МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики физики и информационных технологий
(наименование института полностью)

Кафедра Прикладная математика и информатика
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки, специальности)

Бизнес-информатика
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка архитектуры информационных систем на основе технологий виртуализации

Студент Д.Ю. Стенин
(И.О. Фамилия) (личная подпись)

Руководитель Доктор физико-математических наук, Профессор, А.И. Сафронов
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

Аннотация

С. 104, рис. 15, табл. 24, лит. 20 источников

Информационная система, технологии виртуализации, база данных, uml, idef0, увеличение экономического эффекта, virtualbox, сетевое взаимодействие, архитектурное решение, оптимизация нагрузки на сервер.

Разработана архитектура информационной системы, с применением технологий виртуализации для ОАО «Инжиниринг-сервис». Дано описание ОАО «Инжиниринг-сервис», выполнен технико-экономический анализ, выявлены основные проблемы на основании анализа базового решения. Построена модель «как есть» и её декомпозиция с использованием методологий idef0.

Сформулированы цель и задачи архитектуры, выявлены подходящие технологии и требования к информационной системе. Произведен выбор и обоснование методов проектирования информационной системы, а также проектных решений по увеличению финансовой эффективности, сетевому взаимодействию, оптимизации нагрузки. Построена функциональная модель компании «как должно быть» и её декомпозиция, описано программное, компьютерное, сетевое и технологическое обеспечение. Построена модель сетевого взаимодействия, основанная на технологиях виртуализации и локальная модель задействованных технологий каждой виртуальной машины, описывается решение по организации защиты данных.

Оценена экономическая эффективность реализации проекта.

Работа находится на стадии внедрения (принято решение о внедрении).

Оглавление

Введение

Компании в разное время стремились уменьшить расходы на поддержание инфраструктуры предприятия и увеличить связанность и сплоченность коллектива, ставя перед собой целью достижение большей прибыли. В текущее время для этого используются разнообразные ИТ-технологии, которые применяются так, как этого требует заказчик. В один момент времени инженеры пришли к виртуализации – разделения ресурсов одной вычислительной машины, с целью достижения максимальной управляемости, расширяемости и администрированию систем.

На сегодняшний день уже не вызывает сомнения тот факт, что виртуализация является крайне актуальной и востребованной технологией. Применительно к настольным компьютерам и серверам, виртуализация — это создание на одном физическом сервере или компьютере нескольких «виртуальных» машин, на каждой из которых может быть установлена своя среда — операционная система, приложения, пользовательские настройки и т.п. При этом такие виртуальные машины (ВМ) оказываются абсолютно изолированными друг от друга и ведут себя, как отдельные физические компьютеры.

Объектом исследования является организация ОАО «Инжиниринг-сервис», а предметом исследования стало базовое решение, установленное на физическом сервере и монолитно-ориентированной архитектурой приложения.

Целью работы является разработка архитектуры приложения для уменьшения затрат и увеличения быстродействия системы, посредством применения технологий виртуализации. В рамках работы решаются следующие задачи:

– Анализ предприятия и его базового решения;

– Анализ ИТ-технологий, подходящих под требования и специфику задач заказчика;

– Разработка архитектуры на основе технологий виртуализации;

– Анализ работы проектной системы;

– Сравнение результатов базовой и проектной системы;

Основные решения, выносимые на защиту это:

– Диаграмма сравнения традиционной архитектуры и архитектуры виртуализации;

– Диаграмма сетевого взаимодействия разработанной системы;

– Диаграмма разработанных информационных систем;

– Результаты сравнения затрат к прибыли для базовой и проектной системы;

В работе использованы методики анализа структурным и объектно-ориентированным методами.

В первой главе работы рассматриваются технологии базового решения, разрабатывается гипотеза, благодаря которой можно увеличить эффективность архитектуры. Анализируются схожие текущие разработки рынка и приводятся таблицы их сравнения. Ставится задача на разработку проектного решения и производится анализ бизнес-процессов «Как есть», «Как должно быть». Описываются выводы по первой главе.

Вторая глава работы посвящена построению логической архитектуры системы, в ней анализируется информационное обеспечение, проектируется логическая модель базы данных, собираются требования к аппаратно-программному обеспечению.

Третья глава посвящена физической разработке архитектуры приложения, в ней описывается функциональность проектной системы, производится анализ инструментов виртуализации и подбирается необходимый. Производится разработка архитектуры и локального приложения. Финальным этапом является анализ и обоснование экономической эффективности проектного варианта ИС.

Глава 1 Функциональное моделирование предметной области

1.1 Технико-экономическая характеристика предметной области

Объектом исследования является компания ОАО «Инжиниринг-сервис». Её цель, это разработка программного обеспечения. Организационной формой предприятия является «Открытое Акционерное Общество». Это означает, что уставной капитал компании разделен на акции, которые принадлежат совладельцам. В компании есть директор, который обладает контрольным пакетом акций и является прямым руководителем нижестоящих отделов. В соответствии с рисунком 1 Общая структура предприятия разделяется на несколько функциональных ролей, отделов:

- Технический менеджер,
- Отдел маркетинга,
- Отдел бизнес-анализа,
- Отдел разработки,
- Отдел тестирования.

У всех отделов есть непосредственные руководители, которыми являются директора по маркетингу, производству и финансам.

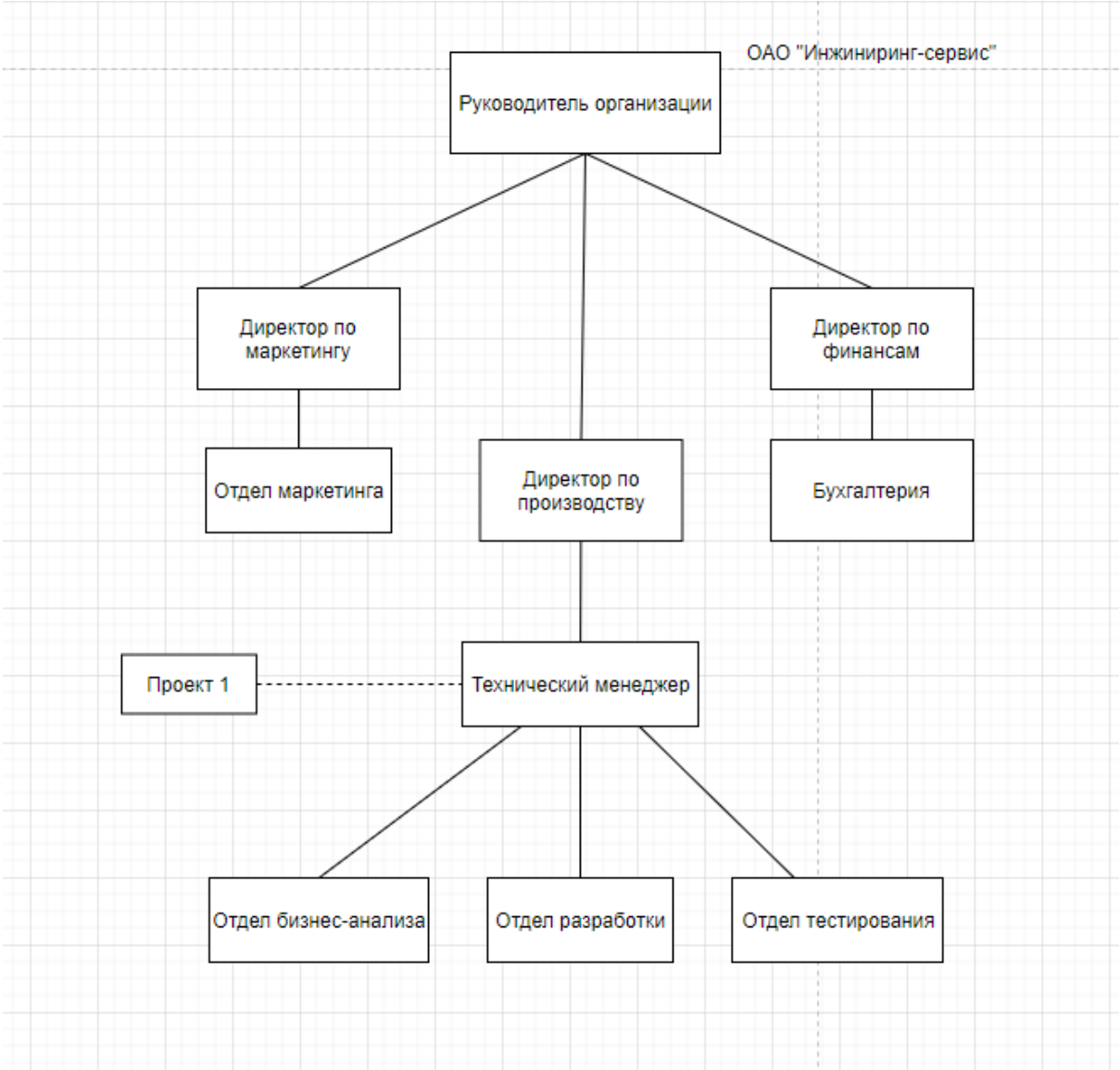


Рисунок 1 – Схема организационной структуры предприятия

Организационная схема является линейной, это означает что распоряжение и контроль за нижестоящими органами происходит сверху вниз. Минусом такой схемы является медленное принятие решений, так как решение зависит от вышестоящего начальника.

1.2 Концептуальное моделирование предметной области

Основной процесс исследуемой организации называется: «Деятельность организации по разработке ПО». Процесс разделяется на несколько модулей, к

каждому модулю привязан отдел, исполняющий конкретную задачу. Построим общую модель исследуемого бизнес-процесса «Как есть» (см. приложение А). В соответствии с рисунком можно рассмотреть общую модель процесса, входную/выходную информацию, стандарты и отделы, с помощью которых достигается конечный результат.

Модель можно декомпозировать на несколько основных этапов разработки любого программного обеспечения (см. приложение Б). Деятельность организации по разработке программного обеспечения выглядит так:

- Получение заказа на разработку;
- Разработка экономического плана;
- Разработка ПО;
- Внедрение информационной системы;

Данный уровень декомпозиции хорошо отображает деятельность всей организации, раскладывая задачи по разным отделам. Тем не менее, отделы могут участвовать в совместных процессах, например, отдел бизнес-анализа участвует в разработке экономического плана и в производстве ПО.

Этап получения заказа на разработку характеризуется связью с конечным клиентом. На данном этапе приходят первоначальные сведения о разрабатываемом продукте, происходит обсуждение требований с клиентом, осуществляется разработка и утверждается заказ на разработку. Результатом данного процесса будет - заказ на разработку. Этапом заведует директор по маркетингу и его исполнители. Этап расчета экономического плана характеризуется оценками затрат и расчета экономического плана проекта. Включает в себя следующие этапы:

- Анализ заказа на разработку;
- Оценка возможности реализации;
- Оценка затрат;
- Разработка экономического плана;

Результатом данных действий будет разработка экономического плана, для расчета затрат и экономической эффективности программного продукта. Ответственным является директор по финансам.

Самыми трудоемкими и интересными являются процессы производства ПО и его поддержка, они задействуют максимальное количество отделов и технической документации. Для получения информации о трудоёмких процессах организации и упрощения процесса реинжиниринга деятельности организации, разработаем декомпозицию производства. В соответствии с рисунком 2, приведена декомпозиция процесса «Производство ПО».

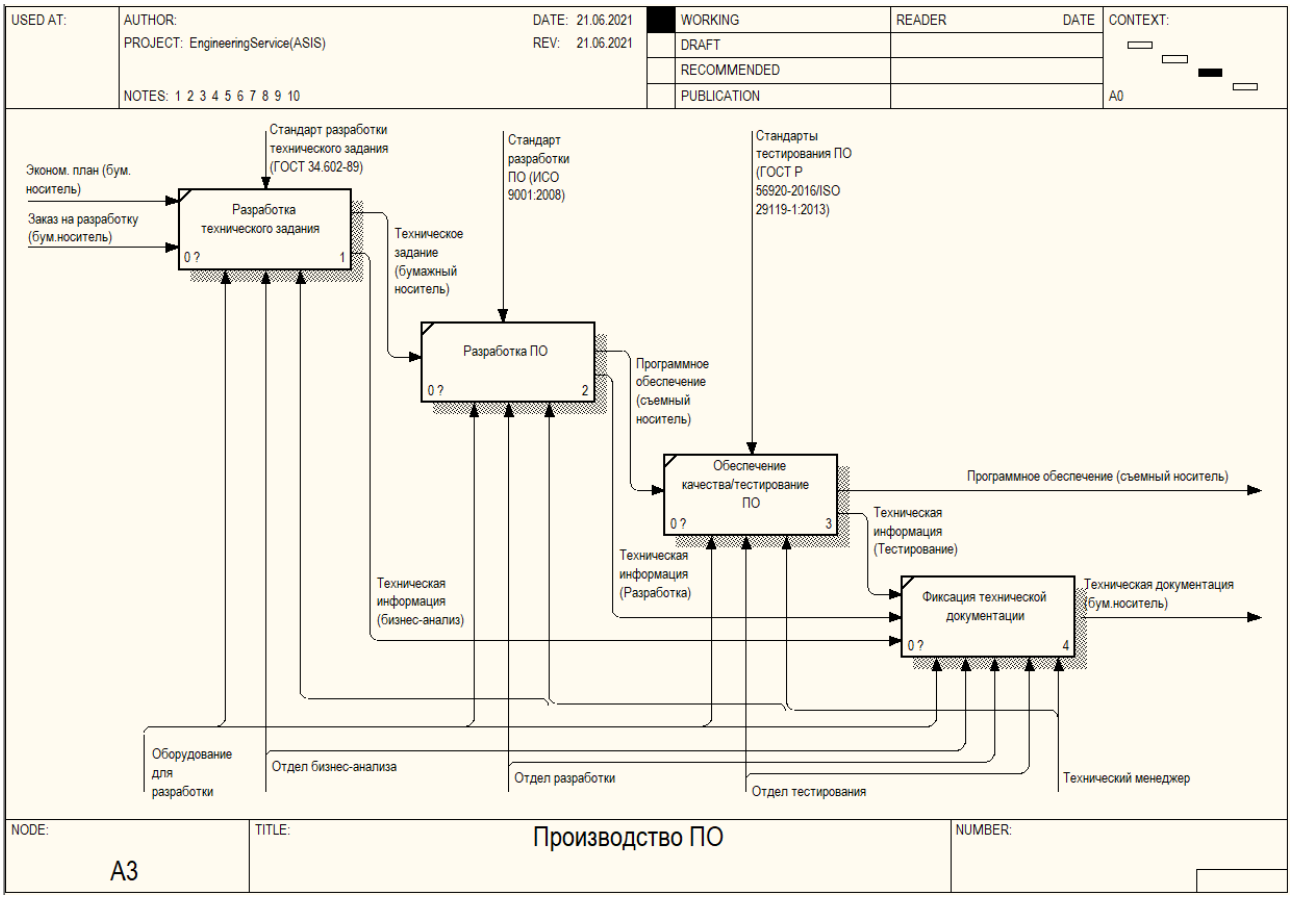


Рисунок 2 – Декомпозиция процесса «Производство ПО», «КАК ЕСТЬ»

Этап разработки характеризуется большими вложениями и выходным результатом, приближенным к финальному. В данном этапе участвует максимальное количество задач обработки информации. Обмен информацией

без информационной системы представляет из себя физическое перемещение носителя с данными. Для организации это дополнительные расходы и возможности для утечки информации, поэтому данному модулю очень необходима автоматизация. Разработка ПО делится на несколько этапов:

– Разработка технического задания;
– Разработка ПО;
– Обеспечение качества/тестирование ПО;
– Фиксация технической документации;

Ответственным является директор по производству. Результатом процесса является программный продукт. Внедрение программного продукта – это процесс передачи ПО заказчику, с соблюдением его требований и норм его организации. Данный этап характеризуется задействованием отдела разработки и отдела маркетинга и является заключительным процессом, результатом которого является прибыль и отчетность.

Следующим по очереди идет этап внедрения и поддержки. Важность данного процесса заключается в получении прибыли, характеризуется он плотной коммуникацией с заказчиком. На данном этапе необходимо привлечение финансовых сотрудников из организации разработчика и заказчика, для разрешения финансовых вопросов. В соответствии с рисунком 3 предоставлена декомпозиция процесса внедрения.

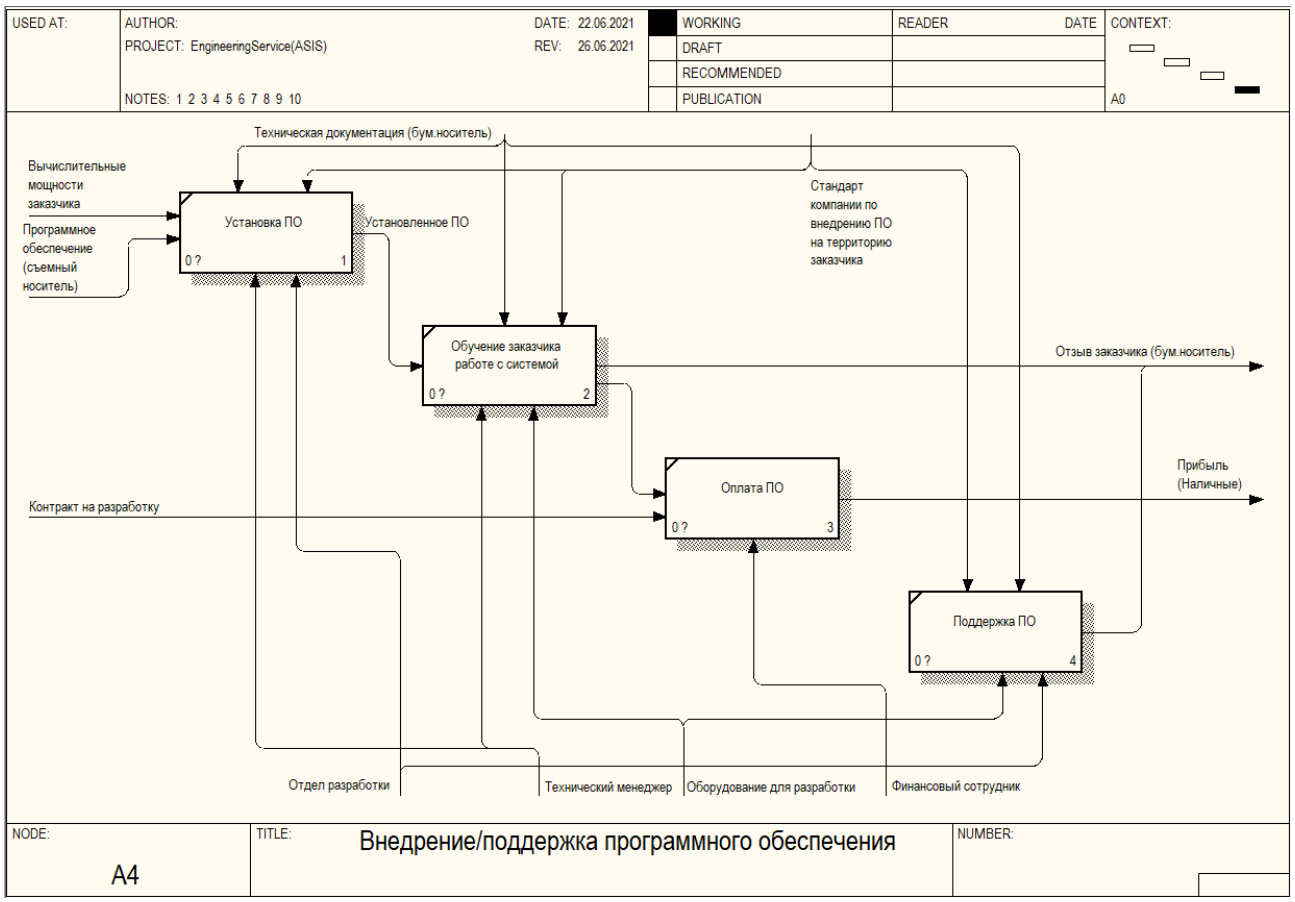


Рисунок 3 - Декомпозиция процесса «Внедрение/поддержка ПО», «КАК ЕСТЬ»

У предоставленного процесса и его декомпозиций существует несколько недостатков, которые характеризуются неправильной работой с информацией:

– Возможность утечки информации на фоне использования физических носителей;

– Сложность передачи информации бумажного носителя;

– Сложность поиска информации на бумажном носителе;

– Небезопасная передача прибыли наличными средствами;

– Возможность утери информации;

– Низкая производительность труда;

– Сложность анализа информации;

– Слабая коммуникация команды из-за отсутствия статуса разработки;

– Несовершенство организации сбора и регистрации исходной информации;

Из недостатков можно сформировать список улучшений, которые необходимы данной организации, которые смогут улучшить её процесс:

– Увеличение скорости обмена данными в организации;

– Увеличение скорости обмена данными с заказчиком;

– Устранение возможности утечки данных;

– Хранение информации в централизованной системе;

– Улучшение финансовых операций, отказ от наличных средств;

– Улучшение коммуникации между командой с помощью единой корпоративной системы;

– Увеличение качества собранной информацией и упрощение её регистрации;

Таким образом, существующей организации необходимо решение всех перечисленных недостатков и реинжиниринг основного процесса. Для того чтобы принять правильные решения по реинжинирингу необходимо проанализировать текущие разработки на рынке, на предмет соответствия требованиям.

1.3 Анализ существующих разработок на предмет соответствия сформулированным требованиям

Рынок корпоративных информационных систем довольно богат, он представляет собой различные системы приватные и публичные. У крупных IT компаний, таких как Microsoft имеется личная корпоративная система. Для данного типа информационных систем был введен отдельный термин “ERP”.

ERP – одна из разновидностей корпоративных информационных систем, которые предназначены для управления всеми ресурсами организации. Корпоративная система гарантирует контроль за административной и операционной частью деятельности компании. В них также входит функционал учета финансовой деятельности, планирования, управление материальными

потоками и т.д. Общий список возможностей таких систем очень велик и представляет из себя большое количество различных вариаций, типизированных по организационной направленности.

Самая популярная на данный момент система ERP – “SAP R/3”. “SAP R/3” была создана бывшими сотрудниками IBM[1], которые основали свою компанию под названием “SAP”.

SAP R/3 - ориентирована на крупные и средние предприятия, разрабатываемая и продаваемая компанией с начала 1990-х годов. Создана в продолжение линеек RF (позднее идентифицированной как R/1) и R/2. Начиная с выпусков середины 2000-х годов название R/3 не используется; ядро ERP-системы, созданной в продолжение линейки, производитель называет SAP ERP ECC.

Далее следует «Microsoft Dynamics», данная информационная система выросла из бухгалтерского программного обеспечения, рассчитанная в основном на средний бизнес с простой корпоративной структурой и производственной системой низкой и средней сложности. Данной системой предоставляются инструменты для управления поставками, закупками, управлением персоналом. Разделено на несколько различных продуктов.

Последним по счету, но не по значению является сервис “Oracle E-Business Suite”. Данный продукт, это целый комплекс программного обеспечения, разработанный для автоматизации финансов, управления персоналом и даже включающий в себя контроль за логистикой, маркетингом. Но при этом разработанный достаточно давно и имеющий прочную поддержку до 2031 года.

Все озвученные сервисы имеют в себе функционал типичный для большинства компаний. Но в случае с разрабатываемой системой, она является достаточно новой на рынке, написана на относительно современных

[1] IBM - американская компания со штаб-квартирой в Армонке (штат Нью-Йорк), один из крупнейших в мире производителей и поставщиков аппаратного и программного обеспечения, а также IT-сервисов и консалтинговых услуг.

технологиях, что увеличивает ее быстродействие по сравнению с конкурентами. Система от Oracle, является самой тяжеловесной системой, но тем не менее функционала по сохранению и управлению задачами в ней не имеется, соответственно, как и менеджера коммуникаций. Кроме-того все описанные системы спроектированы для получения прибыли, а соответственно ориентированы сразу на все направления фирм, но не содержат в себе ключевых инструментов для разработки ПО, как в данном случае. В соответствии с таблицей 1 отражены технические особенности каждой из систем. В ней содержатся все базовые требования к функциям. Благодаря данной таблице впоследствии, можно сделать вывод об предпочтительности функциональности на рынке, у данного типа систем.

