МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №33

ОТЧЕТ ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ассистент |  |  |  | А.А. Бардовский |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

**ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3**

РАЗРАБОТКА WEB-КЛИЕТА МОНИТОРИНГА ИСКЛЮЧЕНИЙ В РАСПРЕДЕЛЕННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЕ

по курсу: ТЕХНОЛОГИИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | | 3136 |  |  |  | В.А Панчук |
|  | номер группы | |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

1. **Цель работы**

Изучить принципы построения MVC-решений, позволяющих распределить бизнес-логику в распределенных информационных системах, научиться использовать методы проектирования приложений доступа к данным, базируясь на принципах Model-First. Освоить механизмы Entity Framework для проектирования Web-клиентов РИС.

1. **Постановка задачи**

1) Изучить материалы лекций, размещенные в личном кабинете.

2) В соответствии с заявленной в лекциях функциональностью, разработать Web-клиент, использующий ASP.NET Core MVC подходы разработки распределенных систем.

3) Web-клиент должен обеспечивать возможность удаленного мониторинга таблицы UserExceptions разработанной ранее архитектуры БД, а также поддержку вызова CRUD-операций (create, read, update, delete) над данной таблицей.

4) Локализовать интерфейс Web-клиента (на русском языке должны быть все его элементы)

5) Разработать дополнительный функционал проекта в соответствии с индивидуальным вариантом.

1. **Результат выполнения лабораторной работы**

**Основная часть задания**

Сценарий работы пользователя и результат выполнения программы

После запуска программы пользователь увидит перед собой главную страницу с надписью: «Добро пожаловать в AirLogger». Пример изображён на рисунке 1

