{

favorites = favorites.GetRange(CAFP \* ((int)page - 1), favorites.Count() - CAFP \* ((int)page - 1));

ViewBag.LastPageDiasbled = "disabled"; //Выключаем переход на следующую страницу

}

else

{

//Если на странице есть CAFP статей, то берем их

favorites = favorites.GetRange(CAFP \* ((int)page - 1), CAFP);

}

ViewBag.CurrentPage = page; //Получаем тукущую страницу

if (page == 1) ViewBag.FirstPageDiasbled = "disabled";

}

return View(favorites);

}

catch (Exception ex)

{

logger.Debug(DateTime.Now.ToString("dd.MM.yyyy HH:mm:ss") + " in /Blogs/Favorites : " + ex.Message);

return RedirectToAction("Error", "Shared");

}

}

## Примеры работы программы

На рисунке 1 представлена страница Авторизации.

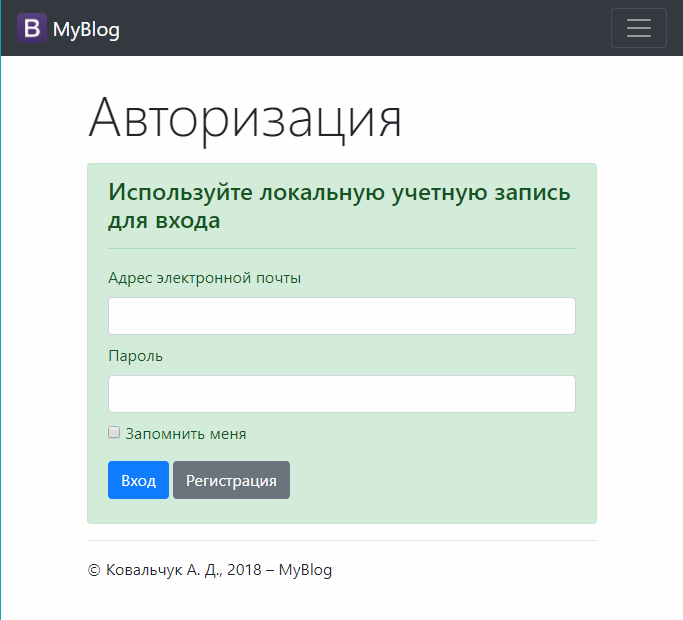


Рисунок 1 – Страница Авторизации

На рисунке 2 представлена страница Регистрации пользователя.