Таблица 1 - Таблица сравнительного анализа аналогов АИС

Требование/Аналог	SAP R/3	Microsoft Dynamics	Oracle E-Business Suite
Коммуникационный сервис	-	-	-
Управление компанией/персоналом	+	+	+
Типизация задач, относительно специализации компании	частично	частично	частично
Менеджер задач	-	+	+
Поддержка/коммуникация	+	+	+
Поддержка тренинг-системы	+	+	-
Поддержка новостной системы	-	-	-
Итог	3	4	4

В данной таблице можно заметить, что, не смотря на то что основной функционал в системах присутствует, более специализированные задачи системам решать сложно, так как они предназначены для компаний общих направлений. С концептуальной точки зрения, проектируемая технология будет решать узкие, специализированные задачи специально для компаний, занимающихся разработкой программного обеспечения. Существующие системы, приведенные в пример, в основном используют монолитную архитектуру, так как разработаны системы довольно давно.

Разрабатываемая система должна поддерживать современные стандарты качества, использовать современные технологии передачи информации и при этом быть удобной в обслуживании и защищенной от внешних атак. Технологии виртуализации позволят защитить приложения и упростить их поддержку разделив их на контейнеры. Проект созданный с участием инструментов виртуализации легко поддается управлению и администрированию, кроме того его очень просто устанавливать на сервера заказчика, достаточно лишь команды на запуск конфигурационных файлов и приложение развернется на сервере. Так как анализ выявил необходимость создания специализированной системы ERP, отвечающий данным особенностям по развертке и установке необходимо описать задачу на разработку ИС.

1.4 Постановка задачи на разработку проекта создания/внедрения ИС

1.4.1 Цель и назначение автоматизированного варианта решения задачи

Целью разрабатываемой системы ERP, является автоматизация процесса разработки ПО у рассматриваемой организации. Необходимо перенести всю информацию с физических и бумажных носителей в информационную среду. Данную информацию очень удобно будет хранить в одном месте, это позволит выполнить поиск по всем данным и найти необходимые в кратчайшие сроки. Кроме этого, важно вести полноценную историю разработки проектов и

программной документации. Система должна учитывать базовый функционал систем ERP и привнести функционал учитывающий специфику компании.

Назначением реализации работы ERP на основе технологий виртуализации может служить:

– Автоматизация задач на производстве;
– Сохранение истории задач и проектов;
– Улучшение контроля за сотрудниками фирмы;
– Улучшение качества и скорости бухгалтерских расчетов;
– Упрощенное сохранение и передача знаний;

Следующим шагом в постановке задачи на разработку будет описание требований к функциональности АИС.

1.4.2 Требования к функциональности АИС

Так как информационная система оказывает тесное влияние на деятельность сотрудников организации, процессы приема, сбора, обработки информации переносятся на систему.

Работа подразделений остается не измененной, лишь с поправкой на то, что все процессы и полученная/отправленная информация перенесена на информационную систему. Источником поступления информации, являются сотрудники организации и заказчик. Вместо обработки одной части информации на бумажных носителях и в разделенном цифровом пространстве, предлагается поместить информацию в централизованное цифровое пространство и взаимодействовать с ней через систему.

ИС должна быть способна:

– Работать с информацией сохранять, удалять, редактировать;
– Поддерживать информацию о пользователях и управлять ими;
– Работать с заказчиком, предоставляя формы обратной связи и способность слежения за разработкой проекта;
– При работе с заказчиком важно разграничивать конфиденциальную информацию организации и заказчика;

– Предоставлять хранилище данных, поддерживать систему контроля версий;

– Быть доступной, внятной, простой для освоения;

– Сохранять техническую документацию;

На основе информации полученной из бизнес-процесса “AS-IS”, можно выявить этапы производства одного проекта по порядку:

– Получение информации о задаче от заказчика;

– Обработка информации и написание требований;

– Обработка требований отделом бизнес-анализа и формирование технического задания;

– Передача технического задания отделу разработки программного обеспечения;

– Разработка ПО;

– Написание тест-кейсов для сотрудников обеспечения качества;

– Передача тест-кейсов сотрудникам отдела по обеспечению качества;

– Тестирование системы;

– Передача информационной системы техническому менеджеру;

– Разработка плана внедрения и поддержки;

– Процесс внедрения и поддержки ПО;

После процесса формирования требований и выявления этапов производственного процесса можно перейти к вводимым формам. Вводимые формы – это способ введения в данных в информационную систему посредством форм для текста и символов. В разрабатываемой системе можно выявить необходимые формы:

– Процедура ввода первичной информации о задаче;

– Разработка технического задания;

– Создание задачи на разработку ПО;

– Создание задачи на тестирование продукта;

– Создание задачи на выдачу продукта;

На основании пунктов о вводимых формах можно сделать вывод, что вся информация, вводимая в систему, является оперативной информацией и лишь после завершения цикла разработки превращается в условно-постоянную. Информация является секретной и защищенной, поэтому не обойтись без механизмов «защиты информации».

Режимом решения задачи является «диалоговый», так как всю информацию предоставляют сотрудники организации. Периодичность решения задачи, связанной с бизнес-процессом постоянная, так как компания ориентирована на расширения, с большой долей вероятности это будут параллельные задачи. На данный момент требования предельно понятные, самое время сформулировать формализованную задачу к системе ИС.

1.4.3 Формализованная постановка задачи

Концептуально, описанный ранее бизнес-процесс разбивается на несколько логических модулей, каждый из которых отвечает за определенный тип задачи.

Особенно выделяются следующие модули:

– Модуль управления пользователями,
– Модуль документооборота,
– Тренинговая система,
– Модуль персональной информации.

Рассмотрим каждый модуль в отдельности и представим возможную концепцию модулей.

Модуль управления пользователями – это модуль, отвечающий за представление пользователя в системе и содержащий сведения о конкретном сотруднике. Соответственно он должен содержать механизмы авторизации пользователей, роли пользователей, механизм аутентификации, возможно более продвинутую систему генерации веб-токенов, для каждого пользователя и механизм шифрования/дешифровки и выдачи прав по данному токену.

Так как наше приложение построено с учетом Web архитектуры, то для данного случая подойдет спецификация «Json Web Token» (JWT). Стандарт «RFC 7519» создание/управление токенами доступа, основанный для формата Json. С помощью данных токенов пользователь, заходя в разные модули будет получать доступ либо отказ от доступа, основанный на правах пользователя. Например, в токен доступа тренера будут включены данные о том, что пользователь может не только просматривать и проходить тренинги, а также назначать тренинги и управлять оценками по завершению тренингов и так далее.

Модуль документации – Это модуль, отвечающий за документооборот предприятия. Предназначен для хранения и упрощенного процесса документооборота. Документооборотом называют управление документами, их подпись, доставку, утверждение и т.д. Документационный модуль содержит хранилище документов, которое использует СУБД для хранения документов. Связан с модулем пользователей, так как использует права пользователя для доступа к определенным типам документов, а также к менеджеру управления отпусками и отгулами, для синхронизации данных сотрудников и оповещение вышестоящего руководства.

Тренинговая система - подразумевает наличие СУБД для хранения тренингов и результатов тренингов для групп сотрудников. Имеет тесную связь с модулем пользователей, так как ей необходимо отслеживать доступ пользователей. А также возможно имеет связи с системой новостей, для оглашения результатов тестирования.

Менеджер сохранения личной информации - имеет тесную связь с СУБД для хранения не конфиденциальных данных, таких как описание предыдущего места работы, данные о навыках и занимаемой должности в компании. При этом личные данные, такие как логин, необходимо расшифровывать из токена, с помощью модуля управления пользователями.

Технологии виртуализации окажут непосредственную поддержку в использовании СУБД, так как для каждого сервиса будет использоваться одна физическая машина, то доступ до одной базы данных оправдан, особенно если

она не будет содержать большого количества обращений в секунду, в ином случае понадобится логическое разделение базы данных с помощью репликации. Масштабирование базы данных в этом случае является серьезной задачей, так как осуществляется работа с большой нагрузкой.

Репликация позволит создать полный логический дубликат базы данных и вместо обращения к одной базе данных, при превышение определенного порога нагрузки запросы будут поступать в копию, тем самым разгружая первую копию базы данных. Однако, при масштабировании следует учесть, что дубликат базы данных не должен иметь возможность записать данные, чтобы не допустить десинхронизации реплицированной базы данных. Данный подход подойдет в том случае, если нагрузка велика именно на аспект считывания данных, это очень эффективный способ масштабирования базы данных.

1.4.4 Требования к архитектуре и реализации АИС

Разработка АИС должна включать в себя несколько особенностей и архитектурных решений, одно из них, это полноценное использование технологий виртуализации. Для удобства использования системы с любого вычислительного устройства подходят веб-технологии, таким образом любой сотрудник компании подключенный к внутренней сети интернет и имеющий любой браузер может работать с системой и оказывать влияние на других пользователей системы. Механизм настройки системы доступен лишь ограниченному числу пользователей – администраторам. Основному же числу пользователей должны быть доступны лишь простые настройки, соответствующие занимаемой должностью и специфические настройки для используемого устройства.

АИС должна быть доступной максимальному количеству пользователей и достаточно специфичной по её гибкости, так как информационная система ориентирована на компании, которые занимаются разработкой программного обеспечения. Для получения наглядных результатов реинжиниринга необходимо разработать модель бизнес-процесса «КАК ДОЛЖНО БЫТЬ».

1.5 Разработка модели бизнес-процесса «КАК ДОЛЖНО БЫТЬ»

Основной целью данной работы является перенос информационных данных на вычислительную машину, для получения большей производительности труда и увеличения получаемых доходов, за счет ускорения разработки в целом. Для более показательной рассмотрим окончательную модель бизнес-процесса «Как должно быть» (см. приложение В). Данная разработана с использованием методологии IDEF0, данная методология хорошо раскрывает все проблемы и улучшения бизнеса. Предоставленные модели будет легко сравнивать если будет использоваться одна методология.

Требования заказчика к разрабатываемой системе поступают в организацию в цифровом виде. Выходная прибыль теперь достигается через цифровой перевод, а отзыв заказчик оставляет в специальной форме ввода, после передачи заказа.

Декомпозиция процесса (см. приложение Г) имеет существенные отличия от диаграммы процесса «Как есть». Разработанный заказ и экономический план теперь имеют информационный вид. Техническая документация и программное обеспечение выходят из одного блока и дополняют функции поддержки ПО на последнем этапе процесса.

Для сравнения двух моделей в соответствии с рисунком 4 приведена декомпозиция модуля – производство ПО. В указанном модуле появилось взаимодействие технологий версионного контроля. Это означает, что сотрудникам теперь нет нужды передавать данные через физический носитель, это можно будет осуществлять с помощью защищенной корпоративной сети.
Программное обеспечение после этапов разработки и тестирования попадает на машину заказчика через специалистов сторонней организации, теперь можно иметь уверенность о том, что на носителе не будет вредоносных объектов. Система позволит пересылать данные сразу после разработки и с помощью автоматических задач устанавливать их, обновляя поддерживаемое приложение.

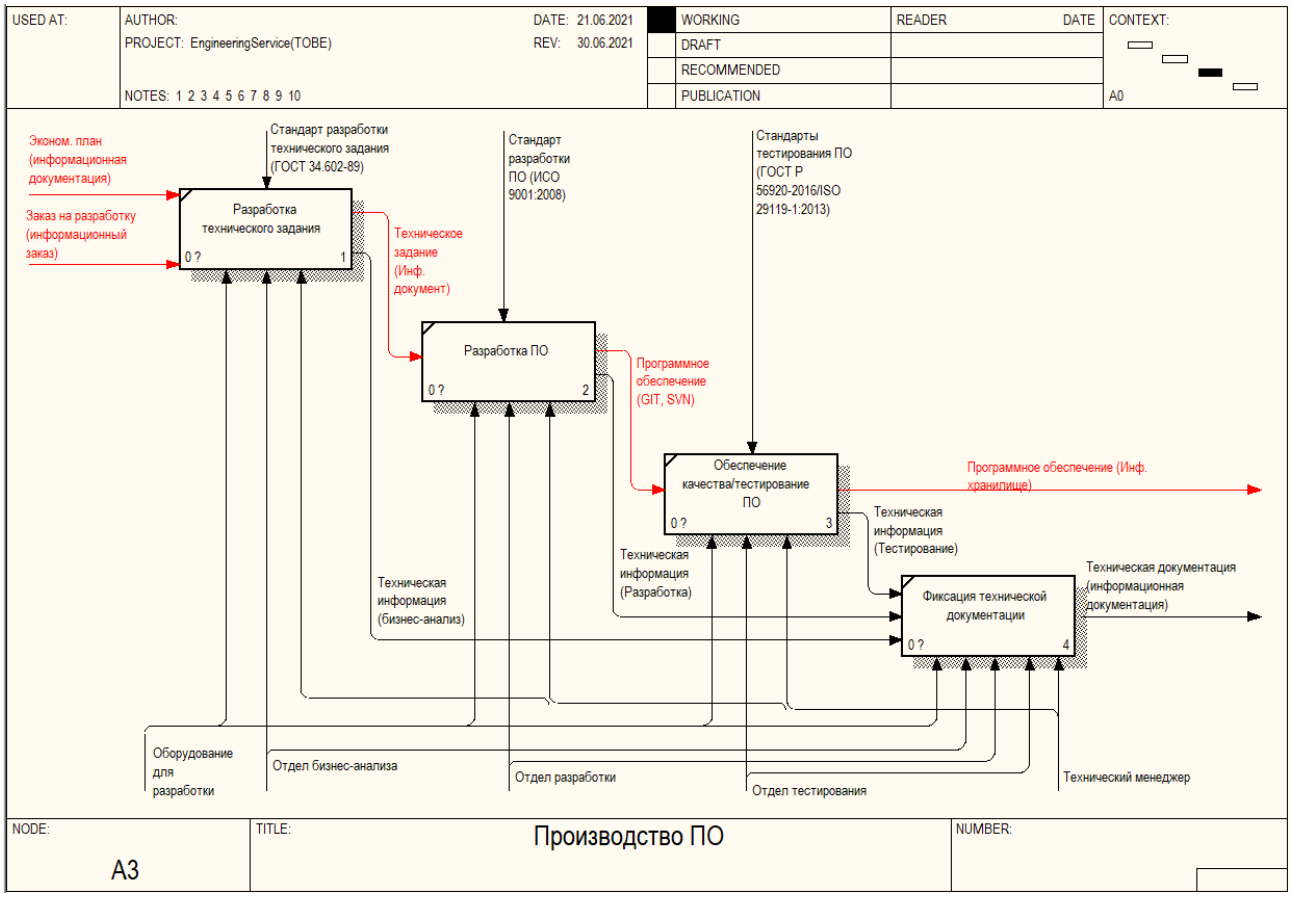


Рисунок 4 – Декомпозиция блока производство ПО, «КАК ДОЛЖНО БЫТЬ»

Декомпозиция блока введение/поддержка ПО предоставлена в соответствии с рисунком 5. Основными изменениями являются конвертация технической документации в информационный вид. Программное обеспечение, произведенное на предыдущем этапе, предоставляется из репозитория разработанной системы. Разработанная система поддерживает работу со средствами версионного контроля – GIT, SVN. Требования о них содержит раздел постановки задачи на разработку проекта. Очень удобно, что техническая документация теперь имеет информационный вид и поставляется объединено с программным кодом приложения, что упрощает его использование.

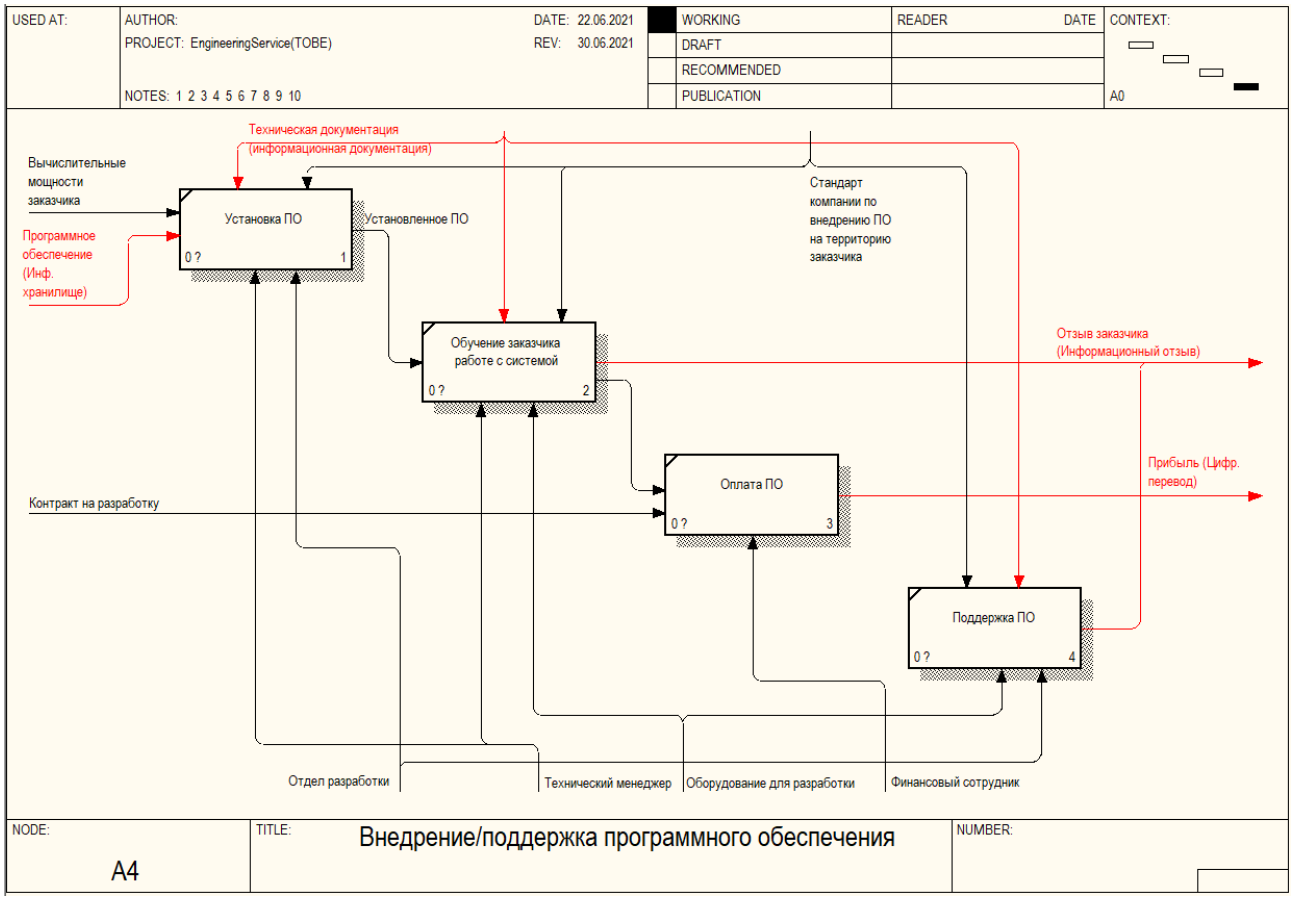


Рисунок 5 – Декомпозиция блока внедрение/поддержка ПО, «КАК ДОЛЖНО БЫТЬ»

В соответствии с рисунком окончательный бизнес-процесс организации будет представлять собой многократные и постоянные обращения к централизованной информационной системе. В конечном результате информация по каждому проекту будет доступна и централизована, компания избавится от утечек информации и будет точно понимать, как поддерживать и улучшать разрабатываемые системы.

Данные, предоставленные в параграфе, отражают бизнес составляющую производства, для оценки технических возможностей системы необходимо разработать диаграмму, основанную на UML методологии.

Целью задачи является достижение отказоустойчивой информационной системы, способной обрабатывать большие объемы информации. Система должна эффективно решать поставленную задачу.

Этап разработки приложения всегда делится на несколько стадий, это можно увидеть в соответствии с рисунком 6. В качестве входной информации всегда являются знания пользователя, которые ему необходимо передать отделам, ответственным за тот или иной этап разработки. Соответственно входной информации, выходная информация будет содержать мысль пользователя в определенных временных рамках, благодаря чему удобно будет отслеживать историю проекта и ход мыслей задействованных сотрудников.

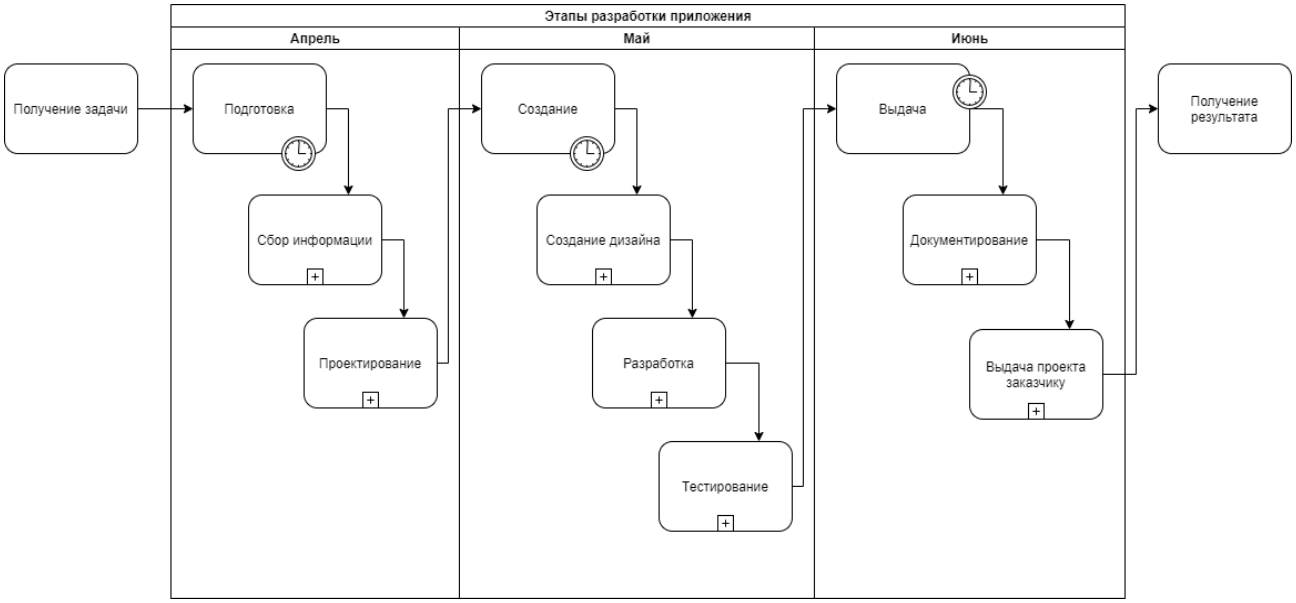


Рисунок 6 – Этапы разработки приложения

Если сотруднику организации потребуется передать задачу далее, ему будет необходимо указать системе проект, на котором он работает и назначить на технического менеджера требуемого отдела. Данные передаются через диалоговые окна, в которых предусмотрены варианты проектов, которые вводятся администраторами и список зарегистрированных пользователей, с указанием должности, для удобства взаимодействия. Сотрудник всегда выбирает тип документа, это может быть техническая задача, бизнес задача, требование от заказчика и т.д. Каждый из этих вариантов можно прикрепить к задаче, для удобства отображения задачи для других пользователей системы.

Система оповещает пользователя о полученной задаче, напоминает о сроках выдачи задачи и позволяет удобно конфигурировать комментарии к задаче. Оповещения приходят также и для других типов документов, например, о необходимости подтвердить график отпуска для другого сотрудника. Пользователь может экспортировать историю задач по определенному проекту в виде одного файла, для создания различных графиков, построенных на основании данной информации. Можно построить график «сложности» проекта, опираясь на потраченное время, количество задач и комментарии сотрудников.

