

ОГЛАВЛЕНИЕ

ВЕДОМОСТЬ ОБЪЕМА ДИПЛОМНОГО ПРОЕКТА	
ВВЕДЕНИЕ.....	7
ОБЗОР ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ.....	9
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	10
2 ПОСТАНОВКА ЗАДАЧИ	13
2.1 Общие определения.....	13
2.2 Определение требований к системе.....	14
3 ПРОЕКТИРОВАНИЕ И МОДЕЛИРОВАНИЕ.....	16
3.1 Логическая модель	16
3.1.1 Основные роли в системе.....	16
3.1.2 Варианты использования системы	16
3.2 Общая архитектура системы.....	17
3.3 Проектирование серверной части.....	21
3.3.1 Программная платформа.....	21
3.3.2 Модель взаимодействия клиентского приложения и сервера	23
4 РЕАЛИЗАЦИЯ СЕРВЕРНОЙ ЧАСТИ	26
4.1 Обоснование выбора инструментов и технологий	26
4.1.1 Среда разработки и язык программирования.....	26
4.1.2 Выбор сервера базы данных.....	27
4.2 Реализация основных модулей сервера.....	29
4.2.1 Модуль обработки веб-запросов.....	30
4.2.2 Механизм авторизации и аутентификации	34
4.2.3 Модуль доступа к данным.....	35
5 РАЗВЕРТЫВАНИЕ И ТЕСТИРОВАНИЕ.....	41
5.1 Развертывание серверной части на облачной платформе	41
5.2 Тестирование веб-сервера.....	46
5.2.1 Критическое тестирование.....	47
5.2.2 Углубленное тестирование	51
6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОЙ СИСТЕМЫ.....	53
6.1 Расчет сметы затрат, цены и прибыли на программную систему	53
6.1.1 Исходные данные	54
6.1.2 Общая характеристика разрабатываемой программной системы.....	55
6.1.3 Определение объема программной системы.....	55
6.1.4 Расчет трудоемкости выполняемой работы	56
6.1.5 Расчет основной заработной платы исполнителей	58
6.1.6 Расчет дополнительной заработной платы исполнителей.....	60

6.1.7	Расчет отчислений в фонд социальной защиты населения	60
6.1.8	Расчет налога на ликвидацию последствий чернобыльской катастрофы..	61
6.1.9	Расчет расходов по статье «Материалы»	61
6.1.10	Расчет расходов по статье «Спецоборудование»	62
6.1.11	Расчет расходов по статье «Машинное время».....	62
6.1.12	Расчет расходов по статье «Научные командировки»	63
6.1.13	Расчет расходов по статье «Прочие затраты»	63
6.1.14	Расчет расходов по статье «Накладные расходы»	64
6.1.15	Расчет общей суммы расходов на разработку	64
6.1.16	Расчет прибыли и отпускной цены	65
6.2	Расчет экономического эффекта от применения ПС у пользователя	66
6.2.1	Исходные данные	66
6.2.2	Расчет объема работ	66
6.2.3	Расчет капитальных затрат.....	67
6.2.4	Расчет экономии основных видов ресурсов в связи с использованием нового программного средства.....	68
6.2.5	Расчет экономического эффекта	69
7	ОХРАНА ТРУДА.....	72
7.1	Производственная санитария, техника безопасности и пожарная профилактика	72
7.1.1	Метеоусловия	72
7.1.2	Вентиляция и отопление	73
7.1.3	Освещение.....	74
7.1.4	Шум.....	75
7.1.5	Электробезопасность.....	75
7.1.6	Излучение	76
7.1.7	Пожарная безопасность.....	77
7.2	Организация рабочего места пользователя ПЭВМ.....	78
	ЗАКЛЮЧЕНИЕ.....	80
	СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ	
	ГРАФИЧЕСКАЯ ЧАСТЬ	

ВВЕДЕНИЕ

На протяжении всей истории человечества, люди решают проблемы, воплощают в жизнь различного рода идеи, создают новые полезные всему миру изобретения и какие-то материальные или нематериальные блага. И каждый раз на пути достижения поставленной цели, будет ли это планирование строительство дома, человеку предстоит:

- определить набор задач, которые необходимо выполнить для достижения цели;
- спланировать работу над поставленными задачами;
- совершать ошибки, сталкиваться с проблемами при работе над очередной задачей, а значит накапливать опыт и использовать его для решения схожей задачи в будущем.

Во время всего вышеперечисленного человек может испытать ряд проблем человеческого фактора: забыть, что ему нужно было сделать в первую очередь, выполнить задачу не совсем так, как планировал, не учесть ряд ошибок, которые он совершил в прошлых задачах, при выполнении очередной задачи и т.п. Все эти проблемы могут пагубно сказаться на конечном результате и поставить под вопрос достижение поставленной цели.

Кроме того, люди зачастую объединяются в группы для того, чтобы быстро и качественно достигнуть какую-либо поставленную цель. Таким образом, формируется команда, которой предстоит достигнуть некоторую общую цель, для достижения которой необходимо по-прежнему разбить работу на задачи, спланировать работу, накапливать опыт [1]. Но в отличие достижения цели одним человеком, работа в команде приносит дополнительные этапы:

- коммуникация – время от времени, различные члены команды должны делиться различного рода информацией: рассказать о своем прогрессе руководителю, передать задачу на выполнение другому члену команды, информировать о проблеме или завершении задачи;
- организация – каждый член команды обязан знать набор своих задач, процесс информирования команды, процесс передачи задачи от одного человека другому и др.;
- синхронизация – видение конечного результата владеют определенные люди, которые должны сформировать и изъяснить это видение для всех исполнителей.

На всех перечисленных этапах члены команды испытывают различного рода проблемы, сказывающиеся на работе каждого из них и на результате совместной работы:

- все члены команды имеют свое собственное представление о конечном результате – это означает, что каждому нужно разъяснить, что нужно делать в

деталях, при этом каждая упущенная деталь в разъяснении может поспособствовать появлению дефектов и проблем в разрабатываемом продукте;

- каждый член команды должен тратить время на различные организационные моменты: уведомление о своем прогрессе по важной задаче, передача работником выполненной в рамках его сферы задачи другому работнику из другой сферы с посвящением во все детали этой задачи и т.п;

- разграничение ответственности может быть очень неясной в большой команде.

Это лишь некоторые из проблем, которые появляются на протяжении всей работы над проектом в целом. Источником всех этих проблем является то, что принятые решения, результаты обсуждений, описания требований и того как должна выглядеть система нигде не фиксируется. Естественно, что большинство проблем можно решить с помощью ведения некоторой документации, но в конечном счете это приведет к огромному количеству документов, с которыми сложно работать, производить какой-либо анализ ошибок и вносить корректировки в работу на основании его [2].

Целью данного дипломного проекта является решение всех вышеперечисленных проблем и создание единой системы для ведения документации, управления задачами, контроля сроков, налаживания коммуникации и анализа проблем в разработке проекта.

ОБЗОР ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

Для изучения платформы .Net Framework и языка C# подходит книга «CLR via C#». Программирование на платформе Microsoft .NET Framework 4.5 на языке C#» Джеффри Рихтер, 2019 год. Эта книга — подробное описание внутреннего устройства и функционирования общезыковой исполняющей среды (CLR) Microsoft .NET Framework версии 4.6. В ней раскрыта система типов .NET Framework и разъяснены способы управления ими. Представлены концепции программирования с широким использованием библиотеки FCL, относящиеся ко всем языкам, ориентированным на работу с .NET Framework. Особое внимание уделено обобщениям, управлению асинхронными операциями и синхронизации потоков. Книга ориентирована на разработчиков любых видов приложений на платформе с .NET Framework: Windows Forms, Web Forms, Web-сервисов, консольных приложений и пр. Второе издание книги выпущено с учетом отзывов читателей и исправлений автора.

В другой книге «Programming ASP.NET Core», автором которой является Дино Эспозито, подробно описывает процесс создания веб-приложений с помощью ASP.NET Core. Данная книга дает исчерпывающие знания по практической разработке веб-приложений с использованием новой платформы Microsoft ASP.NET Core. Автор представляет проверенные методы и хорошо продуманный пример кода для решения реальных проблем с ASP.NET Core. Также в книге описаны ключевые технологии ASP.NET Core, включая MVC для генерации HTML, некоторые моменты по платформе .NET Framework Core, затронута технология объектно-реляционной модели EF Core, механизмы авторизации и аутентификации в ASP.NET, внедрение зависимостей и многое другое. Дино подробно описывает кроссплатформенные возможности ASP.NET Core и то, что изменилось по сравнению со старыми версиями ASP.NET, кроме того он описывает способы по программной реализации производственных решений, включая решения для мобильных устройств.

Книга «Внедрение зависимостей в .NET», автором которой является Марк Симан, рассказывает о внедрении зависимостей и является практическим руководством по их применению в приложениях .NET. Издание содержит основные шаблоны внедрения зависимостей, написанные на «чистом» C#. Кроме того, рассмотрены способы интеграции внедрений зависимостей со стандартными технологиями Microsoft, такими как ASP.NET MVC, а также примеры применения фреймворков StructureMap, Castle Windsor и Unity.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Проект — совокупность мероприятий для разработки нового продукта или улучшения существующего продукта.

Управление проектами — область деятельности, в ходе которой определяются и достигаются чёткие цели проекта при балансировании между объёмом работ, ресурсами [3]. Ключевым фактором успеха проектного управления является наличие чёткого заранее определённого плана, минимизации рисков и отклонений от плана, эффективного управления изменениями.

В свою очередь управление проектами можно детализировать на несколько процессов, каждый из которых нацелен на достижение максимальной эффективности использования ресурсов:

- планирование – заключается в планировании расписания работы сотрудников, назначении ресурсов на конкретные задачи, расчете времени, необходимого на решение каждой из задач и т.д.;

- управление данными – использовать информацию о нагрузке работников, текущих задачах, ходе проекта для прогнозирования и раннего предупреждения рисков и распределения ресурсов;

- управление коммуникациями команды проекта - обсуждение и согласование рабочих вопросов проекта, фиксация проблем проекта и запросов на изменения, их обработка и т.д.

Программное обеспечение для управления проектами — комплексное программное обеспечение, включающее в себя приложения для планирования задач, составления расписания, распределения ресурсов, совместной работы, общения, быстрого управления, документирования и администрирования системы, которое используются совместно для управления крупными проектами. В данном приложении, независимо от его типа, должны быть реализованы все перечисленные выше возможности, связанные с процессом управления проектом. Программное обеспечение для управления проектами могут быть различных типов.

Desktop (Настольные). Программное обеспечение находится на компьютере каждого пользователя. Такие приложения обычно позволяют сохранять информацию в файл, который в дальнейшем может быть выложен в общий доступ для других пользователей или же данные хранятся в центральной базе данных.

Web-based (Веб-интерфейс). Программное обеспечение является веб-приложением, доступ к которому осуществляется с помощью браузера. Данный тип программного решения наиболее распространен в силу своей эффективности для данной предметной области.

Плюсы и минусы веб-ориентированных решений:

- доступ может быть осуществлен с любого компьютера, не требуется установка

дополнительных приложений;

- простой контроль доступа;
- многопользовательский доступ;
- только одна программа, которая установлена на центральном сервере;
- скорость работы ниже, чем у обычных приложений;
- выход из строя сервера приводит к полной недоступности информации [4].

В настоящий момент имеется множество веб-приложений для управления проектами. Далее будет представлен краткий обзор по наиболее популярным системам в данной предметной области.

Jira — коммерческая система отслеживания ошибок, предназначена для организации взаимодействия с пользователями, хотя в некоторых случаях используется и для управления проектами. Главная страница данной системы приводится на рисунке 1.1.

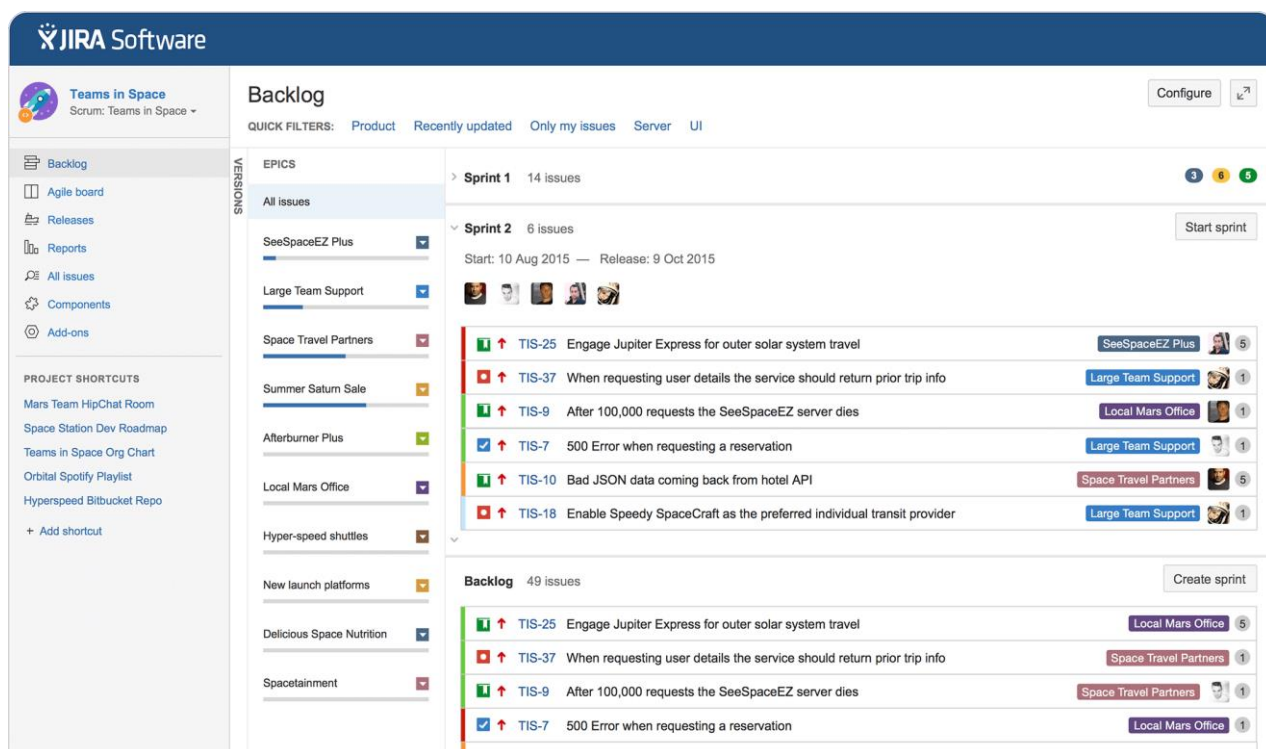


Рисунок 1.1 – Главная страница Jira

Jira была разработана компанией Atlassian и является одним из двух её основных продуктов [5]. Имеет веб-интерфейс, возможности по созданию задач, мониторингу прогресса по проекту и другие возможности. Jira имеет следующие недостатки:

- высокая сложность системы;
- высокая стоимость (для команды в 50 человек годовая подписка достигает 7000\$);
- необходимость иметь специального администратора, который бы обслуживал

пользователей системы в случае возникновения проблем;

– вся система, как и хранимая информация по проектам, находится на внешних серверах, что может оказаться нежелательным для проектов с жесткими требованиями по информационной безопасности.

Trello — программа для управления проектами небольших групп, разработанное Fog Creek Software. Trello использует парадигму для управления проектами, известную как «канбан», метод, который первоначально был популяризирован Toyota в 1980-х для управления цепочками поставок. Главная страница упомянутой системы приводится на рисунке 1.2.

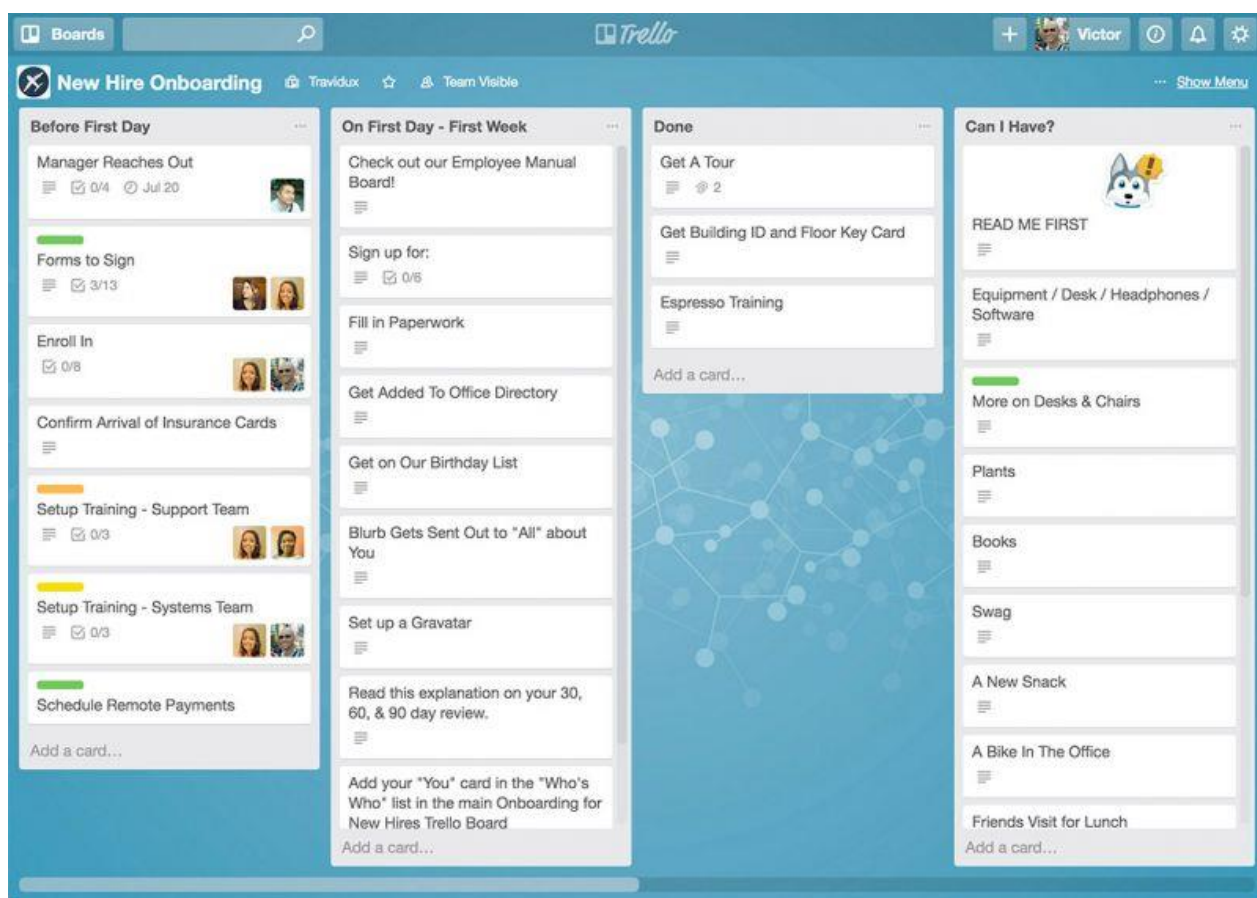


Рисунок 1.2 – Главная страница Trello

Данный инструмент хорошо подходит для планирования бытовых работ, и работы в небольшой команде, но в силу своей простоты имеет ряд недостатков:

- данный инструмент абсолютно не подходит для организации работ на больших проектах в силу отсутствия множества необходимых возможностей;
- административный контроль и различного рода улучшения доступны только в платной версии программы;
- низкий уровень безопасности.

2 ПОСТАНОВКА ЗАДАЧИ

2.1 Общие определения

Пользователь – информация, связанная с реальным человеком, имеющим доступ к использованию данной системы. Набор функционала, которым может воспользоваться пользователь, зависит от назначенных ему прав доступа. Для различных проектов пользователь может иметь различные права доступа.

Администратор – специальный пользователь, который отвечает за управление пользовательскими аккаунтами в системе, выполняет конфигурацию системы на глобальном уровне (в отношении всех проектов) и имеющий непосредственный доступ к базе данных.

