СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ 6

ВВЕДЕНИЕ 7

1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ 9

1.1 Постановка задачи 9

1.2 Обзор существующих аналогов 10

1.2.1 Word Press 10

1.2.2 Joomla! 12

1.2.3 Drupal 14

1.2.4 Вывод 16

1.3 Веб-сервер 16

1.4 Принципы построения клиент-серверных систем 17

2 ОБЗОР ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ 20

2.1 Программная платформа Microsoft .NET 20

2.2 Язык программирования C# 22

2.3 ASP.NET MVC 5 Framework 25

2.4 Язык программирования TypeScript 28

2.5 AngularJS 29

3 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ СИСТЕМЫ 31

3.1 Обзор платформы BBWT 31

3.2 Обзор архитектуры приложения 33

3.2.1 Слой представления 35

3.2.2 Серверный слой 37

3.2.3 Слой доступа к данным 39

3.2.4 Слой сервисов 40

3.3 Пример реализация нового модуля на платформе BBWT 42

4 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ ПЛАТФОРМЫ ДЛЯ РЕАЛИЗАЦИИ КОРПОРАТИВНЫХ САЙТОВ 57

4.1 Расчёт сметы затрат, необходимых для создания ПО 57

4.2 Расчёт экономической эффективности у разработчика 67

4.3 Вывод 69

5 ОБЕСПЕЧЕНИЕ ПОЖАРНОЙ БЕЗОПАСНОСТИ В КОМПАНИИ «ПАНДА СИСТЕМС» 70

ЗАКЛЮЧЕНИЕ 75

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 76

# ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

В настоящей пояснительной записке применяются следующие определения и сокращения.

CMS – Content management system – система управления содержимым.

[CMF](https://ru.wikipedia.org/wiki/CMF) – Content management Framework – фреймворк программной системы.

UTC  – Coordinated Universal Time – всемирное координированное время.

URL – Uniform Resource Locator – единый указатель ресурсов.

URI – Uniform Resource Identifier – унифицированный идентификатор ресурса.

XML – eXtensible Markup Language – расширяемый язык разметки.

CTS – Common Type System – общая система типов.

ООП – объектно-ориентированное программирование.

SQL – structured query language – язык структурированных запросов.

LINQ – Language Integrated Query – интегрированные в язык запросы.

MSIL – Microsoft Intermediate Language – промежуточный язык для платформы Microsoft .NET.

COM – Component Object Model – объектная модель компонентов.

HTTP – HyperText Transfer Protocol – протокол передачи гипертекста.

MVC – Model-View-Controller – модель-представление-контроллер.

MVW – Model-View-Whatever – модель-вид-что угодно.

DOM – Document Object Model – объектная модель документа.

БД – база данных.

СУБД – система управления базами данных.

REST – Representation State Transfer – передача состояния представления.

SOAP – Simple Object Access Protocol – простой протокол доступа к объектам.

API – Application Programming Interface – интерфейс программирования приложений.

JSON – JavaScript Object Notation – текстовый формат обмена данными, основанный на языке JavaScript.

BSON – Binary JavaScript Object Notation – подмножество JSON, предназначенное для хранения данных в двоичном формате.

LоС – Lines of Code – количество строк кода.

# ВВЕДЕНИЕ

В связи с огромным скачком в сфере информационных технологий и популяризации использования интернета в повседневной жизни, сегодня активно происходит перенос различных сервисов в интернет. Многие привычные программы уже сейчас перенесены или переносятся во всемирную сеть, что позволяет пользователям иметь к ним постоянный доступ с любого устройства, позволяющего подключиться к интернету. Не обошло это стороной и различные крупные компании, основной сферой деятельности которых, является доставка и производство товаров. Многие компании уже давно перенесли многие сервисы, будь то заказ товаров или управление доставкой, во всемирную сеть, что позволило облегчить и ускорить работу с ними. Ведь очень часто возникает необходимость иметь к ним постоянный доступ, который не могут предоставить обычные десктопные приложения. Перенос различных сервисов в интернет также позволил пользоваться ими не только сотрудникам компаний, но и их клиентам, благодаря чему упрощается и улучшается взаимосвязь между клиентами и производителем.

Как правило, разработкой онлайн систем, в частности сайтов, занимаются ИТ-компании у которых уже есть опыт в данной области, зачастую работа в данной области становится профильной. Ввиду чего многие проекты, которыми занимается компании, по функциональным требованиям очень похожи. И чтобы облегчить работу программистов и ускорить процесс выполнения проекта, проводится поиск различных решений, которые бы помогли реализовать эти задачи. Одним из таковых решений является создание некого начального проекта-шаблона, на котором потом будут базироваться новые проекты компании. Следует отметить, что не всегда таковое решение является верным, если проекты очень сильно отличаются друг от друга по функциональности и в них сложно выделить какие-либо общие части, то разработка такого шаблонна, может занять очень много времени и ресурсов. Если рассматривать область разработки сайтов для компаний, которые занимаются поставками и производством, то можно легко выработать общие требования, вследствие чего создание шаблона под такие нужды будет иметь смысл.

Требования для такого шаблона будут примерно следующие:

* необходимо наличие сильной административной части для разделения работы сотрудников компании и клиентов;
* необходимо наличие набора различных готовых модулей, предназначенных для работы с большим количеством данных;
* необходима возможность без труда менять приложение под нужды различных заказчиков.

Со стороны разработчиков можно выделить следующие требования к платформе:

* простота в освоение и развертывание проекта;
* гибкая архитектура;
* наличие различных средств упрощения и автоматизация разработки;
* набор готовых решений для работы с различными типами данных;
* готовая реализация основных модулей системы.

Также следует отметить, что данная система не должна оставаться на месте, а постоянно модифицироваться и дополняться новыми готовыми решениями.

Другой стороной разработки такого шаблона является сокращение затрат и времени на производство нового проекта, что несомненно принесет выгоду для компании.

В результате можно сказать, что разработка шаблона не только возможна, но и необходима.

# 1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

## Постановка задачи

В рамках дипломного проекта поставлена цель разработать веб-платформы с использованием технологии ASP.NET MVC, предназначенной для упрощения реализации корпоративных веб-сайтов для нужд заказчиков. Минимальными требованиями к реализации программного продукта являются:

1. Веб-платформа должен иметь модульную архитектуру, позволявшую для программиста быстро и просто изменять уже существующие модули и реализовывать новые, необходимые для заказчика.
2. Модульная архитектура предполагает динамическую схему обрабатываемых данных. Важной деталью являет минимизировать работу программиста с БД для упрощения добавления новых сущностей.
3. Реализовать удобную систему пользовательской функциональности, авторизацию, регистрацию, роли и группы пользователей, права доступа, управление доступом для администратора.
4. Система должна иметь возможность работы с многоязычными интерфейсами: новые языки для локализации должны добавляться, а существующие – редактироваться.
5. Реализовать удобную систему логирования информационных и отладочных сообщений, а также систему конфигурационных ключей. Возможность просмотра и редактирования этих данных с помощью пользовательского интерфейса системы.
6. Необходимо наличие набора готовых модулей предназначенных для решения популярных требований заказчика. Для этого используется Kendo UI, который предоставляет большой набор готовых модулей реализованных на JavaScript’е.

Для достижения поставленной цели необходимо решить следующие задачи:

* изучить существующие функциональные аналоги платформ для упрощения разработки типовых сайтов;
* на основании произведенного анализа выбрать и изучить необходимые технологии для реализации дипломного проекта;
* спроектировать гибкое архитектурное решение платформы для разработки типовых сайтов.

## Обзор существующих аналогов

Платформу BBWT можно отнести к классу CMS предназначенных для реализации узкоспециализированных сайтов. Многие компании, которые занимаются разработкой большого количество проектов в одной области со временем приходят к тому, что не нужды каждый раз все реализовывать с нуля, и тогда возникает необходимость в шаблоне, на котором можно быстро создавать новый проект и который можно легко изменять под нужды заказчика. Как правило, такого рода шаблоны используются только внутри компаний и на открытый рынок не выпускаться. Но кроме них существует множество различных платформ для создания сайтов с большим набором разной функциональности, их мы и рассмотрим.

### 1.2.1 Word Press

WordPress – является одной из самых популярных CMS во всем мире, по независимым данным около 10% всех сайтов реализовано именно на нем. Обусловлено это довольно большими возможностями, предоставляемыми данной системой управления контентом. Данная платформа позволяет создавать как блоги, так и сайты различной сложности, однако, известна эта CMS больше в качестве «движка» для блогов, в чем и является бесспорным лидером[1].

WordPress является оптимальным вариантом для создания, как несложного информативного сайта, так и качественного блога. Несомненным преимуществом данной CMS является огромное множество модулей и шаблонов, которые могут использоваться дополнительно, делая ресурс еще более удобным и привлекательным для его посетителей. Кроме того, некоторые изменения в дизайн сайта можно внести самостоятельно.

Конечно, не существует идеальной CMS, и WordPress не является исключением. Во-первых, сайт может недостаточно быстро функционировать и демонстрировать сбои в работе в случае наличия каких-либо недостатков в скриптах или при большом количестве посещений ресурса.

Не подойдет WordPress и для создания сложных сайтов, отличающихся большой функциональностью, интернет-магазинов или серьезного портала. Для реализации таких идей следует выбирать более универсальные CMS.

На рисунках 1.1 и 1.2 показаны примеры сайтов реализованных на WordPress.

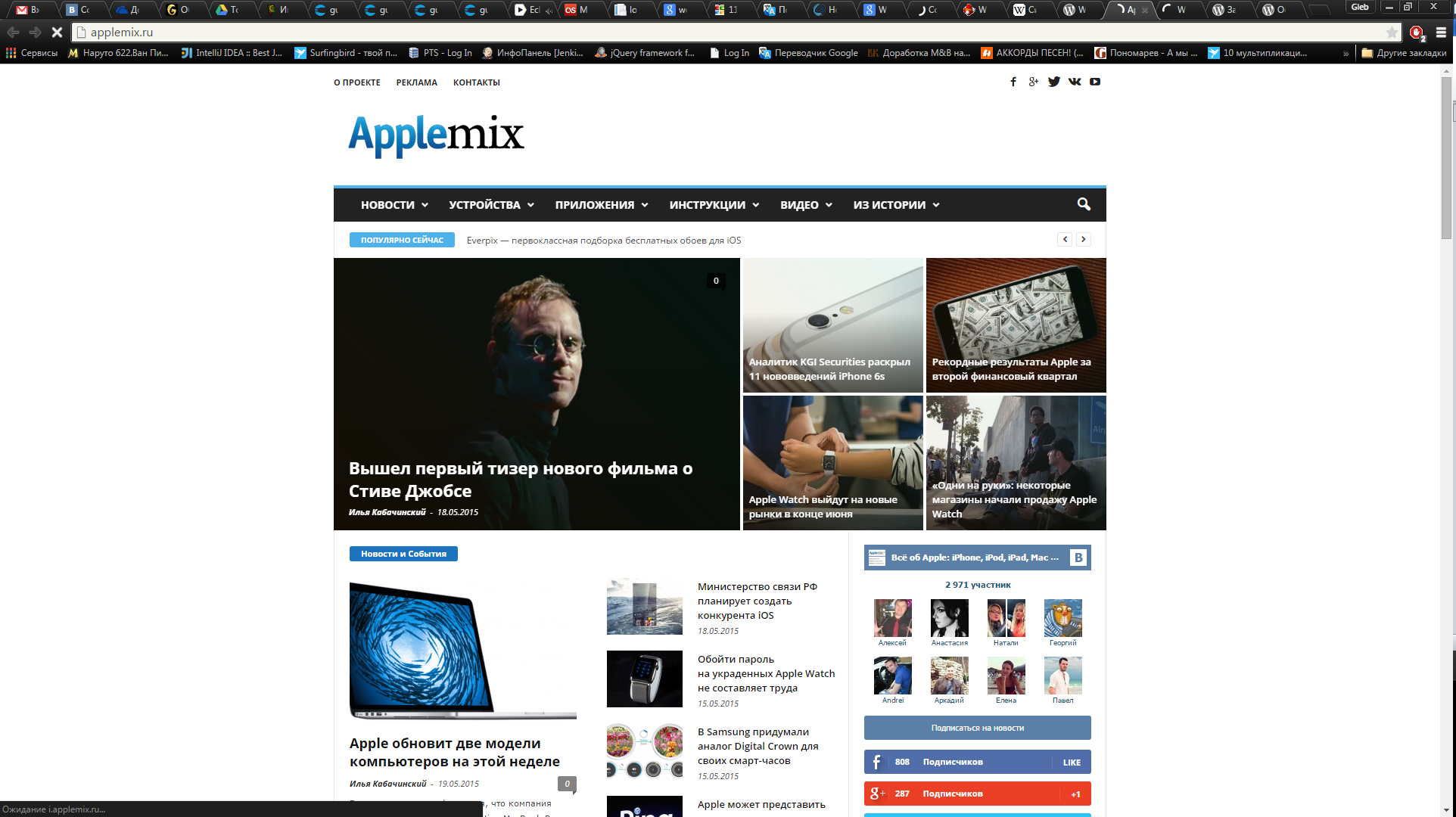


Рисунок 1.1 – AppleMix.ru, пример реализации новостного сайта на WordPress.

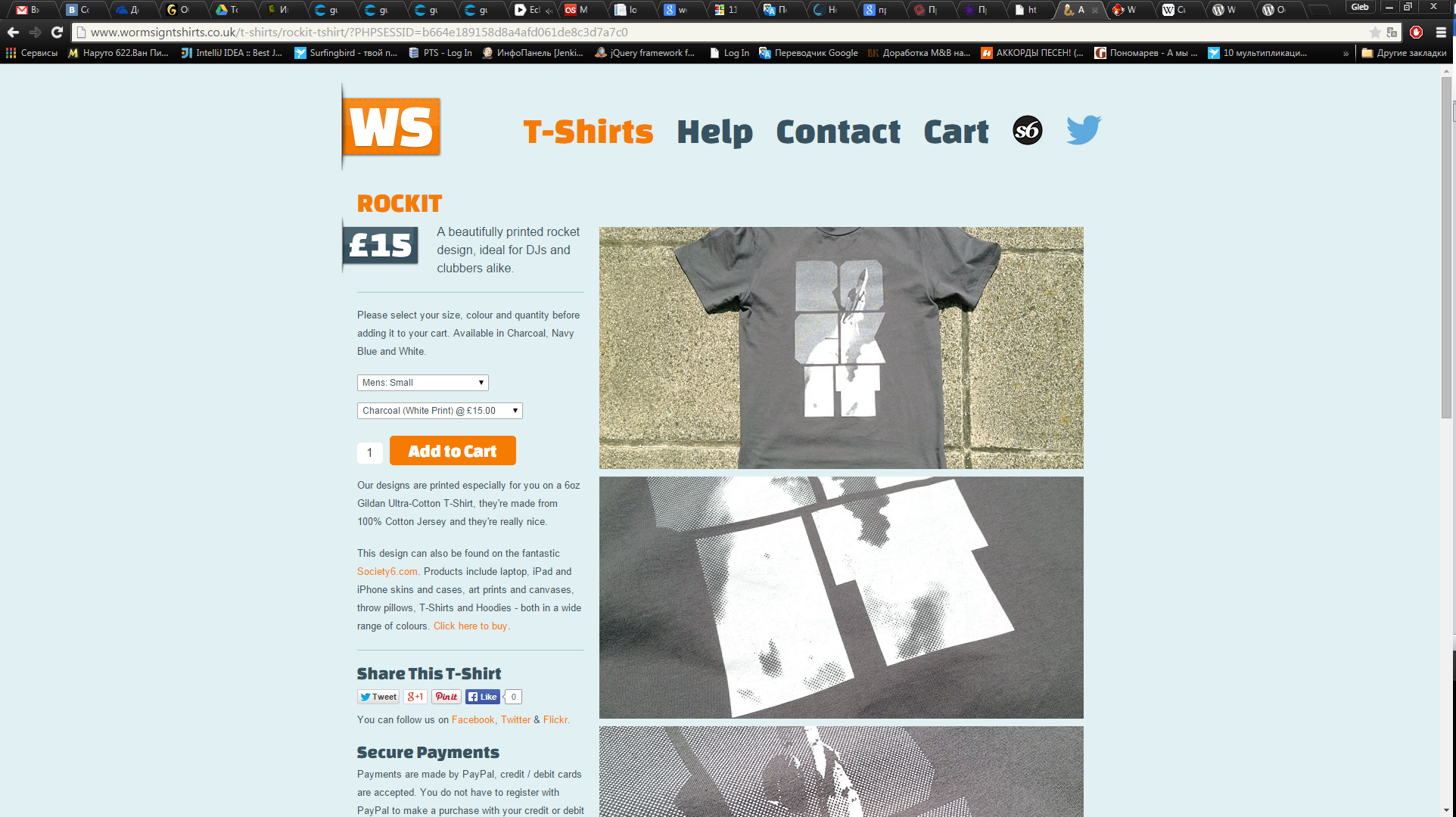


Рисунок 1.2 – WormSigntShirts.co.uk, пример реализации онлайн-магазина на WordPress

Можно выделить следующие достоинства данной платформы:

* локальная установка;
* переносимая основа;
* наличие большого набора модулей;
* наличие совместимости с UTC;
* наличие системы управления пользователями;
* динамическая генерация страниц;
* простой редактор;
* наличие большого выбора шаблонов;
* эргономичный дизайн и пользовательский интерфейс;
* авторизация с помощью социальных сетей.

Однако помимо достоинств платформы можно выделить и следующие недостатки:

* платформа ориентирована на простые сайты: блоги, новостные ресурсы, примитивные онлайн-магазины;

### 1.2.2 Joomla!

Joomla! – это система управления контентом, написанная на языке PHP и использующая в качестве хранилища содержания базу данных MySQL[2]. Joomla! является свободным программным обеспечением, защищённым лицензией GPL. Одной из главных особенностей Joomla! является относительная простота управления при практически безграничных возможностях и гибкости при изготовлении сайтов. Название Joomla! фонетически идентично слову «Jumla», которое в переводе с языка суахили означает «все вместе» или «единое целое», что отражает подход разработчиков и сообщества к развитию системы[3].

