*Федеральное государственное бюджетное образовательное учреждение высшего образования*

*«Владимирский государственный университет*

*имени Александра Григорьевича и Николая Григорьевича Столетовых»*

*Кафедра информационных систем и программной инженерии*

***ПОЯСНИТЕЛЬНАЯ ЗАПИСКА***

***к курсовому проекту по дисциплине   
"Технологии программирования"***

***на тему***

*Проектирование и разработка программной системы   
автоматизации документооборота малого предприятия*

*Выполнил: ст. гр. ПРИ-117*

*Журавлев Н.М.*

*Принял: доц. кафедры ИСПИ*

*Вершинин В.В.*

*Владимир, 2019*

**Аннотация**

В данном курсовом проекте был создан прототип программной системы автоматизации документооборота небольшого предприятия. Созданная программная система предназначена для учёта и хранения данных о документах в рамках предприятия.

Данная система автоматизирует работу с данными для следующих категорий пользователей: директор, администратор, сотрудник отдела продаж, сотрудник отдела покупок, сотрудник бухгалтерии, аналитик и незарегистрированный пользователь системы.

В ходе выполнения данного курсового проекта были выполнены следующие пункты задания для курсового проекта:

1. Анализ предметной области, изучение структуры и функций программной системы.

2. Моделирование структуры объектов предметной области и их взаимодействия на концептуальном уровне.

3. Проектирование структуры программной системы.

4.Реализация программной системы, составление тестов.

Пояснительная записка содержит 49 листов. Включает в себя: 18 рисунков, 2 приложения. Количество использованных источников – 8.

**ANNOTATION**

In this course project was created a prototype software system for automation and accounting of document circulation inside small company. The created software system is designed to record and store data documents inside company.

This system automates the work with data for the following categories of users: director, administrator, sale worker, buy worker, accountant, analyst and unregistered user of the system.

In the course of this course project, the following points of the assignment were completed for the course project:

1. Analysis of the subject area, the study of the structure and functions of the software system.

2. Modeling the structure of objects of the subject area and their interaction at a conceptual level.

3. Designing the structure of the software system.

4. Realization of a software system, compilation of tests.

The explanatory note contains 49 sheets. Includes: 18 figures, 2 applications. The number of sources used is 8.

# Содержание

Содержание 2

1 Введение 4

2 ПОСТАНОВКА ЗАДАЧИ 5

1.1. Основные требования 5

1.2. Использованные технологии 5

3 проектирование системы 6

2.1. Описание предметной области 6

2.2. Словарь предметной области 6

2.3. Диаграмма прецедентов 7

2.4. Диаграмма классов 11

2.5. Диаграмма состояний 13

2.6. Диаграмма последовательностей 14

4 РАЗРАБОТКА системы 15

3.1. Схема базы данных 15

3.2. Интерфейс системы 15

3.3. Прототип системы 19

3.4. Тестирование системы 19

3.5. Использованные средства 20

5 ЗАКЛЮчение 21

6 список источников 22

ПРИЛОЖЕНИЕ А Схемы, таблицы и диаграммы созданные в процессе анализа предметной области 23

ПРИЛОЖЕНИЕ Б Коды классов разработанной системы 28

# Введение

В ходе выполнения курсового была создана система хранения данных позволяющая пользователям осуществлять взаимодействие с этими данными в зависимости от их ролей и состояния системы. С помощью разработанной системы можно значительно упростить и ускорить процессы учета данных и документооборота в рамках небольшого предприятия.

# ПОСТАНОВКА ЗАДАЧИ

2.1. Основные требования

В ходе выполнения курсового проекта необходимо спроектировать и разработать систему контроля документооборота, которая выполняет функцию корпоративной работы с документами. Система также должна вести учет клиентов и пользователей системы, разграничивая их роли и обязанности.

Разработанная система должна поддерживать многопользовательский режим работы, производить корректное добавление, редактирование, удаление и просмотр данных, а также соответствовать основным стандартам разработки программного обеспечения.

2.2. Использованные технологии

* Методология проектирования и разработки RUP
* Язык моделирования UML;
* Графические редакторы Microsoft Visio, Open Model Sphere;
* Платформа разработки ASP.NET;
* Интегрированная среда разработки Microsoft Visual Studio Community;
* Язык программирования высокого уровня C#;
* Языки разметки HTML5, javascript, razor;
* Паттерн проектирования MVC;
* Система управления реляционными базами данных — SQL Server Express;
* Язык структурированных запросов SQL;
* Веб сервер IIS;
* Система управления версиями Git;
* Удаленный репозиторий GitHub;

# проектирование системы

3.1. Описание предметной области

Система автоматизации документооборота предназначена для хранения данных о документах . Система предполагает выполнение функции автоматизации процессов создания, изменения, просмотра и удаления документов. В системе предусматривается различное поведение в зависимости от роли пользователя.

3.2. Словарь предметной области

**Документ** – набор данных описывающий основную информацию о физическом документе, так же как и физический документ ссылается на уже совершенный или планируемый процесс в рамках предприятия. Является обобщающим понятием для всех типов документа.

**Документ продажи** – один из типов документа, представляет собой набор узконаправленных данных о процессе покупки предприятием какого либо товара у одного из содействующих клиентов.

**Документ продажи** – один из типов документа, представляет собой набор узконаправленных данных о процессе продажи предприятием какого-либо товара содействующему клиенту.

**Документ отчета** – один из типов документа, представляет собой набор узконаправленных данных о статистике собранной в рамках временного периода внутри предприятия.

**Внутренний документ** – один из типов документа, представляет собой набор узконаправленных данных о финансовых манипуляциях среди рабочих предприятия (премия, штраф).

**Рабочий** – физическое лицо, которое выполняет определенный перечень обязанностей в рамках предприятия, является пользователем системы.

**Клиент** – физическое или юридическое лицо, поддерживающее сотрудничество с предприятием (покупка, продажа).

**Роль** – значение, которое присваивается каждому пользователю системы, определяет его обязанности и перечень функций доступных рабочему в рамках системы. (подробнее в пункте 2.3 стр. 7)

**Статус документа** – важнейший параметр документа, который оределяет его и состояния и перечень действий, которые возможно производить над ним в рамках каждого состояния. (подробнее в пункте 2.5 стр.13)

**Тип документа** – значение-разновидность документа, которое определяет его узконаправленное содержимое. (подробнее в пункте 2.4 стр.10)

3.3. Диаграмма прецедентов

Основные функции выделенные во время разработки диаграммы прецедентов :

* Авторизация в системе
* Поиск документа по номеру
* Вывод документов по типу
* Вывод списка всех документов
* Создание документа: Создание документа продажи, Создание внутреннего документа, Создание документа закупки, Создание отчетного документа.
* Редактирование документа: Редактирование документа продажи, Редактирование внутреннего документа, Редактирование документа закупки, Редактирование отчетного документа.
* Вывод всей информации о конкретном документе
* Подписание документа
* Поиск документа ожидающего подписания
* Внос в систему информации о клиенте
* Редактирование информации о клиенте
* Удаление информации о клиенте
* Вывод списка всех клиентов
* Отправка документа на подписание
* Уничтожение документа
* Добавление рабочего
* Редактирование информации о рабочем
* Вывод списка всех рабочих
* Удаление рабочего
* Редактирование информации о рабочем
* Отправка документа на редактирование
* Отправка документа на удаление

Актры разрабатываемой системы.

1. **Директор** – осуществляет контроль и руководство над работой пользователей системы, осуществляет одобрение каждого документа внутри информационной системы.
2. **Администратор** – осуществляет контроль над документами и направляет их на подписание директору, отвечает за удаление старых и необязательных для хранения документов, а также осуществляет управление клиентами и работниками системы.
3. **Аналитик** – осуществляет разнообразные статистические подсчеты для оптимизации работы предприятия, выполняет просмотр, создание и редактирование отчетных документов.
4. **Сотрудник отдела закупок** – является ответственным за все документы закупках, выполняет просмотр, создание и редактирование документов этого типа.
5. **Сотрудник отдела продаж** – является ответственным за все документы о продажах, выполняет просмотр, создание и редактирование документов этого типа.
6. **Сотрудник бухгалтерии** – является ответственным за все внутренние документы, выполняет просмотр, создание и редактирование документов этого типа.
7. **Системный аналитик** – является ответственным за работу системы, осуществляет контроль за основным функционалом системы, имеет доступ ко всем ее функциям, а также может вносить изменения во внутреннее устройство системы.

Расширенное описание основных прецедентов.

**1. “Подписание документа”**

**Предусловие.** В системе имеются документы ожидающие подписания.

**Актеры:** “Директор”

**Основной поток.** Актер после нахождения документа в системе обращается к нему и подтверждает подписание, давая системе команду о смене значения статуса документа с “ожидает подписания” на “подписан” и об удалении всех предыдущих комментариев к данному документу.

**Альтернативный поток.** Актер после просмотра документа отказывается его подписать и нажимает соответствующую кнопку, после нажатия которой актеру предлагается оставить комментарий. В этом случае система изменяет значение статуса документа с “ожидает подписания” на “подлежит редактированию” и добавляет документу комментарий, оставленный актером.

**Постусловие.** Если документ успешно отредактирован, на экран должно быть выведено уведомление об этом.

**2. “Поиск документа по номеру”**

**Предусловие.** В системе имеются документы, каждый из которых имеет свой уникальный номер.

**Актеры:** “Директор”, “Администратор”, “Аналитик”, “Сотрудник отдела закупок”, “ Сотрудник отдела продаж”, “ Сотрудник бухгалтерии”.

**Основной поток.** Актер вводит в отображенную форму номер разыскиваемого документа и запускает операцию поиска. В системе организуется поиск по внесенному пользователем номеру среди всех документов системы. Документ находится системой

**Альтернативный поток.** У актера нет прав на просмотр данного документа. В этом случае система возвращает страницу авторизации пользователя. Если документа с введенным номером в системе не обнаружено, система выводит страницу с сообщением об ошибке.

**Постусловие.** После нахождения в системе документа с данным номером система организует вывод на экран полной информации о документе с данным индивидуальным номером.

**3. “Создание документа”**

**Предусловие.** В системе имеются рабочий и клиент привязанные к какому-либо документу.

**Актеры:** “Аналитик”, “Сотрудник отдела закупок”, “ Сотрудник отдела продаж”, “ Сотрудник бухгалтерии”.

**Основной поток.** Актер нажимает на кнопку, которая подразумевает добавление документа конкретного типа. Система направляет его на страницу заполнения формы. Актер заполняет форму данными и подтверждает создание документа нажатием на соответствующую кнопку.

**Альтернативный поток.** Если данные введенные актером в форму не соответствуют валидации, предусмотренной для каждого поля, под соответствующим полем выведется сообщение об ошибке ввода данных.