Программный проект – может означать идею, описание цели или чего-либо другого, по отношению к чему могут выполняться какие-либо задачи.

Задача – означает конкретную единицу работы, которую нужно выполнить конкретным исполнителем по отношению к определенному проекту.

Новая задача, или новая функция, или новое свойство системы – задача, связанная с реализацией чего-то нового.

Дефект – задача, связанная с исправлением определенных проблем.

Иная задача – абстрактный тип задачи, обычно используется для тех случаев, когда другие типы задач не подходят.

Подзадача – дочерняя задача. Необходима для детализации или разбиения некоторой задачи на несколько подзадач (подзадачи могут быть созданы для любого типа задачи кроме типа «подзадача»).

Исполнитель – конкретный пользователь, ответственный за выполнение задачи.

Приоритет – приоритет задачи. Означает важность этой задачи по отношению к другим. Виды приоритетов:

- тривиальный – означает минимальный приоритет задачи (например изменение цвета кнопки, сортировка документов и т.п.);

- незначительный – более высокий приоритет, но все еще не критичный к исполнению (результат работы над проектом может отдаваться даже в случаях, когда не выполнены некоторые задачи с данным видом приоритета);

- высокий – означает высокий приоритет и важность выполнения этой задачи для конечного клиента/заказчика;

- критический – означает высокий приоритет и срочность исполнения задачи как можно скорее;

- блокирующий – также высокий приоритет, но отличается от других подобных тем, что пока не выполнена блокирующая задача, другие члены команды будут заблокированы и не смогут продолжать свою работу.

Предполагаемое время – запланированное время необходимое для выполнения задачи.

Затраченное время – фактически потраченное время на задачу.

2.2 Определение требований к системе

Необходимо разработать систему «Zigo» для реализации удобного и эффективного управления программными проектами. Управление проектами подразумевает собой совокупность мер, мероприятий нацеленных на достижение цели проекта – разработки и внедрения программного обеспечения. В основу управления проектами положена задача, а точнее организация их выполнения среди участников проекта, мониторинг проблем в процессе их выполнения, управление доступом определенных участников к проектам и т.д. Задача должна описывать конкретную область работ в отношении проекта: исправление дефекта, добавление новой функциональности, оптимизация, тестирование, документирование и т.д. Таким образом, совокупность всех выполненных задач по конкретному проекту будет равносильна достижению цели проекта – реализация программного обеспечения.

В связи с этим система должна обладать следующими возможностями:

- создание проекта и снабжение его необходимой информацией и документацией;
- добавление новых участников в проект, а также возможность лишения доступа определенных участников в отношении определенного проекта;
- управление задачами: создание задачи, назначение исполнителя, установка сроков, приоритета перед другими задачами, типа задачи;
- возможность обсуждения проблем, предложений, вариантов улучшения в рамках какой-либо задачи любым участником проекта в любом количестве;
- ведение журнала работ по каждой из задач с описанием выполненных работ и потраченного времени для определения темпа разработки проекта и выявления всех проблем препятствующих быстрому выполнению схожих задач;
- сортировка задач в зависимости от сроков их завершения, приоритета и прочих свойств задачи;
- управление несколькими проектами одновременно;
- управление списком задач для сотрудников и предоставление информации по распределения ресурсов;
- обзор информации о сроках выполнения задач;
- возможность раннего предупреждения о возможных рисках, связанных с проектом;
- обзор информации о рабочей нагрузке;
- предоставление информации о ходе проекта, показатели и их

прогнозирование;

- обсуждение и согласование рабочих вопросов проекта;
- фиксация проблем проекта и запросов на изменения, их обработка;
- ведение рисков проекта и управление ими.

Кроме вышеупомянутых возможностей система должна обладать рядом свойств, без которых эффективность всей системы будет на недостаточном уровне для осуществления гибкого управления проектами:

1) доступность – доступ в систему должен осуществляться из любого устройства, из любого места и в любое время;

2) согласованность и целостность данных – вся информация должна быть сконцентрирована в одном месте либо храниться таким образом, чтобы пользователь системы всегда имел дело с актуальными данными;

3) безопасность – данные должны быть надежно защищены и недоступны для посторонних, по отношению к данным, лиц;

4) простота использования – интерфейс системы должен быть прост и понятен, так чтобы пользователь не должен был обладать специальными техническими знаниями необходимыми для работы с системой;

5) легковесность и простота развертывания – конечное клиентское приложение не должно быть громоздким, отнимать большое количество вычислительных ресурсов и содержать большое количество шагов по установке системы. Также система не должна быть зависима от большого количества сторонних программ для своей работы.

С точки зрения программной архитектуры, система должна быть распределенной и иметь несколько клиентских приложений, каждое из которых нацелено на конкретное устройство.

3 ПРОЕКТИРОВАНИЕ И МОДЕЛИРОВАНИЕ

3.1 Логическая модель

3.1.1 Основные роли в системе

В системе должны быть определены некоторые логические группы пользователей, при попадании в которые пользователь будет иметь возможность пользоваться определенным набором функционалом. Отношение пользователя к той или иной группе будет означать то, что пользователь имеет определенную роль в системе.

В системе должны быть определены две основные роли пользователей:

1) пользователь (User) – наделенные данной ролью пользователи системы могут пользоваться основным функционалом системы достаточным для осуществления деятельности по управлению проектами;

2) администратор (Administrator) – данная роль присваивается пользователем, ответственным за создание новых пользователей в системе, назначение им ролей, управление доступом пользователей к определенным проектам, а также создание проектов в системе.

Между данными двумя ролями существует большая разница как в контексте использования функционала системы, так и в целом отношения к главной деятельности – управлению проектами.

3.1.2 Варианты использования системы

Пользователи, имеющие роль администратор по своей сути не имеют никакого отношения к управлению проектами. Их главная обязанность – введение новых пользователей и проектов в систему, а также разграничение прав доступа. Также они обязаны разрешать любые вопросы, касающиеся работоспособности системы, и восстанавливать учетные записи для пользователей с ролью «User».

Таким образом, для администраторов должен быть предусмотрен и доступен следующий функционал в системе:

- 1) создание первоначальной учетной записи для пользователя;
- 2) назначение роли для пользователя;
- 3) создание проекта;
- 4) назначение для пользователя проектов, с которыми он может работать;
- 5) лишения доступа к определенным проектам для определенного пользователя.

В свою очередь пользователи, относящиеся к роли «User», не могут никаким образом воздействовать на учетные данные других пользователей либо на данные по

проектам, но могут осуществлять основную деятельность в отношении проектов. Т.е. пользователь с ролью «User» это тот самый человек, который имеет прямое отношение к разработке и развитию программного проекта. Это может быть как программист, ведущий разработку на проекте, так и проектный руководитель, организующий работу среди подчиненных, или же тестировщик, обеспечивающий качество выполненных работ, и др. При этом данные пользователи могут вести свою деятельность только в отношении тех проектов, к которым им был дан доступ администратором системы, т.е. они ничего не будут знать о других проектах в системе и любой относящейся к ним информации. В свою очередь, когда пользователю назначается какой-то проект, он получает полный доступ ко всему функционалу и информации в отношении проекта и становится частью команды (группа людей относящихся к одному и тому же проекту).

Таким образом, для осуществления своей деятельности, пользователям с данной ролью необходим следующий функционал в системе:

- 1) редактирование профиля с заданием всех ключевых персональных данных: должность, контакты, фото и т.д;
- 2) просмотр информации о членах команды, включая их контакты;
- 3) получение списка текущих задач, всех задач в проекте, закрытые задачи в прошлом для определенного проекта, а также получение списка задач по множеству другим критериям для формирования статистики, изучения предыдущего опыта и для других целей;
- 4) создание задачи с указанием всех основных параметров: название, описание, сроки, тип, статус и другие данные;
- 5) возможность редактирования основных данных по задаче (за исключением некоторых данных, таких как номер задачи, создатель, дата создания и т.д.);
- 6) назначение исполнителя для определенной задачи;
- 7) возможность обсуждения задачи с остальными членами команды;
- 8) ведение и просмотр журнала работ по конкретной задаче;
- 9) просмотр информации по текущим проектам;
- 10) возможность загрузки документации, относящейся к определенному проекту.

Варианты использования описанного функционала для каждой роли пользователей отображено в приложении.

3.2 Общая архитектура системы

Прежде чем приступить к разработке конечного программного обеспечения, необходимо определиться с его типом и архитектурой, для этого необходимо принять во внимание все основные критерии, определенные в постановке задачи, и

определить необходимые компоненты системы.

В контексте работы с данными разрабатываемое программное обеспечение должно удовлетворять критериям доступности и согласованности это означает то, что каждое пользовательское приложение не может иметь свое собственное хранилище данных, т.к. данные используются совместно с другими пользователями и пришлось бы тратить множество ресурсов на поддержание согласованности данных во всех хранилищах всех пользователей и всегда. Из этого следует вывод, что должна существовать единая база данных, доступ к которой должны иметь приложения пользователей. На данном этапе архитектура разрабатываемого приложения выглядело бы так, как показано на рисунке 3.1

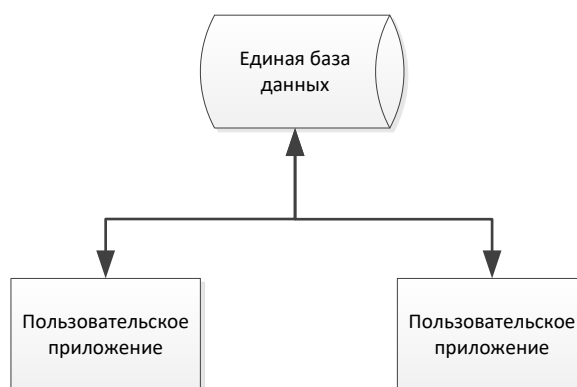


Рисунок 3.1 – Архитектура приложения взаимодействующего с базой данных

Но данная архитектура не удовлетворяет другим определенным критериям:

- простота – приложение не получится использовать сразу же после ее установки, т.к. нужно будет потратить время на ее конфигурацию (работа с базой данных, настройки безопасности, другие настройки необходимые для сторонних сервисов и обработки данных);

- легковесность – приложение будет тратить большое количество ресурсов пользовательской машины, т.к. все операции по обработки данных будут осуществляться на ней, так же должны будут установлены все компоненты по взаимодействию со сторонними сервисами и базой данных, что значительно увеличит размер приложения;

- безопасность – текущая архитектура приложения увеличивает риск доступа к базе данных злоумышленником;

- доступность – пользователи должны иметь возможность работать отовсюду, это значит, что и база данных должна быть доступна всегда.

Для удовлетворения всех вышеперечисленных требований необходимо разбить компонент «Приложение пользователя» на два других программных обеспечения: клиентское приложение и сервер.

Дополнительное звено «Сервер» в архитектуре необходимо для соблюдения вышеописанных критериев системы за счет следующих характеристик:

- доступ к базе данных – только сервер будет иметь прямой доступ к базе данных, таким образом никаких сведений о базе данных не будет храниться на клиентских приложениях, что повышает безопасность доступа к данным;
- обработка – сервер отвечает за обработку всех запросов поступающих с клиентских приложений, таким образом, клиентское приложение больше не будет выполнять никаких затратных действий, а также будет иметь значительно меньший размер и сложность.

Текущая архитектура системы представлена на рисунке 3.2.

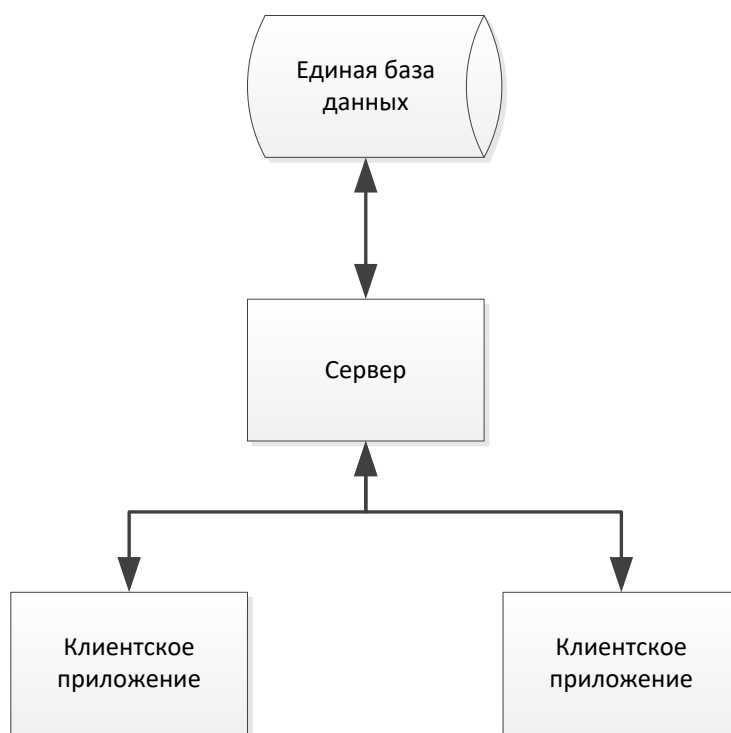


Рисунок 3.2 – Архитектура клиент-серверной системы с доступом к базе данных

Существует лишь один способ обеспечения абсолютной доступности сервера для клиентских приложений – осуществления их взаимодействия в сети Интернет. Сервер, работающий в соответствии со стандартами и протоколами, определенными в сети интернет, называется веб-сервером. С другой стороны клиентские приложения должны устанавливать соединение с сервером и посылать запросы в правильном формате. Теперь система будет содержать три звена: веб-сервер, клиентское веб-приложение и базу данных. При этом предполагается, что веб-сервер будет находиться на одной машине вместе с базой данных либо на отдельной машине, но в рамках одной локальной сети вместе с машиной сервера базы данных, поэтому к базе данных будет отсутствовать любой прямой доступ с любого другого веб-приложения,

кроме разрабатываемого веб-сервера. Новая архитектура системы представлена на рисунке 3.3.



Рисунок 3.3 – Архитектуры веб-системы

На данном этапе все основные критерии соблюдены, остаются некоторые проблемы с разворачиванием клиентского приложения на машине пользователя и использованием его на различных устройствах.

Предполагается, что пользователь будет использовать приложение на мобильном устройстве и на компьютере, при этом операционная система может быть любой. Для того чтобы удовлетворить пользователей компьютеров и ноутбуков уже существует готовое решение, которое независимо от операционной системы, знакомо каждому практически каждому пользователю и работающее в Интернете – браузер.

Для пользователей мобильных устройств можно было бы тоже использовать браузер, но это не лучшее решение в контексте разработки пользовательского интерфейса сразу для двух категорий пользователей. Также существует и другие причины в пользу создания отдельного мобильного приложения (больше возможностей по созданию графического интерфейса, адаптация под разные размеры экранов самая лучшая и др.). Таким образом, для пользователей мобильных устройств необходимо создать отдельное мобильное веб-приложение.

В результате клиентская часть системы делится на два типа: использование браузера пользователями компьютера и использование мобильного приложения пользователями мобильных устройств. Финальная архитектура разрабатываемой системы представлена на рисунке 3.4.

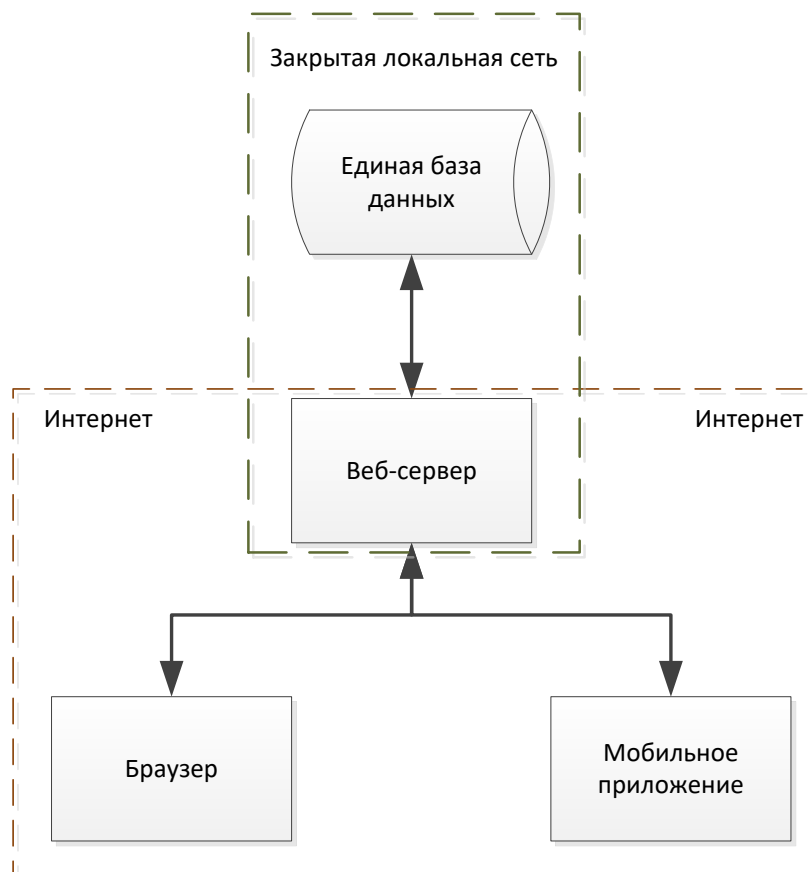


Рисунок 3.4 – Архитектура веб-системы с несколькими типами клиентских приложений

На данном этапе все требования к архитектуре удовлетворены, а значит, она и будет являться основой для разрабатываемой системы.

3.3 Проектирование серверной части

3.3.1 Программная платформа

Любое программное приложение, написанное на любом языке и с использованием каких-либо технологий, работает под управлением определенной платформы. В качестве платформы обычно выступает операционная система (Windows, Linux и др.) либо промежуточные программные платформы (.net, JVM и др.), которые в свою очередь работают под управлением ОС, но не зависят ее

конкретного типа и модели процессора компьютера.

Разрабатываемая серверная часть не должна зависеть от конкретной операционной системы и модели процессора, поэтому необходимо выбрать целевую программную платформу, которая может запускаться на любой из операционных систем. На данный момент существует две наиболее мощные платформы под управлением которой могут работать веб-сервера:

- 1) платформа Java;
- 2) .Net Framework.

Платформа Java - это набор программ, которые облегчают разработку и запуск программ, написанных на языке программирования Java . Платформа Java будет включать механизм выполнения (называемый виртуальной машиной), компилятор и набор библиотек; могут быть также дополнительные серверы и альтернативные библиотеки, которые зависят от требований. Слово «Java», как правило, относится к языку программирования Java, который был разработан для использования с платформой Java. Языки программирования, как правило, выходят за рамки фразы «платформа», хотя язык программирования Java был включен в качестве основной части платформы Java до Java 7.

Основные преимущества Java платформы:

- 1) полная поддержка ООП;
- 2) независимость платформы от конкретной операционной системы;
- 3) поддержка апплетов – небольшие веб-программы, которые предоставляют интерактивные элементы для визуализации и обучения. Они не используются ни для чего, кроме простой анимации, однако апплеты привлекли внимание многих программистов и подтолкнули их к разработке HTML5, Flash и JavaScript;
- 4) автоматическое управление памятью;
- 5) многопоточность;
- 6) широкая область применения (мобильная разработка, веб, настольные приложения).

Недостатки:

- 1) платное коммерческое использование;
- 2) низкая производительность;
- 3) отсутствие собственных инструментов создания графического интерфейса для некоторых типов приложений;
- 4) многословный и сложный код.

.Net Framework – программная платформа, выпущенная компанией Microsoft в 2002 году. Основой платформы является общезыковая среда исполнения Common Language Runtime (CLR), которая подходит для разных языков программирования. Функциональные возможности CLR доступны в любых языках программирования, использующих эту среду. Другая основная часть платформы .NET Framework

является библиотека классов платформы (FCL). Она является вторым основным компонентом платформы .NET, которая упрощают работу программиста, предоставляя ему, более расширенные библиотеки, такие как:

- 1) ASP.NET – библиотека классов для создания веб-приложений;
- 2) ADO.NET – библиотека классов для взаимодействия с базой данных;
- 3) Windows Forms и WPF – обе библиотеки предоставляют возможность создания настольных приложений.

