Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики

К защите допустить:

Заведующая кафедрой
информатики

_____ Н. А. Волорова

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему:

СЕРВИС УПРАВЛЕНИЯ СЕРВЕРАМИ

БГУИР ДП 1-40 04 01 00 077 ПЗ

Студент Д. М. Савченко
Руководитель А. Л. Хотеев
Консультанты:
от кафедры информатики А. Л. Хотеев
по экономической части К. Р. Литвинович
Нормоконтролёр Н. Н. Бабенко

Рецензент

Минск 2017

РЕФЕРАТ

ПРОГРАММНОЕ СРЕДСТВО, ВЕБ-ПРИЛОЖЕНИЕ, УНИВЕРСИТЕТ, ОРГАНИЗАЦИЯ УЧЕБНОГО ПРОЦЕССА, РАСПИСАНИЕ, ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

Цель настоящего дипломного проекта состоит в разработке программной системы, предназначенной для эффективной автоматизации задач участников учебного процесса: студентов и преподавателей.

В процессе анализа предметной области были выделены основные аспекты процесса образования в университетах, которые в настоящее время практически не охвачены автоматизацией. Было проведено их исследование и моделирование. Кроме того, рассмотрены существующие средства, разрозненно применяемые сотрудниками университетов и обучаемыми людьми (так называемые частичные аналоги). Выработаны функциональные и нефункциональные требования.

Была разработана архитектура программной системы, для каждой ее составной части было проведено разграничение реализуемых задач проектирование, уточнение используемых технологий и собственно разработка. Были выбраны наиболее современные средства разработки, широко применяемые в индустрии.

Полученные в ходе технико-экономического обоснования результаты о прибыли для разработчика, пользователя, уровень рентабельности, а также экономический эффект доказывают целесообразность разработки проекта.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Львиная доля всех ИТ продуктов так или иначе связано с серверными технологиями. Сервера применяються для огромной области задач. Начиная от взаимодействия играков в онлайн играх и закнчивая пересылкой электронной почты. Так же сервера это большое количество машин и кода. И важно чтобы все это стабильно работало.

Серверное программное обеспечение — в информационных технологиях — программный компонент вычислительной системы, выполняющий сервисные (обслуживающие) функции по запросу клиента, предоставляя ему доступ к определённым ресурсам или услугам.

Понятия сервер и клиент и закреплённые за ними роли образуют программную концепцию «клиент-сервер». Для взаимодействия с клиентом (или клиентами, если поддерживается одновременная работа с несколькими клиентами) сервер выделяет необходимые ресурсы межпроцессного взаимодействия (разделяемая память, пайп, сокет и т. п.) и ожидает запросы на открытие соединения (или, собственно, запросы на предоставляемый сервис). В зависимости от типа такого ресурса, сервер может обслуживать процессы в пределах одной компьютерной системы или процессы на других машинах через каналы передачи данных (например, COM-порт) или сетевые соединения. Формат запросов клиента и ответов сервера определяется протоколом. Спецификации открытых протоколов описываются открытыми стандартами, например, протоколы Интернета определяются в документах RFC. В зависимости от выполняемых задач одни серверы, при отсутствии запросов на обслуживание, могут простаивать в ожидании. Другие могут выполнять какую-то работу (например, работу по сбору информации), у таких серверов работа с клиентами может быть второстепенной задачей.

1 ПОСТАНОВКА ЗАДАЧИ

Система управления серверами объедняет в себе функцианальность нескольких сервесов. В оддельности каждый сервис очень полезен, а суммарно получается мощный инструмент для системного администратора. Это програмное средство в основном написанное на Node + Angular. Каждый пользователь устанавливает програвное стредство на свой сервер. С помощью веб-приложения поьзаватель может просмотреть всю информацию о своих серверах в удобном виде, а также управлять ими, изменяя конфиги.

1.1 Необходимость разработки сервиса

Использование сервиса будет удобным для клиентов по следующим причинам:

– все заказчики компании имеют собственную принятую систему именования задач проектов, не отличающуюся простотой и наглядностью. «Имя» приходит в теле письма корпоративной почты и менеджер должен самостоятельно создавать проекты, юниты, категории и, наконец, задачи с определенными именами;

– система не предусматривает возможности напоминания об окончании срока, отведенного на выполнение задачи, кроме как выделение цветом в общем списке заданий пользователя;

– интерфейс программы не является достаточно удобным для пользователя, так как приложение консольное;

– система имеет бреши в наборе правил для проверки введенных данных;

– для обработки записей журналов и проведения бухгалтерских операций требуется осуществить немалый набор действий в системе.

Учитывая специфику задачи, было решино в основе использовать Node. Данное решение было принято по следующим причинам:

– все заказчики компании имеют собственную принятую систему именования задач проектов, не отличающуюся простотой и наглядностью. «Имя» приходит в теле письма корпоративной почты и менеджер должен самостоятельно создавать проекты, юниты, категории и, наконец, задачи с определенными именами;

– система не предусматривает возможности напоминания об окончании срока, отведенного на выполнение задачи, кроме как выделение цветом в общем списке заданий пользователя;

– интерфейс программы не является достаточно удобным для пользователя, так как приложение консольное;

– система имеет бреши в наборе правил для проверки введенных данных;

– для обработки записей журналов и проведения бухгалтерских операций требуется осуществить немалый набор действий в системе.

1.2 Формулировка задания

Передо мной была поставлена задача: проанализировать функциональность используемой системы учета времени, изучить модуль «Управление проектами и учет» в Microsoft Dynamics AX 2012, участвовать в разработке системы учета времени с функциональными возможностями, отвечающими требованиям компании и реализовать подсистему расписаний для непосредственного ведения учета рабочего времени в Microsoft Dynamics AX 2012. Таким образом, разработанная система является программным продуктом, который может быть легко импортирован в любую систему Microsoft Dynamics AX 2012 и данный продукт соответствует заявленным требованиям:

– возможность создания проектов и задач, назначения исполнителей, установления бюджета задачи, сроков исполнения, контроль статуса задачи;

– возможность создания расписаний работы и/или свободной регистрации рабочего времени с требуемыми проверками сроков исполнения и превышения бюджета (записи в журнале компании при превышении бюджета);

– возможность создания проекта по его названию (записи в таблице проектов) с регистрацией для него любых необходимых элементов AOT в соответствии с правилами создания проектов в Axapta (функция JobLog).

2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

При проектировании програмного продукта, который решает реальные задачи пользователя, очень важно выяснить главный фукционал и свойства продукта, чтобы сервис отвечал запросам пользователей и производство продукта было аправдано.

Требования к ПО определяют, какие свойства и характеристики оно должно иметь для удовлетворения потребностей пользователей и других заинтересованных лиц. Однако сформулировать требования к сложной системе не так легко. В большинстве случаев будущие пользователи могут перечислить набор свойств, который они хотели бы видеть, но никто не даст гарантий, что это — исчерпывающий список. Кроме того, часто сама формулировка этих свойств будет непонятна большинству программистов: могут прозвучать фразы типа "должно использоваться и частотное, и временное уплотнение каналов "передача клиента должна быть мягкой "для обычных швов отмечайте бригаду, а для доверительных — конкретных сварщиков и это еще не самые тяжелые для понимания примеры.

Чтобы ПО было действительно полезным, важно, чтобы оно удовлетворяло реальные потребности людей и организаций, которые часто отличаются от непосредственно выражаемых пользователями желаний. Для выявления этих потребностей, а также для выяснения смысла высказанных требований приходится проводить достаточно большую дополнительную работу, которая называется анализом предметной области или бизнес-моделированием, если речь идет о потребностях коммерческой организации. В результате этой деятельности разработчики должны научиться понимать язык, на котором говорят пользователи и заказчики, выявить цели их деятельности, определить набор задач, решаемых ими. В дополнение стоит выяснить, какие вообще задачи нужно уметь решать для достижения этих целей, выяснить свойства результатов, которые хотелось бы получить, а также определить набор сущностей, с которыми приходится иметь дело при решении этих задач. Кроме того, анализ предметной области позволяет выявить места возможных улучшений и оценить последствия принимаемых решений о реализации тех или иных функций.

После этого можно определять область ответственности будущей программной системы — какие именно из выявленных задач будут ею решаться, при решении каких задач она может оказать существенную помощь и чем именно. Определив эти задачи в рамках общей системы задач и деятельностей пользователей, можно уже более точно сформулировать требования к ПО.

Анализом предметной области занимаются системные аналитики или бизнес-аналитики, которые передают полученные ими знания другим членам проектной команды, сформулировав их на более понятном разработчикам языке. Для передачи этих знаний обычно служит некоторый набор моделей, в виде графических схем и текстовых документов.

Анализ деятельности крупной организации, такой как банк с сетью региональных отделений, нефтеперерабатывающий завод или компания, производящая автомобили, дает огромные объемы информации. Из этой информации надо уметь отбирать существенную, а также уметь находить в ней пробелы — области деятельности, информации по которым недостаточно для четкого представления о решаемых задачах. Значит, всю получаемую информацию надо каким-то образом систематизировать. Для систематизации сбора информации о больших организациях и дальнейшей разработки систем, поддерживающих их деятельность, применяется схема Захмана (автор — John Zachman, [1,2]) или архитектурная схема предприятия (enterprise architecture framework).

Далее приводится анализ сведений, которые влияют на формулирование требований, выбор архитектуры и дальнейшее проектирование и разработку программного средства.

Главная задача сервеса взаимодействовать с серверами. Для начала нужно понять какие они современные сервера. Т.е. следует изучить стек технологий, а также понять, какие параметры важны для пользователя.