Выводы по главе

Анализ существующих бизнес-процессов отразил эффективность подобных систем. Сбор информации о похожих системах показал, что на данный момент имеются «гиганты» ERP систем, стремящиеся охватить рынок максимально полно, при этом узконаправленные ERP системы довольно редки, либо используются в качестве приватных проектов. Кроме того, приведенные в пример системы написаны с учетом актуальных на тот момент технологий, достаточно массивны и тяжело доступны для компаний, которым нет необходимости переплачивать за неиспользуемые дополнения к основной системе. Разработанный бизнес-процесс рассчитал количество информации, которую необходимо автоматизировать и перенести на вычислительную машину. Были продемонстрированы удобства и для руководящих должностей в компании, вся история разработки проектов доступна с учетом необходимых разрешений для данной информации.

С помощью разработки модели «Как должно быть» была достигнута цель и задачи проекта автоматизации, уменьшено количество рутинного труда, увеличена производственная способность сотрудников. В целом система выглядит удачным подспорьем и способна конкурировать с соперниками по рынку сразу по нескольким параметрам, ориентированности на специфику отдельной компании, легковесность и отказоустойчивость достижение которой невозможно без применения технологий виртуализации.

Глава 2 Логическое проектирование информационной системы

2.1 Выбор технологии логического моделирования ИС

Создание логической структуры проекта заключается в построении диаграмм, отражающих логическую связь и взаимодействие компонентов для достижения цели системы. На данный момент популярными являются два подхода построения логической модели информационной системы:

Объектный – заключается в описании объектов системы и связей между ними, он описывает логическую модель данных в виде объектов и более подробно описывает атрибуты каждого объекта, представляя собой удобный способ формирования логической системы и позволяет переносить архитектуру логической системы сразу в код. Данный подход не описывает связь между различными компонентами системы, а лишь представляет собой техническое описание объектов и их взаимосвязь в рамках одного компонента.

Документарный – представляет собой набор документов, описывающих те или иные функции системы, описывает взаимосвязь между различными компонентами системы на документарном уровне. Минусами данного подхода можно назвать недостаточность данных для переноса документарной модели информационной системы непосредственно в код. Так как документарный подход описывает не объекты системы, а лишь предоставляет документы, описывающие свойства системы. Плюсами являются полное описание всех объектов системы, а соответственно углубленные знания предметной области.

Для данного проекта мы будем использовать объектный подход, так как в работе необходимо раскрыть удобство использования технологий виртуализации, а хорошо они будут раскрываться только при создании мульти-компонентной системы, которая будет удобно масштабироваться и при этом не потребует нескольких вычислительных устройств для работы. В качестве

инструмента для построения логической модели мы будем использовать средства объектно-ориентированного проектирования, использующих нотацию языка UML. Плюсы у данных инструментов заключаются в их многогранности, так как они позволяют описывать как модель базы данных, так и взаимодействия пользователя с различными компонентами системы. На данный момент инструменты разработки логической модели выбраны и можно приступать к разработке логической модели ИС.

2.2 Логическая модель ИС и ее описание

Первоначально необходимо рассмотреть диаграмму модели «use case» (см. приложение Д), чтобы иметь понимание интерфейса приложения, какими компонентами интерфейс будет управлять и взаимодействие внутри системы в целом.

В соответствии с рисунком функционала довольно много, он пересекается и можно сделать вывод что технологии виртуализации дадут нам возможность разделить логически систему на несколько виртуальных машин, что очень упростит взаимодействие как разработчика, так и самого пользователя, так как нагрузка будет распределяться ровно между решаемыми задачами.

Следующим шагом будет разделение самих компонентов системы на абстрактные слои и классы, которые участвуют в передаче и обработке данных.

Первым и самым важным компонентом является средство агрегации технических специалистов, которых и нанимает компания для получения полезного результата от бизнеса.

В соответствии с рисунком 7, диаграмма классов пользовательского компонента представляет из себя логически связанные элементы, так или иначе относящиеся к пользователю или техническому специалисту, который использует параметры пользователя и расширяет их, позволяя работать с пользователем, как с техническим специалистом.

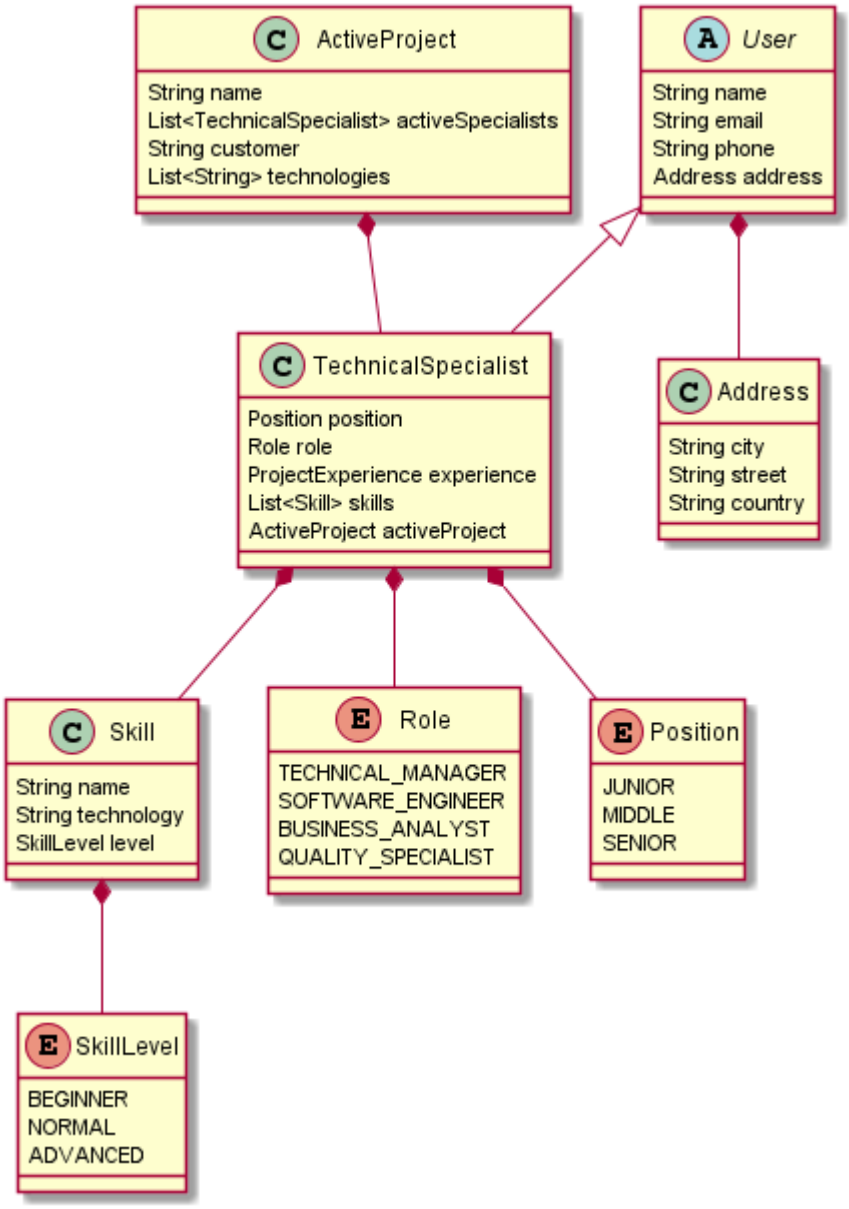


Рисунок 7 – Диаграмма классов компонента “User Manager”

Это первый компонент из четырех, где указаны поля и данные для взаимодействия с пользователями системы. Рассмотрим следующий компонент системы расположенный вторым, как по функциональности, так и по нагрузке на него из других компонентов системы.

Следующим по порядку идет компонент – менеджер документов. В соответствии с рисунком 8, можно наблюдать его наполнение. Во главе компонента, который занимается документооборотом лежит базовый класс “Document”.

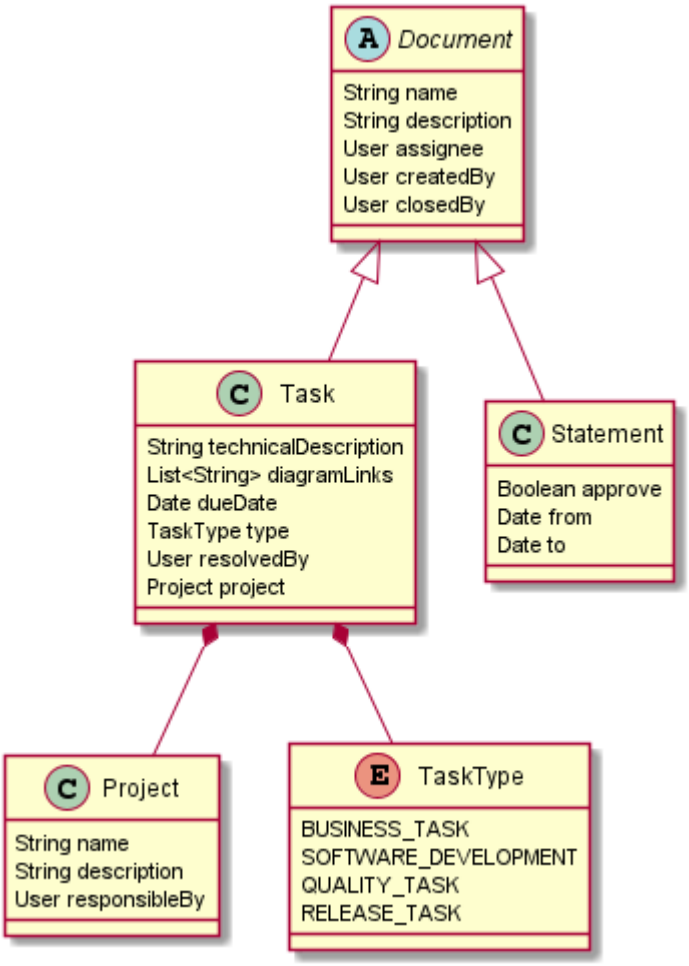


Рисунок 8 – Диаграмма классов компонента “Document Manager”

Данный компонент по своей важности может конкурировать с менеджером пользователей, так как все основные прикладные информации и данные проходят через него, именно данный компонент отвечает за сохранение истории задач и проектов, а также за контролем работы и статусом завершенности различных проектов.

Так как мы ознакомились с основными видами операций и управлением ключевых объектов для проектируемой системы, нам необходимо рассмотреть следующие компоненты, которые являются прикладными и не имеют весомых задач, но позволяют расширять функционал системы, добавляя ему удобные возможности.

Следующий компонент призван сохранять данные пользователя и его пройденные тренинги, его классы изображены в соответствии с рисунком 9.

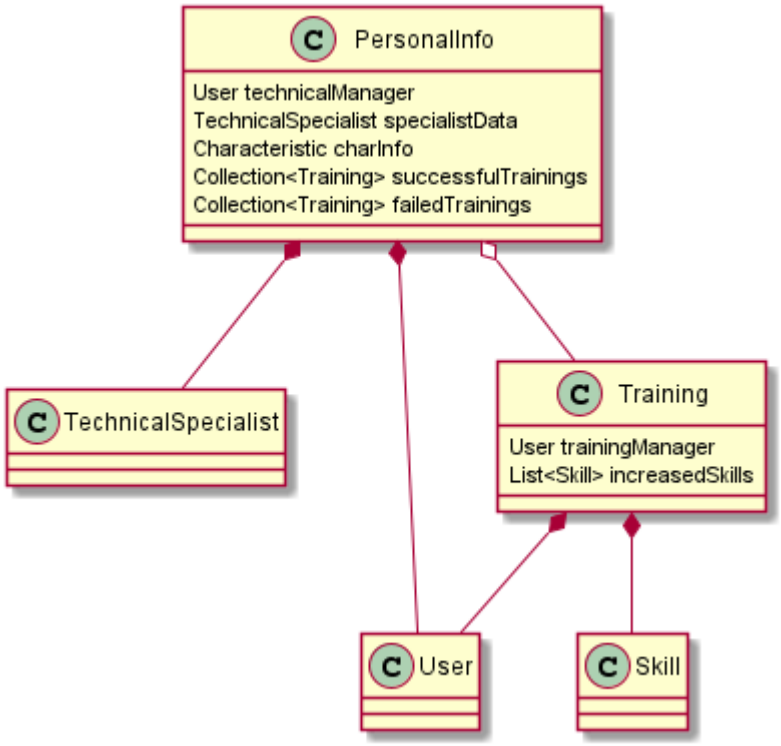


Рисунок 9 – Диаграмма классов компонента “Personal Data Manager”

Компонент управляющий и сохраняющий персональную информацию является связующим звеном между техническими специалистами и данными о навыках и умениях, которые у них присутствуют. Имея данную информацию можно сделать вывод, каких знаний и навыков не хватает сотрудникам для их продвижения по карьерной лестнице и увеличению своей эффективности.

Именно поэтому данный компонент предоставляет всю указанную информацию, позволяет следить за прогрессом сотрудников, выявлять слабые места в их технических знаниях и оперативно устранять недостатки, пользуясь функционалом тренингов.

Поговорив о связующем звене между системой тренингов и системой пользователей, не лишним будет упомянуть модель данных системы тренингов, для учета всех особенностей системы.

В соответствии с рисунком 10 можно увидеть структуру данных курсов, тренингов и закрепленные к ним навыки, для более быстрого подбора тренингов.

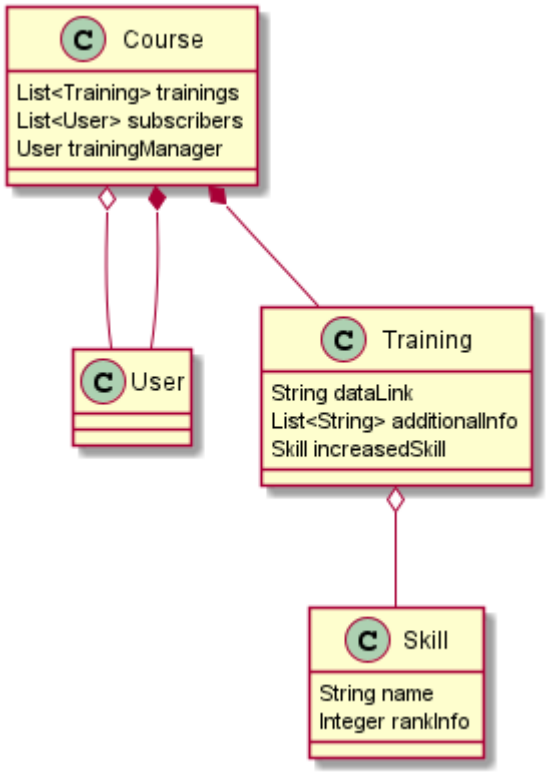


Рисунок 10 – Диаграмма классов компонента “Training System”

Компонент представляет из себя три ключевые сущности, которые могут использованы в ранее описанных компонентах. Это курс, который содержит цепочку тренингов, отвечающего пользователя за собранный курс и коллекцию подписанных пользователей на данный курс. Тренинг представляет из себя абстрактные данные и данные о навыках, которые он развивает.

Тренинг и курс представлены в виде композиции, так как курс не может не содержать в себе тренингов, а отношения между тренингом и навыком представлены в виде агрегации, так как тренинг может и не развивать какого-то либо навыка, а быть просто вводной лекцией для новых систем и знаний.

Кроме представленных компонентов есть еще два, один из них является компонентом для просмотра новостей, в стандартной реализации он может исключаться, так как не каждая компания постоянно снабжается большим количеством новостей и «мессенджер» является аналогом любого мессенджера и задается только целью сохранения конфиденциальных сообщений, ничем по большому счету, не отличаясь от большой массы представителей.

2.3 Информационное обеспечение ИС

2.3.1 Используемые классификаторы и системы кодирования

Вместе с информационным продуктом важную роль занимают классификаторы экономической информации. Данный раздел служит для сокращения хранимой информации, соответственно увеличивает время на поиск информации. Правильно подобранная система кодирования информации позволит усилить фундамент информационной системы, увеличить её производительность и сохранить достоверность информации. Существует две системы классификации, иерархическая и многоаспектная.

Иерархическая система предполагает разделение исходных данных, декомпозируя их до тех пор, пока отношения не станут прозрачны и доступны. Между такими данными обычно возникают иерархии, в соответствии с рисунком 2.10 объект типа “Document” является базовым классом и у данного класса есть потомки, объекты “Task” и “Statement” это нечто иное как иерархическая связь.

Многоаспектная система использует несколько точек зрения для оценки итоговой ИС разделяя все элементы системы на аспекты (фасеты). Данная система делится также, на дескрипторную и фасетную системы. Такая система имеет под собой разделение элементов системы на некие подмножества и выделяется их последовательность с помощью частоты обращений к ним. Что бывает полезно для выделения объектов, которым изначально необходимо максимальное быстродействия, учитывая частоту обращений. Для описания дескрипторной системы понадобится привести аналогию с библиотечным поиском, отбираются ключевые слова и словосочетания, которые описываю предметную область. Из всех приведенных ключевых слов отбираются наиболее часто используемые. Часто используемые поддаются нормализации и сохраняются в системе в качестве дескрипторов. Между дескрипторами возникают связи, которые впоследствии и описывают отношения объектов.

Задачей кодирования является однозначное описание объектов и обеспечение необходимой достоверности информации. Выделяется три системы

кодирования. Регистрационная система используются для объектов, которые не требуют классификации и не зависят от существа решаемых задач, используются для решения абстрактных задач. Порядковая система регистрирует объекты по порядку. Признаки классификации отсутствуют, что впоследствии не позволит получать промежуточные итоги. Серийно-порядковая система используется для объектов с одним признаком, находящихся в определенной подчиненности. Старшему признаку выделяется серия номеров с учетом расширения позиции объекта, а младшему присваиваются порядковые номера в пределах признака старшего.

В соответствии с требованиями к программному продукту, в системе используется серийная система кодирования, учитывающая расширение множества и разделение по признаку классификации. Системы кодирования не привязаны к системе как таковые, таким образом позволяя не заострять внимание на конкретной системе кодирования. Сводные характеристики предоставлены в соответствии с таблицей 2.

Таблица 2 - Сводные характеристики объектов классификации

Наименование классификатора	Значность кода	Система кодирования	Система классификации	Вид классификатора
Код документа	5	Порядковая	Иерархическая	Локальный
Код пользователя	6	Порядковая	Иерархическая	Локальный
Код проекта	5	Порядковая	Иерархическая	Локальный
Код тренинга	5	Порядковая	Иерархическая	Локальный
Код курса	5	Порядковая	Иерархическая	Локальный
Код должности	5	Порядковая	Иерархическая	Локальный

После определения систем кодирования, системы классификации и определения основных моделей ИС, необходимо проанализировать входную

информацию, чтобы определить степень её используемости и перечень основных полей.

2.3.2 Характеристика нормативно-справочной и входной оперативной информации

Для анализа входной информации необходимо продумать, какая информация будет полезна для пользователя, где эту информацию лучше хранить и как классифицировать. Для ответа на эти вопросы необходимо составить таблицу входной информации. В соответствии с таблицей 3, появляется понимание данных, которые системой должны быть сохранены и обработаны

Таблица 3 – Таблица входной информации систем “ERP”

№	Вид информации (Объект)	Источник поступления данных	Периодичность
1	Пользователь	Заполняется техническим менеджером	Зависит от факта найма кадров организацией
2	Документ, включая различные его сценарии (Проектный, Организационный)	Заполняется пользователем, разным типам пользователей доступны документы разных типов	Постоянно, максимальная периодичность
3	Проект	Заполняется техническим менеджером	Зависит от количества проектов компании

Продолжение таблицы 3

№	Вид информации (Объект)	Источник поступления данных	Периодичность
4	Персональная информация	Заполняется зарегистрированным пользователем	Один раз при получении должности, далее обновляется по факту получения новых навыков
5	Навыки пользователя	Заполняется Тренинг-менеджером	После прохождения тренингов/курсов
6	Тренинг	Заполняется Тренинг-менеджером	зависит от количества информации в компании
7	Курс	Заполняется Тренинг-менеджером	зависит от количества информации в компании

Входная информация полезна при разработке системы, представлена в виде таблиц баз данных, каждый элемент таблицы снабжен типом, который позволит ему хранить в себе несколько вложенных параметров. У каждого элемента таблицы есть первичный ключ, некоторые объекты при этом хранят в себе список объектов. Например, курс хранит в себе список предназначенных к нему тренингов и так далее. В соответствии с таблицей 4, можно ознакомиться с хранением данных детальнее.

Таблица 4 – Типизированная таблица входной информации

Объект	Название поля	Перевод	Тип данных	Ключ
Пользователь	Идентификатор	Id	Integer	*
	Имя	Name	String	
	Адрес	Address	Object	emb
	Телефон	Phone	String	
	Доп. Данные	AdditionalData	Collection<String>	
Адрес	Город	City	String	
	Улица	Street	String	
	Дом	House	String	
	Квартира	Apartment	String	
Документ	Название	Name	String	
	Идентификатор	Id	Integer	*
	Описание	Description	String	
	Назначен на	Assignee	User	
	Создан	createdBy	User	
	Закрыт	closedBy	User	
Проект	Имя	Name	String	
	Идентификатор	Id	Integer	*
	Описание	Description	String	
	Ответственный	responsibleBy	User	
	Технический менеджер	technicalManager	User	

Продолжение таблицы 4

Объект	Название поля	Перевод	Тип данных	Ключ
Персональная информация	Идентификатор	Id	Integer	*
	Данные специалиста	specialistData	SpecialistData	
	Характеристика	charInfo	Characteristic	
	Пройденные тренинги	successfulTrainings	Collection<Training>	
	Проваленные тренинги	failedTrainings	Collection<Training>	

После того, как мы поговорили о входной информации, настало время поговорить о выходной из информации, чтобы понимать результат работы системы.

2.3.3 Характеристика выходной информации

Для систем ERP в качестве выходной информации могут служить завершенные документы, завершенные продукты заказчика, экономия времени на централизованное управление компанией. Центральным документом во всей системе выступает документ различных типов, а в частности результат работы, это программное средство проанализированное и разработанное в рамках документа. Кроме того, результатом работы также является история завершенных проектов компании, отмененных проектов и история разработки каждого отдельного проекта. Данную историю можно использовать в качестве портфолио для увеличения количества потенциальных клиентов и соответственно, увеличение получаемой прибыли.

2.4 Проектирование БД ИС

Проектирование базы данных очень важный шаг в рамках любой информационной системы, так как правильно спроектированная база данных позволит получить максимальное быстродействие, уйти от различного рода ошибок связанных с созданием связей и позволит упростить информационную систему в дальнейшем.

Для более полного понимания использования технологий виртуализации, первый вариант данной диаграммы предоставлен без использования технологий виртуализации, так если бы это была монолитная архитектура информационной системы. Диаграмма (см. приложение Е) демонстрирует диаграмму баз данных с учетом использования только одной базы данных, без использования технологий виртуализации. Из диаграммы следует что все связи между объектами укладываются в одну базу данных, при этом объекты связаны в рамках логического взаимодействия. Например, пользователь связан с таблицей адреса один к одному. Объект пользователя хранит в себе идентификатор связанного адреса, что позволяет быстро находить информацию по месту жительства пользователя и не загружать целый каждый раз, когда идет прямое обращение к таблице пользователя.

Так как объектов много и все они хранят в себе полезную информацию, мы можем использовать аппаратно-программные средства современных баз данных, чтобы создать кластер под систему и разделить базы данных, группируя их по общему назначению.

Диаграмма (см. приложение Ж) демонстрирует наглядные изменения, при использовании технологий виртуализации, позволяя разделять элементы баз данных и в соответствии с данным фактом, разделять нагрузку на обращения к соседним элементам системы. Новые базы данных представлены в виде монолитной информационной системы, на практике же, сохранится связь между выделенными объектами, но для подбора необходимых данных, необходимо

будет обратиться к соседнему микро-сервису, чтобы получить данную информацию.