**Постусловие.** После успешного создания документа система вернет страницу со списком документов конкретного типа, на которой отразиться информация о только что добавленном документе.

Полная диаграмма прецедентов представлена в приложении А Рисунок 15.

3.4. Диаграмма классов

Сущности системы выявленные в результате анализа диаграммы прецедентов, описанные в словаре предметной области:

* **Рабочий (Worker)** – сущность определяющая всех пользователей системы. Имеет атрибуты электронная почта (email), ФИО (fio), дата рождения (birthdate), зарплата (salary), дата приема на работу (employdate), роль (role) и определяет методы вывода списка всех рабочих(GetAllWorkers) добавление информации о рабочем (CreateWorker), обновлении информации о рабочем (UpdateWorker), удалении информации о рабочем (DropWorker).
* **Клиент (Client)** – сущность определяет физическое лицо, которому относятся документы системы. Имеет атрибуты официального имени (officialname) и почтового адреса (email) и определяет методы вывода списка всех рабочих(GetAllClients), добавления информации о клиенте (CreateClient), редактирования информации о клиенте (UpdateClient) и удаления информации о клиенте (DropClient).
* **Документ (Document)** – сущность определяющая любой документ системы. Имеет атрибуты номера (docnumber), названия (name), даты создания (creationdate), автора (workerID), статуса (status), комментария (comment), времени хранения (shelflife), типа (type), подписывающего (signerID). Определяет методы вывода списка всех документов (GetAll), поиска документа по номеру (FindDocByNumber), подписи документа (SignDoc), поиска документов ожидающих подписания (FindDocWaiting), отправки документов на подписание (SendDocForSign), удаления документов (DropDoc), отправки документа на обновление (SendDocForUpdate) и отправки документа на удаление (SendDocForDelete).
* **Документ покупки (DocumentBuy)** – сущность определяющая накладную на партию товара для закупки. Имеет атрибуты названия продукции (producname), количества купленной продукции в кг (productweight), цены за 1 кг (productprice) и продавщего клиента (ClientID). Определяет методы вывода всех документов покупки (GetAllBuys), добавления информации о документах закупок (CreateDocBuy) и редактирования (UpdateDocBuy) этих документов.
* **Документ продажи (DocumentSale)** – сущность определяющая накладную на партию товара для продажи. Имеет атрибуты названия продукции (producname), количества проданной продукции в штуках (productamount), цены за 1 штуку (productprice) и покупающего клиента (ClientID). Определяет методы вывода всех документов продажи (GetAllSales), добавления информации о документах продаж (CreateDocSale) и редактирования (UpdateDocSale) этих документов.
* **Документ отчета (DocumentReport)** – сущность определяющая документ со статистическими данными за период времени. Имеет атрибуты периода времени (start) и (end), за который определена статистика (stats) самого содержимого отчета. Определяет методы вывода всех документов отчета (GetAllReports), добавления информации об отчетных документах (CreateDocRepor) и редактирования (UpdateDocReport) этих документов.
* **Внутренний финансовый документ (DocumentInside)** – сущность определяющая внутренний финансовый документ. Имеет атрибуты рабочего (workerID) и денежной разности (размер премии или штрафа) (moneydifference). Определяет методы вывода всех документов отчета (GetAllInsides), добавления информации о внутренних финансовых документов (CreateDocInside) и редактирования (UpdateDocInside) этих документов.

Описание связей между классами:

* Связь между классом Document и классами DocumentBuy, DocumentSale, DocumentInside, DocumentReport. Способ связи – обобщение. Document является родительским классом, а остальные классы его потомками. Они являются конкретными воплощениями общей сущности Document.
* Связь между классами Document и Worker. Способ связи – ассоциация. У каждого элемента класса Document имеется 1 елемент-создатель класса Worker. Кратность связи Document Worker – 1 ко многим.
* Связь между классом Client и классами DocumentBuy и DocumentSale. Способ связи – ассоциация. Связь олицетворяет принадлежность документа донного типа одному из клиентов. У одного клиента может быть много документов, но каждый документ принадлежит строго одному клиенту. Кратность связи Client DocumentBuy и Client DocumentSale – 1 ко многим.
* Связь между классами DocumentInside и Worker. Способ связи – ассоциация. У каждого элемента класса DocumentInside имеется 1 елемент цели (к какому рабочему применим данный документ) класса Worker. Кратность связи Document Worker – 1 ко многим.

Полная диаграмма классов представлена в приложении А, Рисунок 13.

3.5. Диаграмма состояний

На диаграмме состояний представлен жизненный цикл главного объекта информационной системы – документа.

Все возможные состояноия документа:

* **Создан и находится в системе**. Это начальное состояние документа после создания, также в это состояние документ переходит после редактирования.
* **Подлежит редактированию.** В этом состоянии в документе имеются какие-либо неверные данные. В это состояние документ может перевести только Директор и Администратор, добавив к документу комментарий-пояснение.
* **Отправлен наподпись.** В этом состоянии документ проверен на корректность данных Администратором и ожидает утверждения Директора.
* **Подписан.** В это состояние документ приходит после одобрения директора. Документ полностью проверен и может исполняться вне системы.
* **Отправлен на удаление.** Это состояние сигнализирует о том, что в существование документа не имеет смысла в связи с истечением срока его хранения или отсутствия реального процесса описанного в документе. Только перейдя в это состояние документ может быть полностью удален из системы.

Полная диаграмма состояний представлена в приложении А, Рисунок 14.

3.6. Диаграмма последовательностей

На диаграмме последовательностей отражен процесс подписания документа, с описанием выполняющихся действий и функций от актера системы (Директор) до класса системы (Документ) через интерфейс системы.

Полная диаграмма последовательностей представленна в приложении А, Рисунок 16.

# РАЗРАБОТКА системы

4.1. Схема базы данных

Физическая модель разработанной для системы базы данных, которая хранит основную информацию, основана на диаграмме классов.

Схема базы данных представлена в приложении А, Рисунок 12.

4.2. Интерфейс системы

В ходе разработки системы было созданно более 50 web-страниц, составленных в единую структуру. Они поддерживают ссылочную и кнопочную целостность, дают возможность пользователю управлять системой, предусматривают некоторые ошибочные и некорректные действия пользователя.

Основные элементы структуры страниц представленны ниже, Рисунок(1-11):

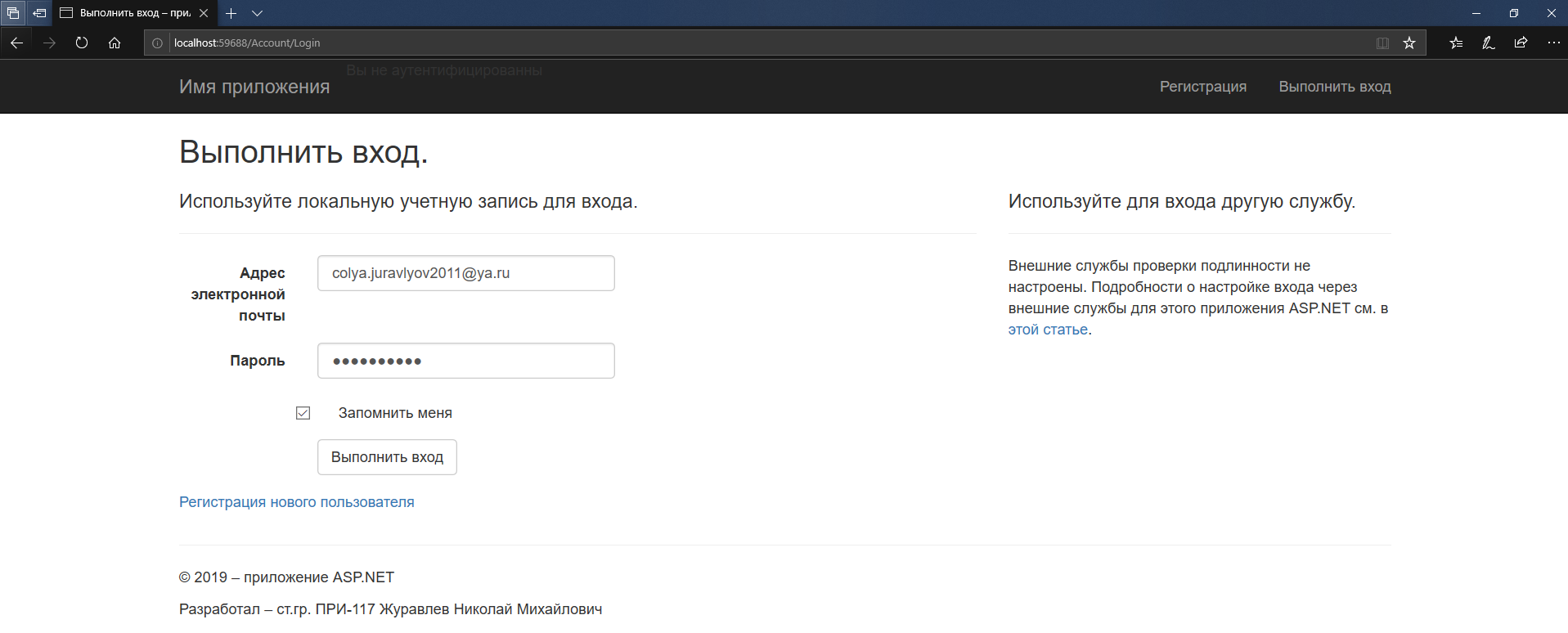


Рисунок 1. Страница входа в систему.