2.1 Серверная архитиктура

Соверменная серверная архитектура направлена на улучшения производительности продукта. Сейчас существуют два основные вида архетектуры: микросервисы и монолитные сервера.

«Микросервисы» — еще один новый термин на шумных улицах разработки ПО. И хотя мы обычно довольно настороженно относимся ко всем подобным новинкам, конкретно этот термин описывает стиль разработки ПО, который находят все более и более привлекательным. За последние несколько лет мы видели множество проектов, использующих этот стиль, и эти результаты весьма позитивными.

Если коротко, то архитектурный стиль микросервисов — это подход, при котором единое приложение строится как набор небольших сервисов, каждый из которых работает в собственном процессе и коммуницирует с остальными используя легковесные механизмы, как правило HTTP. Эти сервисы построены вокруг бизнес-потребностей и развертываются независимо

с использованием полностью автоматизированной среды. Существует абсолютный минимум централизованного управления этими сервисами. Сами по себе эти сервисы могут быть написаны на разных языках и использовать разные технологии хранения данных.

Монолитный сервер — довольно очевидный способ построения подобных систем. Вся логика по обработке запросов выполняется в единственном процессе, при этом вы можете использовать возможности вашего языка программирования для разделения приложения на классы, функции и namespace-ы. Вы можете запускать и тестировать приложение на машине разработчика и использовать стандартный процесс развертывания для проверки изменений перед выкладыванием их в продакшн. Вы можете масштабировать монолитное приложения горизонтально путем запуска нескольких физических серверов за балансировщиком нагрузки.

Монолитные приложения могут быть успешными, но все больше людей разочаровываются в них, особенно в свете того, что все больше приложений развертываются в облаке. Любые изменения, даже самые небольшие, требуют пересборки и развертывания всего монолита. С течением времени, становится труднее сохранять хорошую модульную структуру, изменения логики одного модуля имеют тенденцию влиять на код других модулей. Масштабировать приходится все приложение целиком, даже если это требуется только для одного модуля этого приложения.

2.2 Apache и Nginx

В трендах серверной разработки находятся Apache и Nginx. Это два самых распространенных веб-сервера с открытым исходным кодом в мире. Вместе они обслуживают более 50% трафика во всем интернете. Оба решения способны работать с разнообразными рабочими нагрузками и взаимодействовать с другими приложениями для реализации полного веб-стека.

Несмотря на то, что у Apache и Nginx много схожих качеств, их нельзя рассматривать как полностью взаимозаменямые решения. Каждый из них имеет собственные преимущества и важно понимать какой веб-сервер выбрать в какой ситуации. В этой статье описано то, как каждый из этих веб-серверов ведет себя при различных условиях.

Apache HTTP Server был разработан Робертом Маккулом в 1995 году, а с 1999 года разрабатывается под управлением Apache Software Foundation — фонда развития программного обеспечения Apache. Так как HTTP сервер это первый и самый популярный проект фонда его обычно называют просто Apache.

Веб-север Apache был самым популярным веб-сервером в интернете с 1996 года. Благодаря его популярности у Apache сильная документация и интеграция со сторонним софтом.

Администраторы часто выбирают Apache из-за его гибкости, мощности и широкой распространенности. Он может быть расширен с помощью системы динамически загружаемых модулей и исполнять программы на большом количестве интерпретируемых языков программирования без использования внешнего программного обеспечения.

Nginx В 2002 году Игорь Сысоев начал работу над Nginx для того чтобы решить проблему C10K — требование к ПО работать с 10 тысячами одновременных соединений. Первый публичный релиз был выпущен в 2004 году, поставленная цель была достигнута благодаря асинхронной event-driven архитектуре.

Nginx начал набирать популярность с момента релиза благодаря своей легковесности (light-weight resource utilization) и возможности легко масштабироваться на минимальном железе. Nginx превосходен при отдаче статического контента и спроектирован так, чтобы передавать динамические запросы другому ПО предназначенному для их обработки.

Администраторы часто выбирают Nginx из-за его эффективного потребления ресурсов и отзывчивости под нагрузкой, а также из-за возможности использовать его и как веб-сервер, и как прокси.

2.3 Логирование

Естественной потребностью системного администратора или специалиста по безопасности является некий анализ того, что происходит как на конкретном компьютере конкретного пользователя, так и в локальной сети. Технически задача выполнима, ибо разработчики множества приложений, которыми мы пользуемся, заложили в свои продукты функцию логирования информации*. Информация, которую хранят логи* конкретного компьютера в сети, может сказать много тому, кто, с некоторым знанием предмета, рискнет заглянуть внутрь. Нельзя сказать, что чтение логов* является тайной дисциплиной, которая доступна только просвященным гуру, впрочем, для того, чтобы легко ориентироваться и четко сопоставлять информацию, которая встречается в логах* различных приложений, надо действительно иметь представление о том, что и как, почему и зачем пишется в логи*, а кроме того, четко представлять предметную область изучаемого ПО. Дело в том, что запись информации в логи* (вероятно, в силу некой меньшей приоритетности, чем работа самого приложения) страдает хаотичными веяниями

различных производителей. Соответственно, и интерпретировать такую информацию надо с учетом специфики и, может быть, каких-то рекомендаций производителя.

Для того чтобы грамотно добывать полезную информацию из логов*, иногда достаточно простого текстового редактора и головы, но часто встречаются ситуации, когда лог* и просмотреть довольно сложно, и трактовать правильно тяжело. В этом случае полезно знать о некоторых особенностях структуры различных лог*-файлов и об информации, которая в них встречается.

2.4 Базы данных

Можно с большой степенью достоверности утверждать, что большинство приложений, которые предназначены для выполнения хотя бы какой-нибудь полезной работы, тем или иным образом используют структурированную информацию или, другими словами, упорядоченные данные. Такими данными могут быть, например, списки заказов на тот или иной товар, списки предъявленных и оплаченных счетов или список телефонных номеров ваших знакомых. Обычное расписание движения автобусов в вашем городе - это тоже пример упорядоченных данных.

При компьютерной обработке информации упорядоченные каким либо образом данные принято хранить в базах данных - особых файлах, использование которых вместе со специальными программными средствами позволяет пользователю как просматривать необходимую информацию, так и, по мере необходимости, манипулировать ею.

Существует два основных виды баз данныйх: реляционные и нереляционные(NoSQL).

Реляционной модели данных несколько десятков лет. В реляционной модели база состоит из таблиц, которые состоят из строк и колонок. У каждой колонки есть свой тип данных (строка, число, логическое значение, дата, текст, бинарный блоб). Все строки однотипны.

Обычно каждый вид объектов хранится в отдельной таблице (например, таблица пользователей или таблица проектов). Обычно у каждого объекта есть уникальный идентификатор. Идентификатор может быть как условным, т. е. просто числом, так и вытекающим из предметной области, например номер паспорта человека или ISBN для книг. Обычно пользуются условными идентификаторами.

Объекты (таблицы) могут быть связаны друг с другом с помощью идентификатора. Например, если у нас есть таблица отделов и таблица со-

трудников, то в таблице отделов есть идентификатор отдела, а в таблице сотрудников есть идентификатор сотрудника и идентификатор отдела, к которому он принадлежит. В теории реляционных баз данных этот случай называется “один ко многим” (одному отделу принадлежит много сотрудников).

Возможен также случай “многие ко многим”. Например, есть таблица проектов и таблица разработчиков. Над одним проектом могут работать много разработчиков, и один разработчик может работать над несколькими проектами. В этом случае обычно создается третья таблица — таблица связей с двумя полями: идентификатором проекта и идентификатором разработчика. Каждая связь между разработчиком и проектом выражается в виде строки в таблице связей. Если разработчик пока еще не назначен ни на один проект, то в таблице связей просто не будет ни одной записи про него.

Серверы реляционных БД обеспечивают стандартные операторы доступа к данным в таблицах, такие как SELECT, INSERT, UPDATE и DELETE. Разные серверы предоставляют также некоторые дополнительные операторы. Извлекать данные из таблиц можно по множеству различных критериев. Есть “ядро” стандарта SQL, который поддерживается практически всеми серверами, и всегда есть те или иные расширения стандарта, которые можно использовать при работе с конкретным сервером БД.

Одно из значений термина “NoSQL” — это отход от реляционной модели в пользу более специфических (или более обобщенных) моделей данных. Например, традиционно успешными NoSQL-системами являются системы хранения пар “ключ-значение”, такие как Redis или Memcache. Их модель данных предельно проста — это в сущности ассоциативный массив, где ключи имеют строковый тип, а значения могут содержать любые данные. Как и любой ассоциативный массив, такие системы поддерживают ограниченный набор операций с данными — прочитать значение по ключу, установить значение ключа, удалить ключ и связанное с ним значение. Операция “получить список ключей” может не поддерживаться в таких системах.

Другой пример успешных NoSQL-систем — это документные хранилища. Объекты в таких хранилищах обычно являются ассоциативными массивами свободной структуры, то есть в одной и той же “таблице” могут храниться разные по сути объекты. Примеры систем такого класса — MongoDB и Cassandra. В зависимости от того, какие реально данные хранятся в конкретной базе, ее производительность может сильно варьироваться. Например, если оптимизировать такую “таблицу”, храня в ней однотипные объ-

екты,

Третий пример специализированных NoSQL-систем — это графовые базы данных. Они специальным образом заточены под обработку конкретной структуры данных, причем обычно для работы с большим объемом данных (потому что на небольших объемах может прекрасно справиться стандартная реляционная реализация).