В настоящее время .NET Framework получает развитие в виде .NET Core, изначально предполагающей кроссплатформенную разработку и эксплуатацию.

Основные преимущества использования .Net Framework:

- 1) открытый код платформы;
- 2) абсолютно бесплатное использование;
- 3) полная поддержка ООП;
- 4) стремительное развитие и улучшение производительности;
- 5) автоматическое управление памятью;
- 6) возможность использования нескольких языков программирования;
- 7) оптимизация исполнения кода для конкретной модели процессора.

Недостатки .Net Framework:

- 1) высокий порог вхождения – нужно потратить большое количество времени на изучение платформы;

- 2) нацеленность на ОС Windows (начиная с версии Core платформа более не зависит от конкретной операционной системы).

Для разработки серверной части системы управления проектами будет выбрана платформа .Net в силу следующих причин:

- 1) целевая платформа для разрабатываемого сервера будет Windows, в свою очередь .Net полностью интегрирован и использует все возможности данной ОС;
- 2) бесплатность и производительность;
- 3) мощные инструменты для создания веб приложений и веб-служб.

Таким образом, веб-сервер системы будет разрабатываться для платформы .Net Framework на базе технологий ASP.Net Core.

3.3.2 Модель взаимодействия клиентского приложения и сервера

Для того чтобы веб-сервер мог взаимодействовать с клиентскими приложениями различных типов он должен предоставлять свой собственный API интерфейс.

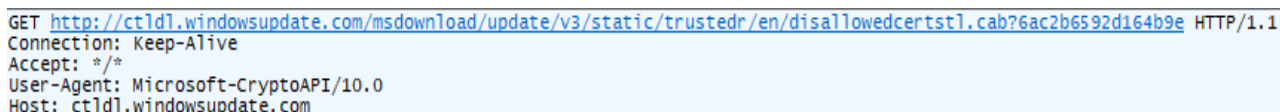
API (программный интерфейс приложения, интерфейс прикладного программирования) – описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой. Т.е. API интерфейс скрывает за собой

некоторый функционал по обработке данных, предоставляя в открытом виде просто способ воспользоваться этим функционалом другим программам.

В контексте веба, взаимодействие между клиентским приложением и сервером должно осуществляться по протоколу HTTP, а значит должен соблюдаться некий набор правил форматов запроса, отправляемых клиентскими приложениями, и форматов ответов, отправляемых сервером в ответ на запрос.

В связи с этим, взаимодействие между клиентом и сервером должно соответствовать архитектуре, называемой REST. REST (сокращение от англ. Representational State Transfer — «передача состояния представления») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы. В определённых случаях (интернет-магазины, поисковые системы, прочие системы, основанные на данных) это приводит к повышению производительности и упрощению архитектуры. В широком смысле компоненты в REST взаимодействуют наподобие взаимодействия клиентов и серверов во Всемирной паутине.

В сети Интернет обращение к API может представлять собой обычный HTTP-запрос (обычно «GET» или «POST»; такой запрос называют «REST-запрос»), а необходимые данные передаются в качестве параметров запроса. Пример REST-запроса представлен на рисунке 3.5



```
GET http://ctld1.windowsupdate.com/msdownload/update/v3/static/trustedr/en/disallowedcertstl.cab?6ac2b6592d164b9e HTTP/1.1
Connection: Keep-Alive
Accept: */*
User-Agent: Microsoft-CryptoAPI/10.0
Host: ctld1.windowsupdate.com
```

Рисунок 3.5 – Формат REST-запроса

Основные секции запроса:

- 1) заголовки — здесь находится информация о типе запроса (Get, Post и др.), идентификационная информация и другие мета-данные;
- 2) адрес, по которому отправляется запрос (также может содержать параметры, если тип запроса GET);
- 3) тело запроса — необходимые данные и параметры (данная секция существует только для типа запроса POST).

Исходя из этого, необходимо разработать Restfull-веб-сервер, т.е. веб-сервер, работающий по протоколу http с поддержкой архитектуры REST.

Реализация API представляет собой совокупность API-методов доступных клиентским приложениям. Каждый API-метод имеет свой тип, адрес и конкретный способ обработки информации.

Таким образом, веб-сервер должен предоставлять информацию о всех

реализованных API-методах с указанием следующей информации:

- адрес API-метода – адрес конкретного метода в интерфейсе API, выполняющий определенную обработку (например `api/user/getallusers`), данный адрес должен объединяться с именем веб-сервера в Интернете и `http` протоколом, для того чтобы можно было обратиться к данному методу;

- тип принимаемого запроса (Get, Post, Put или Delete);

- перечень всех входных параметров и их форматы;

- формат данных отправляемых в ответе.

На веб-сервере будет использоваться всего два типа `http`-запросов:

- Get – для получения данных;

- Post – для изменения данных либо запроса данных с передачей большого количества параметров.

Общие схемы отправки и обработки Get и Post запросов приведены на рисунках 3.6 и 3.7 соответственно.

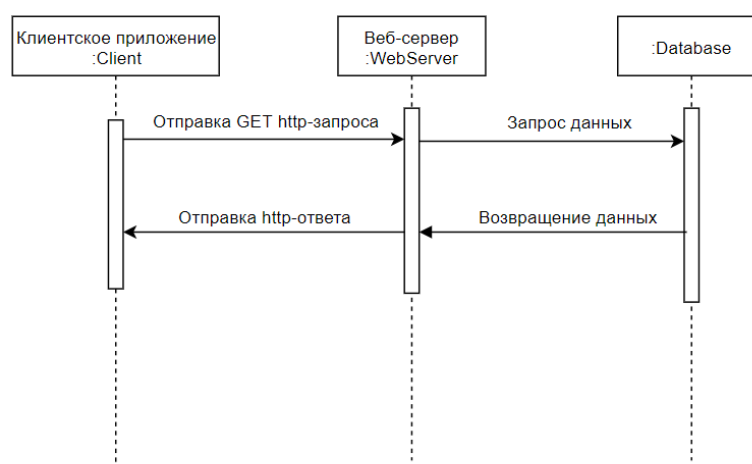


Рисунок 3.6 – Общая схема отправки и обработки Get запроса

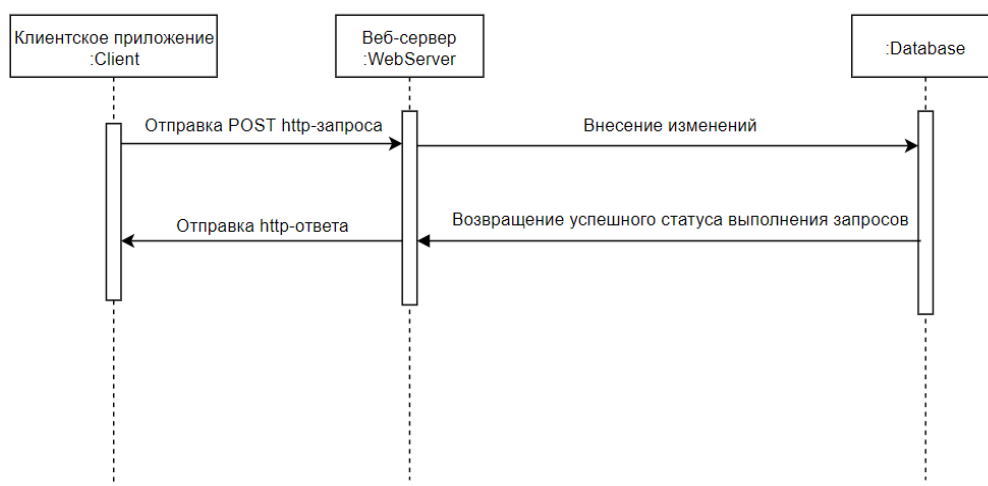


Рисунок 3.7 – Общая схема отправки и обработки POST запроса на изменение данных

4 РЕАЛИЗАЦИЯ СЕРВЕРНОЙ ЧАСТИ

4.1 Обоснование выбора инструментов и технологий

4.1.1 Среда разработки и язык программирования

Существует всего один инструмент, который максимально интегрирован с данной платформой и позволяет использовать все ее возможности – Visual studio.

Microsoft Visual Studio – линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense (умные подсказки). Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

Кроме Visual studio существует еще один инструмент, позволяющий создавать веб-приложения для платформы .Net – среда разработки Rider.

Rider позволяет разрабатывать приложения для .NET Framework, .NET Core и Mono, в том числе .NET-сервисы и библиотеки, игры на движке Unity, кросс-платформенные мобильные Xamarin-приложения, веб-приложения ASP.NET и ASP.NET Core. В нем также присутствуют основные компоненты улучшающий процесс разработки: интеллектуальный редактор, статический анализ кода и автоматическое исправление обнаруженных проблем, встроенный отладчик, интеграция с системами контроля версий и многое другое. Но данный инструмент имеет несколько существенных недостатков по отношению к Visual studio:

1) несмотря на поддержку последних версий .Net данная среда разработки не обладает всеми возможностями, необходимыми для создания полноценного веб-приложения. Для этого приходится ставить сторонние расширения, которые в свою очередь не всегда надежны;

2) чтобы использовать данный инструмент необходимо оформить платную подписку;

3) часто возникают проблемы при отладке кода.

Таким образом, для разработки сервера была выбрана среда разработки Visual Studio 2017.

В качестве целевого языка программирования был выбран C#.

C# (произносится си шарп) — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах как язык разработки приложений для платформы Microsoft .NET Framework. Впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270 [6].

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Данный язык программирования является основным при разработке для .Net и позволяет использовать все возможности платформы.

4.1.2 Выбор сервера базы данных

Выбор базы данных в качестве хранилища для данных обусловлен множеством требований, среди которых: одновременный доступ к данным, безопасность, авторизация, необходимость хранить большие объемы данных, необходимость структуризации данных [7]. Чтобы обеспечить все описанные требования необходимо выбрать подходящую систему управления базами данных. Кроме того следует учитывать некоторые дополнительные требования к СУБД:

- поддержка транзакций – чтобы в случае некоего сбоя в системе, была возможность отменить предыдущие изменения, выполненные в базе данных;

- производительность – запросы в базу должны производиться достаточно быстро, до того как истечет время ожидания ответа от веб-сервера на клиентском приложении;

- простота интеграции – для СУБД должны существовать технологии, обеспечивающие простой способ получения доступа в базу данных из веб-сервера;

- интеграция с .Net.

В качестве СУБД для системы управления проектами «Zigo» была выбрана MSSQL так как данная СУБД полностью удовлетворяет всем вышеизложенным критериям.

Microsoft SQL Server — система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основным используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями [8]. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

SQL Server характеризуется такими особенностями как:

- производительность. SQL Server работает очень быстро;
- надежность и безопасность. SQL Server предоставляет шифрование данных;
- простота. С данной СУБД относительно легко работать и вести администрирование.

Центральным аспектом в MS SQL Server, как и в любой СУБД, является база данных. База данных представляет хранилище данных, организованных определенным способом. Нередко физически база данных представляет файл на жестком диске, хотя такое соответствие необязательно. Для хранения и администрирования баз данных применяются системы управления базами данных (database management system) или СУБД (DBMS). И как раз MS SQL Server является одной из такой СУБД [9].

Для организации баз данных MS SQL Server использует реляционную модель. Эта модель баз данных была разработана еще в 1970 году Эдгаром Коддом. А на сегодняшний день она фактически является стандартом для организации баз данных.

Реляционная модель предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта.

Для идентификации каждой строки в рамках таблицы применяется первичный ключ (primary key). В качестве первичного ключа может выступать один или несколько столбцов. Используя первичный ключ, мы можем ссылаться на определенную строку в таблице. Соответственно две строки не могут иметь один и тот же первичный ключ.

Через ключи одна таблица может быть связана с другой, то есть между двумя таблицами могут быть организованы связи. А сама таблица может быть представлена в виде отношения ("relation").

Для взаимодействия с базой данных применяется язык SQL (Structured Query Language). Клиент отправляет запрос на языке SQL посредством специального API. СУБД выполняет запрос, а затем посылает клиенту результат выполнения.

Изначально язык SQL был разработан в компании IBM для системы баз данных, которая называлась System/R. Хотя в итоге ни база данных, ни сам язык не были впоследствии официально опубликованы, по традиции сам термин SQL нередко произносят как "сиквел".

4.2 Реализация основных модулей сервера

В процессе разработки серверной части активно использовался принцип модульности, позволяющий декомпозировать программное обеспечение на отдельные модули (физически это файлы dll).

Модульный принцип — принцип построения технических систем, согласно которому функционально связанные части группируются в законченные узлы — модули (блоки). Модульность устройства позволяет изменять его возможности, путём использования/наращивания функциональных блоков, выполняющих различные задачи

В процессе разработки была сформирована определенная архитектура из модулей, каждый из которых несет определенную ответственность. Разработанные модули представлены на рисунке 4.1.

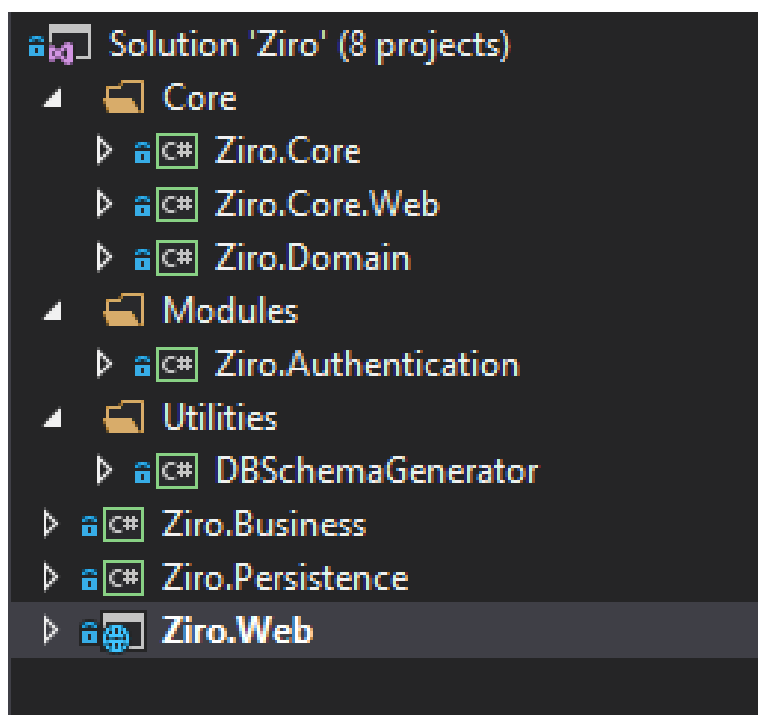


Рисунок 4.1 – Модульная структура сервера

Описание модулей сервера:

1) Ziro.Core и Ziro.Core.Web – содержат в себе все необходимые интерфейсы для работы приложения и неизменные настройки, необходимые для работы приложения;

2) `Ziro.Domain` – содержит классы, описывающие все основные сущности предметной области. Данные классы используются для сопоставления данных полученных из одноименных таблиц;

3) `Ziro.Authentication` – содержит механизмы авторизации и аутентификации, срабатывающие при каждом запросе на сервер;

4) `DBSchemaGenerator` – вспомогательный модуль, предназначенный для автогенерирования схемы базы данных;

5) `Ziro.Persistence` – отвечает за взаимодействие с базой данных, содержит в себе специальные классы-репозитории, которые запрашивают и изменяют данные для одной из таблиц в базе данных;

6) `Ziro.Business` – содержит классы-сервисы отвечающие за обработку данных с использованием классов-репозиторий и других классов, для обеспечения получения необходимых данных в нужном виде;

7) `Ziro.Web` – ASP.NET Core приложение, отвечает за обработку веб-запросов и вызов определенных классов-сервисов для обработки данных.

4.2.1 Модуль обработки веб-запросов

Самым главным модулем в проекте сервера является `Ziro.Web`, который по своей сути и является веб-сервером.

Данный модуль в своей основе использует концепцию MVC.

Платформа ASP.NET MVC представляет собой фреймворк для создания сайтов и веб-приложений с помощью реализации паттерна MVC.

Концепция шаблона MVC (model-view-controller) предполагает разделение приложения на три компонента:

- контроллер (controller) представляет класс, обеспечивающий связь между пользователем и системой, представлением и хранилищем данных. Он получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления;

- представление (view) - это собственно визуальная часть или пользовательский интерфейс приложения. Как правило, html-страница, которую пользователь видит, зайдя на сайт;

- модель (model) представляет класс, описывающий логику используемых данных.

Модель является независимым компонентом - любые изменения контроллера или представления не затрагивают модель. Контроллер и представление являются независимыми компонентами, и поэтому их можно изменять независимо друг от друга.

Общая схема взаимодействия модели контроллера и представления представлена на рисунке 4.2.

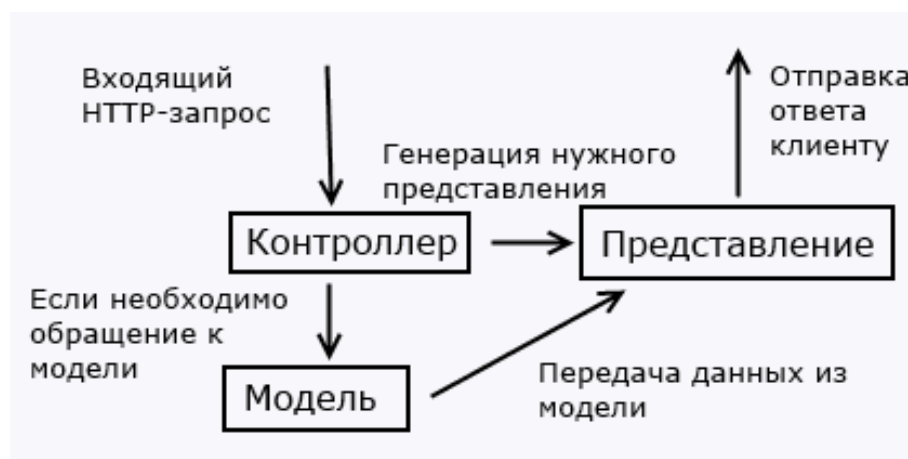


Рисунок 4.2 – Схема взаимодействия компонентов MVC

В ответ на каждый запрос с клиентского приложения, сервер создает определенный объект контроллера и вызывает в нем определенный метод, называемый метод-действие. Таким образом, совокупность всех методов действий всех контроллеров, определенных на сервере составляют API сервера.

При этом, для обработки запроса сервер отбирает нужный контроллер с нужным методом действия если соблюдены все нижеперечисленные условия:

1) адрес назначения, указанный в запросе соответствует шаблону “api/имя_контроллера/имя_метода_действия”. Например, для адреса api/User/GetUsers сервер создать экземпляр класса UserController и вызовет метод GetUsers для обработки запроса;

2) тип запроса должен совпадать с типом метода-действия, т.е. если тип запроса POST, то должен существовать метод-действия помеченный атрибутом POST (и соответствующий предыдущему пункту);

3) формат и количество параметров переданных в запросе должно совпадать с форматом и количеством входных параметров метода-действия.

Но прежде, чем конкретный метод действия начнет обработку запроса, сработают модули авторизации и аутентификации, которые проверяют наличие необходимых прав пользователя, указанного в запросе, вызвать данный метод действия.

Если запрос прошел проверки на авторизацию и аутентификацию, то создаются необходимые сервисы для обработки запроса, которые в свою очередь могут воспользоваться классами-репозиториями для выполнения необходимых запросов к базе данных для изменения имеющихся данных либо получения необходимых данных для формирования ответа для клиента.

В общем виде, конвейер обработки конкретного запроса можно представить в виде схемы, определенной на рисунке 4.3.