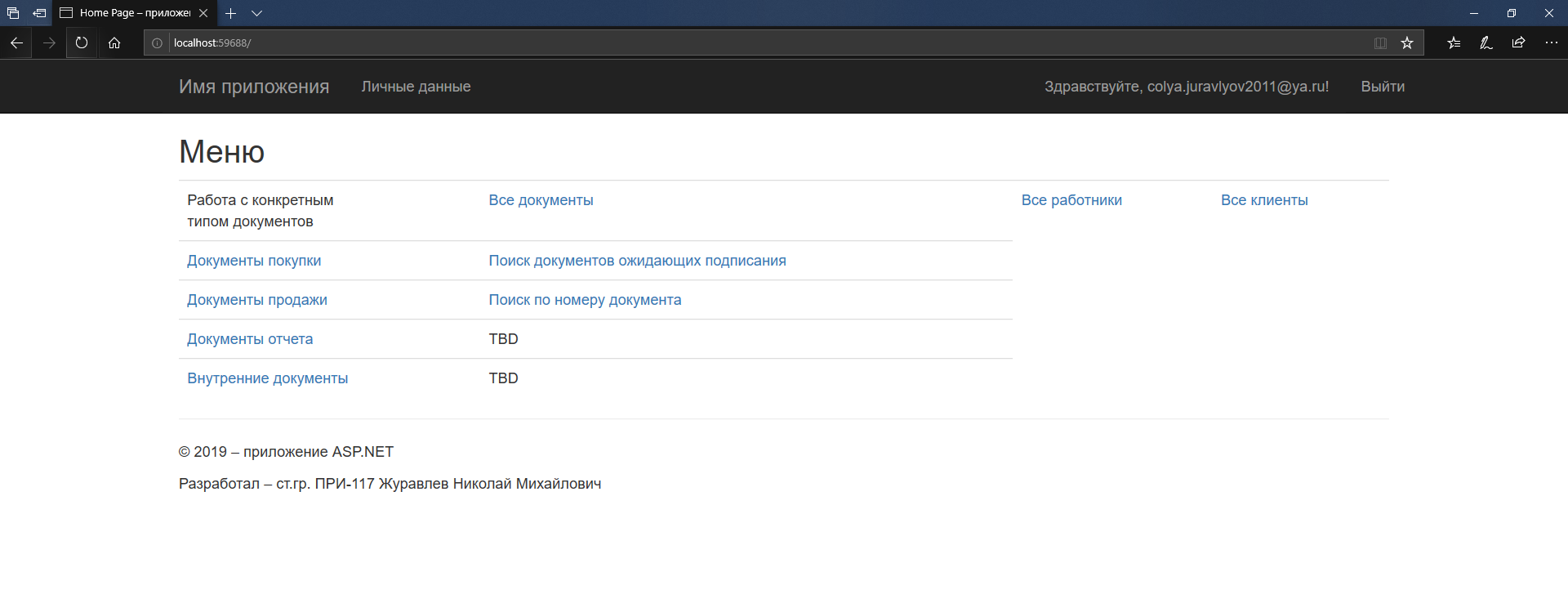


Рисунок 2. Страница меню.

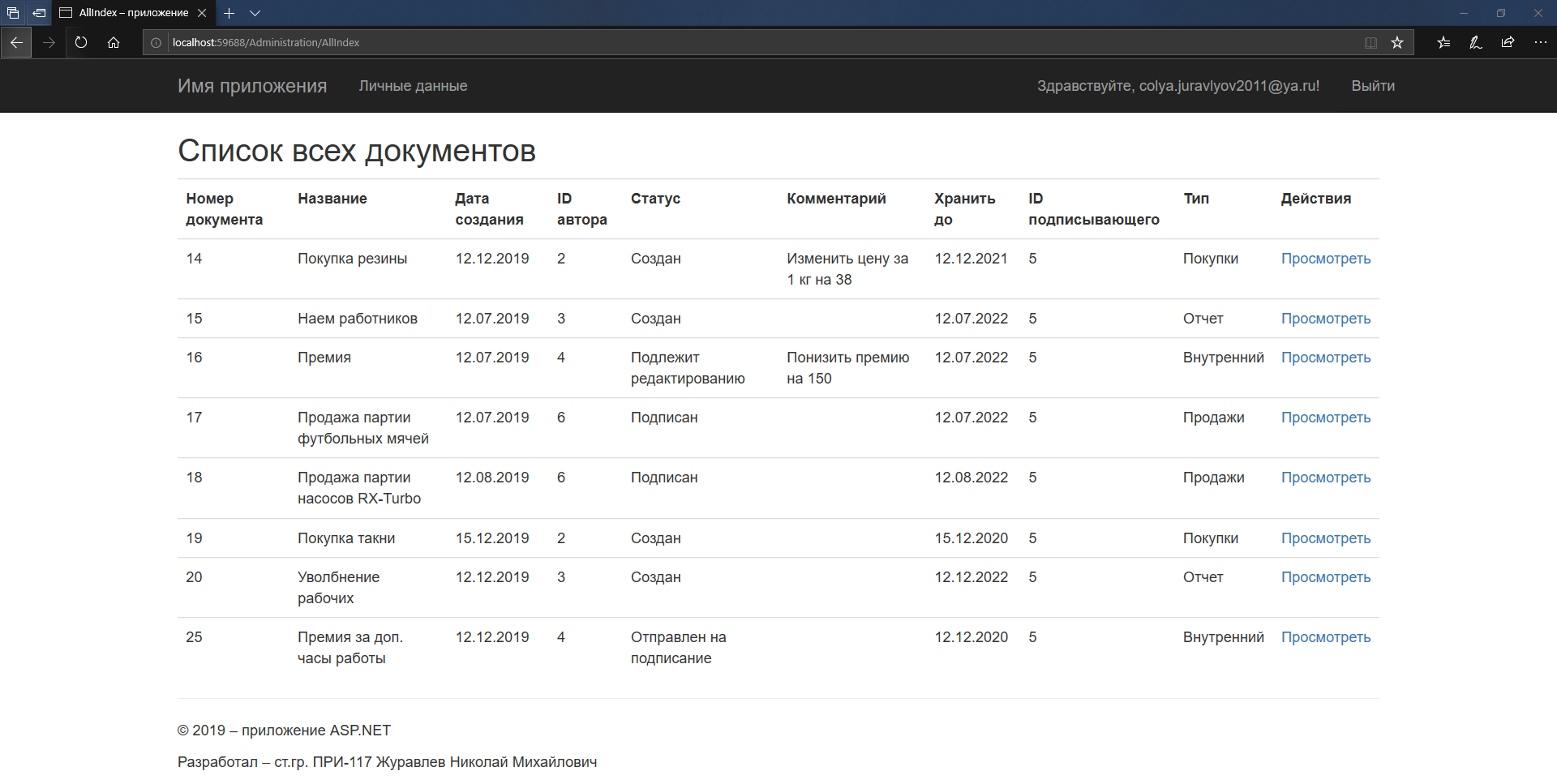


Рисунок 3. Страница со списком всех документов.

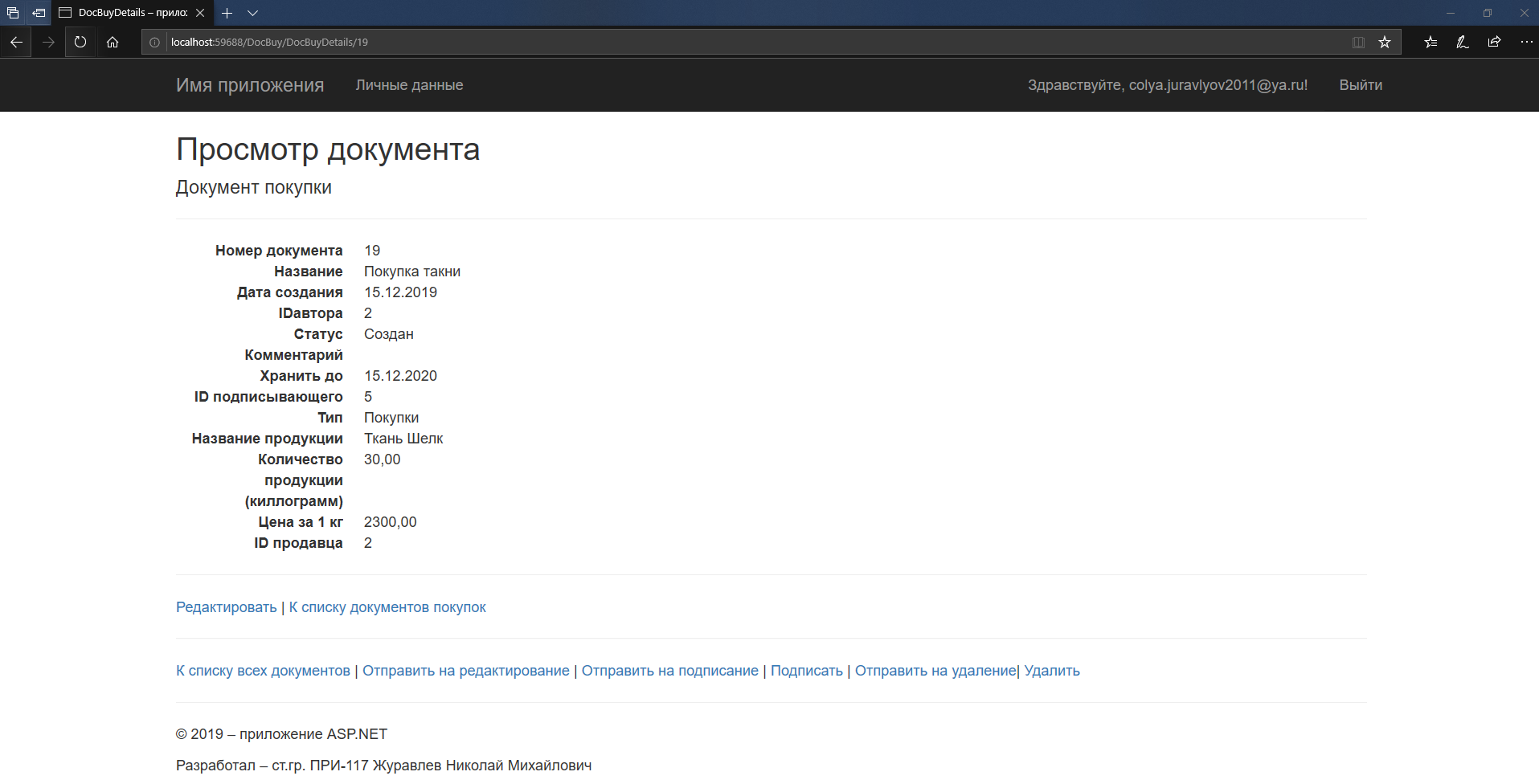


Рисунок 4. Страница просмотра информации о документе.