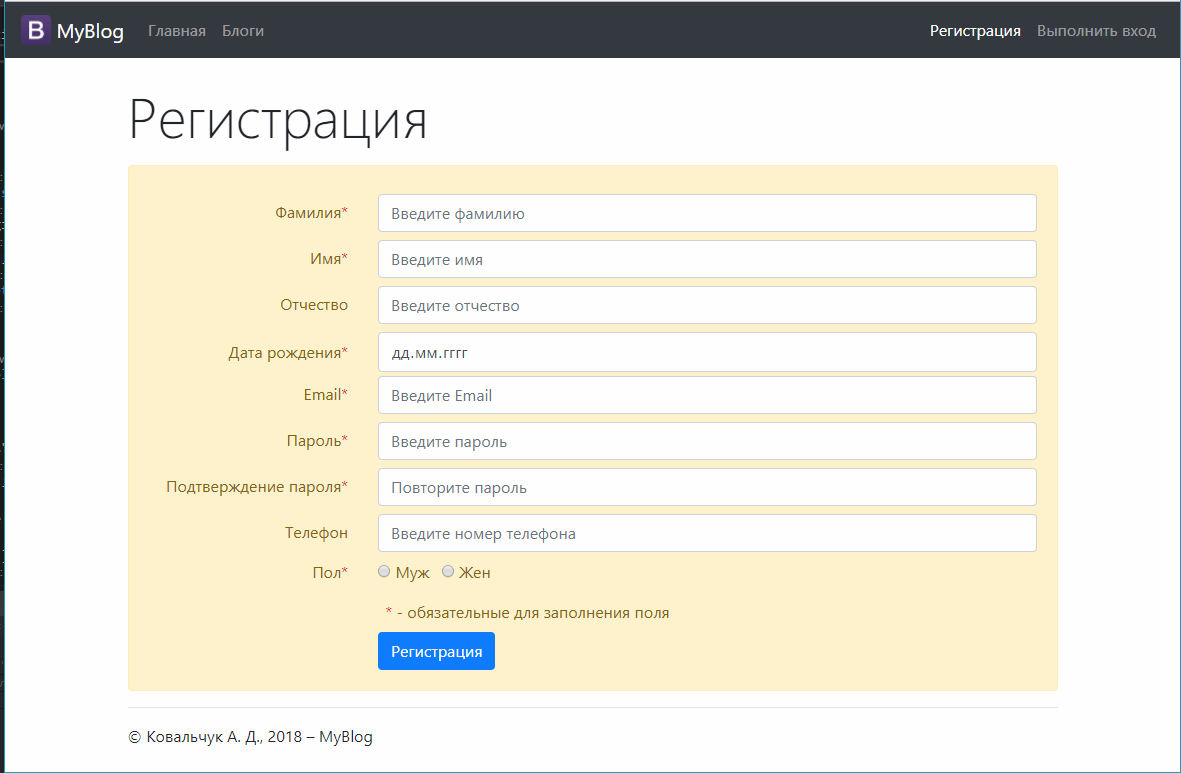


Рисунок 2 – Страница Регистрации пользователя

На рисунке 3 представлена страница Личной информации

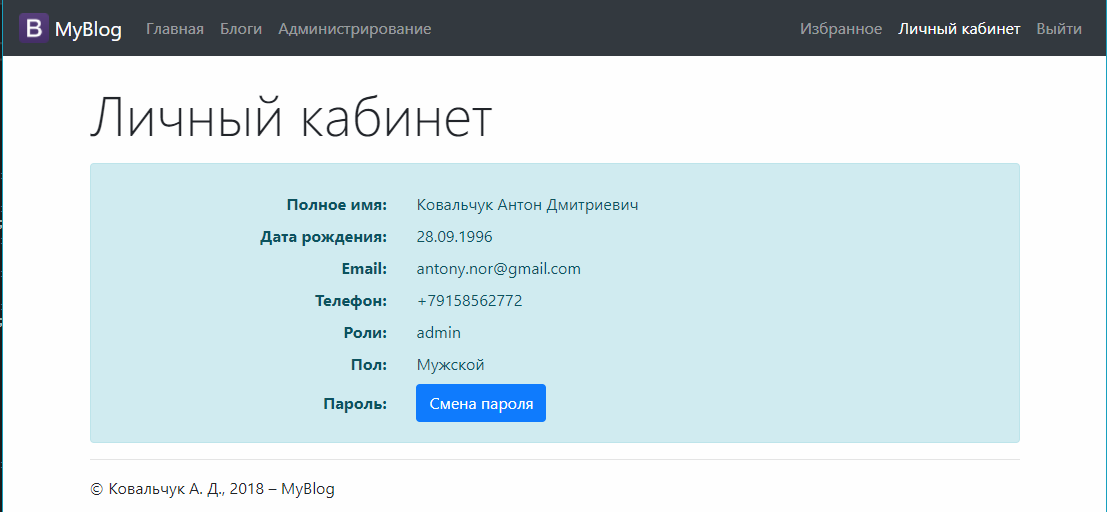


Рисунок 3 – Страница Личной информации

На рисунке 4 представлена страница «Избранное»