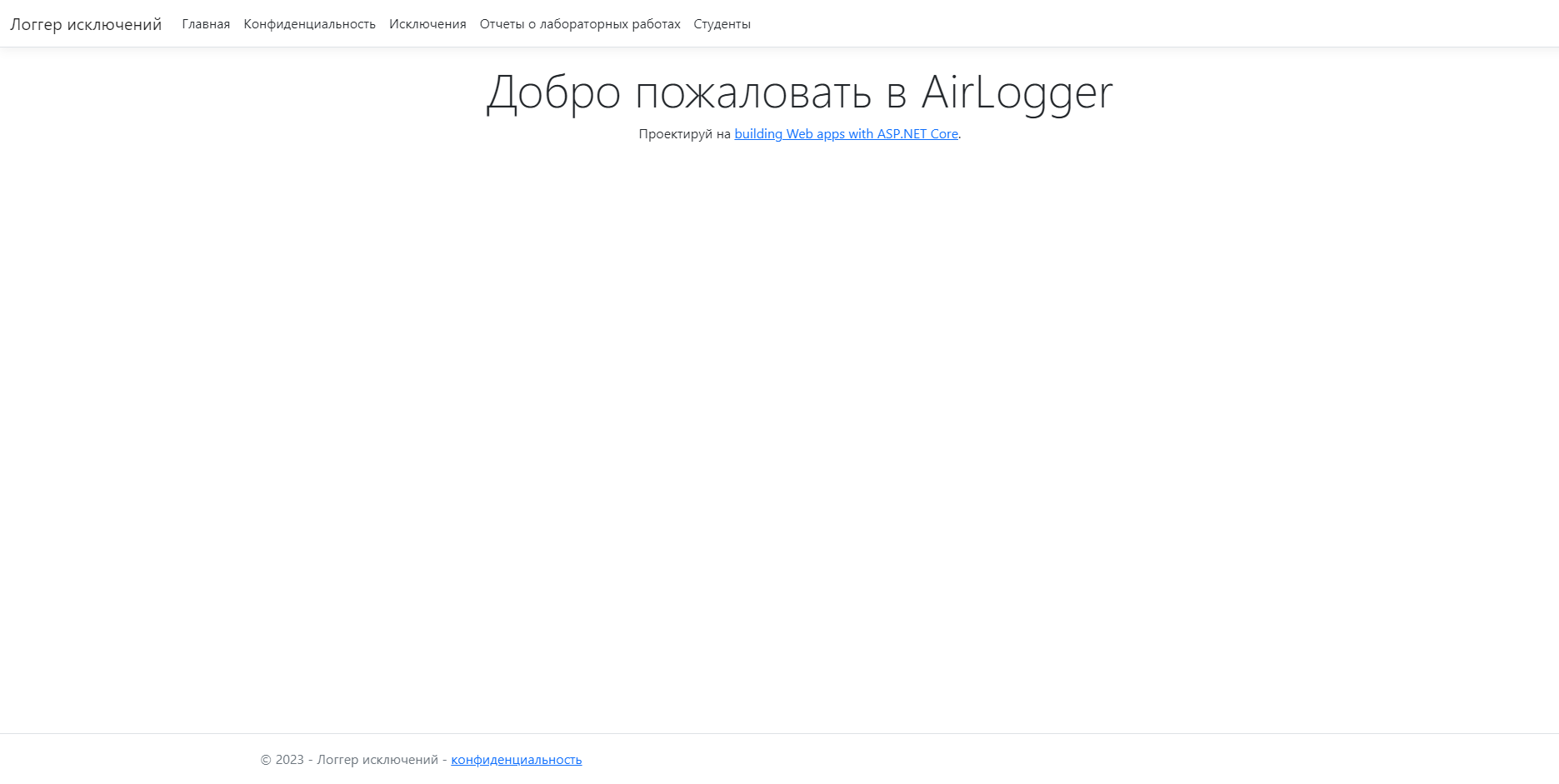


Рисунок 1 – Главная страница

Далее у пользователя есть возможность перейти к различному функционалу разработанной информационной системы. При нажатии на надпись «Исключения» пользователь переходит к исключениям, которые хранятся в базе данных. Пример странички исключений изображён на рисунке 2.

