МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ И 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 кафедры)
_________________09.04.03 Прикладная информатика _____________
(код и наименование направления подготовки, специальности)

__Информационные системы и технологии корпоративного управления__
(направленность (профиль)/специализация

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему_Интеграция мобильного клиента продаж страховых полисов с корпоративной информационной системой страховой компании

Студент _______С.В. Неклюдов_____ _________________
(И.О. Фамилия) (личная подпись)

Научный руководитель _______С.В. Мкртычев_____ _________________
(И.О. Фамилия) (личная подпись)

Руководитель программы д.т.н., доцент С.В. Мкртычев _________________
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____»______________________20_____г.

Допустить к защите
Заведующий кафедрой к.т.н., доцент, А.В. Очеповский _________________
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____»______________________20_____г.

Тольятти 2019

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 3
ГЛАВА 1 АНАЛИЗ МЕТОДОВ ИНТЕГРАЦИИ КОРПОРАТИВНЫХ ПРИЛОЖЕНИЙ 6
1.1 Описание и анализ предметной области 6
1.2 Анализ подходов построения архитектуры корпоративных информационных систем 7
1.3 Анализ современных способов интеграции информационных систем 9
1.4 Анализ современных информационных систем по продаже страховых полисов 21
ГЛАВА 2 МЕТОДЫ РАЗРАБОТКИ И ИНТЕГРАЦИИ МОБИЛЬНОГО ПРИЛОЖЕНИЯ 27
2.1 Методы разработки мобильных приложений 27
2.2 Разработка архитектуры мобильного приложения продажи страховых полисов 33
ГЛАВА 3 МОДЕЛЬ ИНТЕГРАЦИИ МОБИЛЬНОГО ПРИЛОЖЕНИЯ С КОРПОРАТИВНОЙ ИНФОРМАЦИОННОЙ СИСТЕМОЙ СТРАХОВОЙ КОМПАНИИ 38
3.1 Разработка модели интеграции мобильного приложения с КИС страховой компании 38
3.2 Разработка мобильного приложения продажи страховых полисов 48
ГЛАВА 4 АПРОБАЦИЯ МОДЕЛИ ИНТЕГРАЦИИ МОБИЛЬНОГО ПРИЛОЖЕНИЯ С КОРПОРАТИВНОЙ ИНФОРМАЦИОННОЙ СИСТЕМОЙ СТРАХОВОЙ КОМПАНИИ 57
4.1 Процесс внедрения системы для интеграции мобильного приложения с корпоративной информационной системой страховой компании 57
4.2 Процесс интеграции мобильного приложения с корпоративной информационной системой страховой компании 64
4.3 Расчет надежности разработанной информационной системы 68
ЗАКЛЮЧЕНИЕ 71
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 72
ПРИЛОЖЕНИЕ А 76
ПРИЛОЖЕНИЕ Б 78

ВВЕДЕНИЕ

В настоящее время растет количество пользователей интернет-услугами и это не обходит стороной интернет-страхования. Многие страховые компании предлагают свои продукты онлайн. Также растет популярность мобильных услуг, когда пользователь может получить необходимую ему услугу через мобильное приложение.

Однако степень удобства и скорость получение страхового полиса может быть разной и не все страховые компании или интернет-сервисы по продажам страховых услуг имеют мобильные приложение.

Для решения данной проблемы необходимо обеспечить интеграцию мобильного клиента продаж страховых полисов с корпоративной информационной системой (КИС) страховой компании.

Это обуславливает **актуальность** настоящей магистерской диссертации.

Объектом исследования магистерской диссертации является КИС страховой компании.

Предметом исследования магистерской диссертации является интеграция мобильного клиента продаж страховых полисов с КИС страховой компании.

Целью исследования является анализ существующих методов интеграции мобильных приложений с корпоративными системами страховых компаний и разработка модели интеграции, которая обеспечит высокую эффективность реализации продаж страховых услуг.

Для достижения поставленной цели требуется решить следующие задачи:

– Проанализировать подходы к интеграции мобильных приложений с корпоративными информационными системами;

– Проанализировать современные мобильные приложения по продаже страховых полисов;

– Разработать модель интеграции мобильного приложения по продаже страховых полисов с КИС страховой компании;

– Подтвердить эффективность предлагаемой модели на практике.

Исследование состоит из следующих этапов:

– Анализ подходов к построению архитектуры корпоративных информационных систем страховых компаний;

– Анализ способов интеграции корпоративных информационных систем;

– Разработка архитектуры для интеграции мобильного приложения с КИС страховой компании;

– Разработка мобильного приложения;

– Анализ результатов интеграции приложения.

Гипотеза исследования: применение предлагаемой модели интеграции обеспечить простоту интеграции мобильного приложения для продажи страховых полисов с КИС страховой компании.

На защиту выносятся:

– Модель интеграции мобильного приложения для продажи страховых полисов с КИС страховой компании.

– Результаты апробации интегрированного с КИС страховой компании мобильного приложения для продажи страховых полисов.

Результаты диссертационного исследования были представлены в статьях, опубликованных в журналах «Проблемы науки» №2(38) 2019 г. и «Academy» №3(42) 2019 г.

Научная новизна исследования заключается в разработке модели интеграции мобильного приложения с КИС страховой компании.

Практическая значимость работы заключается в разработке мобильного приложения для продажи страховых полисов, которое просто интегрируется с КИС страховой компании.

Методы исследования: методы и модели интеграции компонентов КИС, объектно-ориентированный подход к анализу и проектированию ИС.

Первая глава посвящена анализу архитектуры современных корпоративных информационных систем, методов интеграции с КИС, а также

приведен обзор современных информационных систем по продаже страховых полисов.

Во второй главе рассмотрены существующие методы разработки мобильных приложений. Описана архитектура приложения и разработана модель интеграции мобильного клиента с КИС.

Третья глава содержит этапы проектирования и разработки мобильного приложения и приложения для интеграции КИС страховой компании.

Четвертая глава описывает экспериментальную апробацию приложения и платформу для интеграции, а также приведен расчет надежности интеграционной платформы.

В заключении подводятся итоги выполненной работы.

Диссертация состоит из введения, четырех глав, заключения, списка литературы и приложения.

Работа изложена на 80 с. и включает 37 рисунков, 4 таблицы.

ГЛАВА 1 АНАЛИЗ МЕТОДОВ ИНТЕГРАЦИИ КОРПОРАТИВНЫХ ПРИЛОЖЕНИЙ

1.1 Описание и анализ предметной области

Заключение онлайн-договора страхования мало отличается от стандартного сценария обращения в офис страховой компании и также опирается на законодательства, которую регулируют процесс страхования.

Соответствующие поправки в Закон РФ от 27.11.1992 №4015-1 «Об организации страхового дела в Российской Федерации» были внесены 04.06.2014г. Однако уже через год рынок онлайн-страхования уже достаточно насыщен. Страховые продукты отличаются, возможность продажи их через Интернет считается хорошим тоном среди страховщиков. В первую очередь – страхование выезжающих за границу, а с 01 июля 2015 года — это также обязательное страхование гражданской ответственности транспортных средств (ОСАГО) [1].

