МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Политехнический институт: Заочный
Кафедра «Системы автоматического управления»

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой

_____/ В.И. Ширяев

«_____» _____ 2018 г.

АВТОМАТИЗАЦИЯ ВЫДАЧИ ПРОПУСКОВ НА ОХРАНЯЕМУЮ АВТОМОБИЛЬНУЮ СТОЯНКУ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ – 09.03.01.2018.382.00 ПЗ ВКР

Руководитель работы

Нач. отдела инф. технологий

_____/ Г.В. Легостаев

«____» _____ 2018 г.

Автор работы

студент группы ПЗ-597

_____/ А.П. Митрофанов

«____» _____ 2018 г.

Нормоконтролер

Старший преподаватель каф. САУ

_____/ В.П. Щербаков

«____» _____ 2018 г.

Челябинск 2018

Аннотация

Митрофанов А.П. Автоматизация выдачи пропусков на охраняемую автомобильную стоянку: ЮУрГУ, ПИ: Заочный; 2018, 60 с. 23 ил., библиогр. список – 10 наим., 12 слайдов презентации ф. А4.

В выпускной работе рассматривается процесс создания программного обеспечения для выдачи пропусков на стоянку. Программное обеспечение представляет собой комплекс программ, состоящей из серверной части, написанных на языке высокого уровня C# в интегрированной среде разработки visual studio 2017, с поддержкой веб платформы MVC и программной платформы .NET

Так же выполнено описание комплекса программ, разработанных в результате выполнения дипломного проекта, оценены аналогичные готовые решения, рассчитаны быстродействие, надежность, резервирование, удаленный доступ.

В заключительной части сделаны выводы по эффективности разработки и оценены перспективы ее дальнейшего развития.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность работы. В наше время мало кто задумывается, на что он тратит своё время. Часто сидя на работе, мы не замечаем, как быстро проходит рабочий день, и что мы успеваем сделать за этот небольшой промежуток времени. Бывает ощущение, что «вроде» работал, но, если задуматься, выходит так, что «серьёзных» дел то и не делал вовсе, так, прочитали пару статей, да подписал несколько документов.

Сейчас, для своего удобства, человек пытается автоматизировать практически всё, начиная от разогревания пищи, до беспилотных самолётов. Это не удивительно, ведь тем проще процесс работы, тем больше сил и нервов остаётся на свободное от работы время.

Остаётся только найти самый оптимальный способ, для того чтобы выполнялись все поставленные задачи и чтобы не пришлось переплачивать за не «нужные» функции.

Всё вышеперечисленное делает актуальным и разработку для автоматизации выдачи пропусков на охраняемую автомобильную стоянку.

Цель проекта написать программу для автоматической выдачи пропусков на автомобильную стоянку по конкретному шаблону для ЧОКБ.

Объектом исследования является автоматизация выдачи пропусков.

Предметом исследования являются программные алгоритмы автоматической выдачи пропусков.

Для достижения поставленной цели в работе будут решены следующие задачи:

1. Рассмотрение существующих алгоритмов автоматизации выдачи пропусков.

2. Написание новых алгоритмов для реализации поставленных задач.

3. Выбор более подходящей платформы и инструментов для оптимизации программы.

Новизна. Предлагается алгоритм автоматической выдачи пропусков на охраняемую автомобильную стоянку, определяется и оценивается его эффективность, осуществляется комплексный анализ работы алгоритма в сравнении с другим, применяемым в этой области программным алгоритмам.

1 АНАЛИТИЧЕСКИЙ РАЗДЕЛ

Программа – это набор инструкций и/или алгоритмов, подробно описывающих способы определенных действий. Большинство пользователей используют ПК для конкретных задач, просмотр фильмов, оформление документов и т.п. Такие программные средства называют прикладными. Управление компонентами вычислительной системы и формирование среды для функционирования прикладных программ берёт на себя системное программное обеспечение, наиболее важной составляющей которого является операционная система [6].

В системном программировании программой называются данные, которые используются процессором как инструкции по управлению компьютерной системой. В состав программы может входить как машинный код, исполняемый процессором для достижения некоторой цели, так и необходимые для этого данные. Отличительной особенностью программы является её нахождение в памяти и исполнение процессором.

Системные программы – это программы, которые обеспечивают взаимодействие пользователя с компьютером и создают среду, в которой выполняются прикладные программы. К ним можно отнести «драйвера» и «операционные системы».

Язык программирования — язык, предназначенный для записи программ для ПК, он определяет лексические, синтаксические и семантические правила, определяющие действия и вид программ, выполняющиеся исполнителем. Предназначен для написания ПК программ, представляющих собой набор правил для выполнения компьютером

В программировании существуют языки низкого и высокого уровня.

Язык программирования низкого уровня создан для использования с типом процессора и учитывающий его особенности, так как язык близок к машинному коду (позволяет непосредственно реализовать некоторые команды процессора).

Языки низкого уровня мало похожи на нормальный, привычный человеку язык. Большие программы на таких языках пишутся редко. Зато если программа будет написана на таком языке, то она будет работать быстро, занимая маленький объем и допуская минимальное количество ошибок. Чем ниже и ближе к машинному уровень языка, тем меньше и конкретнее задачи, которые ставятся перед каждой командой.

Достоинством языков низкого уровня это создание самых эффективных программ (кратких и быстрых). Минус заключается в трудном изучении из-за необходимости полного понимания устройств процессора и что такая программа, неприменима для процессоров других типов.

Языки программирования высокого уровня проще в изучении и применении. Программы, написанные на языке программирования высокого уровня, могут быть использованы на любой компьютерной платформе, если для нее существует транслятор данного языка. Такие языки не учитывают свойства конкретных процессоров и не предоставляет прямых средств для обращения к нему. Это может ограничивать программистов, но зато и оставляет меньше возможностей для совершения ошибок.

С появлением языков высокого уровня программисты получили возможность больше времени уделять решению конкретной проблемы, не отвлекаясь на тонкие вопросы организации самого процесса выполнения задания на ПК.

Языкам высокого уровня свойственно работа с комплексными структурами данных. Во многих из них интегрирована поддержка строковых типов, объектов, операций файлового ввода-вывода и т. п. Недостатком таких языков является больший размер программ по сравнению с программами на языке низкого уровня. Поэтому на языках высокого уровня разрабатывается ПО для устройств с большим объёмом памяти. А разные подвиды ассемблера применяются для программирования других устройств, где критичным является размер программы.

Так же в дипломной работе используется язык HTML. Аббревиатура HTML расшифровывается как HyperText Markup Language. То есть HTML – это язык гипертекстовой разметки размещенных в интернете документов. Благодаря его использованию браузеры получают информацию о том, как отображать различные элементы веб-страниц. Некоторые пользователи путают HTML с языками программирования, однако с его помощью нельзя выполнить какие-либо действия – к примеру, открыть всплывающее окно. Это всего лишь разметка – подобная той, с которой сталкивался каждый пользователь программы Microsoft Word. С ее помощью можно создавать таблицы, форматировать текст, добавлять ссылки на фотографии и т. д.

Bootstrap — набор инструментов для разработки веб-приложений и сайтов. Включает HTML и CSS шаблоны оформления для кнопок, меток, веб фор м, и прочих компонентов веб-интерфейса.

Основные инструменты Bootstrap:

- Сетки — заранее изготовленные размеры колонок, которые можно сразу использовать,
- Шаблоны — фиксированный или резиновый шаблон документа.
- Типографика — описания шрифтов, определение классов для шрифтов, таких как код, цитаты и т. п.

- Медиа — представляет некоторое управление видео и изображениями.
- Таблицы — средства оформления таблиц, вплоть до добавления функциональности сортировки.
- Формы — классы для оформления форм и событий, происходящих с ними.
- Навигация — классы оформления для табов, вкладок, меню и панели инструментов.
- Алерты — оформление диалоговых окон, всплывающих окон и подсказок.

1.1 Обзор аналогов

При анализе предметной области были выявлены следующие аналоги, наиболее популярные в России:

Конфигурация "Автостоянка" - предназначена для автоматизации учета на автомобильных стоянках.

Благодаря гибкой системе учета, конфигурация «Автостоянка» позволит удобно вести учет всех клиентов автостоянки, а так же данных об их транспортных средствах. «Автостоянка» позволит контролировать процесс заездов и выездов клиентов, контролировать оплату услуг, получать информацию по должникам, а также информацию по наполняемости территории автостоянки. Получение отчетов по всем данным.

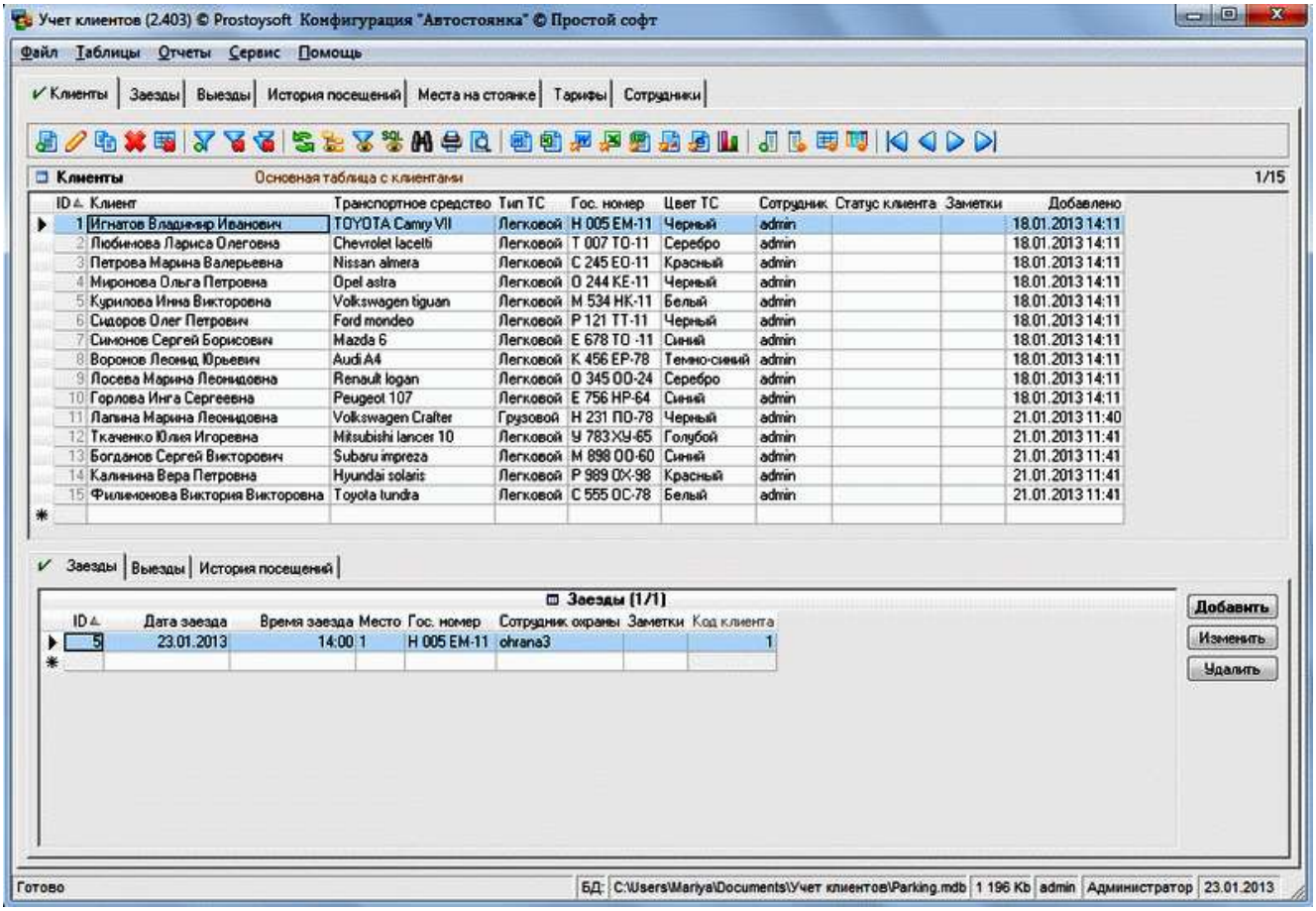

Рисунок 1.1 — Главное окно программы "Автостоянка"

Основные функции программы:

Ведение базы клиентов автостоянки

- регистрация клиентов
- регистрация транспортного средства клиента
- выбор тарифа для клиента

Ведение истории посещений

- возможность просмотра посещений за любой период времени
- учет платежей за стоянку

Регистрация заездов и выездов ТС

- регистрация транспортных средств на автостоянке
- фиксирование даты и время заезда/выезда на автостоянку
- выбор места на автостоянке

Учет мест на автостоянке

- просмотр информации о занятых местах на автостоянке
- просмотр информации о забронированных местах на автостоянке
- просмотр информации о свободных местах на автостоянке

Автостоянка 2.5.8 - Ведение учета на автостоянках

Автостоянка - программа предназначена для ведения учета на автостоянках в электронном виде. Теперь вы получите полный контроль над авто местами, сроками оплаты, базой клиентов и должников. Полностью автоматизированная автомобильная стоянка в вашем распоряжении.