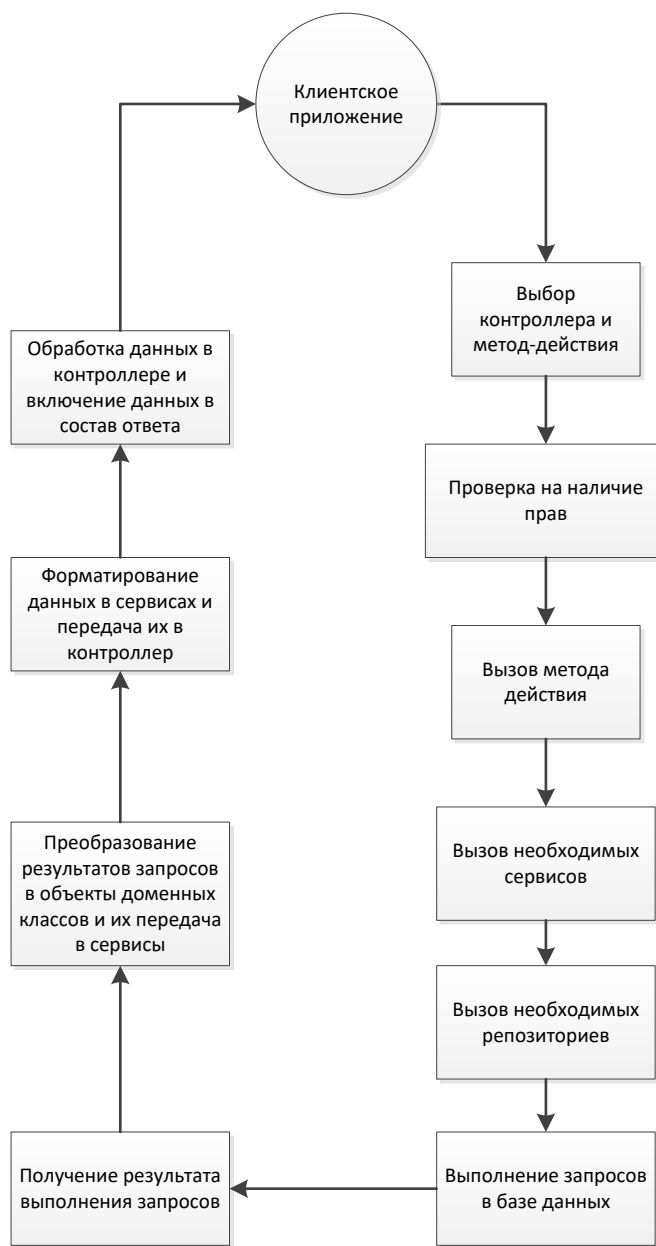


Рисунок 4.3 – конвейер обработки запросов на веб-сервере

Стоит отметить, что в качестве формата данных, используемых в запросах и ответах, используется JSON.

JSON – текстовый формат обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается людьми. Формат JSON был разработан Дугласом Крокфордом.

JSON-текст представляет собой (в закодированном виде) одну из двух структур:

– набор пар ключ: значение. В различных языках это реализовано как запись, структура, словарь, хеш-таблица, список с ключом или ассоциативный массив.

Ключом может быть только строка (регистрозависимая: имена с буквами в разных регистрах считаются разными), значением — любая форма;

– упорядоченный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность.

Это универсальные структуры данных: как правило, любой современный язык программирования поддерживает их в той или иной форме. Они легли в основу JSON, так как он используется для обмена данными между различными языками программирования.

В качестве значений в JSON могут быть использованы:

– запись — это неупорядоченное множество пар ключ:значение, заключённое в фигурные скобки «{ }». Ключ описывается строкой, между ним и значением стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми;

– массив (одномерный) — это упорядоченное множество значений. массив заключается в квадратные скобки «[]». Значения разделяются запятыми;

– число;

– литералы true, false и null;

– строка — это упорядоченное множество из нуля или более символов юникода, заключённое в двойные кавычки. Символы могут быть указаны с использованием escape-последовательностей, начинающихся с обратной косой черты «\» (поддерживаются варианты \', \", \\, \/, \t, \n, \r, \f и \b), или записаны шестнадцатеричным кодом в кодировке Unicode в виде \uFFFF.

Пример объекта, преобразованного в JSON данные, представлен на рисунке 4.4.

```
{
  "firstName": "Иван",
  "lastName": "Иванов",
  "address": {
    "streetAddress": "Московское ш., 101, кв.101",
    "city": "Ленинград",
    "postalCode": "101101"
  },
  "phoneNumbers": [
    "812 123-1234",
    "916 123-4567"
  ]
}
```

Рисунок 4.4 – JSON данные

Данный формат был выбран в силу своей простоты преобразования объектов C# в их JSON представление (сериализация в JSON) и обратного преобразования из

JSON в C# объект (десериализация). Для этого была использована специальная библиотека `Newtonsoft.JSON`, которая содержит весь необходимый функционал по сериализации и десериализации JSON данных.

4.2.2 Механизм авторизации и аутентификации

Механизм авторизации и аутентификации основан на использовании «куки».

Куки – это строка зашифрованных данных, содержащая в себе идентификационные данные пользователя? от имени которого посылаются запросы на сервер. Также в куках находится информация о правах, которыми обладает пользователь в отношении сервера.

На сервере разработан специальный контроллер `AccountController`, который в методе-действия `Login` принимает два параметра – почтовый адрес и пароль пользователя. Если клиентское приложение передаст в данный метод действия почтовый адрес и пароль существующего пользователя, то сервер сформирует объект-куки, подложив туда идентификационную информацию о пользователе и его роль. Далее данный объект преобразуется в строку, шифруется и записывается в секцию заголовков ответа. Клиентское приложение принимает ответ, и сохраняет сформированные на сервере куки в своем локальном хранилище. После этого клиентское приложение отправляет сохраненные куки при каждом запросе. В свою очередь сервер при каждом запросе извлекает куки из него, расшифровывает их и получает сведения о пользователе, который произвел данный запрос. Эти данные используются сервером перед вызовом конкретного метода-действия, чтобы удостовериться что пользователь имеет право обратиться к данному методу действия. При этом сервер руководствуется атрибутами, которые назначаются каждому методу действия. Следующие атрибуты могут быть использованы на сервере и назначаться методам действия:

1) `AllowAnonymous` – означает, что любой пользователь может воспользоваться данным методом-действия, даже если не установлены никакие данные кук;

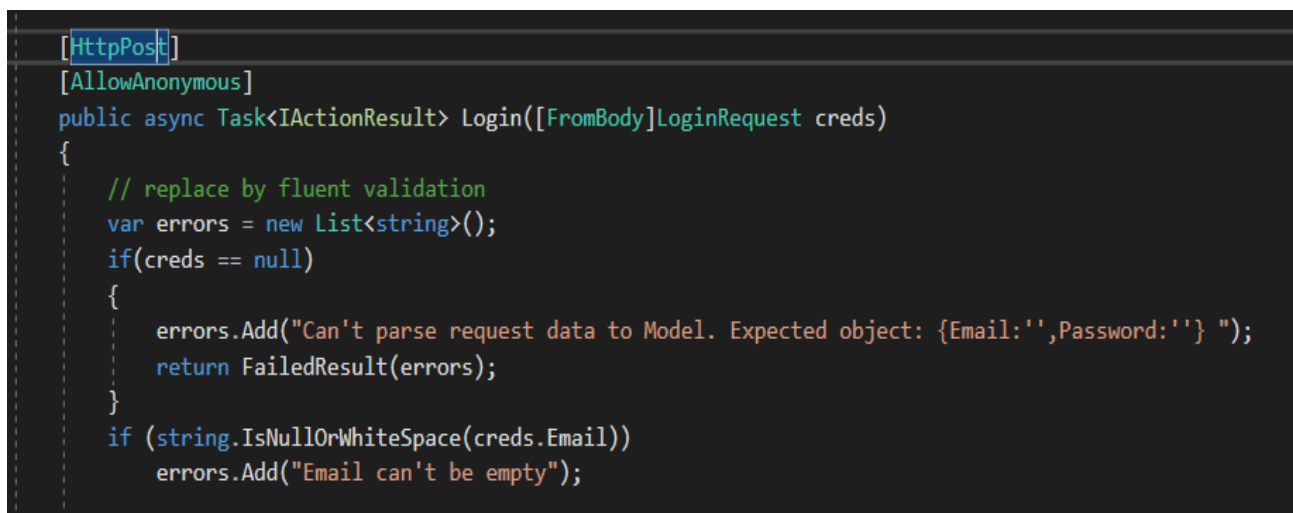
2) `Authorize` – пользователь должен как минимум быть аутентифицирован в системе, чтобы он мог воспользоваться данным методом действия;

3) `Authorize(список_ролей)` – пользователь должен быть аутентифицирован и иметь одну из ролей, указанных в списке ролей атрибута. При этом роли должны быть перечислены через запятую или точку с запятой. Доступные значения: «User» и «Administrator».

Перечисленные атрибуты должны указываться над методом действия для применения соответствующих правил авторизации и аутентификации, также данные атрибуты могут назначаться контроллерам – это будет означать то, что данный атрибут будет применяться ко всем методам действиям данного контроллера. При

этом атрибуты указанные для методов действий переопределяют атрибут указанный в контроллере.

Пример использования атрибута AllowAnonymous для метода действия Login представлен на рисунке 4.5



```
[HttpPost]
[AllowAnonymous]
public async Task<IActionResult> Login([FromBody]LoginRequest creds)
{
    // replace by fluent validation
    var errors = new List<string>();
    if(creds == null)
    {
        errors.Add("Can't parse request data to Model. Expected object: {Email:'',Password:''} ");
        return FailedResult(errors);
    }
    if (string.IsNullOrEmpty(creds.Email))
        errors.Add("Email can't be empty");
}
```

Рисунок 4.5 – Пример использования атрибута авторизации в методе действия

Если пользователь не прошел проверку на аутентификацию или авторизацию в системе, то запрашиваемый метод действия выполняться не будет, а пользователю вернется ответ с ошибкой 403 или 401 и сообщением об ошибке.

При необходимости удаления данных о пользователе на клиентском приложении необходимо обратиться к методу действия Logout в контроллере AccountController.

4.2.3 Модуль доступа к данным

Для взаимодействия с базой данных был разработан специальный модуль Ziro.Persistance. Этот модуль использует специальные классы-репозитории для выполнения запросов к базе данных по отношению к определенной таблице или представлению. Но в самих репозиториях нет никакого sql кода и упоминания базы данных, с которой они работают. Однако присутствует код по работе со специальным классом сессией, а также специальные выражения (select, where и др.).

На самом деле репозитории абстрагированы от конкретной базы данных, с которой они должны работать, они лишь манипулируют объектами классов, которые в точности по своей структуре напоминают таблицы в базе данных, и объектом сессии, для доступа к конкретной базе данных. А все специальные C#-выражения (select, where и др.) автоматически транслируются в запросы к определенным таблицам, а результаты данных запросов автоматически переводятся в объекты конкретных классов, с которыми связаны таблицы базы данных.

Пример класса-репозитория, осуществляющий выполнение запросов к таблице пользователей, представлен на рисунке 4.6.

```
public class UserRepository : IUserRepository
{
    private readonly ISession _session;

    public UserRepository(ISession session)
    {
        _session = session;
    }

    public User GetUser(string email, string password)
    {
        var query = _session.QueryOver<User>().Where(x => x.Email == email && x.PasswordHash == password);
        var result = query.SingleOrDefault();
        return result;
    }
}
```

Рисунок 4.6 – Класс-репозиторий пользователей

Такой способ взаимодействия с базой данных обусловлен использованием двух технологий: Nhibernate и LINQ-запросы.

Nhibernate – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Данная технология ориентирована на платформу Microsoft .NET.

LINQ (Language-Integrated Query) представляет простой и удобный язык запросов к источнику данных. В качестве источника данных может выступать объект, реализующий интерфейс IEnumerable (например, стандартные коллекции, массивы), набор данных DataSet, документ XML. Но вне зависимости от типа источника LINQ позволяет применить ко всем один и тот же подход для выборки данных.

Таким образом, Nhibernate позволяет взаимодействовать с определенной базой данных через объект Session, выполнять запросы к определенной таблице с помощью QueryOver<>, где в угловых скобках нужно указать типа доменного класса, который связан с конкретной таблицей, и сформировать сам запрос с помощью выражений LINQ, который синтаксически напоминает SQL выражение.

Объект Session – главный компонент Nhibernate, он позволяет установить соединение с базой данных, и получить объект QueryOver, с помощью которого можно выполнять запросы в базе данных. Также данный объект позволяет работать с транзакциями. Для того чтобы связать объект Session с определенной базой данных необходимо создать и задать настройки для другого объекта – SessionFactory, который в свою очередь создает объекты сессий. На протяжении работы приложения экземпляр SessionFactory создается и конфигурируется всего один раз и происходит

это в главном модуле приложения на старте веб-сервера. Конфигурация объекта SessionFactory представлена на рисунке 4.7.

```
//nhibernate
services.AddSingleton<ISessionFactory>(x =>
{
    var connectionString = x.GetService<ISystemSettings>().ConnectionString;
    var nhConfiguration = new Configuration().DataBaseIntegration(db =>
    {
        db.ConnectionString = connectionString;
        db.Dialect<MsSql2012Dialect>();
        db.Driver<Sql2008ClientDriver>();
    })
    .SetNamingStrategy(new ZiroNamingStrategy());
    var s = new ConventionModelMapper();
    var mapper = new ModelMapper();
    mapper.AddMappings(Assembly.GetAssembly(typeof(UserMap)).GetExportedTypes());
    var mapping = mapper.CompileMappingForAllExplicitlyAddedEntities();
    nhConfiguration.AddMapping(mapping);
    return nhConfiguration.BuildSessionFactory();
});
services.AddScoped<NHibernate.ISession>(x => x.GetService<ISessionFactory>().OpenSession());
```

Рисунок 4.7 – Конфигурация подключения к базе данных

На рисунке выше представлен процесс конфигурации и встраивания объектов SessionFactory и Session в механизм обработки запросов. Данный код выполняется всего один раз на старте приложения и в процессе него происходит следующее:

1) в конвейер обработки добавляется SessionFactory в единственном экземпляре, при этом в момент создания для него указывается строка подключения к базе данных, драйвер и диалект базы данных, добавляются все классы-сопоставления. Объект данного класса создается всего один раз и уничтожается вместе с завершением работы самого веб-сервера;

2) в конвейер обработки запроса встраивается Session, который создается другим объектом SessionFactory всякий раз, когда приходит новый запрос на обработку. Данный объект создается в момент прихода нового запроса, передается в репозиторий и уничтожается в момент отправки ответа.

Как уже отмечалось в предыдущих главах, в проекте сервера существует модуль Ziro.Domain, где находятся доменные классы, связанные с одноименными таблицами в базе данных. Физически, доменный класс представляет собой обычный класс с набором виртуальных свойств, при этом каждое такое свойство соответствует определенному столбцу с таблицей базы данных. Обычно имя такого класса соответствует названию таблицы, с которой он связан, а имена свойств соответствуют названиям столбцов в таблице. Nhibernate использует эти классы для сопоставления результатов запросов к таблицам к их объектным представлениям. Для того чтобы

связать доменный класс с конкретной таблицей, используются специальные классы-сопоставления, где конкретный класс связывается с определенной таблицей в базе. Пример структуры доменного класса Position и его класса-сопоставления представлены на рисунках 4.8 и 4.9 соответственно.

```
public class Position
{
    public virtual Guid Id { get; set; }
    public virtual string Name { get; set; }
    public virtual ISet<User> Users { get; set; }
}
```

Рисунок 4.8 – Структура доменного класса Position

```
public class PositionMap : BaseEntityMap<Position>
{
    public PositionMap()
    {
        Id(x => x.Id, m => m.Generator(Generators.Guid));

        Property(x => x.Name, m =>
        {
            m.Length(400);
            m.NotNullable(notnull: true);
        });

        Set(x => x.Users,
            c => {
                c.Key(k => k.Column(FKColumnName(nameof(Position))));
                c.Inverse(true);
            },
            r => r.OneToMany());
    }
}
```

Рисунок 4.9 – Сопоставление класса Position с таблицей в базе данных

Данный класс сопоставления можно описать следующим способом: для класса Position Должна существовать таблица Position с тремя полями:

- 1) Id – уникальный идентификатор в таблице Position с типом uniqueIdentifier, поле не может содержать значения null;
- 2) Name – строковое поле типа varchar2 с максимальной длиной в 400 символом, значения null для данного поля допустимы;

3) Users – виртуальное поле, которое предназначено для посылания моментального запроса в базу данных за всеми пользователями, ассоциирующимися с текущей должностью.

Также этот класс указывает на то, что таблица Position связана с таблицей User отношением «один ко многим» и должно существовать поле PositionId в таблице User, как внешний ключ к таблице Position.

Таким образом, для доступа к базе данных используются классы репозитории, которые в свою очередь подключаются к базе данных с помощью объекта Session и выполняют определенные запросы через объект QueryOver<TClass>, где TClass – определенный доменный класс, связанный с определенной таблицей в БД. При этом доменные классы связываются с определенными таблицами посредством файлов сопоставления, а результаты запросов транслируются в объекты доменных классов автоматически с помощью механизмов Nhibernate.

Кроме QueryOver имеется также ряд других способов сгенерировать запрос в базу данных, например Hql и Criterion, но данные технологии считаются устаревшими. Также Nhibernate позволяет выполнить определенный SQL запрос напрямую в базе данных, для этого необходимо вызвать метод CreateSQL у объекта сессии и передать в него SQL запрос в строковом виде.

4.3 Разработка базы данных

В соответствии с требованиями постановки задачи, были определены следующие сущности в системе:

- 1) пользователь (User) – хранит в себе данные учетной записи для конкретного пользователя;
- 2) аватар (Avatar) – фотография или любое другое изображение, отображающееся в профиле пользователя, комментариях и других частях системы;
- 3) должность (Position) – должность работника;
- 4) журнал работ (LogWork) – информация о том, какие работы были произведены и сколько времени на это потребовалось в отношении конкретной задачи;
- 5) проект (Project) – информация о проекте (имя, краткое имя и описание);
- 6) задача (Task) – информация по конкретной задаче (название, описание, номер, автор, исполнитель, затраченное время, статус задачи, тип задачи, дата создания, дата последнего обновления);
- 7) документ проекта (ProjectDocument) – файл документа, принадлежащий конкретному проекту;
- 8) комментарий (Comment) – олицетворяет собой определенный комментарий оставленный определенным пользователем для конкретной задачи.

Для каждой из сущностей были созданы доменные объекты в модуле Ziro.Domain, которые представлены на рисунке 4.10

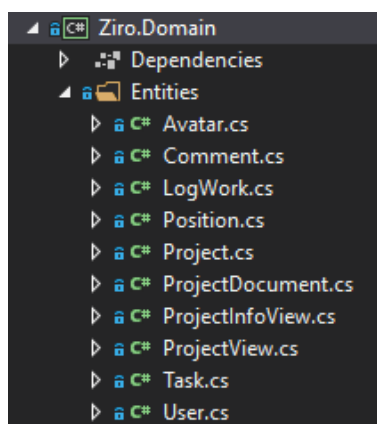


Рисунок 4.10 – Доменные классы

Для каждого доменного класса были созданы классы сопоставления с таблицами базы данных.

Для создания схемы базы данных был разработан специальный модуль DBSchemaGenerator, который использует Nhibernate и разработанные классы-сопоставления для генерации схемы базы данных в виде sql-скрипта, который называется ZiroDB.sql. Данный скрипт создает все основные таблицы, устанавливает связи между ними и накладывает необходимые ограничения. Кроме данного скрипта, создающего все основные таблицы, существует дополнительный скрипт ZiroDB_Views, который необходим для добавления двух представлений, и скрипт ZiroDB_DBMigrations, который необходим для создания миграционной таблицы.

Миграционная таблица необходима для ведения учета выполненных миграционных sql-скриптов на базе данных. Миграционный скрипт в свою очередь содержит в себе необходимый sql для изменения схемы для базы данных, при этом в конце своего исполнения он заносит свой номер в таблицу DBMigrations. Занесенный номер в свою очередь представляет собой текущую версию схемы базы данных, а также не дает выполняться любым миграционным скриптам, которые уже занесли свой номер в эту таблицу.

В результате выполнения всех этих скриптов получится схема базы данных, используемая веб-сервером системы управления Ziro, которая представлена в приложении.