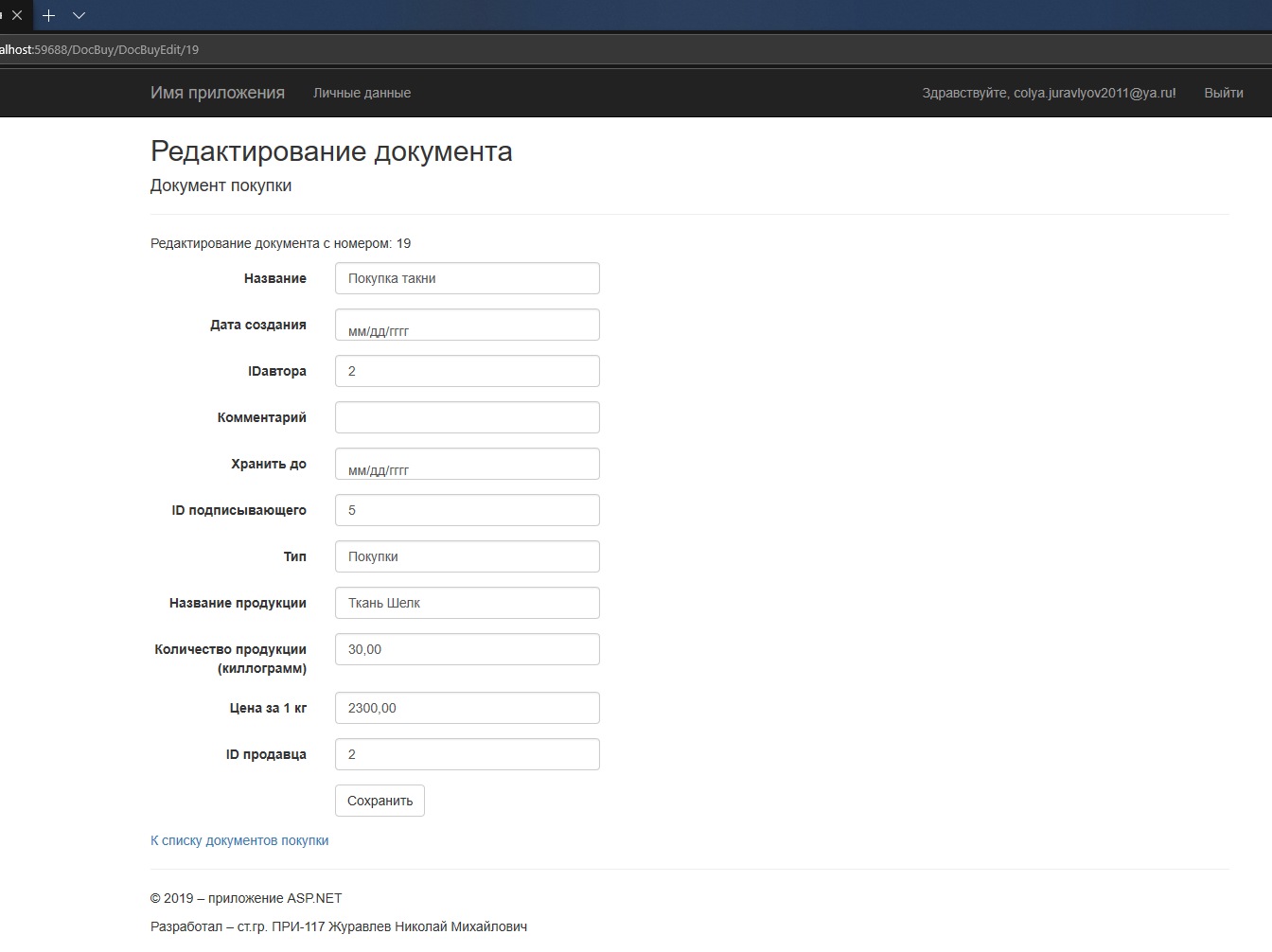


Рисунок 5. Страница редактирования документа покупки.

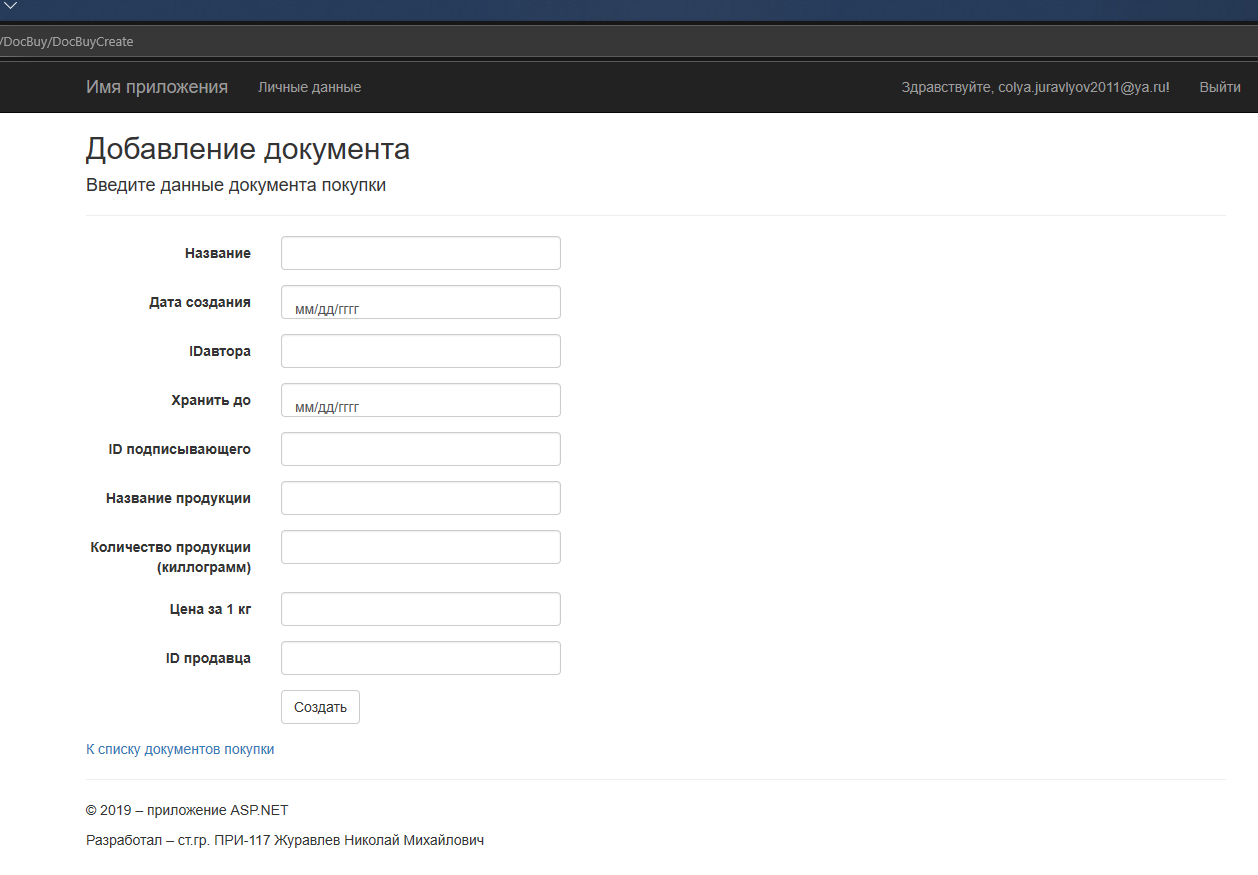


Рисунок 6. Страница добавления документа покупки.

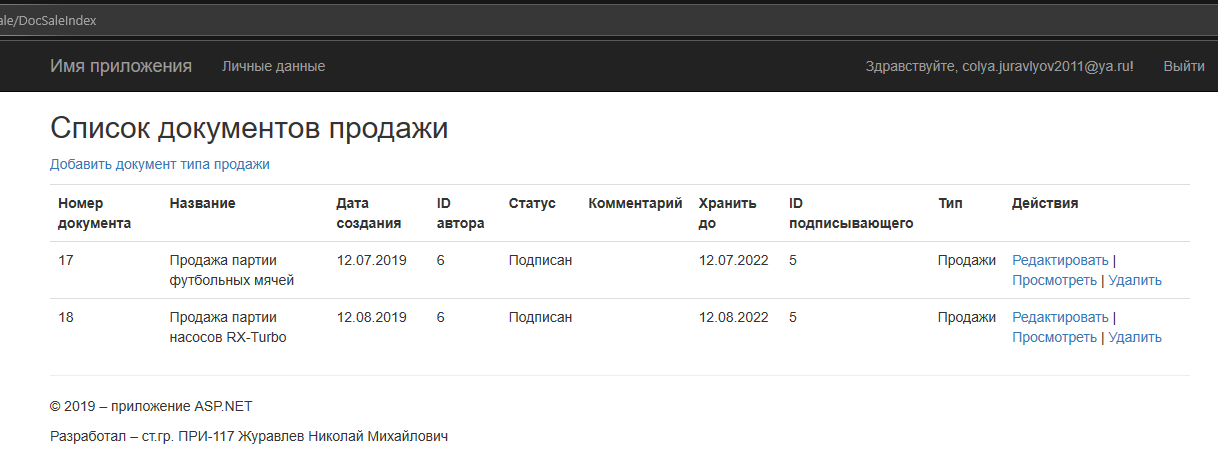


Рисунок 7. Страница списка всех документов продажи.

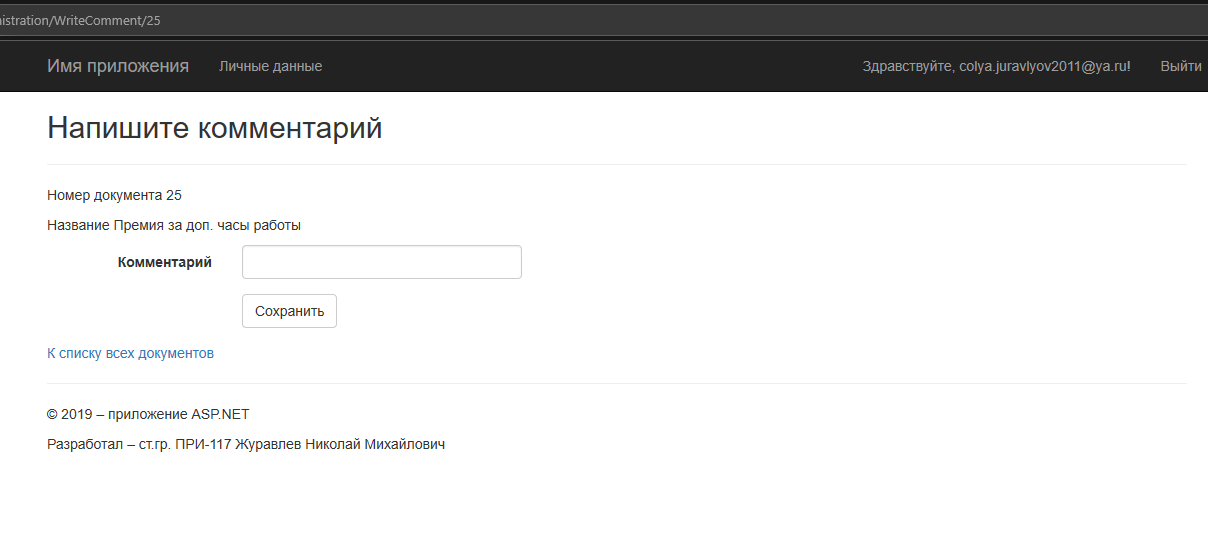


Рисунок 8. Страница отправки документа на редактирование с вводом комментария.