Таким образом пользователи, работающие с системой в отдельном блоке, например, в менеджере новостей просматривая новости, никак не пересекаются с серверным приложением менеджера пользователей, только если не хотят получить специфическую информацию по определенным пользователям.

Для любой информационной системы, для работы в штатном режиме необходимы ресурсы вычислительных машин, для расчета и подведения итоговых ресурсов для виртуальной машины производится финальный расчет ресурсов данной главы.

2.5 Требования к аппаратно-программному обеспечению ИС

Информационная система предназначена для установки на сервер заказчика, таким образом для установки базового приложения понадобится:

– Процессоры с поддержкой (x86-64, ARM, s390x, ppc64le);

– Операционная система с аппаратной поддержкой Hyper-V;

При этом на каждую виртуальную машину системы понадобится:

– 50 Мб оперативной памяти;

– Доступ до вычислительной мощности процессора;

– Количество свободного места на жестком диске, в количестве, зависящем от размера и количества данных в системе;

Так как используются ресурсы, не выходящие за рамки контейнера, то и требования к аппаратно-программному обеспечению изменяются, в зависимости от количества данных, используемых в системе. Количество же контейнеров варьируется от количества функций системы. В базовом варианте программного обеспечения используются 4 микро-сервиса, обеспечивающие базовый функционал системы (см. приложение З).

Выводы по главе

В данной главе был рассмотрен общий функционал системы, спроектированы диаграммы классов и рассмотрены две версии диаграммы базы данных, подходящих системе, для получения максимального количества знаний о применяемых технологиях и отличии информационной системы, от систем более ранних образцов.

Прежде всего, была подобрана технология логического моделирования информационной системы, описаны плюсы и минусы существующих технологий моделирования.

Логическая модель информационной системы описана, в соответствии с необходимой функциональностью, а также описано информационное обеспечение информационной системы, проектирование базы-данных и требования к аппаратно-программному обеспечению.

В результате анализа, спроектированная система по монолитной архитектуре с использованием одного информационного приложения, была разбита на несколько сервисов предоставляющих идентичный функционал, не теряя общей работоспособности и связности данных. Уменьшено общее количество ресурсов благодаря технологиям виртуализации и упрощена архитектура системы, для более удобной поддержки и разработки.

Глава 3 Физическое проектирование ИС

3.1 Описание общей функциональности ИС

Процесс взаимодействия с системой для сотрудника организации, разделяется на несколько этапов. Среди которых выделяют:

- Вход в систему (Авторизация);
- Получение роли и привилегий (Аутентификация);
- Работа в системе с тем или иным модулем;
- Получение задач от вышестоящего руководства;
- Создание задач и изменение их статуса;
- Выдача задач после их выполнения;
- Дальнейшее взаимодействие с пользователями;

Данное взаимодействие тесно связано с выбранной ролью на этапе аутентификации. Этап работы в системе может запрещаться, если роль не соответствует наличию привилегий. Вышестоящее руководство может разделяться в зависимости от роли, например, вышестоящий начальник над сотрудником отдела тестирования, это руководитель отдела тестирования.

Кроме указанных этапов существуют дополнительные функции работы в системе:

- Просмотр новостей;
- Просмотр назначенных тренингов;
- Прохождение тренингов;
- Оформления отпусков;
- Создание и получение задач;
- Назначение задач;
- Отслеживание статуса задач;
- Выполнение задач;

Система довольно целостна и предусматривает наличие множества функций для сотрудников организации. Рассмотрим процесс работы с системой с точки зрения конкретного пользователя. Первоначально происходит вход в систему с помощью персональных данных, далее в процессе аутентификации пользователь получает роль – «Сотрудник отдела тестирования». Доступные модули для данной роли следующие: Документальная система, пользовательская система, тренинг-система. Кроме выполнения рабочих задач, можно проверить наличие специальных тренингов, просмотреть новости компании и т.д. Для реализации приложения необходимо выбрать одну из общеизвестных архитектур ИС, чему и посвящен следующий раздел.

3.2 Выбор архитектуры информационной системы на основе технологий виртуализации

Архитектура ИС - концепция, определяющая структуру, модель, функции и связь компонентов информационной системы. Она отражает из каких составных частей (компонент) будет состоять система и как эти части между собой взаимодействуют. В соответствии с рисунком 11 показаны основные компоненты информационной системы.

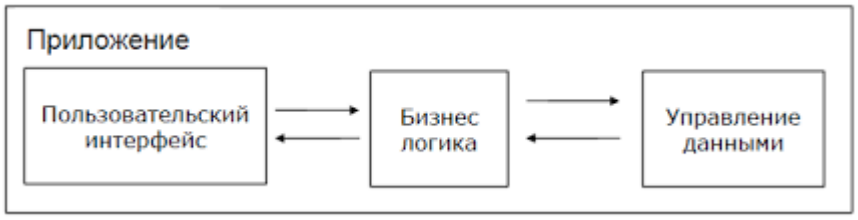


Рисунок 11 – Компоненты информационной системы

Выполняемые функции данных компонент можно разделить на три слоя.

Слой представления (пользовательский интерфейс) – все, что связано с взаимодействием с пользователем: нажатие кнопок, движение мыши, отрисовка изображения, вывод результатов поиска и т.д.

Бизнес логика – правила, алгоритмы реакции приложения на действия пользователя или на внутренние события, правила обработки данных.

Слой доступа к данным – хранение, выбор, модификация и удаление данных, связанных с решаемой приложением прикладной задачей.

Архитектура информационной системы развивалась, в зависимости от решаемой задачи. Для каждого нового приложения выбиралась модифицированная архитектура, заточенная на решение определенной задачи. На данный момент они делятся на два вида, исключая их модификации:

Файл-сервер – файлы в такой архитектуре хранятся на сервере. Слои представления, бизнес-логики и частично доступа к данным расположены на клиенте. Сервер никак не участвует в обработке бизнес-логики, лишь сохраняет и управляет файлами. Количество клиентов такой архитектуры ограничено десятками. Плюсами являются: низкая стоимость разработки, удобство централизованного управления доступом и многопользовательский режим работы с данными. Минусы это: частичная поддержка многопользовательского режима, обработка данных на клиенте может обернуться потерей информации.

Клиент-сервер – архитектура предусматривает обработку бизнес-логики как на сервере, так и на клиенте. Слой представления целиком находится на клиенте, а доступ к данным на сервере, что позволяет более равномерно распределять нагрузку. Особенностью такой архитектуры является использование запросов к серверному ПО, функции приложения равномерно распределены. Плюсами являются: гарантия целостности данных, полная поддержка многопользовательской работы. Минусы это: высокие требования к пропускной способности канала с серверов, слабая защита от взлома, высокая сложность администрирования, высокая сложность разработки.

Архитектура файл-сервер не подходит для данного приложения, так как оно рассчитано на огромное количество пользователей единовременно. Необходима полная поддержка многопользовательского режима, из-за этого выбор сделан в пользу клиент-серверной архитектуры.

В сообществе веб-разработки, существует несколько вложенных архитектур, которые позволяют использовать HTTP[2] протокол таким образом, чтобы избавить от недостатков стандартную архитектуру - клиент-сервер. Стандартный клиент-сервер - это монолитно-ориентированная архитектура, в которой используется один клиент и один сервер. Микро-сервисная архитектура позволяет разбивать слои сервера и клиента, увеличивая пропускную способность приложения. Например, если клиент 1 захочет просмотреть личную информацию, он задействует компонент персональной информации, а клиент 2, просматривая новости, использует только компонент менеджера новостей. В случае с монолитом клиент 1 и клиент 2 будут использовать один и тот же сервер-приложения, что увеличит нагрузку.

В проекте будет использоваться микро-сервисная архитектура, она необходима для логического разделения компонент-серверов, для уменьшения недостатков клиент-серверной архитектуры, а также для наглядной демонстрации принципов виртуализации (см. приложение И). На рисунке можно увидеть, как увеличивается эффективность приложения, благодаря такому подходу.

Типовым решением для взаимодействия между пользовательским интерфейсом и сервером является REST[3]. Данная технология позволит обмениваться информацией между пользовательским интерфейсом и сервером, так и между компонентами сервера, для решения задачи. Например, когда пользователь обращается к сервису новостей, проверяется наличие доступа. Если он отсутствует, то клиент проходит авторизацию повторно, получая новый

---

[2] **HTTP** (англ. *HyperText Transfer Protocol* — протокол передачи гипертекста) — протокол прикладного уровня передачи данных, изначально — в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных.

[3] **REST** (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 сети интернет вызов удалённой процедуры может представлять собой обычный HTTP-запрос (обычно GET или POST; такой запрос называют «REST-запрос»), а необходимые данные передаются в качестве параметров запроса.

токен доступа в сервисе управления пользователями и уже с данной ролью обращается в сервис новостей. Учитывая, что новый токен доступа имеет доступ, пользователь сможет ознакомится со списком доступных ему новостей. Данный процесс описан в соответствии с рисунком 12.

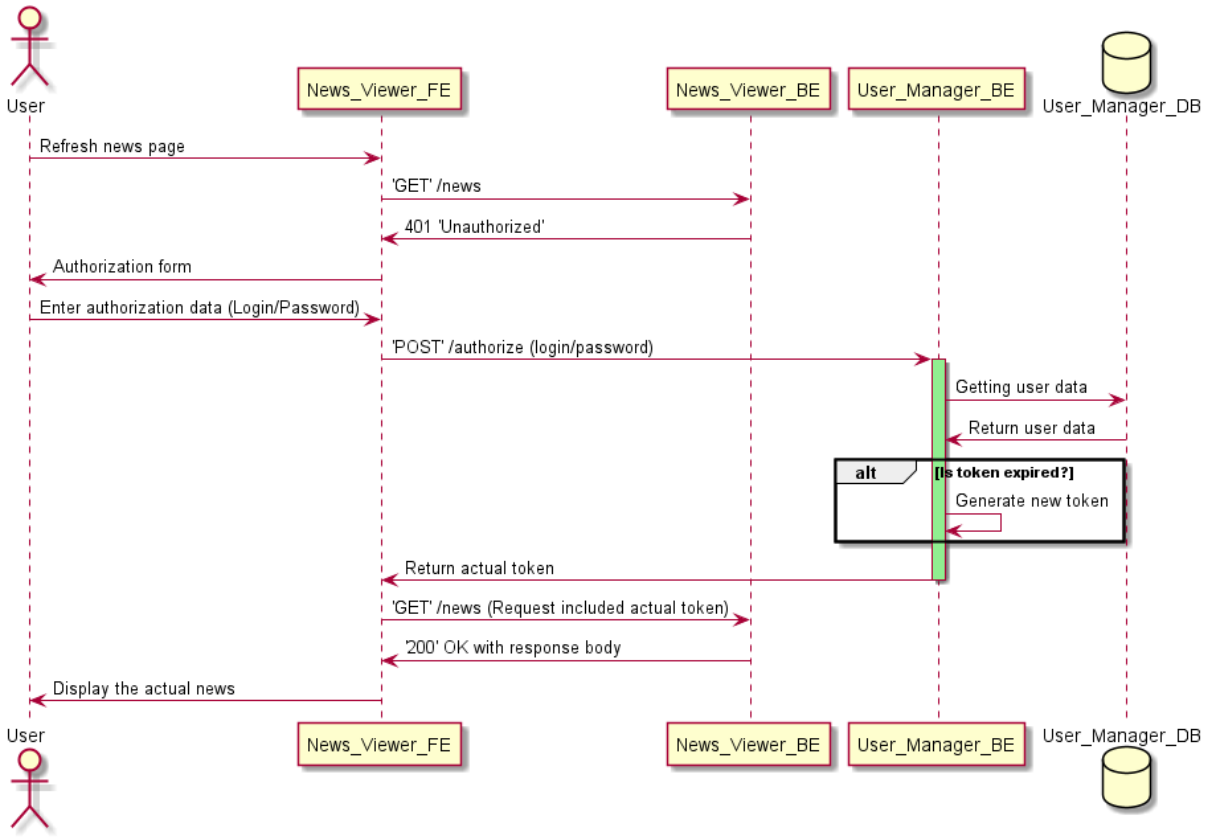


Рисунок 12 – процедура авторизации из стороннего модуля

Выбранные в этом параграфе инструменты позволят раскрыть весь потенциал вычислительных машин, распределяя равномерно нагрузку на разные типы данных. Тип систем ERP подразумевает собой большое количество пользователей одновременно и работу с множеством разно-типовой информации, поэтому выбранная архитектура отлично подходит для разработки приложения. Далее нам предстоит выбрать конкретные технологии, подходящие к данной архитектуре и определиться с набором инструментария для виртуализации приложения.

3.3 Выбор технологии разработки и инструментов виртуализации программного обеспечения

Данный раздел будет разделен на несколько пунктов, для более конкретного описания всех средств разработки. Всего будет использоваться несколько типов продуктов разработки, среди которых можно выделить:

- Среду разработки[4];
- Средство взаимодействия с СУБД;
- Средство разработки конфигурации приложения;
- Язык разработки сервера-приложения;
- Язык разработки клиента-приложения;
- Компилятор приложения;
- Средства ручного тестирования API для веб-приложений;
- Средства контроля версий, поддерживающие CI[5];
- Средства установки приложения на сервер;

Среда разработки представляет из себя комплекс программных средств, упрощающих взаимодействие с программным продуктом. Серверная часть веб-приложений использует программный язык Java. Данный язык поддерживает

---

[4] **Интегри́рованная среда́ разрабо́тки**, ИСР (англ. Integrated development environment — IDE), также единая среда разработки, ЕСР — комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО).

Среда разработки включает в себя:

- текстовый редактор,
- Транслятор (компилятор и/или интерпретатор),
- средства автоматизации сборки,
- отладчик.

[5] **Непрерывная интеграция** (CI, англ. Continuous Integration) — практика разработки программного обеспечения, которая заключается в постоянном слиянии рабочих копий в общую основную ветвь разработки (до нескольких раз в день) и выполнении частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем.

кроссплатформенность, позволяет задействовать технологии ООП[6], кроме того он относительно прост в использовании.

Существует несколько IDE, которые могут подходить для разработки с учетом выбранного языка, в соответствии с таблицей 5 можно увидеть их сравнительную характеристику.

Таблица 5 – Сравнительная таблица

	IntelliJ Idea	NetBeans	Eclipse
Лицензия	Возможность использования бесплатной версии	Только платное использование	Бесплатна
Поддержка других языков разработки	Поддержка с помощью плагинов	Отсутствует	Отсутствует
Поддержка плагинов	Полная	Полная	Полная
Обновления	Регулярные	Редки	Редки
Поддержка средств взаимодействия с СУБД	Полная	Отсутствует	Отсутствует
Поддержка JVM средств контроля памяти	Поддержка с помощью плагинов	Отсутствует	Полная

[6] **Объе́ктно-ориенти́рованное программи́рование** (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

Можно сделать вывод о том, что самой удобной средой разработки для веб-приложения является IntelliJ Idea. Она позволяет нам полную конфигурацию с помощью плагинов, удобную подсветку синтаксиса, редактирование кода с помощью горячих клавиш и средства управления СУБД, это очень важно поскольку в описанном приложении используется сразу несколько СУБД, принадлежащих разным компонентам. Компилятор JVM и система сборки приложения вшита в указанную IDE, это означает что пункты не нуждаются в анализе и выборе.

Самым современным и удобным средством контроля версий является “GIT[7]”. Данное средство интегрировано со средствами постоянной интеграции на платформе “GitLab[8]”.

Удобным языком для разработки приложения-клиента является JavaScript, так как имеет в арсенале фреймворки, способные отображать большие объемы информации, используя малое количество кода и большое количество пресетов, которые поддерживают современные дизайнерские сообщества. Примером является Material Design[9], который и будет использоваться для отображения интерфейса приложения.

Средством разработки приложения-клиента является выбранная IntelliJ IDEA, так как она поддерживает разработку на языке JavaScript, который используется для отображения пользовательских интерфейсов.

---

[7] **Git** - распределённая система управления версиями. Система управления версиями от англ. Version Control System, VCS или Revision Control System - программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

[8] **GitLab** — веб-инструмент жизненного цикла DevOps с открытым исходным кодом, представляющий систему управления репозиториями кода для Git с собственной вики, системой отслеживания ошибок, CI/CD пайплайном и другими функциями.

[9] **Material Design** — стиль графического дизайна интерфейсов программного обеспечения и приложений, разработанный компанией Google.

Инструмент для обеспечения полноценной виртуализации – VMWare. Данный софт позволяет конфигурировать и запускать целую платформу приложений, разбитых на контейнеры и объединенных с помощью общего окружения. Каждое разработанное приложение отделено с помощью виртуальной машины и доступ к ней можно обеспечить с помощью виртуальных средств маршрутизации.

Средства ручного тестирования на языке Java представляют из себя библиотеки для прогона отдельных участков кода, для выявления неисправностей. Самой популярной библиотекой является Junit, популярность приобретена благодаря участию разработчиков языка, в создании библиотеки.

Средства установки приложения предоставит нам описанный ранее сервис GitLab. Система пайплайнов, интегрированная в инструмент жизненного цикла, позволит компилировать, тестировать, собирать и устанавливать на сервер новые версии приложений, после их публикации.

Общий процесс будет выглядеть так:

– Разработка идеи приложения;
– Разработка программного кода;
– Разработка тестов для программного кода;
– Публикация кода;
– Виртуализация кода приложения;
– Установка приложений на виртуальные машины сервера с учетом изменений последней публикации;

Поддержка и работоспособность данного процесса возможна только с участием конфигураций, для всего приложения в целом и его отдельных сервисов. Перед установкой на сервер каждый сервис приложения должен быть установлен на виртуальную машину подхватив изменения последних публикаций. Информационные системы должны быть доступны из среды виртуальных машин. В процессе участвуют разработанные конфигурации:

– Конфигурация git-lab-ci;

– Конфигурация виртуальной машины;

Виртуальная машина будет запущена с предустановленной операционной системой Linux, в данном случае это внутренний сервер организации, но это вполне может быть и машина в сети-интернет с открытым IP адресом и DNS именем.

В данном параграфе были описаны технологии разработки и примененных инструментов. Приложению для начала разработки не хватает описанной физической модели СУБД. В приложении со средствами виртуализации СУБД играет особую роль, так как вырастают требования по конфигурации и отказоустойчивости.

3.4 Выбор СУБД ИС

Для выбора СУБД для приложения необходимо руководствоваться следующими фактами:

– Отказоустойчивость,
– Доступность,
– Поддержка виртуализации,
– Лицензия распространения,
– Опыт использования.

Совместно с выбранными средствами разработки.

Популярные СУБД, поддерживающие технологии виртуализации и хранение данных в облаке это PostgreSql, MongoDB, MySql. MongoDB применяет технологии «документального» сохранения данных, без помощи языка запросов SQL. Два других представителя используют SQL и сохраняют данные в таблицах. В соответствии с таблицей 6, можно вычислить необходимые данные для сравнения.

Таблица 6 – Сравнительная характеристика современных СУБД

	PosgreSQL	MongoDB	MySql
Тип лицензии	Открытый код	Бесплатное использования	Бесплатное использование/коммерческое использование
Поддержка виртуализации	Есть	Есть	Частично
Документация	Не полная	Полная	Полная
Поддержка SQL	Полная	Нет	Полная
Поддержка Json формата хранения	Частичная	полная	Нет
Поддержка транзакций	Полная	Частичная	Полная
Цена поддержки в облаке	Низкая	Высокая	Низкая
Производительность	Средняя	Средняя	Низкая
Поддержка связей/ООП	Да	Частичная	Да
Наличие механизмов сохранения данных, при критических ситуациях	Да	Да	Нет
Сбор и обработка статистики	Частично	Да	Нет

Решающим фактором при выборе является простота, надежность и бесплатное использование. В данном проекте будет использоваться PosgreSQL, но и MongoDB является неплохим выбором, так как предоставляет схожий

функционал и тратит меньше времени на разработку за счет хранения документов. Обе базы данных полностью поддерживают виртуализацию и шардинг, однако PostgreSQL является более распространенным сочетанием с выбранным языком Java.

В данном разделе была выбрана база данных, подходящая под описанную задачу, в следующем разделе речь пойдет о способе хранения данных и физической модели объектов.

3.5 Разработка физической модели данных ИС

Способ хранения у выбранной базы данных поддерживает SQL синтаксис и табличное использование, соответственно в данном параграфе будет разработана физическая модель хранения данных, разделенная на сервисы. Планируется разработать четыре информационных системы, хранящих свою информацию, связанные с помощью веб-взаимодействия:

- Сервис управления пользователями;
- Сервис управления документами;
- Сервис хранения личной информации;
- Тренинговая система;

Далее речь пойдет о физической модели данных для каждого сервиса по списку. Таблицы физической модели данных будут представлять из себя описание хранимых данных в форме базы данных. Описательные таблицы разделены на несколько колонок, среди которых:

- Имя колонки;
- Тип сохраняемой информации, например – varchar(32), text;
- Возможность сохранять пустое значение;
- Стандартное значение, используемое если в поле пустое значение, либо пустая строка;

– Опциональное поле, содержащее описание поля, заполняется в случае необходимости;

3.5.1 Сервис управления пользователями

Данный сервис является самым объемным, агрегирует информацию по пользователям в системе, хранит данные доступа и регистрирует технических специалистов. Его нагрузка заключается в хранении пользователей и предоставления доступа к ним, кроме того он заведует токенами доступа и содержит много конфиденциальной информации, что влияет на требования к безопасности. В соответствии с таблицей 7 можно ознакомится с физической моделью данных сервиса.

Таблица 7 - Пользователь

Field	Type	Nullable	Default
id	uuid	false	
name	varchar(32)	false	
address	text	false	
phone	varchar(32)	false	

Данная таблица содержит краткую информацию о зарегистрированном пользователе. Поле идентификатора, имени, адреса, телефона не могут быть пустыми, так как это базовая информация, которая необходима для связи с пользователем. В соответствии с таблицей 8, пользователи пересекаются с дополнительными данными, связаны как один ко многим, что означает что один пользователь может иметь много строк дополнительных данных.

Таблица 8 – Связь дополнительных данных

Field	Type	Nullable	Default
user_id	uuid	false	
additionalData	text	false	

Упомянутая таблица пользователей пересекается еще и с таблицей адреса пользователя, данная таблица содержит подробную информацию о местонахождении. Модель данных продемонстрирована в соответствии с таблицей 9.

Таблица 9 – Адрес пользователя

field	Type	Nullable	Default
user_id	uuid	false	
city	text	false	
street	text	true	
house	text	true	
apartment	text	true	

В предоставленной таблице, мы можем наблюдать изменения. Некоторые данные могут содержать значение нал, и в этом случае заменятся на строковое значение empty, что будет означать отсутствие данных.

Следующая таблица связывает сущности пользователя с его ролью и назначенным проектом. Роль представляется в роли перечислений и может содержать только одно из предустановленных выражений. Модель данных предоставлена в соответствии с таблицей 10.

Таблица 10 – Технический специалист

Field	Type	Nullable	Default	Description
user_id	uuid	false		
role	varchar(32)	false		TECHNICAL_MANAGER, SOFTWARE_ENGINEER, BUSINESS_ANALYST, QUALITY_SPECIALIST

Продолжение таблицы 10

Field	Type	Nullable	Default	Description
active_project_id	uuid	false		связывает идентификатор проекта и технического специалиста

Поле идентификатора пользователя, связано с полем из таблицы user, а поле идентификатора активного проекта отсылает нас к таблице из сервиса документооборота, который описан в следующем пункте.

3.5.2 Сервис управления документами

Сервис документооборота представлен в виде обработчика различных документов, относящихся как к трудовым задачам, так и к отпускам и прочих активностей. Главной сущностью, является задача, она предназначена для хранения информации о задаче и содержит большое количество полей, модель данных предоставлена в соответствии с таблицей 11.