Основные положения обмена информацией в электронной форме между страхователем и страховщиком отмечены в статье 6.1 Закона РФ от 27.11.1992 №4015-1 «Об организации страхового дела в Российской Федерации». Выделим следующие моменты:

– страхователь взаимодействует со страховщиком в электронной форме, используя официальный сайт страховщика в информационной и телекоммуникационной сети Интернет, то есть страхователь должен напрямую использовать официальный сайт страховщика;

– Информация, отправленная страховщику и подписанная простой электронной подписью застрахованного лица, признается электронным документом, эквивалентным документу на бумаге, подписанному собственноручной подписью этого лица. В этом случае использование простой электронной подписи должно соответствовать требованиям 63-ФЗ;

– договор страхования должен быть подписан квалифицированной ЭП страховщика, в соответствии с 63-ФЗ.

Также необходимо учитывать, что заключение договора страхования с физическим лицом подразумевает предоставление персональных данных. Соответственно, данные, передаваемые страхователем, должны быть защищены с учетом требований 152-ФЗ «О персональных данных» [1].

1.2 Анализ подходов построения архитектуры корпоративных информационных систем

Сегодня на российских предприятиях и в организациях созданы ИТ-инфраструктуры, множество прикладных систем (бухгалтерский учет, персонал, ERP и CRM-системы, биллинг, другие прикладные программы, в том числе специализированные). Эти ИТ-системы собирают достаточно исходной информации для поддержки принятия оперативных решений. Но трудность заключается в том, что эти «необработанные» подробные данные должны быстро собираться и обрабатываться (отслеживаться и проверяться, приводить к единообразным форматам, агрегироваться и анализироваться), для чего необходимо использовать интеграционные и аналитические технологии [2].

Кроме того, по мере увеличения числа прикладных систем на предприятиях и в организациях проблема их интеграции выходит на первый план: внедрение новых, замена или модернизация существующих обычно требует либо изменения интеграционных связей между ними, либо создания их с нуля. Более того, постоянное развитие бизнеса, бизнес-процессов предприятия приводит к постоянным изменениям в интеграционных связях между приложениями, даже если новые приложения не появляются.

В настоящее время наиболее развитым подходом к решению задач интеграции приложений является разработка корпоративной информационной системы (КИС) предприятия в соответствии с концепцией сервис-ориентированной архитектуры. — SOA (Service-Oriented Architecture) [3].

Центральным звеном SOA является корпоративная сервисная шина — ESB (Enterprise Service Bus) которая показана на рисунке 1.1.

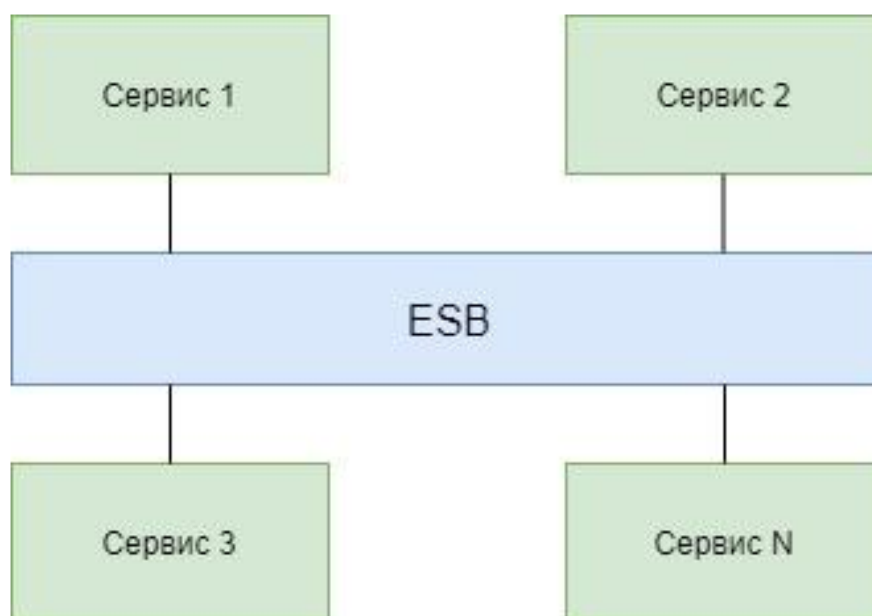

Рисунок 1.1 – Схема корпоративной сервисной шины

Можно выделить несколько сценариев применений сервис-ориентированной архитектуры:

– управление крупными территориально-распределенными компаниями;

– управление услугами и бизнес-процессами;

– управление процессами, требующими интенсивной обработки документов.

Так же стоит упомянуть еще один подход к построению КИС – платформа SAP, которая была разработана для комплексной автоматизации крупных предприятий. Практически все программное обеспечение написано на языке собственной разработки ABAP/4, такое решение позволяет сделать решение независимым от аппаратуры, операционной системы и СУБД (рисунок 1.2).

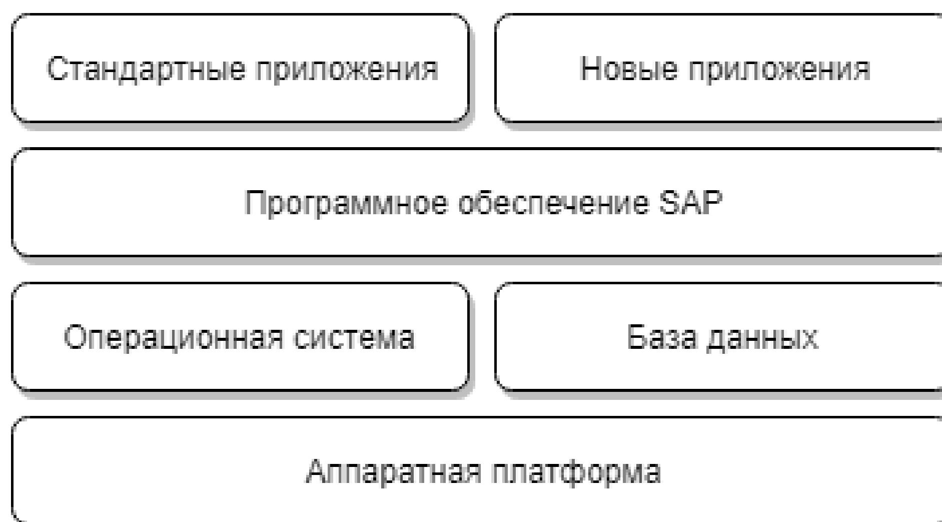

Рисунок 1.2 – Структура платформы SAP

SAP ERP реализована по архитектуре клиент/сервер что позволяет масштабировать систему в зависимости от масштаба организации. Эта система позволяет объединять несколько филиалов или предприятий в одну информационную среду независимо от их местонахождения.

1.3 Анализ современных способов интеграции информационных систем

Одним из наиболее выраженных направлений развития современных корпоративных информационных систем (ИС) является их ориентация на интеграцию друг с другом посредством формирования единого информационного пространства (ЕИП) предприятия (рисунок 1.3).