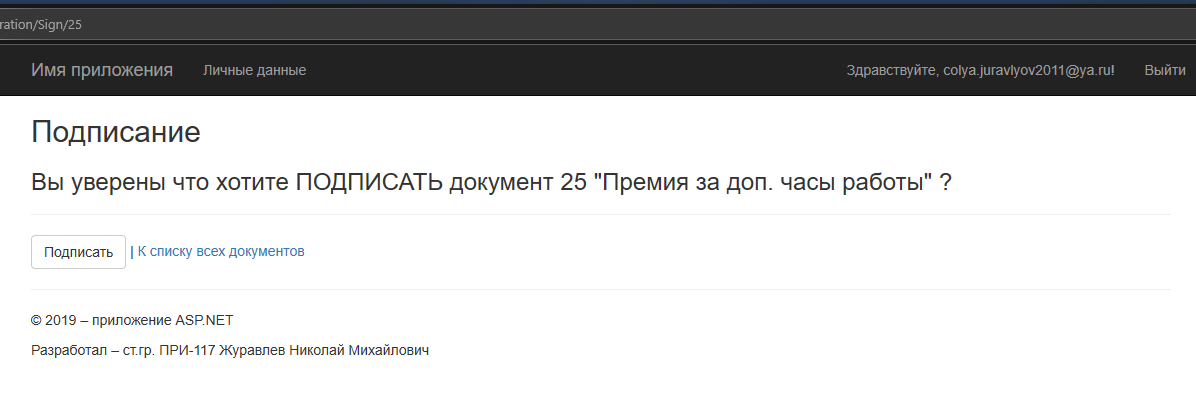


Рисунок 9. Страница подписания документа.

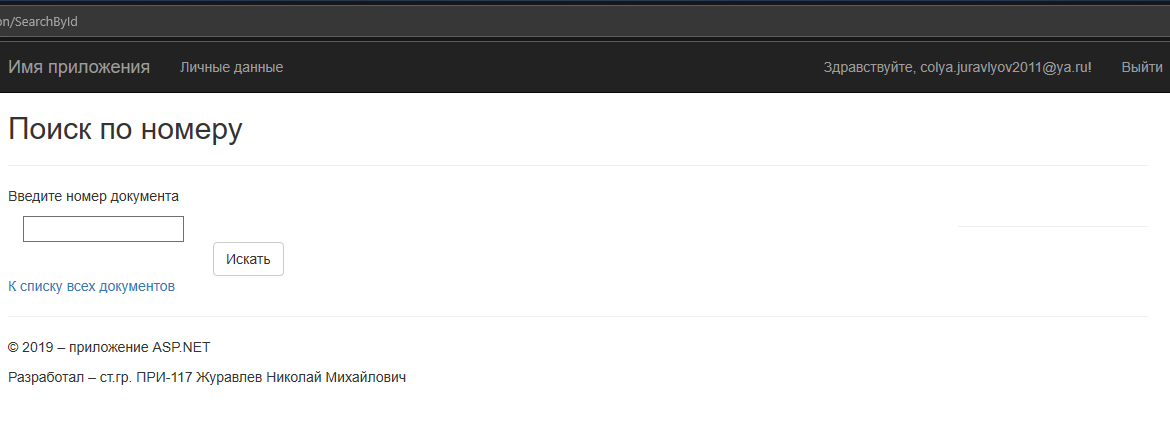


Рисунок 10. Страница поиска документа по номеру.

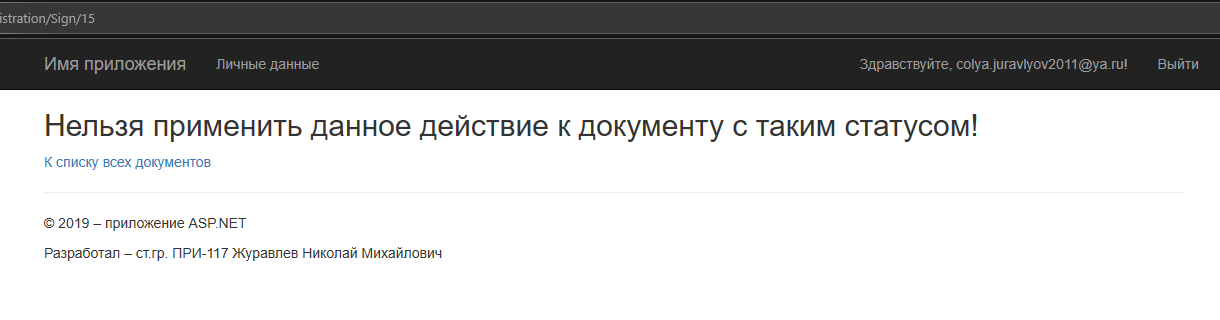


Рисунок 11. Страница, вызывающаяся при попытке произвести недоступное действие.

4.3. Прототип системы

Код основных классов системы представлен в приложении. Полностью код разработанной системы хранится в удаленном репозитории по ссылке: [https://github.com/seeyuointhespring/DocumentsCirculation](https://github.com/seeyuointhespring/DocumentsCirculation%20)

4.4. Тестирование системы

В рамках проверки работоспособности разработанной системы был написан класс для тестирования методов доступа к данным: создание, вызов и удаление записи с информацией о документе.

Код разработанного теста и результат его работы представлены в приложении Б, Рисунок(17-18).

3.5. Использованные средства

По мимо технологий описанных выше, в процессе разработки системы были использованны операции логгирования для выявления ошибок доступа к базе данных и определения информации о выполнении приложения.

В системе автоматически была реализованна система аутентификации, в связи с чем в проект добавилась база данных описывающая всех пользователей системы и их роли. Также автоматически был создан стандартный интерфейс ASP.NET MVC приложения.

# ЗАКЛЮчение

В ходе курсового проекта была разработана система автоматизации документооборота для малого предприятия. Система успешно осуществляет хранение и контроль данных, манипулирование ими, а также упрощает и структурирует процессы учета, просмотра, добавления, удаления и изменения информации о документах для сотрудников предприятия (пользователей системы). В ходе разработки программной системы были осуществлены все поставленные цели и задачи.

# список источников

1. Практическая программная инженерия на основе учебного примера / Л. А. Мацяшек, Б. Л. Лионг; пер. с англ. - 2-е изд. (эл.). - М.: БИНОМ. Лаборатория знаний, 2012. - 956 с.
2. Введение в проектирование и разработку приложений на языке программирования С# / Э.Н. Самохвалов, Г.И. Ревунков, Ю.Е. Гапанюк - М.: Издательство МГТУ им. Н. Э. Баумана, 2018. -248 с.
3. Липаев В.В. Проектирование программных систем. М.: Высш.шк, 1990
4. Шмуллер Дж. Освой самостоятельно UML за 24 часа. 3-е изд. М.: изд-дом «Вильямс», 2005
5. Microsoft Docs. Руководство по программированию на C#. [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/aspnet/web-pages/overview/getting-started/introducing-razor-syntax-c
6. Эндрю Троелсен, Филипп Джепикс Язык прграммирования C# 6.0 и платформа .NET 4.6 / М.: Вильямс, 2016. -1040 с.
7. Оптимизация приложений на платформе .NET / Голдштейн С., Зурбалев Д., Флатов И. и др. - М.: ДМК Пресс, 2014.
8. Habr.com Сообщество IT-специалистов [Электронный ресурс]. URL: https://habr.com/ru/post/176001/

ПРИЛОЖЕНИЕ А  
Схемы, таблицы и диаграммы созданные в процессе анализа предметной области

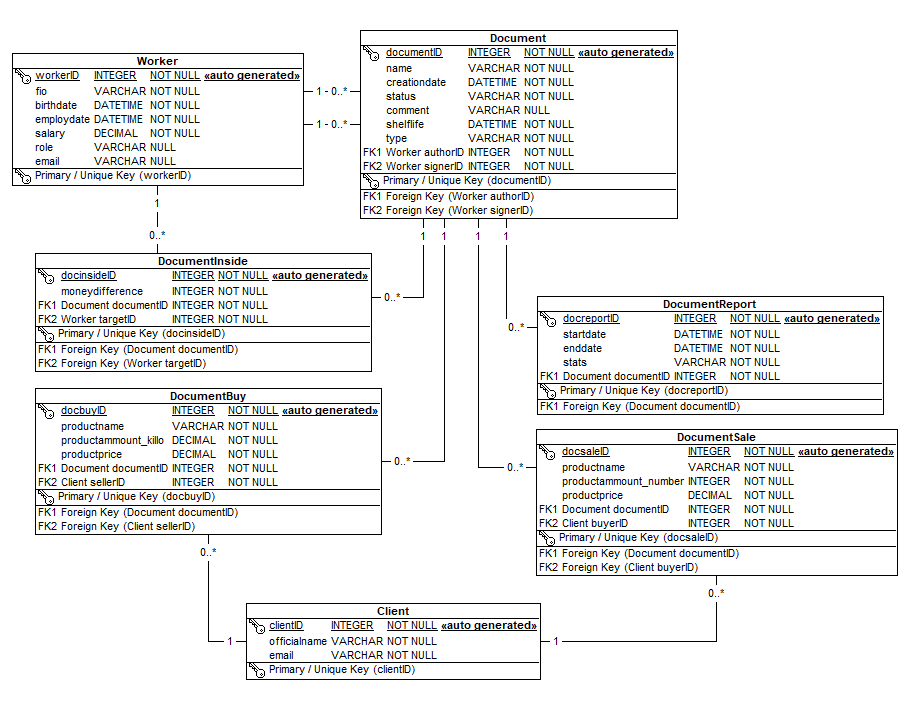


Рисунок 12. Физическая диаграмма базы данных.

Рисунок 13. Диаграмма классов.

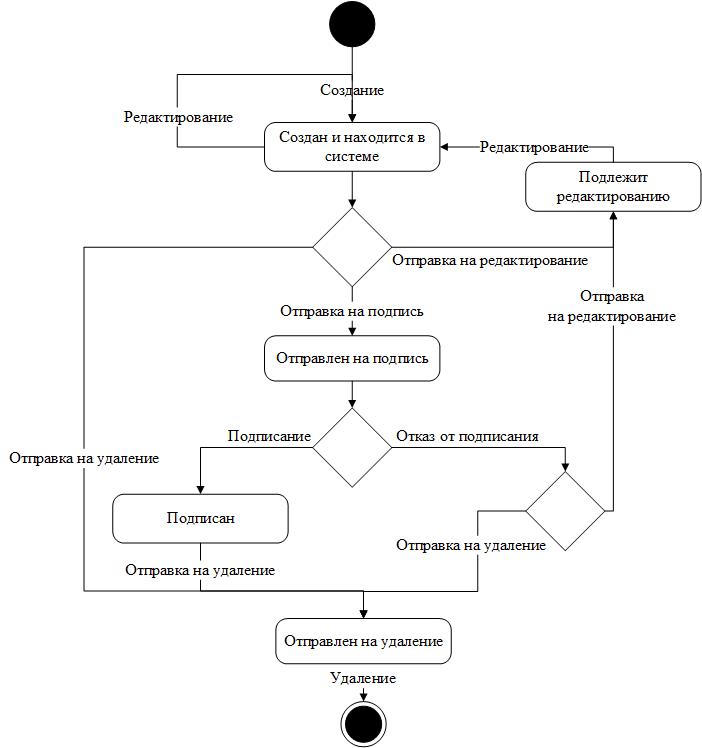


Рисунок 14. Диаграмма состояний.