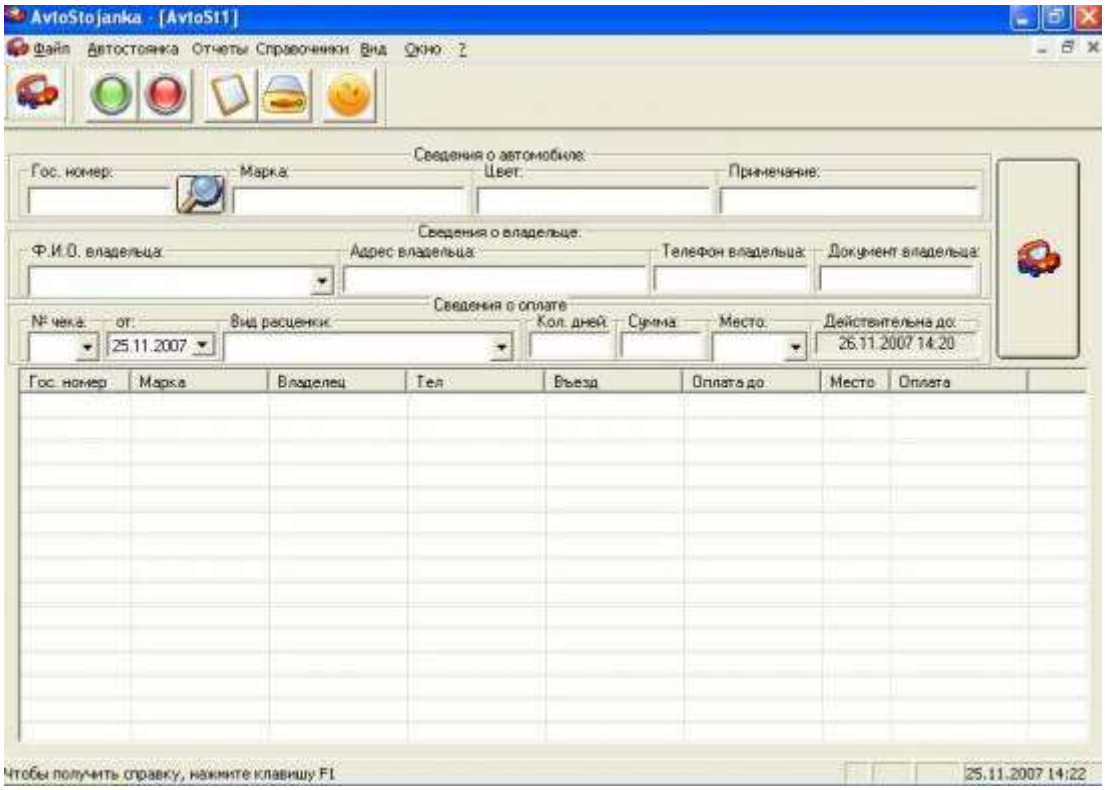

Рисунок 1.2 — Главное окно программы Автостоянка 2.5.8

Основные функции программы:

Контролирует оплату услуг автостоянки. Выписывает квитанцию об оплате услуг автостоянки.

Программа формирует следующие виды отчетов:

1) выручка – показывает принятые платежи клиентов.

2) информацию о должниках – предоставляет данные о владельцах, неоплативших услуги стоянки.

3) полный отчет о владельце – его автотранспортном средстве, когда и в какое время он посещал автостоянку и другую информацию.

4) отчет о занятости мест – показывают информацию о свободных и занятых мест.

Выполнив анализ имеющихся аналогов, можно сделать вывод о том, что в имеющихся программных комплексах отсутствует выдача пропусков какой либо формы, программы сложны в обращении, так как имеют очень обширный функционал «лишних» функций и требует немалых финансовых вложений, так

как копии данных программ, на одно рабочее место, варьируется в пределах восьми тысячи рублей.

1.2 Преимущества разрабатываемого комплекса

Главным преимуществом разрабатываемого комплекса является то, что он представляет собой специализированное программное обеспечение, удовлетворяющее необходимым требованиям и способное за короткое время выдать пропуск определённой формы. Тем самым мы сможем экономить время на составлении и подписи пропуска, а так же предотвратить финансовые потери. Мой комплекс написан на Web-интерфейсе, который является кроссплатформенным, его не нужно инсталлировать на диск, обновлять и устанавливать для него дополнительное ПО.

1.3 Постановка задачи

Необходимо разработать серверное программное обеспечение для выдачи пропусков на охраняемую автомобильную стоянку.

В серверном приложении должны быть реализованы следующие функции:

- Создан Web интерфейс
- Создана реализационная БД
- Создана автоматическая генерация пропусков на печать по шаблону
- Создан адаптивный дизайн с использованием Bootstrap
- Создание кроссплатформенности
- Поддержка штрафных баллов

Требование к шаблону пропуска:

- Номер пропуска
- Ф.И.О.
- должность
- модель автомобиля
- номер автомобиля
- место для фотографии

1.4 Обоснование выбора языка программирования

Язык программирования C++

C++ – объектно-ориентированный язык программирования, является надстройкой над C, в нем присутствуют основные принципы ООП (объектно-ориентированное программирование) наследование, инкапсуляции, полиморфизм [1].Однако программисты, использующие C++, остаются незащищенными от

многих и часто опасных особенностей C, низкоуровневыми работами с памятью и трудности в синтаксисе.

Существует множество библиотек для C++, основное назначение которых - облегчить программирование приложений под Windows, предоставив готовые классы.

Язык программирования Visual Basic

Visual Basic позволяет работать со сложными элементами интерфейса, библиотеками кода и средствами доступа к данным при минимальных затратах времени и сил [3].

Visual Basic – это язык для работы с объектами, а не объектно-ориентированный язык в обычном понимании этого слова. В Visual Basic нет классического наследования, нет поддержки создания параметризованных классов, нет собственных средств создания многопоточных приложений - и этот список можно продолжать еще долго.

Язык программирования Java

Язык программирования Java – объектно-ориентированный язык, который в отношении синтаксиса многое унаследовал от C++. Язык Java в синтаксическом отношении проще и логичнее, чем C++. Java как платформа предоставляет в распоряжение программистов большое количество библиотек, в которых содержится описаний классов и интерфейсов. С их помощью можно создавать стопроцентные приложения Java с возможностью обращения к базам данных, поддержкой передачи почтовых сообщений, с клиентской частью, которой необходим только web-браузер, или наоборот, с клиентской частью, обладающей изощренным интерфейсом.

Одна из серьезных проблем языка в том, что при создании сложного приложения на Java вам придется использовать только этот язык для создания всех частей этого приложения. В Java предусмотрено не так уж много средств для межъязыкового взаимодействия.

Язык программирования C# и платформа .NET на момент создания АИК платформа .NET и программирование на C# очень развита для программирования [5].

.NET представляет собой новый способ создания распределенных, настольных и встроенных приложений..NET не имеет ничего общего с COM (кроме мощных средств интеграции двух платформ). Для типов .NET не нужны ни фабрики классов, ни регистрация в системном реестре. Эти основные элементы COM не скрыты - их просто больше нет.

Специально для новой платформы Microsoft разработала новый язык программирования - C# (Си Шарп). Этот язык, как и Java, очень многое позаимствовал из C++. Однако на C# сильно повлиял и Visual Basic 6.0.

Можно сказать, что C# впитал в себя многое из того лучшего, что есть в самых разных языках программирования, и если у вас есть опыт работы с C++, Java или Visual Basic, то вы найдете в C# много знакомого.

Очень важно отметить, что платформа .NET является полностью независимой от используемых языков программирования. Можно использовать несколько .NET-совместимых языков программирования (скорее всего, вскоре их будет множество) даже в рамках одного проекта. Разобраться с самим языком C# достаточно просто. Наибольшие усилия потребуются, чтобы познакомиться с многочисленными пространствами имен и типами библиотеки базовых классов .NET. С этими типами (как и со своими собственными, созданными, например, на C#) можно работать из любого .NET-совместимого языка [4].

Благодаря своей простоте, гибкости, скорости выполнения задач, богатой функциональности, кроссплатформенности а так же поддержке программной платформы .NET, было принято решение использования объектно-ориентированного языка программирования C# в интегрированной среде разработки visual studio 2017, с шаблоном проектирования MVC, предполагающую разделение данных приложения, пользовательского интерфейса и управляющую логики на три отдельных компонента: модель, представление и контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать консольные приложения, приложения с графическим интерфейсом, веб-сайты, веб-приложения, веб-службы для всех платформ.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов

инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения(например, клиент Team Explorer для работы с Team Foundation Server).

1.5 Обоснование выбора операционной системы

Грубо говоря, операционная система– это главный комплекс программ в компьютере. Операционная система нужна в первую очередь для того, чтобы организовывать доступ обычных программ (интернет-браузера, музыкального плеера и др.) к ресурсам компьютера [2].

Кроме того, ОС создает интерфейс, через который человек работает с программами. То есть операционная система организовывает создание картинки, которую мы видим на экране, обрабатывает движения и клики мыши, нажатие клавиш на клавиатуре, воспроизводит звук в динамиках и т.д. Сама по себе операционная система не дает возможности выполнять какие либо действия (просматривать веб-страницы, работать с текстовыми документами), для этого необходимы дополнительные прикладные программы.

Самыми используемыми ОС являются продукты Windows от компании Microsoft. Их доля составляет около 90% рынка настольных ПК. Остальные 10% приходятся на операционные системы семейства Linux и MacOS.

Ниже приведены рисунки с разных версии Windows.

Если на компьютере установлена Windows XP, то вы увидите такую надпись:

Рисунок 1.3 — Заставка Windows XP

Windows Vista выглядит так:

Рисунок 1.4 — Заставка Windows Vista

Одна из версии Windows 7 надпись выглядит следующем образом:

Рисунок 1.5 — Заставка Windows 7

А вот логотип Windows 8:

Рисунок 1.6 — Заставка Windows 8

Многие пользователи не спешат менять привычную и удобную систему Windows 8 на новую разработку того же производителя. Острой необходимости в этом по большому счету и нет. Windows 8 по-прежнему поддерживается Microsoft, а значит иметь на своем компьютере эту операционную систему.

MacOS – это операционная система от компании Apple, её можно встретить на компьютерах от этого производителя. Она начала свое распространение вместе с появлением первых продуктов от Apple и развивалась параллельно с Windows. Но надо знать, что эту систему можно установить так же на ряд других компьютеров не от этих производителей.

Linux сегодня считается наиболее «экзотическим» вариантом для настольного компьютера. Эта операционная система распространяется бесплатно (для многих именно это считается её главным преимуществом перед конкурентами). Она имеет несколько модификаций (их принято называть дистрибутивами), каждая из которых распространяется и поддерживается различными компаниями.

В качестве обычной «домашней» версии наиболее распространен дистрибутив называемый Ubuntu. За последнее время Ubuntu сделала большой шаг вперед в деле приближения Linux к рядовому пользователю, например такой дистрибутив как Linux XP является этим подтверждением, но он распространяется как платный продукт. Однако по-прежнему на неё решаются перейти или хотя бы попробовать только некоторые энтузиасты и люди, которые близко работают с компьютерами (например, программисты и системные администраторы).

Мы же хотим чтобы наша программа работала на всех операционных системах, то есть поддерживала кроссплатформенность с использованием web интерфейса

1.6 Кроссплатформенность

Кроссплатформенность - способность программного обеспечения работать более чем на одной аппаратной платформе и (или) операционной системе. Обеспечивается благодаря использованию высокоуровневых языков

программирования, сред разработки и выполнения, поддерживающих условную компиляцию, компоновку и выполнение кода для различных платформ. Типичным примером является программное обеспечение, предназначенное для работы в операционных системах Linux и Windows одновременно [7].

Понятие кроссплатформенности, понимает под собой адаптивно разработанный и реализованный дизайн проекта, который способен корректно отображаться на различных устройствах, имеющих доступ в сеть интернет.

Все пользователи интернета пользуются разными браузерами на персональных компьютерах, у кого-то могут быть новейшие версии, кто-то же пользуется старыми, также бывают случаи, когда пользователи сайтов используют офисные компьютеры, на которых установлены браузеры первых версий. Задача верстальщика, написать сайт, который будет адекватно отображаться во всех возможных версиях, которыми могут воспользоваться потенциальные посетители ресурса. По мимо персонального компьютера, для доступа в интернет используют различные гаджеты, смартфоны, планшетные, электронные книги и т.д. , и необходимо учитывать возможность использования именно этих устройств.

Основное различие между платформами, это размер монитора. Отображение браузером стандартных элементов разметки страницы тоже разные. Скорость соединения с интернетом на мобильных устройствах значительно медленнее, чем на домашних компьютерах, что вынуждает создавать более легкие версии сайтов.

Рассмотрим различия списком:

· Размер монитора

· Возможности браузера

· Скорость соединения с интернетом

Размер монитора - даже пользователи домашних/офисных компьютеров не могут похвастаться одинаковым размером монитора, что сильно бы облегчило роль создания различных интерфейсов для компьютерной продукции. Но в отличии от мобильных устройств, мониторы на домашних компьютерах имею достаточно большое разрешение, и браузеры, которыми обычно пользуются, имеют широкие возможности для реализации этого пространства. В мобильных устройствах экран довольно маленький, что влечет сужение активной области работы и выставлении соответствующих ограничений на минимальную ширину сайта, что влечет за собой возможное ухудшение дизайна проекта.

Возможности браузера - одно из самых ярких примеров, это работа js, в мобильных версиях браузеров. Сейчас существую подходы, позволяющие создавать красивые анимационные эффекты на мобильных устройствах, но

несколько лет назад, об этом можно было только мечтать. Также не у всех на мобильных телефонах могут стоять последние версии браузеров с поддержкой последних версий языков, таких как HTML5, CSS3, что требует написание верстки на уже устаревающих версиях этих языков в погоде на кроссплатформенностью.