Рисунок 1.3 - Обобщенная схема ЕИП

С точки зрения практической значимости ЕИП призван сохранять целостность данных и возможность их использования различными пользователями в различных информационных системах в соответствии с их профилем деятельности. Используя ЕИП можно построить процессный подход к управлению, который будет цели устранения фрагментации в работе, путем сокращения организационных и информационных недостатков [4]. Организации единого информационного пространства изучается учеными последние несколько лет, но основная проблема состоит не только в выборе и развитии технологий интеграции ИС, но также и в создании неразрывного представления данных. Следовательно, новизна исследований, связанных с ЕИП, опирается на исследование методов структурирования информации, получаемой из различных ИС. Для разработки этой концептуальной основы ЕИП использует принципы теории множеств, управления проектами, графов, и т.д.

Единое информационное пространство можно организовать путем интеграции всех информационных систем предприятия с целью поддержания целостности данных в любой момент времени, так как сейчас на любом предприятии существует множество информационных и программных систем, и их можно объединить в единое пространство сохраняя консистентность данных.

Обычно, информационная система состоит из следующих компонентов:

– платформа, на которой работают основные компоненты системы, включая аппаратное (железо) и системное программное обеспечение;

– приложения, реализующие бизнес-логику для работы с системой данных. Состоит из таких компонентов как: бизнес-логика, пользовательского интерфейса, компонентов поддержки (интерфейсов) и сервера приложений, который обеспечивает хранение и доступ к компонентам приложения;

– данные, с которыми работает система. Она состоит из различных СУБД;

– бизнес-процессы, представляющие из себя сценарии работы пользователя с системой.

Следовательно, интеграция информационных систем - это интеграция одного или нескольких компонентов информационных систем (объектов интеграции):

– платформ;

– данных;

– приложений;

– бизнес-процессов.

Интегрированием корпоративных платформ пытаются покрыть следующие требования:

1. Корректная кроссплатформенная коммуникация приложений между собой (например, между RedHat Linux, Centos, Windows);

2. Корректная работа написанных кроссплатформенных приложений.

Существует различные подходы для интеграции приложений и в каждом из этих подходов могут использоваться различные технологии:

– технология виртуализации;

– удаленный вызов процедур (REST, RPC, Web-сервисы и пр);

– ПО промежуточного слоя (Microsoft.Net, Java Runtime).

Сейчас набирает популярность подход виртуализации. Виртуализация операционной системы - это программного обеспечения, которое позволяет на «железном» сервере запускать несколько образов операционной системы одновременно. Виртуализация описывает технологию, в которой приложение, гостевая операционная система или хранилище данных отделены от оборудования или программного обеспечения [5]. Основное использование технологии виртуализации - виртуализация серверов, которая использует программный уровень, называемый гипервизором, для эмуляции базового оборудования. Это часто включает память процессора, ввод-вывод и сетевой трафик. Гостевая операционная система взаимодействует с программной эмуляцией этого оборудования, и зачастую гостевая операционная система даже не подозревает, что находится на виртуализированном оборудовании (рисунок

1.4). Хотя производительность этой виртуальной системы не равна производительности операционной системы, работающей на настоящем оборудовании, концепция виртуализации работает, поскольку большинству гостевых операционных систем и приложений не требуются полные мощности базового оборудования. Это обеспечивает большую гибкость, контроль и изоляцию, устраняя зависимость от конкретной аппаратной платформы. Первоначально предназначенная для виртуализации серверов, концепция виртуализации распространилась на приложения, сети, данные и настольные компьютеры. Примеры технологий виртуализации: Microsoft Hyper-V, KVM, Openshift, Virtuozzo, VMware, Xen и пр.

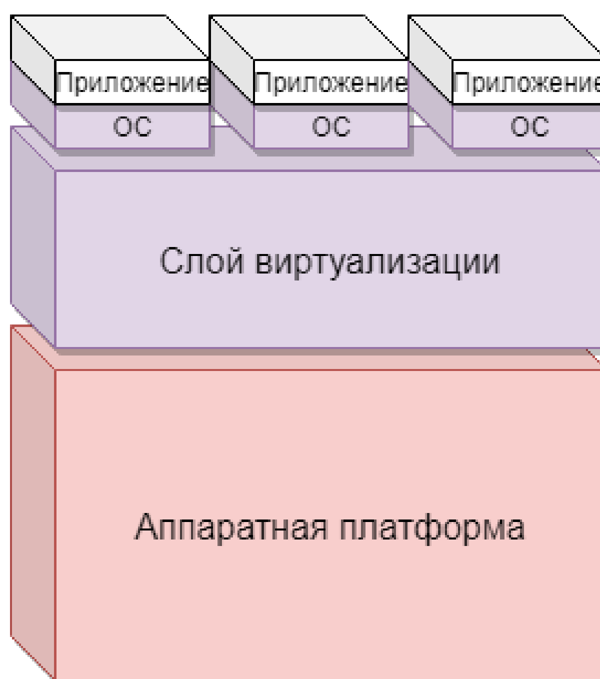

Рисунок 1.4 – Традиционная архитектура виртуализации

Технологии удаленных вызовов процедур (RPC - это аббревиатура для удаленного вызова процедур) - это технология меж сервисного взаимодействия, которая позволяет компьютерной программе вызывать процедуру в другом адресном пространстве (обычно на другом компьютере или сервере, соединенном сетью). Программист не заботится о деталях реализации удаленного взаимодействия: с точки зрения кода, вызов аналогичен вызовам локальной процедуры.

RPC является популярной технологией для реализации клиент-серверной модели распределенных вычислений. Удаленный вызов процедуры инициируется клиентом, отправляющим сообщение на удаленный сервер для выполнения определенной процедуры. Ответ возвращается клиенту. Важное различие между вызовами процедур и удаленными сайтами вызовов процедур заключается в том, что в первом случае вызов может завершиться неудачей из-за проблем в сети. В этом случае даже не гарантируется, что процедура была вызвана.

Идея RPC датируется 1976 годом, когда она была описана в RFC 707. Одно из самых ранних коммерческих применений этой технологии было сделано Xerox в «Courier» в 1981 году. Первой популярной реализацией для Unix был RPC от Sun (теперь он называется ONC RPC), которая используется в качестве основы сетевой файловой системы, и до сих пор используется на многих платформах. В данный момент широко используются SOAP и REST протоколы, основа современных веб-сервисов.

REST определяет набор архитектурных принципов для разработки веб-сервисов, предназначенных для системных ресурсов, включая способы обработки и передачи состояний ресурсов через HTTP для различных клиентских приложений, написанных на разных языках программирования. За последние несколько лет REST стал преобладающей моделью проектирования веб-сервисов [7]. Фактически, REST оказал такое большое влияние на Web, что почти вытеснил дизайн интерфейса на основе SOAP и WSDL, благодаря его значительно более простому стилю разработки.

Технология удаленного вызова также содержит и недостатки главный из которых является необходимость работоспособности всех сервисов, с которыми происходит коммуникация. Можно привести простой пример, допустим есть какой-то сервис каталог, который содержит различную информацию, и в определенное время этот сервис синхронизируется с другими приложениями. Может быть так что часть из систем в это время находятся в нерабочем состоянии.