Рисунок 15. Диаграмма прецедентов.



Рисунок 16. Диаграмма последовательностей.

ПРИЛОЖЕНИЕ Б  
Коды классов разработанной системы

**МОДЕЛИ (MODELS)**

**Document.cs:**

using System;

using System.ComponentModel.DataAnnotations;

namespace DocumentsCirculation.Models

{

public class Document

{

[Display(Name = "Номер документа")]

public int documentID { get; set; }

[Display(Name = "Название")]

public string name { get; set; }

[DataType(DataType.Date)]

[Display(Name = "Дата создания")]

public DateTime creationdate { get; set; }

[Display(Name = "IDавтора")]

public int authorID { get; set; }

[Display(Name = "Статус")]

public string status { get; set; }

[Display(Name = "Комментарий")]

public string comment { get; set; }

[DataType(DataType.Date)]

[Display(Name = "Хранить до")]

public DateTime shelflife { get; set; }

[Display(Name = "ID подписывающего")]

public int signerID { get; set; }

[Display(Name = "Тип")]

public string type { get; set; }

}

}

**DocumentInside.cs:**

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace DocumentsCirculation.Models

{

public class DocumentInside : Document

{

public int docinsideID { get; set;}

[Display(Name = "Разница в деньгах")]

public int moneydifference { get; set;}

[Display(Name = "ID рабочего")]

public int targetID { get; set; }

public List<DocumentInside> DocumentInsedeList { get; set; }

public string toString()

{

return "Название: " + name + " Дата создания: " + creationdate + " ID автора: " + authorID + " Хранить до: " +shelflife +

" ID подписывающего: " + signerID+" Разница в деньгах: "+moneydifference+" ID рабочего: "+targetID;

}

}

}

**КЛАССЫ ДОСТУПА К ДАННЫМ (DAO)**

**DAO.cs:**

using System.Data.SqlClient;

namespace DocumentsCirculation.DAO

{

public class DAO

{

//private const string ConnectionString = @"Initial Catalog = DocCirculation;" + @"Data Source=.\SQLEXPRESS;" + @"Integrated Security=True;" + @"Pooling=False";

private readonly string ConnectionString = System.Configuration.ConfigurationManager.ConnectionStrings["ConnectDocCirculation"].ConnectionString;

public SqlConnection Connection { get; set; }

public void Connect()

{

Connection = new SqlConnection(ConnectionString);

Connection.Open();

Logger.InitLogger();

}

public void Disconnect()

{

Connection.Close();

}

}

}

**DocInsideDAO.cs:**

using DocumentsCirculation.Models;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

namespace DocumentsCirculation.DAO

{

public class DocInsideDAO: DAO

{

public List<DocumentInside> GetAllInsides()

{

Logger.InitLogger();

Logger.Log.Info("Метод вызова всех записей");

Connect();

List<DocumentInside> DList = new List<DocumentInside>();

try

{

SqlCommand command = new SqlCommand("select \* from Document, DocumentInside where Document.documentID=DocumentInside.documentID", Connection);

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

DocumentInside inside = new DocumentInside();

inside.documentID = Convert.ToInt32(reader["documentID"]);

inside.name = Convert.ToString(reader["name"]);

inside.creationdate = Convert.ToDateTime(reader["creationdate"]);

inside.authorID = Convert.ToInt32(reader["authorID"]);

inside.status = Convert.ToString(reader["status"]);

inside.comment = Convert.ToString(reader["comment"]);

inside.shelflife = Convert.ToDateTime(reader["shelflife"]);

inside.signerID = Convert.ToInt32(reader["signerID"]);

inside.type = Convert.ToString(reader["type"]);

inside.moneydifference = Convert.ToInt32(reader["moneydifference"]);

inside.targetID = Convert.ToInt32(reader["targetID"]);

inside.documentID = Convert.ToInt32(reader["documentID"]);

DList.Add(inside);

}

reader.Close();

}

catch (Exception)

{

// Обработка исключения

}

finally

{

Disconnect();

}

return DList;

}

public bool AddInside(DocumentInside inside)

{

bool result = true;

Connect();

try

{

SqlCommand addparent = new SqlCommand("insert into Document (name, creationdate, authorID, status, comment, shelflife, signerID, type) "

+ "VALUES (@name, @creationdate, @authorID, @status, @comment, @shelflife, @signerID, @type)", Connection);

SqlCommand addheir = new SqlCommand("insert into DocumentInside (moneydifference, targetID, documentID)"

+ "values (@moneydifference, @targetID, @documentID)", Connection);

addparent.Parameters.Add(new SqlParameter("@name", inside.name));

addparent.Parameters.Add(new SqlParameter("@creationdate", inside.creationdate));

addparent.Parameters.Add(new SqlParameter("@authorID", inside.authorID));

addparent.Parameters.Add(new SqlParameter("@status", "Создан"));

addparent.Parameters.Add(new SqlParameter("@comment", ""));

addparent.Parameters.Add(new SqlParameter("@shelflife", inside.shelflife));

addparent.Parameters.Add(new SqlParameter("@signerID", inside.signerID));

addparent.Parameters.Add(new SqlParameter("@type", "Внутренний"));

addparent.ExecuteNonQuery();

addparent.CommandText = "Select @@Identity";

int id = Convert.ToInt32(addparent.ExecuteScalar());

addheir.Parameters.Add(new SqlParameter("@moneydifference", inside.moneydifference));

addheir.Parameters.Add(new SqlParameter("@targetID", inside.targetID));

addheir.Parameters.Add(new SqlParameter("@documentID", id));

addheir.ExecuteNonQuery();

}

catch (Exception e)

{

Logger.Log.Error("ERROR: " + e.Message);

result = false;

}

finally { Disconnect(); }

return result;

}

public bool ChangeInside(int id, DocumentInside inside)

{

bool result = true;

Connect();

try

{

string forheir = string.Format("update DocumentInside set moneydifference=@moneydifference, targetID=@targetID " +

"where documentID='{0}'", id);

string forparent = string.Format("update Document set name=@name, creationdate=@creationdate, authorID=@authorID," +

" status=@status, shelflife=@shelflife, signerID=@signerID where documentID='{0}'", id);

SqlCommand changeheir = new SqlCommand(forheir, Connection);

SqlCommand changeparent = new SqlCommand(forparent, Connection);

changeheir.Parameters.AddWithValue("@moneydifference", inside.moneydifference);

changeheir.Parameters.AddWithValue("@targetID", inside.targetID);

changeparent.Parameters.AddWithValue("@name", inside.name);

changeparent.Parameters.AddWithValue("@creationdate", inside.creationdate);

changeparent.Parameters.AddWithValue("@authorID", inside.authorID);

changeparent.Parameters.AddWithValue("@status", "Создан");

changeparent.Parameters.AddWithValue("@shelflife", inside.shelflife);

changeparent.Parameters.AddWithValue("@signerID", inside.signerID);

changeparent.Parameters.AddWithValue("@type", inside.type);

changeheir.ExecuteNonQuery();

changeparent.ExecuteNonQuery();

}

catch (Exception e)

{

Logger.Log.Error("ERROR: " + e.Message);

result = false;

}

finally { Disconnect(); }

return result;

}

}

}

**AdministrationDAO.cs:**

using DocumentsCirculation.Models;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using DocumentsCirculation.DAO;

namespace DocumentsCirculation.DAO

{

public class AdministrationDAO:DAO

{

public bool DropDoc(int id)

{

Logger.InitLogger();

Logger.Log.Info("Метод удаления записи");

bool result = true;

Connect();

Document doc = new Document();

//string type;

try

{

string forinside = string.Format("Delete from DocumentInside where documentID='{0}'", id);

string forreport = string.Format("Delete from DocumentReport where documentID='{0}'", id);

string forbuy = string.Format("Delete from DocumentBuy where documentID='{0}'", id);

string forsale = string.Format("Delete from DocumentSale where documentID='{0}'", id);

string forparent = string.Format("Delete from Document where documentID='{0}'", id);

SqlCommand dropinside = new SqlCommand(forinside, Connection);

SqlCommand dropreport = new SqlCommand(forreport, Connection);

SqlCommand dropsale = new SqlCommand(forsale, Connection);

SqlCommand dropbuy = new SqlCommand(forbuy, Connection);

SqlCommand dropparent = new SqlCommand(forparent, Connection);

//string type;

string forgetting = string.Format("Select type from Document where documentID='{0}'", id);

SqlCommand gettype = new SqlCommand(forgetting, Connection);

SqlDataReader reader = gettype.ExecuteReader();

while (reader.Read())

{

doc.type = Convert.ToString(reader["type"]);

Logger.Log.Info("Значение переменной doc.type:"+doc.type);

}

reader.Close();

Logger.Log.Info("Значение переменной doc.type после закрытия reader:" + doc.type);

switch (doc.type)

{

case "Продажи":

dropsale.ExecuteNonQuery();

dropparent.ExecuteNonQuery();

break;

case "Покупки":

dropbuy.ExecuteNonQuery();

dropparent.ExecuteNonQuery();

break;

case "Внутренний":

dropinside.ExecuteNonQuery();

dropparent.ExecuteNonQuery();

break;

case "Отчет":

dropreport.ExecuteNonQuery();

dropparent.ExecuteNonQuery();

break;

default:

//ошибка

break;

}

}

catch (Exception e)

{

Logger.Log.Error("ERROR: " + e.Message);

result = false;

}

finally { Disconnect(); }

return result;

}

public bool SendForSign(int id)

{

bool result = true;

Connect();

try

{

string forsend = string.Format("Update Document set status=@status where documentID='{0}'", id);

SqlCommand sendforsign = new SqlCommand(forsend, Connection);

sendforsign.Parameters.AddWithValue("@status", "Отправлен на подписание");

sendforsign.ExecuteNonQuery();

}

catch (Exception)

{

result = false;

}

finally { Disconnect(); }

return result;

}

public bool Sign(int id)

{

bool result = true;

Connect();

try

{

string forsign = string.Format("Update Document set status=@status where documentID='{0}'", id);

SqlCommand sign = new SqlCommand(forsign, Connection);

sign.Parameters.AddWithValue("@status", "Подписан");

sign.ExecuteNonQuery();

}

catch (Exception)

{

result = false;

}

finally { Disconnect(); }

return result;

}

public bool SendForChange(int id,Document doc)

{

bool result = true;