5 РАЗВЕРТЫВАНИЕ И ТЕСТИРОВАНИЕ

5.1 Развертывание серверной части на облачной платформе

Существует несколько различных способов развертывания веб-сервера и базы данных, среди них:

- 1) использование собственной машины и IIS;
- 2) использование веб хостинг-сервисов;
- 3) использование облачных платформ.

Для первого способа придется иметь свой собственный сервер, свою команду по поддержке этого сервера, статический IP-адрес и настроенный IIS.

IIS (Internet Information Services, до версии 5.1 — Internet Information Server) — набор серверов для нескольких служб Интернета от компании Microsoft. IIS распространяется с Windows NT.

Основным компонентом IIS является веб-сервер, который позволяет размещать в Интернете сайты. IIS поддерживает протоколы HTTP, HTTPS, FTP, POP3, SMTP, NNTP.

Первый способ имеет ряд недостатков:

- 1) необходимость закупки серверов;
- 2) необходимо иметь специально обученный персонал по поддержке серверов, их настройке и др.;
- 3) при необходимости увеличить вычислительные мощности, необходимо произвести ряд долгосрочных действий.

Использование веб хостинг-сервисов позволяет использовать сервера другой организации и вся ответственность по обеспечению работы веб-сервера будет лежать на них. Веб хостинг сервис это тип интернет сервиса, который позволяет пользователям сделать сайт доступным для всех, используя всемирную паутину.

Использование веб-хостинг сервисов позволяет избавиться от первых двух недостатков, но по-прежнему стоит вопрос о возможности конфигурации вычислительных мощностей и возможности добавления дополнительных серверов в систему.

Третий способ самый оптимальный в силу того, что облачная платформа выступает как в роли хостинг сервиса, так и в роли сервиса для аренды вычислительных ресурсов, предлагая все возможности по конфигурированию и масштабированию системы. При этом стоимость услуг будет пропорционально количеству и качеству арендованных серверов в облаке. Таким образом, наилучшим способом разворачивания веб-сервера Ziro будет использование облачной платформы. При этом среди большого количества облачных платформ была выбрана платформа Azure в силу ряда причин:

- 1) простота конфигурации серверов;
- 2) низкая стоимость аренды вычислительных мощностей (первый месяц использования – бесплатный);
- 3) автоматизация разворачивания новых версий системы на веб-серверах;
- 4) полная поддержка .Net платформы и всех ее самых новых версий.

Данная платформа использовалась для тестирования системы всеми участниками команды разработки данной системы. Планируется, что данная платформа будет использоваться и в то время, когда система будет на этапе эксплуатации.

Для того чтобы развернуть веб-сервер и базу данных первый раз на облачной платформе нужно выполнить ряд шагов, рассмотренных далее.

Сначала необходимо зарегистрироваться на сайте <https://azure.microsoft.com> и указав все данные, включая способ оплаты, личные данные владельца аккаунта и т.д. Форма регистрации представлена на рисунке 5.1.

Free Trial
Learn more ▾

1 About you

FIRST NAME: dfvsdfv LAST NAME: sdfvsdfv COUNTRY/REGION: Ukraine
CONTACT EMAIL: azure@setevoy.kiev.ua
COMPANY/SCHOOL: dsdsv WORK PHONE: 5556 644

2 Verification by phone COMPLETE

3 Verification by card

This information is collected only to verify your identity. You will not be charged unless you explicitly upgrade to a paid offer.
You might see a temporary hold on your bank card statement. Your card will not be charged.

PAYMENT METHOD: New Credit/Debit Card

CARD NUMBER: ADDRESS LINE 1:
CARD TYPE: Visa EXPIRATION DATE: MM/YY ADDRESS LINE 2: Optional -
CVV: CITY:
NAME ON CARD: STATE: POSTAL CODE: 01032 -
PHONE NUMBER: Prefix - Number -

4 Agreement

☒ I agree to the [subscription agreement](#), [offer details](#), and [privacy statement](#).
☒ Microsoft may use my email and phone to provide special Microsoft Azure offers.

Sign up →

Рисунок 5.1 – Регистрация нового аккаунта Azure

После этого откроется панель управления облачными ресурсами, где можно управлять всеми серверами, их конфигурациями, создавать новые ресурсы, просматривать статистику использования серверов, получать информацию о

денежных расходах за использование облачных ресурсов и многое другое. Данная панель представлена на рисунке 5.2.

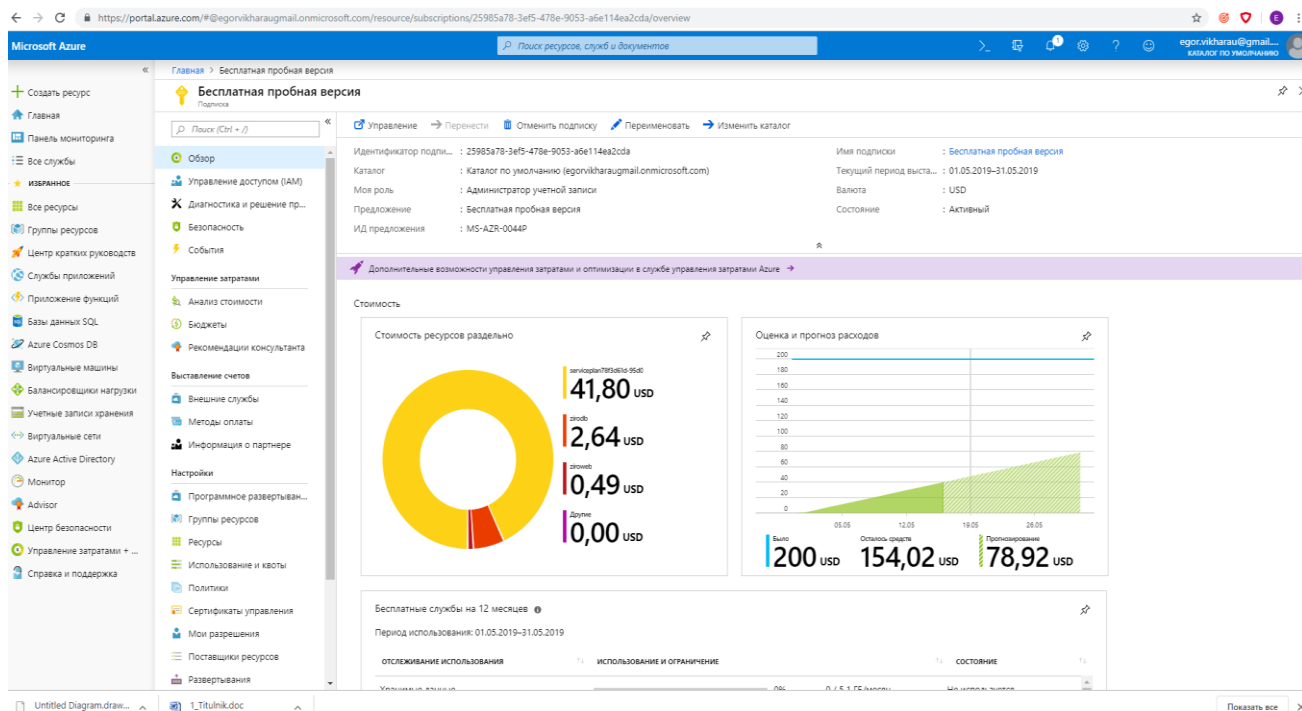


Рисунок 5.2 – Панель управления облачными ресурсами

Теперь необходимо создать два виртуальных сервера: сервер базы данных и веб-сервер. Для этого нужно переместиться на левую панель и там выбрать пункт «Все ресурсы». Далее необходимо нажать кнопку «Добавить», которая будет находиться в верхнем правом углу. После этого откроется окно создания нового ресурса, на котором нужно будет выбрать типа ресурса, вписать его имя. Далее, в зависимости от типа выбранного ресурса необходимо будет прописать дополнительные настройки. На данном этапе необходимо создать три ресурса:

1) сервер базы данных MSSQL – указав имя сервера, регион размещения и версию MSSQL, а также при необходимости можно указать максимальный объем данных баз данных, которые будут там находиться (чем больше максимальный объем будет указан, тем больше будет стоимость аренды данного сервера базы данных на облачной платформе);

2) база данных – указав имя базы данных, учетную запись администратора и имя сервера из предыдущего пункта. После добавления базы данных будет автоматически создана строка подключения к ней. Данная строка будет использоваться разработанным веб-сервером для подключения к базе данных;

3) веб-сервер – указав имя веб-сервера, доменное имя сайта веб-сервера доступное для пользователей в Интернете, платформа веб-сервера (в данном случае необходимо выбрать версию ASP.NET Core).

В результате выполнения перечисленных шагов во вкладке «Все ресурсы» будут отображены облачные ресурсы так, как показано на рисунке 5.3.

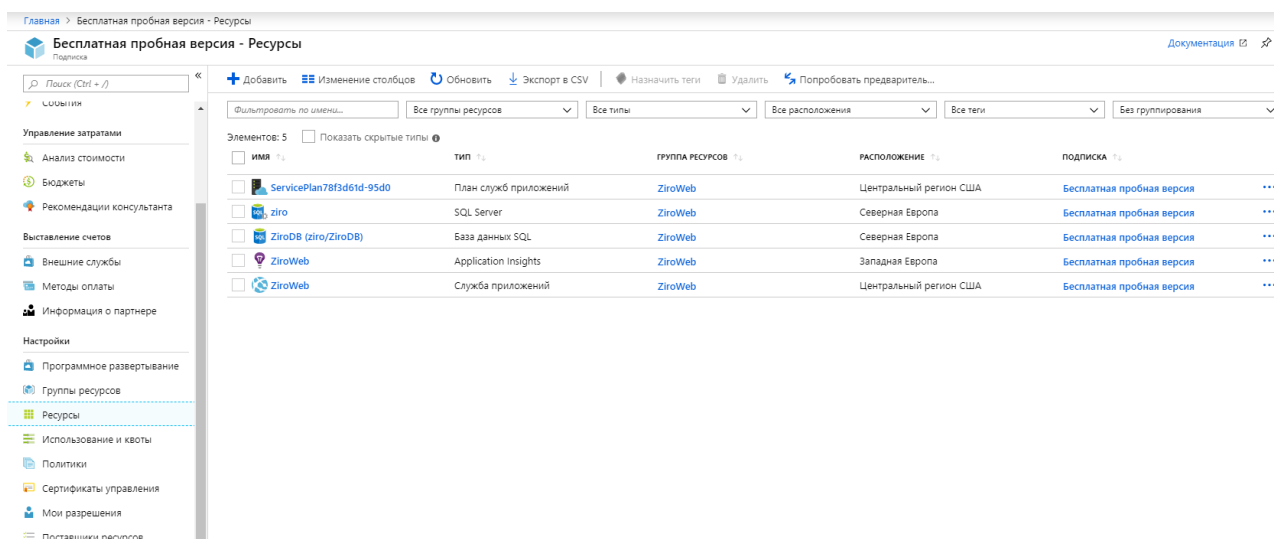


Рисунок 5.3 – Облачные ресурсы системы Ziro

На данном этапе у нас уже развернут веб-сервер, который по запросу возвращает некоторую страницу, определенную по умолчанию, а также пустая база данных.

Теперь необходимо создать все необходимые таблицы в базе данных, для этого нужно перейти в ресурс базы данных, по нажатию на базу данных в списке, представленном на рисунке 5.4. Потом необходимо выбрать редактор запросов. Далее откроется окно выполнения запросов, представленное на рисунке 5.4.

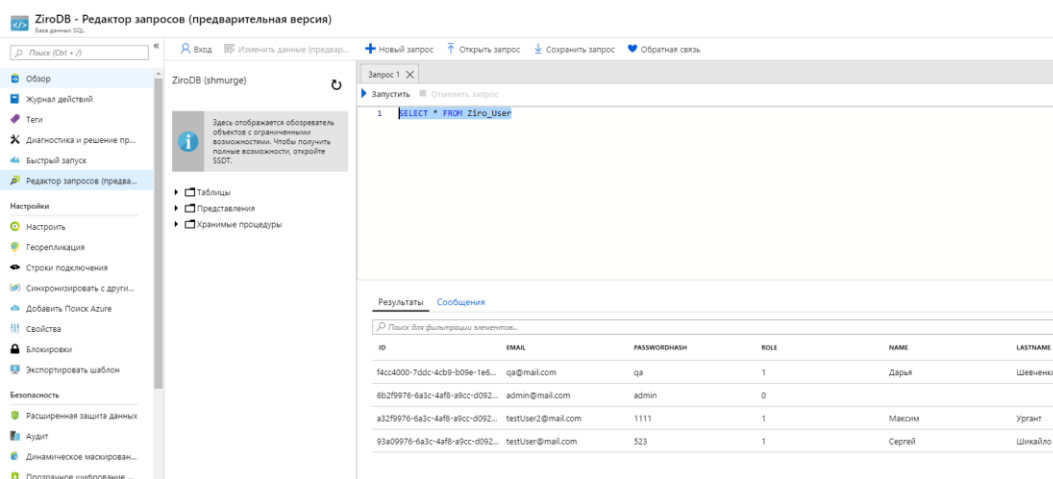


Рисунок 5.4 – Редактор запросов базы данных

Остается только последовательно выполнить содержимое скриптов ZiroDB.sql, ZiroDB_DBMigrations и ZiroDB_Views для добавления в базу всех необходимых структур. Таким образом, можно выполнять любой запрос в облачную базу данных.

Теперь остается только поместить программное обеспечение серверной части для созданного облачного веб-сервера. При этом процесс загрузки файлов веб-приложения на определенный веб-сервер называется публикацией. Для публикации разработанного программного обеспечения веб-сервера необходимо перейти в облачный ресурс веб-сервера, выбрав веб-сервер в списке, представленном на рисунке 5.3. После этого откроется окно управления веб-сервером, представленное на рисунке 5.5.

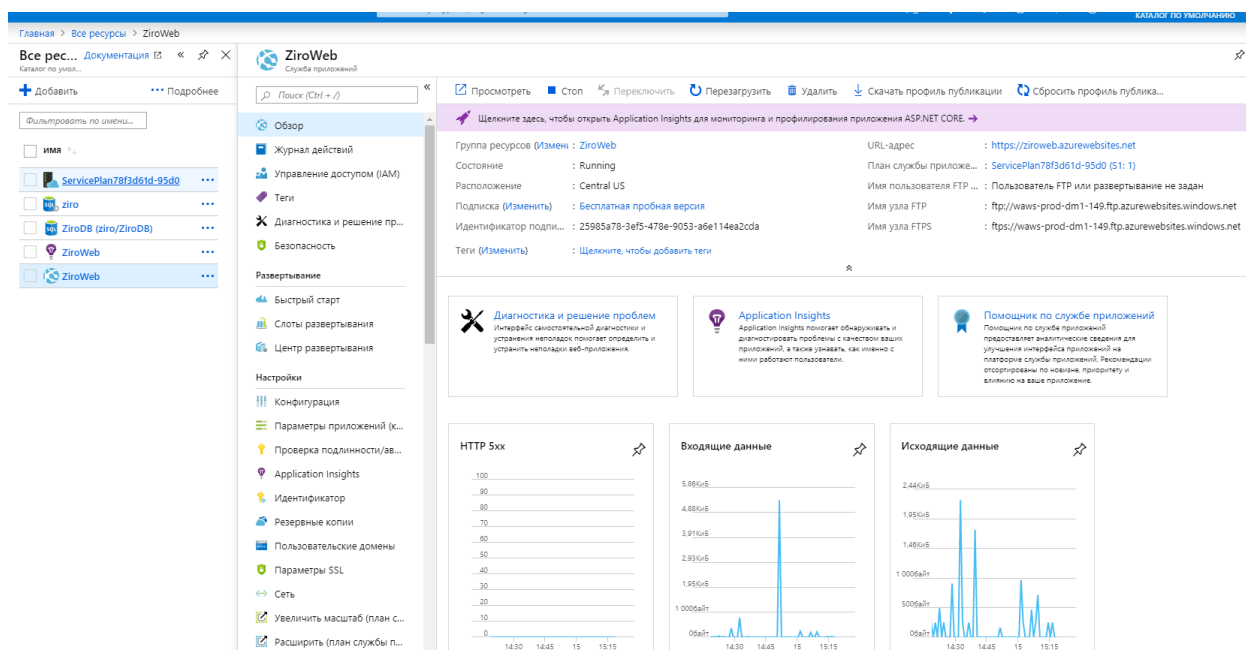


Рисунок 5.5 – Окно управления веб-сервером

Здесь можно увидеть URL-адрес, по которому можно обратиться к серверу из браузера и других клиентских веб-приложений.

Теперь необходимо нажать на кнопку «Скачать профиль публикации», после чего загрузится xml файл, который содержит в себе все необходимые настройки, для развертывания веб-сервера в облаке (адрес, по которому будет разворачиваться веб-сервер; логин и пароль для аутентификации на облачной платформе, версия платформы .Net и многое другое).

Visual Studio имеет встроенный инструмент для автоматического развертывания веб-приложения на удаленном хостинг-сервисе или облачной платформе – публикация («Publishing»). Данный инструмент позволяет использовать профили публикации для загрузки файлов веб-сервера. Для этого необходимо открыть главный проект веб-сервера в Visual Studio, нажать по нему правой кнопкой мыши и выбрать пункт Publish, после чего откроется окно, на котором необходимо будет произвести все необходимые настройки для автоматического развертывания веб-приложения в Интернете либо загрузить профиль публикации, в котором будут находиться все

необходимые настройки. Окно автоматической публикации веб-приложений среды Visual Studio представлено на рисунке 5.6.

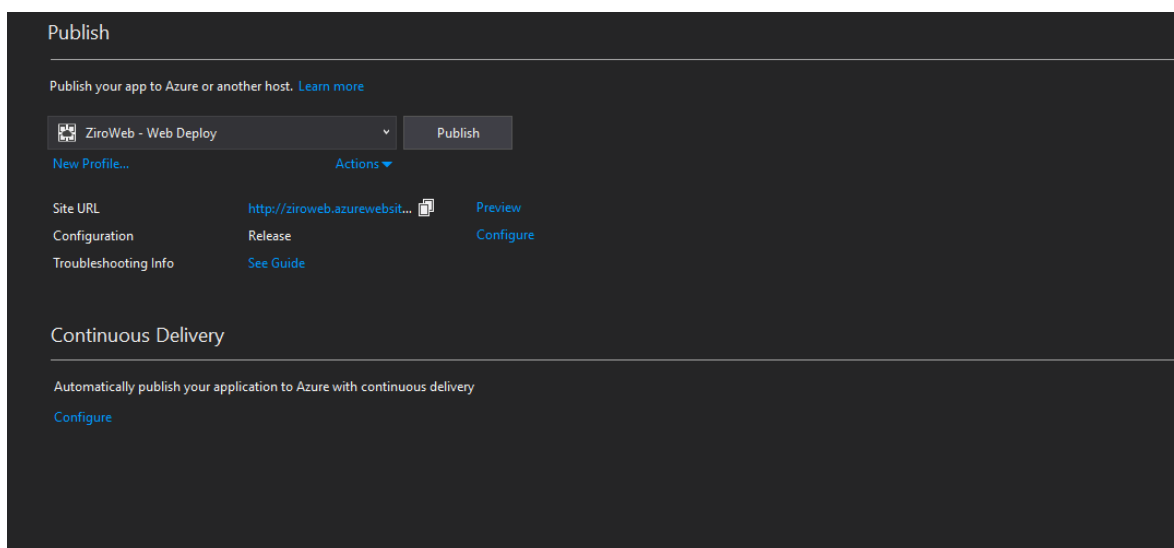


Рисунок 5.6 – окно публикации веб-приложения

На данном этапе необходимо нажать кнопку **New Profile**, после чего в открывшемся окне нажать кнопку **Import Profile**, который в свою очередь откроет стандартный проводник, в котором нужно будет найти и выбрать загруженный профиль публикации. После чего данный профиль окажется в выпадающем списке вариантов публикации. Теперь необходимо появившийся профиль и нажать кнопку **Publish**. После этого Visual studio запустит процесс публикации, состоящий из нескольких шагов:

- 1) компиляция и построения dll файлов;
- 2) загрузка dll файлов на сервер, указанный в профиле публикации;
- 3) открытие веб-сайта, для которого публиковались файлы.