Система управления содержанием Joomla! является ответвлением широко известной CMS Mambo. Команда независимых разработчиков отделилась от проекта Mambo по причине разногласий в экономической политике. 16 сентября 2005 года в свет вышла первая версия Joomla!, являющаяся по сути переименованной Mambo и включающая в себя исправления найденных на тот момент ошибок и уязвимостей.

Разработчики обещают, что к выходу версии 2.0 CMS будет полностью переписан и адаптирован под PHP 6. В данный момент последними версиями являются 1.0.15 и 1.5.10.

CMS Joomla! включает в себя различные инструменты для изготовления веб-сайта. Важной особенностью системы является минимальный набор инструментов при начальной установке, который обогащается по мере необходимости. Это снижает загромождение административной панели ненужными элементами, а также снижает нагрузку на сервер и экономит пространство на хостинге.

Joomla! позволяет отображать интерфейс фронтальной и административной части на любом языке. Каталог расширений содержит множество языковых пакетов, которые устанавливаются штатными средствами администрирования. Доступны пакеты русского, украинского, белорусского и ещё некоторых языков постсоветского пространства.



Рисунок 1.3 –Vanerumgroup.com, пример реализации корпоративного сайта на Joomla!

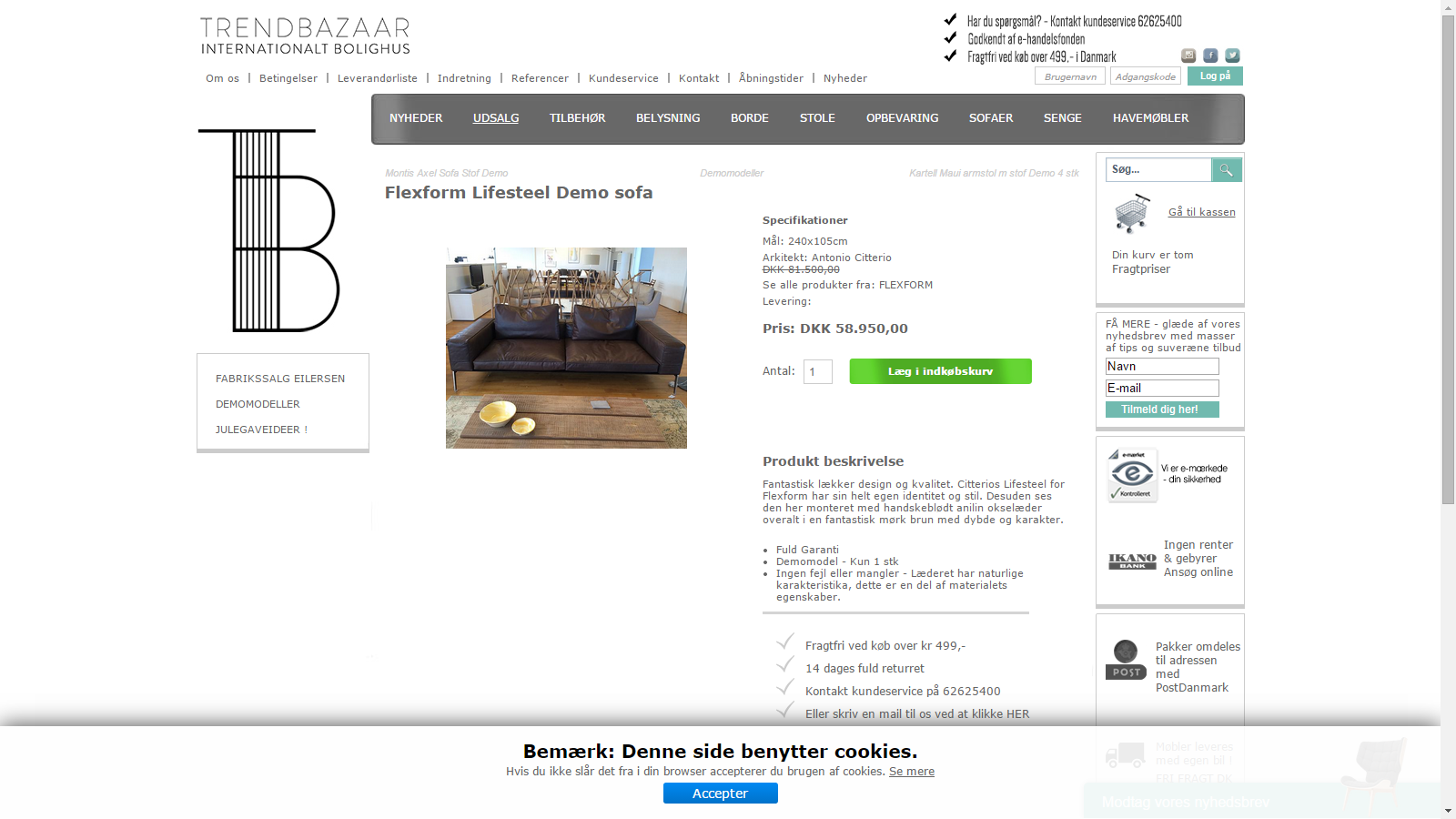


Рисунок 1.4 – Trendbazaar.dk, пример реализации онлайн-магазина на Joomla!

Достоинства данной платформы:

* распространяется бесплатно;
* локальная установка;
* переносимая основа;
* наличие большого набора модулей;
* наличие системы управления пользователями;
* наличие большого выбора шаблонов;
* эргономичный дизайн и пользовательский интерфейс;

Недостатки:

* проблемы при обновлении платформы и компонентов;
* сложна в освоение;
* долгая скорость загрузки страниц из-за больших компонентов;

### 1.2.3 Drupal

Drupal – [система управления содержимым](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D1%81%D0%BE%D0%B4%D0%B5%D1%80%D0%B6%D0%B8%D0%BC%D1%8B%D0%BC), используемая также как каркас для веб-приложений ([CMF](https://ru.wikipedia.org/wiki/CMF)), написанная на языке [PHP](https://ru.wikipedia.org/wiki/PHP) и использующая в качестве хранилища данных реляционную базу данных (поддерживаются [MySQL](https://ru.wikipedia.org/wiki/MySQL), [PostgreSQL](https://ru.wikipedia.org/wiki/PostgreSQL" \o "PostgreSQL) и другие). Drupal является [свободным программным обеспечением](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), защищённым лицензией [GPL](https://ru.wikipedia.org/wiki/GNU_General_Public_License), и развивается усилиями энтузиастов со всего мира[4].

В Drupal предлагается гибкая схема организации структуры сайта на основе таксономии. Таксономия – механизм, позволяющий создавать произвольное количество тематических категорий для содержимого сайта и ассоциировать их с модулями, обеспечивающими ввод и вывод информации. Категории могут представлять плоские или иерархические списки, либо сложные структуры, где элемент может иметь несколько «родителей» и несколько дочерних элементов. С помощью подобной схемы одними и теми же модулями возможна организация различных вариантов структуризации содержимого. Например, легко создаётся сквозной список «ключевых слов» для всех документов сайта и т.п.

Другая парадигма появилась с созданием в Drupal расширения Content Construction Kit (CCK). CCK позволяет дополнять документы новыми полями различных типов – от полей ввода URL и e-mail, до полей хранения и отображения мультимедийных файлов. Также посредством дополнительных модулей к CCK (например, Node reference) можно организовать связи между документами, не используя механизм таксономии.

Drupal имеет модульную архитектуру с компактным ядром, предоставляющим API, к которому могут обращаться модули. Стандартный набор модулей включает такие функции, как новостная лента, блог, форум, загрузка файлов, сборщик новостей, голосования, поиск и другие. Дизайн сайта меняется также посредством специальных модулей – «тем оформления».

Дизайн сайта на Drupal строится на основе сменных тем оформления. Как таковой нет единственной схемы построения дизайна. Взамен Drupal даёт возможность использовать различные «движки тем», использующие шаблоны, удобные для редактирования (шаблоны XML в движках xtemplate и Smarty или шаблоны на HTML и встроенный PHP в движке phptemplate и т. п.), либо создавать темы оформления, напрямую обращающиеся к API Drupal. В комплект поставки Drupal включён движок тем на основе phptemplate и несколько примеров тем. Другие движки тем можно скачать на сайте проекта.



Рисунок 1.5 – Forbes.ru, пример реализации сайта на Drupal

Достоинства данной платформы:

* распространяется с открытым исходным кодом;
* гибкая архитектура;
* огромное число модулей;
* встроенная система кеширования, способной снизить нагрузку на сервер и сократить время генерации страницы.
* встроенная поисковая система.

Недостатки:

* + сложен в освоении;
  + нет обратной совместимости версий.

### 1.2.4 Вывод

Основываясь на результатах ознакомления с существующими функциональные аналогами платформы для упрощения разработки типовых сайтов, можно утверждать, что большинство бесплатных CMS не обладают необходимой мощностью и функциональности, а также часто довольно сложны в освоении.

Основными недостатками рассмотренных систем можно считать:

* узкая направленность некоторых платформ; реализовать что-либо больше сайта блога зачастую бывает очень сложно;
* разработка на многих платформы сильно усложнены за счет большого количества ненужных модулей.

## 1.3 Веб-сервер

Веб-сервер – это сервер, принимающий HTTP‑запросы от клиентов. Обычно в роли клиентов выступают веб-браузеры, и выдающий им HTTP‑ответы, вместе с которыми передаются HTML‑страницы, изображения, файлы, медиа-потоки или другие данные. Веб-серверы являются основой Всемирной паутины.

Фактически веб-сервером называют как программное обеспечение, обеспечивающее все вышеперечисленные действия, так и компьютер, на котором это программное обеспечение работает. Клиенты получают доступ к веб-серверу по средствам единого указателя ресурсов – URL‑адресу нужной им веб-страницы или другого ресурса, располагаемого на сервере.

Единый указатель ресурсов (англ. URL – Uniform Resource Locator) – это единообразный локатор (определитель местонахождения) веб-ресурса. URL был изобретён Тимом Бернерсом-Ли в 1990 году в стенах Европейского совета по ядерным исследованиям в Женеве, Швейцария. URL стал фундаментальной инновацией в Интернете. Изначально URL предназначался для обозначения мест расположения ресурсов (чаще всего файлов) во Всемирной паутине. [5]

Сейчас URL применяется для обозначения адресов почти всех ресурсов Интернета. Стандарт URL закреплён в документе RFC 1738, прежняя версия была определена в RFC 1630. Сейчас URL позиционируется как часть более общей системы идентификации ресурсов URI, сам термин URL постепенно уступает место более широкому термину URI. Стандарт URL регулируется организацией IETF и её подразделениями.

На просторах сети Интернет в данный момент работает около 390 миллионов различных веб-серверов. Среди которых самыми популярными являются: Apache HTTP-Server от компании Apache и IIS от компании Microsoft. Статистика использование веб-серверов представлена на рисунке 1.7



Рисунок 1.7 – Статистика использование веб-серверов на конец 2014 г.

## 1.4 Принципы построения клиент-серверных систем

Для разработки клиент-серверных систем имеется два подхода:

* + построение систем на основе двухзвенной архитектуры;
  + построение систем на основе трехзвенной архитектуры.

Двухзвенная архитектура состоит из клиентской и серверной части. Как правило, серверная часть представляет собой сервер БД, на котором расположены общие данные. А клиентская часть представляет приложение, которое связывается с сервером БД, осуществляет к нему запросы и получает ответы. Такие системы используются в локальных сетях, т. к. нет затруднений с установкой клиентской части. Также системы с такой архитектурой более безопасны, т.к. могут использовать собственные протоколы передачи данных, не известные злоумышленникам. Поэтому многие крупные компании, которые располагаются не в едином месте и для соединения подразделений используют глобальную сеть Интернет, выбирают именно такую архитектуру построения клиент-серверных систем. Двухзвенная архитектура показана на рисунке 1.9.

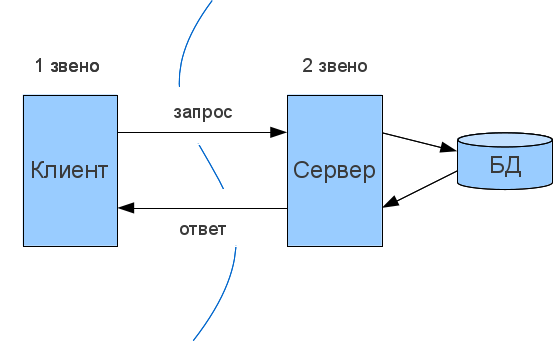


Рисунок 1.9 – Трехзвенная архитектура

При разработке информационных систем, рассчитанных на широкую аудиторию, возникают проблемы с использованием двухзвенной архитектуры. Во-первых, пользователю необходимо иметь в наличии клиентскую часть, а во-вторых, у неопытного пользователя, могут возникнуть проблемы с конфигурированием такой системы. Поэтому в последнее время, более часто разрабатывают приложения на базе трехзвенной архитектуры.

Трезвенная архитектура также состоит из двух частей: клиента и сервера. Но серверная часть в этой архитектуре представляет собой сервер приложений и сервер БД. А в качестве клиента выступает web‑браузер. Такая система очень проста для пользователя. Ему необходимо знать только адрес сервера приложения и наличие web‑браузера на рабочем компьютере. Все данные представляются в виде html‑разметки с использование графики (jpeg, gif, flash), каскадных слоев CSS и JavaScript. Передача запросов от клиента к серверу приложений происходит по средствам CGI‑интерфейса. Сервер приложений общается с сервером БД, используя другой интерфейс, зависящий от того, на основе каких средств строится конкретная информационная система. Недостатками такой архитектуры является использование общеизвестных протоколов передачи данных. Злоумышленник может осуществить взлом системы, если она будет недостаточно хорошо проверять поступившие запросы от клиента[18]. Трехзвенная архитектура показана на рисунке 1.9.

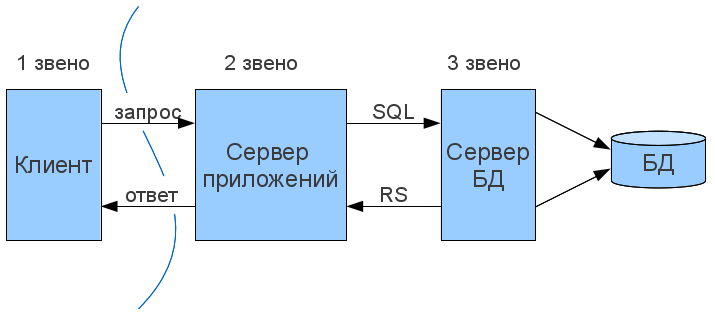


Рисунок 1.9 – Трехзвенная архитектура.

При разработке клиент-серверных приложений необходимо учитывать:

* + на каких пользователей будет рассчитана данная информационная система;
  + какие требования предъявляются к безопасности.

Если информационная система должна быть общедоступной и рассчитана на широкую аудиторию, то необходимо использовать трехзвенную архитектуру.

Если информационная система используется внутри предприятия, доступ имеют к ней ограниченные пользователи и требуется создать максимально безопасную и защищенную систему, то следует отдать предпочтение двухзвенной архитектуре.

# 2 ОБЗОР ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ

## 2.1 Программная платформа Microsoft .NET

Программная платформа Microsoft .NET является одной из реализаций стандарта ECMA-335 и представляет собой современный инстру­мент для соз­дания клиентских и серверных приложений для операционной системы Windows. Первая общедоступная версия .NET Framework вышла в феврале 2002 года. С тех пор платформа активно развивалась и на данный момент было выпущено шесть версии данного продукта. На данный момент номер последней версии .NET Framework – 4.5.3. Платформа Microsoft .NET была призвана решить некоторые наболевшие проблемы, скопившиеся на момент её выхода в средствах разработки приложений под Windows. Ниже перечислены некоторые из них [6]:

* сложность создания надежных приложений;
* сложность развертывания и управления версиями приложений;
* сложность создания переносимого программного обеспечения;
* отсутствие общей целевой платформы для создателей компиляторов;
* проблемы с безопасным исполнением непроверенного кода;
* великое множество различных технологий и языков программирования, которые не совместимы между собой.

Многие из этих проблем были решены в платформе Microsoft .NET. Далее более подробно рассматривается ее внутреннее устройство.

Основными компонентами Microsoft .NET являются общая языковая исполняющая среда и стандартная библиотека классов. Исполняющая среда представляет собой виртуальную машину и набор сервисов обслуживающих исполнение программ, написанных для Microsoft .NET. Ниже приводится перечень выполняемых ею задач [6]:

* загрузка и исполнение управляемого кода;
* управление памятью при размещении объектов;
* изоляция памяти приложений;
* проверка безопасности кода;
* преобразование промежуточного языка в машинный код;
* доступ к расширенной информации о типах;
* обработка исключений, включая межъязыковые исключения;
* взаимодействие между управляемым и неуправляемым кодом;
* поддержка сервисов для разработки.

Программы, написанные для Microsoft .NET представляют собой набор типов, взаимодействующих между собой. Microsoft .NET имеет общую систему типов (Common Type System, CTS). Данная спецификация описывает определения и поведение типов, создаваемых для Microsoft .NET [6]. В част­ности, в данной спецификации описаны возможные члены типов, механизмы сокрытия реализации, правила наследования, типы-значения и ссылочные типы, особенности полиморфизма и другие возможности.

Некоторые из возможностей, предоставляемых Microsoft .NET:

* верификация кода, расширенная информация о типах во время исполнения;
* сборка мусора, безопасность типов;
* генерация и сохранение внутри сборок подробных метаданных о типах из которых со стоит исполняемая программа.

Одной из особенностей Microsoft .NET, обеспечивающей переносимость программ без необходимости повторной компиляции, является представление исполняемого кода приложений на общем промежуточном языке. Промежуточный язык является стековым, объектно-ориентированным ассемблером [6].

Данный язык очень удобен в качестве целевого языка для создателей компиляторов и средств автоматической проверки кода для платформы Microsoft .NET. Наличие промежуточного языка и необходимость создания производительных программ подразумевают наличие преобразования промежуточного кода в машинный код во время исполнения программы. Одним из компонентов общей языковой исполняющей среды, выполняющим данное преобразование, является компилятор времени исполнения, транслирующий промежуточный язык в машинные инструкции, специфические для архитектуры компьютера, на котором исполняется программа.