Скорость соединения с интернетом - спорный вопрос в наше время, но если устройство не подключено к сети WiFi, то, как правило мобильные компании предоставляют не высокую скорость соединения. Можно с уверенностью сказать, что скорость мобильного интернета в десятки раз меньше скорости домашнего или офисного интернета, а все хотят, чтобы все работало мгновенно.

1.7 Выбираем платформу интерфейса

Преимущества ASP.NET MVC.

В октябре 2007 г. в Microsoft анонсировали новую платформу веб-разработки - MVC, построенную на основе ASP.NET и явно спроектированную непосредственно в ответ на развитие технологий, подобных Rails, а также в качестве реакции на критику в адрес Web Forms. В последующих разделах будет показано, каким образом эта новая платформа позволила преодолеть ограничения Web Forms и вновь вывести ASP.NET на передовой уровень.

Архитектурный шаблон MVC.

Важно различать архитектурный шаблон MVC и инфраструктуру ASP.NET MVC Framework. Шаблон MVC далеко не нов (его появление датируется 1978 г. и связано с проектом Smalltalk в Xerox PARC), но в наши дни он завоевал огромную популярность в качестве шаблона для веб-приложений по перечисленным ниже причинам:

Взаимодействие пользователя с приложением MVC осуществляется в соответствии с естественным циклом: пользователь предпринимает действие, в ответ на которое приложение изменяет свою модель данных и доставляет обновленное представление пользователю. Затем цикл повторяется. Это хорошо укладывается в схему веб-приложений, предоставляемых в виде последовательностей запросов и ответов HTTP.

Веб-приложения, нуждающиеся в комбинировании нескольких технологий (например, баз данных, HTML-разметки и исполняемого кода), обычно разделяются на ряд слоев или уровней. Полученные в результате шаблоны естественным образом вписываются в концепции MVC.

За счет принятия и адаптации шаблона MVC инфраструктура ASP.NET MVC Framework составляет сильную конкуренцию Ruby on Rails и аналогичным платформам, выводя модель MVC в авангард развития мира .NET. Обобщая опыт

и наиболее рекомендуемые приемы, обнаруженные разработчиками, которые используют другие платформы, ASP.NET MVC во многих отношениях превзошла даже то, что может предложить Rails.

Расширяемость.

Инфраструктура MVC Framework построена в виде набора независимых компонентов, удовлетворяющих интерфейсу .NET или созданы на основе абстрактного базового класса. Компоненты, подобные системе маршрутизации, механизму визуализации и фабрике контроллеров, можно заменять другими компонентами с собственной реализацией. В общем случае для каждого компонента MVC Framework предлагает три возможности:

Использование стандартной реализации компонента в том виде, как она есть (этого должно быть достаточно для большинства приложений).

Создание подкласса из стандартной реализации с целью корректировки существующего поведения.

Полная замена компонента новой реализацией интерфейса или абстрактного базового класса.

Различные компоненты, а также способы и причины их возможной настройки или замены будут рассматриваться в следующих статьях.

Жесткий контроль над HTML и HTTP.

Инфраструктура ASP.NET MVC генерирует ясный и соответствующий стандартам код разметки. Ее встроенные вспомогательные методы HTML производят соответствующий стандартам вывод, но существует также гораздо более значимое философское изменение по сравнению с Web Forms. Вместо генерации громадного объема трудно поддающейся управлению HTML-разметки инфраструктура MVC Framework стимулирует создание простых и элегантных элементов, оформленных стилями CSS.

Конечно, если действительно требуется использовать некоторые готовые виджеты для таких сложных элементов пользовательского интерфейса, как окна выбора даты или каскадные меню, применяемый в ASP.NET MVC подход "никаких специальных требований" к разметке, позволяет легко использовать наилучшие библиотеки для построения пользовательских интерфейсов, подобные jQuery или Bootstrap CSS. К примеру, библиотека jQuery настолько эффективно поддерживается, что поставляется в качестве встроенной части стандартного шаблона проекта ASP.NET MVC в Visual Studio наряду с другими популярными библиотеками, такими как Bootstrap, Knockout и Modernizr.

Сгенерированные ASP.NET MVC страницы не содержат никаких данных View State, поэтому они меньше типовых страниц ASP.NET Web Forms по размеру. Несмотря на современные быстрые соединения, такая экономия трафика

по-прежнему повышает комфорт конечного пользователя и помогает сократить затраты, связанные с запуском популярных веб-приложений.

Инфраструктура ASP.NET MVC работает в тесном сотрудничестве с HTTP. При этом имеется контроль над запросами, передаваемыми между браузером и сервером, что позволяет очень точно настраивать пользовательский интерфейс по своему усмотрению. Технология AJAX проста, и ей не нужны какие-то автоматические обратные отправки запросов для взаимодействия с состоянием клиентской стороны.

Тестируемость.

Естественное разделение различных ответственностей приложения по независимым друг от друга частям программного обеспечения, которое поддерживается архитектурой MVC, позволяет изначально строить легко сопровождаемые и тестируемые приложения. Однако проектировщики ASP.NET MVC на этом не остановились. Для каждого фрагмента компонентно-ориентированного проекта инфраструктуры они обеспечили структурированность, необходимую для удовлетворения требований модульного тестирования и средств имитации.

В среду Visual Studio добавлен набор мастеров для автоматизированного создания проектов модульного тестирования, которые могут быть интегрированы с такими инструментами модульного тестирования с открытым кодом, как NUnit и xUnit. Даже если вам никогда ранее не приходилось создавать модульные тесты, у вас будет все необходимое для успешного старта.

Далее, в этом руководстве, вы увидите примеры написания ясных и простых модульных тестов для контроллеров и действий ASP.NET MVC, предоставляющих фиктивные либо имитированные реализации компонентов инфраструктуры для эмуляции любого сценария с использованием разнообразных стратегий тестирования и имитации.

Тестируемость касается не только модульного тестирования. Приложения ASP.NET MVC успешно работают также с инструментами тестирования, встроенными в средства автоматизации пользовательского интерфейса.

Существующая платформа ASP.NET производства Microsoft, предлагающий проверенный набор компонентов и средств для разработки эффективных и высокопроизводительных веб-приложений. Главное преимущество заключается в инфраструктуре ASP.NET MVC построенной на основе платформы .NET, вы можете писать код на любом языке .NET и имеете доступ к одним и тем же функциям API-интерфейсов, определеных не только в MVC Framework, но и в обширной библиотеке классов .NET, а также во множестве библиотек .NET от независимых разработчиков.

Во-вторых, готовые средства платформы ASP.NET, такие как аутентификация, членство, роли, профили и интернационализация, могут существенно сократить объем кода, который придется писать и поддерживать в любом веб-приложении, и в проекте MVC Framework они столь же эффективны, как в классическом проекте Web Forms. Лежащая в основе платформа ASP.NET предоставляет развитый набор инструментов, на базе которых строятся веб-приложения с помощью MVC Framework.

Инфраструктура ASP.NET MVC имеет открытый код.

В отличие от предшествующих платформ веб-разработки производства Microsoft, первоначальный исходный код ASP.NET MVC доступен для свободной загрузки и для модификации и компиляции с целью получения собственной версии этой инфраструктуры. Это неоценимо при отладке кода, обращающегося к системному компоненту, когда требуется пошагово выполнить его код (и даже ознакомиться с комментариями программистов, написавших этот код). Это также полезно, если вы создаете усовершенствованный компонент и хотите видеть, какие существуют возможности разработки, или узнать, как действительно работают встроенные компоненты.

Вдобавок упомянутая возможность удобна и тогда, когда не устраивает работа того или иного компонента, когда требуется найти ошибку или когда необходимо получить доступ к тому, что недоступно с помощью других средств - интересующий компонент можно просто изменить самостоятельно. Однако при этом придется отслеживать свои изменения и повторно их применять при модернизации до более новой версии инфраструктуры.

Будем использовать платформу MVC, построенную на основе ASP.NET, так как она является одной из самых мощных и поддерживает кроссплатформенность.

1.8 Шаблон MVC

MVC — это конструкционный шаблон, который описывает способ построения структуры нашего приложения, сферы ответственности и взаимодействие каждой из частей в данной структуре.

Идея, которая лежит в основе конструкционного шаблона MVC, очень проста: нужно чётко разделять ответственность за различное функционирование в наших приложениях:

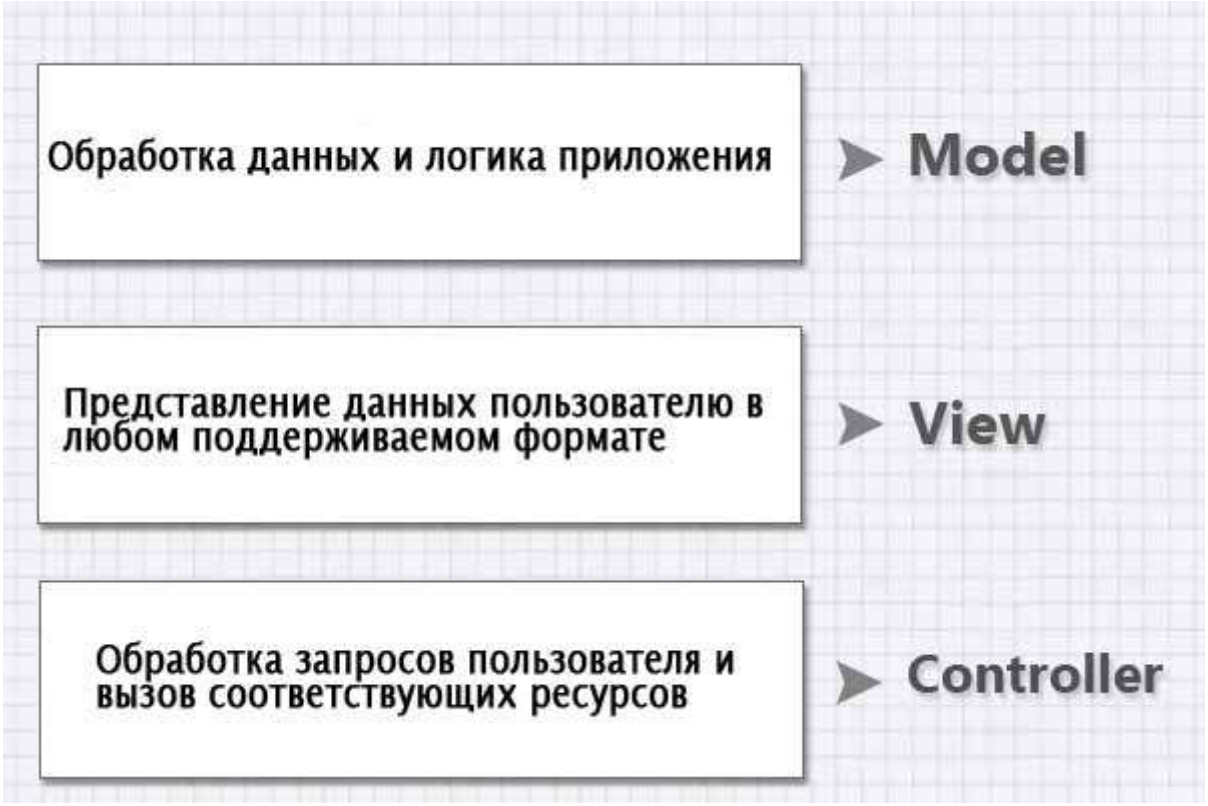

Рисунок 1.7 — Шаблон MVC

Контроллер (Controller)

Контроллер управляет запросами пользователя (получаемые в виде запросов HTTP GET или POST, когда пользователь нажимает на элементы интерфейса для выполнения различных действий). Его основная функция — вызывать и координировать действие необходимых ресурсов и объектов, нужных для выполнения действий, задаваемых пользователем. Обычно контроллер вызывает соответствующую модель для задачи и выбирает подходящий вид.

Модель (Model)

Модель - это данные и правила, которые используются для работы с данными, которые представляют концепцию управления приложением. В любом приложении вся структура моделируется как данные, которые обрабатываются определённым образом. Что такое пользователь для приложения — сообщение или книга? Только данные, которые должны быть обработаны в соответствии с правилами (дата не может указывать в будущее, e-mail должен быть в определённом формате, имя не может быть длиннее X символов, и так далее).

Модель даёт контроллеру представление данных, которые запросил пользователь (сообщение, страницу книги, фотоальбом, и тому подобное). Модель данных будет одинаковой, вне зависимости от того, как мы хотим представлять их пользователю. Поэтому мы выбираем любой доступный вид для отображения данных.

Модель содержит наиболее важную часть логики нашего приложения, логики, которая решает задачу, с которой мы имеем дело (форум, магазин, банк, и

тому подобное). Контроллер содержит в основном организационную логику для самого приложения (очень похоже на ведение домашнего хозяйства).

Вид (View)

Вид обеспечивает различные способы представления данных, которые получены из модели. Он может быть шаблоном, который заполняется данными. Может быть несколько различных видов, и контроллер выбирает, какой подходит наилучшим образом для текущей ситуации.

Веб приложение обычно состоит из набора контроллеров, моделей и видов. Контроллер может быть устроен как основной, который получает все запросы и вызывает другие контроллеры для выполнения действий в зависимости от ситуации.

Самое очевидное преимущество, которое мы получаем от использования концепции MVC — это чёткое разделение логики представления (интерфейса пользователя) и логики приложения.

Поддержка различных типов пользователей, которые используют различные типы устройств является общей проблемой наших дней. Предоставляемый интерфейс должен различаться, если запрос приходит с персонального компьютера или с мобильного телефона. Модель возвращает одинаковые данные, единственное различие заключается в том, что контроллер выбирает различные виды для вывода данных.