Таблица 11 - Задача

Field	Type	Nullable	Default	Description
id	uuid	false		
name	varchar(64)	false		
description	text	true		
assignee	uuid	true		
created_by	uuid	false		
closed_by	uuid	true		
technical_description	text	true		техническое описание задачи

Продолжение таблицы 11

Field	Type	Nullable	Default	Description
due_date	date	true		дата выдачи
type	varchar(32)	false		возможные значения: BUSINESS_TASK, SOFTWARE_DEVELOPMENT, QUALITY_TASK, RELEASE_TASK
resolved_by	uuid	true		
project_id	uuid	true		отображает отношение задачи к определенному проекту

Так как в данной таблице существует большое количество полей, необходимых для другой сущности, поля, выделенные серым, перешли из базовой сущности в соответствии с принципами наследования. Следующая таблица представляет из себя ячейку для хранения действий, в соответствии с таблицей 12.

Таблица 12 – Действие

Field	Type	Nullable	Default	Description
id	uuid	false		
name	varchar(64)	false		
description	text	true		
assignee	uuid	true		
created_by	uuid	false		
closed_by	uuid	true		

Продолжение таблицы 12

Field	Type	Nullable	Default	Description
approved	boolean	false	false	наличие подтверждения действия
from	date	true		с какой даты осуществляется действие
to	date	true		до какой даты осуществляется действие

Действия в рамках проекта могут быть разнообразны, самый простой пример – это планирование отпуска. Таблица содержит поля с какого и по какое число оно происходит и отметку о подтверждении действия. Тип действий также может представлять из себя увольнение или запланированное мероприятие.

Следующая таблица поможет нам в сохранении данных проектов, они необходимы для создания ссылок на сотрудников и планирования персонала для руководителей организации. В соответствии с таблицей 13, предоставлена физическая структура данных.

Таблица 13 – Проекты

Field	Type	Nullable	Default
id	uuid	false	
name	varchar(32)	false	
technical_manager	uuid	false	ответственный технический менеджер
description	text	true	описание проекта
customer	text	false	информация о заказчике

Описанная таблица является важной, так как хранит информацию о непосредственном заказчике, которая может являться конфиденциальной. Средства защиты обязательны, для данной таблицы.

Так как описанные данные не содержат персональной информации о сотрудниках организации, а они необходимы для ведения статистики и отображения персональной информации сотрудников и распределения по проектам, существуют сервисы хранения личной информации и тренинговая система, описанные в следующих пунктах.

3.5.3 Сервис хранения личной информации

Следующий сервис представляет из себя данные о сотрудниках, имеет ссылки на технического специалиста, созданного в сервисе менеджер пользователей. Первая модель данных представляет из себя связь между картой персональной информации и идентификатором технического специалиста. Таблица содержит текстовое поле, для описания характера и черт рассматриваемого сотрудника, они необходимы для грамотного управления сотрудниками. Модель данных предоставлена в соответствии с таблицей 14.

Таблица 14 – Персональная информация

Field	Type	Nullable	Default
id	uuid	false	
name	varchar(32)	false	
technical_specialist	uuid	false	
character_information	text	true	null

Запись в таблице создается на этапе создания технического специалиста, а заполняется менеджером тренингов. Данная модель пересекается с двумя моделями связей успешные тренинги и провальные тренинги для каждого сотрудника, эти данные хранятся для анализа сотрудников, выявления сильных

и слабых качеств. Модель данных предоставлена в соответствии с таблицами 15 и 16.

Таблица 15 – Успешно пройденные тренинги

Field	Type	Nullable	Default
personal_information_id	uuid	false	
training_id	uuid	false	

Таблица 16 – Проваленные тренинги

Field	Type	Nullable	Default
personal_information_id	uuid	false	
training_id	uuid	false	

Обе таблицы ссылаются на карту персональной информации пользователя и на идентификатор тренинга, предоставляемый следующим сервисом.

3.5.4 Тренинговая система

Тренинговый сервис представляет из себя систему – отвечающую за проведение тренингов, сохранение информации о собранных курсах и может содержать большое количество данных. В соответствии с таблицей 17 описана физическая модель тренинга.

Таблица 17 – Тренинг

Field	Type	Nullable	Default	Description
id	uuid	false		
skill_name	varchar(32)	false		
additional_info	json-data	true		Данные тренинга, для изучения и прохождения теста

Колонка дополнительной информации имеет тип json и может содержать множество данных в указанном формате. Это позволит не создавать большое количество таблиц и данные тренинга вытаскивать прямо из таблицы и с помощью ORM[10] преобразовать это в читаемый объект.

Следующая таблица содержит в себе информацию о курсе, это несколько тренингов, содержащих одну тему. Курс содержит в себе связи на назначенных пользователей и на список тренингов. Физическая модель данных предоставлена в соответствии с таблицами 18, 19, 20.

Таблица 18 – Курс

Field	Type	Nullable	Default
id	uuid	false	
name	varchar(32)	false	
training_manager	uuid	false	

Таблица 19 – Курс к пользователям

Field	Type	Nullable	Default
course_id	uuid	false	
user_id	uuid	false	

Таблица 20 – Курс к тренингам

Field	Type	Nullable	Default
course_id	uuid	false	
training_id	uuid	false	

[10] **ORM** (англ. Object-Relational Mapping, рус. объектно-реляционное отображение, или преобразование) — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Существуют как проприетарные, так и свободные реализации этой технологии.

Данные таблицы содержат связь идентификаторов от курса к пользователям и от курса к тренингам, соответственно проецируя связь один ко многим. Описанная в данном пункте модель данных может отличаться от финальной версии в связи с решением проблем на этапе разработки.

Итогом данной главы является разработанная физическая модель. Она как нельзя лучше подходит к выбранной базе данных и должна поддерживать целостность данных и работать с технологиями виртуализации. Такая распределенность физической модели поможет более наглядно отобразить преимущества технологий виртуализации. Все необходимые данные для разработки ИС описаны, поэтому в следующем пункте речь пойдет непосредственно о разработке программного обеспечения.

3.6 Разработка программного обеспечения ИС

Разработка ПО основанная на технологиях виртуализации в основном состоит из различного рода конфигураций, так как управление памятью и разделение приложения на несколько сервисов трудоемкий процесс, это обуславливается наличием конфигураций для каждого отдельного сервиса и всех сервисов в целом, для достижения их взаимной активности. На этапах разработки будет использоваться технология docker, позволяющая добиться контейнеризации сервисов.

Процесс разработки разделяется на:

– Этап подготовки к разработке;

– Разработку отдельного сервиса;

– Разработку конфигурации отдельного сервиса;

– Разработку конфигурации docker-compose для объединения сервисов;

– Проверку работоспособности написанных конфигураций;

Далее речь пойдет о разработке отдельного сервиса.

3.6.1 Этап разработки отдельного сервиса

Этап подготовки - включает в себя создание репозиториев в удаленном хранилище gitlab, для удобного процесса разработки и удаленного сохранения данных. Далее следует разработка программного кода для каждого репозитория и его сохранение в удаленном хранилище, после этого разрабатывается конфигурация для внедрения приложения в образ и последним этапом осуществляется разработка общего docker-compose файла для объединения всех приложений в единую систему, которую контролирует пользовательский интерфейс доступный пользователю из вне. Основным источником команд для пользовательского интерфейса является пользователь, на которого и ориентирована данная система.

Прежде всего, была создана группа, объединяющая все созданные репозитории для упрощенного управления сервисами. В соответствии с рисунком 13 изображен вид сервисов в группе.

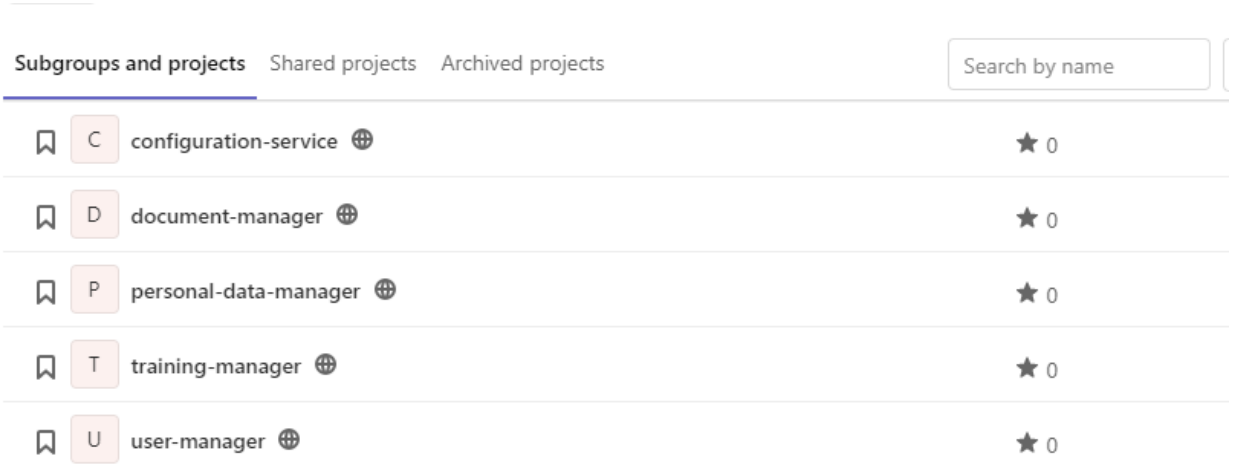


Рисунок 13 – Группировка приложений в сервисе gitlab

В соответствии с паттерном проектирования MVC[11] программный код каждого сервиса разделен по разным слоям для удобства и навигации между классами. Слой вида представляет собой модели, реализующие посредством ORM сопоставление программных объектов с таблицами базы данных. Код модели пользователь предоставлен в листинге 1.

Листинг 1. Модель вида пользователь в таблице пользователей

```
package com.wirax.vkr.usermanager.model;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import javax.persistence.*;
import java.util.UUID;

@Data
@Entity
@NoArgsConstructor
@AllArgsConstructor
public class User {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private UUID id;
   private String firstName;
   private String lastName;
   @OneToOne(cascade = CascadeType.ALL)
   private Address address;
   private String phone;
   private String additionalData;
}
```

Где мы видим поля идентификатора, имени, фамилии, адреса, телефона и дополнительных данных, которые соприкасаются с таблицей пользователей в базе данных. Поле адреса реализует связь OneToOne, которая описана в таблице логической модели данных. Аннотации данных, пустого конструктора и конструктора всех полей реализуются на этапе компиляции, т.е. подставляются в проект уже после компиляции и предоставляются технологией “Lombok”.

Слой представления является слоем передачи данных, данный слой еще называется “DTO” – “Data Transfer Object”, объект передачи данных. Эти

[11] **Model-View-Controller** (**MVC**, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

объекты отвечают за предоставление данных между сервисами и пользовательским интерфейсом, сделано это для разделения публичной информации от внутренней информации базы данных. Пример такого объекта предоставлен в листинге 2.

Листинг 2 – Объект передачи данных, пользователь

```
package com.wirax.vkr.usermanager.model.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.UUID;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class UserDto {
    private UUID id;
    private String firstName;
    private String lastName;
    private AddressDto address;
    private String phone;
    private String additionalData;
}
```

Где продемонстрированы отраженные данные из таблицы «Пользователь», так как в данном случае нет необходимости скрывать какие-либо данные от внешней среды.

Перед тем, как приступить к разбору слоя – контроллер, необходимо затронуть еще несколько важных вещей, связывающих слои модели и представления. Объект “Mapper” реализованный для упрощения конвертации объекта из слоя модели в слой представления и обратно. Mapper позволяет построить новый объект слоя, на основе объекта модели, при этом отбрасывая ту информацию, которой нет на слое представления. В проекте я использовал технологию “MapStruct” реализованная, как плагин для системы maven. Пример использования маппера предоставлен в листинге 3.

Листинг 3 – Конвертация объектов слоев модели и представления

```
package com.wirax.vkr.usermanager.mapper;

import com.wirax.vkr.usermanager.model.User;
import com.wirax.vkr.usermanager.model.dto.UserDto;
import org.mapstruct.Mapper;

@Mapper
public interface UserMapper {
   User toEntity(UserDto dto);

   UserDto toDto(User entity);
}


package com.wirax.vkr.usermanager.mapper;

import com.wirax.vkr.usermanager.model.Address;
import com.wirax.vkr.usermanager.model.User;
import com.wirax.vkr.usermanager.model.dto.AddressDto;
import com.wirax.vkr.usermanager.model.dto.UserDto;
import javax.annotation.processing.Generated;

@Generated(
   value = "org.mapstruct.ap.MappingProcessor",
   date = "2021-09-26T16:54:53+0400",
   comments = "version: 1.4.2.Final, compiler: javac, environment: Java 15 (Oracle Corporation)"
)
public class UserMapperImpl implements UserMapper {

   @Override
   public User toEntity(UserDto dto) {
      if ( dto == null ) {
         return null;
      }

      User user = new User();

      user.setId( dto.getId() );
      user.setFirstName( dto.getFirstName() );
      user.setLastName( dto.getLastName() );
      user.setAddress( addressDtoToAddress( dto.getAddress() ) );
      user.setPhone( dto.getPhone() );
      user.setAdditionalData( dto.getAdditionalData() );

      return user;
   }

   @Override
   public UserDto toDto(User entity) {
      if ( entity == null ) {
         return null;
      }

      UserDto userDto = new UserDto();

      userDto.setId( entity.getId() );
      userDto.setFirstName( entity.getFirstName() );
      userDto.setLastName( entity.getLastName() );
      userDto.setAddress( addressToAddressDto( entity.getAddress() ) );
      userDto.setPhone( entity.getPhone() );
```

```
        userDto.setAdditionalData( entity.getAdditionalData() );

        return userDto;
    }

    protected Address addressDtoToAddress(AddressDto addressDto) {
        if ( addressDto == null ) {
            return null;
        }

        Address address = new Address();

        address.setId( addressDto.getId() );
        address.setUser( addressDto.getUser() );
        address.setCity( addressDto.getCity() );
        address.setStreet( addressDto.getStreet() );
        address.setCountry( addressDto.getCountry() );

        return address;
    }

    protected AddressDto addressToAddressDto(Address address) {
        if ( address == null ) {
            return null;
        }

        AddressDto addressDto = new AddressDto();

        addressDto.setId( address.getId() );
        addressDto.setUser( address.getUser() );
        addressDto.setCity( address.getCity() );
        addressDto.setStreet( address.getStreet() );
        addressDto.setCountry( address.getCountry() );

        return addressDto;
    }
}
```

На листинге изображен интерфейс, помеченный аннотацией Mapper и являющийся указанием, к реализации указанных методов. На этапе компиляции системы сборки maven, генерируется код, реализующий описанные интерфейсы. Это решает проблему доступа объектов в информационной системе.

В системе еще существует потребность доступа к базе данных, но отсутствует желание к ручному управлению данными через запросы. В этом нам помогает реализация ORM Hibernate под названием JPA. Данная технология предоставляет удобное управление базой данных, не используя синтаксис языка SQL и прибегая к нему только в самых изощренных запросах. Для реализации доступа к базе данных необходимо разработать класс Repository, который и будет являться слоем доступа к базе данных. Пример предоставлен в листинге 4.

Листинг 4 – Доступ к базе данных посредством ORM, пользователь

```
package com.wirax.vkr.usermanager.repository;

import com.wirax.vkr.usermanager.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.UUID;

@Repository
public interface UserRepository extends JpaRepository< User, UUID > {
}
```

Где JpaRepository это реализованная технология JPA, а данные обобщения это, тип первичного ключа таблицы и его объект из слоя модели. Репозиторий обладает стандартной функциональностью для любого объекта, список доступных методов предоставлен в листинге 5.

Листинг 5 – Список доступных методов для модели

```
package org.springframework.data.jpa.repository;

import java.util.List;
import org.springframework.data.domain.Example;
import org.springframework.data.domain.Sort;
import org.springframework.data.repository.NoRepositoryBean;
import org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.data.repository.query.QueryByExampleExecutor;

@NoRepositoryBean
public interface JpaRepository<T, ID> extends PagingAndSortingRepository<T, ID>, QueryByExampleExecutor<T>
{
   List<T> findAll();
   List<T> findAll(Sort sort);
   List<T> findAllById(Iterable<ID> ids);
   <S extends T> List<S> saveAll(Iterable<S> entities);
   void flush();
   <S extends T> S saveAndFlush(S entity);
   <S extends T> List<S> saveAllAndFlush(Iterable<S> entities);

   /** @deprecated */
   @Deprecated
   default void deleteInBatch(Iterable<T> entities) {
      this.deleteAllInBatch(entities);
   }

   void deleteAllInBatch(Iterable<T> entities);
   void deleteAllByIdInBatch(Iterable<ID> ids);
   void deleteAllInBatch();

   /** @deprecated */
   @Deprecated
```

```
    T getOne(ID id);
    T getById(ID id);
    <S extends T> List<S> findAll(Example<S> example);
    <S extends T> List<S> findAll(Example<S> example, Sort sort);
}
```

Данные методы доступны для использования через модифицированный класс Repository, продемонстрированный ранее. Метод save будет использовать поиск объекта с первичным ключем в базе данных, если объект будет найден, то его данные будут обновлены, если нет, то создан новый объект и присвоен новый идентификатор. Указанных методов вполне хватит для разработки полноценного сервиса и перечисленные методы сильно упростят задачу разработчика по имплементации бизнес-требований.

Настало время поговорить про слой контроллера – данный слой отвечает за предоставление методов Rest API, для передачи данных между сервисами, либо пользовательскому интерфейсу по запросу (см. приложение К).

Где существует несколько аннотаций, таких как @RestController – означает что в данном классе расположены http методы, для передачи информации и внутренний параметр отображает путь к ресурсу, в данном случае полный путь это “/user-manager/api/v1/user”. Аннотация @Autowired – обладает способностью внедрения зависимостей, после процедуры поиска по пакетам в разрабатываемом приложении, она создаст объект класса и передаст его в агрумент конструктора, который инициализирует финальную переменную пользовательского сервиса. Список аннотаций маппинга @PostMapping, @GetMapping, @DeleteMapping, @PutMapping оповещает о том, что метод, над которым расположена аннотация является методом предоставления данных. Таким образом мы получаем методы Rest API:

- GET (user-manager/api/v1/user),
- POST (user-manager/api/v1/user),
- DELETE (user-manager/api/v1/user?id=uuid),
- GET (user-manager/api/v1/user).

Методы доступны из вне и используются сторонними сервисами для передачи данных и управления доступной функциональностью. Стоит заметить, что все описанные сервисы для всех моделей имеют схожую имплементацию контроллера, сервисного слоя и слоя данных.

Итогом данной разработки являются готовые к установке сервисы, которые имеют функционал взаимодействия и подключение к базе данных. Далее необходимо будет разработать конфигурацию для обертки сервиса в образ, чтобы впоследствии с ним можно было взаимодействовать внутри контейнера. Таким образом следующие разделы расскажут о конфигурации приложения с использованием виртуализации-контейнеризации и технологии docker.

Главное преимущество технологий виртуализации в контексте приложения – это разделение сервисов на образы, что называется отдельным термином – контейнеризация. Контейнеризация позволяет использовать разные языки и технологии в разных сервисах и написав общую конфигурацию – объединить все сервисы в полноценную систему, тем самым реализуя архитектуру приложения с помощью технологий виртуализации. Данные преимущества позволяет реализовать программный продукт – docker. Функционал данного ПО позволяет разделять большую систему на множество подсистем, а впоследствии объединять их с помощью общей конфигурации.

Для сборки образа каждого сервиса понадобится файл конфигурации – dockerfile. Данный файл позволит собрать образ приложения, которое будет готово для запуска с помощью специальной команды, описанной в файле. Для каждого сервиса может быть использован свой язык программирования, компилятор, интерпретатор и т.д. Файл конфигурации для сервиса с названием - менеджер пользователей, предоставлен в соответствии с листингом 6.

Листинг 6 – Dockerfile конфигурация для сервиса user-manager

```
FROM adoptopenjdk:11-jre-hotspot
ARG JAR_FILE=target/*.jar
```

```
COPY ${JAR_FILE} application.jar
ENTRYPOINT ["java","-jar","/application.jar"]
```

Где FROM это образ-ресурс, на основе которого будет запущено приложение, в данном случае это язык Java 11 версии и соответственно его инструмент компиляции и запуска openjdk 11, ARG JAR_FILE – это объявление константы и запись в него строки, которая указывает на запускаемый файл приложения. Функция COPY копирует файл по записанному пути в константу в корневой каталог проекта и изменяет его имя на application.jar. ENTRYPOINT – это основная команда запуска приложения. После того как образ будет собран и помещен в удаленное хранилище gitlab, посредством конфигурации gitlab-ci.yml. Сохраненные образы можно будет запустить командой docker-compose или docker run. Запуск можно совершить, зная путь до образа и имея доступ с машины-клиента, которая пытается запустить сохраненный образ.

Данный подраздел позволил создать образ из написанного приложения и поместить его в удаленное хранилище, для последующего запуска. Следует заметить, что все остальные сервисы собраны и помещены в хранилище идентичным образом. Далее речь пойдет о способе запуска и объединения сервисов в полноценное приложение.

3.6.2 Этап разработки общей конфигурации

Способ объединения всех разработанных и собранных сервисов заключается в написании общей конфигурации, где будут описаны особенности запуска, зависимые сервисы, доступные порты и т.д. Разработанный файл должен поддержать сборку из удаленного хранилища, так как перемещать сохраненные удаленно образы на локальную машину неудобно. Разработанный файл конфигурации docker-compose (см. приложение Л).

Где указана версия конфигурационного файла и перечислены следующие сервисы:

- База данных,
- Менеджер пользователей,

– Менеджер документов,
– Менеджер персональных данных,
– Менеджер тренингов.

Стоит отметить несколько ключевых моментов. В конфигурацию включен сервис базы данных из открытого хранилища docker-hub, при запуске данного файла образ будет скачен из сети-интернет и запущен, с помощью, описанной разработчиками конфигурации. Каждый сервис имеет ссылку на образ в удаленном хранилище gitlab, образ сервиса попадает туда после каждого коммита кодовой базы в головную ветку проекта (см. приложение М). Кроме того, у сервисов указаны порты из внутреннего сервиса во вне, это означает то, что к сервисам можно обращаться, даже не находясь в контейнере, что позволяет разработчику обратиться к сервисам без пользовательского интерфейса с целью отладки или по другим нуждам. Пользовательский интерфейс не присутствует в списке сервисов, так как не является важным для него, он может быть запущен другой вычислительной машиной и обращаться к ним через открытые порты основной машины в открытые порты контейнера, который запущен на основной вычислительной машине (см. приложение Н).

Итогом данного подпункта и параграфа в целом является готовый программный продукт, который может оперировать различными данными, подключенный к базе данных и способный обрабатывать запросы пользователя через пользовательский интерфейс. Функциональность разработанного приложения описана в следующем параграфе данной работы.

3.7 Описание разработанной функциональности ИС

Функциональность приложения разделяется на несколько слоев, среди которых выделяется:

– Пользовательский интерфейс,
– Серверное приложение.

Пользовательский интерфейс предназначен для предоставления данных пользователю в графическом формате, чтобы информация была доступной, краткой и содержательной. Он так же является сервисом и отражен в списке образов, готовых к установке на виртуальную машину.

Средства хранения информации принимают информацию от пользовательского интерфейса, занимаются обработкой данных и выдают пользователю результат запрашиваемой операции.

Общий технологический процесс работы с информацией предоставлен на рисунке 14.