Множество источников и статей говорит, что использовать технологии удаленного вызова, следует только тогда, когда взаимодействия между сервисами инициируется самим пользователем, который сам контролирует результат. Этот способ плохо применим если нужно чтобы интеграция происходила автоматически. Подход был разработан для разработки распределённых систем, когда сервисы одной системы работают на различных станциях.

Концепция программного обеспечения middle-layer заключается в разработке прикладного программного обеспечения, не использующего службы конкретной операционной системы (например, Windows API), а использующего службы программного обеспечения промежуточного слоя. Промежуточное программное обеспечение – это программное обеспечение, которое связывает программные компоненты или корпоративные приложения, ниже на рисунке 1.5 показана архитектура этого похода. Промежуточное программное обеспечение – это программный уровень, который находится между операционной системой и приложениями на каждой стороне распределенной компьютерной сети. Как правило, он поддерживает сложные, распределенные приложения для бизнеса.

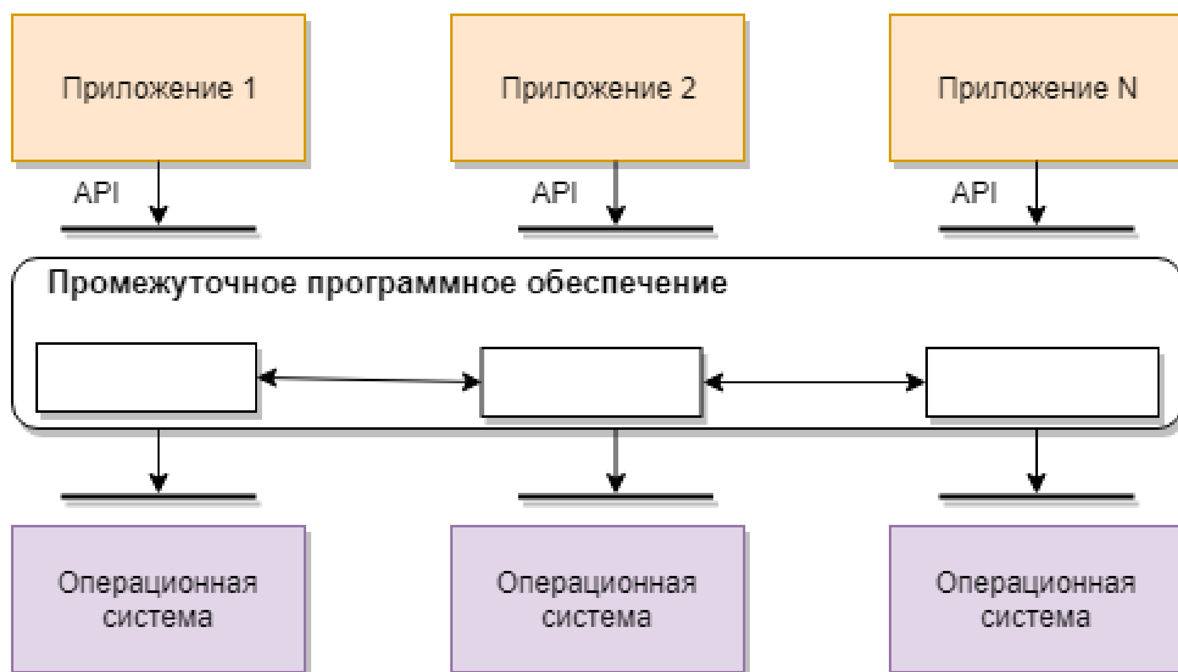

Рисунок 1.5 – Middleware архитектура

Так же это ПО можно считать инфраструктурой, которая облегчает создание бизнес-приложений и предоставляет основные сервисы, такие как параллелизм, транзакции, многопоточность, обмен сообщениями и инфраструктуру SCA для приложений на основе сервис-ориентированной архитектуры (SOA). Оно также обеспечивает безопасность и обеспечивает высокую доступность для предприятия.

Промежуточное программное обеспечение включает в себя веб-серверы, серверы приложений, системы управления контентом и аналогичные инструменты, которые поддерживают разработку и доставку приложений. Он особенно важен для информационных технологий, основанных на расширяемом языке разметки (XML), протоколе простого доступа к объектам (SOAP), веб-службах, инфраструктуре SOA, Web 2.0, протоколе облегченного доступа к каталогам (LDAP) и т.д.

Из определения информационная система работает с различными хранилищами данных. Обычно под хранилищем данных подразумевают базу данных, но также можно считать и обычные файлы или различные таблицы, документы и т.д. Интеграция на уровне данных использует обмен данными из различных систем. Эти данные могут быть с интегрированы более простым путем на этом уровне, нежели на уровне приложений, так как современные СУБД (система управления базами данных) из коробки могут поддерживать различные протоколы для разных платформ, что упрощает доступ к данным.

Есть два подхода к интеграции данных:

- хранилище данных;
- универсальный доступ к данным.

Универсальный доступ к данным - это стратегия Microsoft по обеспечению доступа к данным на предприятии. Компании, разрабатывающие решения для баз данных, сталкиваются с рядом проблем, стремясь получить максимальные бизнес-преимущества от данных и информации. Благодаря OLE DB и ADO универсальный доступ к данным обеспечивает высокопроизводительный доступ к различным источникам информации, включая реляционные и нереляционные

источники, и простой в использовании интерфейс программирования, который не зависит от инструмента и языка. Эти технологии позволяют корпорациям интегрировать разнообразные источники данных, создавать простые в обслуживании решения и использовать различные инструменты, приложения и платформы (рисунок 1.6).