Помимо изолирования видов от логики приложения, концепция MVC существенно уменьшает сложность больших приложений. Код получается гораздо более структурированным, и, тем самым, облегчается поддержка, тестирование и повторное использование решений.

Когда вы используете рабочую среду, базовая структура MVC уже подготовлена, и вам остаётся только расширить структуру, размещая ваши файлы в соответствующих директориях для соответствия шаблону MVC. Кроме того, у вас будет набор функций, которые уже написаны и хорошо протестированы.

Важная черта концепции .NET состоит в том, что она является механизмом позволяющим перейти от HTML-представлений данных к представлениям, дополненным программируемой информацией на базе языка. Специфика XML состоит в том, что он полностью отделяет данные как таковые от их внешнего представления. Этот язык является главным элементом систем нового поколения, открывающим содержащую в них информацию для структурирования, редактирования и программирования ее обработок. Он позволяет доставлять данные гораздо более эффективным способом на широкий спектр цифровых устройств, позволяющих представлять целый комплекс взаимодействующих между собою служб.

В состав Microsoft .NET входят

- платформа Microsoft .NET - включает в себя инфраструктуру .NET, инструментальные средства разработки и эксплуатации служб нового поколения, пользовательские среды на базе .NET для создания информационно-насыщенных клиентских систем, стандартные блоки служб .NET, а также программы для устройств .NET
- Продукты и службы Microsoft .NET - к ним относятся ОС Windows .NET с интегрированным набором стандартных блоков служб, веб портал MSN.NET, оффисный комплект Office.NET, комплект разработчика Visual Studio .NET и веб-портал малого бизнеса bCentral для .NET
- Службы .NET сторонних производителей - корпоративные и вертикальные службы

Microsoft .NET включает комплексы приложений, служб и устройств в работу по созданию индивидуализированной цифровой среды. Это означает, что появляется радикально новое поколение программных средств, способных функционировать как единая интегрированная служба.

HTTP запрос, или сообщение состоит из трех частей: строки запроса, заголовков, и тела HTTP сообщения.

Строка запроса, или стартовая строка: в запросе к серверу — строка, которая содержит тип запроса (метод), URI запрашиваемой страницы, и версия HTTP протокола (например HTTP/1.1). В ответе от сервера эта строка содержит версию HTTP протокола, и код ответа. Код ответа представляет собой целое число из трех цифр. За ним обычно следует отделённая пробелом поясняющая фраза, поясняющая код, например: 200 OK, или 404 Not Found.

Методы (типы) HTTP запроса: GET, POST, PUT, PATCH, HEAD, DELETE, TRACE. Чаще всего в HTTP запросе используются методы GET, или POST: GET — используется для запроса содержимого web-страницы по указанному URI. URI — это адрес страницы без указания домена, например: /internet/chto-takoe-http-zapros-soobshhenie/ вместо webistore.ru/internet/chto-takoe-http-zapros-soobshhenie/. POST — используется для отправки информации, на сервер. При отправке данных методом POST, они указываются в теле HTTP сообщения, а не в строке ввода адреса в браузере, как это делается при передаче данных методом GET. Например, при отправке комментария через форму, которая расположена под статьей, информация на сервер передается именно методом POST. Так же с помощью метода POST загружают файлы на сервер.

HTTP заголовки — это часть запроса, в котором содержатся различные параметры, которые используются для правильного построения web-страницы.

Тело HTTP сообщения — содержит полученные от сервера данные, например сформированную web-страницу в виде HTML кода, либо ресурс, например картинку.

1.9 Концепция смежных технологий AJAX

AJAX — подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате, при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее.

В классической модели веб-приложения:

- Пользователь заходит на веб-страницу и нажимает на какой-нибудь её элемент.
- Браузер формирует и отправляет запрос серверу.
- В ответ сервер генерирует совершенно новую веб-страницу и отправляет её браузеру и т. д., после чего браузер полностью перезагружает всю страницу.

При использовании AJAX:

- Пользователь заходит на веб-страницу и нажимает на какой-нибудь её элемент.
- Скрипт (на языке JavaScript) определяет, какая информация необходима для обновления страницы.
- Браузер отправляет соответствующий запрос на сервер.
- Сервер возвращает только ту часть документа, на которую пришёл запрос.
- Скрипт вносит изменения с учётом полученной информации (без полной перезагрузки страницы).

Экономия трафика

Использование AJAX позволяет значительно сократить трафик при работе с веб-приложением благодаря тому, что вместо загрузки всей страницы достаточно загрузить только изменившуюся часть или вообще только получить/передать набор данных в формате JSON или XML, а затем изменить содержимое страницы с помощью JavaScript.

Уменьшение нагрузки на сервер.

При правильной реализации AJAX позволяет снизить нагрузку на сервер в несколько раз.

В частности, все страницы сайта чаще всего генерируются по одному шаблону, включая неизменные элементы («шапка», «навигационная панель»,

«подвал» и т. д.), для генерации которых требуются обращения к разным файлам, время на обработку скриптов (а иногда и запросы к БД) — всё это можно опустить, если заменить полную загрузку страницы генерацией и передачей лишь содержательной части. Дизайн страницы также обычно содержит множество файлов, связанных с оформлением (картинки, стили), на повторную обработку которых не надо тратить время, используя AJAX (экономия на количестве HTTP-соединений значительно выгоднее, чем на сокращении трафика каждого из них).

Ускорение реакции интерфейса.

Поскольку загрузка изменившейся части значительно быстрее, то пользователь видит результат своих действий быстрее и без мерцания страницы (возникающего при полной перезагрузке).

Почти безграничные возможности для интерактивной обработки

Например, при вводе поискового запроса в Google выводится подсказка с возможными вариантами запроса. На многих сайтах при регистрации пользователь вводит имя, и сразу же видит, доступно это имя или нет. AJAX удобен для программирования чатов, административных панелей и других инструментов, которые выводят меняющиеся со временем данные.

Движок представлений Razor

Представление в ASP.NET MVC содержит не только стандартный код html, но и также вставки кода на языке C#. Для обработки кода, содержащего как элементы html, так и конструкции C#, используется движок представлений.

В действительности при вызове метода View контроллер не производит рендеринг представления и не генерирует разметку html. Контроллер только готовит данные и выбирает, какое представление надо возвратить в качестве объекта ViewResult. Затем уже объект ViewResult обращается к движку представления для рендеринга представления в выходной ответ.

По умолчанию в ASP.NET MVC Core используется один движок представлений - Razor. Хотя при желании мы можем также использовать какие-то другие сторонние движки или создать свой движок представлений самостоятельно.

Цель движка представлений Razor является определение переход от разметки html к коду C#.

Синтаксис Razor довольно прост - все его конструкции предваряются символом @, после которого происходит перед к коду C#.

Все конструкции Razor можно условно разделить на два виде: однострочные выражения и блоки кода.

Пример применения однострочных выражений:

Дата: @DateTime.Now.ToLongDateString()

В данном случае используется объект DateTime и его метод ToLongDateString()

Или еще один пример:

```
<p>@(20 + 30)</p>
```

Так как перед скобками стоит знак @, то выражение в скобках будет интерпретироваться как выражение на языке C#. Поэтому браузер выведет число 50, а не "20 + 30".

Но если вдруг мы создаем код html, в котором присутствует символ @ не как часть синтаксиса Razor, а сам по себе, то, чтобы его отобразить, нам надо его дублировать:

```
<p>@@DateTime.Now =@DateTime.Now.ToLongDateString()</p>
```

Блоки кода могут иметь несколько выражений. Блок кода заключается в фигурные скобки, а каждое выражение завершается точкой запятой аналогично блокам кода и выражениям на C#:

В блоках кода мы можем определить обычные переменные и потом их использовать в представлении.

Весь код в пределах блока расценивается как код c#. Однако с помощью конструкции @: мы можем в блоке кода выводить на веб-страницу текст:

```
@{
    string head = "Hello world";
    @: <b>Привет мир!</b>
    head = head + "!!";
}
<p>@head</p>
```

Если необходимо вывести значение переменной без каких-либо html-элементов, то мы можем использовать специальный снипет <text>:

```
@{
    int i = 8;
    <text>@i</text>
}
    <text>@(i+1)</text>
```

В Razor могут использоваться комментарии. Они располагаются между символами @**@:

2 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА

2.1 Эффективность алгоритма

Алгоритм представляет собой последовательность шагов, которая призвана решить определенную задачу. Иными словами алгоритм - это способ решения этой задачи. В этом качестве алгоритм применяется для обозначения метода решения любых, в том числе и повседневных задач [8].

Алгоритм может иметь входные данные, над которыми производятся вычисления, а также может иметь выходной результат - одно значение или набор значений. По сути задача алгоритма состоит в преобразовании входных значений в выходные.

Важным критерием алгоритма выступает эффективность. Алгоритм может прекрасно решать поставленную задачу, но при этом быть не эффективным. Как правило, под эффективностью алгоритма подразумевается время, за которое должен выполняться данный алгоритм.

Общее время выполнения программы зависит от двух факторов:

времени выполнения каждого оператора

частоты выполнения каждого оператора

Время выполнение каждого оператора зависит от среды выполнения, операционной системы и других системных характеристик.

В зависимости от эффективности существует много типов алгоритмов, среди которых можно выделить следующие типы алгоритмов (перечислены в порядке уменьшения эффективности):

Константный (1)

Приложение выполняет фиксированное количество операций, которые, как правило, требуют постоянного времени.

Примером может служить следующий набор операций:

```
int x = 10;
if(x > 0)
{
    x--;
}
else
{
    x++;
}
```

Рисунок 2.1— Константный алгоритм

Логарифмический (logN)

Выполняется медленнее, чем программы с постоянным временем. Примером подобного алгоритма может служить алгоритм бинарного поиска.

```
public static int Rank(int key, int[] numbers)
{
    int low = 0;
    int high = numbers.Length - 1;
    while (low <= high)
    {
        // находим середину
        int mid = low + (high - low) / 2;
        // если ключ поиска меньше значения в середине
        // то верхней границей будет элемент до середины
        if (key < numbers[mid]) high = mid - 1;
        else if (key > numbers[mid]) low = mid + 1;
        else return mid;
    }
    return -1;
}
```

Рисунок 2.2 — Логарифмический алгоритм

К примеру, если у нас в массиве numbers 8 элементов, то чтобы найти нужный нам элемент, нам надо последовательно делить количество элементов на 2. То есть для поиска нужного элемента нам надо 3 выполнить цикл while. И данный результат, как раз описывается логарифмической функцией: log28 = 3;

Рост времени выполнения при росте N будет увеличиваться на некоторую постоянную величину [10].

Линейный (N)

Как правило, встречается там, где метод основан на цикле. К примеру, функция факториала:

```
private static int Factorial(int n)
{
    int result = 1;
    for(int i=1; i<=n; i++)
    {
        result *= i;
    }
    return result;
}
```

Рисунок 2.3 — Линейный алгоритм

Квадратичный (N2)

Как правило, методы, которые соответствуют данному алгоритму, содержит два цикл - внешний и вложенный, которые выполняются для всех значений вплоть до N. В качестве примера можно привести программу сортировки пузырьком (bubble sort) массива из N элементов, в которой в худшем случае нам надо совершить обход N*N элементов с помощью двух циклов:

```
private static void BubbleSort(int[] nums)
{
    int temp;
    for (int i = 0; i < nums.Length - 1; i++)
    {
        for (int j = i + 1; j < nums.Length; j++)
        {
            if (nums[i] > nums[j])
            {
                temp = nums[i];
                nums[i] = nums[j];
                nums[j] = temp;
            }
        }
    }
}
```

Рисунок 2.4 — Квадратичный алгоритм

Кубический (N3)

В программах, которые соответствуют этому алгоритму, используются три цикла, например:

```
char[] chars = new char[] { 'A', 'B', 'C' };
for(int i=0; i<chars.Length; i++)
{
    for(int j=0; j<chars.Length;j++)
    {
        for(int k=0; k<chars.Length;k++)
        {
            Console.WriteLine($"{chars[i]}{chars[j]}{chars[k]}");
        }
    }
}
```

Рисунок 2.5 — Кубический алгоритм

Здесь рассмотрены только некоторые виды сложностей алгоритмов, которых на самом деле гораздо больше. Графически их можно представить следующим образом:

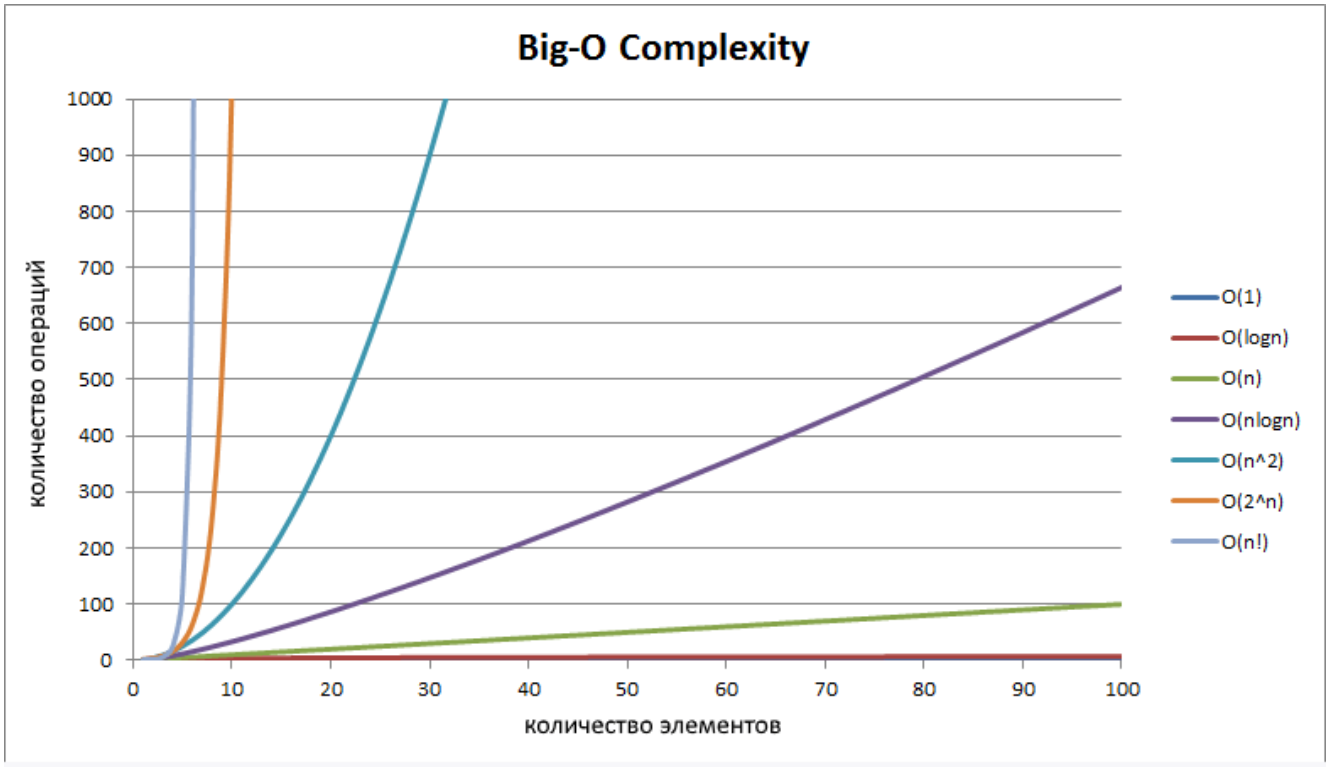

Рисунок 2.4 — Графическое изображение алгоритмов

В то же время надо сделать несколько замечаний. В данном случае дана идеальная модель сложности алгоритма. Но в первую очередь надо отметить, что подразумевается, что воздействие окружения (операционной системы) на выполнение программы ничтожно мало. Хотя в реальности, естественно, окружение может привносить свой вклад в конечную производительность приложения [9].

Модель надежности программного обеспечения по модели Шумана

Модель Шумана основана на следующих допущениях:

- общее число команд в программе на машинном языке постоянно;
- в начале компоновочных испытаний число ошибок равно некоторой постоянной величине, и по мере исправления ошибок их становится меньше. В ходе испытаний программы новые ошибки не вносятся;
- ошибки изначально различимы, по суммарному числу исправленных ошибок можно судить об оставшихся;
- интенсивность отказов программы пропорциональна числу остаточных ошибок.

Предполагается, что до начала тестирования (т.е. в момент $\tau=0$) имеется M ошибок. В течение времени тестирования τ обнаруживается $\varepsilon 1(\tau)$ ошибок в расчете на одну команду в машинном языке.

Тогда удельное число ошибок на одну машинную команду, оставшихся в системе после времени тестирования τ, равно:

$$\varepsilon_2(\tau) = \frac{M}{I} - \varepsilon_1(\tau)$$

(1)

где I - общее число машинных команд, которое предполагается постоянным в рамках этапа тестирования.

Предполагается, что значение функции количества ошибок Z(t) пропорционально числу ошибок, оставшихся в программе после израсходованного на тестирование времени τ.

Z (t) = C * ε2 (τ),

где C - некоторая постоянная, t - время работы программы без отказов.

Тогда, если время работы программы без отказа t отсчитывается от точки t = 0, а τ остается фиксированным, функция надежности, или вероятность безотказной работы на интервале от 0 до t, равна

$$t_{cp} = \frac{1}{C \cdot \left(\frac{M}{I} - \varepsilon_1(\tau) \right)}$$

(2)

Нам необходимо найти начальное значение ошибок M и коэффициент пропорциональности C. Эти неизвестные оцениваются путем пропуска функционального теста в двух точках переменной оси отладки τа и τв, выбранных так, что ε1(τа)<ε1(τd).

В процессе тестирования собирается информация о времени и количестве ошибок на каждом прогоне, т.е. общее время тестирования τ складывается из времени каждого прогона:

τ = τ1 + τ2 + τ3 + … + τn.

Предполагая, что интенсивность появления ошибок постоянна и равна λ, можно вычислить ее как число ошибок в единицу времени,

$$\lambda = \frac{\sum_{i=1}^{n} A_i}{\tau}$$

(3)

где Ai - количество ошибок на i - ом прогоне.

Тогда

$$t_{cp} = \frac{\tau}{\sum_{i=1}^{n} A_i}$$

(4)

Имея данные для двух различных моментов тестирования τа и τв, можно сопоставить уравнения (3) при τа и τb:

$$\frac{1}{\lambda_a} = \frac{1}{C \cdot \left(\frac{M}{I} - \varepsilon_1(\tau_a) \right)} \quad (5)$$

$$\frac{1}{\lambda_b} = \frac{1}{C \cdot \left(\frac{M}{I} - \varepsilon_1(\tau_b) \right)} \quad (6)$$

Из соотношений (5) и (6) найдем неизвестный параметр С и М:

$$M^* = I \cdot \frac{\frac{\lambda_b}{\lambda_a} \cdot \varepsilon_1(\tau_a) - \varepsilon_1(\tau_b)}{\frac{\lambda_b}{\lambda_a} - 1} \quad (7)$$

$$C^* = \frac{\lambda_{\tau_a}}{\frac{M^*}{I} - \varepsilon_1(\tau_a)} \quad (8)$$

Наш исходный код содержит 2 000 командных строк, из них, до начала эксплуатации (после периода отладки), 15 командных строк содержат ошибки. После 20 дней работы обнаружена 1 ошибка. Найдём среднее время безошибочной работы программы и интенсивность отказов программы при коэффициенте пропорциональности, равном 0,7.

E1(t)= 0,0005

E2(t)= 0,007

P(t)= 0,906649

tcp= 204,0816

λ= 0,0049 - интенсивность отказов

2.2 Резервное копирование

Самое ценное в любой компьютерной системе — это хранящаяся информация, потеря которой может обернуться серьезными неприятностями и даже финансовым крахом как для компании в целом, так и для конкретных пользователей. Восстановить информацию удается далеко не всегда, да и придётся потратить немало денег, сил и времени.

К сожалению, даже самое надежное оборудование, и лицензионное программное обеспечение не гарантирует полной сохранности данных. Причин потерять важные данные великое множество — здесь и банальный выход из строя винчестера, и сбой файловой системы, и повреждение файлов вирусами и нежелательное изменение или удаление данных самими пользователями и т.п. Потерять информацию даже можно и в результате возникновения пожара, кражи компьютера. Иными словами, как ни печально это сознавать, но вероятность потери информации была, есть и будет всегда. Один из способов предотвратить неприятности — производить резервирование данных. Последнее актуально не только на предприятия, где резервное копирование/восстановление данных выполняет один или несколько серверов, контролируемых высококвалифицированными администраторами, но и дома. Здесь не по карману дорогостоящие программно-аппаратные решения, которые требуют знаний, зато уместны недорогие, простые, легко настраиваемые программы, которые позволили бы восстанавливать информацию в экстренных ситуациях.

О вариантах резервирования данных.

Резервное копирование данных подразумевает регулярное создание копий нужной информации, которые обычно хранятся на каких-либо носителях, периодически перезаписываются и позволяют оперативно восстановить данные в случае аварийной ситуации. В качестве носителей могут использоваться локальные и сетевые жесткие диски, магнитооптические диски и т.п. Иногда для повышения надежности производится дублирование резервных копий на FTP. Резервироваться могут как отдельные папки и файлы, так и разделы диска и даже диск целиком — с применением разного ПО.

Резервирование диска, подразумевающее создание снимка системы (snapshot) или образа диска, выполняется путем резервного копирования данных по секторам. При этом в создаваемый архив включается вся операционная система, системный реестр Windows, драйверы устройств, установленные приложения вместе с пользовательскими настройками, а также служебные области диска, скрытые от пользователя. Образ диска включает образы всех разделов диска, а также образ начальной области диска (нулевой дорожки), в том числе главную загрузочную запись (Master Boot Record). Возможно создание

образа не всего диска, а отдельных его разделов. В таком случае в создаваемый архив включаются все файлы и папки разделов, независимо от их атрибутов (в том числе скрытые и системные файлы), таблица размещения файлов (FAT) и корневой каталог.

Отдать предпочтение одной из двух технологий резервирования данных невозможно — у каждой есть как плюсы, так и минусы. Резервирование диска позволяет в случае сбоя файловой системы или выхода из строя жесткого диска полностью восстановить диск, включая ОС и установленные программы. Восстановленные данные будут идентичны созданному образу диска, а не состоянию операционной системы и файлов на момент краха. Однако систему после краха не всегда стоит восстанавливать на основе созданного ранее полного образа диска, так как возможно, что причиной сбоя послужили многочисленные ошибки в системном реестре (неверные ключи, ссылки на несуществующие файлы, отсутствующие драйверы устройств и пр.). Вполне возможно, что в рухнувшей системе имелись вирусы и шпионские компоненты. При восстановлении диска из его образа подобные проблемы никуда не исчезнут и нормальная работа будет по-прежнему невозможна — поэтому необходимы полная переустановка системы и восстановление только важных папок и файлов. Именно для таких ситуаций и незаменимо резервное копирование папок и файлов. Кроме того, данный вид резервирования требует гораздо меньше времени и дискового пространства, в то время как для создания резервной копии только одного раздела жесткого диска объемом 50 Гбайт потребуется дополнительно примерно столько же места, которого на жестком диске может и не быть.

Полное, инкрементное и дифференциальное резервное копирование.

Как в отношении резервирования дисков/разделов, так и при резервировании папок/файлов в зависимости от приложения могут предоставляться разные варианты проведения резервного копирования: полное, инкрементное и дифференциальное.

Полное резервное копирование данных обеспечивает включение в архив всех архивируемых данных по состоянию на момент его создания. Поэтому полный архив оказывается значительным по объему, а его создание требует гораздо больше времени, чем другие варианты резервирования, зато восстановить данные из полного архива можно значительно быстрее.

Инкрементный архив содержит только данные, изменившиеся с момента проведения последнего полного или инкрементного резервирования. Поэтому такой архив обычно имеет гораздо меньший размер и создается быстрее. Но так как он содержит не все архивируемые данные, то для восстановления информации необходимо иметь все предыдущие инкрементные архивы и

изначально созданный полный архив. В сравнении с инкрементным резервным копированием, приводящему к созданию целой цепочки архивов, при дифференциальном копировании создается один независимый файл, в котором содержатся все изменения данных, имевшие место после формирования исходного полного архива. Как правило, восстановить информацию из дифференциального архива можно быстрее, чем из инкрементного, так как не требуется последовательно обрабатывать предыдущие архивы. Однако если цепочка инкрементных архивов невелика, возможна и обратная ситуация.

Полное резервирование, с одной стороны, может использоваться как самостоятельный способ резервирования, а с другой — оно является основой последующего инкрементного или дифференциального резервирования. Учитывая, что настройки системы и необходимый набор приложений меняются не часто, а значит постоянно пересоздавать полный исходный образ не потребуется, вполне оправданно один раз затратить немалое время на его создание, чтобы в дальнейшем получить долгосрочную возможность быстрого восстановления системы в исходное состояние. Однако чаще всего полное резервирование используется в комбинации с инкрементным или дифференциальным резервированием. Целесообразно создавать полный архив раз в месяц, а инкрементный или дифференциальный — каждый день. Такой подход позволяет в любой момент восстановить нужные данные с минимальными затратами времени и дискового пространства.

После дефрагментации нужно заново создать образ диска, так как во время дефрагментации изменяется положение файлов на диске.

Программы для резервирования данных

Сегодня, общее количество программ для различных способов резервирования информации исчисляется десятками. В одних программах реализована только возможность резервного копирования папок и файлов, другие ориентированы на создание образов дисков, третьи предоставляют обе возможности. В разном объеме в них реализована поддержка инкрементного и дифференциального резервирования. множество носителей, которые могут использоваться для хранения резервных копий

Выше изложены плюсы различных технологий резервирования, выбор делает для себя каждый сам, исходя из желаний и предпочтений и возможностей, например от свободного места на жёстком диске или времени, которое пользователь может уделить резервированию. Эта, безусловно, важная технология способна выручить вас, особенно тогда, когда на жёстком диске имеется важная информация. Я бы рекомендовал использовать облачное хранение

данных, так как ваша информация хранится буквально в интернете и вы можете скачать её в любой точке мира даже с телефона или планшета.

2.3 Удалённый доступ

В наше время, вместе с доступностью планшетов и мощных сотовых телефонов, есть возможность управлять компьютером дистанционно. Это удобно пользователям стационарного компьютера, так как с помощью планшетов и телефонов можно управлять компьютером находясь далеко от него. Кроме того, могут возникнуть ситуации, когда потребуется зайти на домашний компьютер, сидя в офисе на работе. Исправить положение достаточно просто, нужно лишь настроить удаленный доступ к компьютеру.