Очень важным примером NoSQL-систем являются обычные файловые системы, такие как Ext4 или NTFS. Они предназначены для хранения объектов в виде иерархической структуры с содержимым свободного формата. Сами базы данных, реляционные и NoSQL, обычно используют для хранения своего содержимого именно файловые системы, и иногда взаимодействие между этими двумя подсистемами становится важным в том или ином случае.

Еще один важный случай — системы полнотекстового поиска, такие как Elastic Search или Google Search Engine.

2.5 Docker

Докер — это открытая платформа для разработки, доставки и эксплуатации приложений. Docker разработан для более быстрого выкладывания ваших приложений. С помощью docker вы можете отделить ваше приложение от вашей инфраструктуры и обращаться с инфраструктурой как управляемым приложением. Docker помогает выкладывать ваш код быстрее, быстрее тестировать, быстрее выкладывать приложения и уменьшить время между написанием кода и запуска кода. Docker делает это с помощью легковесной платформы контейнерной виртуализации, используя процессы и утилиты, которые помогают управлять и выкладывать ваши приложения.

В своем ядре docker позволяет запускать практически любое приложение, безопасно изолированное в контейнере. Безопасная изоляция позволяет вам запускать на одном хосте много контейнеров одновременно. Легковесная природа контейнера, который запускается без дополнительной нагрузки гипервизора, позволяет вам добиваться больше от вашего железа.

Платформа и средства контейнерной виртуализации могут быть полезны в следующих случаях: упаковывание вашего приложения (и так же используемых компонент) в docker контейнеры; раздача и доставка этих контейнеров вашим командам для разработки и тестирования; выкладывания этих контейнеров на ваши продакшены, как в дата центры так и в облака.

Docker прекрасно подходит для организации цикла разработки. Docker

позволяет разработчикам использовать локальные контейнеры с приложениями и сервисами. Что в последствии позволяет интегрироваться с процессом постоянной интеграции и выкладывания (continuous integration and deployment workflow).

Например, ваши разработчики пишут код локально и делятся своим стеком разработки (набором docker образов) с коллегами. Когда они готовы, отравляют код и контейнеры на тестовую площадку и запускают любые необходимые тесты. С тестовой площадки они могут оправить код и образы на продакшен

Основанная на контейнерах docker платформа позволят легко портировать вашу полезную нагрузку. Docker контейнеры могут работать на вашей локальной машине, как реальной так и на виртуальной машине в дата центре, так и в облаке.

Портируемость и легковесная природа docker позволяет легко динамически управлять вашей нагрузкой. Вы можете использовать docker, чтобы развернуть или погасить ваше приложение или сервисы. Скорость docker позволяет делать это почти в режиме реального времени.

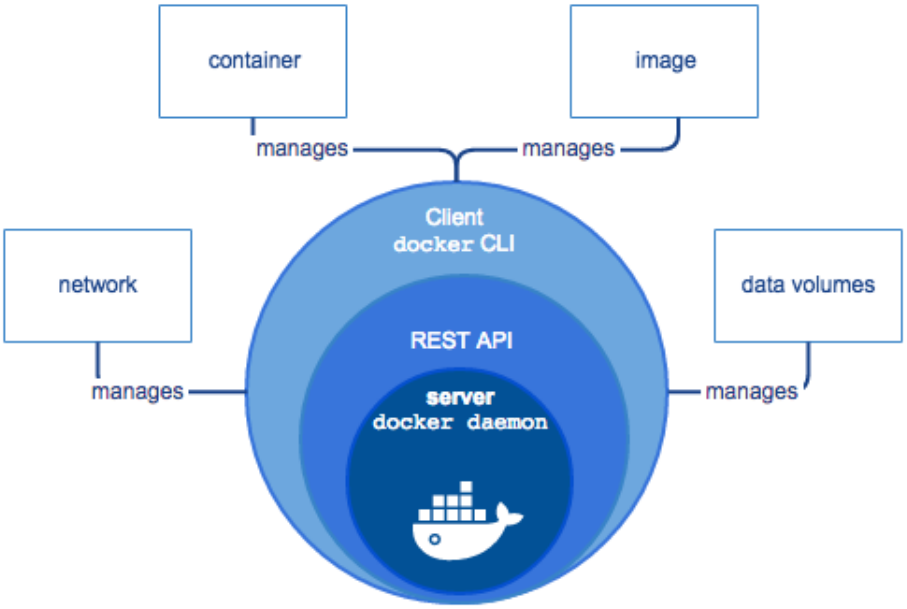

Рисунок 2.1 – Конь

Docker легковесен и быстр. Он предоставляет устойчивую, рентабельную альтернативу виртуальным машинам на основе гипервизора. Он особенно полезен в условиях высоких нагрузок, например, при создания собственного облака или платформа-как-сервис (platform-as-service). Но он так же полезен для маленьких и средних приложений, когда вам хочется получать больше из имеющихся ресурсов.

Docker состоит из двух главных компонент:

– docker: платформа виртуализации с открытым кодом;

– docker Hub: наша платформа-как-сервис для распространения и управления docker контейнерами;

2.6 Memcached и Redis

Memcached — программное обеспечение, реализующее сервис кэширования данных в оперативной памяти на основе хеш-таблицы. С помощью клиентской библиотеки (для C/C++, Ruby, Perl, PHP, Python, Java, .Net и др.) позволяет кэшировать данные в оперативной памяти множества доступных серверов. Распределение реализуется путём сегментирования данных по значению хэша ключа по аналогии с сокетами хэш-таблицы. Клиентская библиотека, используя ключ данных, вычисляет хэш и использует его для выбора соответствующего сервера. Ситуация сбоя сервера трактуется как промах кэша, что позволяет повышать отказоустойчивость комплекса за счет наращивания количества memcached серверов и возможности производить их горячую замену. В API memcached есть только базовые функции: выбор сервера, установка и разрыв соединения, добавление, удаление, обновление и получение объекта, а также Compare-and-swap. Для каждого объекта устанавливается время жизни, от 1 секунды до бесконечности. При исчерпании памяти более старые объекты автоматически удаляются. Для PHP также есть уже готовые библиотеки PECL для работы с memcached, которые дают дополнительную функциональность.

В первом приближении может показаться, что Redis мало чем отличается от Memcached. И действительно, как Redis, так и Memcached хранят данные в памяти и осуществляют доступ к ним по ключу. Оба написаны на Си и распространяются под лицензией BSD. Но в действительности, между Redis и Memcahced больше различий, чем сходства.

В первую очередь, Redis умеет сохранять данные на диск. Можно настроить Redis так, чтобы данные вообще не сохранялись, сохранялись периодически по принципу copy-on-write, или сохранялись периодически и писались в журнал (binlog). Таким образом, всегда можно добиться требуе-

мого баланса между производительностью и надежностью.

Redis, в отличие от Memcached, позволяет хранить не только строки, но и массивы (которые могут использоваться в качестве очередей или стеков), словари, множества без повторов, большие массивы бит (bitmaps), а также множества, отсортированные по некой величине. Разумеется, можно работать с отдельными элементами списков, словарей и множеств. Как и Memcached, Redis позволяет указать время жизни данных (двумя способами — «удалить тогда-то» и «удалить через . . . »). По умолчанию все данные хранятся вечно.

Интересная особенность Redis заключается в том, что это — однопоточный сервер. Такое решение сильно упрощает поддержку кода, обеспечивает атомарность операций и позволяет запустить по одному процессу Redis на каждое ядро процессора. Разумеется, каждый процесс будет прослушивать свой порт. Решение нетипичное, но вполне оправданное, так как на выполнение одной операции Redis тратит очень небольшое количество времени (порядка одной стотысячной секунды). В Redis есть репликация. Репликация с несколькими главными серверами не поддерживается. Каждый подчиненный сервер может выступать в роли главного для других. Репликация в Redis не приводит к блокировкам ни на главном сервере, ни на подчиненных. На репликах разрешена операция записи. Когда главный и подчиненный сервер восстанавливают соединение после разрыва, происходит полная синхронизация (resync).

Также Redis поддерживает транзакции (будут последовательно выполнены либо все операции, либо ни одной) и пакетную обработку команд (выполняем пачку команд, затем получаем пачку результатов). Притом ничто не мешает использовать их совместно.

Еще одна особенность Redis — поддержка механизма publish/subscribe. С его помощью приложения могут создавать каналы, подписываться на них и помещать в каналы сообщения, которые будут получены всеми подписчиками. Что-то вроде IRC-чатика.

2.7 Jenkins

Jenkins – это инструмент непрерывной интеграции, который чаще всего используется для разработки программного обеспечения. Это среда автоматизации, которая выполняет повторяющиеся задания. Jenkins может выполнять и контролировать выполнение команд на удаленных системах, а также всего того, что можно выполнить из командной строки.

Непрерывная интеграция (continuous integration) — это очень, очень

хорошо. Вы настраиваете ее один раз, и все все процессы интеграции происходят без вашего участия. Некоторые плюсы использования Jenkins:

– когда кто-то ломает билд, вы узнаете об этом сразу, что позволяет быстро устранить проблему.

– вы можете автоматизировать прогон тестов, деплой приложения на тестовый сервер, проверку code style и тому подобные вещи.

– также в Jenkins можно хранить собранные deb-пакеты, отчеты о прогоне тестов или Javadoc/Doxygen/EDoc-документацию.