После завершения публикации на облачном веб-сервере окажутся все необходимые файлы для работы системы Ziro. Каждый последующий раз, когда нужно будет поменять версию системы, используемой в облаке, на более новую, нужно будет просто перейти в Visual Studio и заново опубликовать библиотеки системы для уже сохраненного профиля публикации.

5.2 Тестирование веб-сервера

Тестирование программного обеспечения — процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определенным образом (ISO/IEC TR 19759:2005).

Тестирование программного обеспечения будет проводится в рамках функционального тестирования, разделив его на критическое и углубленное.

Тестирование программы состоит из разработки тестовых случаев (тестов), их запуска и анализа полученных результатов.

Тестовые случаи – это алгоритмы проверки функциональности программы.

Каждый тестовый случай должен обладать следующими свойствами:

- четкой целью проверки;
- известными начальными условиями тестирования;
- строго определенной средой тестирования;
- тестовыми данными и ожидаемым результатом тестирования.

Тестирование производилось в браузерах указанных в таблице 5.1.

Таблица 5.1 – Матрица конфигурации веб-приложения

	Google Chrome 74.0.3396.79	Microsoft Edge 42	Mozilla FireFox 56.x	Opera 11
Windows 10	+	+	+	+

5.2.1 Критическое тестирование

Критическое тестирование – это процесс поиска ошибок в программе при стандартной ее работе (при правильной последовательности действий, при заполнении полей только теми данными, которые предназначены для этих полей, использование функциональности приложения верным способом и т. д.).

Тест критического пути является одним из самых распространенных видов функционального тестирования, в частности, для веб-проектов. Частота данного тестирования обусловлена в первую очередь необходимостью периодической проверки всего приложения в сжатые сроки. Также данный вид тестирования позволяет выявить самые быстро находимые дефекты и исправить приложение в более сжатые сроки.

С помощью данного вида тестирования покрываются все сценарии стандартного использования приложения, исключая негативные сценарии, при котором приложение используется нестандартным способом (ввод строки в поле, предназначенное только для чисел, попытка добавить существующую запись с существующим идентификатором и так далее). Данный тип тестирования также характеризует тестирование по глубине его проведения.

Это и есть критическое тестирование, благодаря которому можно проверить правильность работы часто используемых функций. Данный тип тестирования является наиболее быстрым и критически важным для пользователя.

Тестовые случаи для критического тестирования веб-приложения развернутого по адресу <https://ziroweb.azurewebsites.net/> представлены в таблице 5.2.

Таблица 5.2 – Тестовые случаи для критического тестирования

№	Название модуля/экрана	Описание тестового случая	Ожидаемые результаты	Тестовый случай пройден? Да/Нет	Комментарии
1	2	3	4	5	6
1	Запуск приложения	Запуск приложения в различных браузерах: 1) запуск браузера; 2) ввод адреса тестируемого веб-сайта.	1) открытие браузера; 2) загрузка страницы входа в систему.	Да	
2	Аутентификация	Аутентификация 1) переход на тестируемый веб-сайт в браузере; 2) ввод в поле Email значения «test@mail.com»; 3) ввод в поле пароля «123»; 4) нажатие кнопки «Войти».	1) открытие стартовой страницы пользователя.	Да	
3	Авторизация пользователя	Аутентификация: 1) переход на тестируемый веб-сайт в браузере; 2) ввод в поле Email значения «user@mail.com»; 3) ввод в поле пароля «user»; 4) нажатие кнопки «Войти».	1) открытие стартовой страницы пользователя со списком текущих задач; 2) показ верхнего меню, состоящего из задач, проектов и команд, а также отображение изображения пользователя в верхнем правом углу.	Да	
4	Авторизация администратора	Авторизация администратора: 1) переход на тестируемый веб-сайт в браузере; 2) ввод в поле Email значения admin@mail.com; 3) ввод в поле пароля «user» и нажатие кнопки «Войти».	1) открытие стартовой страницы администратора со списком проектов системы 2) показ верхнего меню, состоящего из пользователей и проектов.	Да	

Продолжение таблицы 5.2

1	2	3	4	5	6
5	Редактирование профиля пользователя	Редактирование профиля пользователя: 1) авторизация на веб-сайте как пользователь; 2) нажатие на изображение; пользователя в верхнем правом углу пользователя 3) нажатие кнопки «Редактировать»; 4) ввод данных и загрузка новой фотографии.	1) обновление фотографии в верхнем правом углу; 2) переход на страницу текущих проектов.	Да	
6	Создание задачи	Создание задачи 1) авторизация на веб-сайте как пользователь; 2) нажатие кнопки «Создать задачу» в открывшемся окне; 3) заполнить все поля и нажать кнопку создать.	1) переход на страницу со списком текущих задач; 2) появление созданной задачи в списке.	Да	
7	Создание пользователя администратором	Создание пользователя администратором: 1) авторизация в приложении как администратор; 2) переход на вкладку «Пользователи» в верхнем меню и нажатие кнопки «Создать»; 3) заполнение всех данных в открывшемся окне.	1) переход на страницу с пользователями; 2) появление созданного пользователя в списке.	Да	
8	Создание пользователя администратором	Создание проекта администратором: 1) авторизация в приложении как администратор; 2) переход на вкладку «Проекты» и нажатие кнопки «Создать» 3) заполнение всех данных и нажатие кнопки «Сохранить».	1) переход на страницу с проектами; 2) появление созданного проекта в списке.	Да	

Продолжение таблицы 5.2

1	2	3	4	5	6
9	Комментирование задачи	Комментирование задачи: 1) авторизация в приложении как пользователь; 2) переход на любую из задач в списке задач на появившейся странице; 3) выбор вкладки «Комментарии» в отобразившейся странице информации о задаче; 4) нажатие кнопки «Добавить комментарий»; 5) ввод текста в появившемся поле и нажатие кнопки «Добавить».	1) поле с вводом комментария исчезает; 2) добавленный комментарий появляется в списке комментариев к задаче в самом конце.	Да	
10	Создание записи в журнале работ	Создание записи в журнале работ: 1) авторизация в приложении как пользователь; 2) переход на любую из задач в списке задач на появившейся странице; 3) выбор вкладки «Журнал работ» в отобразившейся странице информации о задаче; 4) нажатие кнопки «Добавить»; 5) ввод текста и потраченного времени в появившихся полях и нажатие кнопки «Добавить».	1) поля с вводом данных исчезают; 2) добавленная запись появляется в журнале работ, а также увеличивается значение в поле «Затраченное время» у задачи на размер введенного значения потраченного времени.	Да	

5.2.2 Углубленное тестирование

Углубленное (расширенное) тестирование – это процесс поиска ошибок в программе в нестандартных, непредвиденных ситуациях (например, при некорректно вводимых данных).

Примеры тестовых случаев для углубленного тестирования представлены ниже в таблице 5.3.

Таблица 5.3 - Тестовые случаи для углубленного тестирования

№	Название модуля/экрана	Описание тестового случая	Ожидаемые результаты	Тестовый случай пройден? Да/Нет	Комментарии
1	2	3	4	5	6
1	Аутентификация пользователя	Аутентификация пользователя с незаполненными полем «Пароль» или «Email»: 1) открытие тестируемого веб-сайта в браузере; 2) ввод одного из полей; 3) нажатие кнопки «Войти».	1) показ сообщения об ошибке о необходимости ввода пароля или Email	Да	
2	Создание пользователя другим пользователем	Попытка создание пользователя другим пользователем: 1) вход в систему под учетными данными пользователя; 2) ввод значения «\users\create» в адресной строке браузера в дополнение к имени тестируемого сайта; 3) нажатие клавиши «Ввод».	1) переход на страницу с показом ошибки о запрещении доступа и кодом ошибки 403	Да	
3	Создание проекта пользователем	Попытка создания проекта пользователем: 1) вход в систему как пользователь; 2) ввод имени сайта и «\projects\create» в дополнение к нему; 3) нажатие клавиши «Ввод».	1) переход на страницу с показом ошибки о запрещении доступа и кодом ошибки 403	Да	

Продолжение таблицы 5.3

1	2	3	4	5	6
4	Оставление пустого комментария к задаче	Комментирование задачи: 1) авторизация в приложении как пользователь; 2) переход на любую из задач в списке задач на появившейся странице; 3) выбор вкладки «Комментарии» в отобразившейся странице информации о задаче; 4) нажатие кнопки «Добавить комментарий»; 6) оставление пустым поле с текстом и нажатие кнопки «Добавить».	1) отображение сообщение об ошибке «Комментарий не может быть пустым».	Да	
5	Создание пользователя	Создание пользователя без указания Email: 1) авторизация в приложении как администратор; 2) переход на вкладку «Пользователи» в верхнем меню; 3) нажатие кнопки «Создать»; 4) заполнение всех данных в открывшемся окне кроме поля «Email».	1) отображение сообщение об ошибке «Поле Email не может быть пустым».	Да	

6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОЙ СИСТЕМЫ

6.1 Расчет сметы затрат, цены и прибыли на программную систему

В современных рыночных экономических условиях программное средство выступает преимущественно в виде продукции научно-технических организаций, представляющей собой функционально завершенные и имеющие товарный вид программные средства, реализуемые покупателям по рыночным отпускным ценам. Все завершенные разработки программных средств являются научно-технической продукцией.

Выбор эффективных проектов связан с их экономической оценкой и расчетом экономического эффекта.

У разработчика экономический эффект выступает в виде чистой прибыли, а у пользователя – в виде экономии различных ресурсов, получаемой за счет:

- снижения трудоемкости расчетов и алгоритмизации программирования и отладки программ за счет использования программного средства в процессе разработки автоматизированных систем обработки данных;
- сокращения расходов на оплату машинного времени и других ресурсов на отладку программ;
- снижения расходов на материалы (магнитные ленты, магнитные диски и прочие материалы);
- улучшения показателей основной деятельности предприятий в результате использования программного средства.

Стоимостная оценка программного средства у разработчиков предполагает составление сметы затрат, которая включает следующие статьи:

- заработная плата исполнителей основная (Z_0) и дополнительная (Z_d);
- отчисления в фонд социальной защиты населения ($Z_{сз}$);
- отчисления на развитие здравоохранения и охрану здоровья ($Z_{оз}$);
- налоги, от фонда оплаты труда (H_e);
- материалы и комплектующие (M);
- спецоборудование (P_c);
- машинное время (P_m);
- расходы на научные командировки ($P_{нк}$);
- прочие прямые затраты (P_z);
- накладные расходы (P_n).

На основании сметы затрат рассчитывается себестоимость и отпускная цена программного средства.

6.1.1 Исходные данные

Исходные данные для расчета заносятся в таблицу (таблица 6.1).

Таблица 6.1 – Исходные данные для расчета

№ пп	Наименование показателя	Единица измерения	Условные обозначения	Норматив
1	2	3	4	5
1.	Коэффициент изменения скорости обработки информации	ед.	$K_{ск}$	0,5
2.	Численность разработчиков	чел.	$Ч_p$	2
3.	Тарифная ставка 1-го разряда в организации	руб.	$T_{м1}$	36,4
4.	Тарифный коэффициент	ед.	T_k	2,48
5.	Норматив дополнительной заработной платы рассчитывается по формуле	%	H_d	16
6.	Норматив отчислений в фонд социальной защиты населения	ед.	$H_{сз}$	35
7.	Норматив налога, уплачиваемого единым платежом	%	$H_{не}$	4
8.	Норматив расходов на командировки в целом по научной организации	%	$H_{рнк}$	30
9.	Норматив прочих затрат в целом по научной организации	%	$H_{пз}$	20
10.	Норматив накладных расходов в целом по научной организации	%	$H_{рн}$	100
11.	Уровень рентабельности программного средства	%	$У_{рн}$	53
12.	Норматив расходов на сопровождение и адаптацию	%	$H_{рса}$	4
13.	Норматив отчислений в местный и республиканский бюджеты	%	$H_{мр}$	3,9
14.	Норматив НДС	%	$H_{дс}$	20
15.	Налог на прибыль при отсутствии льгот	%	H_n	24

Продолжение таблицы 6.1

1	2	3	4	5
16.	Норматив расхода машинного времени на отладку 100 строк исходного кода	час	N_{MB}	12
17.	Количество дней в году	день	D_r	365
18.	Количество праздничных дней в году	день	D_p	9
19.	Количество выходных дней в году	день	D_v	104
20.	Количество дней отпуска	день	D_o	24

6.1.2 Общая характеристика разрабатываемой программной системы

Данное программное средство является программным средством общего назначения, основные функции:

- организация ввода информации;
- контроль, предварительная обработка и ввод информации;
- управление вводом/выводом;
- генерация структуры базы данных;
- манипулирование данными;
- организация поиска и поиск в базе данных;
- обслуживание файлов;
- обработка ошибочных и сбойных ситуаций;
- справка и обучение.

Разрабатываемое программное средство входит в третью группу сложности.

6.1.3 Определение объема программной системы

Объем программного средства определяется путем подбора аналогов на основании классификации типов программных средств, каталога функции программного средства и аналогов программного средства в разрезе функций, которые постоянно обновляются и утверждаются в установленном порядке. Общий объем программного средства рассчитывается по формуле:

$$V_o = \sum_{i=1}^n V_i, \quad (6.1)$$

где V_o – общий объем программного средства, строка исходного кода;

V_i – объем функций программного средства, строка исходного кода;

n – общее число функций.

Содержание и объем функций разрабатываемого программного средства приведено в таблице 6.2.

Таблица 6.2 – Содержание и объем функций разрабатываемого программного средства

№ функции	Содержание функции	Объем функций (строки исходного кода)
101	Организация ввода информации	150
102	Контроль, предварительная обработка и ввод информации	450
111	Управление вводом/выводом	2 400
201	Генерация структуры базы данных	4 300
207	Манипулирование данными	9 550
208	Организация поиска и поиск в базе данных	5 480
304	Обслуживание файлов	420
506	Обработка ошибочных и сбойных ситуаций	410
604	Справка и обучение	720
	ИТОГО	23 880

$$V_o = 150 + 450 + 2\,400 + 4\,300 + 9\,550 + 5\,480 + 420 + 410 + 720 \\ = 23\,880 \text{ строк исходного кода.}$$

6.1.4 Расчет трудоемкости выполняемой работы

На основании общего объема программного средства определяется нормативная трудоемкость (T_H). Нормативная трудоемкость устанавливается с учетом сложности программного средства. Выделяется три группы сложности, в которых учтены следующие составляющие программного средства; языковой интерфейса, ввод-вывод, организация данных, режим работы, операционная, а также техническая среда. Кроме того, устанавливаются дополнительные коэффициенты сложности программной системы.

С учетом дополнительного коэффициента сложности $K_{СЛ}$ умноженного на нормативную трудоемкость можно рассчитать общую трудоемкость программного

средства по формуле:

$$T_o = T_n \cdot K_{cl}, \quad (6.2)$$

где T_o – общая трудоемкость ПС, человеко-день;

T_n – нормативная трудоемкость ПС, человеко-день;

K_{cl} – дополнительный коэффициент сложности ПС.

Поскольку разрабатываемое ПС относится к группе сложности 1, то нормативная трудоемкость T_n равна 442. ПС обладает характеристикой функционирования в расширенной операционной среде, поэтому коэффициент сложности K_{cl} равен 0.08.

$$T_o = 442 \cdot 0,08 = 35,36 \text{ человеко} - \text{дней.}$$

При решении сложных задач с длительным периодом разработки ПС трудоемкость определяется по стадиям разработки (техническое задание – ТЗ, эскизный проект – ЭП, технический проект – ТП, рабочий проект – РП и внедрение – ВН) с учетом новизны, степени использования типовых программ и удельного веса трудоемкости стадий разработки ПС и общей трудоемкости разработки ПС. При этом на основании общей трудоемкости рассчитывается уточненная трудоемкость с учетом распределения по стадиям (T_y).

$$T_y = \sum_{i=1}^m T_i, \quad (6.3)$$

где T_i – трудоемкость разработки ПС на i -й стадии, человеко-день;

m – количество стадий разработки.

Трудоемкость ПС по стадиям определяется с учетом новизны и степени использования в разработке типовых программ и ПС.

$$T_{cti} = d_{cti} \cdot K_n \cdot K_T \cdot T_o, \quad (6.4)$$

где T_{cti} – трудоемкость разработки ПС на i -й стадии (технического задания, эскизного проекта, технического проекта, рабочего проекта и внедрения), человеко-день;

K_n – поправочный коэффициент, учитывающий степень новизны ПС;

K_T – поправочный коэффициент, учитывающий степень использования в разработке типовых программ и ПС;

d_{cti} – удельный вес трудоемкости i -й стадии разработки ПС в общей

трудоемкости разработки ПС.

На основании данных в приложениях разрабатываемое ПС относится к 3-ей группе сложности и имеет степень новизны В, при этом $K_{сл}=0,08$, а $K_{н}=0,7$. Коэффициенты удельных весов трудоемкостей для каждой стадии: $TЗ=0,09$; $ЭП=0,07$; $ТП=0,07$; $РП=0,61$; $ВН=0,16$.

Исходя из этих данных, можно рассчитать трудоемкость ПС на каждой стадии:

$$T_{стТЗ} = 0,09 * 0,7 * 0,7 * 35,36 = 1,56 \text{ человеко – дней.}$$

$$T_{стЭП} = 0,07 * 0,7 * 0,7 * 35,36 = 1,21 \text{ человеко – дней.}$$

$$T_{стТП} = 0,07 * 0,7 * 0,7 * 35,36 = 1,21 \text{ человеко – дней.}$$

$$T_{стрП} = 0,61 * 0,7 * 0,7 * 35,36 = 10,57 \text{ человеко – дней.}$$

$$T_{стВН} = 0,16 * 0,7 * 0,7 * 35,36 = 2,77 \text{ человеко – дней.}$$

Тогда уточненная трудоемкость с учетом распределения по стадиям равна:

$$T_y = 1,56 + 1,21 + 1,21 + 10,57 + 2,77 = 17,33 \text{ человеко – дня.}$$

6.1.5 Расчет основной заработной платы исполнителей

Эффективный фонд времени работы одного работника ($\Phi_{эф}$) рассчитывается по формуле:

$$\Phi_{эф} = D_{г} - D_{п} - D_{в} - D_{о}, \quad (6.5)$$

где $D_{г}$ – количество дней в году, день;

$D_{п}$ – количество праздничных дней в году, день;

$D_{в}$ – количество выходных дней в году, день;

$D_{о}$ – количество дней отпуска, день.

При утверждении плановой численности разработчиков продолжительность разработки определяется по формуле:

$$T_p = \sum_{i=1}^m \frac{T_i}{\chi_{pi} \cdot \Phi_{эф}}, \quad (6.6)$$

где T_p – срок разработки ПС (лет);

T_i – трудоемкость разработки ПС на i -й стадии (человеко-дней);

χ_{pi} – численность разработчиков ПС на i -й стадии (чел.);

m – число стадий.

Уточненная трудоемкость и общая плановая численность разработчиков служат базой для расчета основной заработной платы. По данным о специфике и сложности выполняемых функций составляется штатное расписание группы специалистов-исполнителей, участвующих в разработке ПС, с определением образования, специальности, квалификации и должности.

Таким образом, можно рассчитать эффективный фонд времени:

$$\Phi_{эв} = 365 - 9 - 104 - 24 = 228 \text{ дней.}$$

А также срок разработки ПС при количестве разработчиков на всех стадиях разработки ПС равным 2.

$$T_p = \frac{17,3264}{2 * 228} = 0,038 \text{ лет.}$$

В соответствии с «Рекомендациями по применению «Единой тарифной сетки» рабочих и служащих народного хозяйства» и тарифными разрядами и коэффициентами должностей руководителей научных организаций и вычислительных центров, бюджетных учреждений науки непромышленных отраслей народного хозяйства каждому исполнителю устанавливается разряд и тарифный коэффициент.