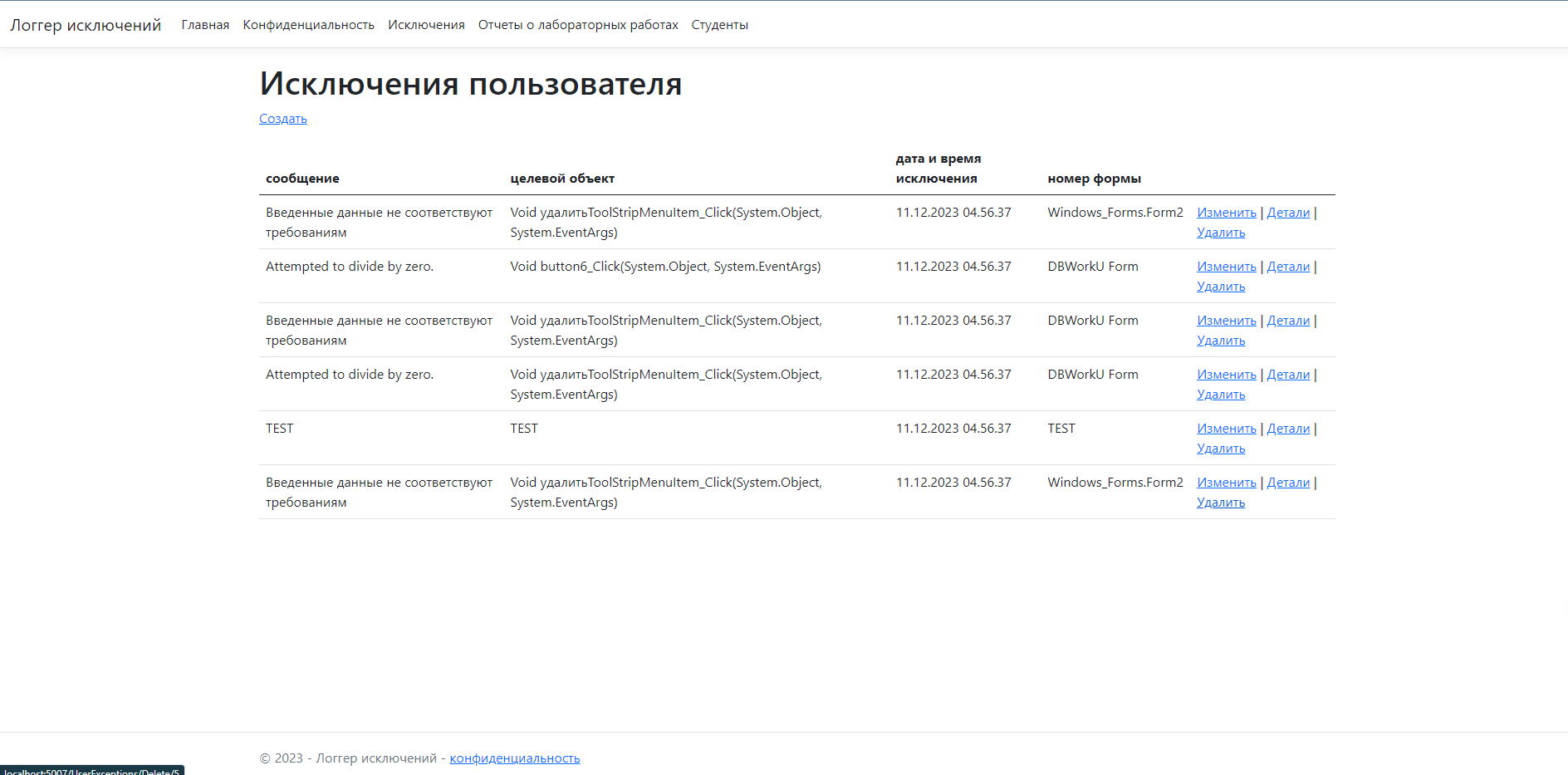


Рисунок 2 – Страница с исключениями

Для того, чтобы пользователю добавить новое исключение в базу данных, необходимо нажать на ссылку «Создать». После чего ввести данные в соответствующие поля (сообщение, целевой объект, индекс формы, дата и время исключения). Далее нужно нажать на кнопку «Создать». Пример добавления исключения изображён на рисунке 3

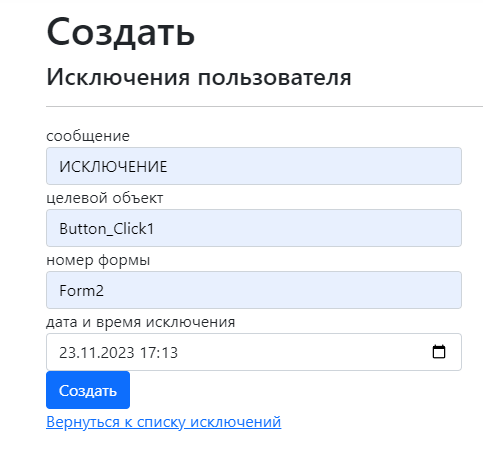


Рисунок 3 – Создание записи

При корректном вводе данных пользователь увидит результат добавления исключение в БД. Пример на рисунке 4. Если пользователь передумал добавлять исключение, то есть возможность вернуться к БД с исключениями, нажав на ссылку «Вернуться к списку исключений».