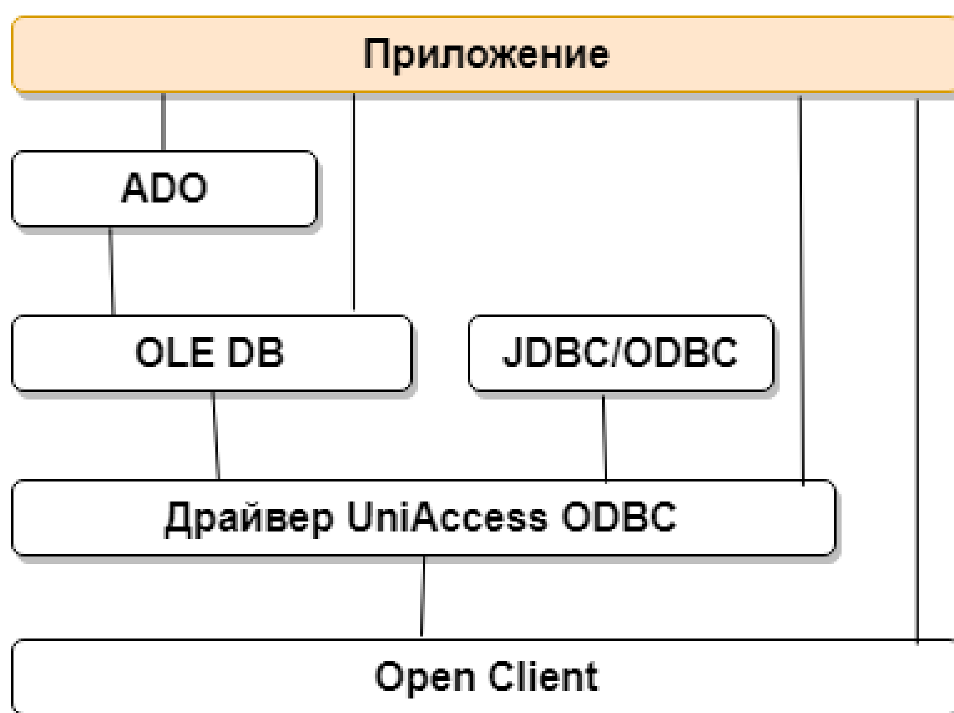

Рисунок 1.6 – Концепция универсального доступа к данным

Универсальный доступ к данным не требует от пользователя переносить данные в одно хранилище данных, что является дорогостоящим и требует много времени. Универсальный доступ к данным основан на открытых спецификациях с широкой поддержкой и работает со всеми основными платформами баз данных. Универсальный доступ к данным является эволюционным шагом по сравнению с современными стандартными интерфейсами, включая ODBC, RDO и DAO; и расширяет функциональность этих известных и проверенных технологий. Доступ к данным основан на способности OLE DB получать доступ к данным всех типов, и он полагается на ADO, чтобы предоставить модель программирования, которую разработчики приложений будут использовать.

Концепция хранилища данных содержит в себе идеи создания корпоративного хранилища данных. Она включает в себя объединение данных из нескольких разнородных источников, которые хранятся с использованием различных технологий и обеспечивают единое представление данных. Интеграция данных становится все более важной в случае слияния систем двух компаний или объединения приложений в одной компании для обеспечения единого представления информации данных компании.

Вероятно, наиболее известной реализацией интеграции данных является создание warehouse (хранилища данных предприятия). Это позволяет компании проводить анализы на основе данных в хранилище данных. Это было бы невозможно сделать с данными, доступными только в исходной системе. Причина в том, что исходные системы могут не содержать соответствующих данных из других систем, и даже если данные имеют одинаковые имена, они могут ссылаться на разные объекты.

Интеграция на уровне Интеграция приложений (или интеграция корпоративных приложений) – это разделение процессов и данных между различными приложениями. Как для небольших, так и для крупных организаций стало критически важным приоритетом подключение разрозненных приложений и использование совместной работы приложений в масштабах всего предприятия для повышения общей эффективности бизнеса, повышения масштабируемости и снижения затрат на ИТ.

Существуют следующие подходы к интеграции приложений:

- интерфейсы;
- обмен сообщениями (корпоративная сервисная шина);
- Service oriented architecture (сервис-ориентированная архитектура);
- интеграция юзер-интерфейсов (на уровне представления).

Программный интерфейс (API – application programming interface) это набор инструментов, определений и протоколов для построения и интеграции прикладного программного обеспечения. Он позволяет вашему продукту или

сервису взаимодействовать с другими продуктами и услугами, не зная, как они реализованы.

API может упростить разработку приложений, что может сэкономить время разработчиков и деньги компаний. Когда вы разрабатываете новые инструменты и продукты или управляете существующими, интерфейсы предоставляют вам гибкость, позволяет упростить дизайн, администрирование и использование.

Сервис-ориентированная архитектура (SOA) - модульный подход к разработке программного обеспечения, основанный на использовании распределённых, слабо связанных компонентов, оснащённых стандартизированными интерфейсами для взаимодействия по стандартизированным протоколам. SOA позволяет пользователям комбинировать большое количество средств из существующих сервисов для формирования приложений.

Она так же включает в себя набор принципов проектирования, которые структурируют разработку системы и предоставляют средства для интеграции компонентов в целостную и децентрализованную систему.

Архитектура объединяет все взаимодействующие сервисы, которые могут быть интегрированы в различные программные системы, принадлежащие отдельным бизнес-доменам.

Сервис-ориентированная архитектура содержит две важные роли. Сервис провайдер – представляет из себя сервис, которые предоставляет функционал и сервисы для использования другими.

Сервис консьюмер – который содержит в себе мета-данные и содержит в себе все необходимые клиентские компоненты, которые могут быть использованы [7].

При таком походе приложение состоит из нескольких уровней. Эту структуру можно видеть на рисунке 1.7.