Connect();

try

{

string forchange = string.Format("Update Document set status=@status, comment=@comment where documentID='{0}'", id);

SqlCommand sendforchange = new SqlCommand(forchange, Connection);

sendforchange.Parameters.AddWithValue("@comment", doc.comment);

sendforchange.Parameters.AddWithValue("@status", "Подлежит редактированию");

sendforchange.ExecuteNonQuery();

}

catch (Exception)

{

result = false;

}

finally { Disconnect(); }

return result;

}

public bool SendForDrop(int id)

{

bool result = true;

Connect();

try

{

string forsend = string.Format("Update Document set status=@status where documentID='{0}'", id);

SqlCommand sendforsign = new SqlCommand(forsend, Connection);

sendforsign.Parameters.AddWithValue("@status", "Отправлен на удаление");

sendforsign.ExecuteNonQuery();

}

catch (Exception)

{

result = false;

}

finally { Disconnect(); }

return result;

}

public List<Document> GetAll()

{

Connect();

List<Document> DList = new List<Document>();

try

{

SqlCommand command = new SqlCommand("select \* from Document", Connection);

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

DocumentSale doc = new DocumentSale();

doc.documentID = Convert.ToInt32(reader["documentID"]);

doc.name = Convert.ToString(reader["name"]);

doc.creationdate = Convert.ToDateTime(reader["creationdate"]);

doc.authorID = Convert.ToInt32(reader["authorID"]);

doc.status = Convert.ToString(reader["status"]);

doc.comment = Convert.ToString(reader["comment"]);

doc.shelflife = Convert.ToDateTime(reader["shelflife"]);

doc.signerID = Convert.ToInt32(reader["signerID"]);

doc.type = Convert.ToString(reader["type"]);

DList.Add(doc);

}

reader.Close();

}

catch (Exception)

{

// Обработка исключения

}

finally

{

Disconnect();

}

return DList;

}

public List<Document> GetAllWaitingForSign()

{

Connect();

List<Document> DList = new List<Document>();

try

{

SqlCommand command = new SqlCommand("select \* from Document where status='Отправлен на подписание'", Connection);

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

DocumentSale doc = new DocumentSale();

doc.documentID = Convert.ToInt32(reader["documentID"]);

doc.name = Convert.ToString(reader["name"]);

doc.creationdate = Convert.ToDateTime(reader["creationdate"]);

doc.authorID = Convert.ToInt32(reader["authorID"]);

doc.status = Convert.ToString(reader["status"]);

doc.comment = Convert.ToString(reader["comment"]);

doc.shelflife = Convert.ToDateTime(reader["shelflife"]);

doc.signerID = Convert.ToInt32(reader["signerID"]);

doc.type = Convert.ToString(reader["type"]);

DList.Add(doc);

}

reader.Close();

}

catch (Exception)

{

// Обработка исключения

}

finally

{

Disconnect();

}

return DList;

}

public Document GetById(int id)

{

Connect();

Document D = new Document();

try

{

string forsearch = string.Format("select \* from Document where documentID='{0}'", id);

SqlCommand command = new SqlCommand(forsearch, Connection);

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

DocumentSale doc = new DocumentSale();

doc.documentID = Convert.ToInt32(reader["documentID"]);

doc.name = Convert.ToString(reader["name"]);

doc.creationdate = Convert.ToDateTime(reader["creationdate"]);

doc.authorID = Convert.ToInt32(reader["authorID"]);

doc.status = Convert.ToString(reader["status"]);

doc.comment = Convert.ToString(reader["comment"]);

doc.shelflife = Convert.ToDateTime(reader["shelflife"]);

doc.signerID = Convert.ToInt32(reader["signerID"]);

doc.type = Convert.ToString(reader["type"]);

D = doc;

}

reader.Close();

}

catch (Exception)

{

// Обработка исключения

}

finally

{

Disconnect();

}

return D;

}

}

}

**КОНТРОЛЛЕРЫ (CONTROLLERS)**

**DocInsideController.cs:**

using DocumentsCirculation.DAO;

using DocumentsCirculation.Models;

using System;

using System.Collections.Generic;

using System.Web.Mvc;

namespace DocumentsCirculation.Controllers

{

public class DocInsideController : Controller

{

DocInsideDAO docinside = new DocInsideDAO();

AdministrationDAO admin = new AdministrationDAO();

// GET: DocInside

[Authorize(Roles = "SysAdmin, Administrator, Director, Accountant")]

public ActionResult DocInsideIndex()

{

return View(docinside.GetAllInsides());

}

// GET: DocInside/Details/5

[Authorize(Roles = "SysAdmin, Administrator, Director, Accountant")]

public ActionResult DocInsideDetails(int id)

{

List<DocumentInside> diList = docinside.GetAllInsides();

int pos = 0;

for (int i = 0; i < diList.Count; i++)

if (id == diList[i].documentID)

{

pos = i;

}

return View(diList[pos]);

}

// GET: DocInside/Create

[Authorize(Roles = "SysAdmin, Accountant")]

public ActionResult DocInsideCreate()

{

return View();

}

// POST: DocInside/Create

[HttpPost]

public ActionResult DocInsideCreate(DocumentInside di)

{

try

{

if (docinside.AddInside(di))

return RedirectToAction("DocInsideIndex");

else return View("DocInsideCreate");

}

catch

{

return View("DocInsideCreate");

}

}

// GET: DocInside/Edit/5

[Authorize(Roles = "SysAdmin, Accountant")]

public ActionResult DocInsideEdit(int id)

{

List<DocumentInside> diList = docinside.GetAllInsides();

int pos = 0;

for (int i = 0; i < diList.Count; i++)

if (id == diList[i].documentID)

{

pos = i;

}

if (diList[pos].status == "Подлежит редактированию" | diList[pos].status == "Создан")

{

return View(diList[pos]);

}

else return View("WrongStatus");

}

// POST: DocInside/Edit/5

[HttpPost]

public ActionResult DocInsideEdit(int id, DocumentInside di)

{

try

{

if (docinside.ChangeInside(id,di))

return RedirectToAction("DocInsideIndex");

else return View("DocInsideEdit");

}

catch

{

return View("DocInsideEdit");

}

}

// GET: DocInside/Delete/5

[Authorize(Roles = "SysAdmin, Administrator")]

public ActionResult DocInsideDelete(int id)

{

List<DocumentInside> diList = docinside.GetAllInsides();

int pos = 0;

for (int i = 0; i < diList.Count; i++)

if (id == diList[i].documentID)

{

pos = i;

}

if (diList[pos].status == "Отправлен на удаление")

{

return View(diList[pos]);

}

else return View("WrongStatus");

}

// POST: DocInside/Delete/5

[HttpPost]

public ActionResult DocInsideDelete(int id, FormCollection collection)

{

try

{

if (admin.DropDoc(id))

return RedirectToAction("DocInsideIndex");

else return View("DocInsideDelete");

}

catch

{

return View("DocInsideDelete");

}

}

}

}

**AdministrationController.cs:**

using DocumentsCirculation.DAO;

using DocumentsCirculation.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace DocumentsCirculation.Controllers

{

public class AdministrationController : Controller

{

static Document doc = new Document();

AdministrationDAO admin = new AdministrationDAO();

//GET

[Authorize(Roles = "SysAdmin, Administrator")]

public ActionResult SendForSign(int id)

{

List<Document> docList = admin.GetAll();

int pos = 0;

for (int i = 0; i < docList.Count; i++)

if (id == docList[i].documentID)

{

pos = i;

}

if (docList[pos].status == "Создан")

{

return View(docList[pos]);

}

else return View("WrongStatus");

}

//POST

[HttpPost]

public ActionResult SendForSign(int id, FormCollection collection)

{

try

{

if (admin.SendForSign(id))

return RedirectToAction("AllIndex");

else return View("Mistake");

}

catch

{

return View("Mistake");

}

}

//GET

[Authorize(Roles = "SysAdmin, Director")]

public ActionResult Sign(int id)

{

List<Document> docList = admin.GetAll();

int pos = 0;

for (int i = 0; i < docList.Count; i++)

if (id == docList[i].documentID)

{

pos = i;

}

if (docList[pos].status == "Отправлен на подписание")

{

return View(docList[pos]);

}

else return View("WrongStatus");

}

//POST

[HttpPost]

public ActionResult Sign(int id, FormCollection collection)

{

try

{

if (admin.Sign(id))

return RedirectToAction("AllIndex");

else return View("Mistake");

}

catch

{

return View("Mistake");

}

}

[Authorize(Roles = "SysAdmin, Administrator, Director")]

public ActionResult SendForDrop(int id)

{

List<Document> docList = admin.GetAll();

int pos = 0;

for (int i = 0; i < docList.Count; i++)

if (id == docList[i].documentID)

{

pos = i;

}

return View(docList[pos]);

}

//POST

[HttpPost]

public ActionResult SendForDrop(int id, FormCollection collection)

{

try

{

if (admin.SendForDrop(id))

return RedirectToAction("AllIndex");

else return View("Mistake");

}

catch

{

return View("Mistake");

}

}

// GET: Administration

[Authorize(Roles = "SysAdmin, Administrator, Director")]

public ActionResult AllIndex()

{

return View(admin.GetAll());

}

//Get

[Authorize(Roles = "SysAdmin, Director")]

public ActionResult WriteComment(int id)

{

List<Document> dList = admin.GetAll();

int pos = 0;

for (int i = 0; i < dList.Count; i++)

if (id == dList[i].documentID)

{

pos = i;

}

return View(dList[pos]);

}

//Post

[HttpPost]

public ActionResult WriteComment(int id, Document d)

{

try

{

if (admin.SendForChange(id,d))

return RedirectToAction("AllIndex");

else return View("Mistake");

}

catch

{

return View("Mistake");

}

}

// GET: Administration

[Authorize(Roles = "SysAdmin, Director")]

public ActionResult AllForSignIndex()

{

return View(admin.GetAllWaitingForSign());

}

//Post

[HttpPost]

public ActionResult SearchById(int id)

{

string type= "---";

List<Document> dList = admin.GetAll();

for (int i = 0; i < dList.Count; i++)

if (id == dList[i].documentID)

{

type = dList[i].type;

}

Logger.Log.Info("Значение переменной d.documentID " + id);

switch (type)

{

case "Продажи":

return RedirectToRoute(new{ controller = "DocSale", action = "DocSaleDetails",id = id });

case "Покупки":

return RedirectToRoute(new { controller = "DocBuy", action = "DocBuyDetails", id=id });

case "Внутренний":

return RedirectToRoute(new { controller = "DocInside", action = "DocInsideDetails", id= id });

case "Отчет":

return RedirectToRoute(new { controller = "DocReport", action = "DocReportDetails", id= id });

default:

return View("Mistake");