Ручное управление памятью всегда являлось подверженной ошибкам работой. Ошибки в управлении памятью являются одними из наиболее сложных в устранении типами программных ошибок, также эти ошибки обычно приводят к непредсказуемому поведению программы, поэтому в Microsoft .NET управление памятью происходит автоматически [16]. Когда объект дан­ных перестает быть нужным, занятая под него память автоматически освобождается и используется для построения новых объектов.

Имеются различные методы реализации такого автоматического распределения памяти [6]. В Microsoft .NET для автоматического управления памятью используется механизм сборки мусора. Существуют различные алгоритмы сборки мусора со своими достоинствами и недостатками. В Microsoft .NET используется алгоритм пометок в сочетании с различными оптимизациями, такими как разбиение всех объектов по поколениям и использование различных куч для больших и малых объектов.

Ниже перечислены, без приведения подробностей, некоторые важные

функции исполняемые общей языковой исполняющей средой:

* + обеспечение многопоточного исполнения программы;
  + поддержание модели памяти, принятой в исполняющей средой;
  + поддержка двоичной сериализации;
  + управление вводом и выводом;
  + структурная обработка исключений;

## 2.2 Язык программирования C#

C# – объектно-ориентированный, типобезопасный язык программирования общего назначения. Язык создавался с целью повысить продуктивность программистов. Для достижения этой цели в языке гармонично сочетаются простота, выразительность и производительность промежуточного кода, получаемого после компиляции. Язык C# является платформенно нейтральным, но создавался для хорошей работы с Microsoft .NET [7]. Этот язык сочетает простой синтаксис, похожий на синтаксис языков C++ и Java, и полную поддержку всех современных объектно-ориентированных концепций и подходов. В качестве ориентира при разработке языка было выбрано безопасное программирование, нацеленное на создание надежного и простого в сопровождении кода [8].

Язык имеет богатую поддержку парадигмы объектно-ориен­тирован­ного программирования, включающую поддержку инкапсуляции, наследования и полиморфизма. Отличительными чертами C# с точки зрения парадигмы ООП являются:

* + Унифицированная система типов. В C# сущность, содержащая данные и методы их обработки, называется типом. В C# все типы производны от одного базового класса.
  + Классы и интерфейсы. В классической объектно-ориентированной парадигме существуют только классы. В C# дополнительно существуют и другие типы, например, интерфейсы. Интерфейс – это сущность, напоминающая классы, но содержащая только определения членов. Конкретная реализация указанных членов интерфейса происходит в типах, реализующих данный интерфейс. В частности, интерфейсы могут быть использованы при необходимости проведения множественного наследования.
  + Свойства, методы и события. В чистой объектно-ориентированной парадигме все функции являются методами. В C# методы являются лишь одной из возможных разновидностей членов типа, в C# типы также могут содержать свойства, события и другие члены. Свойство – это такая разновидность функций, которая инкапсулирует часть состояния объекта. Событие – это разновидность функций, которые реагируют на изменение состояния объекта [7].

В большинстве случаев C# обеспечивает безопасность типов в том смысле, что компилятор контролирует чтобы взаимодействие с экземпляром типа происходило согласно контракту, который он определяют. Например, компилятор C# не скомпилирует код, который обращается со строками, как если бы они были целыми числами. Говоря более точно, C# поддерживает статическую типизацию, в том смысле что большинство ошибок типов обнаруживаются на стадии компиляции. За соблюдение более строгих правил безопасности типов следит исполняющая среда. Статическая типизация позволяет избавиться от широкого круга ошибок, возникающих из-за ошибок типов. Она делает написание и изменение программ более предсказуемыми и надежными, кроме того, статическая типизация позволяет существовать таким средствам как автоматическое дополнение кода и его предсказуемый статический анализ.

C# полагается на автоматическое управление памятью со стороны исполняющей среды, предоставляя совсем немного средств для управления жизненным циклом объектов. Не смотря на это, в языке все же присутствует поддержка работы с указателями. Данная возможность предусмотрена для случаев, когда критически важна производительность приложения или необходимо обеспечить взаимодействие с неуправляемым кодом [7].

Как уже упоминалось C# не является платформенно зависимым языком. Благодаря усилиям компании Xamarin возможно писать программы на языке C# не только для операционных систем Microsoft, но и ряда других ОС. Существуют инструменты создания приложений на C# для серверных и мобильных платформ, например, iOS, Android, Linux и других.

Далее приводится краткий обзор развития языка.

Первая версия C# была похожа по своим возможностям на Java 1.4, несколько их расширяя: так, в C# имелись свойства (выглядящие в коде как поля объекта, но на деле вызывающие при обращении к ним методы класса), индексаторы (подобные свойствам, но принимающие параметр как индекс массива), события, делегаты, циклы foreach, структуры, передаваемые по значению, автоматическое преобразование встроенных типов в объекты по запросу, атрибуты, встроенные средства взаимодействия с неуправляемым кодом и прочее [9].

Версия Microsoft .NET 2.0 привнесла много новых возмож­нос­тей в срав­нении с предыдущей версией, что отразилось и на языках под эту платформу. C# 2.0 окончательно вышел 7 ноября 2005 года вместе с Visual Studio 2005 и Microsoft .NET 2.0. Ниже перечислены новые возможности в версии 2.0.

* + Возможность разделения реализации класса на несколько файлов.
  + Параметризованные типы. В отличие от шаблонов C++, они поддерживают некоторые дополнительные возможности и работают на уровне виртуальной машины. Вместе с тем, параметрами обобщённого типа не могут быть выражения, они не могут быть полностью или частично специализированы, не поддерживают шаблонных параметров по умолчанию, от шаблонного параметра нельзя наследоваться.
  + Новая форма итератора, позволяющая создавать сопрограммы с помощью ключевого слова yield.
  + Обнуляемые типы-значения, представляющие собой те же самые типы-значения, но способные принимать также значение null. Такие типы позволяют улучшить взаимодействие с базами данных через язык SQL.
  + Поддержка 64-разрядных вычислений позволяет увеличить адресное пространство и использовать 64-разрядные примитивные типы данных [9].
  + Третья версия языка имела одно большое нововведение – LINQ, проекта Microsoft по добавлению синтаксиса языка запросов, напоминающего SQL, в языки платформы Microsoft .NET.
  + Ключевые слова select, from, where, позволяющие делать запросы из SQL, XML, коллекций и т. п.
  + Инициализацию объекта вместе с его свойствами.
  + Деревья выражений – лямбда-выражения теперь могут быть представлены в виде структуры данных, доступной для обхода во время выполнения, тем самым позволяя транслировать строго типизиро­ванные C#-выра­жения в другие домены (например, выражения SQL).
  + Вывод типов локальной переменной.
  + Безымянные типы данных.
  + Методы-расширения – добавление метода в существующий класс с помощью ключевого слова this при первом параметре статической функции.

C# 3.0 совместим с C# 2.0 по генерируемому MSIL-коду; улучшения в языке чисто синтаксические и реализуются на этапе компиляции [9].

Visual Basic.NET 10.0 и C# 4.0 были выпущены в апреле 2010 года, одновременно с выпуском Visual Studio 2010. Новые возможности в версии 4.0:

* + возможность использования позднего связывания;
  + именованные и опциональные параметры;
  + новые возможности COM-взаимодействия;
  + ковариантность и контрвариантность интерфейсов и делегатов.
  + контракты в коде [9].

В C# 5.0 было немного нововведений, но они носят большую практическую ценность. В новой версии появилась упрощенная поддержка выполнения асинхронных функций с помощью двух новых слов – *async* и *await*. Ключевым словом *async* помечаются методы и лямбда-выражения, которые внутри содержат ожидание выполнения асинхронных операций с помощью оператора *await*, который отвечает за преобразования кода метода во время компиляции.

## 2.3 ASP.NET MVC 5 Framework

**ASP.NET MVC** – это инфраструктура для разработки веб-приложений от Microsoft, которая сочетает в себе эффективность и аккуратность архитектуры «модель-представление-контроллер»*,* новейшие идеи и приемы гибкой разработки, а также все лучшее из существующей платформы ASP.NET [10]. Она представляет собой полномасштабную альтернативу традиционной технологии ASP.NET Web Forms, предоставляя преимущества для всех проектов веб-разработки, кроме самых тривиальных.

На момент своего появления в 2002 г. платформа ASP.NET оказалась огромным шагом вперед. На тот момент стек технологий от компании Microsoft был представлен следующим образом:

* 1. ASP.NET WebForms. Набор компонентов пользовательского интерфейса (страниц, кнопок и так далее), объектно-ориентированная программная модель графического пользовательского интерфейса с возможность сохранения состояния.
  2. ASP.NET. Представляет собой способ размещения веб-приложений, созданных с помощью платформы .NET Framework, позволяющий взаимодействовать с запросами и ответами посредством протокола HTTP.
  3. .NET Framework, многоязычная платформа управляемого кода.

В Web Forms разработчики из Microsoft попытались сокрыть как протокол HTTP (с присущим ему отсутствием состояния), так и язык HTML за счет моделирования пользовательского интерфейса в виде иерархии объектов, представляющих серверные элементы управления. Каждый элемент управления отслеживает собственное состояние между запросами, по мере необходимости визуализируя себя в виде HTML-разметки, и автоматически соединяя события клиентской стороны с соответствующим кодом их обработки на стороне сервера [10].

Создание технологии Web Forms было большим шагов в развитии веб-разработки, однако она обладала следующими недостатками:

1. Ресурсоемкость возможности сохранения состояния. Действительный механизм поддержки состояния между запросами требует передачи огромных блоков данных между клиентом и сервером. Объем этих данных может достигать сотен килобайт даже в самых скромных веб-приложениях. Эти данные предаются туда и обратно с каждым запросом, приводя к замедлению реакции и увеличивая требования к ширине полосы пропускания сервера.
2. Жизненный цикл страницы. Механизм подключения событий клиентской стороны к коду обработчиков событий на стороне сервера, являющийся частью жизненного цикла страницы, может быть чрезвычайно сложным и хрупким.
3. Ограниченный контроль над HTML-разметкой. В ранних версиях Web Forms выходная HTML-разметка не соответствовала веб-стандартам или неэффективно использовала каскадные таблицы стилей, а серверные элементы управления генерировали непредсказуемые и сложные значения для идентификаторов, к которым было трудно получать доступ с помощью JavaScript. Эти проблемы были значительно смягчены в последних выпусках Web Forms, но получение ожидаемой HTML-разметки по-прежнему может оказаться затруднительным.
4. Низкая тестируемость. Проектировщики Web Forms даже не предполагали, что автоматизированное тестирование станет неотъемлемой частью разработки программного обеспечения. В результате этого, построенная ими тесно связанная архитектура совершенно не была приспособлена для модульного тестирования.

В октябре 2007 г. в Microsoft анонсировали новую платформу веб-разработки – MVC [11], построенную на основе ASP.NET и спроектированную в ответ на развитие новейших веб-технологий, а также в качестве реакции на критику в адрес Web Forms.

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 [11]. 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.

Инфраструктура MVC Framework построена в виде набора независимых компонентов, которые удовлетворяют интерфейсу .NET или созданы на основе абстрактного базового класса. Компоненты, подобные системе маршрутизации, механизму визуализации и фабрике контроллеров, можно легко 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;
* создание подкласса из стандартной реализации с целью корректировки существующего поведения;
* полная замена компонента новой реализацией интерфейса или абстрактного базового класса.

Инфраструктура ASP.NET MVC генерирует ясный и соответствующий стандартам код разметки. Вместо генерации громадного объема трудно поддающейся управлению HTML-разметки инфраструктура MVC Framework стимулирует создание простых и элегантных элементов, оформленных стилями CSS.

Сгенерированные ASP.NET MVC страницы не содержат никаких данных о состоянии страницы, поэтому они меньше типовых страниц ASP.NET Web Forms. 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.

Естественное разнесение 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. Тестируемость касается не только модульного тестирования. Приложения ASP.NET MVC успешно работают также с инструментами тестирования, встроенными в средства автоматизации пользовательского интерфейса. Можно создавать тестовые сценарии, которые имитируют взаимодействие с пользователем, не беспокоясь о том, какие структуры HTML-элементов, классы CSS или идентификаторы будут сгенерированы инфраструктурой, равно как и о неожиданных изменениях структуры.

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

К тому же, технология ASP.NET MVC предоставляет готовые средства, такие как аутентификация, членство, роли, профили и интернационализация, которые могут существенно сократить объем кода, Лежащая в основе платформа ASP.NET предоставляет развитый набор инструментов, на базе которых строятся веб-приложения с помощью MVC Framework.

## 2.4 Язык программирования TypeScript

TypeScript – [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), представленный [Microsoft](https://ru.wikipedia.org/wiki/Microsoft) в [2012 году](https://ru.wikipedia.org/wiki/2012_%D0%B3%D0%BE%D0%B4) и позиционируемый как средство разработки [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5), расширяющее возможности языка [JavaScript](https://ru.wikipedia.org/wiki/JavaScript). Разработчиком языка TypeScript является [Андерс Хейлсберг](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D0%B9%D0%BB%D1%81%D0%B1%D0%B5%D1%80%D0%B3,_%D0%90%D0%BD%D0%B4%D0%B5%D1%80%D1%81) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Anders Hejlsberg) создавший ранее [Turbo Pascal](https://ru.wikipedia.org/wiki/Turbo_Pascal" \o "Turbo Pascal), [Delphi](https://ru.wikipedia.org/wiki/Delphi_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)" \o "Delphi (язык программирования)) и [C#](https://ru.wikipedia.org/wiki/C_Sharp). Спецификации языка открыты и опубликованы в рамках соглашения Open Web Foundation Specification Agreement (OWFa 1.0)[[12]](https://ru.wikipedia.org/wiki/TypeScript#cite_note-OWFa-1.0-7).

TypeScript является обратно совместимым с JavaScript и компилируется в последний. Фактически, после компиляции программу на TypeScript можно выполнять в любом современном браузере или использовать совместно с серверной платформой [Node.js](https://ru.wikipedia.org/wiki/Node.js). Код экспериментального компилятора, транслирующего TypeScript в JavaScript, распространяется под [лицензией Apache](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%86%D0%B5%D0%BD%D0%B7%D0%B8%D1%8F_Apache). Его разработка ведётся в публичном репозитории через сервис [GitHub](https://ru.wikipedia.org/wiki/GitHub" \o "GitHub).

TypeScript отличается от JavaScript возможностью явного определения типов (статическая типизация), поддержкой использования полноценных классов (как в традиционных объектно-ориентированных языках), а также поддержкой подключения модулей. Что призвано повысить скорость разработки, облегчить читаемость, рефакторинг и повторное использования кода, помочь осуществлять поиск ошибок на этапе разработки и компиляции, и, возможно, скорость выполнения программ.

Планируется, что в силу полной обратной совместимости адаптация существующих приложений на новый язык программирования может происходить поэтапно, путём постепенного определения типов.

На момент релиза представлены файлы для восприятия расширенного синтаксиса TypeScript для [Vim](https://ru.wikipedia.org/wiki/Vim" \o "Vim) и [Emacs](https://ru.wikipedia.org/wiki/Emacs" \o "Emacs), а также плагин для [Microsoft Visual Studio](https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio).

Одновременно с выходом спецификации разработчики подготовили файлы с декларациями статических типов для некоторых популярных [JavaScript-библиотек](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_JavaScript), среди которых [jQuery](https://ru.wikipedia.org/wiki/JQuery" \o "JQuery).

## 2.5 AngularJS

AngularJS — [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)-[фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) с [открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Предназначен для разработки [одностраничных приложений](https://ru.wikipedia.org/wiki/Single_Page_Application). Его цель — расширение браузерных приложений на основе [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller) шаблона, а также упрощение тестирования и разработки[13].

Фреймворк работает с [HTML](https://ru.wikipedia.org/wiki/HTML), содержащим дополнительные пользовательские атрибуты, которые описываются директивами, и связывает ввод или вывод области страницы с моделью, представляющей собой обычные переменные JavaScript. Значения этих переменных задаются вручную или извлекаются из статических или динамических JSON-данных.

AngularJS позиционирует себя как MVW-фреймворк, улучшающий HTML. Он собрал концепции из разных языков программирования, как JavaScript, так и серверных, и также делает из HTML нечто динамическое. Мы получаем подход к разработке приложений, основанный на данных. Нет нужды обновлять Модель, DOM или делать какие-то другие затратные по времени операции, например, исправлять ошибки браузеров. Мы концентрируемся на данных, данные же заботятся об HTML, а мы просто занимаемся программированием приложения.

Позиция AngularJS по работе с данными и другими инженерными концепциями отличается от таких фреймворков, как Backbone.js and Ember.js. Мы довольствуемся уже известным нам HTML, а Angular самостоятельно его улучшает. Angular обновляет DOM при любых изменениях Модели, которая живёт себе в чистых Объектах JavaScript с целью связи с данными. Когда обновляется Модель, Angular обновляет Объекты, которые содержат актуальную информацию о состоянии приложения.

В Angular существует разделение обязанностей и динамический HTML. А это значит, что наши данные живут в Модели, наш HTML живёт в виде маленького шаблона, который будет преобразован в Вид, а Контроллер мы используем для соединения двух этих понятий, обеспечивая поддержку изменений Модели и Вида. То есть, навигация может выводиться динамически, создаваясь из одного элемента списка, и автоматически повторяться для каждого пункта из Модели.

Разница между MVC и MVVM в том, что MVVM специально предназначен для разработки интерфейсов. Вид состоит из слоя презентации, ВидМодель содержит логику презентации, а Модель содержит бизнес-логику и данные. MVVM была разработана для облегчения двусторонней связи данных, на чём и процветают фреймворки типа AngularJS. Мы сосредоточимся на пути MVVM, так как в последние годы Angular склоняется именно туда.

Двусторонняя связь данных – очень простая концепция, предоставляющая синхронизацию между слоями Модели и Вида. Изменения Модели передаются в Вид, а изменения Вида автоматически отражаются в Модели. Таким образом, Модель становится актуальным источником данных о состоянии приложения.