Такая возможность предусмотрена в самой операционной системе Windows. Достаточно просто зайти в «Панель управления», нажать на вкладку «Система», выйти в меню «Удаленный доступ». Чтобы точно понять, как настроить удаленный доступ к компьютеру через интернет, достаточно выбрать опцию, которая разрешает помощнику на расстоянии подключаться к вашему ПК. Все, теперь система не будет сопротивляться, если в нее захочет зайти другой пользователь. Так же есть множество специализированных программ для удобного удалённого управления своим домашним компьютером, такие как: TeamViewer или Ammyy Admin, которые являются самыми предпочтительными для дома и на многих предприятиях, так как вторая, является абсолютно бесплатной.

Пользователи операционной системы Windows 7 должны выбрать вариант «подключиться к удаленному столу с проверкой подлинности». Седьмая операционка может поддерживать управление рабочим столом и подключением помощника на расстоянии. Удаленный доступ к компьютеру может быть двух видов: эта система позволит управлять мышью и клавиатурой, все действия будут отображаться на двух ПК. В дистанционном варианте рабочий стол видит только управляющий, у другого пользователя появится заблокированный экран.

2.4 Требование к аппаратной части

Программа разрабатывалась на ПК со стандартной комплектацией, среднего уровня. Так как работа происходит с небольшой БД, системе не требуется высокая производительность, достаточно будет ПК уровня офисной конфигурации.

3 РЕАЛИЗАЦИЯ РАБОТЫ

Структура проекта веб-приложения MVC

При создании проекта веб-приложения ASP.NET MVC компоненты MVC разделяются по папкам проекта, которые указаны на следующем рисунке.

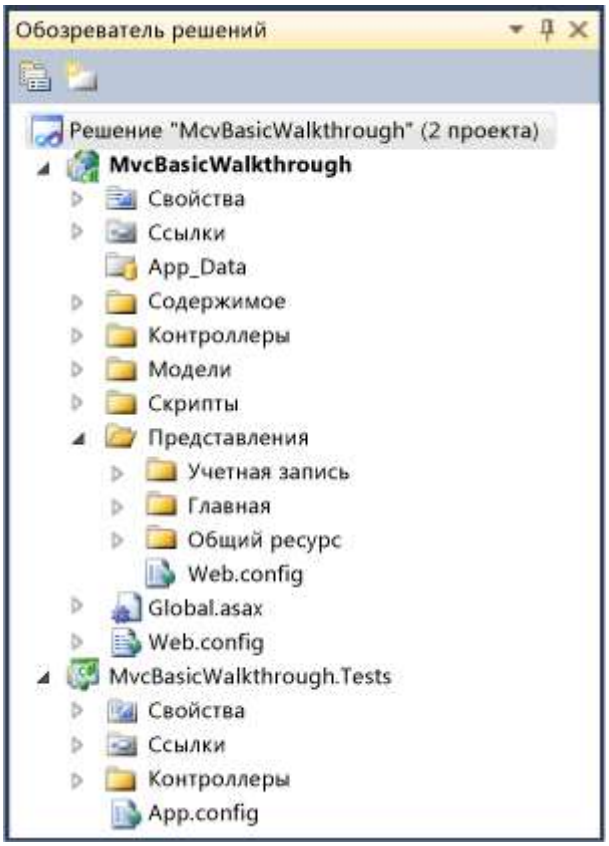

Рисунок 3.1 — Структура MVC

По умолчанию проекты MVC содержат следующие папки.

App_Data — физическое хранилище данных. Эта папка выполняет те же функции, что и для веб-сайтов ASP.NET, которые используют страницы веб-форм.

Content — рекомендуемое расположение для добавления файлов содержимого (например, файлов каскадных таблиц стилей, изображений и пр.). В общем случае папка Content предназначена для статических файлов.

Controllers — рекомендуемое расположение для контроллеров. Имена контроллеров в платформе MVC должны оканчиваться на «Controller», например HomeController, LoginController или ProductController.

Models — предназначена для классов, которые представляют модель приложения для веб-приложения MVC. Эта папка обычно содержит код, который определяет объекты и логику взаимодействия с хранилищем данных. Сами объекты модели обычно располагаются в отдельных библиотеках классов. Тем не менее при создании нового приложения классы можно расположить в этой папке, чтобы переместить их в отдельные библиотеки на более поздней стадии разработки.

Scripts — рекомендуемое расположение для файлов скриптов, поддерживающих приложение. Эта папка по умолчанию содержит файлы платформы ASP.NET Ajax и библиотеку jQuery.

Views — рекомендуемое расположение для представлений. Представления используют файлы ViewPage (ASPX), ViewUserControl (ASCX) и ViewMasterPage (MASTER) в дополнение к остальным файлам, которые связаны с отображением представлений. Папка Views содержит папки для всех контроллеров. Название папки состоит из префикса имени контроллера. Например, если существует контроллер с именем HomeController, то в папке Views будет вложенная папка с именем Home. При загрузке представления платформой ASP.NET MVC в папке «Views\имя_контроллера» по умолчанию выполняется поиск файла ViewPage (ASPX), который имеет имя требуемого представления. По умолчанию в папке Views находится папка Shared, которая не соответствует ни одному контроллеру. Папка Shared используется для представлений, которые используются на нескольких контроллерах.Например, главную страницу веб-приложения можно поместить в папку Shared.

В дополнение к перечисленным выше папкам веб-приложение MVC использует код в файле Global.asax для установки глобальных параметров маршрутизации URL-адресов по умолчанию, а также использует файл Web.config для настройки приложения.

3.1 Интерфейс программы

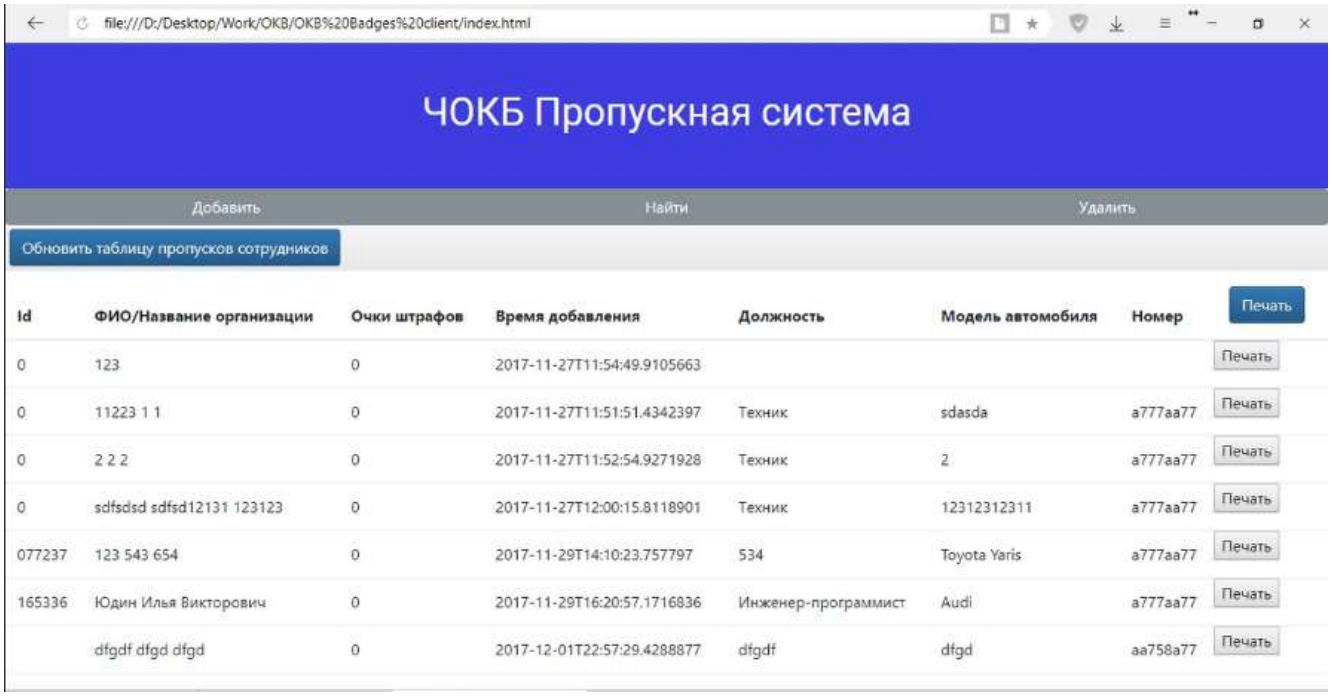

Рисунок 3.2 — Главное окно программы

```
function PrintPass(tuple) {
    var printWindow = window.open('common/print', '', 'left=50,top=50,width=800,height=640,toolbar=0,scrollbars=1,status=0');
    printWindow.onload = function () {
        var body = printWindow.document.getElementById("print-content");

        for (var i = 0; i < tuple.length; i++) {
            switch (CheckSubject(tuple[i])) {
                case 1:
                    var shE = printWindow.document.querySelector('#shadowPassEmployee');
                    var empl = shE.content.getElementById("passEmployee");
                    var e_copy = empl.cloneNode(true);

                    var post = tuple[i].item1.post;
                    var model = tuple[i].item2.model;
                    var fio = tuple[i].item1.surname + ' '
                        + tuple[i].item1.firstname + ' '
                        + tuple[i].item1.patronymic;

                    e_copy.childNodes[1][0].value = tuple[i].item1.badge.number;
                    e_copy.childNodes[1][1].value = EditFIO(fio);
                    e_copy.childNodes[1][2].value = EditPost(post);
                    e_copy.childNodes[1][3].value = EditModel(model);
                    e_copy.childNodes[1][4].value = tuple[i].item2.number;
                    body.appendChild(e_copy);

                    break;
                case 2:
                    break;
            }
        }
        printWindow.print();
```

Рисунок 3.3 — Функция заполнения формы для печати пропусков

Рисунок 3.4 — Кнопка добавления сотрудника

Рисунок 3.5 — Кнопка поиска сотрудника (по имени, имени/фамилии, по Ф.И.О, по номеру машины, по марке машины)

Рисунок 3.6 — Кнопка удаления сотрудника

Рисунок 3.7 — Кнопка печати шаблона.