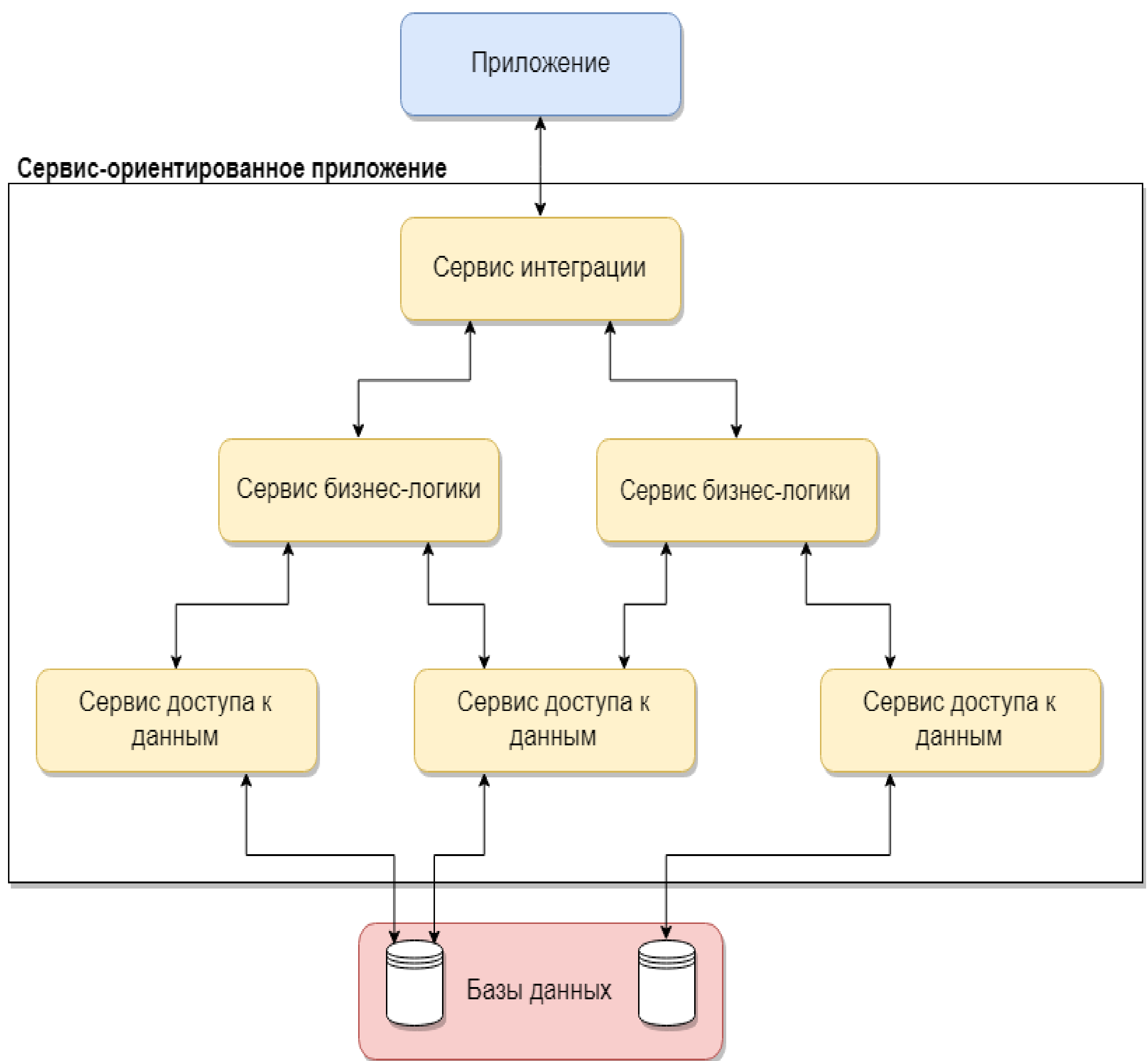

Рисунок 1.7 – Уровни сервис-ориентированного приложения

Самый верхний уровень содержит одну или несколько служб интеграции, каждая из которых контролирует потоки запросов. Каждый сервис интеграции вызывает один или несколько бизнес-сервисов.

Второй уровень состоит из сервисов, каждый из которых выполняет бизнес-задачу относительно низкого уровня. Сервис интеграции может вызвать серию бизнес-сервисов.

Третий уровень состоит из служб доступа к данным, каждая из которых выполняет относительно техническую задачу чтения и записи в области хранения данных, такие как базы данных и очереди сообщений. Служба доступа к данным чаще всего вызывается из бизнес-уровня, но простой доступ к службам позволяет использовать ее по-разному.

Основными идеями SOA являются:

– публикация функционала корпоративных приложений в виде Веб-сервисов;

– создание новых приложений на основе веб-сервисов путем их объединения.

Стоимость создания новых приложений на основе существующих веб-сервисов будет значительно ниже, чем разработка приложений с нуля или обширная интеграция с другими системами.

На уровне представления интеграция достигается путем соединения нескольких различных приложений в качестве одного приложения с общим пользовательским интерфейсом (UI). Интеграция на уровне представления ранее использовалась для интеграции приложений, которые иначе не могли быть подключены, но технология интеграции приложений с тех пор развивалась и стала более сложной, что делает этот подход менее распространенным.

Наиболее целостный подход к системной интеграции - это интеграция на уровне бизнес-процессов. На этом уровне интеграции происходит интеграция приложений, интеграция данных и людей [7]. Предприятия могут использовать интеграцию приложений для определения взаимодействия отдельных приложений с целью автоматизации важных бизнес-процессов, что приводит к более быстрой доставке товаров и услуг для клиентов, снижение вероятности человеческих ошибок и снижение эксплуатационных расходов.

Идеи, лежащие в основе интеграции бизнес-процессов, достаточно просты:

– создается сценарий бизнес-процесса, существующего в организации, опишите в нем все операции взаимодействия, которые возникают между пользователями системы и между самими системами. Исходя из этого, бизнес-процесс представляет из себя элемент, который логически интегрирует и включает в себя различные системы. Бизнес-сценарий формируется с помощью специального программного обеспечения, который дальше будет контролировать ход этого бизнес-процесса в соответствии со сценарием;

– операции взаимодействия с системами в рамках бизнес-процесса подробно описаны в терминах обмена информацией: события, используемые службы, приложения, правила, форматы обмена, и т.п.;

– интегрирующие системы, которые участвуют в бизнес-процессе, подключаются через адаптеры к интегрирующему программному обеспечению, с помощью которого описывается сценарий бизнес-процесса. Таким образом, становится возможным автоматический обмен информацией между системами;

– разработанный процесс добавляется на «панель управления» менеджера, где он сможет управлять существующими бизнес-процессами, отслеживать их состояние, принимать решения по операциям процессов, которые требуют участия пользователя и т.д. Если не требуется взаимодействие пользователя с системами, то это выполняется путем автоматической интеграции ПО [8].

1.4 Анализ современных информационных систем по продаже страховых полисов

Для формирования функциональных требований нового проектируемого приложения необходимо провести обзор аналогичных информационных систем и приложений. Рассмотрим наиболее популярные системы.

1. «КонтиТревел» - онлайн сервис страхования туристов. Этот сервис помогает найти самые дешевые полисы для выезжающих за границу туристов (ВЗР). Помимо классических туристических видов страховых услуг, предлагает:

– Страхование жилья на время отъезда в путешествие;

– Страхование мобильного устройства от разбойного нападения, грабежа, кражи, хулиганства не только на время поездки, но и в повседневной жизни;

– Страхование пассажиров различных видов транспорта от несчастного случая;

– Страхование от несчастного случая в отеле;

– Страхование на случай опоздания на стыковочный рейс;

– Страхование домашних животных на время поездки и многое другое.

На рисунке 1.8 представлен основной интерфейс этого сервиса.