Angular использует простые Объекты JavaScript для синхронизации Модели и Вида, в результате чего обновлять любой из них легко и приятно. Angular преобразовывает данные в JSON и лучше всего общается методом REST. При помощи такого подхода проще строить фронтенд-приложения, потому что всё состояние приложения хранится в браузере, а не передаётся с сервера по кусочкам, и нет опасения, что состояние будет испорчено или потеряно.

Связываем мы эти значения через выражения Angular, которые доступны в виде управляющих шаблонов. Также мы можем связывать Модели через атрибут под названием ng-model. Angular использует свои атрибуты для разных API, которые обращаются к ядру Angular.

# 3 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ СИСТЕМЫ

## 3.1 Обзор платформы BBWT

Платформа BBWT (Blueberry Web Template) представляет собой гибкую и мощную платформу, предназначенную для разработки решений для корпоративных сайтов, нацеленных на реализации различных логистических задач. Основным плюсом данной платформы является использование популярных решение для проектирования сайтов, ввиду чего данная платформа крайне проста в освоении и не требует больших затрат времени для начала работы с ней.

BBWT впервые была разработана британской компанией Blueberry Software в 2008 году, и успешно просуществовала на протяжении 6 лет. За этот период на ней было реализовано большое 30 корпоративных сайтов, но технологии не стоят на месте и сейчас платформа находиться в стадии “перерождения”.

Многолетний опыт помог определить недостатки данной платформы, и выработать новые функциональные требования к платформе. При реализации новой версии были приложены все усилия для исправления недостатков и внедрению новых функций. Одним из важных нововведений является перенос клиентской части на новый мощным фронтэнд фреймворком AngularJS, который позволил многократно ускорить работу всей платформы и упростить реализацию фронтэнд задач.

Платформа BBWT в своей стандартно реализации предоставляет следующий набор решений:

* архитектуру приложения. Архитектура построена с учетом базовых нужд логистических веб приложений, основными концепциями проектирования архитектуры были простота расширения приложения, а также возможность изменения уже существующие модули.
* сайт. Готовый сайт с адаптивной табличной версткой, который можно легко меняет под нужды и желания заказчик.
* виджеты. Набор готовых виджетов для работы с различными типами данных. Все виджеты настроены для эффективной работы в рамках платформы, и имеют примеры типовых использований.
* административный модуль. Мощная административная сторона с расширенной системой ролей, групп и прав доступа. Административная сторона имеет огромную важность для корпоративных сайтов ввиду повышенной необходимости зашиты данных и грамотном управление персоналом.

Как отмечалось раннее, при разработке данной платформы важным показателем являлось простота в освоение, но в тоже время гибкость архитектуры и наличие большого набора возможностей. Для этой цели основой для платформы был выбран .NET Framework, популярная платформа разработки от компании Microsoft, который предоставляет большой стек технологий. Данный фреймворк позволил упростить разработку и вывести проект на новый уровень разработки.

В BBWT используются следующие технологии из платформы .NET Framework:

* Entity Framework – объектно-ориентированная технология доступа к данным, предоставляет возможность взаимодействия с объектами как посредством [LINQ](https://ru.wikipedia.org/wiki/LINQ) в виде LINQ to Entities, так и с использованием Entity SQL;
* .NET MVC Framework – [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) для создания [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5), который реализует [шаблон](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [Model-view-controller](https://ru.wikipedia.org/wiki/Model-view-controller" \o "Model-view-controller). Платформа ASP.NET MVC представляет собой альтернативу схеме веб-форм ASP.NET при создании веб-приложений.

Для реализаций различных фронтэнд задач в BBWT используются следующие технологии:

* TypeScript – [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), представленный [Microsoft](https://ru.wikipedia.org/wiki/Microsoft) в [2012 году](https://ru.wikipedia.org/wiki/2012_%D0%B3%D0%BE%D0%B4) и позиционируемый как средство разработки [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5), расширяющее возможности [JavaScript](https://ru.wikipedia.org/wiki/JavaScript);
* AngularJS – [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)-[фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) с [открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Предназначен для разработки [одностраничных приложений](https://ru.wikipedia.org/wiki/Single_Page_Application). Его цель –расширение браузерных приложений на основе [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller) шаблона, а также упрощение тестирования и разработки;
* Bootstrap – свободный набор инструментов для создания [сайтов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B0%D0%B9%D1%82) и [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5). Включает в себя [HTML](https://ru.wikipedia.org/wiki/HTML) и [CSS](https://ru.wikipedia.org/wiki/CSS) шаблоны оформления для [типографии](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0), веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейсов, учитывая [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) расширения;
* Kendo UI – клиентский jQuery фреймворк предоставляющий большой набор виджетов (таблицы, графики/диаграммы, выпадающий списки т.д.), в основе которого лежит jQuery. В данном фреймворке максимально используются такие технологии, как HTML5 и CSS3 (при этом обеспечивается поддержка достаточно старых браузеров за счет graceful degradation).

## 3.2 Обзор архитектуры приложения

Создание архитектуры приложения – это процесс формирования структурированного решения, отвечающего всем техническим и операционным требованиям и обеспечивающего оптимальные общие атрибуты качества, такие как производительность, безопасность и управляемость. Он включает принятие ряда решений на основании широкого диапазона факторов. Каждое из этих решений может иметь существенное влияние на качество, производительность, удобство обслуживания и общий успех приложения.

Архитектура приложения должна объединять бизнес-требования и технические требования через понимание вариантов использования с последующим нахождением путей их реализации в ПО. Цель архитектуры – выявить требования, оказывающие влияние на структуру приложения. Хорошая архитектура снижает бизнес-риски, связанные с созданием технического решения. Хорошая структура обладает значительной гибкостью, чтобы справляться с естественным развитием технологий, как в области оборудования и ПО, так и пользовательских сценариев, и требований. Архитектор должен учитывать общий эффект от принимаемых проектных решений, обязательно присутствующие компромиссы между атрибутами качества (такими как производительность и безопасность) и компромиссы, необходимые для выполнения пользовательских, системных и бизнес-требований.

Необходимо помнить, что архитектура должна:

* раскрывать структуру системы, но скрывать детали реализации;
* реализовывать все варианты использования и сценарии;
* по возможности отвечать всем требованиям различных заинтересованных сторон;
* выполнять требования, как по функциональности, так и по качеству.

Архитектура ПО часто описывается как организация или структура системы, где система представляет набор компонентов, выполняющих определенную функцию или набор функций. Иначе говоря, основное назначение архитектуры – организация компонентов с целью обеспечения определенной функциональности. Такую организацию функциональности часто называют группировкой компонентов по функциональным областям. На рис. 3.1 представлена типовая архитектура приложения, компоненты которого сгруппированы по функциональным областям.

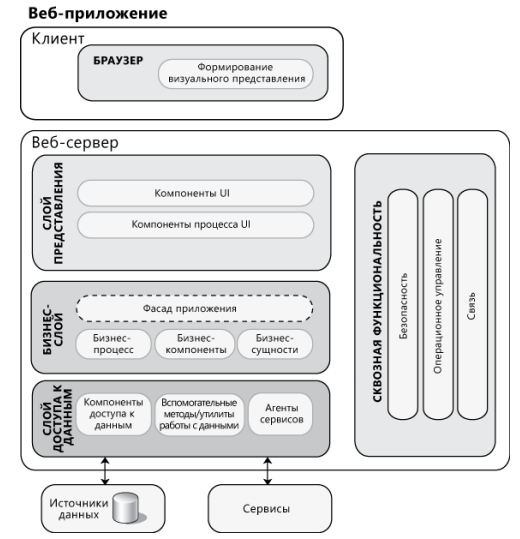


Рисунок 3.1 –Типовая архитектура ПО.

Дипломный проект состоит из четырех слоев:

* Слой представления;
* Серверный слой;
* Слой доступа к данным;
* Слой сервисов.

Каждый модуль реализует определенные функции и решает свои задачи. Все модули реализованы на языке C# c применением .NET Framework 4.5.

### 3.2.1 Слой представления

Слой представления приложения обеспечивает отображение страниц и взаимодействие с пользователем. Слой представления – это часть приложения, которую видит пользователь и с которой он взаимодействует, поэтому этот слой должен удовлетворять многим требованиям. К ним относятся общие факторы, такие как удобство использования, дизайн и возможность взаимодействия с пользователем. Неудовлетворительное взаимодействие с пользователем будет иметь чрезвычайно негативные последствия для приложения, удачного во всех остальных отношениях. При проектировании приложения важно с самого начала обеспечить привлекательный и интуитивный интерфейс, поскольку на взаимодействие с пользователем оказывают влияние многие аспекты архитектуры приложения.

Слой представления Веб-приложения состоит из серверных компонентов и клиентских компонентов. Как правило, вся логика представления располагается в серверных компонентах, а клиентские компоненты только отображают HTML, однако для удобство пользователя в проект было добавлено взаимодействие с фреймворком AngularJS. Эта клиентская технологии, позволяет выполнить логику на клиенте. Это делается для улучшения взаимодействия с пользователем.

Работа веб модуля начинается с загрузки контекста AngularJS из файла app.ts для старта фреймворка AngularJS.

AngularJS – [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) с [открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Предназначен для разработки [одностраничных приложений](https://ru.wikipedia.org/wiki/Single_Page_Application). Его цель - расширение браузерных приложений на основе [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller) шаблона, а также упрощение тестирования и разработки.

Фреймворк работает с [HTML](https://ru.wikipedia.org/wiki/HTML), содержащим дополнительные пользовательские атрибуты, которые описываются директивами, и связывает ввод или вывод области страницы с моделью, представляющей собой обычные переменные JavaScript. Значения этих переменных задаются вручную или извлекаются из статических или динамических JSON - данных.

При разработке системы был выбран подход одностраничных приложений. Архитектура реализации одностраничных приложений представлена на рисунке 3.2.



Рисунок 3.2 – Архитектура реализации одностраничных приложений.

Первым шагом для создания веб приложения будет создание файла app.js, в котором будет описан контекст приложения, и файла index.html в который будет добавлен вызов этого модуля при помощи директивы ng-app.

В дальнейшем для добавления страницы в проект нужно добавить её путь и название в контекст, а также контроллер, который будет соответствовать этой странице.

### 3.2.2 Серверный слой

При проектировании серверного слоя приложения необходимо продумать реализацию логики и длительных рабочих процессов. Использование выделенного слоя, который будет реализовывать логику и рабочие процессы, может упростить обслуживание и улучшить тестируемость приложения, а также позволит централизовать и повторно использовать общие функции логики. Основным компонентом данного слоя является API по которому происходит общение между клиентом и сервером.

API определяет функциональность, которую предоставляет программа, при этом API позволяет абстрагироваться от того, как именно эта функциональность реализована. Если программу рассматривать как [чёрный ящик](https://ru.wikipedia.org/wiki/%D0%A7%D1%91%D1%80%D0%BD%D1%8B%D0%B9_%D1%8F%D1%89%D0%B8%D0%BA), то API — это множество «ручек», которые доступны пользователю данного ящика и которые он может вертеть и дёргать. Программные компоненты взаимодействуют друг с другом посредством API. При этом обычно компоненты образуют иерархию — высокоуровневые компоненты используют API низкоуровневых компонентов, а те, в свою очередь, используют API ещё более низкоуровневых компонентов. По такому принципу построены [протоколы передачи данных](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) по Интернету.

В веб-разработке, как правило, определенный набор HTTP-запросов, а также определение структуры HTTP-ответов, для выражения которых используют XML или [JSON](https://ru.wikipedia.org/wiki/JSON) форматы. WebAPI является практически синонимом для веб-службы, хотя в последнее время за счет тенденции [Web 2.0](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1_2.0" \o "Веб 2.0) осуществлен переход от SOAP к REST типу коммуникации. Веб-интерфейсы, обеспечивающие сочетание нескольких сервисов в новых приложениях, известны как гибридные.

Классы, реализующие логику WebAPI, называться контролерами и располагаться в папке Controllers. Данные классы содержат в себе логику методов WebAPI, все данные методы поддерживают архитектуру REST, а для взаимодействия с ними используются методы HTTP:

* GET;
* POST;
* PUT;
* DELETE.

Поскольку в WebAPI методы контроллера не являются прямыми ресурсами и сопоставляются с методами HTTP, то и весь механизм маршрутизации действует не как при определении обычных маршрутов. Все определения маршрутов для WebAPI находятся в файле WebApiConfig.cs (в папке App\_Start).

Проект имеет следующую структуру:

* каталог App\_Data хранятся все необходимые файлы и ресурсы. После развертывания приложения только непосредственно приложение может работать с этой папкой, доступ же простых пользователей в эту папку запрещен;
* каталог App\_Start включает весю функциональность конфигурации приложения, который в предыдущих версиях содержался в файле Global.asax, а теперь перенесен в набор статичных классов, вызываемых в Global.asax. Эти статичные классы содержат некоторую логику инициализации приложения, выполняющуюся при запуске;
* Web.config файл конфигурации приложения, который находится в корневой папке приложения;
* каталог Content содержит некоторые вспомогательные файлы, которые не включают код на C# или JavaScript, и которые развертываются вместе с приложением. В частности, здесь содержаться файлы стилей CSS. В этой папке есть файл Site.css, который содержит стили приложения, а также папка со стилями фреймворка KendoUI, включающую стили CSS и изображения;
* каталог Controllers содержит контроллеры - классы, отвечающие за работу WebAPI. По умолчанию здесь находятся два контроллера - HomeController и AccountController;
* каталог OData содержит контроллеры - классы, отвечающие за работу OData API;
* каталоги Images и Scripts содержат соответственно изображения и скрипты на JavaScript, используемые в приложении. По умолчанию эти папки уже содержат файлы, в частности, в папку Scripts уже помещены файлы библиотеки jQuery;
* каталог Models содержит модели, используемые приложением. По умолчанию здесь определена одна модель - AccountModel, которая представляет отдельную учетную запись;
* каталог Views содержит представления. Представления группированы по папкам, каждая из которых соответствует одному контроллеру. После получения и обработки запроса контроллер отправляет одно из этих представлений, заполненных некоторыми данными клиенту. Кроме того, имеется папка общих для контроллеров представлений.

### 3.2.3 Слой доступа к данным

Слой доступа к данным предоставляет функционал по работе с данными хранящимися в базе данных. Слой доступа к данным должен отвечать требованиям приложения, работать эффективно и безопасно и обеспечивать простоту обслуживания и расширения в случае изменения бизнес-требований. Компонентами слоя доступа к данным являются компоненты, обеспечивающие функциональность доступа к данным, размещаемым в системе, и компоненты агентов сервисов, обеспечивающие функциональность доступа к данным, которые предоставляются другими серверными системами через Веб-сервисы. Кроме того, слой доступа к данным также может включать компоненты, обеспечивающие вспомогательные функции и утилиты.

При реализации слоя доступа к данным была выбрана технология ADO.NET Entity Framework. Платформа ADO.NET Entity Framework позволяет разработчикам создавать приложения для доступа к данным, работающие с концептуальной моделью приложения, а не напрямую с реляционной схемой хранения. Цель состоит в уменьшении объема кода и снижении затрат на сопровождение приложений, ориентированных на обработку данных.  Благодаря ее возможностям был решен ряд проблем, связанных с реализацией слоя доступа к данным.

Первое что необходимо отметить это простую расширяемость данной платформы, она позволяет использовать любы популярные инфраструктуры доступа к данным, включая ADO.NET, LINQ to SQL, ADO.NET Entity Framework и NHibernate. А также использовать различные реляционные хранилища, не входящим в семейство продуктов Microsoft SQL Server.

Вторым плюсом данной платформы является возможность использовать несколько подходов для создания баз данных:

* Database First, данный подход позволяет по готовой базе данных сгенерировать модель с которой в дальнейшем происходит работа;
* Model First, он позволял создать вручную с помощью визуального редактора модель, и по ней создать базу данных;
* Code First, его суть заключается в том, что сначала пишется код модели на С#, а затем по нему генерируется база данных.

Данные подходы позволяют работать как с уже существующей базой данных, так и создавать ее «налету», при написании кода. На платформе BBWT активно используется подход Code First. Классы модели по которым генерируются таблицы в базе данных находиться в сборке BBWT.Data.

Кроме приведенных плюсов платформа ADO.NET Entity Framework содержит в себе множество полезных компонентом для упрощения работы с базами данных. Среде них следует отметить LINQ to Entity. Он позволяет выполнять запросы через строго типизированные сущности или запрашивать реляционные данные, используя синтаксис LINQ. Что сильно упрощает построение запросов к базе данных.

Основная логика работы с базой данных реализована в классе DataContext, который располагается в сборке BBWT.Domain. Данный класс предоставляет надо методов предназначенный для доступа к данным хранящимся в базе данных. Как было сказано выше ADO.NET Entity Framework предоставляет возможность работы с разными инфраструктурами доступа к данным. В виду этого на данном уровне был реализован паттерн репозиторий, который позволяет разделить ответственность за постоянство бизнес-модели приложения. Благодаря ему мы изолируем логику непосредственной работы с базой данных от остального приложения. Это позволяет изменять DataContext под наши требования, без ущерба для остальной логики приложения. IDataContext – интерфейс отвечающий за описание логики доступа к данным располагается в сборке BBWT.Domain. А класс DataContext является его реализацией.

Также на этом уровне реализована функциональность, предназначенный для миграции данных. Этот модуль отвечает за сохранение необходимых данных при переносе приложения на новую базу данных. Обычно он сохраняет данные для различных сущностей перечислений, однако он может хранить и более сложные сущности. Логика его работы заключаться в том, что при переходе на новую базу данных он заполняет ее прописанными в нем данными, необходимыми для полного функционирования системы. Следует отметить, что речь не идет о сохранение всех данных, которые были в базе данных, а только о необходимом минимуме. Логика данного модуля реализована в классе Configuration, который находиться в сборке BBWT.Domain.

### 3.2.4 Слой сервисов

Обычным подходом при создании приложения, которое должно обеспечивать сервисы для других приложений, а также реализовывать непосредственную поддержку клиентов, является использование слоя сервисов, который предоставляет доступ к функциональности приложения. Слой сервисов обеспечивает альтернативное представление, позволяющее клиентам использовать другой механизм для доступа к приложению. Его еще называют слоем логики.