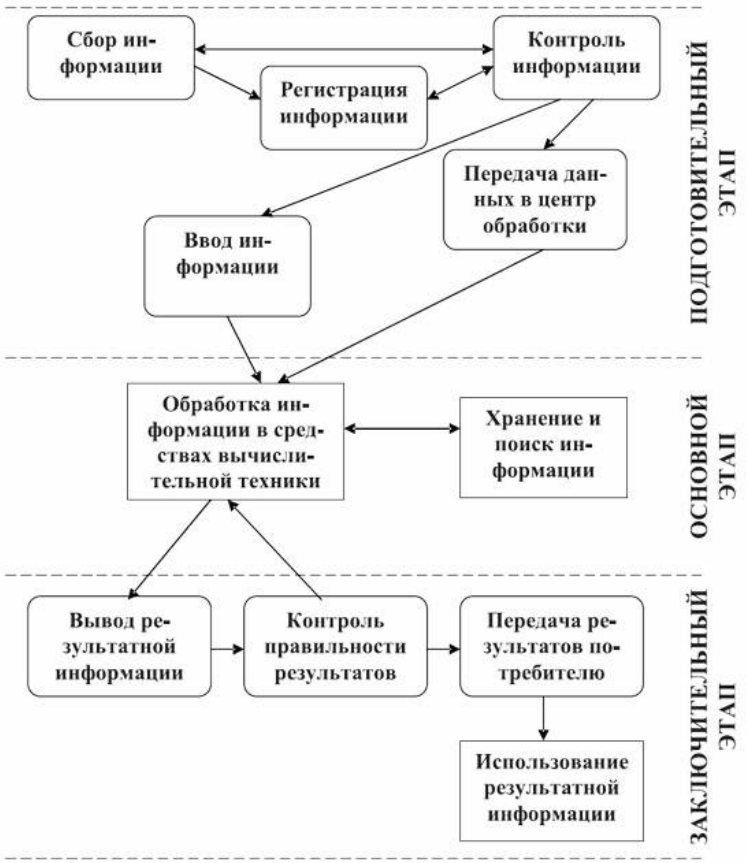


Рисунок 14 – Технологический процесс обработки информации

Подготовительный этап - это работа пользователя и пользовательского интерфейса, далее информация попадает в основной этап, один из сервисов серверного приложения, подключенный к базе данных. Сохранение и манипуляция данных происходит так же на данном этапе. В заключительном

этапе участвуют серверные и клиентские приложения совместно, для контроля и передачи информации в графически оформленном виде.

Сервисы, ориентированные на работу с данными, предоставляют интерфейс в виде методов Rest API, для взаимодействия с клиентской частью. Пример запросов приведет в соответствии с рисунком 15.

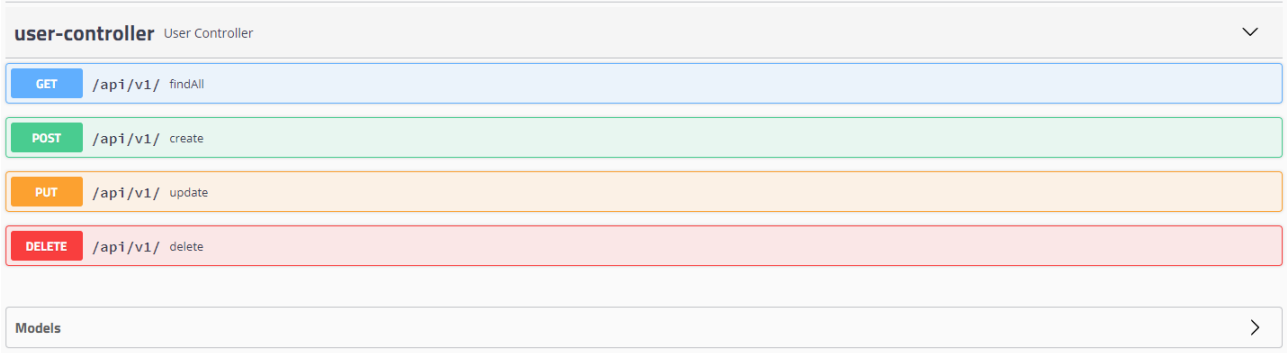


Рисунок 15 – Методы взаимодействия для модели данных пользователя

Стандартные методы взаимодействия описаны для каждой модели данных и их хватает для имплементации базового функционала приложения, но существуют и менее банальные алгоритмы в которых участвуют несколько пользователей и несколько сервисов сразу. Одним из таких алгоритмов является получение токена авторизации и регистрация нового сотрудника в системе с помощью вышестоящего руководителя, технического менеджера. Процедура регистрации в системе для нового пользователя (см. приложение О) является примером разработанной функциональности.

В данном алгоритме участвуют несколько сервисов помеченных как user_interface, user_manager, document_manager. Данные сервисы разработаны и присутствуют на диаграмме доступных сервисов. Несмотря на сложность алгоритма, он состоит из комбинации действий сотрудников и простых API Rest методов, не переходящих за рамки CRUD интерфейса.

Описанный параграф демонстрирует функциональность системы и его техническую сторону реализации, следующим шагом является увеличение показателя качества с помощью методов тестирования ПО.

3.8 Тестирование программного продукта

3.8.1 Выбор технологий тестирования программного продукта

Технологии тестирования, выбранные для разрабатываемого проекта, должны быть подходящими для библиотеки Spring. Основной критерий выбора будет исходить из технологий, которые задействованы в проекте. Самое простое и подходящее решение это JUnit[12]. Библиотека подходит для тестирования проектов на других языках и является самой распространенной и простой. Кроме перечисленных достоинств, библиотека базируется на средствах виртуализации и разработана для Java Virtual Machine. Что означает её высокую совместимость для запуска в контейнере.

Для полноценного использования модульного тестирования необходимо средство, позволяющее обработать компоненты Spring Boot и создать условия для исполнения программного кода. Дело в том, что механизм внедрения зависимостей будет работать только в том случае, если контекст приложения будет запущен, чего не происходит при модульном тестировании, именно поэтому необходим механизм обработки вызовов зависимостей, для избежание ситуаций, когда необходимая зависимость будет пустой. В этом случае используется библиотека Mockito[13], она позволяет обрабатывать вызовы несуществующих зависимостей, что упрощает процесс модульного тестирования.

Исключая данный пункт нам еще понадобится средство запуска приложения – Spring Boot и средство сборки приложения Maven. А позднее тестирование будет проходить при процессе непрерывной интеграции, что

[12] JUnit — библиотека для модульного тестирования программного обеспечения на языке Java.

[13] Mockito - платформа для тестирования с открытым исходным кодом для Java , выпущенного под лицензией MIT . Платформа позволяет создавать тестовые двойные объекты в автоматических модульных тестах для целей разработки через тестирование или разработка на основе поведения.

повысит его качество и наглядно отобразит статус тестирования для каждой новой ветки разработки.

3.8.2. Описание принципа тестирования компонентов

Выделяют несколько фаз тестирования ПО, разделяются по видам.

Интеграционное тестирование - одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе. Обычно интеграционное тестирование проводится после модульного тестирования и предшествует системному тестированию. Обычно интеграционное тестирование применяется к отдельному API Rest методу и тестируются все вызовы всех компонентов в группе, при этом фреймворк запуска приложения должен быть активен, что потребляет ресурсы и занимает больше времени, чем следующая фаза тестирования.

Модульное тестирование - процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки. Данная фаза позволяет не нагружать систему поднятием фреймворка и не потребляет большое количество ресурсов. При этом, все вызовы сторонних компонент, репозиториев и т.д. Должны подвергнуться обработке с помощью Mockito.

Системное тестирование - это тестирование программного обеспечения, выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям. Системное тестирование относится к методам тестирования чёрного ящика, и, тем самым, не требует знаний о внутреннем устройстве системы.

В разрабатываемом проекте используется модульное тестирование, так как разбиение сервисов уменьшает количество логики в нем и соответственно уменьшается потребность в интеграционных и системных тестах. В рамках настоящего продуктового продукта, все виды тестов были-бы оправданны, вопрос только в том, сколько ресурсов можно пожертвовать на написание, поддержание и выполнение тестов. Так как выбрана определенная фаза

тестирования и перечислены подходящие технологии, в данном разделе будет рассмотрен сам программный код тестов.

3.8.3. Описание программного кода тестирования ИС

Модульное тестирование ИС происходит над каждым сервисом приложения, что обеспечивает его качество, так как основные алгоритмы любого приложения исполняются в слое сервиса. Сервис связывает контроллер с Rest API и базу данных, что является самым эффективным местом для тестирования логики приложения. Класс тестов для сервиса пользователей является хорошим примером (см. приложение П).

Названия методов тестирования, используется логика разделения – ожидаемый результат, действие, вызов. Пример вернуть ошибку, когда идентификатор операции обновления, не идентичен.

Аннотация “MockitoJUnitRunner” – позволяет показать библиотеке Spring, что в данном классе будут использованы не натуральные классы, данные классы используются как конфигурируемые подмены оригинальных.

Тестируемый сервис помечен аннотацией внедрения конфигурируемых классов, чтобы при тесте заменить оригинальные классы, конфигурируемыми.

Метод конфигурации – используется для настройки заменяемого метода и настроен так, чтобы при вызове зависимостей возвращался необходимый для тестов результат.

Методы тестирования – используют методы сравнения, в случае их неудачи возвращают ошибку и тест помечается красным цветом. Что не дает разработчиком шанса пропустить данный код в основную ветку приложения.

Стоит заметить, все разработанные сервисы, использующие базу данных покрыты идентичными методами тестирования и в случае если после разработки нового функционала, старый будет недоступен средства непрерывной интеграции заметят это, а соответственно это заметит разработчик приложения, что является очень удобной опцией и позволяет устранять большинство ошибок программного кода, до выдачи заказчику. Подход разработки с помощью микро-

сервисной архитектуры и технологий виртуализации облегчает сервисную логику приложения и уменьшает её количество, что влияет на количество тестов, которые необходимо разработать и соответственно влечет за собой уменьшение времени разработки.

3.9 Обоснование экономической эффективности разработки ИС

Данный параграф очень важен, так как технологии виртуализации сильно влияют на экономическую составляющую проекта. Данный проект можно устанавливать на виртуальную машину, в связи с этим различия с базовым вариантом существующей информационной системы будут довольно большими.

Виртуализация позволяет конфигурировать потребление памяти и разделять физическую машину, в связи с этим, появилось много компаний, предоставляющих арендованную машину с количеством ресурсов, зависящих от абонентской платы. Такой тип применения виртуальных машин стали называть – ИТ-облако.

ИТ-облако - набор из определенного количества железных ресурсов, размещенных в инфраструктуре облачного провайдера. В составе облака могут быть сетевые устройства, серверы, дисковые накопители, каналы связи, разнообразное программное обеспечение. Основная задача облака — передача доступа к информации, находящейся в облачном хранилище, и предоставление виртуальных аппаратных мощностей. Разницы в эффективности между базовым вариантом системы на физической машине и базовым вариантом системы на виртуальной машине не будет. Так как используются железные ресурсы, расходуется электроэнергия. Это имеет смысл только в том случае, когда облачная машина становится невыгодна или не доступна. Сравнивать нужно между облачным подходом и физическим подходом, при этом использование виртуальных машин или нет не важно.

3.8.1 Расчет общей экономической эффективности за 3 года

В облачном хранилище будет использована виртуализация, так как без неё не удастся сконфигурировать подходящую вычислительную мощность для разных абонентов на одной физической машине. Таблица сравнения базового решения за 3 года с предлагаемым приведена в соответствии с таблицей 21.

Таблица 21 – Сравнение затрат базового и проектного решений

Сравнение затрат	Без виртуализации	С виртуализацией	Выгода руб/%	
Аппаратное обеспечение	6.508.736 руб.	1.562.034 руб.	4.946.702 руб.	76,0%
Подсистема хранения данных	0 руб.	1.079.762 руб.	(1.079.762 руб.)	0,0%
Сетевая инфраструктура	552.273 руб.	138.060 руб.	414.212 руб.	75,0%
Потребление энергии и тепло-отведение	1.143.723 руб.	294.395 руб.	849.327 руб.	74,3%
Рабочее пространство	552.460 руб.	102.732 руб.	449.729 руб.	81,4%
Настройка серверов (на этапе внедрения)	575.147 руб.	43.089 руб.	532.058 руб.	92,5%
Администрирование серверов	3.088.958 руб.	2.656.472 руб.	432.487 руб.	14,0%
Расходы, возникшие во время простоя серверов (косвенные расходы)	193.103 руб.	125.512 руб.	67.591 руб.	35,0%

Анализируя данные, предоставленные в таблице можно сделать несколько промежуточных выводов.

Аппаратное обеспечение не требуется к приобретению, но касается это только серверов, для мониторинга данных и конфигурации сети все равно необходимо минимальное обеспечение, это позволит сэкономить 76% бюджета.

Подсистема хранения данных не нужна в случае с приобретением сервера, но нужна в случае виртуализации, так как все характеристики удаленного сервера оплачиваются абонентом, соответственно выгоды в данном пункте нет.

Сетевая инфраструктура и потребление энергии почти целиком ложиться на плечи компании, у которой происходит аренда. Рабочие пространства для конфигурации сервера требуют лишь минимальных вложений. Настройка серверов ложиться на компанию, у которой происходит аренда.

Администрирование серверов идентично, в том и в другом случае необходимо администрировать ресурсы. Расходы во время простоя почти идентичны с базовым вариантом.

Далее необходимо посчитать необходимые инвестиции и экономию в результате тех или иных особенностей виртуальных машин и общие расходы. Общие затраты приведены в форме совокупной стоимости владения. TCO - TCO “Total cost of ownership”, или совокупная стоимость владения — общие расходы, которые возникают у компании из-за владения каким-либо активом, например, IT-инфраструктурой. Все расчеты приведены в соответствии с данными о поддержки приложения за три года в сравнении с подходом без виртуализации за 3 прошлых года. Стоит заметить, что данные о пунктах, которые зависят только от компании – владельца, такие как цена аренды, поддержки, доп. выгоды, акциях и т.д. могут быть разными. Результат сбора данных об экономии и общих расходах предоставлена в таблице 22.

Таблица 22 – Результат сбора данных об экономии и общих расходах

Необходимые инвестиции				
Аренда виртуальной машины в облаке	0 руб.	842.287 руб.	(842. 287 руб.)	0,0%
Поддержка виртуальной машины в облаке	0 руб.	483.868 руб.	(483 868 руб.)	0,0%
Дополнительные выгоды				
Экономия на способности масштабирования трафика на время профилактических работ			216.478 руб.	
Экономия на способности переноса данных клиента на время профилактических работ			577.244 руб.	
Экономия на динамическом распределении нагрузки в облаке			1.041.335 руб.	
Экономия за счет централизованного применения обновлений			1.059.140 руб.	
TCO - прямые расходы	12.421.297 руб.	4.308.502 руб.	8.112.795 руб.	65,3%
TCO - косвенные расходы	193.103 руб.	125.512 руб.	67.591 руб.	35,0%
Общее TCO (за 3 года)	12.614.400 руб.	4.434.014 руб.	8.180.386 руб.	64,8%

Приведенные данные в таблице отражают увеличение экономической эффективности в проектируемом варианте ИС. Это означает что технологии виртуализации способствуют получению дополнительной выгоды. Далее будет

совершен расчет раздельной экономической эффективности за 3 года, что даст более точное понимание об ожидаемой выгоде за каждый год.

3.8.2 Расчет раздельной экономической эффективности

В следующих расчетах будет участвовать только подход с технологиями виртуализации, так как ранее общая экономическая эффективность проектируемой системы была доказана. Раздельная выгода по годам для проекта с технологиями виртуализации приведена в соответствии с таблицей 23.

Таблица 23 – Ожидаемая дополнительная выгода от вложений в VM, разделенная на годовые отрезки

Ожидаемая основная выгода от VM	Год 1	Год 2	Год 3
Аппаратное обеспечение	1.494.475 руб.	1.643.925 руб.	1.808.302 руб.
Подсистема хранения данных	359.921 руб.	359.921 руб.	359.921 руб.
Сетевая инфраструктура	125.137 руб.	137.654 руб.	151.422 руб.
Потребление энергии и тепло-отведение	246.925 руб.	281.503 руб.	320.899 руб.
Рабочее пространство	135.870 руб.	149.451 руб.	164.408 руб.
Настройка серверов (на этапе внедрения)	154.082 руб.	176.299 руб.	201.677 руб.
Администрирование серверов	125.262 руб.	143.286 руб.	163.939 руб.
Расходы, возникшие во время простоя серверов	22.530 руб.	22.530 руб.	22.530 руб.
Выгода	1.944.360 руб.	2.194.727 руб.	2.473.257 руб.

Основная выгода на всех трех отрезках выглядит довольно внушительно и означает увеличение экономической эффективности от использования

технологий виртуализации в проекте. Кроме основной общей выгоды, стоит еще подсчитать косвенную выгоду, данные приведены в соответствии с таблицей 24.

Таблица 24 – Ожидаемая дополнительная выгода от вложений в VM, разделенная на годовые отрезки

Дополнительные выгоды	Год 1	Год 2	Год 3
Экономия на способности масштабирования трафика на время профилактических работ	62.928 руб.	71.753 руб.	81.797 руб.
Экономия на способности переноса данных клиента на время профилактических работ	167.819 руб.	191.319 руб.	218.105 руб.
Экономия на динамическом распределении нагрузки в облаке	314.610 руб.	346.058 руб.	380.667 руб.
Экономия за счет централизованного применения обновлений	307.913 руб.	351.034 руб.	400.193 руб.
Выгода	853.270 руб.	960.164 руб.	1.080.763 руб.

Косвенная выгода у данного проекта по сумме получается меньше, чем общая, но она тоже присутствует. Из данных приведенных в этом параграфе можно сделать вывод о том, что технологии виртуализации сокращают финансирование ПО, но увеличивают выгоду. Это особенно заметно с течением времени, третий год показывает максимальную выгоду от вложений, что наверняка не является пределом возможностей технологий виртуализации.

Выводы по главе

В данной главе была проделана работа по разработке программного продукта, который ориентирован на установку в облако. Описана

функциональность ИС и её различия по сравнению с базовым вариантом, выбраны технологии, позволяющие более наглядно раскрыть потенциал приложения с применением средств виртуализации. База данных для приложения подобрана в соответствии с его требованиями, а именно она позволяет агрегировать данные из разных сервисов и позволяет работу в нескольких потоках для баз данных каждого из сервисов, разработанного приложения. Описана функциональность информационной системы, разработана физическая модель и предоставлен листинг программного кода приложения.

После подведения итогов по разработке проведено полноценное модульное тестирование приложения и проведена оценка и обоснование экономической эффективности в связи с установкой приложения, которое разработано с применением технологий контейнеризации в ИТ-облако.

Решение об экономической эффективности оценено в связи с тем фактом, что установка программного продукта на физическую вычислительную машину заказчика будет почти идентична с текущим решением и это не увеличит экономическую эффективность проекта и не задействует все преимущества технологий виртуализации. В связи с чем рассчитана экономическая эффективность внедрения информационной системы на физическое оборудование заказчика и установкой в облако. На основе собранных данных удалость сделать вывод об экономической эффективности технологий виртуализации и подсчитать общую выгоду проектного варианта информационной системы, в отличии от базового варианта, которым пользуется заказчик.

Заключение

Основными результатами работы является разработка архитектуры приложения, на основе технологий виртуализации. После внедрения разработанного приложения наблюдалась положительная динамика. Затраты на поддержку и администрирование приложения были уменьшены, а пользователи системы отметили положительный опыт использования, в основном связанный с разделением приложения и способностью обращаться к разным возможностям приложения, не перегружая отдельные его модули. При этом, облачные вычисления позволили заказчику оптимизировать расходы предприятия, связанные с администрацией и поддержкой приложения. Что положительно повлияло статистику организации и отзыв сотрудников, после введения приложения, основанного на технологиях виртуализации.

Исходные данные были получены благодаря методу объектно-ориентированного анализа базового приложения заказчика. Эксперименты проведены на базе исследований в области облачных вычислений и получения информации от организаций, которые применили данный подход. Результатом исследования стало сравнение показателей базового варианта системы и проектного.

Актуальность работы была доказана, так как технологии виртуализации способствовали решению основных потребностей заказчика и оптимизации базового решения.

Достигнута цель данной работы, разработана архитектура приложения для уменьшения затрат и увеличения быстродействия системы, посредством применения технологий виртуализации.

В рамках работы были решены следующие задачи:

– Выполнен анализ предприятия и его базового решения;

– Выполнен анализ ИТ-технологий, подходящих под требования и специфику задач заказчика;

– Разработана архитектура на основе технологий виртуализации;
– Выполнен анализ работы проектной системы;
– Выполнено сравнение результатов базовой и проектной системы;

Результатом анализа предприятия стало выявление основных проблем текущего решения и требований заказчика. После анализа ИТ-технологий подходящих под решение потребностей выявлены технологии виртуализации, на которых построено проектное решение. С уверенностью можно сказать, что проделанная работа повлияла на экономическую эффективность заказчика. Результатом разработки архитектуры стало полноценное построение локальной и сетевой архитектуры проектного приложения. После этапа разработки и внедрения системы были подведены итоги работы. Итоги работы были сравнены с базовыми из чего можно сделать вывод о пользе решения, построенного на базе технологий виртуализации.

Список используемой литературы

1. ГОСТ 7.32-2001. Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления. Введ. 2001-04-09.

2. ГОСТ 7.1-2003. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание документа. Общие требования и правила составления. Введ. 2003-25-11.

3. Шелухин, О.И. Моделирование информационных систем [Электронный ресурс] : учеб. пособие для вузов / О.И. Шелухин. — 2-е изд., перераб. и доп. — М. : Горячая Линия — Телеком, 2012. — 516с.

4. Рудинский, И.Д. Технология проектирования автоматизированных систем обработки информации и управления [Электронный ресурс]: учеб. пособие / И.Д. Рудинский. – М.:Горячая линия – Телеком, 2011. – 304 с.

5. Балдин, К.В. Информационные системы в экономике [Электронный ресурс]: учебник / К.В. Балдин, В.Б. Уткин. — 7-е изд. — М. : Дашков и К°, 2012. - 395 с.

6. Грекул, В.И. Управление внедрением информационных систем [Текст]: учебник / В.И. Грекул, Г.Н. Денищенко, Н.Л. Коровкина. – М.: ИНТУИТ, 2017; Саратов: Вузовское образование, 2017. – 224 с.

7. Вдовин, В.М. Предметно-ориентированные экономические информационные системы [Электронный ресурс]: учебное пособие / В.М. Вдовин, Л.Е. Суркова, А.А. Шурупов. — 3-е изд. — М. : Дашков и К°, 2013. — 388 с.

8. Грекул, В.И. Методические основы управления ИТ-проектами [Текст]: учебник / В.И. Грекул, Н.Л. Коровкина, Ю.В. Куприянов. – М.: ИНТУИТ, 2017; Саратов: Вузовское образование, 2017. – 392 с.

9. Волкова, В.Н. Системный анализ информационных комплексов [Электронный ресурс] : учеб. пособие / В.Н. Волкова. – Изд. 2-е, стер. – С.-Пб.: Лань, 2016. – 336 с.

10. Буренин, С.Н. Web-программирование и базы данных [Электронный ресурс]: учеб. практикум / С.Н. Буренин - М. : Моск. гуманит. ун-т, 2014. - 120 с.

11. Гринберг, А.С. Информационные технологии управления [Электронный ресурс] : учеб. пособие / А.С. Гринберг, Н.Н. Горбачев, А.С. Бондаренко. – М.: ЮНИТИ-ДАНА, 2017. – 478 с.

12. Золотов, С.Ю. Проектирование информационных систем [Электронный ресурс] : учеб. пособие / С.Ю. Золотов. - Томск: Эль Контент, 2013. — 86 с.

13. Хаммер, Майкл. Быстрее, лучше, дешевле. Девять методов реинжиниринга бизнес-процессов [Электронный ресурс] / Майкл Хаммер, Хершман Лиза. – М.: Альпина Паблишер, 2016. – 352 с.

14. Карпова, И.П. Базы данных. Курс лекций и материалы для практических занятий : учеб. пособие для вузов / И. П. Карпова. — СПб. : Питер, 2013. — 240 с.

15. Реинжиниринг бизнес-процессов [Электронный ресурс] : учеб. пособие / А.О. Блинов [и др.] ; под ред. А.О. Блинова. — М. : ЮНИТИ-ДАНА, 2012. — 341 с.

16. Essential Business Process Modeling 1st Edition / ed. By Michael Havey. – Canada: O'Reilly Media. 2005. – 354 p.

17. Business Process Management: Concepts, Languages, Architectures / ed. By Weske Mathias. – Germany: Springer. 2019. – 417 p.