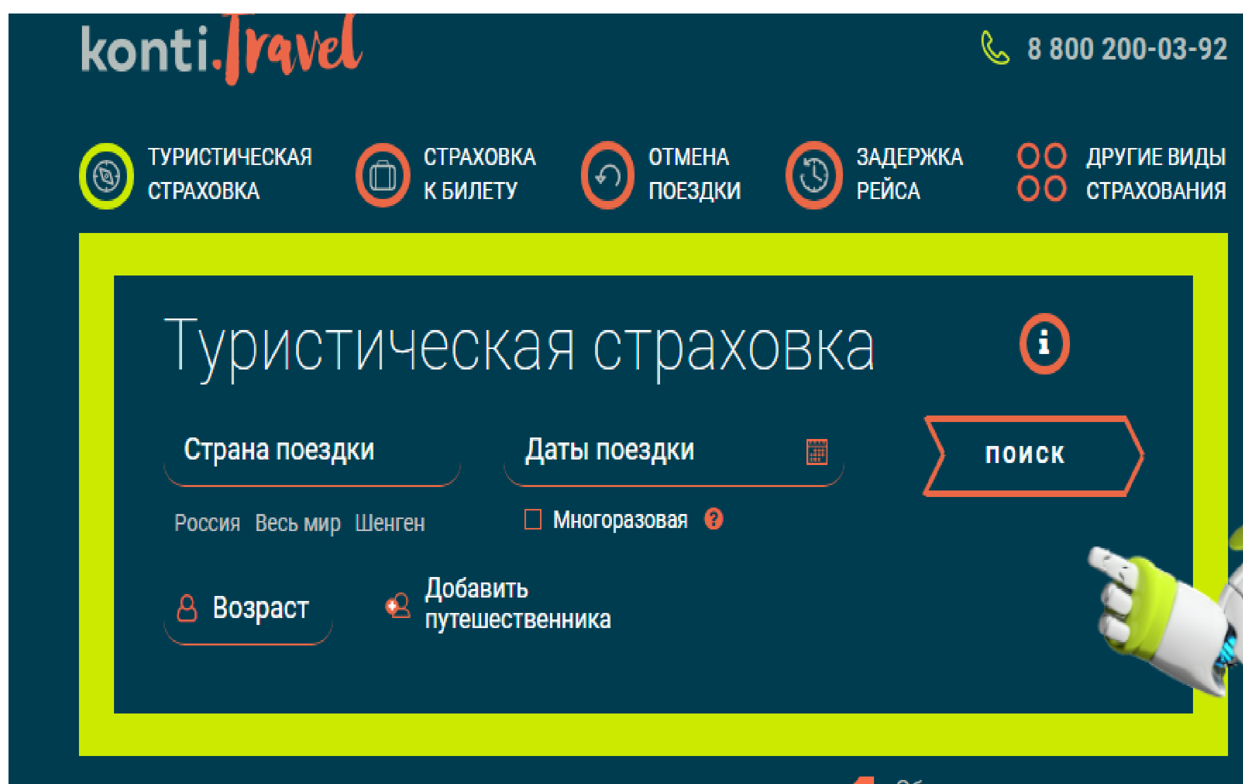

Рисунок 1.8 – Интерфейс сервиса «КонтиТревел»

2. Сервис «Сравни Купи» предлагает оформить ОСАГО и КАСКО помимо туристической страховки, но имеет несколько меньше дополнительных опций для туристов, чем КонтиТревел.

На сайте довольно удобная форма для поиска страховки, но после внесения данных туриста, по каждой страховой компании предлагается несколько вариантов с разной стоимостью.

Понять в чем разница между этими полисами невозможно.

В отличие от КонтиТревел, количество представленных страховых компаний меньше, скидки и акции здесь не предусмотрены, для успешного выбора страховки форму поиска необходимого полиса нужно будет внимательно изучить информацию. Главное окно можно видеть на рисунке 1.9.

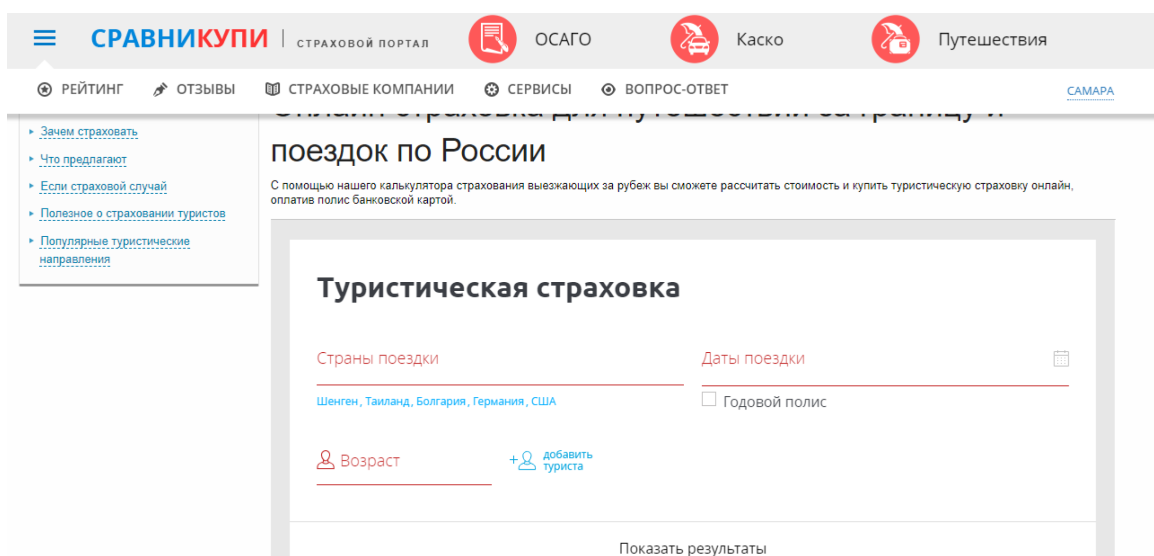

Рисунок 1.9 – Главное окно сервиса «СравниКупи»

3. Сервис «Давай Сравним» тоже уступает КонтиТревел в количестве предлагаемых туристу страховых компаний и дополнительных рисков.

На сервисе не предусмотрены скидки и акции. «Давай Сравним» так же работает со страховыми компаниями, имеющими очень разный рейтинг надежности. Помимо некоторых представленных на КонтиТревел и «Сравни Купи» страховщиков, здесь предлагают оформить полис в страховых компаниях «Эстер», «Антал» и «Гайде».

Форма поиска страховки достаточно конкретная, найти необходимый полис будет легко, уточняющих параметров достаточно, много подсказок. Можно даже сказать, что есть некоторая перегруженность.

4. У федеральной страховой компании «Росгосстрах Жизнь» есть приложение «Кабинет клиента РГС-Жизнь». Оно не предоставляет возможности клиентам купить страховку. В этом приложение можно только ознакомиться с уже купленными страховки и следить за последней информацией по ней.

5. Компания «Ингосстрах» поддерживает приложение IngoMobile, которое позволяет купить туристическую страховку, отправить заявку на расчет стоимости страховых полисов, получить инструктаж при наступлении страхового случая и т.д. Интерфейс приложения представлен на рисунке 1.10.

Рисунок 1.10 – Интерфейс приложения IngoMobile

6. У компании «АльфаСтрахование» разработано несколько мобильных решений, основное из которых – «АльфаСтрахование Mobile». Отличительной особенностью этого приложения является то, что необходимо быть клиентом компании чтобы иметь возможность воспользоваться этим приложением. По моему мнению это и является главным недостатком. На рисунке 1.11 представлен интерфейс «АльфаСтрахование Mobile».

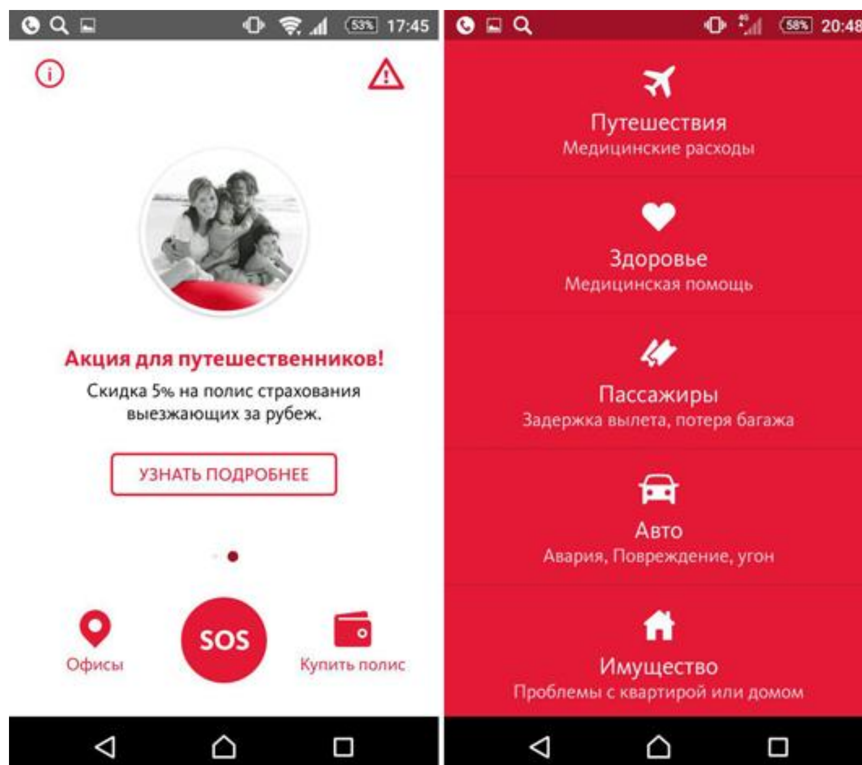

Рисунок 1.11 – Интерфейс приложения «АльфаСтрахование Mobile»