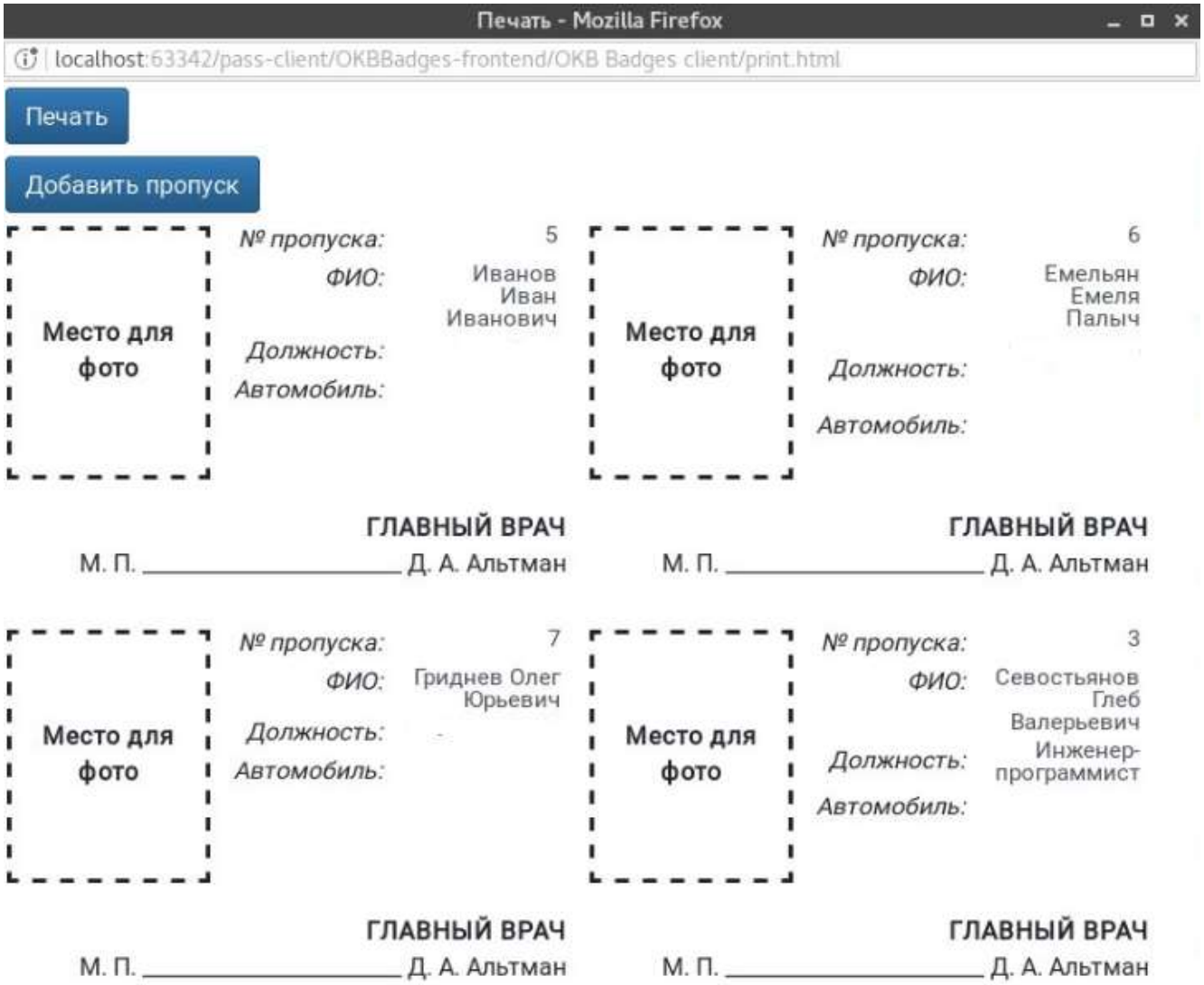

Рисунок 3.8 — Шаблон программы

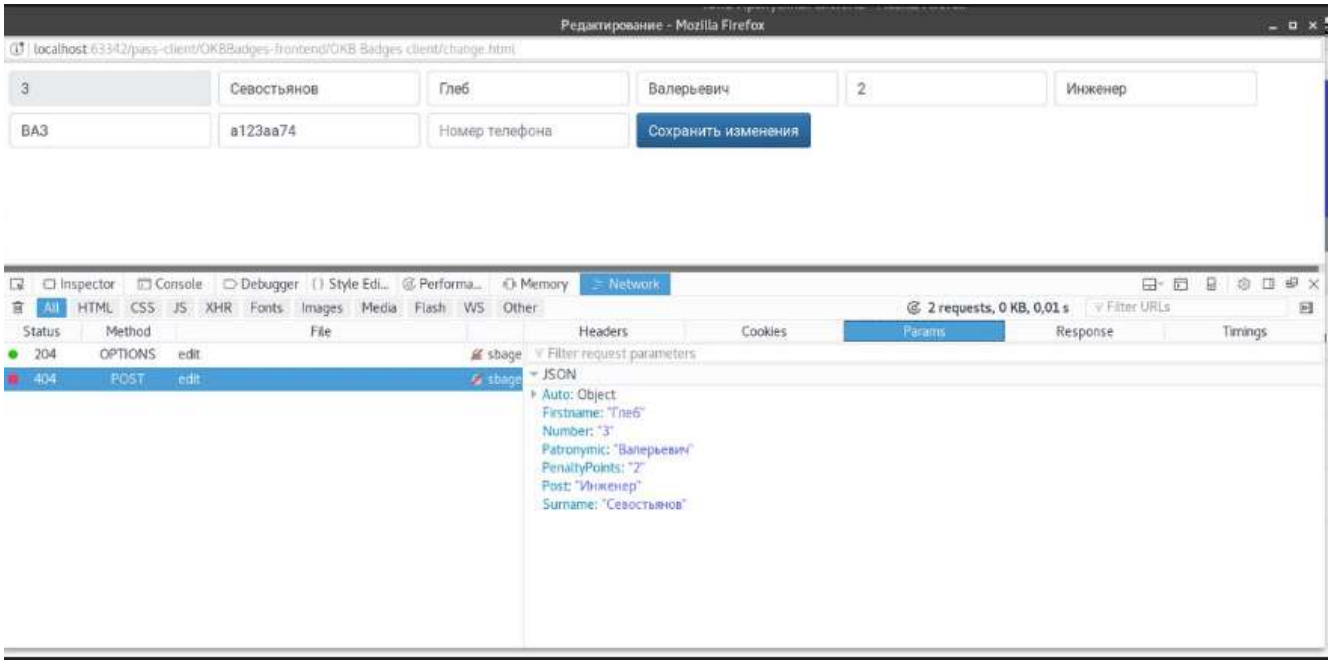

Рисунок 3.9 — Редактирование сотрудника

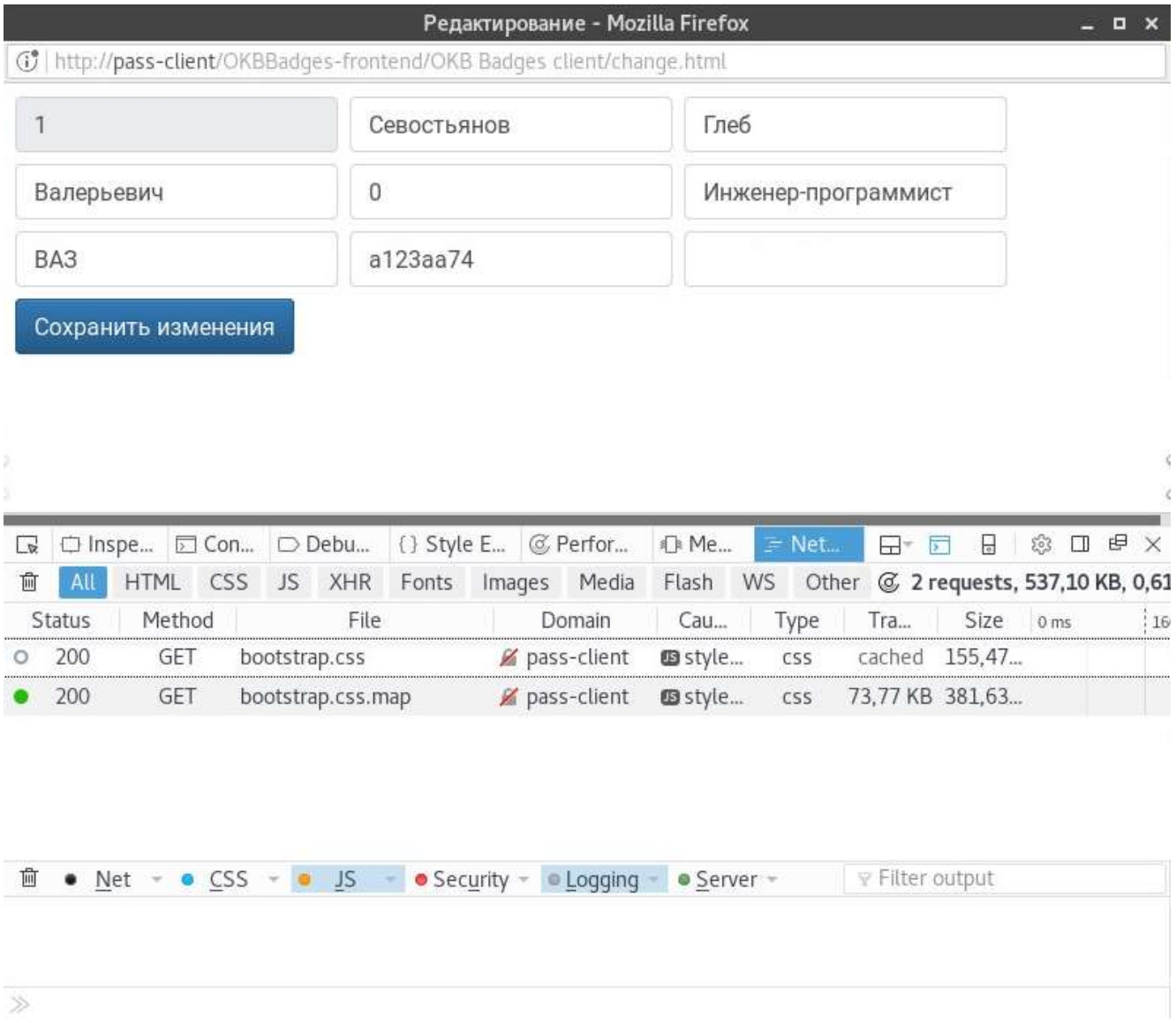

Рисунок 3.10 — Редактирование сотрудника с другим разрешением экрана

ЗАКЛЮЧЕНИЕ

Целью работы являлась разработка системы автоматической выдачи пропусков. Для достижения цели решены следующие задачи:

– изучены технологии программирования на языках высокого уровня;

– произведен обзор существующих решений;

– выбрана мультиплатформа для реализации;

– выбрана среда разработка Visual Studio;

– определены требования к операционной системе;

– спроектирована система выдачи пропусков;

– разработаны алгоритмы программы;

– разработан интерфейс.

Все поставленные задачи успешно выполнены, а цель работы достигнута. Программа получила простой интерфейс. В дальнейшем планируется развитие системы в сторону добавление новых алгоритмов.

Программой планируется пользоваться и вести отсчёт из разных кабинетов, поэтому при использовании аналогов возникает необходимость устанавливать программу на 8 различных персональных компьютеров. Лицензионная версия аналогичного программного обеспечения стоит 8000 руб. для одного места, оснащённого персональным компьютером. Следовательно, для реализации системы на аналогичных решениях потребуется 64000 руб. потратить на приобретение лицензионной продукции, которая не обеспечит требования, перечисленные в техническом задании.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Липпман, С. Язык программирования C++: Вводный курс / С. Липпман. - М.: Вильямс И.Д., 2007. - 896 с.

2 Кундиус, В.А. Теоретические основы разработки и реализации языков программирования / В.А. Кундиус. - М.: КноРус, 2013. - 184 с.

3 Новичков, В.С. Начала программирования на языке QBASIC: Учебное пособие / В.С. Новичков, А.Н. Пылькин. - М.: ГЛТ, 2007. - 268 с.

4 Орлов, С. Теория и практика языков программирования: Учебник для вузов. Стандарт 3-го поколения / С. Орлов. - СПб.: Питер, 2013. - 688 с.

5 Троелсен, Э. Язык программирования C# 5.0 и платформа .NET 4.5 / Э. Троелсен. - М.: Вильямс И.Д., 2013. - 1312 с.

6 Гергель, В.П. Современные языки и технологии параллельного программирования: Учебник/ предисл.: В.А. Садовничий. / В.П. Гергель. - М.: МГУ, 2012. - 408 с.

7 Гавриков, М.М. Теоретические основы разработки и реализации языков программирования: Учебное пособие / М.М. Гавриков, А.Н. Иванченко, Д.В. Гринченков. - М.: КноРус, 2010. - 184 с.

8 Кнут, Д.Э. Искусство программирования. В 3 т. Т 1. Основные алгоритмы / Д.Э. Кнут; пер. с англ. — 3-е изд. — М.: Издательский дом "Вильяме", 2000. — 720 с.

9 Вирт, Н. Алгоритмы и структуры данных/Н. Вирт; пер. с англ. — М.: Мир, 1989. —360 с.

10 Давыдов, В.Г. Программирование и основы алгоритмизации: учебное пособие / В.Г. Давыдов. — М.: Высш. шк., 2003. — 447 с.

ПРИЛОЖЕНИЕ А. Исходный код программы

В папке Models следующие файлы:

Automobile.cs – класс авто, перечислены марка авто, проверка на корректность номера.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

namespace BadgeSystem.Models
{
    public class Automobile
    {
        public int Id { get; set; }
        public string Model { get; set; }
        public string Number { get; set; }

        public Automobile()
        {
            Model = string.Empty;
            Number = string.Empty;
        }

        public override bool Equals(object obj)
        {
            try
            {
                Automobile automobie = (Automobile)obj;
                if (Model.Equals(automobie.Model,
StringComparison.CurrentCultureIgnoreCase) &&
                    Number.Equals(automobie.Number,
StringComparison.CurrentCultureIgnoreCase))
                    return true;
                else return false;
            }
            catch { return false; }
        }

        public override int GetHashCode()
        {
            return Id.GetHashCode() ^ Model.GetHashCode() ^
Number.GetHashCode();
        }
```

```
        public override string ToString()
        {
            return $"{Id} {Model} {Number}";
        }

        private static bool Validate(string number)
        {
            string pattern =
@"[а|в|е|к|м|н|о|р|с|т|у|х]\d{3}[а|в|е|к|м|н|о|р|с|т|у|х]{2}\d{2,3}"
;
                Regex regex = new Regex(pattern,
  RegexOptions.IgnoreCase);
                Match match = regex.Match(number);
                if (regex.IsMatch(number))
                    return true;
                else return false;
            }

            private static bool ValidateTransit(string number)
            {
                string pattern =
  @"[а|в|е|к|м|н|о|р|с|т|у|х]{2}\d{3}[а|в|е|к|м|н|о|р|с|т|у|х]\d{2,3
  }";
                Regex regex = new Regex(pattern,
  RegexOptions.IgnoreCase);
                Match match = regex.Match(number);
                if (regex.IsMatch(number))
                    return true;
                else return false;
            }

            public static bool ValidateNumber(string Number)
            {
                if (!Validate(Number))
                    if (!ValidateTransit(Number))
                        return false;
                return true;
            }
        }
  }
  Badge.cs – класс полей для пропуска, id, номер, дата выдачи,
  проверка пропуска на актуальность.
  using BadgeSystem.Data;
```

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BadgeSystem.Models
{
    public class Badge
    {
        public int Id { get; set; }
        public int Number { get; set; }
        public DateTime ExpirationTime { get; set; }

        public Badge()
        {
            ExpirationTime = new DateTime();
        }

        public Badge(int Number, DateTime ExpirationTime)
        {
            this.Number = Number;
            this.ExpirationTime = new DateTime(ExpirationTime.Year
+ 1, ExpirationTime.Month, ExpirationTime.Day);
        }

        public override bool Equals(object obj)
        {
            try
            {
                Badge badge = (Badge)obj;
                if (this.Number == badge.Number)
                    return true;
                else return false;
            }
            catch { return false; }
        }

        public bool CheckRelevance() =>
            ExpirationTime <= DateTime.Now;

        public static int GenerateNumberEmployee(DbSubjects
_dataBase)
```

```
        {
            List<Employee> ListEmployee = _dataBase.Employees
                .Include(e => e.Badge)
                .OrderBy(e => e.Badge.Number)

                .ToList();

            for (int i = 1; i <= ListEmployee.Count; i++)
                if (i != ListEmployee[i - 1].Badge.Number)
                    return i;

            return ListEmployee.Count == 1500 ? -1 :
ListEmployee.Count + 1;
        }

        public static int GenerateNumberCompany(DbSubjects
_dataBase)
        {
            List<Company> ListCompany = _dataBase.Companies
                .Include(e => e.Badge)
                .OrderBy(c => c.Badge.Number)

                .ToList();

            int BadgeNumber = 1500;
            for (int i = 1; i <= ListCompany.Count; i++)
            {
                BadgeNumber += i;
                if (BadgeNumber != ListCompany[i -
1].Badge.Number)
                    return BadgeNumber;
            }

            return ListCompany.Count == 500 ? -1 :
ListCompany.Count + BadgeNumber;
        }
    }
}
```

Company.cs – класс, содержащий поля описание/примечание компаний.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

```
namespace BadgeSystem.Models
{
    public class Company:Subject
    {
        public string Title { get; set; }
        public string Note { get; set; }

        public Company() : base()
        {
            Title = string.Empty;
        }

        public override bool Equals(object obj)
        {
            try
            {
                Company company = (Company)obj;
                if (Title.Equals(company.Title,
StringComparison.CurrentCultureIgnoreCase) &&
                    PenaltyPoints == company.PenaltyPoints &&
                    Badge.Equals(company.Badge) &&

Automobiles.SequenceEqual(company.Automobiles))
                    return true;
                else return false;
            }
            catch { return false; }
        }

        public bool CheckEmptyField()
        {
            if (Title == null || Title.Equals(string.Empty) ||
                Automobiles == null) return true;
            if (Automobiles.Count == 0) return true;

            return false;
        }
    }
}
dbSubjects – производит связь с БД.
using BadgeSystem.Models;
using Microsoft.EntityFrameworkCore;
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BadgeSystem.Data
{
    public class DbSubjects:DbContext
    {
        public DbSet<Employee> Employees { get; set; }
        public DbSet<Company> Companies { get; set; }

        public DbSubjects(DbContextOptions<DbSubjects> options) :
base(options) { }
    }
}
```

Employee.cs – класс выдачи информации о сотруднике, Ф.И.О, должность.