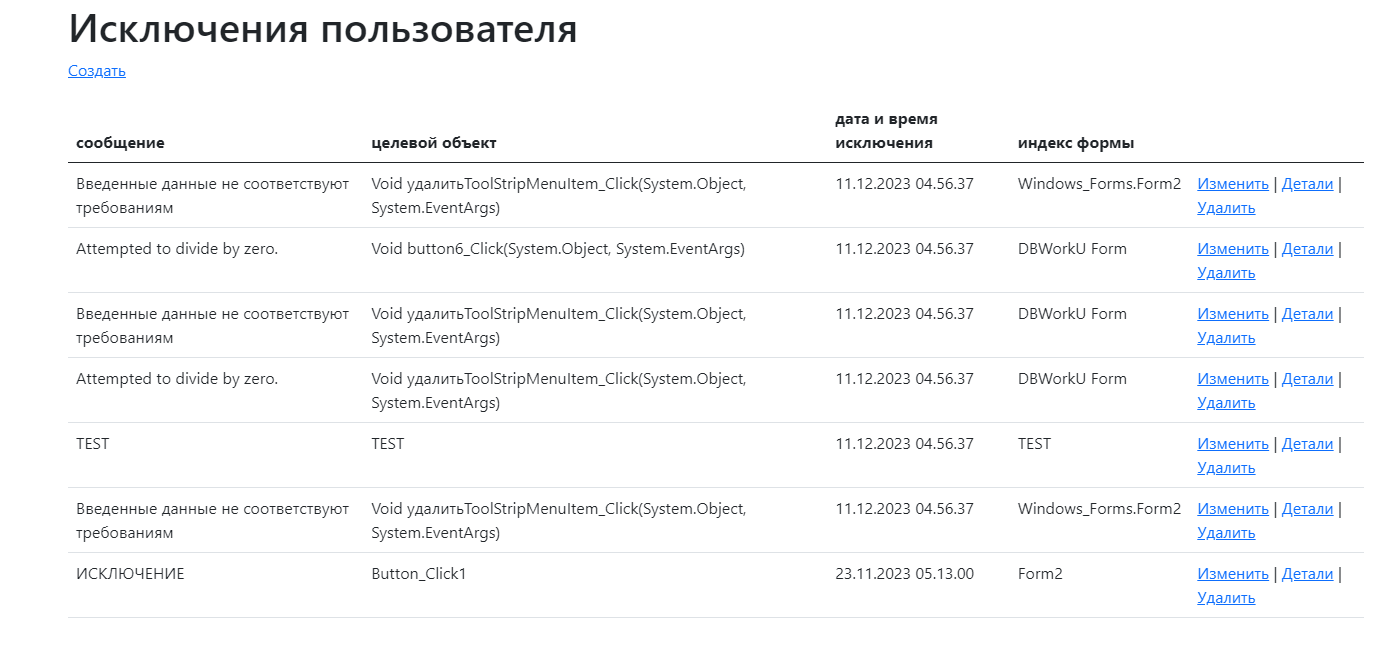


Рисунок 4 – Результат создания записи

Для того, чтобы пользователю изменить определенную запись в базе данных, необходимо нажать на кнопку «Изменить» у записи, в которой должны произойти изменения. После чего редактировать данные в соответствующих полях (сообщение, целевой объект, индекс формы, дата и время исключения). Далее нужно нажать на кнопку «Сохранить». Пример редактирования исключения продемонстрирован на рисунке 5.

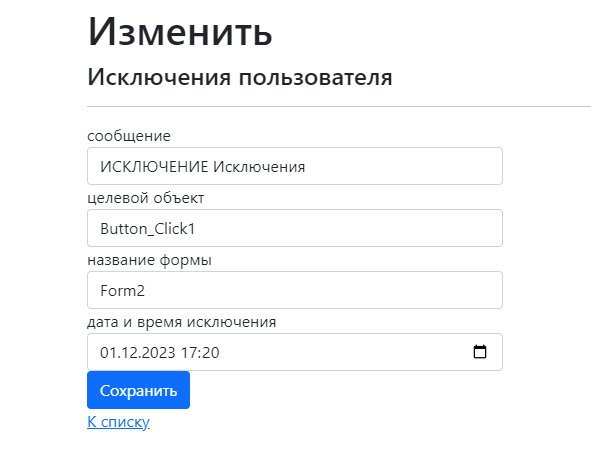


Рисунок 5 – Редактирование записи

При корректном вводе новых данных пользователь увидит результат обновления записи. Если пользователь передумал изменять исключение, то есть возможность вернуться к БД с исключениями, нажав на ссылку «К списку».

Для того, чтобы пользователю посмотреть информацию об определенной записи в базе данных, необходимо нажать на ссылку «Детали» у записи, с которой нужно ознакомиться. Пример странички «Детали» продемонстрирован на рисунке 6.

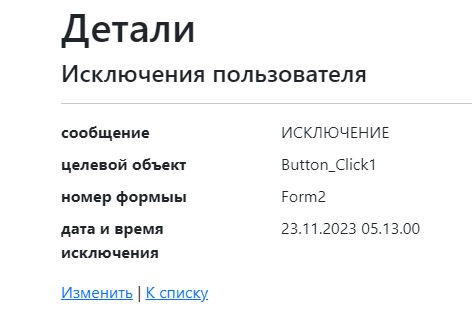


Рисунок 6 – Детали записи

После того, как пользователь увидел детали записи, есть возможность изменить данную запись, нажав на ссылку «Изменить» или вернуться к БД с исключениями, нажав на ссылку «К списку».