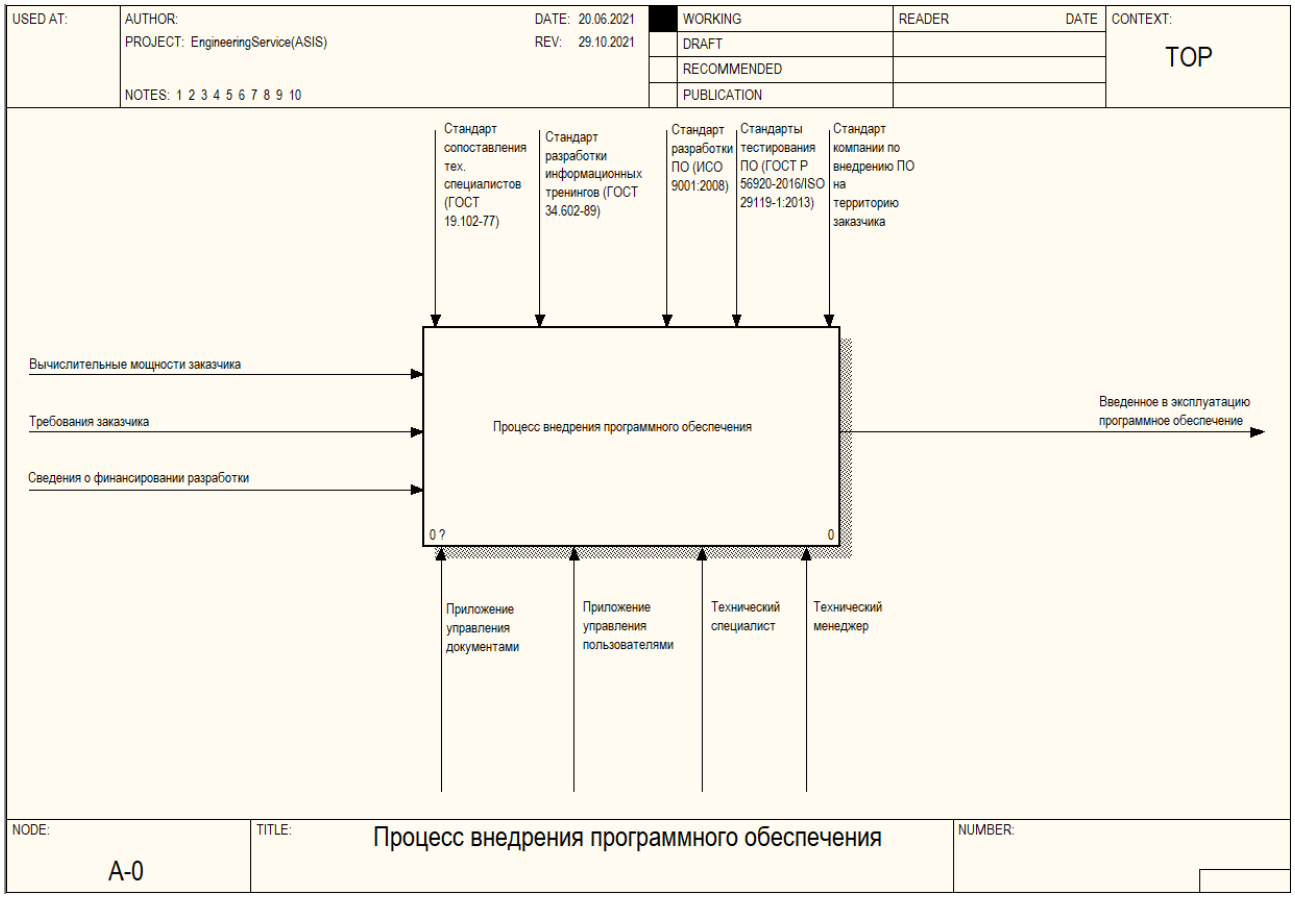
18. Clean Code: A Handbook of Agile Software Craftsmanship / ed. By Robert C. Martin. – USA: Prentice Hall. 2008. – 464 p.

19. Spring in Action / ed. By Walls Craig. – USA: Manning Publications. 2011 – 424 p.

20. Hibernate in Action (In Action series) / ed. By Christian Bauer, Gavin King. – Canada: O'Reilly Media. 2005. – 542 p.

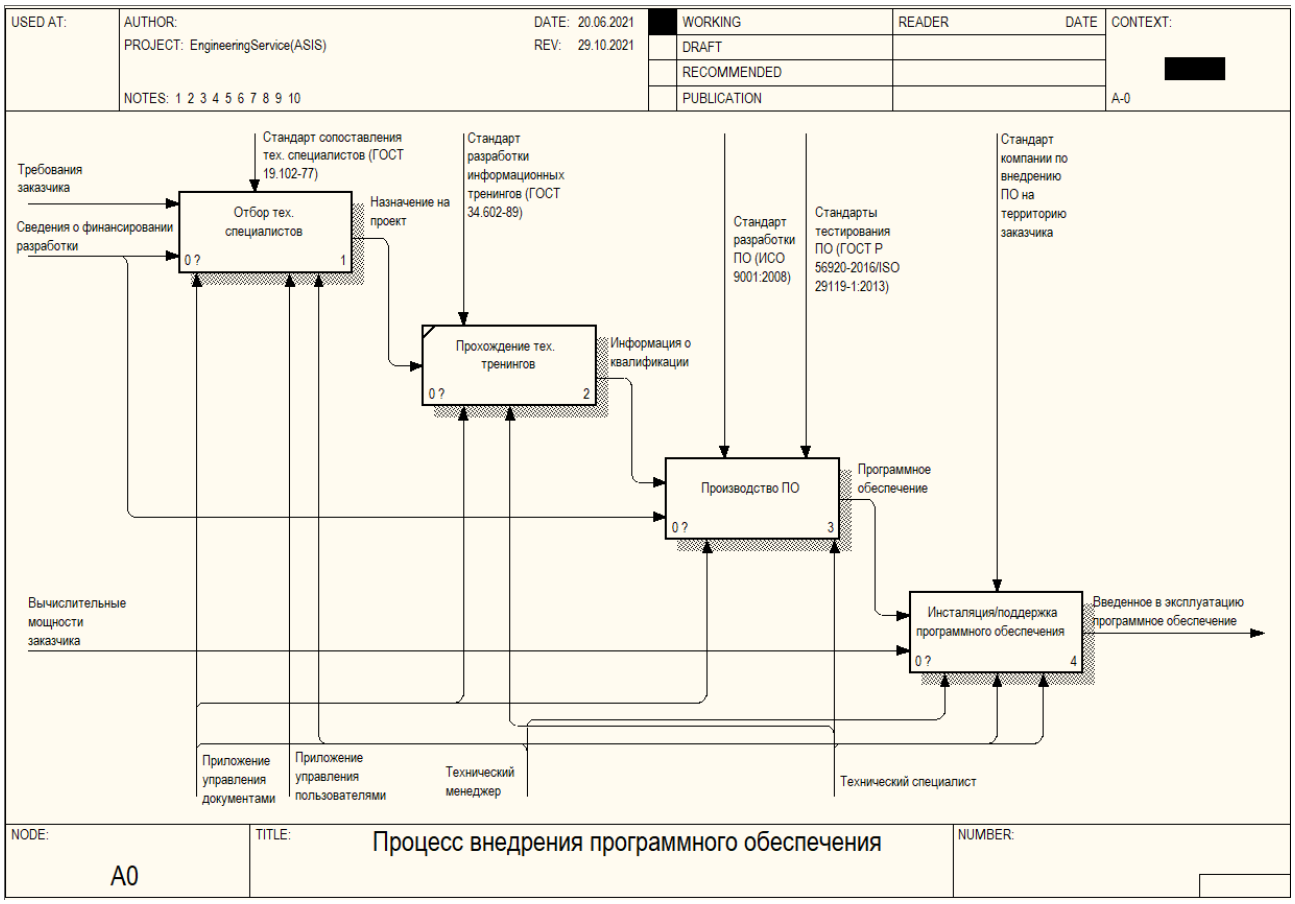
Приложение А

Деятельность организации по разработке ПО, «КАК ЕСТЬ»



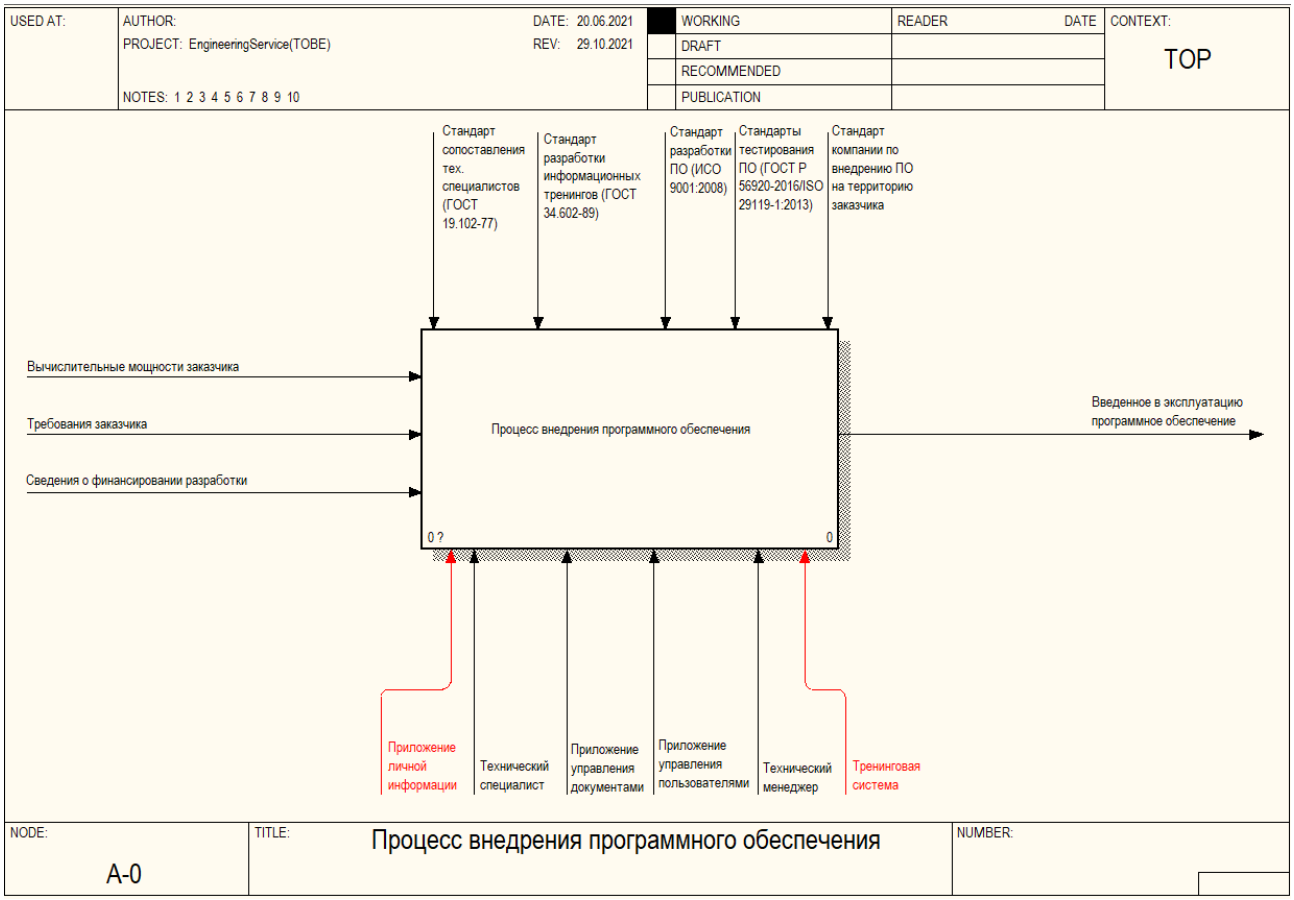
Приложение Б

Декомпозиция процесса «Деятельность организации по разработке ПО», «КАК ЕСТЬ»



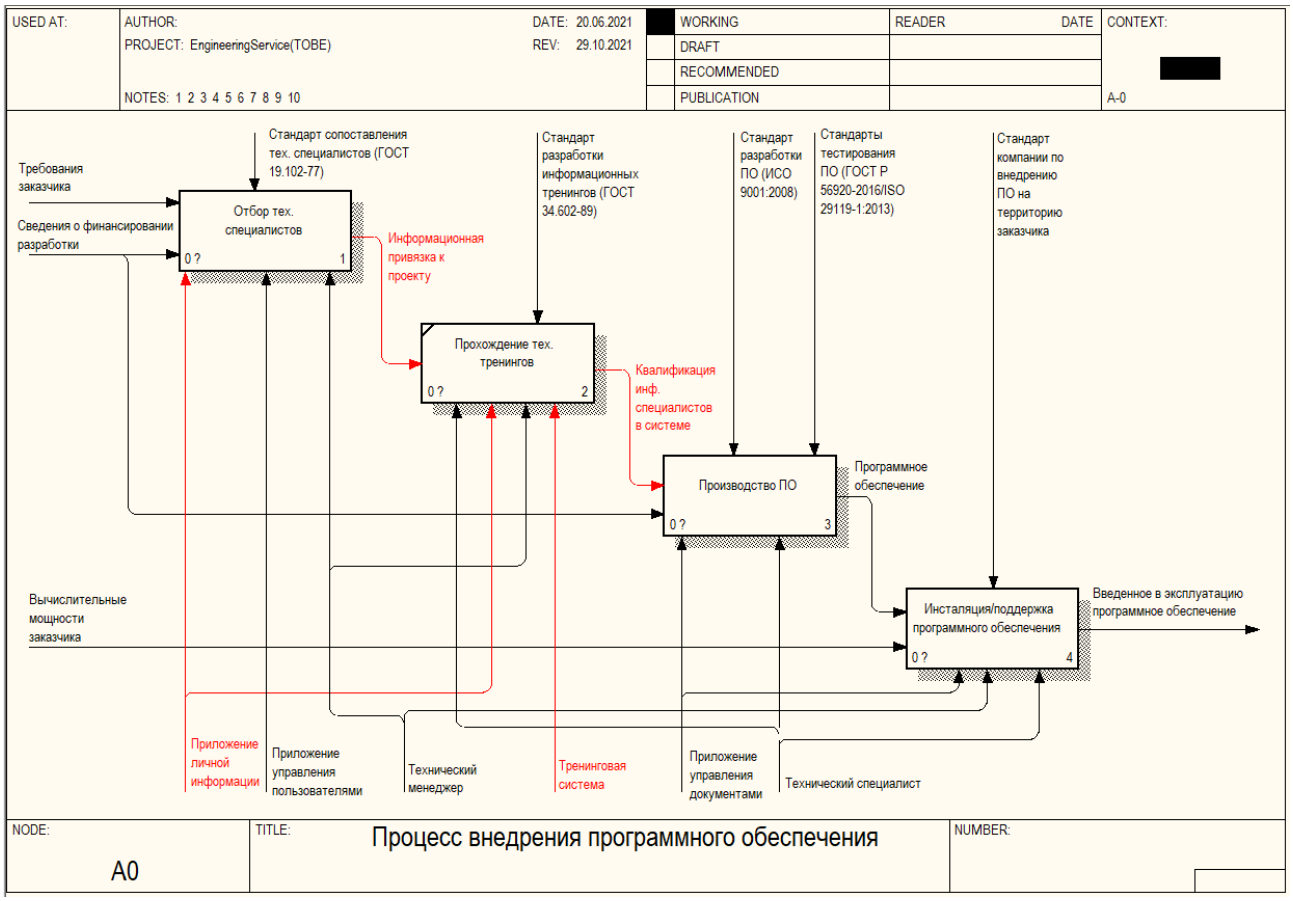
Приложение В

Деятельность организации по разработке ПО, «КАК ДОЛЖНО БЫТЬ»



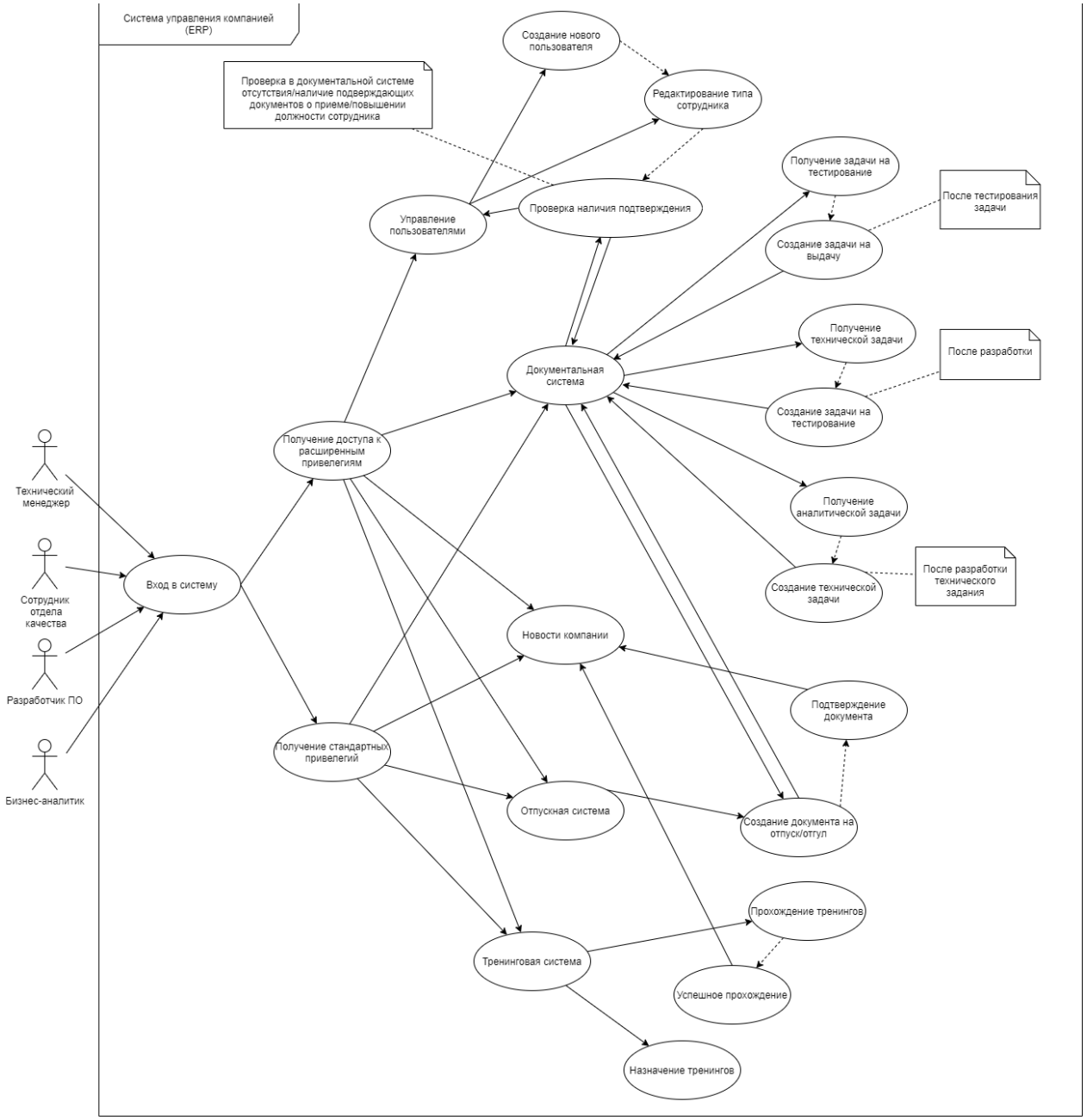
Приложение Г

Декомпозиция деятельности организации по разработке ПО, «КАК ДОЛЖНО БЫТЬ»



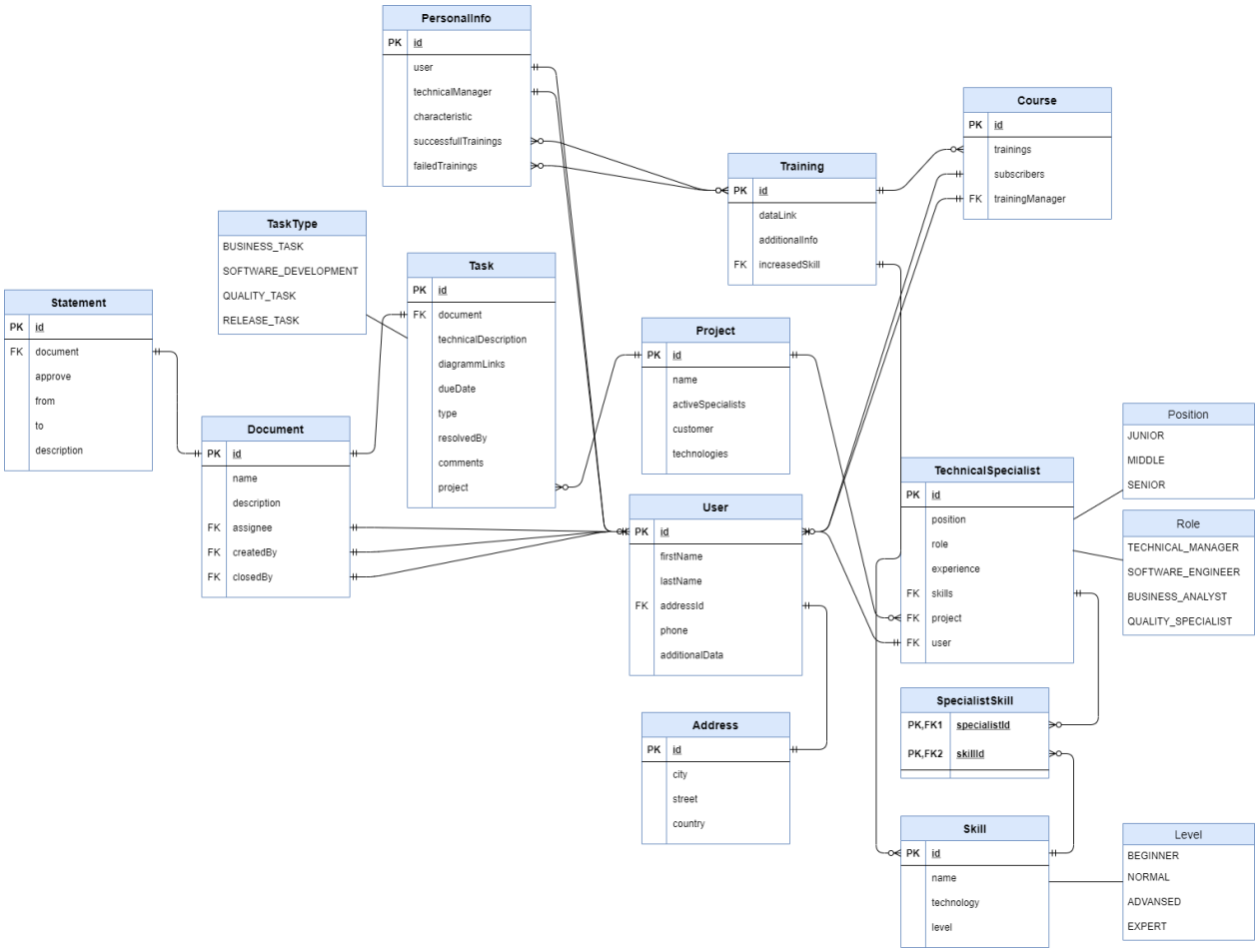
Приложение Д

Диаграмма вариантов использования (Use-case)