3 АНАЛИЗ ПРОТОТИПОВ

3.1 Ajenti

Визитной карточкой панели Ajenti служит приятный интерфейс, реализованный с использованием AJAX. Мы получаем понятную среду, не перегруженную установками и настройками, в которой легко освоится администратор, имеющий относительно небольшой опыт. Архитектура модульная, в настоящее время доступны плагины, позволяющие производить настройку и мониторинг самой системы и некоторых популярных сервисов: системных параметров — сети и UPS/питания, пакетных менеджеров (APT, Zypper, Pacman), учетных записей пользователей и групп (/etc/passwd и /etc/group), заданий cron, монтирования дисковых разделов (/etc/fstab), работы upstart, rc.d, init.d и lm-sensors, настройка DNS (/etc/resolv.conf и /etc/hosts), правил Netfilter, просмотр журналов; серверов и сервисов — веб (Apache 2, nginx и lighttpd), Samba, MySQL, PostgreSQL, DHCPD, BIND9, NFSD, Squid и SARG, Bacula и других. В Ajenti нет каких-либо мастеров, которые помогут настроить сервис в пошаговом режиме, поэтому необходимо представлять процесс и параметры. В большинстве случаев плагин предлагает удобную форму для доступа к конфигурационным файлам, частично автоматизируя некоторые операции. Но интерфейс содержит все предварительные настройки, поэтому часто необходимо лишь заполнить предложенные поля. К тому же новичку будет удобнее править конфиги через браузер, нежели изучать особенности работы с vi. Например, для веб-сервера можно быстро создать виртуальный сайт, буквально одной кнопкой, но заполнять параметры придется самостоятельно.

3.2 ISPConfig

ISPConfig — панель управления хостингом для Linux, которая позволяет настраивать новые веб-сайты, аккаунты электронной почты (POP3, IMAP — Courier, Dovecot), FTP (PureFTPd), записи DNS (BIND, MyDNS), MySQL и виртуализацию OpenVZ. При помощи одного интерфейса поддерживается управление несколькими физическими серверами. Обеспечивает настройку виртуального хостинга на основе IP или доменных имен для Apache2 или nginx, работает с PHP через mod_php, suPHP, fastcgi или PHP-FPM. Поддерживает настройки для FTP, SFTP, SCP и для Apache2 Ruby, Python и WebDAV, реализована проверка на спам (whitelists, blacklists, проверка заголовков и контент-фильтр) и вирусы для входящей почты, email-автоответчик, сбор статистики (Webalizer и/или AWStats), настройка firewall

(UFW или bastille), выполнение заданий по расписанию (cron, jailed cron, web cron). Возможно использование shell-доступа для пользователей (обычный и jail), SFTP, SCP, авторизация по паролю или ключу. Для DNS-сервера возможно создание записей типа A, AAAA, ALIAS, CNAME, HINFO, MX, NS, PTR, RP, SRV, TXT. Поддерживает IPv4 и IPv6. Клиенты могут управлять базами данных MySQL при помощи утилиты phpMyAdmin. Функциональность расширяется при помощи аддонов, правда, некоторые из них предлагаются за дополнительную плату. На сегодня предложен биллинг-модуль, приложение для мониторинга работы на Android (Monitor App for Android), плагины RoundCube, SquirrelMail, Exchange и VMware. Archipel

По возможностям опенсорсные системы виртуализации вполне могут сравниться с коммерческими, но явно уступают в простоте развертывания и управления. Собственно, так было всегда, *nix-программы строятся как бы из блоков, и каждый собирает себе систему по своему усмотрению, в том числе и подбирает нужный GUI, если в нем есть необходимость. И конечно, со временем появляются соответствующие разработки.

3.3 Archipel

Archipel — масштабируемое решение для удобного управления с помощью графического интерфейса гипервизорами и виртуальными машинами, размещенными на локальном и удаленных физических серверах. Для обмена сообщениями используется протокол XMPP, это позволяет Archipel работать в реальном времени, все ответы хостов или систем сразу отображаются в интерфейсе. К тому же для управления системами также можно использовать любые XMPP-клиенты. Состоит из двух частей: интерфейса, написанного при помощи JavaScript, и агента, который установлен на все гипервизоры KVM, Xen, OpenVZ или VMware. Для запуска интерфейса понадобится любой веб-сервер и сервер ejabberd (XMPP). Модули PHP, Ruby или SQL базы данных не требуются. Возможно использование нескольких XMPP как реплики или различные точки доступа. Интерфейс позволяет оценить состояние всех VM, собранных в одном месте, при большом их количестве отобрать нужные можно при помощи фильтров. Новые VM создаются буквально одним кликом. При этом новым VM имя может быть дано автоматически (вместо непонятного сочетания букв и цифр используются астероиды Солнечной системы). Существующие VM легко подключить к интерфейсу управления, для этого на гипервизор достаточно установить агент. Поддерживаются все основные команды управления VM (старт/стоп/-пауза), управление сетью, DHCP, планировщик, снапшоты и Live migration

на другой хост. Выводится статистика в реальном времени об использовании памяти, загрузке CPU, месте на диске, средней загрузке и прочем. Журналы и модуль Health позволяют быстро найти причину проблем. К удаленным системам можно подключаться при помощи встроенного VNC-клиента (JavaScript). Виртуальные машины можно упаковать в пакеты и перенести на другой узел.

3.4 Gosa

Сегодня доступны проекты, которые предоставляют администратору единый центр управления всей ИТ-инфраструктурой. Одним из самых продвинутых решений можно назвать GOsa2, который представляет собой LDAP-ориентированную систему, позволяющую управлять учетными записями *nix и Samba, правами пользователей и групп, компьютерами, списками рассылок, телефонами и факсами, приложениями и настройками основных сетевых служб: DHCP, DNS, HTTP, SMTP и многих других. Возможностей много, но для удобства все функции вынесены в плагины, поэтому конфигурация собирается под конкретные требования и не содержит ничего лишнего. В настоящее время реализовано более 30 плагинов, обеспечивающих управление такими сервисами, как Squid, DansGuardian, Postfix, Courier-IMAP, Maildrop, GNARWL, Cyrus-SASL, OpenSSL, ISC DHCP, WebDAV, PureFTPd, PPTP, Kerberos, Asterisk, Nagios, OPSI, Netatalk, FAI, rsyslog, серверами коллективной работы — SOGo, OpenGroupware, Kolab, Scalix. Все сервисы могут работать на разных серверах. Учетные записи пользователей объединяются в группы, которым назначаются разрешенные приложения. При создании новых аккаунтов применяются шаблоны с прописанными правами доступа к объектам. Набор разрешений ACL состоит из типа, определяющего видимость, объектов (пользователей/групп) и разрешений. Разрешения определяют все возможные действия — создание, удаление, перемещение, чтение, запись и так далее. Интерфейс локализован, настройки сводятся к заполнению предложенных параметров, поэтому ошибиться трудно.

4 ТРЕБОВАНИЯ К ПРОЕКТИРУЕМОЙ СИСТЕМЕ

По результатам изучения предметной области, анализа литературных источников, выбора технологий реализации и обзора существующих систем-аналогов сформулируем общие и функциональные требования к проектируемому программному средству.

4.1 Общие требования

Назначение разработки Итогом дипломного проектирования должен быть проект, позволяющий пользователям управлять своими серверами через web-интерфейс. Продукт должен быть интересен большому и среднему бизнесу, который хочет сократить время на администрирования своих серверов. Возможность строить графики различной сложности для удобного и быстрого анализа.

Входные данные Входными данными являются данные с серверов. Они получаются с помощью софта установленного на сервера клиента. При создании графиков на сайте, пользователь должен иметь возможность указывать интересующие его параметры событий, по этим параметрам строятся графики.

Выходные данные Выходными данными можно считать информацию, которую получают владельцы приложений-потребителей о всех действия и всей информации с серверов. Функционал позволяющий строить разнообразные графики, позволяющий анализировать информацию и сообщать о нарушении работы.

Требования к временным характеристикам Сохранение событий должно происходить быстро и не вызывать задержек в приложении-потребителе, так как это может вызывать недовольство и желание отказаться от текущего продукта.

Требования к надежности Система должна работать без остановок, так как сервера имеют большую важность для клиента.

Требования к составу и параметрам технических и программных средств Node для сервера. Angular для клиента. nxinx + docker - веб-сервера для поддержки клиента.

4.2 Функциональные требования

К разрабатываемой автоматизированной системе выдвигаются следующие функциональные требования.

1 При открывании главной страницы сайта перед пользователем должна появиться страница с приветствием и информацией о системе. На этой же странице пользователь сможет найти ссылки на страницы создания приложения и входа в своё приложение. Опции «Create new app» и «Login in app» представлены кнопками, при нажатии на которые пользователь перенаправляется на соответствующие страницы для совершения дальнейших действий. В верхней части страницы присутсвует хедер, на котором находится несколько полезный ссылок: Переход на главную страницу, переход на страницу документации.

2 Страница создания приложения содержит поля «Email», «Password», «Confirm password», кнопку «Create». Нажатие на кнопку «Create» в случае успеха осуществляет перенаправление пользователя на страницу с информацией о том что на электронный ящик вышлено письмо для активации аккаунта. Для активации аккаунта необходимо перейти по ссылке в письме. После этого пользователь имеет возможность заходить в свое приложение. Сразу после активации происходит перенаправление на страницу приложения.

3 Создание приложения осуществляется путем ввода запрашиваемой информации в форму. Все поля на форме обязательны для заполнения. Пользователь обязан ввести почту, пароль, подтвердить пароль. Пароль должен содержать не менее 6 знаков. Поля «Password» и «Confirm password» должны совпадать. Если приложение с такой почтой уже есть в системе, то должно быть выведено сообщение «Приложение с таким логином уже существует».

4 Окно авторизации содержит поля «Email» и «Password», а также кнопку «Войти». Нажатие на кнопку «Войти» в случае успеха осуществляет перенаправление пользователя на страницу приложения.