В данном сценарии пользователи могут выполнить доступ к приложению через слой представления, который обменивается данными с компонентами серверного слоя либо напрямую, либо через фасад приложения в серверном слое, если методы связи требуют композиции функциональности. Между тем, внешние клиенты и другие системы могут выполнять доступ к приложению и использовать его функциональность путем взаимодействия с серверным слоем через интерфейсы сервисов. Это улучшает возможности приложения для поддержки множества типов клиентов, способствует повторному использованию и более высокому уровню композиции функциональности в приложениях. В некоторых случаях слой представления может взаимодействовать с серверным слоем через слой сервисов. Но это не является обязательным условием. Если физически слой представления и бизнес-слой располагаются на одном уровне, они могут взаимодействовать напрямую.

 Проектирование отдельного слоя сервисов нужно, если планируется развертывать серверный слой удаленно или предполагается предоставлять серверную логику через Веб-сервис. Отсутствие конкретизации деталей клиентов, которые будут использовать сервисы, обеспечит максимальные возможности их повторного использования. Если серверный слой располагается удаленно, нужно проектировать слабо детализированные методы сервисов, чтобы сократить количество обращений к сети и обеспечить слабое связывание. Важно не реализовывать правила в интерфейсе сервиса, это может создать ненужные зависимости между компонентами и клиентами и усложнить задачу по поддержанию стабильности интерфейса, важно не реализовывать правила в интерфейсе сервиса, это может создать ненужные зависимости между компонентами и клиентами и усложнить задачу по поддержанию стабильности интерфейса. Для выполнения требования по обеспечению возможности взаимодействия с клиентом были выбраны соответствующие протоколы и механизмы транспортировки, такие как REST и ODATA.

На платформе BBWT сервисы реализованы с помощью паттерна репозиторий, который, как было отмечено в предыдущем пункте, позволяет изолировать логику сервисов от остальной логики программы. Это позволяет гарантировать, что изменения на уровне сервисов не повлечет изменения логики других уровней. Классы, отвечающие за данную логику, располагаются в сборке BBWT.Service.

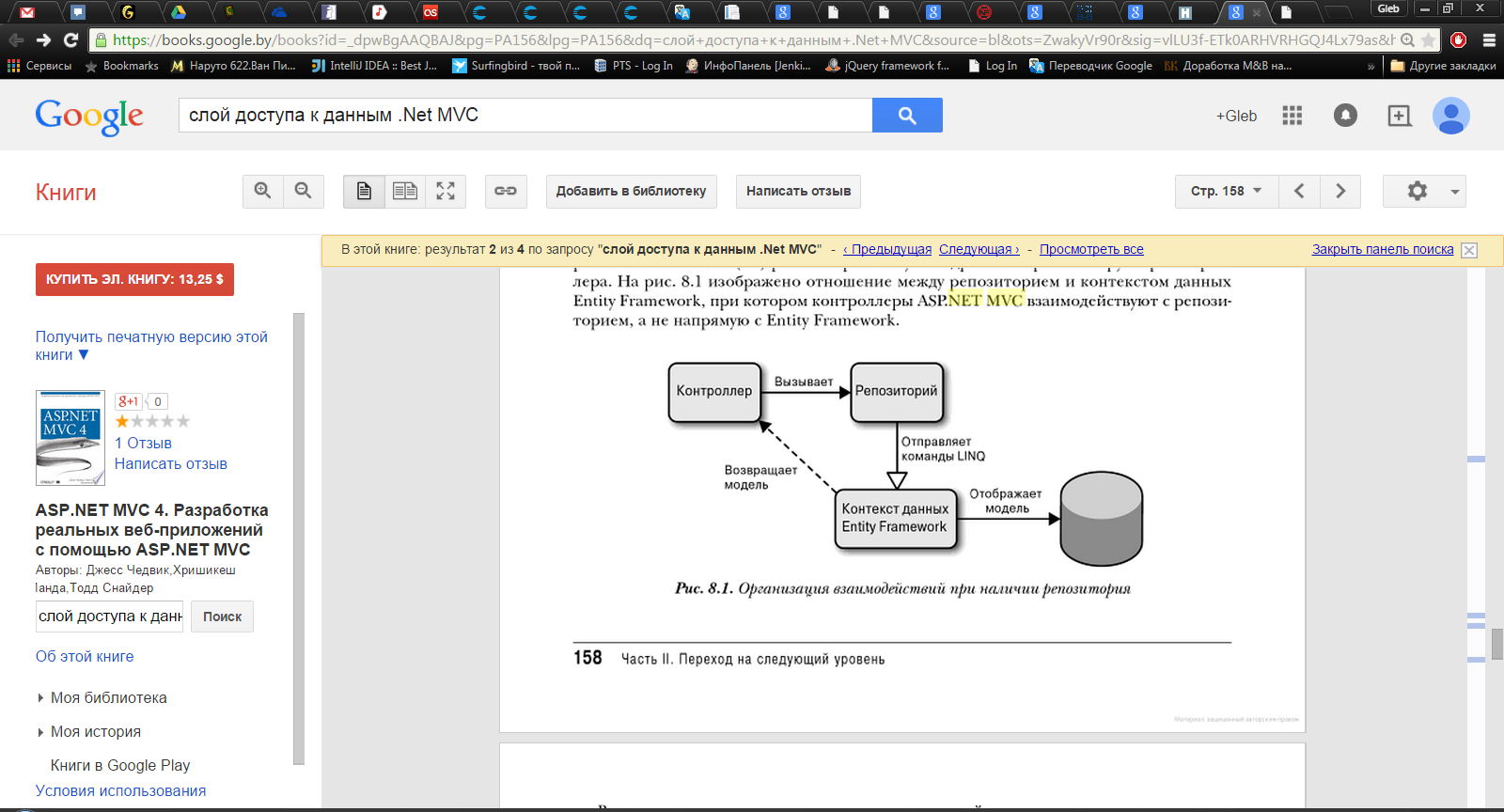


Рисунок 3.3 – Организация работы при наличии паттерна Репозиторий.

На данном этапе также необходимо задумать о данных, которые будут возвращать сервисы. Передача сущностей, полученных напрямую из базы данных не является лучшим решением, ввиду того что данные сущности часто содержать много лишней информации, и как правиле совершенно ненужной для клиента. Во избежание ненужных затрат на передачу ненужных данным на платформе BBWT используется еще один паттерн проектирования Data Transfer Object. Данный шаблон позволяет уменьшить количество передаваемых данных за счет хранения в сущностях только данных необходимых для вызова информации. Так же такой подход упрощает сериализации данным для удобной передачи по сети. Сущности такого типа хранят в себе только простые типы данных, а вместо ссылки на другие объекты только их идентификаторы. Как правило, на одну сущность уровня данных создаются две такие сущности. Одна с минимальным необходимым набором данных предназначенная для отображения сущности в списках. Вторая более развернутая для работы со связями между сущностей. Все сущности данного типа описаны в сборке BBWT.DTO.

### 3.3 Пример реализация нового модуля на платформе BBWT

В данном разделе будет поэтапно расписано создание нового модулю на реальном проекте, базирующемся на платформе BBWT. Проект предназначен для учета приборов отвечающих за измерения показаний тепло-обогревательного оборудования, а также учета измерений и составления различных расчетов на их основе. Примером послужит реальная задача по созданию модуля, предназначенного для управления расписаниями проверки приборов в жилом здании. Данный модуль должен отвечать следующим функциональным требованиям:

* необходима возможность просмотра всех расписаний, а также поиск по заданным параметрам;
* необходима возможность создания и редактирования расписаний;
* необходима возможность создания для расписания трех различных графиков работ;
* необходима возможность управления потребителями, отображение и возможность изменения их статусов.

В начале создания нового модуля необходимо определить сущности, с которыми он будет работать. Данные сущности будут представлять основные объекты, участвующие в работе модулю. Можно выделить два типа сущностей это сущности-объекты и сущности-перечисления. Первые описывают устройство объекта, являясь своего рода чертежом для реального объекта предметной области. Вторые представляют собой перечисления разного характера будь то статусы или типы того или иного объекта.

Сущности необходимые для нового модулю:

* Расписание (Schedule), основная сущность, с которой будет происходить работа. Непосредственно связана с объектом, для которого и составляется данное расписание. Расписание создаётся вместе с новым объектом, а непосредственное создание расписания отсутствует;
* План графиков работ (OperatingSchedule), сущность для представления планов графиков работ, всего в Расписание может быть от 1-го до 3-ех планов;
* График работ (OperationWork), сущность представляет график работ и напрямую ссылается на ответственного за выполнение данной работы;
* Расписание работ (ScheduleWork), расписание работ;
* Статус справки при расписание (ReceiptsStatus), сущность-перечисление, всего два статуса: Редактируется, Готова;
* Статус квитанций при расписание (InquiriesStatus), сущность-перечисление, всего три статуса: Редактируются, Готовы, Распечатаны;
* Тип графика работ (OperatingScheduleType), сущность-перечисление, всего может быть три типа: Основной, 1-ый дополнительный, 2-ой дополнительный;
* Статус графика работ (OperationScheduleStatus), сущность-перечисление, может быть всего пять статусов: Отсутствует, Планируется, Согласованный, Утвержденный, Прошедший;

Кроме этого в создании нового модулю принимают участи две уже существующие сущности:

* Объект (Facility), сущность представляет жилой объект, к которому и задаться расписание проверки приборов;
* Персона (Person), сущность представляет собой различные лица участвующие в работе системы, в данном случае подразумеваются лица ответственные за проведение проверки.

В итоге выработки требований к новым сущностям получается следующая диаграмма классов, представленная на рисунке 3.3.

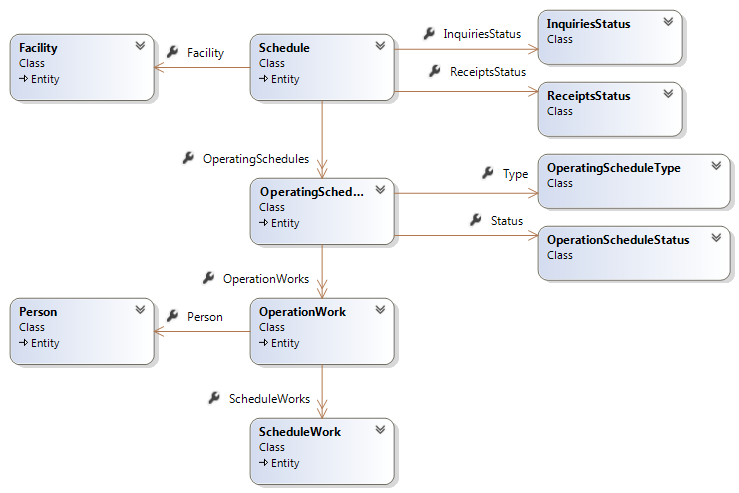


Рисунок 3.4 – Диаграмма классов для реализуемого модуля.

Для выработанных сущностей необходимо создать классы, которые будут их, представляют и создать таблицы в БД для их хранения. Работы на данном этапе проходят в двух сборках:

* BBWT.Data, в нем содержатся классы, описывающие сущности;
* BBWT.Domain, в нем реализуется логика работы с БД.

На платформе BBWT используются поход программирования Code First, предоставленным технологи Entity Framework, данным подход позволяет по созданным классам генерировать модели для хранения в БД. Данный подход сильно облегчает создание новых сущностей и изменения старых. Все что для этого необходимо – создать классы, описывающие сущности, и зарегистрировать в интерфейсе IDataContext, в реализациях которого имплементируется логика работы с базой данных, в данном случае это класс DataContext’а. Все новые классы, описывающие сущности, должны наследоваться от класса Entity, для которого создана логика предназначенная для работы с БД, в частности он содержит поле int Id, необходимое для идентификации сущности. Классы перечисление наследуются от класса Type, он во многом похож на Entity, но логика работы с ним несколько упрошена, так как объекты такого типа не требуют всех возможностей классов сущностей, в частности для их идентификации используется поле byte Code. Очень часто перечисления состоят из строго фиксированного количества вариантов, которые со временем не меняются, в этом случае в классах задается перечень возможных значений, что облегчает дальнейшую работу с ними ввиду уменьшения запросов к БД. Следует отметить, что для классов со статическим количеством вариантов значений необходимо добавить эти значение в Auto Migration. Это позволит не волноваться о потерях перечислений при переносе приложения на реальный сервер.

Код класса-сущности описывающего Расписание:

namespace BBWT.Data

{

public class Schedule : Entity

{

public virtual Facility Facility { get; set; }

public virtual ServiceIsta IstaService { get; set; }

public virtual ReceiptsStatus ReceiptsStatus { get; set;}

public virtual InquiriesStatus InquiriesStatus {get;set;}

public DateTime? Date { get; set; }

public DateTime? ReceiptsDate { get; set; }

public DateTime? DateCommercialAccounting { get; set; }

public string Description { get; set; }

public virtual ICollection<OperatingSchedule>

OperatingSchedules { get; set; }

}

}

Код класса-перечисления, описывающего Статус графика работ:

namespace BBWT.Data

{

public class OperationScheduleStatus: Type

{

public const byte Missing = 1;

public const byte Planed = 2;

public const byte Coordinated = 3;

public const byte Approved = 4;

public const byte Past = 5;

public string Name { get; set; }

}

}

Регистрация классов в DataContext происходит при его инициализации. После этого Entity Framework сам создаст необходимые таблицы и связи в базе данных.

public IDbSet<Schedule> Schedules { get; set; }

public IDbSet<ScheduleWork> ScheduleWorks { get; set; }ы

public IDbSet<WorkContractor> WorkContractors { get; set; }

public IDbSet<ReceiptsStatus> ReceiptsStatuses { get; set; }

public IDbSet<InquiriesStatus> InquiriesStatuses { get; set; }

public IDbSet<OperationWork> OperationWorks { get; set; }

public IDbSet<OperatingSchedule> OperatingSchedules { get; set; }

public IDbSet<OperatingScheduleType> OperatingScheduleTypes

{ get; set; }

public IDbSet<OperationScheduleStatus> OperationScheduleStatuses { get; set; }

Следующий этап разработки нового модуля заключается в создании методов для управления данными классов, которые были созданы на уровне доступа к данным. Данные методы предназначены для создания, удаления и получения данных из БД. Реализации данных методов происходит с помощью шаблона проектирования репозиторий (Repository). Он позволяет описать необходимый перечень методов для работы с данными и не привязываться к конкретным их реализациям. Это очень полезно, ведь благодаря этому при изменении способа работы с БД или вообще использования другой базы данных необходимо будет поменять только логику реализаций методов, а не весь код, который с ними работал.

Реализации данной системы находиться в сборке BBWT.Service, она содержит как интерфейсы, которые расположены в каталоге Interfaces, так и классы реализации, расположеные в Classes. Кроме интерфейсов и классов необходимых для работы с данными, здесь же располагаются реализации логики Сервисного уровня, которая также использует шаблон репозиторий.

Необходимые методы описываются в интерфейсе IIstaService.

IQueryable<Schedule> GetAllSchedules();

Schedule GetScheduleById(int id);

OperatingSchedule GetOperatingScheduleById(int id);

OperationWork GetOperationWorkById(int id);

ScheduleWork GetScheduleWorkById(int id);

void CreateSchedule(Schedule schedule);

void UpdateSchedule(int id, Action<Schedule> updateStrategy);

void DeleteSchedules(int id);

void CreateOperatingSchedule (

OperatingSchedule operatingSchedule);

void UpdateOperatingSchedule(int id,

Action<OperatingSchedule> updateStrategy);

void DeleteOperatingSchedule(int id);

void CreateOperationWork(OperationWork operationWork);

void UpdateOperationWork(int id,

Action<OperationWork> updateStrategy);

void DeleteOperationWork(int id);

void CreateScheduleWork(ScheduleWork scheduleWork);

void UpdateScheduleWork(int id,

Action<ScheduleWork> updateStrategy);

void DeleteScheduleWork(int id);

А их реализации в классе IstaService который является реализацией интерфейса IIstaService.

public IQueryable<Schedule> GetAllSchedules()

{

return this.context.Schedules;

}

public Schedule GetScheduleById(int id)

{

return this.context.Schedules.Find(id);

}

public void CreateSchedule(Schedule schedules)

{

this.context.Schedules.Add(schedules);

this.context.Commit();

}

public void UpdateSchedule(int id,

Action<Schedule> updateStrategy)

{

var db = this.context.Schedules.Find(id);

if (db != null)

{

updateStrategy(db);

this.context.Commit();

}

}

public void DeleteSchedules(int id)

{

var c = this.context.Schedules.Find(id);

if (c == null)

{

return;

}

this.context.Schedules.Remove(c);

this.context.Commit();

}

Новый этап разработки происходит на серверном слое, его реализация распределена между двумя сборками:

* BBWT.DTO, сборка, содержащая сущности, с помощью которых происходит обмен информации между клиентом и сервером;
* BBWT.Web, здесь находиться реализации API и серверной логики.

Первым делом необходимо создать сущности серверного уровня, данные сущности необходимы для обмена информацией между системами в частности между клиентом и сервером. Данные сущности реализованы с помощью паттерна Data Transfer Object (DTO). Данный шаблон позволяет уменьшить количество передаваемых данных за счет хранения в сущностях только данных необходимых для вызова информации. Так же такой подход упрощает сериализации данным для удобной передачи по сети. Сущности такого типа хранят в себе только простые типы данных, а вместо ссылки на другие объекты только их идентификаторы, что позволяет не тянуть за собой много лишних данных. Как правило, на одну сущность уровня данных создаются две сущности DTO. Одна с минимальным необходимым набором данных предназначенная для отображения сущности в списках. Вторая более развернутая для работы со связями между сущностей.

public class ScheduleListDTO

{

public int Id { get; set; }

public int FacilityId { get; set; }

public int IstaServiceId { get; set; }

public DateTime? Date { get; set; }

public string MainWorkStatus { get; set; }

public string AdditionalFirstWorkStatus { get; set; }

public string AdditionalSecondWorkStatus { get; set; }

}

public class ScheduleDTO

{

public int Id { get; set; }

public FacilityDTO Facility { get; set; }

public int IstaServiceId { get; set; }

public byte ReceiptsStatusId { get; set; }

public byte InquiriesStatusId { get; set; }

public DateTime? Date { get; set; }

public DateTime? ReceiptsDate { get; set; }

public string Description { get; set; }

public virtual ICollection<OperatingScheduleDTO>

OperatingSchedules { get; set; }

public virtual ICollection<PersonalAccountScheduleDTO>

PersonalAccounts { get; set; }

}

Следует отметить, что для сущностей, которые на уровне данных наследовались от класса Type, существует единая DTO сущность TypeDTO, это было сделано по той причине, что все сущности данного типа состоят только из двух одинаковых полей, и большая часть работы с ними протекает абсолютно одинаково. И чтобы не создавать множество одинаковых классов в DTO, было принято решение на серверном уровне работать с ними как с одной сущностью.