Для того, чтобы пользователю удалить определенную запись в базе данных, необходимо нажать на ссылку «Удалить» у записи, которую требуется удалить из БД. Пример изображён на рисунке 7

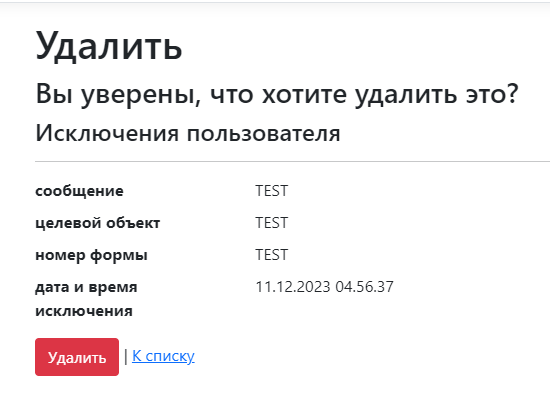


Рисунок 7 – Удаление записи

Далее нажать на кнопку «Удалить». После чего пользователь увидит результат удаления записи. Пример главной странички после удаления записи изображён на рисунке 8.

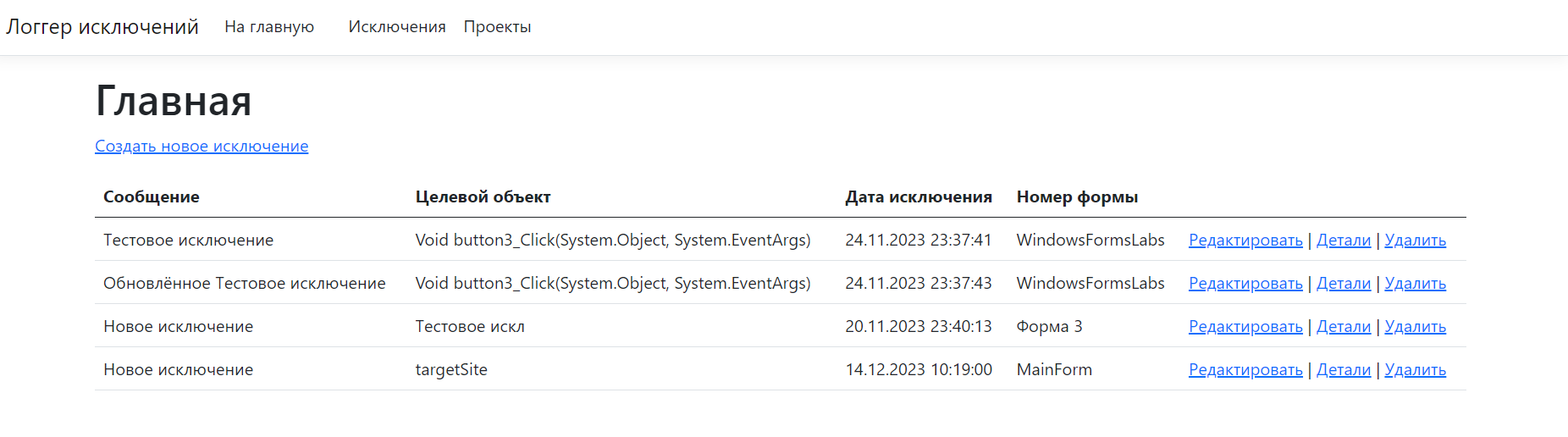


Рисунок 8 – Результат удаления записи

Если пользователь передумал удалять исключение, то есть возможность вернуться к БД с исключениями, нажав на ссылку «К списку».

**Листинг программы**

Листинг кода файла UserExceptionController:

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Threading.Tasks;

**using** Microsoft.AspNetCore.Mvc;

**using** Microsoft.AspNetCore.Mvc.Rendering;

**using** Microsoft.EntityFrameworkCore;

**using** MyDB.Data;

**using** WebApplication1.Models;

**namespace** WebApplication1.Controllers

{

**public** **class** UserExceptionsController : Controller

{

**private** **readonly** MyDBContext \_context;

**public** UserExceptionsController(MyDBContext context)

{

\_context = context;

}

*// GET: UserExceptions*

**public** **async** Task<IActionResult> Index()

{

**return** View(**await** \_context.UserException.ToListAsync());

}

*// GET: UserExceptions/Details/5*

**public** **async** Task<IActionResult> Details(**int**? id)

{

**if** (id == **null**)

{

**return** NotFound();

}

**var** userException = **await** \_context.UserException

.FirstOrDefaultAsync(m => m.Id == id);

**if** (userException == **null**)

{

**return** NotFound();

}

**return** View(userException);

}

*// GET: UserExceptions/Create*

**public** IActionResult Create()

{

**return** View();

}

*// POST: UserExceptions/Create*

*// To protect from overposting attacks, enable the specific properties you want to bind to.*

*// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.*

[HttpPost]

[ValidateAntiForgeryToken]

**public** **async** Task<IActionResult> Create([Bind("Id,Message,TargetSite,DateTimeExc,IndexForm")] UserException userException)

{

**if** (ModelState.IsValid)

{

\_context.**Add**(userException);

**await** \_context.SaveChangesAsync();

**return** RedirectToAction(nameof(Index));

}

**return** View(userException);

}

*// GET: UserExceptions/Edit/5*

**public** **async** Task<IActionResult> Edit(**int**? id)

{

**if** (id == **null**)

{

**return** NotFound();

}

**var** userException = **await** \_context.UserException.FindAsync(id);

**if** (userException == **null**)

{

**return** NotFound();

}

**return** View(userException);

}

*// POST: UserExceptions/Edit/5*

*// To protect from overposting attacks, enable the specific properties you want to bind to.*

*// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.*

[HttpPost]

[ValidateAntiForgeryToken]

**public** **async** Task<IActionResult> Edit(**int** id, [Bind("Id,Message,TargetSite,DateTimeExc,IndexForm")] UserException userException)

{

**if** (id != userException.Id)

{

**return** NotFound();

}

**if** (ModelState.IsValid)

{

**try**

{

\_context.Update(userException);

**await** \_context.SaveChangesAsync();

}

**catch** (DbUpdateConcurrencyException)

{

**if** (!UserExceptionExists(userException.Id))

{

**return** NotFound();

}

**else**

{

**throw**;

}

}

**return** RedirectToAction(nameof(Index));

}

**return** View(userException);

}

*// GET: UserExceptions/Delete/5*

**public** **async** Task<IActionResult> Delete(**int**? id)

{

**if** (id == **null**)

{

**return** NotFound();

}

**var** userException = **await** \_context.UserException

.FirstOrDefaultAsync(m => m.Id == id);

**if** (userException == **null**)

{

**return** NotFound();

}

**return** View(userException);

}

*// POST: UserExceptions/Delete/5*

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

**public** **async** Task<IActionResult> DeleteConfirmed(**int** id)

{

**var** userException = **await** \_context.UserException.FindAsync(id);

**if** (userException != **null**)

{

\_context.UserException.**Remove**(userException);

}

**await** \_context.SaveChangesAsync();

**return** RedirectToAction(nameof(Index));

}

**private** **bool** UserExceptionExists(**int** id)

{

**return** \_context.UserException.Any(e => e.Id == id);

}

}

}

**Индивидуальная часть задания**

**Постановка задачи**

Добавить MVC-контроллер, позволяющий на новой странице разработанного Web-клиента выполнять (CRUD – create, read, update, delete) операции с сущностью «ReportLabWork» разработанной модели данных.

**Основная часть задания**

Сценарий работы пользователя и результат выполнения программы

Для того, чтобы пользователю добавить новый отчет в базу данных, необходимо нажать на ссылку «Создать». После чего ввести данные в соответствующие поля (название отчета, дата отчета, версия отчета, номер проекта, номер студента). Далее нужно нажать на кнопку «Создать». Пример добавления проекта изображён на рисунке 9.

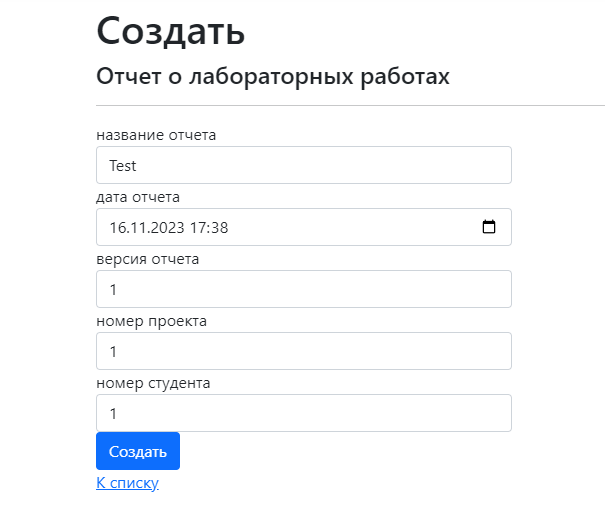


Рисунок 9 – Создание записи

При корректном вводе данных пользователь увидит результат добавления отчета в БД. Пример на рисунке 10. Если пользователь передумал добавлять проект, то есть возможность вернуться к БД с отчетами, нажав на ссылку «К списку».