5 Авторизация приложения осуществляется путем ввода запрашиваемой информации в форму авторизации. Все поля на форме авторизации обязательны для заполнения. Если приложения с такой почтой и паролем нет в системе, то выводится сообщение «Приложения с такой комбинацией почты и пароля не существует».

6 На странице приложения находятся ссылки на настройки приложения. Основное поле занимает рабочая форма для составления графиков. При нажатии на ссылку “настройки приложения” переходит переход на страницу настроек.

7 Главные интерфейс представляет возможность выбора информации по разделам. Раздел с логами. Раздел с docker информацией. Раздел с n-ginx информацией.

5 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ

5.1 Js

JavaScript изначально создавался для того, чтобы сделать web-странички «живыми». Программы на этом языке называются скриптами. В браузере они подключаются напрямую к HTML и, как только загружается страничка – тут же выполняются.

Программы на JavaScript – обычный текст. Они не требуют какой-то специальной подготовки.

В этом плане JavaScript сильно отличается от другого языка, который называется Java.

JavaScript может выполняться не только в браузере, а где угодно, нужна лишь специальная программа – интерпретатор. Процесс выполнения скрипта называют «интерпретацией».

Во все основные браузеры встроен интерпретатор JavaScript, именно поэтому они могут выполнять скрипты на странице. Но, разумеется, JavaScript можно использовать не только в браузере. Это полноценный язык, программы на котором можно запускать и на сервере, и даже в стиральной машинке, если в ней установлен соответствующий интерпретатор.

5.2 Angular

AngularJS представляет собой opensource JavaScript-фреймворк, использующий шаблон MVC. Собственно использование MVC является его одной из отличительных особенностей.

Для описания интерфейса используется декларативное программирование, а бизнес-логика отделена от кода интерфейса, что позволяет улучшить тестируемость и расширяемость приложений.

Другой отличительной чертой фреймворка является двустороннее связывание, позволяющее динамически изменять данные в одном месте интерфейса при изменении данных модели в другом. Таким образом, AngularJS синхронизирует модель и представление.

Кроме того, AngularJS поддерживает такие функциональности, как Ajax, управление структорой DOM, анимация, шаблоны, маршрутизация и так далее. Мощь фреймворка, наличие богатого функционала во многом повлияла на то, что он находит свое применение во все большем количестве веб-приложений, являясь на данный момент наверное одним из самых популярных javascript-фреймворков.

Официальный сайт фреймворка: http://angularjs.org/. Там вы можете найти сами исходные файлы, обучающие материалы и другие сопроводительные материалы относительно библиотеки.

5.3 Node

Чуть более недели назад на хабре появилась статья, в которой затрагивалась «проблема»: Node.js — это JavaScript или нет. Некоторые аргументы, представленные в статье были справедливыми, но, увы, безосновательны. Другие же аргументы были вовсе абсурдными и не правдивыми. Я не буду писать о знаниях автора статьи в данной области, даже не буду давать ссылки на это статью (дабы статья перенесена в черновики, она осталась только в архивах). Я же просто сравню скрипты Node.js и JavaScript в таком виде, в котором все его привыкли видеть.

Введение

Для начала обратимся к Википедии и узнаем, что есть такое Node.js и JavaScript: Node или Node.js — серверная реализация языка программирования JavaScript, основанная на движке V8. Предназначена для создания масштабируемых распределённых сетевых приложений, таких как веб-сервер. Node.js по целям использования сходен с каркасами Twisted на языке Python и EventMachine на Ruby. В отличие от большинства программ JavaScript, этот каркас исполняется не в браузере клиента, а на стороне сервера. JavaScript — прототипно-ориентированный сценарный язык программирования. Является диалектом языка ECMAScript.

Что ж, определение Node.js немного расплывчато, и надо сказать, не корректно. Тогда посмотрим на информацию на официальном сайте: Node.js is a platform built on Chrome’s JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Самое главное слово здесь — платформа. Оно и характеризует весь Node.js. Из всего вышесказанного можно сделать предварительный вывод, что Node.js это среда выполнения JavaScript, точно как браузер, с той лишь разницей, что у нас нет доступа к DOM (а собственно, зачем он нужен на стороне сервера?; однако существует библиотека для работы с DOM — jsdom).

5.4 Express

Express.js, или просто Express, каркас web-приложений для Node.js, реализованный как свободное и открытое программное обеспечение под лицензией MIT. Он спроектирован для создания веб-приложений и API[3]. Де-факто является стандартным каркасом для Node.js. Автор фреймворка, TJ Holowaychuk, описывает его как созданный на основе написанного на языке Ruby каркаса Sinatra, подразумевая, что он минималистичен и включает большое число подключаемых плагинов. Express может являться backend'ом для программного стека MEAN, вместе с базой данных MongoDB и каркасом AngularJS для frontend'а.

5.5 D3

D3.js (или просто D3) — это JavaScript-библиотека для обработки и визуализации данных с невероятно огромными возможностями. Я, когда впервые узнал про нее, наверное, потратил не менее двух часов, просто просматривая примеры визуализации данных, созданных на D3. И конечно, когда мне самому понадобилось строить графики для небольшого внутреннего сайта на нашем предприятии, первым делом вспомнил про D3 и с мыслью, что "сейчас я всех удивлю крутейшей визуализацией", взялся изучать исходники примеров. . .

. . . и понял, что сам абсолютно ничего не понимаю! Странная логика работы библиотеки, в примерах целая куча строк кода, чтобы создать простейший график — это был конечно же удар, главным образом по самолюбию. Ладно, утер сопли — понял, что с наскоку D3 не взять и для понимания этой библиотеки надо начинать с самых ее основ. Потому решил пойти другим путем — взять за основу для своих графиков одну из библиотек — надстроек на D3. Как выяснилось, библиотек таких довольно много — значит не один я такой, непонимающий (говорило мое поднимающееся из пепла самолюбие).

Попробовав несколько библиотек, остановился на dimple как на более или менее подходящей для моих нужд, отстроил с ее помощью все свои графики, но неудовлетворенность осталась. Некоторые вещи работали не так, как хотелось бы, другой функционал без глубокого копания в dimple не удалось реализовать, и он был отложен. Да и вообще, если нужно глубоко копать, то лучше это делать напрямую с D3, а не с дополнительной настройкой, богатый функционал которой в моем случае используется не более, чем на пять-десять процентов, а вот нужных мне настроек наоборот не хватало. И поэтому случилось то, что случилось — D3.js.

5.6 MongoDB

MongoDB реализует новый подход к построению баз данных, где нет таблиц, схем, запросов SQL, внешних ключей и многих других вещей, которые присущи объектно-реляционным базам данных.

Со времен динозавров было обычным делом хранить все данные в реляционных базах данных (MS SQL, MySQL, Oracle, PostgresSQL). При этом было не столь важно, а подходят ли реляционные базы данных для хранения данного типа данных или нет.

В отличие от реляционных баз данных MongoDB предлагает документо-ориентированную модель данных, благодаря чему MongoDB работает быстрее, обладает лучшей масштабируемостью, ее легче использовать.

Но, даже учитывая все недостатки традиционных баз данных и достоинства MongoDB, важно понимать, что задачи бывают разные и методы их решения бывают разные. В какой-то ситуации MongoDB действительно улучшит производительность вашего приложения, например, если надо хранить сложные по структуре данные. В другой же ситуации лучше будет использовать традиционные реляционные базы данных. Кроме того, можно использовать смешенный подход: хранить один тип данных в MongoDB, а другой тип данных - в традиционных БД.

Вся система MongoDB может представлять не только одну базу данных, находящуюся на одном физическом сервере. Функциональность MongoDB позволяет расположить несколько баз данных на нескольких физических серверах, и эти базы данных смогут легко обмениваться данными и сохранять целостность.

6 ОБЗОР АРХИТЕКТУРЫ СИСТЕМЫ

В этом разделе детально рассмотрим архитекруту системы. Это важный этам процесса разработки приложения. В этом этапе принимают участия самые проффесиональные разработчики. Т.к. этот этап будет влиять на весь дальнейший процесс разработки и ошибки,сделанные на этом этапе буду стоить очень дорого.

6.1 Клиент-серверная архетектура

Для современного веб-приложения характерна клиент-серверная архетектура. Архитектура клиент-сервер определяет лишь общие принципы взаимодействия между компьютерами, детали взаимодействия определяют различные протоколы. Данная концепция нам говорит, что нужно разделять машины в сети на клиентские, которым всегда что-то надо и на серверные, которые дают то, что надо. При этом взаимодействие всегда начинает клиент, а правила, по которым происходит взаимодействие описывает протокол.

Существует два вида архитектуры взаимодействия клиент-сервер: первый получил название двухзвенная архитектура клиент-серверного взаимодействия, второй – многоуровневая архитектура клиент-сервер (иногда его называют трехуровневая архитектура или трехзвенная архитектура, но это частный случай).

Принцип работы двухуровневой архитектуры взаимодействия клиент-сервер заключается в том, что обработка запроса происходит на одной машине без использования сторонних ресурсов. Двухзвенная архитектура предъявляет жесткие требования к производительности сервера, но в тоже время является очень надежной (рисунок 6.1).

Здесь четко видно, что есть клиент (1-ый уровень), который позволяет человеку сделать запрос, и есть сервер, который обрабатывает запрос клиента.

Если говорить про многоуровневую архитектуру взаимодействия клиент-сервер, то в качестве примера можно привести любую современную СУБД (за исключением библиотеки SQLite, которая в принципе не использует концепцию клиент-сервер). Суть многоуровневой архитектуры заключается в том, что запрос клиента обрабатывается сразу несколькими серверами. Такой подход позволяет значительно снизить нагрузку на сервер из-за того, что происходит распределение операций, но в то же самое время данный подход не такой надежный, как двухзвенная архитектура (рисунок 6.2).