ErrorViewModel.cs – класс для показа ошибок.

PageInfo.cs – класс для отображения интерфейса страницы.

Phone.cs – класс для добавления телефона с проверкой правильной записи телефона.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

namespace BadgeSystem.Models
{
    public static class Phone
    {
        public static bool ValidatePhone(string Phone)
        {
            string Pattern = @"8\d{10}";

            Regex Regex = new Regex(Pattern,
RegexOptions.IgnoreCase);
            Match Match = Regex.Match(Phone);
            if (Regex.IsMatch(Phone))
                return true;


            else return false;
```

```
        }
    }
}
```

Subject.cs – перечислены методы которые наследуются: поля для записи очков штрафа, телефона, марки авто, пропуск, методы для проверки корректности записи пропусков.

В папке Views следующие файлы:

AllList.cshtml – разметка главной страницы.

change.cshtml – страница, для изменения информации о сотруднике.

```
@model Subject

@{
    ViewData["Title"] = "Изменение";
}

<!DOCTYPE html>
<html lang="en">
<body id="body">
    <template id="employee">
        <div class="form-employee-container w-30"
id="objEmployee">
            <form class="form-inline">
                <div class="form-row">
                    <div class="col m-1">
                        <input type="text" class="form-control"
id="badge-number" readonly>
                    </div>
                </div>
                <div class="form-row">
                    <div class="col m-1">
                        <input type="text" class="form-control"
id="surname">
                    </div>
                </div>
                <div class="form-row">
                    <div class="col m-1">
                        <input type="text" class="form-control"
id="firstname">



                    </div>
                </div>
                <div class="form-row">
```

```
                    <div class="col m-1">
                        <input type="text" class="form-control"
id="patronymic">
                    </div>
                </div>
                <div class="form-row">
                    <div class="col m-1">

                        <input type="text" class="form-control"
id="post">
                    </div>
                </div>
                <div class="form-row">
                    <div class="col m-1">
                        <input type="text" class="form-control"
id="phone">
                    </div>
                </div>
                <div class="form-row">
                    <div class="col m-1">
                        <table id="table" class="table"></table>
                    </div>
                </div>
                <button type="button" class="btn btn-primary m-1"
onclick="SaveEmployee(); /*window.close();
opener.document.location.reload();*/">Сохранить изменения</button>
            </form>
        </div>
    </template>
    <template id="company">
        <div class="form-company-container" id="objCompany">
            <form class="form-inline">
                <div class="form-row">
                    <div class="col m-1">
                        <input type="text" class="form-control"
id="number-company" placeholder="Номер пропуска" readonly>
                    </div>

                </div>
                <div class="form-row">
                    <div class="col m-1">
                        <input type="text" class="form-control"
id="title" placeholder="Название компании">
```

```
                    </div>
                </div>
                <div class="form-row">
                    <div class="col m-1">
                        <input type="text" class="form-control"
id="penaltyPoints_C" placeholder="Очки штрафов">
                    </div>
                </div>
                <button type="button" class="btn btn-primary m-1"
onclick="SaveCompany(); /*window.close();
opener.document.location.reload();*/">Сохранить изменения</button>
            </form>
        </div>
    </template>
</body>
</html>
find.cshtml – страница поиска сотрудника.
!DOCTYPE html>

@using Microsoft.AspNetCore.Mvc;
<html lang="ru">

@{
    ViewData["Title"] = "Поиск";
}
<head>
    <script src="~/js/find.js"></script>
    <script src="~/js/actionsSubjects.js"></script>
</head>
<body>
    <div class="dropdown-container">
        <div class="dd-container">
            <div class="dropdown">

                <button class="btn btn-secondary dropdown-toggle"
href="#" role="button" id="dropdownFind" data-toggle="dropdown"
aria-haspopup="true" aria-expanded="false">
                    Выберите тип поиска
                </button>
                <div class="dropdown-menu" aria-
labelledby="dropdownMenuLink">
                    <h3 class="dropdown-header">Найти
сотрудника:</h3>
```

```
                    <ul>
                        <li><a href="#"
onclick="addFieldForFindByFIO(); return(false)">по Ф.И.О</a></li>
                        <li><a href="#"
onclick="addFieldForFindBySurname(); return(false)">по
фамилии</a></li>
                        <li><a href="#"
onclick="addFieldForFindByFirstname(); return(false)">по
имени</a></li>
                        <li><a href="#"
onclick="AddFieldForEmployeeFI(); return(false)">по имени и
фамилии</a></li>
                    </ul>
                    <h6 class="dropdown-header">Найти автомобиль:
</h6>
                    <ul>

                        <li><a href="#"
onclick="AddFieldForFindAuto(); return(false)">по номеру</a></li>
                    </ul>
                    <h6 class="dropdown-header">Найти организацию:
</h6>
                    <ul>
                        <li><a href="#"
onclick="addFieldForFindByTitle(); return(false)">по
названию</a></li>
                    </ul>
                    <h6 class="dropdown-header">найти пропуск</h6>
                    <ul>
                        <li><a href="#"
onclick="addFieldForFindByNumberBadges(); return(false)">по
номеру</a></li>
                    </ul>
                </div>
            </div>
        </div>
    </div>

    <div class="container">
        <div id="container-input"></div>
    </div>
    <div id="container-table" class="container-table"></div>
```

```
    <script>

    </script>
</body>
</html>
print.cshtml – страница печати.
addView.cshtml – страница добавления компании
addView.cshtml – страница добавления сотрудника
@model Badge

@{
    ViewData["Title"] = "Добавление сотрудника";
}

<html lang="ru">
<head>
    <script src="~/js/add.js"></script>
    <title>Добавить элемент</title>
</head>

<body>
    <div class="text-center">
        <form class="form-inline">
            <div class="form-row">
                <div class="input-container">

                    <input type="text" class="form-control"
id="number-employee" value="@Model.Number" readonly>
                </div>
                <div class="input-container">
                    <input type="text" class="form-control"
id="surname" placeholder="Фамилия сотрудника">
                </div>
                <div class="input-container">
                    <input type="text" class="form-control"
id="firstname" placeholder="Имя сотрудника">
                </div>
                <div class="input-container">

                    <input type="text" class="form-control"
id="patronymic" placeholder="Отчество сотрудника">
                </div>
```

```
                <div class="input-container">
                    <input type="text" class="form-control"
id="post" placeholder="Должность">
                </div>
Продолжение приложения А

                <div class="input-container">
                    <input type="text" class="form-control"
id="phone" placeholder="Номер телефона">
                </div>
                <div class="input-container">
                    <input type="text" class="form-control"
id="model" placeholder="Марка автомобиля">
                </div>
                <div class="input-container">
                    <input type="text" class="form-control"
id="autonumber" placeholder="Номер автомобиля">
                </div>
                <div class="input-container">
                    <button type="button" class="btn btn-primary"
onclick="AddAuto();">Добавить автомобиль</button>
                </div>
                <div class="container-table">
                    <table id="table" class="table"></table>
                </div>
                <div class="input-container">
                    <button type="button" class="btn btn-primary"
onclick="AddEmployee();">Добавить сотрудника</button>
                </div>
            </div>
        </form>
    </div>
</body>
</html>
list.cshtml – вставная конструкция
@model List<Subject>

<div>
    <table id="table" class="table">
        <tr>
            <th class="text-center">Номер пропуска</th>
            <th class="text-center">ФИО/Название организации</th>
            <th class="text-center">Должность</th>
```

```
            <th class="text-center">Модель автомобиля</th>
            <th class="text-center">Номер автомобиля</th>
            <th class="text-center">Номер телефона</th>
            <th class="text-center">Примечание</th>
            <th class="text-center">Действие</th>
        </tr>


        @{
            foreach (var subject in Model)
            {
                foreach (var auto in subject.Automobiles)
                {
                    if(subject.GetType() == new
Employee().GetType())
                    {
                         Employee employee = (Employee)subject;
                        <tr>
                            <th class="text-
center">@employee.Badge.Number</th>
                            <th class="text-
center">@string.Format("{0} {1} {2}", employee.Surname,
employee.Firstname, employee.Patronymic)</th>
                            <th class="text-
center">@employee.Post</th>
                            <th class="text-
center">@auto.Model</th>
                            <th class="text-
center">@auto.Number</th>
                            <th class="text-
center">@employee.Phone</th>
                            <th> </th>
                            <th class="text-center">
                                <div class="dropdown">
                                    <div role="button"
id="dropdownFind" data-toggle="dropdown" aria-haspopup="true"
aria-expanded="false">
                                        Действие
                                    </div>
                                    <div class="dropdown-menu"
aria-labelledby="dropdownMenuLink">
                                        <ul>
```

```
                                        <li><div role="button"
onclick="Change(@employee.Id)">Изменить</div></li>
                                        <li><div role="button"
onclick="deleteEmployee(@employee.Id)">Удалить</div></li>
                                            </ul>
                                        </div>
                                    </div>
                                </th>

                            </tr>
                        }

                        if(subject.GetType() == new
Company().GetType())
                        {
                            Company company = (Company)subject;
                            <tr>


                                <th class="text-
center">@company.Badge.Number</th>
                                <th class="text-
center">@company.Title</th>
                                <th> </th>
                                <th class="text-
center">@auto.Model</th>
                                <th class="text-
center">@auto.Number</th>
                                <th class="text-
center">@company.Phone</th>
                                <th class="text-
center">@company.Note</th>
                            </tr>

                        }
                    }
                }
            }
        </table>
</div>
```

_Layout.cshtml – общий шаблон всех страниц, встраивается в остальные конструкции.

В папке Controllers следующие файлы:

CommonController.cs – общий контроллер, связан с главным окном.
CompanyController.cs – контролер работы с кнопками.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BadgeSystem.Data;
using BadgeSystem.Models;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BadgeSystem.Controllers
{
    public class CompanyController : Controller
    {
        DbSubjects _dataBase;
        public CompanyController(DbSubjects context)
        {
            _dataBase = context;
        }

        public IActionResult Add()
        {
            int BadgeNumber =
Badge.GenerateNumberCompany(_dataBase);
            if (BadgeNumber == -1) return StatusCode(507, new {
Message = "reached record limit" });

            Badge badge = new Badge(BadgeNumber, DateTime.Now);
            return View("addView", badge);
        }

        [HttpPost]
        public IActionResult FindByTitle([FromBody] string Title)
        {
            if (Title == null || Title.Equals(string.Empty))
return StatusCode(204, new { Message = "Incorrect data" });

            List<Subject> result = new List<Subject>();
            result.AddRange(_dataBase.Companies
                .Include(c => c.Badge)
                .Include(c => c.Automobiles)
```

```
                .Select(c => c)
                .Where(c => c.Title.Equals(Title)));

            return PartialView("~/Views/PartialViews/list.cshtml",
result);
        }

        [HttpPost]
        public async Task<IActionResult> AddToDataBase([FromBody]
Company company)
        {
            if (company == null) return StatusCode(204, new {
Message = "incomplete data" });
            if (company.CheckEmptyField()) return StatusCode(204,
new { Message = "incomplete data" });
            if (!company.Validate()) return StatusCode(204, new {
Message = "not valid auto or phone " });
            if (!company.Badge.CheckRelevance()) return
StatusCode(204, new { Message = "invalid badge" });

            if (_dataBase.Companies.Count() > 0)
                if (_dataBase.Companies.Any(c => c.Badge ==
company.Badge))
                    return StatusCode(500, new { Message =
"company already exist" });

            await _dataBase.Companies.AddAsync(company);
            await _dataBase.SaveChangesAsync();
Окончание приложения А
            return Ok();
        }
    }
}
```

EmployeeController.cs – контроллер выдачи информации о сотруднике.