}

}

//Get

[Authorize(Roles = "SysAdmin, Director, Administrator")]

public ActionResult SearchById()

{

try

{

return View();

}

catch

{

return View("WrongStatus");

}

}

}

}

**ПРЕДСТАВЛЕНИЯ (VIEWS)**

**/Home/Index.cshtml:**

@{

ViewBag.Title = "Home Page";

}

<div>

<h2>Меню</h2>

<table class="table">

<tr>

<td>Работа с конкретным<br> типом документов</td>

<td>@Html.RouteLink("Все документы", new { controller = "Administration", action = "AllIndex" }, new { @class = "link" })</td>

<td>@Html.RouteLink("Все работники", new { controller = "Worker", action = "WorkerIndex" }, new { @class = "link" })</td>

<td>@Html.RouteLink("Все клиенты", new { controller = "Client", action = "ClientIndex" }, new { @class = "link" })</td>

</tr>

<tr>

<td>@Html.RouteLink("Документы покупки", new { controller = "DocBuy", action = "DocBuyIndex" }, new { @class = "link" })</td>

<td>@Html.RouteLink("Поиск документов ожидающих подписания", new { controller = "Administration", action = "AllForSignIndex" }, new { @class = "link" })</td>

</tr>

<tr>

<td>@Html.RouteLink("Документы продажи", new { controller = "DocSale", action = "DocSaleIndex" }, new { @class = "link" })</td>

<td>@Html.RouteLink("Поиск по номеру документа", new { controller = "Administration", action = "SearchById" }, new { @class = "link" })</td>

</tr>

<tr>

<td>@Html.RouteLink("Документы отчета", new { controller = "DocReport", action = "DocReportIndex" }, new { @class = "link" })</td>

<td>TBD</td>

</tr>

<tr>

<td>@Html.RouteLink("Внутренние документы", new { controller = "DocInside", action = "DocInsideIndex" }, new { @class = "link" })</td>

<td>TBD</td>

</tr>

</table>

</div>

**AllIndex.cshtml:**

@model IEnumerable<DocumentsCirculation.Models.Document>

@{

ViewBag.Title = "AllIndex";

}

<h2>Список всех документов</h2>

<table class="table">

<tr>

<th>

Номер документа

</th>

<th>

Название

</th>

<th>

Дата создания

</th>

<th>

ID автора

</th>

<th>

Статус

</th>

<th>

Комментарий

</th>

<th>

Хранить до

</th>

<th>

ID подписывающего

</th>

<th>

Тип

</th>

<th>

Действия

</th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.documentID)

</td>

<td>

@Html.DisplayFor(modelItem => item.name)

</td>

<td>

@Html.DisplayFor(modelItem => item.creationdate)

</td>

<td>

@Html.DisplayFor(modelItem => item.authorID)

</td>

<td>

@Html.DisplayFor(modelItem => item.status)

</td>

<td>

@Html.DisplayFor(modelItem => item.comment)

</td>

<td>

@Html.DisplayFor(modelItem => item.shelflife)

</td>

<td>

@Html.DisplayFor(modelItem => item.signerID)

</td>

<td>

@Html.DisplayFor(modelItem => item.type)

</td>

<td>

@switch (item.type)

{

case "Продажи":

@Html.RouteLink("Просмотреть", new { controller = "DocSale", action = "DocSaleDetails", id = item.documentID }, new { id = item.documentID, @class = "link" })

break;

case "Покупки":

@Html.RouteLink("Просмотреть", new { controller = "DocBuy", action = "DocBuyDetails", id = item.documentID }, new { id = item.documentID, @class = "link" })

break;

case "Отчет":

@Html.RouteLink("Просмотреть", new { controller = "DocReport", action = "DocReportDetails", id = item.documentID }, new { id = item.documentID, @class = "link" })

break;

case "Внутренний":

@Html.RouteLink("Просмотреть", new { controller = "DocInside", action = "DocInsideDetails", id = item.documentID }, new { id = item.documentID, @class = "link" })

break;

default:

<p>Не верный тип</p>

break;

}

</td>

</tr>

}

</table>

**DocInsideCreate.cshtml:**

@model DocumentsCirculation.Models.DocumentInside

@{

ViewBag.Title = "DocInsideCreate";

}

<h2>Создание внутреннего документа</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Введите данные внутреннего документа</h4>

<hr />

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

<div class="form-group">

@Html.LabelFor(model => model.name, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.name, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.name, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.creationdate, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.creationdate, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.creationdate, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.authorID, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.authorID, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.authorID, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.shelflife, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.shelflife, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.shelflife, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.signerID, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.signerID, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.signerID, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.moneydifference, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.moneydifference, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.moneydifference, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.targetID, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.targetID, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.targetID, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Создать" class="btn btn-default" />

</div>

</div>

</div>

}

<div>

@Html.ActionLink("К списку внутренних документов", "DocInsideIndex")

</div>

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

**DocInsideDetails.cshtml:**

@model DocumentsCirculation.Models.DocumentInside

@{

ViewBag.Title = "DocInsideDetails";

}

<h2>Просмотр внутреннего документа</h2>

<div>

<h4>Внутренний документ</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.documentID)

</dt>

<dd>

@Html.DisplayFor(model => model.documentID)

</dd>

<dt>

@Html.DisplayNameFor(model => model.name)

</dt>

<dd>

@Html.DisplayFor(model => model.name)

</dd>

<dt>

@Html.DisplayNameFor(model => model.creationdate)

</dt>

<dd>

@Html.DisplayFor(model => model.creationdate)

</dd>

<dt>

@Html.DisplayNameFor(model => model.authorID)

</dt>

<dd>

@Html.DisplayFor(model => model.authorID)

</dd>

<dt>

@Html.DisplayNameFor(model => model.status)

</dt>

<dd>

@Html.DisplayFor(model => model.status)

</dd>

<dt>

@Html.DisplayNameFor(model => model.comment)

</dt>

<dd>

@Html.DisplayFor(model => model.comment)

</dd>

<dt>

@Html.DisplayNameFor(model => model.shelflife)

</dt>

<dd>

@Html.DisplayFor(model => model.shelflife)

</dd>

<dt>

@Html.DisplayNameFor(model => model.signerID)

</dt>

<dd>

@Html.DisplayFor(model => model.signerID)

</dd>

<dt>

@Html.DisplayNameFor(model => model.type)

</dt>

<dd>

@Html.DisplayFor(model => model.type)

</dd>

<dt>

@Html.DisplayNameFor(model => model.moneydifference)

</dt>

<dd>

@Html.DisplayFor(model => model.moneydifference)

</dd>

<dt>

@Html.DisplayNameFor(model => model.targetID)

</dt>

<dd>

@Html.DisplayFor(model => model.targetID)

</dd>

</dl>

</div>

<hr />

<p>

@Html.ActionLink("Редактировать", "DocInsideEdit", new { id = Model.documentID })|

@Html.ActionLink("К списку внутренних документов", "DocInsideIndex")

</p>

<hr />

<p>

@Html.ActionLink("К списку всех документов", "AllIndex", "Administration") |

@Html.RouteLink("Отправить на редактирование", new { controller = "Administration", action = "WriteComment", id = Model.documentID }, new { id = Model.documentID, @class = "link" }) |

@Html.RouteLink("Отправить на подписание", new { controller = "Administration", action = "SendForSign", id = Model.documentID }, new { id = Model.documentID, @class = "link" }) |

@Html.RouteLink("Подписать", new { controller = "Administration", action = "Sign", id = Model.documentID }, new { id = Model.documentID, @class = "link" }) |

@Html.RouteLink("Отправить на удаление", new { controller = "Administration", action = "SendForDrop", id = Model.documentID }, new { id = Model.documentID, @class = "link" })|

@Html.ActionLink("Удалить", "DocInsideDelete", new { id = Model.documentID })

</p>

**ТЕСТЫ (TESTS)**

**TestInsertSelectClass.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using NUnit.Framework;

using DocumentsCirculation.Models;

using DocumentsCirculation.DAO;

namespace DocumentsCirculation

{

[TestFixture]

public class TestInsertSelectClass

{

/\*Тест для класса DocInsideDAO, который осуществляет доступ к бд

В тесте вызываются и прверяются операции добавления, выборки и удаления объекта внутреннего документа \*/

[Test]

public void TestDocInsideDAO()

{

DocumentInside docInside = new DocumentInside();

DocInsideDAO dao = new DocInsideDAO();

AdministrationDAO admindao = new AdministrationDAO();

docInside.name = "Штраф за опоздание";

docInside.creationdate = Convert.ToDateTime("2019-12-12");

docInside.authorID = 1;

docInside.shelflife = Convert.ToDateTime("2022-12-12");

docInside.signerID = 1;

docInside.moneydifference = -1000;

docInside.targetID = 2;

dao.AddInside(docInside);

List<DocumentInside> diList = dao.GetAllInsides();

int pos = diList.Count()-1;

Assert.AreEqual(docInside.toString(),diList[pos].toString());

admindao.SendForDrop(diList[pos].documentID);

admindao.DropDoc(diList[pos].documentID);

diList = dao.GetAllInsides();

pos = diList.Count() - 1;

Assert.AreNotEqual(docInside.toString(), diList[pos].toString());

}

}

}

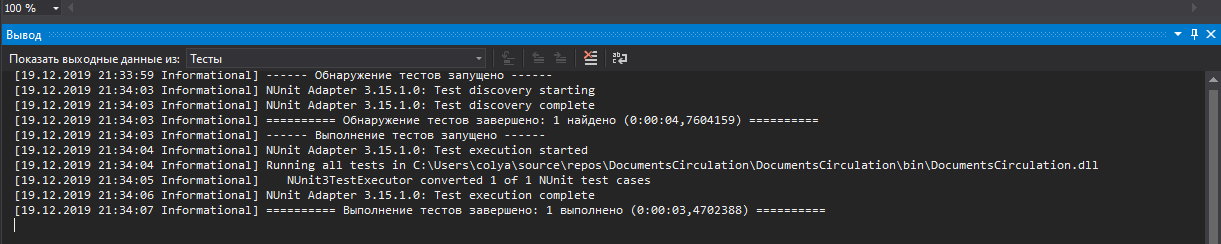


Рисунок 17. Выполнение тестов.

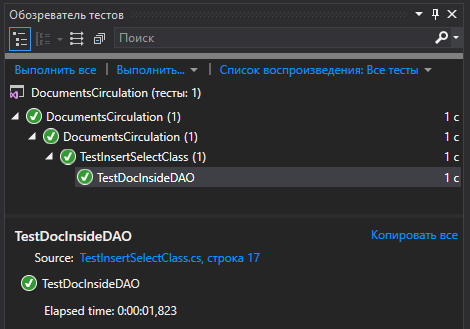


Рисунок 18. Результат выполнения тестов.