Типичный пример трехуровневой модели клиент-сервер. Если говорить в контексте систем управления базами данных, то первый уровень –

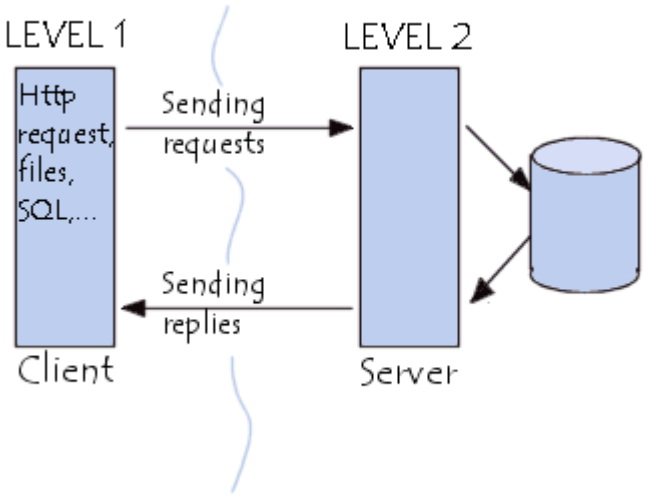

Рисунок 6.1 – Схема

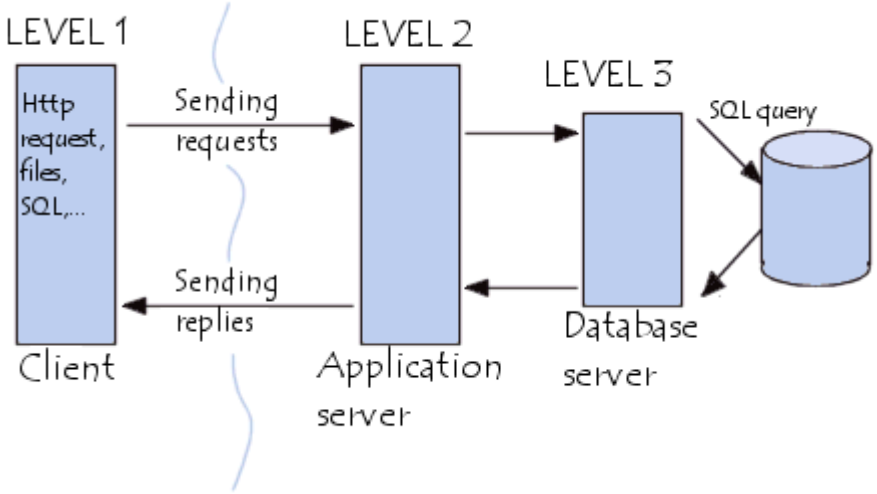

Рисунок 6.2 – Схема

это клиент, который позволяет нам писать различные SQL запросы к базе данных. Второй уровень – это движок СУБД, который интерпретирует запросы и реализует взаимодействие между клиентом и файловой системой, а третий уровень – это хранилище данных.

Если мы посмотрим на данную архитектуру с позиции сайта. То первый уровень можно считать браузером, с помощью которого посетитель заходит на сайт, второй уровень – это связка Nginx + Node, а третий уровень – это база данных.

Главным преимуществом модели клиент-сервер является то, что код клиента и сервера разделен. Все высоконагруженные вычисления происходят на мощный серверах, а клиентская машина, к которым предъявляются

менее серьезные требования, не несет большой нагрузки.

Учьтя всё вышесказанное можно смоделировать клиент-серверную структуру для нашего приложения. Она будут представлена трехуровневой системой. В добавок к ней главный сервер веб-приложения будет связан не только с сервером базы данных, но и с серверами пользователей, которые предоставляют api с данными для своих систем воддельности.

6.2 Проектирование базы данных

Данная глава содержит обзор всех сущностей. В проекте используется нерелиацинная база дынных mongoDB с открытым исходным кодом. Рисунок представляет сущности с полями.

В данном приложении сущности будут храниться для предоставления статистики, которая изменяется со временем, а так же для управления юзерами.

6.3 UML-Диаграммы прецедентов

UML (англ. Unified Modeling Language – унифицированный язык моделирования) – язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур [1].

UML является языком широкого профиля, это – открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможная генерация кода [1].

Использование UML позволяет также разработчикам программного обеспечения достигнуть соглашения в графических обозначениях для представления общих понятий, таких как класс, компонент, обобщение, агрегация и поведение, и больше сконцентрироваться на проектировании и архитектуре.

Прецеденты – это технология определения функциональных требований к системе. Работа прецедентов заключается в описании типичных взаимодействий между пользователями системы и самой системой. В терминах прецедента пользователи называются актерами. Актер (actor) представляет собой некую роль, которую пользователь играет по отношению к системе. Актерами могут быть пользователь, торговый представитель пользователя,

менеджер по продажам и товаровед и т.д. Актеры действуют в рамках прецедентов [1].

7 РЕАЛИЗАЦИЯ И ИСПОЛЬЗОВАНИЕ

8 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ВНЕДРЕНИЯ ПРОГРАММНОГО СРЕДСТВА

8.1 Введение и исходные данные

Разработка программного продукта “Сервис для управления серверами” позволит создать приложение, которое позволит управлять конфигурациями серверов, смотреть статистику их работы.

В текущий момент, для того чтобы проделать всё это, необходимо самостоятельно собирать данные в хранилища, писать процедуры и функции для извлечения и обработки данных и удобного вывода, необходимо переключаться между конфигами вручную и менять их. Все это занимает достаточно много времени. И не только при реализации в первый раз, но и последующие.

Применение программного продукта “Сервис управления серверами” позволит значительно улучшить взаимодействие системного администратора с серверами, уменьшения времени на мониторинг информации и изменения конфигураций сервера, а также реализации удобного пользовательского интерфейса на странице.

Целью технико-экономического обоснования программного средства является определение экономической выгодности создания данного программного продукта и дальнейшего применения.

8.2 Расчет сметы затрат и цены программного продукта

Программный комплекс относится к 1-й группе сложности. Т.к. позволяет взаимодействовать с удаленными устройствами в реальном времени. Категория новизны – “В”. Для оценки экономической эффективности разработанного программного средства проводится расчет прибыли от разработки одной системы. Расчеты выполнены на основе методического пособия [1].

Коэффициент сложности, который учитывает дополнительные затраты труда, связанные с обеспечением интерактивного доступа и хранения, и поиска данных в сложных структурах [2, приложение 4, таблица П.4.2]

$$К_с = 1 + \sum_{i=1}^{n} К_i = 1 + 0{,}06 + 0{,}07 = 1{,}18, \tag{8.1}$$

где $К_i$ — коэффициент, соответствующий степени повышения сложности за счет конкретной характеристики;
n — количество учитываемых характеристик.

Коэффициент $К_т$, учитывающий степень использования при разработке стандартных модулей, для разрабатываемого приложения, в котором степень охвата планируется на уровне около 50%, примем равным $0{,}7$ [2, приложение 4, таблица П.4.5].

Коэффициент новизны разрабатываемого программного средства $К_н$ примем равным $0{,}9$, так как разрабатываемом программное средство принадлежит определенному параметрическому ряду существующих программных средств [2, приложение 4, таблица П.4.4].

Исходя из выбранных коэффициентов, общая трудоемкость разработки $Т_о = Т_н \cdot К_с \cdot К_т \cdot К_н = 520 \cdot 1{,}18 \cdot 0{,}7 \cdot 0{,}9 = 387$ чел./д.

Для расчета срока разработки проекта примем число разработчиков $Ч_р = 3$. Исходя из комментария к постановлению Министерства труда и социальной защиты Республики Беларусь от 05.10.16 №54 «Об установлении расчетной нормы рабочего времени на 2017 год» [3], эффективный фонд времени работы одного человека составит

$$Ф_{эф} = Д_г - Д_п - Д_в - Д_о = 365 - 9 - 103 - 21 = 232 \text{ д.}, \qquad (8.2)$$

где $Д_г$ — количество дней в году;
$Д_п$ — количество праздничных дней в году;
$Д_в$ — количество выходных дней в году;
$Д_о$ — количество дней отпуска.

Тогда трудоемкость разработки проекта

$$Т_р = \frac{Т_о}{Ч_р \cdot Ф_{эф}} = \frac{387}{3 \cdot 232} = 0{,}56 \text{ г.} = 203 \text{ д.} \qquad (8.3)$$

Исходя из того, что разработкой будет заниматься 3 человека, можно запланировать фонд рабочего времени для каждого исполнителя

$$Ф_{пi} = \frac{Т_р}{Ч_р} = \frac{203}{3} \approx 68 \text{ д.} \qquad (8.4)$$

8.3 Расчет сметы затрат

Основной статьей расходов на создание ПО является заработная плата разработчиков проекта. Информация об исполнителях перечислена в таблице 8.3. Кроме того, в таблице приведены данные об их тарифных разрядах,

приведены разрядные коэффициенты, а также по формулам 8.5 и 8.6 рассчитаны месячный и часовой оклады.

$$\text{Т}_\text{м} = \text{Т}_\text{м}^1 \cdot \text{Т}_\text{к}, \tag{8.5}$$

$$\text{Т}_\text{ч} = \frac{\text{Т}_\text{м}}{\Phi_\text{р}}, \tag{8.6}$$

где $\text{Т}_\text{м}$ — месячный оклад;
$\text{Т}_\text{м}^1$ — тарифная ставка 1-го разряда (положим ее равной 265 руб.);
$\text{Т}_\text{к}$ — тарифный коэффициент;
$\text{Т}_\text{ч}$ — часовой оклад;
$\Phi_\text{р}$ — среднемесячная норма рабочего времени (в 2017 г. составляет $168{,}3$ ч. [3]).