Посредником между сущностями уровня данных и сущностями серверного уровня выступает AutoMapper, его задача состоит в автоматическом преобразовании объектов в другие. Благодаря использованию LINQ в своей основе AutoMapper является очень удобным и простым инструментом. Конфигурация маппинга объектов задается в MapperConfig который располагается в сборке BBWT.Web, также в нем можно задавать логику, по которой происходит преобразование объекта, в частности указывается какие поля к чему относятся и обрабатываются проверки на пустые поля. Стоит отметить, что AutoMapper позволяет преобразовывать как один объект, так и набор объектов, главное, чтобы данный набор реализовывал интерфейс IQueryable. Благодаря этому еще больше упрощается работа по маппингу объектов.

Mapper.CreateMap<OperationWork, OperationWorkDTO>()

.ForMember(c => c.PersonId, r => r.MapFrom(x => x.Person.Id));

Mapper.CreateMap<OperationWorkDTO, OperationWork>()

.ForMember(c => c.Person, r => r.Ignore())

.ForMember(c => c.ScheduleWorks, r => r.Ignore());

Mapper.CreateMap<OperatingSchedule, OperatingScheduleDTO>()

.ForMember(c => c.StatusId, r => r.MapFrom(x => x.Status == null ? (byte) 0 : x.Status.Code))

.ForMember(c => c.TypeId, r => r.MapFrom(x => x.Type == null ? (byte) 0 : x.Type.Code));

Mapper.CreateMap<OperatingScheduleDTO, OperatingSchedule>()

.ForMember(c => c.Status, r => r.Ignore())

.ForMember(c => c.Type, r => r.Ignore())

.ForMember(c => c.OperationWorks, r => r.Ignore());

Само же преобразование происходит двумя путями. При преобразовании одного объекта вызывать объект AutoMapper и непосредственно указывается, в какой объект следует преобразовать. При преобразовании группы объектов у структуры, реализующей IQueryable, вызывается специальный метод, предназначенный для этих целей.

mapper.Map<ScheduleDTO>(istaService.GetScheduleById(id));

istaService.GetAllOperationScheduleStatuses().Project()

.To<TypeDTO>();

Итак, наша разработка плавно переходит к месту, где встречаются все описанные изменения, а именно API. Под API подразумевается набор методов, через которые происходит общение клиент-сервер. Данный момент очень важен в разработке всех онлайн систем, ведь без грамотного API общение сервером может быть нет только затруднено, а иногда вообще не возможно.

На платформе BBWT используются два протокола API для реализации общения клиент-сервер:

* HTTP для работы с малыми объёмами данных и реализации логики по сохранению и удалению объектов;
* OData для работы с большими коллекциями данных, позволяющий выполнять операции с ресурсами, используя в качестве запросов HTTP-команды, и получать ответы в форматах XML или [JSON](https://ru.wikipedia.org/wiki/JSON).

Как было сказано выше, HTTP в основном используется для работы с единичными объектами, будь то получение объекта по идентификатору, сохранение или его удаление. Также в некоторых случаях данный протокол используется для получения небольших наборов данных, чаще всего это DTO объекты сущностей-перечислений.

[HttpGet]

public ScheduleDTO GetScheduleById(int id)

{

return mapper.Map<ScheduleDTO>(

istaService.GetScheduleById(id));

}

[HttpGet]

public IQueryable<TypeDTO> GetAllOperationScheduleStatuses()

{

return istaService.GetAllOperationScheduleStatuses()

.Project().To<TypeDTO>();

}

[HttpPost]

public void SaveSchedule(ScheduleDTO dto)

{

if (dto.Id == 0)

{

var schedule = mapper.Map<Schedule>(dto);

...

istaService.CreateSchedule(schedule);

}

else

{

istaService.UpdateSchedule(

dto.Id,

schedule =>

{

mapper.Map(dto, schedule);

...

});

}

}

OData в свою очередь используют для работы с большими коллекциями данных, также она реализовывает большой набор возможностей для реализации серверного постраничного просмотра данных и их фильтрации. Это очень удобно, ведь подгружать большие коллекции данным на клиент очень медленно, а фильтрация таких наборов – очень затратная операция. Еще одним плюсом данного протокола являет возможность работы с ним многих клиентских виджетов, но об этом позже.

public class ScheduleODataController : ODataController

{

private IIstaService istaService;

public ScheduleODataController(IIstaService istaService)

{

this.istaService = istaService;

}

public IQueryable<ScheduleListDTO> Get()

{

return this.istaService.GetAllSchedules()

.Project().To<ScheduleListDTO>();

}

}

Все API контролеры регистрируются в специальном WebApiConfig, который расположен в сборке BBWT.Web, он позволяет легко добавлять, и изменять уже существующие методы, по которым происходит общение между клиентом и сервером.

Завершающим этапом разработки нового модулю является создание UI части на клиенте и контролера для его работы. Данная задача сопоставима по сложности со всеми выше перечисленными, и занимает около половины всего времени необходимого для создания новых модулей.

Клиентская сторона платформы BBWT работает на фреймворке AngularJS. Основным достоинством данного фреймворка является возможность делать одностраничные веб приложения. Парадигмой такого подхода является возможность использовать всю функциональность обычного сайта, но без перехода между страницами. То есть, сайт загружается один раз, а контент подгружается частями при необходимости. Это позволяет сильно укорить работу веб приложения.

Для реализации UI для работы с новым модулем необходимо создать контент для отображения, который будет подгружать при работе с расписаниями. Нам понадобиться два разных контента, один для работы со списком всех расписаний, и второй для создания и редактирования выбранного расписания. Для каждого из них необходимо реализовать свой контролер, который будет отвечать за их работу.

Контент, который будет подгружаться при работе со всеми расписания представлен виджетом KendoGrid из библиотеки KendoUI. Данный виджет представляет собой таблицу для отображения и работы с данными. KendoGrid очень гибок и много функционален, а также может работать c API протоколом OData. Кроме этого необходимо реализовать функционал для фильтрации данных, тут опять нам на помощь приходят виджеты от KendoUI. Логика работы KendoGrid’а и фильтрации описана в контроллере ScheduleCtrl. Следует отметить в данном случае идет работа с ScheduleListDTO, который возвращает только необходимые данные для отображения.

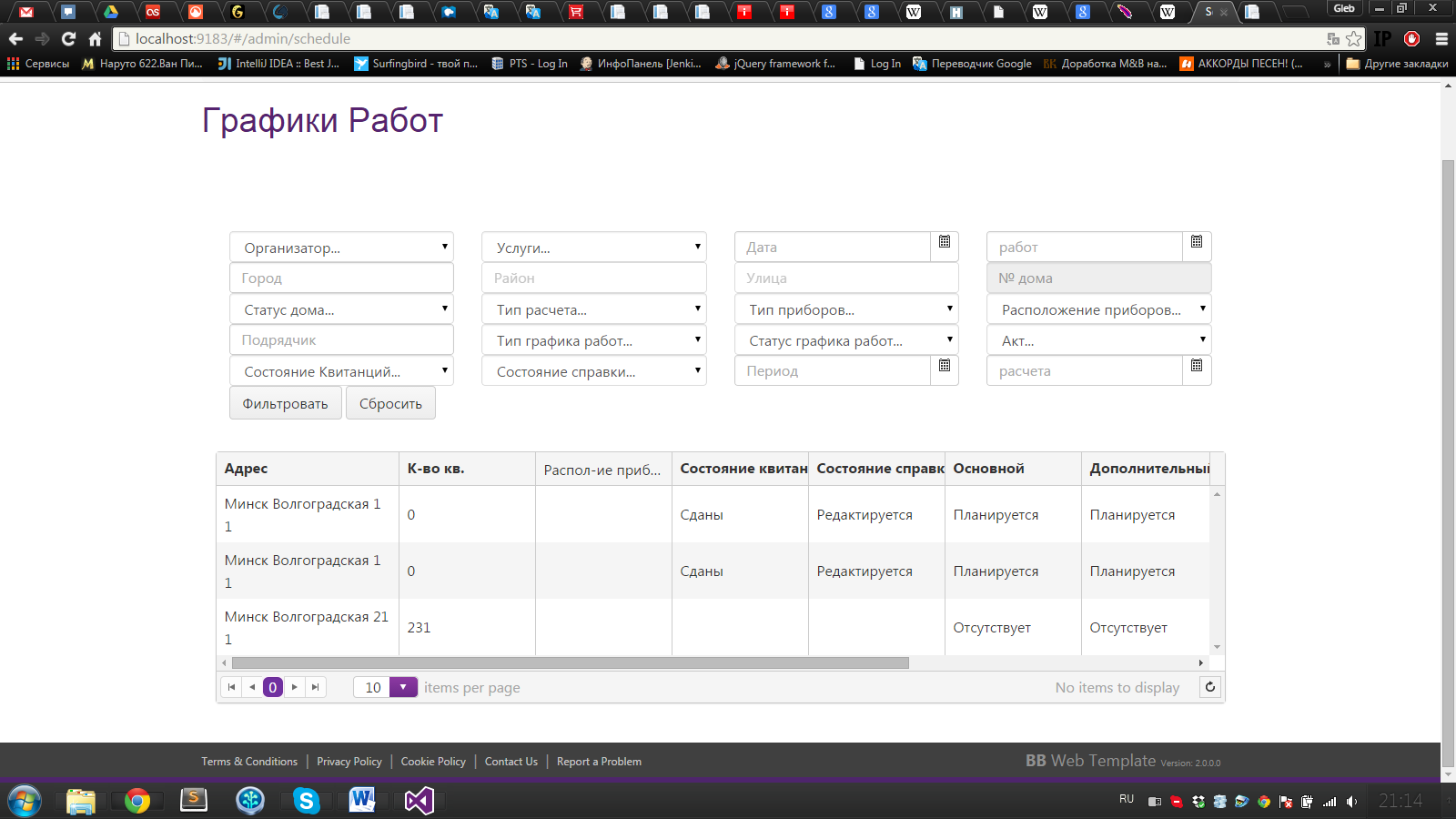


Рисунок 3.5 – Контент отвечающий за работу со всеми расписаниями

this.GridOptions =

{

pageable: { refresh: true, pageSizes: true },

columns: [

{

field: "InquiriesStatus",

title: "Состояние квитанций",

width: "150px"

},

{

field: "MainWorkStatus",

title: "Основной",

width: "150px",

template: "#= (MainWorkStatus == null) ?

'Отсутствует' : MainWorkStatus #"

},

...

{

field: "Id",

title: " ",

width: "150px",

sortable: false,

template: "<a ng-click=\"ScheduleCtrl.ViewDetails(#= Id #)\">Редактировать</a> <a ng-click=\"ScheduleCtrl.Delete(#= Id #)\">Удалить</a>"

}],

sortable: true,

filterable: false

};

Логика, отвечающая за работу фильтрации, предназначенная для формирования запроса к OData, по которому API осуществляет фильтрацию.

this.ApplyFilter = () => {

var filterConditions = [];

if (this.Filter.Organizer) {

filterConditions.push({ field: 'Organizer', operator:

'contains', value: this.Filter.Organizer });

}

if (this.Filter.IstaServices) {

filterConditions.push({ field: 'IstaServices', operator:

'contains', value: this.Filter.IstaServices });

}

if (this.Filter.DateWorkStart) {

filterConditions.push({ field: 'DateWorkStart', operator:

'contains', value: this.Filter.DateWorkStart });

}

if (this.Filter.DateWorkEnd) {

filterConditions.push({ field: 'DateWorkEnd', operator:

'contains', value: this.Filter.DateWorkEnd });

}

if (this.Filter.Settlement) {

filterConditions.push({ field: 'Settlement', operator:

'contains', value: this.Filter.Settlement });

}

if (this.Filter.District) {

filterConditions.push({ field: 'District', operator:

'contains', value: this.Filter.District });

}

...

this.ScheduleDS.filter(filterConditions);

}

Контент для отображения работы непосредственно с выбранным расписанием, представляет собой набор различных виджетов, предоставленных KendoUI. Все они необходимы для работы с теми или иными типами данными, будь то календари для дат, выпадающие списки для перечислений и т.п. Благодаря возможностям AngularJS данный контент можно использовать как для создания нового расписания, так и для редактирования старого. Код для управления контентом располагается в ScheduleDetailsCtrl. И, в отличие от предыдущего, здесь работа ведется с сущностью ScheduleDTO, которая кроме полного набора своих полей предоставляет коллекции зависимых сущностей, OperatingScheduleDTO и OperationWorkDTO.

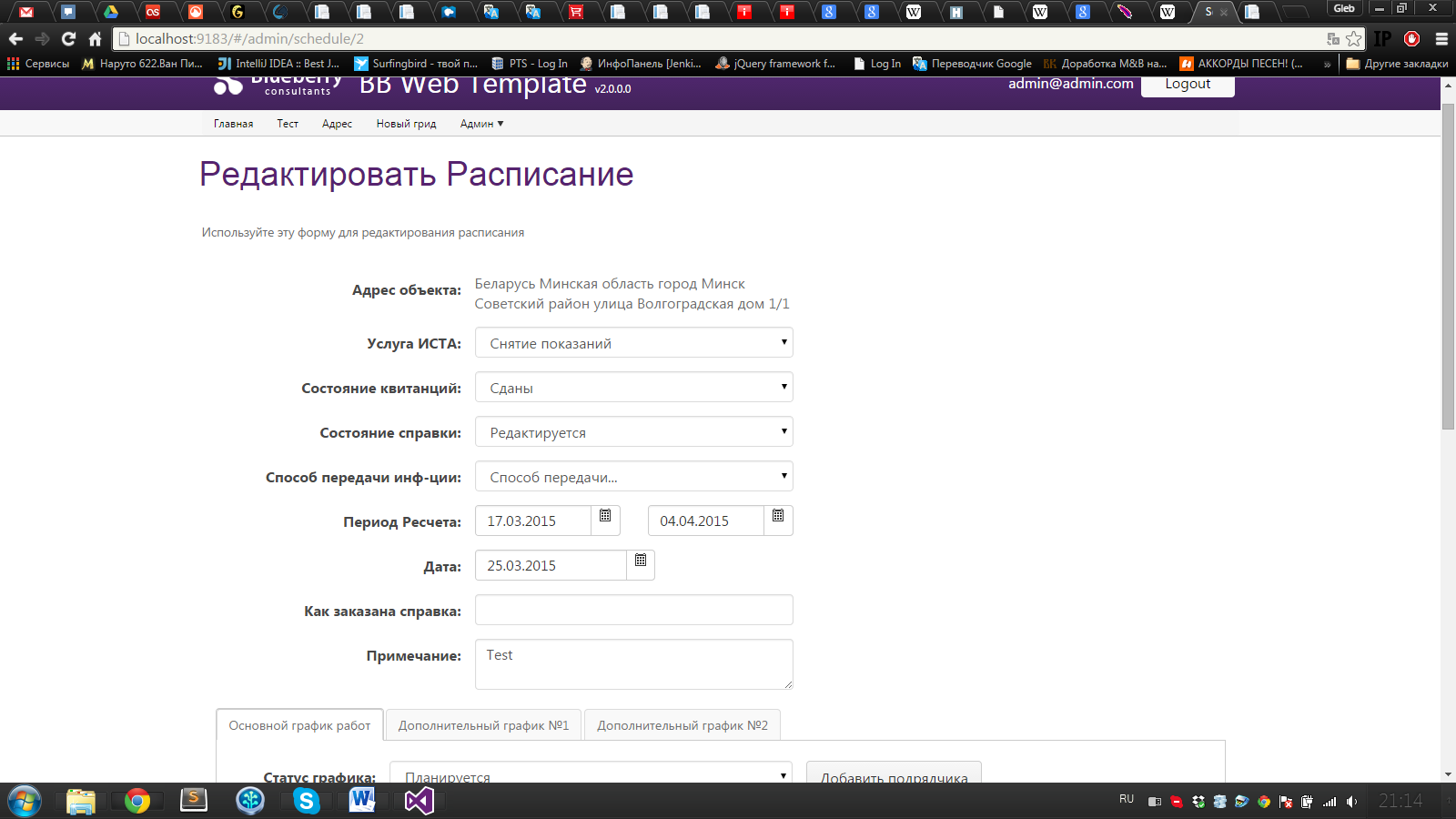
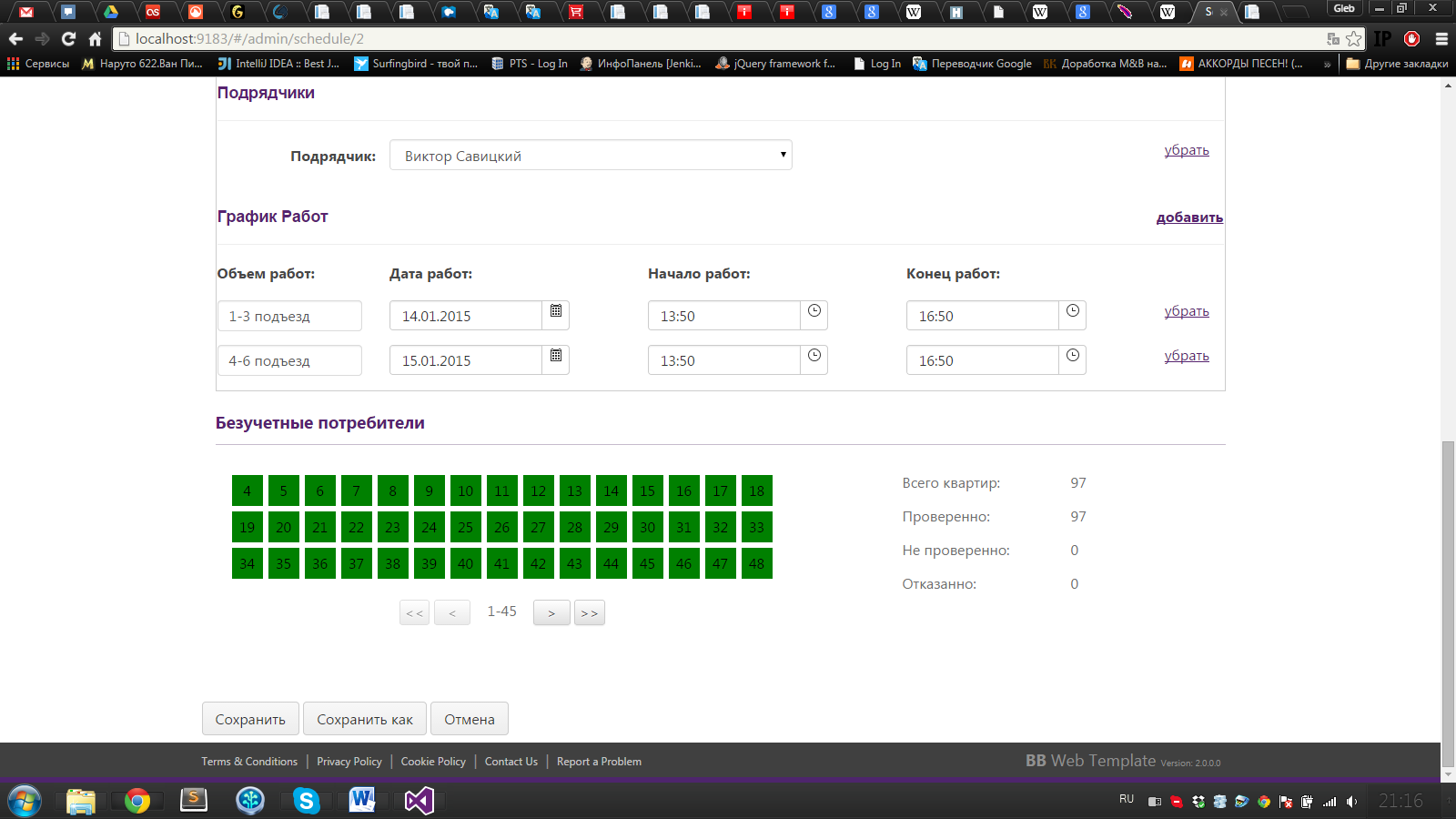
  