Месячная тарифная ставка каждого исполнителя (T_M) определяется путем умножения действующей месячной тарифной ставки 1-го разряда (T_{M1}) на тарифный коэффициент (T_K), соответствующий установленному тарифному разряду:

$$T_M = T_{M1} * T_K. \quad (6.7)$$

Часовая тарифная ставка рассчитывается путем деления месячной тарифной ставки на установленный при семичасовом рабочем дне фонд рабочего времени (Φ_p):

$$T_{\text{ч}} = \frac{T_M}{\Phi_p}, \quad (6.8)$$

где $T_{\text{ч}}$ – часовая тарифная ставка, руб.;

T_M – месячная тарифная ставка, руб.

Оба работника имеют одинаковую квалификацию и принадлежат 10-му разряду. Тарифная месячная и часовая ставки для них будут равны:

$$T_M = 36,4 * 2,48 = 90,27 \text{ руб.}$$

$$T_{\text{ч}} = \frac{90,72}{133} = 0,68 \text{ руб.}$$

Основная заработная плата исполнителей на конкретное ПС рассчитывается по формуле:

$$Z_{oi} = \sum_{i=1}^n T_{\text{чи}} \cdot T_{\text{ч}} \cdot \Phi_{\text{эi}} \cdot K, \quad (6.9)$$

где n – количество исполнителей, занятых разработкой конкретного ПС;

$T_{\text{чи}}$ – часовая тарифная ставка i -го исполнителя (руб.);

$\Phi_{\text{эi}}$ – эффективный фонд рабочего времени i -го исполнителя (дней);

$T_{\text{ч}}$ – количество часов работы в день (ч);

K – коэффициент премирования.

$$Z_o = 2 * 0,68 * 133 * 1 = 180,54 \text{ руб.}$$

6.1.6 Расчет дополнительной заработной платы исполнителей

Дополнительная заработная плата на конкретное ПС ($Z_{\text{дi}}$) включает выплаты, предусмотренные законодательством о труде (оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей), и определяется по нормативу в процентах к основной заработной плате:

$$Z_{\text{дi}} = \frac{Z_{oi} \cdot H_{\text{д}}}{100}, \quad (6.10)$$

где $Z_{\text{дi}}$ – дополнительная заработная плата исполнителей на конкретное ПС, руб.;

$H_{\text{д}}$ – норматив дополнительной заработной платы, %.

$$Z_{\text{д}} = \frac{180,54 * 16}{100} = 28,89 \text{ руб.}$$

6.1.7 Расчет отчислений в фонд социальной защиты населения

Отчисления в фонд социальной защиты населения ($Z_{\text{Сзи}}$) определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей,

определенной по нормативу, установленному в целом по организации:

$$З_{сзi} = \frac{(З_{oi} + З_{di}) \cdot Н_{сз}}{100}, \quad (6.11)$$

где $Н_{сз}$ – норматив отчислений в фонд социальной защиты населения, %.

$$З_{сзi} = \frac{(180,544 + 28,88704) \cdot 35}{100} = 73,3 \text{ руб.}$$

6.1.8 Расчет налога на ликвидацию последствий чернобыльской катастрофы

Налоги, рассчитываемые от фонда оплаты труда, определяются в соответствии с действующими законодательными актами по нормативам в процентном отношении к сумме всей заработной платы, относимой на ПС (налог, уплачиваемый единым платежом, включая налог на ликвидацию последствий чернобыльской катастрофы и отчисления в фонд занятости ($Н_{ei}$)):

$$Н_{ei} = \frac{(З_{oi} + З_{di}) \cdot Н_{не}}{100}, \quad (6.12)$$

где $Н_{не}$ – норматив налога, уплачиваемого единым платежом (%).

$$Н_{ei} = \frac{(180,544 + 28,88704) \cdot 4}{100} = 8,38 \text{ руб.}$$

6.1.9 Расчет расходов по статье «Материалы»

Расходы по статье «Материалы» (М) определяются на основании сметы затрат, разрабатываемой на ПС с учетом действующих нормативов. По статье «Материалы» отражаются расходы на бумагу и прочие расходные материалы, необходимые для разработки ПС. Сумма затрат материалов рассчитывается по формуле:

$$М_i = Н_{Mi} \cdot \frac{V_{oi}}{100}, \quad (6.13)$$

где $Н_{Mi}$ – норма расхода материалов в расчете на 100 строк исходного кода ПС,

руб.;

V_{oi} – общий объем ПС на конкретное ПС, строка исходного кода.

$$M_i = 0,038 * \frac{23\,880}{100} = 9,07 \text{ руб.}$$

6.1.10 Расчет расходов по статье «Спецоборудование»

Расходы по статье «Спецоборудование» (P_{ci}) включает затраты средств на приобретение вспомогательных специального назначения технических и программных средств, необходимых для разработки конкретного ПС, включая расходы на их проектирование, изготовление, отладку, установку и эксплуатацию. Затраты по этой статье определяются в соответствии со сметой расходов, которая составляется перед разработкой ПС. Данная статья включается в смету расходов на разработку ПС в том случае, когда приобретаются специальное оборудование или специальные программы, предназначенные для разработки и создания только данного ПС:

$$P_{ci} = \sum_{i=1}^n \Pi_{ci}, \quad (6.14)$$

где Π_{ci} – стоимость конкретного специального оборудования, руб.;

n – количество применяемого специального оборудования.

Необходимо учесть следующие расходы по данной статье:

- хостинг-план для размещения веб-сайта в интернете) – 28,9 рублей;
- VS Resharper – 55,23 рублей.

$$P_{ci} = 28,9 + 55,23 = 84,13 \text{ руб.}$$

6.1.11 Расчет расходов по статье «Машинное время»

Расходы по статье «Машинное время» (P_{mi}) включают оплату машинного времени, необходимого для разработки и отладки ПС, которое определяется машино-часами в зависимости от характера решаемых задач и типа ПЭВМ:

$$P_{mi} = \Pi_{mi} \cdot \frac{V_{oi}}{100} \cdot H_{MB}, \quad (6.15)$$

где Π_{mi} – цена одного машино-часа, руб.;

V_{oi} – общий объем ПС, строка исходного кода;

H_{MB} – норматив расхода машинного времени на отладку 100 строк исходного кода, машино-час.

Цена одного машино-часа (Π_{Mi}) для обеспечения работы веб-сервера составляет 2 рубля.

$$P_{Mi} = 2 * \left(\frac{23\,380}{100} \right) * 12 = 5\,731,2 \text{ руб.}$$

6.1.12 Расчет расходов по статье «Научные командировки»

Расходы по статье «Научные командировки» (P_{Hki}) на конкретное ПС определяются по нормативу, разрабатываемому в целом по научной организации, в процентах к основной заработной плате:

$$P_{Hki} = \frac{Z_{oi} \cdot H_{PHK}}{100}, \quad (6.16)$$

где H_{PHK} – норматив расходов на командировки в целом по научной организации, %.

$$P_{Hki} = \frac{180,54 * 30}{100} = 54,16 \text{ руб.}$$

6.1.13 Расчет расходов по статье «Прочие затраты»

Расходы по статье «Прочие затраты» (Π_{zi}) на конкретное ПС включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы. Определяются по нормативу, разрабатываемому в целом по научной организации, в процентах к основной заработной плате:

$$\Pi_{zi} = \frac{Z_{oi} \cdot H_{\Pi z}}{100}, \quad (6.17)$$

где $H_{\Pi z}$ – норматив прочих затрат в целом по научной организации.

$$\Pi_{zi} = \frac{180,544 * 20}{100} = 36,11 \text{ руб.}$$

6.1.14 Расчет расходов по статье «Накладные расходы»

Затраты по статье «Накладные расходы» (P_{Hi}), связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды (P_{Hi}), относятся на конкретное ПС по нормативу (H_{PH}) в процентном отношении к основной заработной плате исполнителей. Норматив устанавливается в целом по научной организации:

$$P_{Hi} = \frac{Z_{oi} \cdot H_{PH}}{100}, \quad (6.18)$$

где P_{Hi} – накладные расходы на конкретное ПС, руб.;

H_{PH} – норматив накладных расходов в целом по научной организации.

$$P_{Hi} = \frac{180,544 \cdot 100}{100} = 180,54 \text{ руб.}$$

6.1.15 Расчет общей суммы расходов на разработку

Общая сумма расходов по всем статьям сметы (C_{Pi}) на ПС рассчитывается по формуле:

$$C_{Pi} = Z_{oi} + Z_{di} + Z_{czi} + H_{ei} + M_i + P_{ci} + P_{mi} + P_{nki} + \Pi_{zi} + P_{Hi}. \quad (6.19)$$

Кроме того, организация-разработчик осуществляет затраты на сопровождение и адаптацию ПС (P_{CAi}), которые определяются по нормативу (H_{PCA})

$$C_{Pi} = 180,54 + 28,89 + 73,3 + 8,377 + 9,07 + 84,13 + 5731,2 + 54,16 + 36,11 + 180,544 = 6386,33 \text{ руб.}$$

$$P_{cai} = \frac{C_{Pi} \cdot H_{pca}}{100}, \quad (6.20)$$

где H_{pca} – норматив расходов на сопровождение и адаптацию, %.

$$P_{cai} = \frac{6\,386,33 \cdot 4}{100} = 255,45 \text{ руб.}$$

6.1.16 Расчет прибыли и отпускной цены

Рентабельность и прибыль по создаваемому ПС определяются исходя из результатов анализа рыночных условий, согласования с заказчиком отпускной цены. Прибыль рассчитывается по формуле:

$$П_{\text{псi}} = \frac{C_{\text{пi}} \cdot Y_{\text{рпi}}}{100}, \quad (6.21)$$

где $П_{\text{псi}}$ – прибыль от реализации ПС, руб.;

$Y_{\text{рпi}}$ – уровень рентабельности ПС, %;

$C_{\text{пi}}$ – себестоимость ПС, руб.

$$П_{\text{псi}} = \frac{6\,641,78 \cdot 53}{100} = 3\,520,14 \text{ руб.}$$

Прогнозируемая цена ПС без налогов ($Ц_{\text{пi}}$):

$$Ц_{\text{пi}} = C_{\text{пi}} + П_{\text{сi}}. \quad (6.22)$$

$$Ц_{\text{пi}} = 6\,641,78 + 3\,520,14 = 10\,161,92 \text{ руб.}$$

Отчисления и налоги в местный и республиканский бюджеты единым платежом ($O_{\text{мрi}}$):

$$O_{\text{мрi}} = \frac{Ц_{\text{пi}} \cdot H_{\text{мр}}}{100\%}, \quad (6.23)$$

$$O_{\text{мрi}} = \frac{(10\,161,92 \cdot 3,9)}{100} = 396,31 \text{ руб.}$$

где $H_{\text{мр}}$ – норматив отчислений в местный и республиканский бюджеты (%).

Налог на добавленную стоимость (НДС_i):

$$\text{НДС}_i = \frac{(Ц_{\text{пi}} + O_{\text{мр}}) \cdot H_{\text{дс}}}{100\%}, \quad (6.24)$$

где $H_{\text{дс}}$ – норматив НДС, %;

$O_{\text{мр}}$ – отчисления и налоги в местный и республиканский бюджеты, руб.;

Π_{pi} – прогнозируемая цена, руб.

$$\text{НДС}_i = \frac{(10\,161,92 + 396,31) \cdot 20}{100} = 2\,111,65 \text{ руб.}$$

Прогнозируемая отпускная цена (Π_{oi}):

$$\Pi_{\text{oi}} = \Pi_{\text{pi}} + O_{\text{mpi}} + \text{НДС}_i. \quad (6.25)$$

$$\Pi_{\text{oi}} = 10\,161,92 + 396,31 + 2\,111,65 = 12\,669,89 \text{ руб.}$$

6.2 Расчет экономического эффекта от применения ПС у пользователя

6.2.1 Исходные данные

Исходные данные для расчета экономического эффекта от применения ПС у пользователя сведены в таблицу 6.3.

Таблица 6.3 – Исходные данные для расчета экономического эффекта

Наименование показателей	Обозначения	Единицы измерения	Значение показателя	
			в базовом варианте	в новом варианте
1. Средняя трудоемкость работ в расчете на 100 КБ	T_{c1} T_{c2}	человеко-час на 100 КБ	1,8	1,7
2. Средний расход машинного времени в расчете на 100 КБ	$M_{в1}$ $M_{в2}$	машино-час на 100 КБ	9,0	8,91
3. Средний расход материалов в расчете на 100 КБ	$M_{т1}$ $M_{т2}$	руб. на 100 КБ	0,032	0,028
4. Количество типовых задач, решаемых за год	$Z_{т2}$	задача	9	
5. Ставка налога на прибыль	$H_{\text{п}}$	%	18	

6.2.2 Расчет объема работ

Объем работ в зависимости от функциональной группы и назначения ПС можно

определить по формуле:

$$A = V_{\text{пс}} \cdot K_{\text{пс}}, \quad (6.26)$$

где $V_{\text{пс}}$ – объем ПС в натуральных единицах измерения;

$K_{\text{пс}}$ – коэффициент применения ПС.

$$A = 23\,880 \cdot 0,5 = 11\,940.$$

6.2.3 Расчет капитальных затрат

Общие капитальные вложения (K_o) заказчика (потребителя), связанные с приобретением, внедрением и использованием ПС, рассчитываются по формуле:

$$K_o = K_{\text{пр}} + K_{\text{ос}} + K_{\text{тс}} + K_{\text{об}}, \quad (6.27)$$

где $K_{\text{пр}}$ – затраты пользователя на приобретение ПС по отпускной цене разработчика с учетом стоимости услуг по эксплуатации и сопровождению, руб.;

$K_{\text{тс}}$ – затраты на доукомплектацию ВТ техническими средствами в связи с внедрением нового ПС, руб.;

$K_{\text{об}}$ – затраты на пополнение оборотных средств в связи с использованием нового ПС, руб.

$$K_o = 12\,669,89 + 126,7 + 0 + 126,7 = 12923,29 \text{ руб.}$$

Затраты на освоение ПС и на пополнение оборотных средств рекомендуется рассчитывать по формулам:

$$K_{\text{ос}} = K_{\text{пр}} \cdot H_{\text{кос}}, \quad (6.28)$$

где $H_{\text{кос}}$ – норматив затрат пользователя на освоение ПС, равный 0,01.

$$K_{\text{ос}} = 12\,669,89 \cdot 0,01 = 126,7 \text{ руб.}$$

$$K_{\text{об}} = K_{\text{пр}} \cdot H_{\text{коб}}, \quad (6.29)$$

где $H_{\text{коб}}$ – норматив затрат на пополнение оборотных средств в связи с использованием нового ПС, равный 0,01.

6.2.4 Расчет экономии основных видов ресурсов в связи с использованием нового программного средства

Экономия затрат на заработную плату при использовании нового ПС в расчете на объем выполненных работ:

$$C_3 = C_{3e} * A_2, \quad (6.30)$$

где C_{3e} – экономия затрат на заработную плату при решении задач с использованием нового ПС в расчете на 100 КБ, руб.;

A_2 – объем выполненных работ с использованием нового ПС (100 КБ).

$$C_3 = 0,06787 * 107\,460 = 7\,293,70 \text{ руб.}$$

Экономия затрат на заработную плату в расчете на 100 КБ (C_{3e}):

$$C_{3e} = \frac{Z_{cm} \cdot (T_{c1} - T_{c2}) : T_{\text{ч}}}{D_p}, \quad (6.31)$$

где Z_{cm} – среднемесячная заработная плата одного программиста, руб.;

T_{c1} , T_{c2} – снижение трудоемкости работ в расчете на 100 строк кода (человеко-часов);

$T_{\text{ч}}$ – количество часов работы в день (ч);

D_p – среднемесячное количество рабочих дней.

$$C_{3e} = \frac{90,3 * (1,8 - 1,7)}{133} = 0,06787 \text{ руб.}$$

Объем выполненных работ с использованием нового ПС (100 КБ):

$$A_2 = A_o \cdot Z_{t2}, \quad (6.32)$$

где A_o – объем работ необходимый для решения одной задачи (100 КБ);

Z_{t2} – количество типовых задач, решаемых за год (задач).

$$A_2 = 11\,940 * 9 = 107\,460.$$

Экономия затрат на оплату машинного времени (C_m) в расчете на выполненный объем работ в результате применения новой программной системы можно рассчитать

по следующей формуле (с использованием необходимого объема работ):

$$C_M = C_{Me} \cdot A_2, \quad (6.33)$$

где C_{Me} – экономия затрат на оплату машинного времени при решении задач с использованием нового ПС в расчете на 100 КБ.

$$C_M = 0,18 \cdot 107\,460 = 19\,342,8 \text{ руб.}$$

Экономия затрат на материалы (C_{MT}) при использовании нового ПС в расчете на объем выполненных работ:

$$C_{MT} = C_{MTe} \cdot A_2, \quad (6.34)$$

где C_{MTe} – экономия затрат на материалы в расчете на 100 КБ при использовании нового ПС.

$$C_{MT} = 0,004 \cdot 107\,460 = 429,84 \text{ руб.}$$

$$C_{MTe} = C_{M1} - C_{M2}, \quad (6.35)$$

где C_{M1} , C_{M2} – средний расход материалов у пользователя в расчете на 100 КБ при использовании соответственно базового и нового ПС, руб.

$$C_{MTe} = 0,032 - 0,028 = 0,004 \text{ руб.}$$

Общая годовая экономия текущих затрат, связанных с использованием нового ПС (C_o):

$$C_o = C_z + C_{oz} + C_M + C_{MT}. \quad (6.36)$$

$$C_o = 7\,293,7 + 19\,342,8 + 429,84 = 27\,066,35 \text{ руб.}$$

6.2.5 Расчет экономического эффекта

Внедрение нового ПС позволит пользователю сэкономить на текущих затратах, т.е. практически получить на эту сумму дополнительную прибыль. Для пользователя в качестве экономического эффекта выступает лишь чистая прибыль –

дополнительная прибыль, остающаяся в его распоряжении ($\Delta\Pi_{\text{ч}}$), которые определяются по формуле:

$$\Delta\Pi_{\text{ч}} = C_o - \frac{C_o \cdot H_{\text{п}}}{100}, \quad (6.37)$$

где $H_{\text{п}}$ – ставка налога на прибыль (%);

C_o – общая годовая экономия затрат, руб.

$$\Delta\Pi_{\text{ч}} = 27\,066,35 - \frac{27\,066,35 \cdot 18}{100} = 22\,194,4 \text{ руб.}$$

В процессе использования нового ПС чистая прибыль в конечном итоге возмещает капитальные затраты. Однако, полученные при этом суммы результатов (прибыли) и затрат (капитальных вложений) по годам приводят к единому времени – расчетному году (за расчетный год принят 2019 год) путем умножения результатов и затрат за каждый год на коэффициент приведения ($ALFA_t$), который рассчитывается по формуле:

$$ALFA_t = (1 + E_{\text{н}})^{t_p - t}, \quad (6.38)$$

где $E_{\text{н}}$ – норматив приведения разновременных затрат и результатов;

t_p – расчетный год, $t_p=1$;

t – номер года, результаты и затраты которого приводятся к расчетному (2019-1, 2020-2, 2021-3, 2022-4).

Норматив приведения разновременных затрат и результатов ($E_{\text{н}}$) для программных систем в существующей практике принимается в пределах 0,2–0,4. Например, при нормативе 0,4 коэффициентам приведения ($ALFA_t$) по годам будут соответствовать следующие значения:

$$ALFA_1 = (1 + 0,4)^{1-1} = 1 - \text{расчетный год};$$

$$ALFA_2 = (1 + 0,4)^{1-2} = 0,714 - 2020 \text{ год};$$

$$ALFA_3 = (1 + 0,4)^{1-3} = 0,510 - 2021 \text{ год};$$

$$ALFA_4 = (1 + 0,4)^{1-4} = 0,364 - 2022 \text{ год}.$$

Результаты расчета экономического эффекта сведены с учетом коэффициентов приведения по годам (2019, 2020, 2021 и 2022), а также смета затрат на разработку и

использование программной системы у пользователя сведены в таблицу 6.4.