Тогда основная заработная плата исполнителей составит

$$\begin{aligned} \text{З}_\text{о} &= \sum_{i=1}^{n} \text{Т}_\text{чi} \cdot \text{Т}_\text{ч} \cdot \Phi_\text{пi} \cdot K = \\ &= (6{,}27 + 5{,}48 + 4{,}17) \cdot 8 \cdot 68 \cdot 1{,}3 = 11\,258{,}62 \text{ руб.}, \end{aligned} \tag{8.7}$$

где $\text{Т}_\text{чi}$ — часовая тарифная ставка i-го исполнителя, руб.;
$\text{Т}_\text{ч}$ — количество часов работы в день;
$\Phi_\text{пi}$ — плановый фонд рабочего времени i-го исполнителя, д.;
K — коэффициент премирования (принятый равным $1{,}3$).

Дополнительная заработная плата включает выплаты, предусмотренные законодательство о труде: оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей, и определяется по нормативу, установленному в организации, в процентах к основной заработной плате. Приняв данный норматив $\text{Н}_\text{д} = 20\%$, рассчитаем дополнительные выплаты

$$\text{З}_\text{д} = \frac{\text{З}_\text{о} \cdot \text{Н}_\text{д}}{100\%} = \frac{11\,258{,}62 \cdot 20\%}{100\%} = 2251{,}72 \text{ руб.} \tag{8.8}$$

Отчисления в фонд социальной защиты населения и в фонд обязательного страхования определяются в соответствии с действующим законодательством по нормативу в процентном отношении к фонду основной и

дополнительной зарплат по следующим формулам

$$\begin{aligned} З_{сз} &= \frac{(З_о + З_д) \cdot Н_{сз}}{100\%}, \\ З_{ос} &= \frac{(З_о + З_д) \cdot Н_{ос}}{100\%}. \end{aligned} \tag{8.9}$$

В настоящее время нормы отчислений в ФСЗН $Н_{сз} = 34\%$ и в фонд обязательного страхования $Н_{ос} = 0{,}6\%$. Исходя из этого, размеры отчислений

$$\begin{aligned} З_{сз} &= \frac{(11\,258{,}62 + 2251{,}72) \cdot 34\%}{100\%} = 4593{,}52 \text{ руб.}, \\ З_{ос} &= \frac{(11\,258{,}62 + 2251{,}72) \cdot 0{,}6\%}{100\%} = 81{,}06 \text{ руб.} \end{aligned} \tag{8.10}$$

Расходы по статье «Материалы» отражают траты на магнитные носители, бумагу, красящие материалы, необходимые для разработки ПО определяются по нормативу к фонду основной заработной платы разработчиков. Исходя из принятого норматива $Н_{мз} = 5\%$ определим величину расходов

$$М = \frac{З_о \cdot Н_{мз}}{100\%} = \frac{11\,258{,}62 \cdot 5\%}{100\%} = 562{,}93 \text{ руб.} \tag{8.11}$$

Расходы по статье «Машинное время» включают оплату машинного времени, необходимого для разработки и отладки ПО, которое определяется по нормативам на 100 строк исходного кода. Норматив зависит от характера решаемых задач и типа ПК; для текущего проекта примем $Н_{мв} = 15\%$ [2, приложение 6]. Примем величину стоимости машино-часа $Ц_м = 0{,}8$ руб. Тогда, применяя понижающий коэффициент $0{,}5$, получим величину расходов

$$Р_м = Ц_м \cdot \frac{V_о}{100} \cdot Н_{мв} = 0{,}8 \cdot \frac{28\,940}{100} \cdot 15\% \cdot 0{,}5 = 1736{,}40 \text{ руб.} \tag{8.12}$$

Расходы по статье «Научные командировки» определяются по нормативу в процентах к основной заработной плате. Принимая норматив равным $Н_{рнк} = 15\%$ получим величину расходов

$$Р_{нк} = \frac{З_о \cdot Н_{рнк}}{100\%} = \frac{11\,258{,}62 \cdot 15\%}{100\%} = 1688{,}79 \text{ руб.} \tag{8.13}$$

Расходы по статье «Прочие затраты» включают затраты на приобретение и подготовку специальной научно-технической информации и специ-

альной литературы. Определяются по нормативу в процентах к основной заработной плате. Принимая норматив равным $Н_{пз} = 20\%$ получим величину расходов

$$П_з = \frac{З_о \cdot Н_{пз}}{100\%} = \frac{11\,258{,}62 \cdot 20\%}{100\%} = 2251{,}72 \text{ руб.} \tag{8.14}$$

Затраты по статье «Накладные расходы», связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды, относятся к конкретному ПО по нормативу в процентном отношении к основной заработной плате исполнителей. Принимая норматив равным $Н_{нр} = 50\%$ получим величину расходов

$$Р_н = \frac{З_о \cdot Н_{нр}}{100\%} = \frac{11\,258{,}62 \cdot 50\%}{100\%} = 5629{,}31 \text{ руб.} \tag{8.15}$$

Общая сумма расходов по смете определяется как сумма вышерассчитанных показателей

$$\begin{gathered} С_п = З_о + З_д + З_{сз} + З_{ос} + М + Р_м + Р_{нк} + П_з + Р_н = \\ = 30\,054{,}07 \text{ руб.} \end{gathered} \tag{8.16}$$

Рентабельность определяется из результатов анализа рыночных условий и переговоров с потребителями ПО. Исходя из принятого уровня рентабельности $У_{рп} = 10\%$, прибыль от реализации ПО составит

$$П_о = \frac{С_п \cdot У_{рп}}{100\%} = \frac{30\,054{,}07 \cdot 10\%}{100\%} = 3005{,}41 \text{ руб.} \tag{8.17}$$

На основании расчета прибыли и уровня себестоимости рассчитаем прогнозируемую цену программного средства без учета налогов

$$Ц_п = С_п + П_о = 30\,054{,}07 + 3005{,}41 = 33\,059{,}48 \text{ руб.} \tag{8.18}$$

Далее рассчитаем налог на добавленную стоимость

$$НДС = \frac{Ц_п \cdot Н_{ДС}}{100\%} = \frac{33\,059{,}48 \cdot 20\%}{100\%} = 6611{,}90 \text{ руб.} \tag{8.19}$$

НДС включается в прогнозируемую отпускную цену

$$Ц_о = Ц_п + НДС = 33\,059{,}48 + 6611{,}90 = 39\,671{,}38 \text{ руб.} \tag{8.20}$$

Организация-разработчик участвует в освоении и внедрении ПО и

несет соответствующие затраты, которые определяются по нормативу $Н_о = 10\%$ от себестоимости ПО в расчете на три месяца

$$Р_о = \frac{С_п \cdot Н_о}{100\%} = \frac{30\,054{,}07 \cdot 10\%}{100\%} = 3005{,}41 \text{ руб.} \quad (8.21)$$

Кроме того, организация-разработчик осуществляет сопровождение ПО, которое также оплачивается заказчиком. Расчет осуществляется в соответствии с нормативом $Н_с = 20\%$ от себестоимости ПО

$$Р_с = \frac{С_п \cdot Н_с}{100\%} = \frac{30\,054{,}07 \cdot 20\%}{100\%} = 6010{,}81 \text{ руб.} \quad (8.22)$$

Экономическим эффектом разработчика будет являться сумма прибыли с вычетом налога на прибыль

$$П_ч = П_о - \frac{П_о \cdot Н_п}{100\%} = 3005{,}41 - \frac{3005{,}41 \cdot 18\%}{100\%} = 2464{,}44 \text{ руб.} \quad (8.23)$$

8.4 Оценка экономической эффективности применения ПС у пользователя

В результате применения нового ПО пользователь может понести значительные капитальные затраты на приобретение и освоение ПО, доукомплектованием ЭВМ новыми техническими средствами и пополнение оборотных средств. Однако, если приобретенное ПО будет в достаточной степени эффективнее базового, то дополнительные капитальные затраты быстро окупятся.

Для определения экономического эффекта от использования нового ПО у потребителя необходимо сравнить расходы по всем основным статьям сметы затрат на эксплуатацию нового ПО с расходами по соответствующим статьям базового варианта. При этом за базовый вариант примем ручной вариант. Исходные данные для расчета приведены в таблице 8.4.

Общие капитальные вложения заказчика (потребителя) вычисляются следующим образом

$$\begin{aligned} К_о = К_{пр} + К_{ос} + К_с = 39\,671{,}38 + 3005{,}41 + 6010{,}81 = \\ = 48\,687{,}6 \text{ руб.} \end{aligned} \quad (8.24)$$

[1)] На февраль 2017 г. [4]

где $К_{пр}$ — затраты пользователя на приобретение ПО по отпускной цене;
$К_{ос}$ — затраты пользователя на освоение;
$К_{с}$ — затраты пользователя на оплату услуг по сопровождению.

В качестве типичного примера использования разрабатываемого ПС предполагается сценарий ее использования ежедневно 4 раза в день, при этом предполагается снижение трудоемкости с $Т_{с1} = 2$ чел./ч. до $Т_{с2} = 0{,}1$ чел./ч.. Приняв среднемесячную заработную плату работника $З_{см} = 716{,}5$ руб., рассчитаем экономию затрат на заработную плату в расчете на одну задачу $С_{зе}$ и за год $С_{з}$

$$С_{зе} = \frac{З_{см} \cdot (Т_{с1} - Т_{с2})}{Ф_{р}} = \frac{716{,}5 \cdot (2 - 0{,}1)}{168{,}3} = 8{,}09 \text{ руб.}, \quad (8.25)$$

$$С_{з} = С_{зе} \cdot А_{2} = 8{,}09 \cdot 1460 = 11\,811{,}4 \text{ руб.}, \quad (8.26)$$

где $Ф_{р}$ — среднемесячная норма рабочего времени, ч.