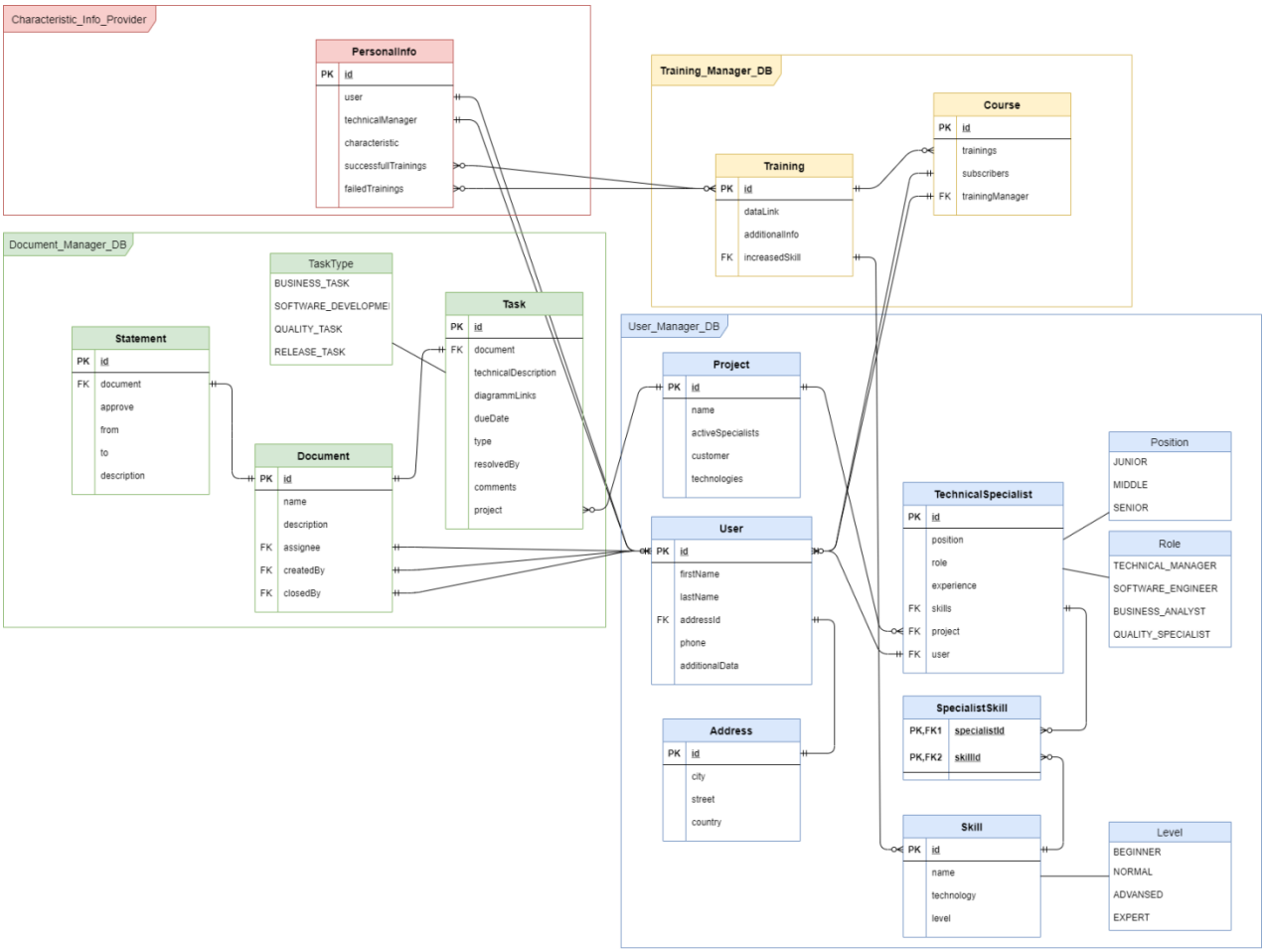
Приложение Е

Монолитное представление базы данных информационной системы “ERP”



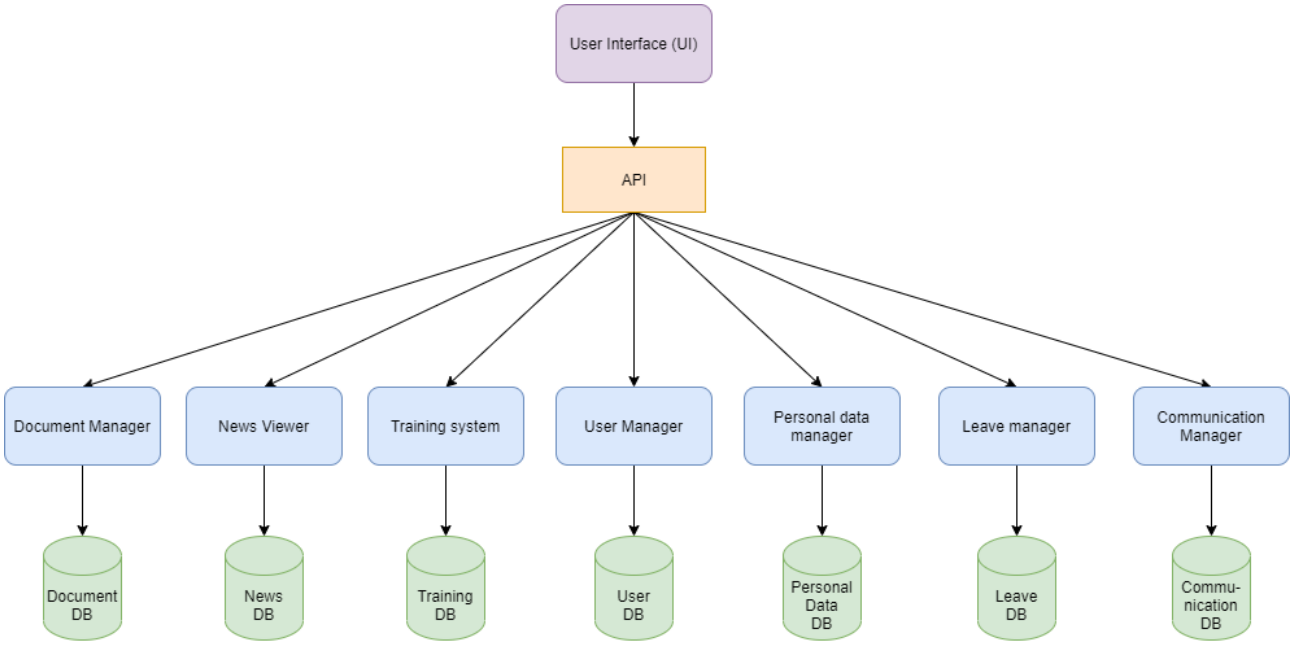
Приложение Ж

Разделенное представление базы данных информационной системы “ERP” (с использованием технологий виртуализации)



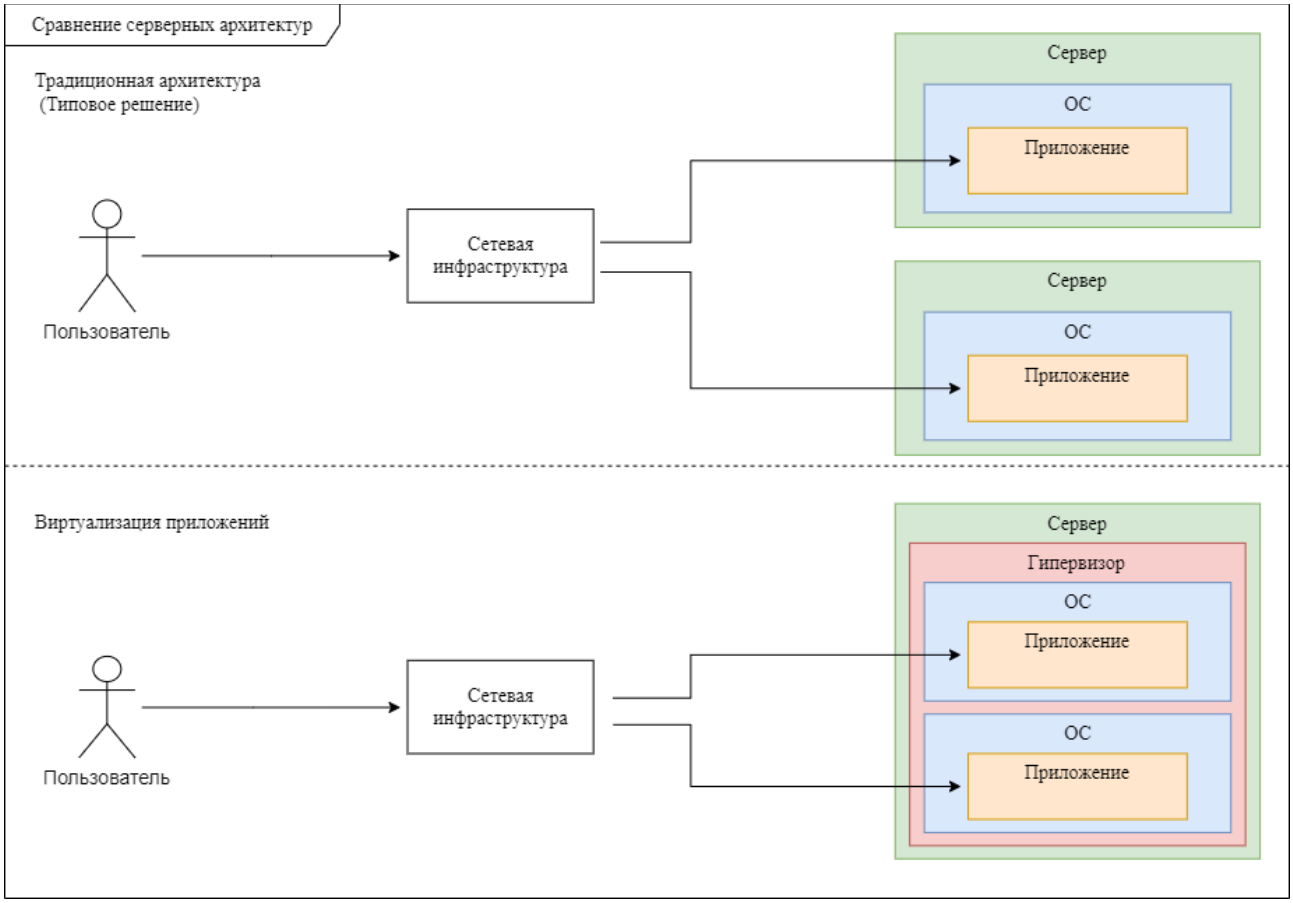
Приложение 3

Архитектура виртуализации приложения



Приложение И

Диаграмма архитектуры виртуализации



Приложение К

Листинг слоя контроллера, объект пользователь

```
package com.wirax.vkr.usermanager.controller;

import com.wirax.vkr.usermanager.mapper.UserMapper;
import com.wirax.vkr.usermanager.model.User;
import com.wirax.vkr.usermanager.model.dto.UserDto;
import com.wirax.vkr.usermanager.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.UUID;

@RestController("/user")
public class UserController {
    public final UserService userService;
    private UserMapper userMapper;

    @Autowired
    public UserController(UserService userService) {
        this.userService = userService;
    }

    @PostMapping
    public User create(@RequestBody UserDto user) {
        return userService.create(userMapper.toEntity(user));
    }

    @PutMapping
    public User update(@RequestParam UUID id, @RequestBody UserDto user) {
        return userService.update(id, userMapper.toEntity(user));
    }

    @GetMapping
    public List<User> findAll() {
        return userService.findAll();
    }

    @DeleteMapping
    public void delete(@RequestParam UUID id) {
        userService.delete(id);
    }
}
```

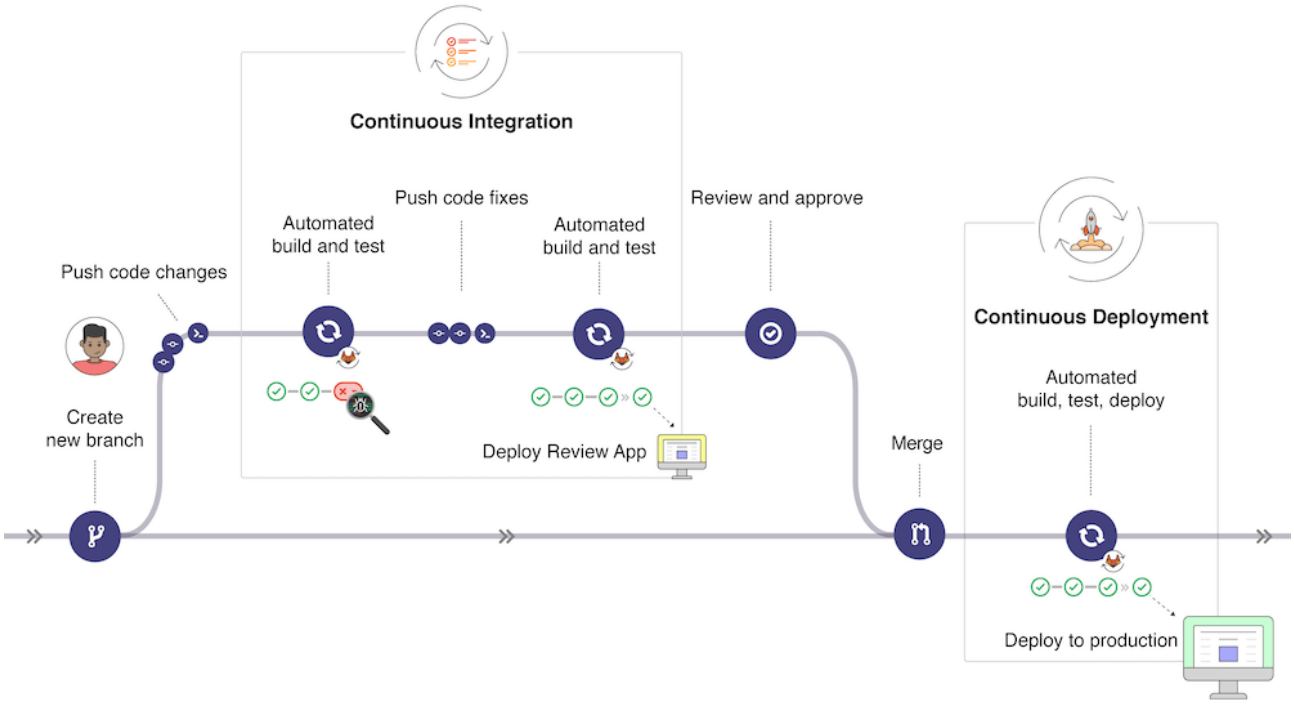
Приложение Л

Листинг Docker-compose конфигурация

```
version: "3.9"
services:
  postgres:
    container_name: postgres_container
    image: postgres:latest
    environment:
      POSTGRES_USER: ${POSTGRES_USER:postgres}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD:postgres}
      PGDATA: /data/postgres
    volumes:
      - postgres:/data/postgres
    ports:
      - "5432:5432"
    networks:
      - postgres
    restart: unless-stopped

  user-manager:
    image: "registry.gitlab.com/wirax-vkr/user-manager"
    ports:
      - "8010:8010"
    depends_on:
      - postgres
  document-manager:
    image: "registry.gitlab.com/wirax-vkr/document-manager"
    ports:
      - "8020:8020"
    depends_on:
      - user-manager
    restart: always

  personal-data-manager:
    image: "registry.gitlab.com/wirax-vkr/personal-data-manager"
    ports:
      - "8030:8030"
    depends_on:
      - document-manager
    restart: always

  training-manager:
    image: "registry.gitlab.com/wirax-vkr/training-manager"
    ports:
      - "8040:8040"
    depends_on:
      - personal-data-manager
    restart: always
```

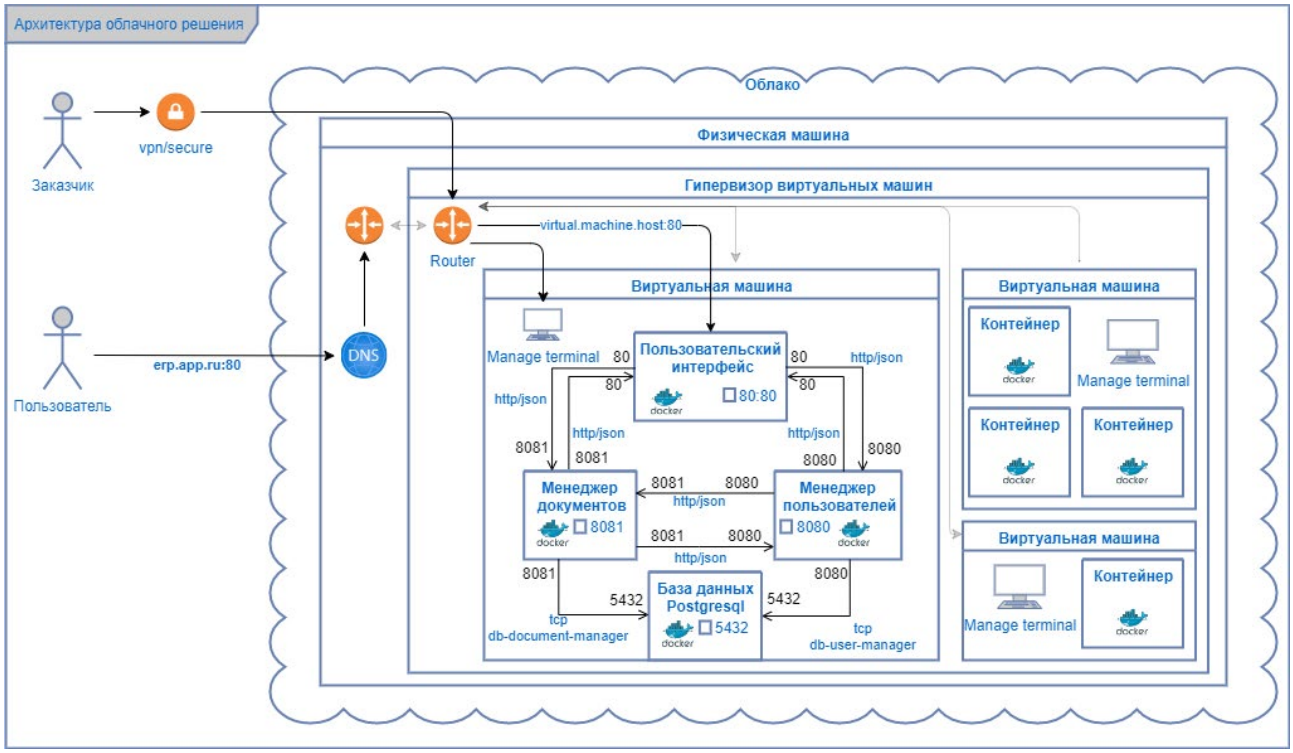
Приложение М

План развертки приложения / средства CI



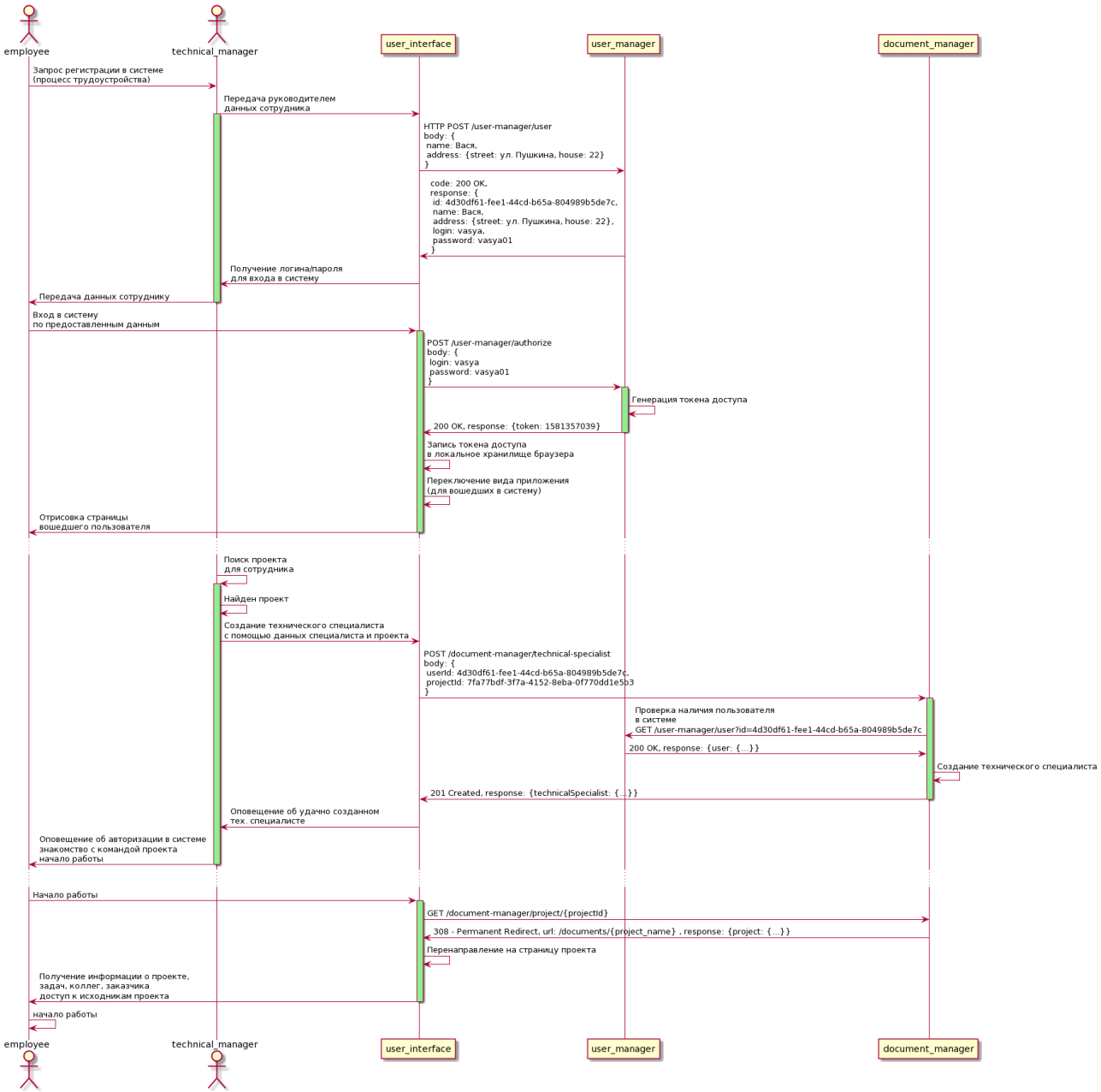
Приложение Н

Сетевое взаимодействие приложения



Приложение О

Процедура регистрации нового пользователя



Приложение П

Листинг класса тестов для сервиса пользователей

```
package com.wirax.vkr.usermanager.service;

import com.wirax.vkr.usermanager.model.User;
import com.wirax.vkr.usermanager.repository.UserRepository;
import org.junit.*;
import org.junit.runner.RunWith;
import org.mockito.*;

@RunWith(MockitoJUnitRunner.class)
public class UserServiceTest {
   @Mock
   private UserRepository userRepository;

   @InjectMocks
   private UserService userService;

   private static List<User> findAllUsersMock = Arrays.asList(new User("Vasiliy", "Petrov"), new
User("Maksim", "Sidorov"));

   @Before
   public void setUp() throws Exception {
      when(userRepository.save(any())).thenAnswer(i -> i.getArguments()[0]);
      doNothing().when(userRepository).deleteById(any());
      when(userRepository.findAll()).thenReturn(findAllUsersMock);
   }

   @Test
   public void returnOK_IfUserCreateUses() {
      User creatingUser = new User("Vasily", "Petrov");

      User userVasilyPetrov = userService.create(creatingUser);

      assertEquals(userVasilyPetrov, creatingUser);
   }

   @Test(expected = IllegalArgumentException.class)
   public void returnException_whenUpdateIds_notEquals() {
      User creatingUser = new User(UUID.randomUUID(), "Vasily", "Petrov");

      userService.update(UUID.randomUUID(), creatingUser);
   }

   @Test
   public void returnNothing_whenUserDelete_methodUsed() {
            userService.delete(UUID.randomUUID());

}

   @Test
   public void returnListOfPrecreatedUsers_whenMethodFindAll_methodUsed() {
      List<User> actualResult = userService.findAll();

      assertEquals(findAllUsersMock.size(), actualResult.size());
      assertEquals(findAllUsersMock, actualResult);
   }
}
```