Таблица 6.4 – Данные экономического эффекта от использования нового ПС

Показатели	Ед. измерения	2019	2020	2021	2022
Результаты:					
Прирост прибыли за счет экономии затрат (П _ч)	руб.	22 194,4	22 194,4	22 194,4	22 194,4
То же с учетом фактора времени	руб.	22 194,4	15 846,8	11 319,15	8 078,76
Затраты:					
Приобретение, адаптация и освоение ПС (К _{пр})	руб.	12 669,89	-	-	-
Освоение ПС (К _{ос})	руб.	126,7	-	-	-
Доукомплектование ВТ техническими средствами (К _{тс})	руб.	-	-	-	-
Пополнение оборотных средств (К _{об})	руб.	126,7	126,7	126,7	126,7
Всего затрат	руб.	12 923,29	126,7	126,7	126,7
То же с учетом фактора времени	руб.	12 923,29	90,46	64,62	46,12
Экономический эффект:					
Превышение результата над затратами	руб.	9 271,11	15 756,34	11 254,53	8 032,64
То же с нарастающим итогом	руб.	9 271,11	25 027,45	36 281,98	44 314,63
Коэффициент приведения	единиц	1	0,714	0,510	0,364

Таким образом, отпускная цена предлагаемой разработки составляет 12 700 рублей, окупаемость ее будет достигнута уже на первом году применения. При этом эффект от использования в течение четырех лет составит 44 315 рублей.

7 ОХРАНА ТРУДА

7.1 Производственная санитария, техника безопасности и пожарная профилактика

Работающие с ПЭВМ могут подвергаться воздействию различных опасных и вредных производственных факторов, основными из которых являются повышенные уровни: электромагнитного, рентгеновского, ультрафиолетового и инфракрасного излучения; статического электричества; нерациональная организация рабочего места; запыленности воздуха рабочей зоны; повышенное или пониженное содержание аэроионов в воздухе рабочей зоны; повышенный или пониженный уровень освещенности рабочей зоны, содержание в воздухе рабочей зоны оксида углерода, озона, аммиака, фенола, формальдегида и полихлорированных фенилов; напряжение зрения, памяти, внимания; длительное статическое напряжение; статического электричества; запыленности воздуха рабочей зоны; большой объем информации, обрабатываемой в единицу времени; монотонность труда; нерациональная организация рабочего места; эмоциональные перегрузки.

Работа с ПЭВМ проводится в соответствии с Санитарными нормами и правилами «Требования при работе с видеодисплейными терминалами и электронно-вычислительными машинами» и Гигиеническим нормативом «Предельно-допустимые уровни нормируемых параметров при работе с видеодисплейными терминалами и электронно-вычислительными машинами», утвержденными постановлением Министерства здравоохранения от 28.06.2013 г. № 59 и Типовой инструкцией по охране труда при работе с персональными ЭВМ, утвержденной постановлением Министерства труда и социальной защиты от 24.12.2013 № 130.

Согласно вышеуказанных документов площадь помещения на одного пользователя ПЭВМ на базе плоских дискретных экранов (жидкокристаллические, плазменные) составляет не менее 4,5 м².

7.1.1 Метеоусловия

Для создания нормальных метеорологических условий наиболее целесообразно уменьшить тепловыделения от самого источника — монитора, что предусматривается при разработке его конструкции

В производственных помещениях, в которых работа с использованием ПЭВМ является основной (операторские, расчетные, посты управления, залы вычислительной техники) для обеспечения необходимых показателей микроклимата предусмотрены системы отопления, вентиляции и кондиционирования воздуха, а также обеспечиваются оптимальные параметры микроклимата для категории работ 1а

и 1б (табл. 7.1) согласно вышеуказанных нормативных документов.

Таблица 7.1 – Оптимальные параметры микроклимата для помещений с ВДТ, ЭВМ и ПЭВМ

Период года	Категория работ	Температура воздуха, °С, не более	Относительная влажность воздуха, %	Скорость движения воздуха, м/с
Холодный	легкая-1а	22-24	40-60	0,1
	легкая-1б	21-23	40-60	0,1
Теплый	легкая-1а	23-25	40-60	0,1
	легкая-1б	22-24	40-60	0,2

Работа с компьютером относится к категории 1а (работы, производимые сидя и сопровождающиеся незначительным физическим напряжением, при которых расход энергии составляет до 120 ккал/ч, т.е. до 139 Вт).

Интенсивность теплового излучения работающих от нагретых поверхностей технологического оборудования, осветительных приборов, инсоляции на постоянных рабочих местах не превышает значений, указанных в табл. 7.2 согласно Санитарных норм и правил.

Таблица 7.2 – Предельно допустимые уровни интенсивности излучения в инфракрасном и видимом диапазоне излучения на расстоянии 0,5 м со стороны экрана ВДТ, ЭВМ и ПЭВМ

Диапазоны длин волн	400-760 нм	760-1050 нм	свыше 1050 нм
Предельно допустимые уровни	0,1 Вт/м ²	0,05 Вт/м ²	4,0 Вт/м ²

Для создания нормальных метеорологических условий наиболее целесообразно уменьшить тепловыделения от самого источника — монитора, что предусматривается при разработке его конструкции.

В производственных помещениях для обеспечения необходимых показателей микроклимата предусмотрены системы отопления, вентиляции и кондиционирования воздуха.

7.1.2 Вентиляция и отопление

Воздух рабочей зоны помещения соответствует санитарно-гигиеническим требованиям по содержанию вредных веществ и частиц пыли, приведенным в Санитарных нормах и правилах «Требованию к контролю воздуха рабочей зоны», Гигиеническом нормативе «Предельно допустимые концентрации вредных веществ в

воздухе рабочей зоны», утв. пост. Министерства здравоохранения от 10.10.2017 г. № 92.

В помещениях, проводится ежедневная влажная уборка и систематическое проветривание после каждого часа работы.

Уровни положительных и отрицательных аэроионов, а также коэффициент униполярности в воздухе всех помещений, где расположены ПЭВМ, соответствуют значениям, указанным в табл. 7.3.

Таблица 7.3 – Уровни ионизации и коэффициент униполярности воздуха помещений при работе с ВДТ, ЭВМ и ПЭВМ

Уровни	Число ионов в 1 см ³ воздуха		Коэффициент униполярности (У)
	n+	n-	
Минимально допустимые	400	600	$0,4 \leq У < 1,0$
Оптимальные	1500-3000	3000-5000	
Максимально допустимые	50000	50000	

Одним из мероприятий по оздоровлению воздушной среды является устройство вентиляции и отопления. Задачей вентиляции является обеспечение чистоты воздуха и параметров метеорологических условий на рабочих местах. Чистота воздушной среды достигается удалением загрязненного или нагретого воздуха из помещения и подачей в него свежего воздуха. Для поддержания нормального микроклимата необходим достаточный объем вентиляции, для чего в вычислительном центре предусматривается кондиционирование воздуха, осуществляющее поддержание постоянных параметров микроклимата в помещении независимо от наружных условий.

Параметры микроклимата поддерживаются в холодный период года за счет системы водяного отопления с нагревом воды до 100°С, а в теплый - за счет кондиционирования, с параметрами отвечающими требованиям СНБ 4.02.01-03.

7.1.3 Освещение

В помещении для эксплуатации ПЭВМ предусмотрены естественное и искусственное освещение. Естественное освещение на рабочих местах осуществляется через световые проемы, ориентированные преимущественно на север, северо-восток, восток, запад или северо-запад и обеспечивает коэффициент естественной освещенности не ниже 1,5 %. Оконные проемы оборудованы регулируемыми устройствами типа жалюзи, занавесей.

Для внутренней отделки интерьера помещений используются материалы с коэффициентом отражения для потолка – 0,7- 0,8; для стен – 0,5- 0,6; для пола – 0,3.

Искусственное освещение в помещениях осуществляется системой общего равномерного освещения. Освещенность поверхности стола в зоне размещения рабочего документа должна составлять 300-500 люкс. Освещенность поверхности экрана не более 300 люкс. В качестве источников света применяем люминесцентные лампы типа ЛБ. Коэффициент запаса для осветительных установок общего освещения принимается равным 1,4, а коэффициент пульсации – не более 5 %.

7.1.4 Шум

Основными источниками шума в помещениях, оборудованных ЭВМ, являются принтеры, множительная техника и оборудование для кондиционирования воздуха, в самих ЭВМ — вентиляторы систем охлаждения и трансформаторы.

В табл. 7.4 приведены нормированные уровни шума согласно Санитарных норм и правил «Требования при работе с видеодисплейными терминалами и электронно-вычислительными машинами», которые обеспечиваются за счет использования малошумного оборудования, а также различных звукопоглощающих устройств (перегородки и т. п.).

Таблица 7.4 – Предельно-допустимые уровни звука, эквивалентные уровни звука и уровни звукового давления в октавных полосах частот при работе с ВДТ, ЭВМ и ПЭВМ и периферийными устройствами

Категория нормы шума	Уровни звукового давления, дБ, в октавных полосах со среднегеометрическими частотами, Гц									Уровни звука и эквивалент ные уровни звука, дБА
	31.5	63	125	250	500	1000	2000	4000	8000	
I	86	71	61	54	49	45	42	40	38	50
II	93	79	70	63	58	55	52	50	49	60
III	96	83	74	68	63	60	57	55	54	65
IV	103	91	83	77	73	70	68	66	64	75

7.1.5 Электробезопасность

Помещение вычислительного центра по степени опасности поражения электрическим током относится к помещениям без повышенной опасности.

Основные меры защиты от поражения током:

- изоляция и недоступность токоведущих частей;
- защитное заземление ($R_3 = 4 \text{ Ом}$ ГОСТ 12.1.030 - 81).

Первая помощь при поражениях электрическим током состоит из двух этапов:

освобождение пострадавшего от действия тока и оказание ему доврачебной медицинской помощи. После освобождения пострадавшего от действия электрического тока необходимо оценить его состояние. Во всех случаях поражения электрическим током необходимо вызвать врача независимо от состояния пострадавшего.

7.1.6 Излучение

При работе с дисплеем могут возникать следующие опасные факторы: электромагнитные и электростатические поля, ультрафиолетовое и инфракрасное излучение.

Уровни физических факторов на рабочих местах пользователей, создаваемые ПЭВМ и периферийными устройствами, не превышают предельно-допустимые уровни: электромагнитных и электростатических полей (табл. 7.5, 7.6), ультрафиолетового (табл. 7.7), установленных Гигиеническим нормативом «Предельно допустимые уровни нормируемых параметров при работе с видеодисплейными терминалами и электронно-вычислительными машинами».

Таблица 7.5 – Предельно допустимые уровни электромагнитных полей от экранов ВДТ, ЭВМ и ПЭВМ

Наименование параметра	Предельно-допустимые уровни
Напряженность электрического поля в диапазоне частот: 5 Гц-2 кГц 2-400 кГц	не более 25,0 В/м не более 2,5 В/м
Плотность магнитного потока магнитного поля в диапазоне частот: 5 Гц-2 кГц 2-400 кГц	не более 250 нТл не более 25 нТл
Напряженность электростатического поля	не более 15 кВ/м

Таблица 7.6 – Предельно допустимые уровни электромагнитных полей при работе с ВДТ, ЭВМ, ПЭВМ от клавиатуры, системного блока, манипулятора «мышь», беспроводных системам передачи информации и иных периферийных устройств

Диапазоны частот	0,3-300 кГц	0,3-3 МГц	3-30 МГц	30-300 МГц	0,3-300 ГГц
Предельно допустимые уровни	25 В/м	15 В/м	10 В/м	3 В/м	10 мкВт/см ²

Таблица 7.7 – Предельно допустимые уровни интенсивности излучения в ультрафиолетовом диапазоне на расстоянии 0,5 м со стороны экрана ВДТ, ЭВМ и ПЭВМ

Диапазоны длин волн	200-280 нм	280-315 нм	315-400 нм
Предельно допустимые уровни	не допускаетс я	0,0001 Вт/м ²	0,1 Вт/м ²

Наиболее эффективным и часто применяемым методом защиты от электромагнитных излучений является установка экранов. Экранируют либо источник излучения, либо рабочее место. Часто экран устанавливают непосредственно на монитор.

При работе монитора на экране кинескопа накапливается электростатический заряд, создающий электростатическое поле. При этом персонал, работающий с монитором, приобретают электростатический потенциал. Заметный вклад в общее электростатическое поле вносят электризующиеся от трения поверхности клавиатуры и мыши.

7.1.7 Пожарная безопасность

По взрывопожарной и пожарной опасности помещения и здания для ЭВМ относятся к категории Д согласно ТКП 474-2013. Здания для ВЦ и части зданий другого назначения, в которых предусмотрено размещение ЭВМ, относятся к 2 степени огнестойкости согласно ТКП 45-2.02-315-2018.

Для предотвращения распространения огня во время пожара с одной части здания на другую устраивают противопожарные преграды в виде стен, перегородок, дверей, окон. Особое требование предъявляется к устройству и размещению кабельных коммуникаций.

Нормы первичных средств пожаротушения для вычислительных центров приведены в табл. 7.8.

Таблица 7.8 – Примерные нормы первичных средств пожаротушения для вычислительного центра

Помещение	Площадь, м ²	Углекислотные огнетушители ручные	Порошковые огнетушители
Вычислительный центр	100	1	1

Для ликвидации пожаров в начальной стадии применяются первичные средства пожаротушения: внутренние пожарные водопроводы, огнетушители типа ОВП-10, ОУ-2, асбестовые одеяла и др.

Эвакуация персонала вычислительного центра осуществляется через эвакуационные выходы. Количество и общая ширина эвакуационных выходов определяются в зависимости от максимального возможного числа эвакуирующихся через них людей и предельно допустимого расстояния от наиболее удаленного места возможного пребывания людей до ближайшего эвакуационного выхода согласно ТКП 45-2.02-315-2018.

Расчетное время эвакуации устанавливается по реальному расчету времени движения одного или нескольких потоков людей через эвакуационные выходы из наиболее удаленных мест размещения людей. Необходимое время эвакуации устанавливается на основе данных о критической продолжительности пожара с учетом степени огнестойкости здания, категории производства по взрывной и пожарной опасности. Для успешной эвакуации необходимо, чтобы расчетное время было меньше необходимого.

7.2 Организация рабочего места пользователя ПЭВМ

Рабочее место - это часть пространства, в котором персонал осуществляет трудовую деятельность, и проводит большую часть рабочего времени. Рабочее место, хорошо приспособленное к трудовой деятельности, правильно и целесообразно организованное, в отношении пространства, формы, размера обеспечивает ему удобное положение при работе и высокую производительность труда при наименьшем физическом и психическом напряжении.

При правильной организации рабочего места производительность труда возрастает с 8 до 20 %. Конструкция рабочего места и взаимное расположение всех его элементов соответствует антропометрическим, физическим и психологическим требованиям ГОСТ 12.2.032-78.

При размещении рабочих мест с ПЭВМ расстояние между рабочими столами с видеомониторами (в направлении тыла поверхности одного видеомонитора и экрана другого видеомонитора) составляет не менее 2,0 м, а расстояние между боковыми поверхностями видеомониторов – не менее 1,2 м.

Рабочие места с ПЭВМ в помещениях с источниками вредных производственных факторов должны размещаться в изолированных кабинах с организованным воздухообменом.

Рабочие места с ПЭВМ при выполнении творческой работы, требующей значительного умственного напряжения или высокой концентрации внимания, рекомендуется изолировать друг от друга перегородками высотой 1,5-2,0 м.

Экран видеомонитора должен находиться на расстоянии 0,6-0,7 м от глаз пользователя, но не ближе 0,5 м с учетом размеров алфавитно-цифровых знаков и символов.

Конструкция рабочего стола обеспечивает оптимальное размещение на рабочей поверхности используемого оборудования с учетом его количества и конструктивных особенностей, характера выполняемой работы. При этом допускается использование рабочих столов различных конструкций, отвечающих современным требованиям эргономики. Поверхность рабочего стола имеет коэффициент отражения 0,5-0,7.

Конструкция рабочего стула (кресла) обеспечивает поддержание рациональной рабочей позы при работе с ПЭВМ, позволяет изменять позу с целью снижения статического напряжения мышц шейно-плечевой области и спины для предупреждения развития утомления. Тип рабочего стула (кресла) выбирают с учетом роста пользователя, характера и продолжительности работы с ПЭВМ. Рабочий стул (кресло) подъемно-поворотный, регулируемый по высоте и углам наклона сиденья и спинки, а расстояние спинки от переднего края сиденья, при этом регулировка каждого параметра независимая, легко осуществляемая и имеет надежную фиксацию. Поверхность сиденья, спинки и других элементов рабочего стула (кресла) полумягкая, с нескользящим, слабо электризующимся и воздухо-проницаемым покрытием, обеспечивающим легкую очистку от загрязнений.

Поверхности периферийных устройств (клавиатура, манипулятор «мышь», принтер, сканер и другое) необходимо протирать мягкой ветошью с применением специальных или бытовых чистящих средств, не содержащих кислот и отбеливателей, не реже 1 раза в неделю, а при необходимости и чаще. Протирка периферийных устройств производится при выключенном оборудовании.

ЗАКЛЮЧЕНИЕ

В процессе выполнения дипломного проекта была создана база данных и веб-сервер для системы управления проектами Ziro, предоставляющий API интерфейс для клиентских частей системы.

В процессе разработки и проектирования веб-сервера учитывались достоинства и недостатки существующих аналогов, а также проанализированы и подобраны подходящие и новейшие технологии и инструменты, позволяющие решить поставленную задачу с наилучшими показателями безопасности, производительности и удобства использования.

Разработанная серверная часть успешно интегрировалась с результатами разработки клиентских частей системы. В результате полученная система была успешно развернута на облачной платформе Azure. Основной функционал доступный при работе с системой:

- аутентификация и авторизация пользователей в системе;
- управление доступом пользователей в режиме администратора;
- создание проектов в режиме администратора;
- редактирование профиля;
- создание и редактирование задач;
- формирования списка задач с помощью большого количества фильтров;
- обсуждение задач;
- назначение исполнителей для задач;
- контроль потраченного времени и ведение журнала работ;
- загрузка документации для проекта;
- просмотр информации по текущей команде и проектам.

Серверная часть разработана с использованием основных принципов программирования (ООП, SOLID, DRY и др.), а также гарантирует высокий уровень безопасности и производительности.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

- 1 Джеффри Рихтер, «CLR Via C#», 1996;
- 2 Дино Эспозито, «Programming ASP.NET Core», 2019;
- 3 Марк Симан, «Внедрение зависимостей в .NET», 2016;
- 4 Никифоров Сергей Николаевич, «Методы защиты информации. Шифрование данных»;
- 5 Джон Даемон, «The Design of Rijndael: AES - The Advanced Encryption Standard (Information Security and Cryptography)», 2005;
- 6 Дилип Найк. «Стандарты и протоколы Интернета, Channel Trading Ltd.», 1999 г.;
- 7 Несеева А.Г. Тексты лекций по дисциплине «Компьютерные системы и сети». Ч.2. - М.: МГТУ ГА, 2003 с;
- 8 «Основы криптографии» - А.П. Алферов, А.Ю. Зубов, А.С. Кузьмин, А.В. Черемушкин;
- 9 Язык программирования C# [Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp/net/3.2.php> - Дата доступа: 04.03.2019;
- 10 Программирование веб-приложений для платформы .Net Framework Core на языке C#. [Электронный ресурс]. – Режим доступа: <http://wm-help.net/lib/b/book/4054355128/757>;
- 11 Санитарные нормы и правила «Требования при работе с видеодисплейными терминалами и электронно-вычислительными машинами» и Гигиенический норматив «Предельно-допустимые уровни нормируемых параметров при работе с видеодисплейными терминалами и электронно-вычислительными машинами», утвержденные постановлением МЗ РБ от 28.06.2013 г. № 59;
- 12 Лазаренков, А. М. Охрана труда в машиностроении: учебное пособие / А. М. Лазаренков. — Минск: ИВЦ Минфина, 2017. — 446 с;
- 13 Лазаренков А.М., Ушакова И.Н. Охрана труда: Учебно-методическое пособие для практических занятий. – Мн.: БНТУ, 2011. – 205 с.