Рисунок 3.6 – Контент отвечающий за работу с расписанием

На этом и завершается работа по созданию нового модулю для сайта, базирующегося на платформе BBWT.

По завершению хочется отметить, что платформа BBWT является достаточно простым и удобным инструментом для создания корпоративных сайтов. На данный момент это уже вторая версия платформы и она разительно отличается от своей предшественницы. За счет перехода на новый UI движок, и обновлению стека технологий, платформа стала более производительной и приобрела много полезных и удобных инструментов для работы. Но на этом разработка данной платформы не заканчивается уже сейчас проводятся работы по внедрению автоматизации по создания необходимого кода для новых сущностей. Как было видно из примера реализации, сейчас система страдает из-за того, что приходится постоянно писать и делать одно и тоже, но в скором времени и этот недочет будет исправлен.

Также следует отметить, что не стоят на месте и технологии, которые используются на платформе. Тот же KendoUI, который активно используется на платформе, постоянно обновляет перечень предоставляемых виджетов. А AngularJS сейчас является одной из самых востребованных и активно развивающихся платформ.

Возможно, в будущем платформе BBWT снова потребуется полные редизайн, но на данный момент она является очень мощной и современной платформой для разработки корпоративных сайтов.

# 4 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ ПЛАТФОРМЫ ДЛЯ РЕАЛИЗАЦИИ КОРПОРАТИВНЫХ САЙТОВ

Программные продукты являются продукцией, реализуемой покупателям по рыночным ценам. Широкое применение вычислительной техники требует постоянного обновления и совершенствования программ­ных средств. Выбор эффективных проектов программных средств связан с их экономической оценкой и расчетом экономического эффекта. Поэтому разрабатываемые средства должны быть не только совершенными в техническом плане, но и выгодными с точки зрения экономики.

Экономический эффект у разработчика ПО зависит от объёма инвестиций в разработку проекта, цены на готовый программный продукт и количества проданный копий, и проявляется в виде роста чистой прибыли.

Целью дипломного проекта является разработка платформы предназначенной для разработки решений для корпоративных сайтов, нацеленных на реализации различных логистических задач. Данный тип платформ широко распространен, так как предоставляет собой удобный инструмент для быстрой реализации различный задач. Как правило большинство больший компаний занимающихся разработкой ПО в определенной сфере со временем приходят к пониманию того что большинство задач в данной сфере довольно типовые и не нет нужды каждый раз разрабатывать их заново. В виду этого, как правило, компанию принимают решения о разработке платформы, которая будет содержать в себе готовый набор стандартный решений. Разработка данных платформ, как правило, занимает немало времени и средств, но они в дальнейшем они сильно ускорят разработку. Следует отменить что такие платформы как правило являться внутренним продуктом компаний, и все затраты на его разработку компания берет на себя.

## 4.1 Расчёт сметы затрат, необходимых для создания ПО

При ориентации на рынок программы представляют собой готовую продукцию, которая создается в два этапа: первый – определение требований, проектирование, кодирование, сборка и тестирование, отладка ПО; второй – тиражирование, реализация, сопровождение. На первом этапе создается программный продукт и затраты интегрируются в виде себестоимости Спi. На втором этапе выделяются расходы на производство и расходы на реализацию.

Оценка стоимости создания ПО со стороны разработчика предполага­ет составление сметы затрат, которая включает следующие статьи расходов:

* заработную плату исполнителей, основную (3o) и дополнительную (Зд);
* отчисления в фонд социальной защиты населения (Зсз);
* налоги от фонда оплаты труда (Не);
* материалы и комплектующие (М);
* спецоборудование (Рс);
* машинное время (Рм);
* расходы на научные командировки (Рнк);
* прочие прямые расходы (П3);
* накладные расходы (Рн);
* расходы на сопровождение и адаптацию (Рса).

Исходные данные для разрабатываемого проекта указаны в таблице 4.1.

На основании сметы затрат и анализа рынка ПО определяется плано­вая отпускаемая цена. Для составления сметы затрат на создание ПО необ­ходима предварительная оценка трудоемкости ПО и его объёма. Расчет объ­ёма прог­раммного продукта предполага­ет определение типа программного обеспечения, техническое обоснование функций ПО и определение объёма каждой функций. Соглас­но классификации типов программного обеспечения [1], разрабатываемое ПО классифицируется как ПО общего назначения.

Таблица 4.1 – Исходные данные

|  |  |  |
| --- | --- | --- |
| Наименование | Условное  обозна­­чение | Значение |
| Категория сложности |  | 2 |
| Коэффициент сложности, ед. | Кс | 1,12 |
| Степень использования при разработке стандартных модулей, ед. | Kт | 0,8 |
| Коэффициент новизны, ед. | Кн | 0,8 |
| Годовой эффективный фонд времени, дн. | Фэф | 230 |
| Продолжительность рабочего дня, ч. | Тч | 8 |
| Месячная тарифная ставка первого разряда, Вr | ТМ1 | 292000 |
| Коэффициент премирования, ед. | K | 1,5 |
| Норматив дополнительной заработной платы, ед. | Нд | 25 |
| Норматив отчислений в ФСЗН и обязательное страхование, % | Нсз | 34,6 |
| Норматив командировочных расходов, % | Нк | 20 |
| Норматив прочих затрат, % | Нпз | 15 |
| Норматив накладных расходов, % | Нрн | 85 |
| Прогнозируемый уровень рентабельности, % | Урп | 30 |
| Норматив НДС, % | Ндс | 20 |
| Норматив налога на прибыль, % | Нп | 18 |
| Норматив расхода материалов, % | Нмз | 3 |
| Норматив расхода машинного времени, ч. | Нмв | 4,5 |
| Цена одного часа машинного времени, Вr | Нмв | 5000 |
| Норматив расходов на сопровождение и адаптацию ПО, % | Нрса | 30 |

Общий объём программного продукта определяется исходя из коли­чества и объёма функций, реализованных в программе:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | (4.1) |
| 1 | | | | |
| где | Vi | – | объём отдельной функции ПО, LоС; | |
|  | n | – | общее число функций. | |

На стадии технико-экономического обоснования проекта рассчитать точный объём функций невозможно. Вместо вычисления точного объёма функций применяются приблизительные оценки на основе данных по нормативам [1], которые приняты в организации.

Каталог аналогов программного обеспечения предназначен для пред­варительной оценки объёма ПО методом структурной аналогии. Уточненный объём ПО рассчитывает­ся по формуле:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | (4.2) | |
| 1 | | | | |
| где | Vi | – | уточнённый объём отдельной функции ПО, LоС; | |
|  | n | – | общее число функций. | |

Перечень и объём функций программного модуля перечислен в таб­лице 4.2. По приведенным данным уточненный объём некоторых функций изменился, и общий объём ПО составил Vo *=* 53 360 LоС, общий уточнен­ный общем ПО – Vy *=* 45 650 LоС.

Таблица 4.2 – Перечень и объём функций программного модуля

|  |  |  |  |
| --- | --- | --- | --- |
| №  функции | Наименование (содержание) | Объём функции, LоС | |
| по каталогу  () | уточнённый  () |
| 101 | Организация ввода информации | 150 | 200 |
| 102 | Контроль, предварительная обработка и ввод информации | 450 | 450 |
| 105 | Обработка входного заказа и формирование таблиц | 1340 | 1500 |
| 109 | Организация ввода/вывода информации в интерактивном режиме | 320 | 300 |
| 111 | Управление вводом/выводом | 2400 | 1700 |
| 201 | Генерация структуры базы данных | 4300 | 4000 |
| 203 | Формирование баз данных | 2180 | 2000 |
| 204 | Обработка наборов и записей баз данных | 2670 | 1800 |
| 207 | Манипулирование данными | 9550 | 5500 |
| 208 | Организация поиска и поиска в базе данных | 5480 | 5000 |
| 210 | Загрузка баз данных | 2780 | 2200 |
| 304 | Обслуживание файлов | 420 | 420 |
| 305 | Обработка файлов | 720 | 720 |
| 506 | Обработка ошибочных и сбойных ситуаций | 410 | 600 |
| 507 | Обеспечение интерфейса между компонентами | 970 | 900 |
| 601 | Отладка приложения в интерактивном режиме | 4300 | 4300 |
| 605 | Вспомогательные и сервисные программы | 580 | 580 |
| 701 | Математическая статистика и прогнозирование | 9320 | 9000 |
| 707 | Графический вывод результатов | 480 | 480 |
| 708 | Интерактивный редактор текста | 4540 | 4000 |
| Итог |  | 53360 | 45650 |

По уточненному объёму ПО и нормативам затрат труда в расчете на единицу объёма определяются нормативная и общая трудоемкость разра­ботки ПО. ПО относится ко второй категории сложности: предполагается наличие сложного интеллектуального языкового интерфейса с пользователем, также необходимо обеспечить переносимость ПО [1]. Согласно укрупненным нормам времени на разработку [1] нормативная трудоемкость разрабатываемого проекта составля­ет Тн = 923 чел./дн.

Нормативная трудоемкость служит основой для оценки общей трудоемкости Тo. Используем формулу (4.3) для оценки общей трудоемкости для небольших проектов:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | (4.3) |
| 1 | | | | |
| где | Kc | – | коэффициент, учитывающий сложность ПО; | |
|  | Kт | – | поправочный коэффициент, учитывающий степень использования при разработке стандартных модулей; | |
|  | Кн | – | коэффициент, учитывающий степень новизны ПО. | |

Дополнительные затраты труда на разработку ПО учитываются через коэффициент сложности, который вычисляется по формуле:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | (4.4) |
| 1 | | | | |
| где | Ki | – | коэффициент, соответствующий степени повышения слож­­­­­­­­­­­­­ности ПО за счет конкретной характеристики; | |
|  | n | – | количество учитываемых характеристик. | |

Наличие двух характеристик сложности позволяет [1] вычислить коэффициент сложности

|  |  |
| --- | --- |
|  | (4.5) |

Разрабатываемое ПО использует стандартные компоненты. Степень использования стандартных компонентов определяется коэффициентом ис­пользования стандартных модулей – Кт. Согласно справочным данным [1] указанный коэффициент равен Кт = 0,8. Трудоемкость создания ПО также зависит от его новизны и наличия аналогов. Разрабатываемое ПО не является новым, существуют аналогичные разработки у различных компаний, которые, однако уступают доступной функциональности. Влияние степени новизны определяется коэффициентом новизны – Кн. Согласно спра­вочным данным [1] для разрабаты­ваемого ПО Кн = 0,9. Подставив приведенные выше коэффициенты для разрабатываемого ПО в формулу (4.3) получим общую трудоемкость разра­ботки

|  |  |  |
| --- | --- | --- |
|  | (4.6) |  |

На основе общей трудоемкости и требуемых сроков реализации проекта вычисляется плановое количество исполнителей. Численность исполнителей проекта рассчитывается по формуле

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | (4.7) |
| 1 | | | | |
| где | To | – | объём отдельной функции ПО, LоС; | |
|  | Фэф | – | общее число функций. | |
|  | Тр | – | срок разработки проекта, лет. | |

Эффективный фонд времени работы одного разработчика вычисляет­ся по формуле

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | (4.8) |
| 1 | | | | |
| где | Дг | – | количество дней в году, дн.; | |
|  | Дп | – | количество праздничных дней в году, не совпадающих с выходными днями, дн.; | |
|  | Дв | – | количество выходных дней в году, дн.; | |
|  | До | – | количество дней отпуска, дн. | |

Согласно данным, приведённым в производственном календаре для пятидневной рабочей недели в 2015 году для Беларуси, фонд рабочего времени составит

|  |  |
| --- | --- |
|  | (4.9) |

Учитывая срок разработки Тр = 0,75 года, общую трудоёмкость и фонд эффективного времени одного работника, полученные ранее, можно рассчитать численность исполнителей проекта

|  |  |  |
| --- | --- | --- |
|  |  | (4.10) |

Вычисленные оценки показывают, что для выполнения запланированного проекта в указанные сроки необходимо два рабочих. Информация о работниках перечислена в таблице 4.3.

Таблица 4.3 – Работники, занятые в проекте

|  |  |  |  |
| --- | --- | --- | --- |
| Исполнители | Разряд | Тарифный  коэффициент | Чел./дн. занятости |
| Программист I категории | 13 | 3,04 | 175 |
| Программист I категории | 13 | 3,04 | 175 |
| Программист I категории | 13 | 3,04 | 175 |
| Ведущий программист | 15 | 3,48 | 175 |

Месячная тарифная ставка одного работника вычисляется по формуле

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | (4.11) |
| 1 | | | | |
| где |  | – | месячная тарифная ставка 1-го разряда, руб.; | |
|  | Тк | – | тарифный коэффициент, соответствующий установленному тарифному разряду; | |
|  | Фр | – | среднемесячная норма рабочего времени, час. | |

Подставив данные из таблицы 4.3 в формулу (4.11), приняв значение тарифной ставки 1-го разряда ТМ1 = 292 000 руб. и среднемесячную норму рабочего времени Фр = 168 часов получаем

|  |  |  |
| --- | --- | --- |
| = 5 290 руб./час. |  | (4.12) |
|  |  |  |
| = 6 050 руб./час. |  | (4.13) |

Основная заработная плата исполнителей на конкретное ПО рассчи­тывается по формуле

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | (4.14) |
| 1 | | | | |
| где |  | – | часовая тарифная ставка *i-*го исполнителя, руб./час; | |
|  | Тч | – | общее число функций. | |
|  | Фп | – | плановый фонд рабочего времени *i*-го исполнителя, дн.; | |
|  | К | – | коэффициент премирования. | |

Подставив ранее вычисленные значения и данные из таблицы 4.3 в формулу (4.14) и приняв коэффициент премирования К=1,5 получим

|  |  |  |
| --- | --- | --- |
|  |  | (4.15) |

Дополнительная заработная плата включает выплаты предусмотрен­ные законодательством от труде и определяется по нормативу в процентах от основной заработной платы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | (4.16) |
| 1 | | | | |
| где | Нд | – | норматив дополнительной заработной платы, %. | |

Приняв норматив дополнительной заработной платы Нд = 25% и под­ставив известные данные в формулу (4.16) получим

|  |  |  |
| --- | --- | --- |
|  |  | (4.17) |

Согласно действующему законодательству отчисления в фонд соци­альной защиты населения составляют 34%, в фонд обязательного страхо­вания – 0,6%. Общие отчисления на социальную защиту рассчитываются по формуле

|  |  |  |
| --- | --- | --- |
|  |  | (4.18) |

Подставив вычисленные ранее значения в формулу (4.18) получаем

|  |  |  |
| --- | --- | --- |
|  |  | (4.19) |

По статье «материалы» проходят расходы на носители информации, бумагу и другие материалы, используемые при разработке ПО. Приняв норматив Нмз = 3%, получим

|  |  |  |
| --- | --- | --- |
|  |  | (4.20) |

Расходы по статье «машинное время» включают оплату машинного времени, необходимого для разработки и отладки ПО, которое определяется в машино-часах на 100 строк исходного кода и вычисляются по формуле

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | (4.21) |
| 1 | | | | |
| где | Цм | – | цена одного часа машинного времени, руб.; | |
|  | Цмв | – | норматив расхода машинного времени на отладку 100 строк исходного кода, часов. | |