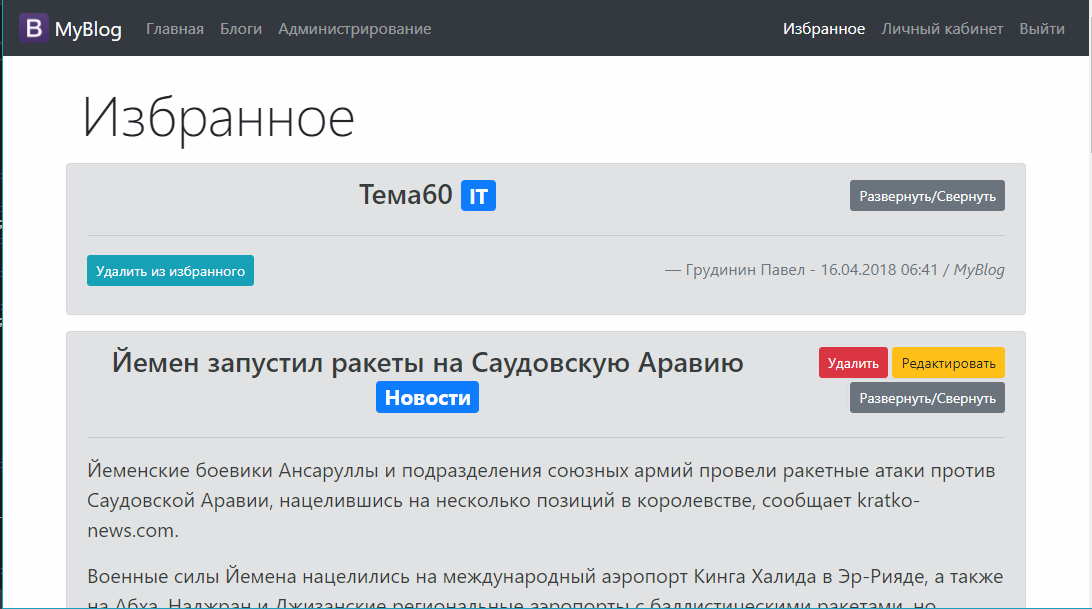


Рисунок 4 – Страница «Избранное»

Пример пользовательского веб-элемента, который работает без перезагрузки страницы представлен на рисунке 5.

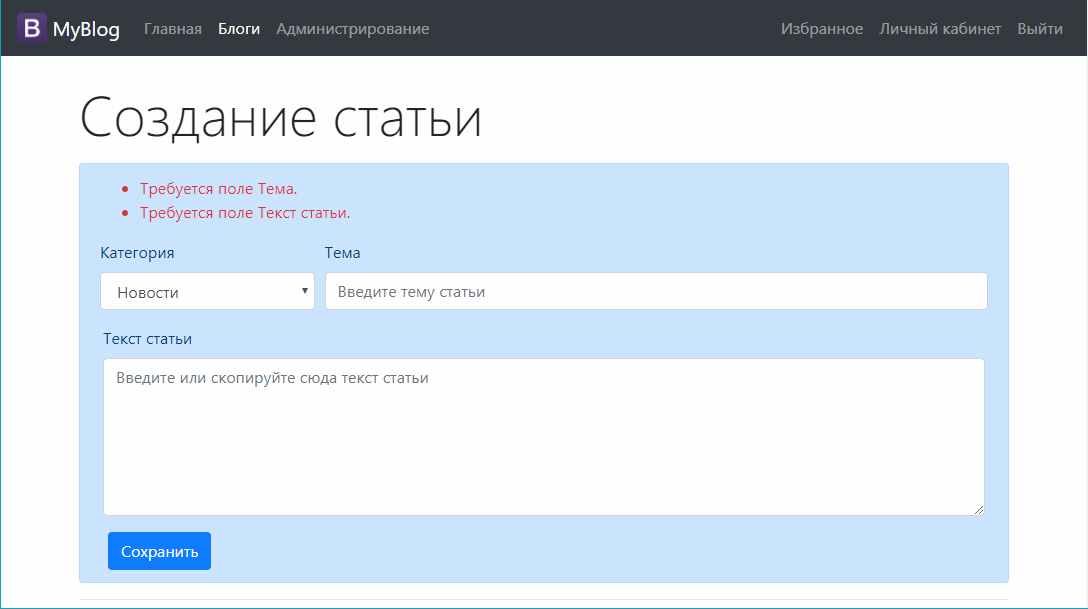


Рисунок 5 – Страница Создания статьи

Пример файла с пользовательскими логами представлен на рисунке 6.

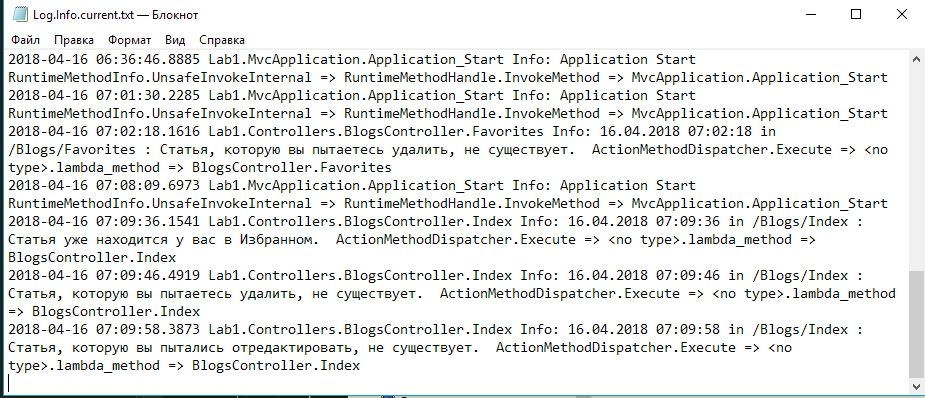


Рисунок 6 – Журнал пользовательских ошибок