Экономия с учетом начислений на зарплату

$$С_{н} = С_{з} \cdot \frac{100\% + K}{100\%} = 11\,811{,}4 \cdot \frac{100\% + 1{,}3}{100\%} = 11\,964{,}95 \text{ руб.}, \quad (8.27)$$

где K — норматив начислений на зарплату, руб.

Экономия за счет сокращения простоев сервиса

$$С_{с} = \frac{(П_{1} - П_{2}) \cdot Д_{рг} \cdot С_{п}}{60} = \frac{(50 - 10) \cdot 300 \cdot 79{,}8}{60} = 15\,960{,}00, \text{ руб.}, \quad (8.28)$$

где $Д_{рг}$ — плановый фонд работы сервиса, д.

Тогда общая годовая экономия текущих затрат, связанных с использованием нового ПО

$$С_{о} = С_{н} + С_{с} = 11\,964{,}95 + 15\,960{,}00 = 27\,924{,}95 \text{ руб.} \quad (8.29)$$

Внедрение нового ПО позволит пользователю сэкономить на текущих затратах, то есть получить на эту сумму дополнительную прибыль. Для пользователя в качестве экономического эффекта выступает лишь чистая прибыль. Принимая размер ставки налога на прибыль $Н_{п} = 18\%$ получим

$$\Delta П_{ч} = С_{о} - \frac{С_{о} \cdot Н_{п}}{100\%} = 27\,924{,}95 - \frac{27\,924{,}95 \cdot 18\%}{100\%} = 22\,898{,}46 \text{ руб.} \quad (8.30)$$

В процессе использования нового ПО чистая прибыль в конечном итоге возмещает капитальные затраты. Однако полученные при этом суммы прибыли и затрат по годам приводят к единому времени – расчетному году (за расчетный год принят 2017-й год) путем умножения результатов и затрат за каждый год на коэффициент дисконтирования α. При расчете используются следующие коэффициенты: 2017 г. – 1, 2018 г. – $0{,}8696$, 2019 г. – $0{,}7561$, 2020 г. – $0{,}6575$. Все рассчитанные данные экономического эффекта сводятся в таблицу 8.5.

В результате технико-экономического обоснования применения программного средства были получены следующие значения показателей эффективности:

– чистая прибыль разработчика составит $\text{П}_\text{ч} = 2464{,}44$ руб.;

– затраты заказчика окупятся уже на четвертом году использования;

– экономическая эффективность для заказчика, выраженная в виде чистого дисконтированного дохода, составит $3594{,}17$ руб. за четыре года использования данного ПС; более высокий прирост прибыли заказчик получит по истечению данного срока.

Полученные результаты свидетельствуют об эффективности разработки и внедрения проектируемого программного средства.

Таблица 8.1 – Исходные данные

Наименование показателя	Буквенные обозначение	Единицы измерения	Количество
1	2	3	4
Группа сложности		единиц	1
Дополнительный коэффициент сложности	$\sum_{i=1}^{n} К_i$	единиц	0,18
Степень охвата функций стандартными модулями	$К_т$	0,7	
Коэффициент новизны	$К_н$	0,9	
Количество дней в году	$Д_г$	365	
Количество праздничных дней в году	$Д_п$	9	
Количество выходных дней в году	$Д_в$	103	
Количество дней отпуска	$Д_о$	21	
Количество разработчиков	$Ч_р$	3	
Тарифная ставка первого разряда, руб.	$Т_ч^1$	265	
Среднемесячная норма рабочего времени, ч.	$Ф_р$	168,3	
Продолжительность рабочей смены, ч.	$Т_ч$	8	
Коэффициент премирования	K	1,3	
Норматив дополнительной заработной платы	$Н_д$	20%	
Норматив отчислений в ФСЗН	$Н_{сз}$	34%	
Норматив отчислений по обязательному страхованию	$Н_{ос}$	0,6%	
Норматив расходов по статье «Материалы»	$Н_{мз}$	5%	
Норматив расходов по статье «Машинное время»	$Н_{мв}$	15%	
Понижающий коэффициент к статье «Машинное время»		0,5	
Стоимость машино-часа, руб.	$Ц_м$	0,8	
Норматив расходов по статье «Научные командировки»	$Н_{рнк}$	15%	
Норматив расходов по статье «Прочие затраты»	$Н_{пз}$	20%	
Норматив расходов по статье «Накладные расходы»	$Н_{нр}$	50%	
Уровень рентабельности	$У_{рп}$	10%	
Ставка НДС	$Н_{дс}$	20%	

Таблица 8.2 – Перечень и объём функций программного модуля

№ функции	Наименование (содержание)	Объём функции, LoC
101	Организация ввода информации	100
102	Контроль, предварительная обработка и ввод информации	500
109	Организация ввода/вывода информации в интерактивном режиме	190
111	Управление вводом/выводом	2600
204	Обработка наборов и записей базы данных	1900
207	Манипулирование данными	8000
208	Организация поиска и поиск в БД	7500
304	Обслуживание файлов	500
305	Обработка файлов	800
309	Формирование файла	1000
506	Обработка ошибочных и сбойных ситуаций	500
507	Обеспечение интерфейса между компонентами	750
601	Отладка прикладных программ в интерактивном режиме	4300
707	Графический вывод результатов	300
	Общий объем	28 940

Таблица 8.3 – Работники, занятые в проекте

Исполнители	Разряд	Тарифный коэффициент	Месячный оклад, руб.	Часовой оклад, руб.
Руководитель проекта	17	3,98	1054,70	6,27
Ведущий инженер-программист	15	3,48	922,20	5,48
Инженер-программист II категории	11	2,65	702,25	4,17

Таблица 8.4 – Исходные данные

Наименование показателя	Условное обозначение	Значение в базовом варианте	Значение в новом варианте
Затраты пользователя на приобретение ПО	$К_{пр}$	—	39 671,38 руб.
Затраты пользователя на освоение	$К_{ос}$	—	3005,41 руб.
Затраты пользователя на сопровождение	$К_{с}$	—	6010,81 руб.
Трудоемкость на задачу, чел./ч.	$Т_{с1}$,$Т_{с2}$	2	0,1
Средняя зарплата[1)], руб.	$З_{см}$	716,5	716,5
Количество выполняемых задач	$А_1$,$А_2$	1460	1460
Время простоя сервиса, мин. в день	$П_1$,$П_2$	50	10
Стоимость одного часа простоя, руб.	$С_{п}$	79,8	79,8

Таблица 8.5 – Расчет экономического эффекта от использования нового ПО

Наименование показателя	2017 г.	2018 г.	2019 г.	2020 г.
Результаты				
Коэффициент приведения	1	0,8696	0,7561	0,6575
Прирост прибыли за счет экономии затрат ($П_ч$), руб.	—	22 898,46	22 898,46	22 898,46
То же с учетом фактора времени, руб.	—	19 912,50	17 313,53	15 055,74
Затраты				
Приобретение ПО ($К_{пр}$), руб.	39 671,38	—	—	—
Освоение ПО ($К_{ос}$), руб.	3005,41	—	—	—
Сопровождение ПО ($К_с$), руб.	6010,81	—	—	—
Всего затрат ($К_о$), руб.	48 687,6	—	—	—
То же с учетом фактора времени, руб.	48 687,6	—	—	—
Экономический эффект				
Превышение результата над затратами, руб.	−48 687,6	19 912,5	17 313,53	15 055,74
То же с нарастающим итогом, руб.	−48 687,6	−28 775,1	−11 461,57	3594,17

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Википедия – UML диаграммы [Электронный ресурс]. — Электронные данные. — Режим доступа: https://ru.wikipedia.org/wiki.

[2] Палицын, В.А. Технико-экономическое обоснование дипломных проектов: Метод. пособие для студ. всех спец. БГУИР. В 4-х ч. Ч. 4: Проекты программного обеспечения / В.А. Палицын. — Мн : БГУИР, 2006. — 76 с.

[3] Пещенко, Е.А. Производственный календарь на 2017 год [Электронный ресурс]. — Электронные данные. — Режим доступа: http://www.mintrud.gov.by/system/extensions/spaw/uploads/files/Kommetarij-2017-RV.pdf. — Дата доступа: 06.04.17.

[4] Белстат. О начисленной средней заработной плате работников в феврале 2017 г. [Электронный ресурс]. — Электронные данные. — Режим доступа: http://www.belstat.gov.by/ofitsialnaya-statistika/solialnaya-sfera/trud/operativnaya-informatsiya_8/o-nachislennoi-srednei-zarabotnoi-plate-rabotnikov/o-nachislennoy-sredney-zarabotnoy-plate-rabotnikov-v-fevrale-2017-nbsp-g/. — Дата доступа: 07.04.17.

ПРИЛОЖЕНИЕ А
(обязательное)
Фрагменты исходного кода

```
<!-- index.html -->
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8" />
   <link rel="stylesheet" href="./node_modules/semantic-ui-css/semantic.min.
      css"/>
</head>
<body>
   <div id="root"></div>

<!-- Dependencies -->
   <script src="./node_modules/react/dist/react.js"></script>
   <script src="./node_modules/react-dom/dist/react-dom.js"></script>

   <!-- Main -->
   <script src="./dist/bundle.js"></script>
</body>
</html>
```