Согласно справочным данным [1], норматив расхода ма­шинного времени на отладку 100 строк исходного кода составляет Нмв = 12, применяя понижающий коэффициент 0,3 получаем Н'мв = 3,6. Цена одного часа машинного времени составляет Цм = 5000 руб. Подставляя известные данные в формулу (4.21) получаем

|  |  |  |
| --- | --- | --- |
|  |  | (4.22) |

Расходы по статье «научные командировки» вычисляются как процент от основной заработной платы. Вычисления производятся по формуле

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | (4.23) |
| 1 | | | | |
| где | Нк | – | норматив командировочных расходов по отношению к основной заработной плате, %. | |

Подставляя ранее вычисленные значения в формулу (4.23) и приняв значение Нк = 20% получаем

|  |  |  |
| --- | --- | --- |
|  |  | (4.24) |

Статья расходов «прочие затраты» включает в себя расходы на приобретение и подготовку научно-технической информации и литературы. Затраты вычисляются по формуле

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | (4.25) |
| 1 | | | | |
| где | Нпз | – | норматив прочих затрат в целом по организации, %. | |

Приняв значение норматива прочих затрат Нпз = 15% и подставив вычисленные ранее значения в формулу (4.25) получаем

|  |  |  |
| --- | --- | --- |
|  |  | (4.26) |

Статья «накладные расходы» учитывает расходы, необходимые для содержания аппарата управления, на общехозяйственные нужны. Данная статья затрат вычисляется по формуле

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | (4.27) |
| 1 | | | | |
| где | Нрн | – | норматив накладных расходов в организации, %. | |

Приняв норму накладных расходов Нрн = 85% и подставив извест­ные данные в формулу (4.27) получаем

|  |  |  |
| --- | --- | --- |
|  |  | (4.28) |

Общая сумма расходов по смете на ПО равна

|  |  |
| --- | --- |
| Ср = 3о + Зд + Зсз + М + Рм + Рнк + Пз + Рн = руб. | (4.29) |

Расходы на сопровождение и адаптацию, которые несет производитель ПО, рассчитываются по формуле

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | (4.30) |
| 1 | | | | |
| где | Нрса | – | норматив расходов на сопровождение и адаптацию ПО, %. | |

Приняв значение норматива Нрса = 35% и подставив ранее вычисленные значения в формулу (4.30) получаем

|  |  |  |
| --- | --- | --- |
|  |  | (4.31) |

Полная себестоимость создания ПО включает сумму затрат на разработку, сопровождение и адаптацию равна

|  |  |  |
| --- | --- | --- |
|  |  | (4.32) |

## 4.2 Расчёт экономической эффективности у разработчика

Важная задача при выборе проекта для финансирования это расчет экономической эффективности проектов и выбор наиболее выгодного проекта. Разрабатываемое ПО является заказным, т. е. разрабатывается для одного заказчика на заказ. На основании анализа рыночных условий и договоренности с заказчиком об отпускной цене прогнозируемая рентабельность проекта составит Урп = 35%. Прибыль рассчитывается по формуле

|  |  |  |  |
| --- | --- | --- | --- |
|  | | (4.33) | |
| 1 | | | |
| где | Пс – прибыль от реализации ПО заказчику, Br; | |
|  | Урп – уровень рентабельности ПО, %. | |

Подставив известные данные в формулу (4.33) получаем прогнозиру­емую прибыль от реализации ПО

|  |  |  |
| --- | --- | --- |
|  |  | (4.34) |

Прогнозируемая цена ПО без учета налогов, включаемых в цену вы­числяется по формуле

|  |  |  |
| --- | --- | --- |
|  |  | (4.35) |

Подставив данные в формулу (4.35) получаем цену ПО без налогов

|  |  |  |
| --- | --- | --- |
|  |  | (4.36) |

Налог на добавленную стоимость рассчитывается по формуле

|  |  |  |  |
| --- | --- | --- | --- |
|  | | (4.37) | |
| 1 | | | |
| где | Ндс – норматив НДС, %. | |

Норматив НДС составляет Ндс = 20%, подставляя известные значения в формулу (4.37) получаем

|  |  |  |
| --- | --- | --- |
|  |  | (4.38) |

Расчет прогнозируемой отпускной цены осуществляется по формуле

|  |  |  |
| --- | --- | --- |
|  |  | (4.39) |

Подставляя известные данные в формулу (4.39) получаем прогнози­руемую отпускную цену

|  |  |  |
| --- | --- | --- |
|  |  | (4.40) |

Приняв значение налога на прибыль Нн = 18%, чистую прибыль от реализации проекта можно рассчитать по формуле

|  |  |  |
| --- | --- | --- |
|  |  | (4.41) |

Таблица 4.4 – Рассчитанные данные

|  |  |  |
| --- | --- | --- |
| Наименование | Условное  обозначение | Значение |
| Нормативная трудоемкость, чел./дн. | Тн | 923 |
| Общая трудоемкость разработки, чел./дн. | То | 661 |
| Численность исполнителей, чел. | Чр | 4 |
| Часовая тарифная ставка программиста 1-разряда, Br /ч. |  | 5 290 |
| Часовая тарифная ставка ведущего программиста, Br /ч. |  | 6 050 |
| Основная заработная плата, Br | Зо | 23 814 000 |
| Дополнительная заработная плата, Br | Зд | 5 953 500 |
| Отчисления в фонд социальной защиты, Br | Зсз | 10 418 625 |
| Затраты на материалы, Br | М | 741 420 |
| Расходы на машинное время, Br | Рм | 3 474 000 |
| Расходы на командировки, Вг | Рк | 4 762 800 |
| Прочие затраты, Вг | Пз | 3 572 100 |
| Накладные расходы, Br | Рн | 20 241 900 |
| Общая сумма расходов по смете, Br | Ср | 72 951 345 |
| Расходы на сопровождение и адаптацию, Br | Рса | 25 532 980 |
| Полная себестоимость, Br | Сп | 98 484 325 |
| Прогнозируемая прибыль, Br | По | 128 029 625 |
| НДС, Br | НДС | 25 605 925 |
| Прогнозируемая отпускная цена ПО, Br | По | 153 635 550 |
| Чистая прибыль, Br | Пч | 24 227 146 |

## 4.3 Вывод

Программное обеспечение разрабатывалось для одного заказчика в связи с этим экономическим эффектом разработчика будет являться чистая прибыль от реализации Пч = 24 227 146 руб. Рассчитанные данные приведены в таблице 4.4. Таким образом было произведено технико-экономическое обоснование раз­рабатываемого проекта, составлена смета затрат и рассчитана прогнозируе­мая прибыль, и показана экономическая целесообразность разработки.

# 5 ОБЕСПЕЧЕНИЕ ПОЖАРНОЙ БЕЗОПАСНОСТИ В КОМПАНИИ «ПАНДА СИСТЕМС»

Целью дипломной работы явился сравнительный анализ платформ для разработки веб систем в области управления логистикой для корпоративных нужд. Целью данного анализа сравнения платформы BBWT с другими аналогами, представленными на рынке. Для этих платформ выявить критерии сравнения и по этим критериям сделать сравнительный анализ. Компания «Панда систем» занимается аутсорсинг разработкой одной из таких платформ. Данная система носит название BBWT. Заказчиком для разработки данной является компания «Blueberry Software». Данная система разрабатывается более 5 лет. Команда по разработке системы в текущее время занимается созданием и поддержкой 2-й версии системы. В компании «Панда систем» на данный момент работает более 10 сотрудников.

Исследование выполнялось при прохождении преддипломной практики в компании «Панда систем»

Исход любого пожара во многом зависит от того, насколько своевременно была вызвана помощь пожарной охраны, приняты меры по эвакуации людей, от готовности всего обслуживающего персонала к действию по тушению пожара. Поэтому каждый сотрудник должен хорошо знать и четко выполнять свои обязанности при возникновении пожара. Для обеспечения пожарной безопасности в оффисе компании «Панда системс» разработан план эвакуации работников при пожаре. Четкое выполнение плана эвакуации дает возможность безопасно эвакуировать людей при пожаре. План эвакуации людей при пожаре со 3-го этажа здания бизнес-центра по адресу: г. Минск, ул. Платонова 43 представлен на рисунке 5.1, инструкция к нему указано на рисунке 5.2.



Рисунок 5.1 – План эвакуации людей при пожаре со 3-го этажа офисного здания.

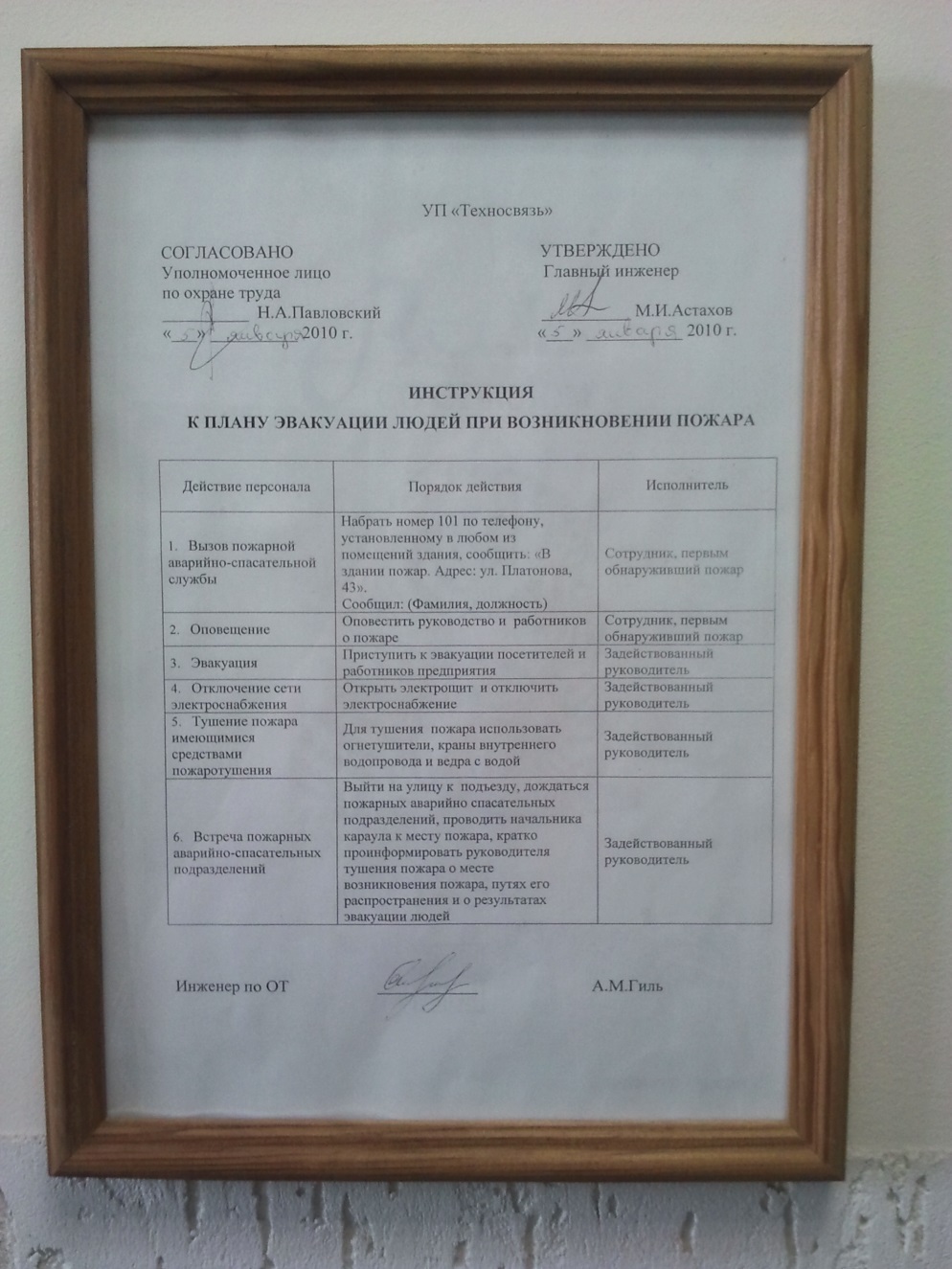


Рисунок 5.2 – Инструкция к плану эвакуации людей при пожаре.

Пожарные выходы обозначены специальными табло и знаками. Один из пожарных выходов представлен на рисунке 5.3. Также в офисе имеются табло указывающие направления к пожарным выходам, рис 5.4.



Рисунок 5.3 – Пожарный выход со 3-го этажа офисного здания.



Рисунок 5.4 – Указатель направления к пожарному выходу.

В помещениях офиса на потолках размещены пожарные извещатели, которые показаны на рисунке 5.5. Они автоматически оповещают МЧС и администрацию здания о пожарной тревоге.



Рисунок 5.5 – Пожарный извещатель, расположенный на 3-м этаже Офисного здания.

На рисунке 5.6 показаны основные средства пожаротушения огнетушители и пожарные краны. На этаже расположено 3 пожарных крана, 6 огнетушителей.

|  |  |
| --- | --- |
| WP_20150505_006  а) | 20130201183224-dsc01010-cfd03f98a08c6f8b  б) |

а – пожарный кран, б – огнетушитель

Рисунок 5.6 – Средства пожаротушения, расположенные на 3-м этаже офисного здания.

На рисунке 5.7 показан речевой извещатель, по которому администрация бизнес-центра производит экстренное информирование работников офисов, в том числе и о случаях пожарной тревоги.



Рисунок 5.7 – Речевой извещатель, расположенный на 3-м этаже Офисного здания.

 Пожарные краны, а также огнетушители должны быть постоянно доступны для использования. Они не должны загораживаться другими объектами. Порядок и условия хранения первичных средств пожаротушения индивидуальных и коллективных спасательных средств, инвентаря, должны обеспечить возможность их использования при пожаре. Работники офиса обязаны немедленно сообщать вышестоящему руководству офиса обо всех обнаруженных нарушениях противопожарных требований и неисправностях пожарной техники, технических средств противопожарной защиты, систем оповещения о пожаре и управления эвакуацией, средств связи, первичных средств пожаротушения и принимать меры по их устранению. В организации не реже одного раза в год организовывается проведение тренировочных занятий для персонала по эвакуации людей из здания в соответствии с планами эвакуации. Для обеспечения безопасной эвакуации не допускается: уменьшать минимальную эвакуационную ширину и высоту, а также загромождать проходы, выходы, двери на путях эвакуации, эвакуационные выходы; изменять направление открывания дверей, препятствующее выходу из зданий и помещений; использовать лифты, подъемники эскалаторы для эвакуации людей; складировать под маршами лестничных клеток горючие материалы и устраивать различные помещения, за исключением узлов управления центрального отопления и водомерных узлов.

Таким образом, изложенные выше предложения обеспечивают пожарную безопасность в компании ЧУП «Панда Системс».

# ЗАКЛЮЧЕНИЕ

В результате работы над дипломным проектом была разработана платформа, предназначенная для реализации корпоративных сайтов.

Платформа предназначена облегчить труд программистов при разработке корпоративных сайтов.

Платформа разработана на языке C# с использование технологий MVC .NET 4.5, ADO.NET Entity Framework и AngularJS. Платформа предоставляет следующие готовые решения:

* гибкую архитектуру приложения;
* сайт шаблон;
* модуль для администрирования;
* набор готовых виджетов для работы с различными типами данных.

Основными достоинствами данной платформы являются:

* простота в освоение, ввиду использования большого количества популярных и удобных технологий;
* легкая расширяемость на любых уровнях архитектуры под нужды заказчика;
* большой набор готовых модулей, упрощающий разработку новых сайтов базирующихся на данной платформе;
* постоянные обновления, нацеленные на улучшение платформы в целом.

Основными недостатками являются:

* узкоспециализированая область применения;
* большие затраты ресурсов на сервере.

Однако за счет своих достоинств, за счет снижения времени и трудоемкости разработки новых сайтов, создание платформы данной платформы является верным решение, для компании, основная область работы которой заключена в реализации корпоративных сайтов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] WordPress! [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/WordPress

[2] Joomla! [Электронный ресурс]. – Электронные данные. – Режим доступа:<http://ru.wikipedia.org/wiki/Joomla>

[3] Хаген Г.Ф. Создание веб-сайтов с помощью Joomla! – М.: «Вильямс», 2009. – 608 с.

[4] Drupal   [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://ru.wikipedia.org/wiki/Drupal>

[5] [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://ru.wikipedia.org/wiki/URL>

[6] Кристиан Нейгел и др. C# 5.0 и платформа .NET 4.5 для профессионалов  — М.: [«Диалектика»](https://ru.wikipedia.org/w/index.php?title=%D0%94%D0%B8%D0%B0%D0%BB%D0%B5%D0%BA%D1%82%D0%B8%D0%BA%D0%B0_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2013. — 1440 с..

[7] Albahari, Joseph. C# 5.0 in a Nutshell / Joseph Albahari, Ben Albahari. – 5th edition. – O’Reilly Media, Inc, 2012. – June. – 1062 P.

[8] Волосевич, А. А. Язык C# и основы платформы .NET: Учебно-метод. пособие по курсу «Избранные главы информатики» для студ.  
спец. I-31 03 04 «Информатика» всех форм обуч. / А. А. Волосевич. – Минск, 2006. – 60 с.

[9] C Sharp [Электронный ресурс]. – Электронные данные. – ­Режим доступа: http://ru.wikipedia.org/wiki/C\_Sharp.

[10] Сандерсон C. ASP.NET MVC Framework с примерами на C# для профессионалов – М.: «Вильямс», 2009. – 608 с.

[11] Фримен А. ASP.NET MVC 5 с примерами на C# 5.0 для профессионалов – М.: «Вильямс», 2014. – 736 с.

[12] TypeScript [Электронный ресурс]. – Электронные данные. – ­Режим доступа: https://ru.wikipedia.org/wiki/TypeScript.

[13] AngularJS [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://ru.wikipedia.org/wiki/AngularJS>.

[14] ППБ РБ 01-2014 Правила пожарной безопасности Республики Беларусь: издание официальное. – Минск, 2014. – 208с.

[15] ППБ РБ 1.02-94 Правила пожарной безопасности Республики Беларусь при эксплуатации технических средств противопожарной защиты. – Минск, 2009. – 17с.

[16] НПБ 28-2001 Пожарная техника. Огнетушители. Требования к эксплуатации – Минск, 2001. – 15с.