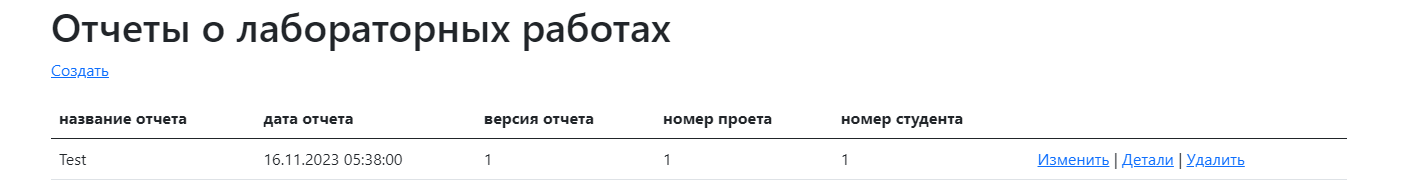


Рисунок 10 – Результат создания записи

**Листинг программы**

Листинг кода файла ReportLabWorksController:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using MyDB.Data;

using WebAspT.Models;

namespace WebAspT.Controllers

{

public class ReportLabWorksController : Controller

{

private readonly MyDBContext \_context;

public ReportLabWorksController(MyDBContext context)

{

\_context = context;

}

// GET: ReportLabWorks

public async Task<IActionResult> Index()

{

return \_context.ReportLabWork != null ?

View(await \_context.ReportLabWork.ToListAsync()) :

Problem("Entity set 'MyDBContext.ReportLabWork' is null.");

}

// GET: ReportLabWorks/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.ReportLabWork == null)

{

return NotFound();

}

var ReportLabWork = await \_context.ReportLabWork

.FirstOrDefaultAsync(m => m.id\_ReportLW == id);

if (ReportLabWork == null)

{

return NotFound();

}

return View(ReportLabWork);

}

// GET: ReportLabWorks/Create

public IActionResult Create()

{

return View();

}

// POST: ReportLabWorks/Create

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("id\_ReportLW,titleReportLW,dateReportLW,versionReportLW,id\_Project,id\_Student")] ReportLabWork ReportLabWork)

{

if (ModelState.IsValid)

{

\_context.Add(ReportLabWork);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(ReportLabWork);

}

// GET: ReportLabWorks/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null || \_context.ReportLabWork == null)

{

return NotFound();

}

var ReportLabWork = await \_context.ReportLabWork.FindAsync(id);

if (ReportLabWork == null)

{

return NotFound();

}

return View(ReportLabWork);

}

// POST: ReportLabWorks/Edit/5

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int? id, [Bind("id\_ReportLW,titleReportLW,dateReportLW,versionReportLW,id\_Project,id\_Student")] ReportLabWork ReportLabWork)

{

if (id != ReportLabWork.id\_ReportLW)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(ReportLabWork);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!ReportLabWorkExists(ReportLabWork.id\_ReportLW))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(ReportLabWork);

}

// GET: ReportLabWorks/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null || \_context.ReportLabWork == null)

{

return NotFound();

}

var ReportLabWork = await \_context.ReportLabWork

.FirstOrDefaultAsync(m => m.id\_ReportLW == id);

if (ReportLabWork == null)

{

return NotFound();

}

return View(ReportLabWork);

}

// POST: ReportLabWorks/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int? id)

{

if (\_context.ReportLabWork == null)

{

return Problem("Entity set 'MyDBContext.ReportLabWork' is null.");

}

var ReportLabWork = await \_context.ReportLabWork.FindAsync(id);

if (ReportLabWork != null)

{

\_context.ReportLabWork.Remove(ReportLabWork);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool ReportLabWorkExists(int? id)

{

return (\_context.ReportLabWork?.Any(e => e.id\_ReportLW == id)).GetValueOrDefault();

}

}

}

1. **Вывод**

В ходе лабораторной работы, были изучены принципы построения MVC-решений, позволяющих распределить бизнес-логику в РИС. Освоен навык использования методов проектирования приложений доступа к данным, базируясь на принципах Model-First. В результате был разработан Web-клиент, обеспечивающий возможность удаленного мониторинга таблиц, а также поддержку вызова CRUD-операций (create, read, update, delete) над данными таблицами.