7. Еще одна страховая компания, которая предоставляет услуги покупки страхового полиса и имеет мобильное приложение, является «Ренессанс Страхование». Она предоставляет все те же самые стандартные функции, как и приложения других компаний (рисунок 1.12).

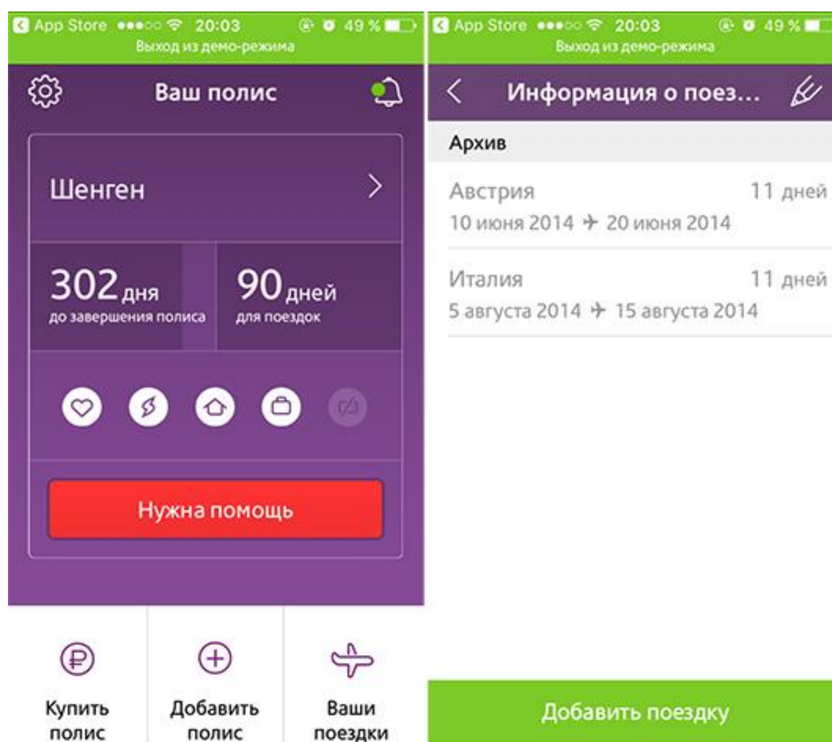

Рисунок 1.12 – Мобильное приложение Ренессанс Страхование

На основе обзора были сформированы критерии разрабатываемого мобильного приложения: оно должно позволять произвести расчет и покупку страхового полиса, сохранение купленных полисов, для последующей возможности пользователю просмотреть их. Так же предложенная далее архитектура позволит подключить к системе не только android-приложение, так же и ios или web-приложение и т.д.

Выводы к главе 1

1)Рассмотрены современные способы интеграции информационных систем: технологии виртуализации, удаленный вызов процедур (например, REST, SOAP и т.д.), программное обеспечение промежуточного слоя. На основе это анализа было сформулировано, что разрабатываемая модель интеграции должна уметь работать с различными интерфейсами корпоративных информационных систем.

2) Проанализированы современные информационные системы по продаже страховых полисов. Были рассмотрены как мобильные приложения, так и веб-версии.

3) Сформулированы основные функции, которые должно иметь разрабатываемое приложение: расчет и покупка полисов, хранение полисов, уведомления.

ГЛАВА 2 МЕТОДЫ РАЗРАБОТКИ И ИНТЕГРАЦИИ МОБИЛЬНОГО ПРИЛОЖЕНИЯ

2.1 Методы разработки мобильных приложений

Информационные ресурсы, системы и технологии являются неотъемлемыми, быстро развивающимися элементами современной человеческой деятельности. Об этом свидетельствуют статистика, опыт и различные исследования зарубежных и отечественных компаний. «В 1997 году на рынке сотовой связи появилась технология WAP, позволяющая устанавливать программы на мобильные телефоны напрямую из Интернета не используя кабель, подсоединенный к компьютеру. Это момент можно считать точкой отсчета, когда начался процесс «мобилизации». Производство мобильных устройств с большими сенсорными экранами в начале 2000-х годов, которое позволило улучшить технологии создания новых мобильных приложений, и это стало качественным прорывом в разработке приложений. В 2010 мобильные устройства стали оснащаться более производительными процессорами, которые позволяют использовать в разработке мобильных приложений все современные информационные технологии. Большая часть разработчиков начали осваивать и уходить в новые направления профессиональной деятельности, связанные с разработкой мобильных приложений, и адаптироваться к тенденциям рынка» [8].

В настоящее время разработка и создание мобильных приложений является одной из самых популярных направлений в области ИТ. Мобильная разработка направлена на создание приложений, которые могут удовлетворить потребительские запросы, решать проблемы, алгоритм которых неизвестен заранее. Теперь приложения способны проводить аналитический анализ поступающей информации и помогают пользователю принимать те или иные решения, контролировать процессы и решать другие задачи с наименьшими для него затратами времени. Это, в свою очередь, помогает оптимизировать бизнес-процессы, повысить эффективность принимаемых решений и производительность.

«Мобильное приложение - это специальная программа для телефона, установленная на конкретной платформе, которая обладает определенными функциональными возможностями, позволяющими выполнять определенные действия в зависимости от задач, которые необходимо решить» [9]. Его можно представить, как адаптер, который позволяет пользователю работать с различной информацией. Поэтому приложения можно разделить на:

- приложения-события;
- приложения службы;
- игры;
- интернет магазины;
- промо-приложения;
- бизнес-приложения;
- системные приложения;
- навигационные и поисковые сервисы;
- мультимедийные приложения;
- социальные сети;
- контентные приложения и др.

Современные мобильные технологии машинного зрения делаются на основе сравнения и поиска моделей на картинках. Вследствие развития технологий этих технологий, мобильные приложения в состоянии определять штрих-коды, распознавать лица, вычислять объекты и т.д. Технология компьютерного зрения обширно используется при разработке медицинского ПО, где необходимо сравнивать огромное количество медицинских показателей, включая изображения, чтобы была возможность определить и правильно поставить диагноз. Технология дополненной реальности (virtual reality) является одной из самых растущих и перспективных направлений развития машинного зрения (распознаваемый объект дополняется информацией, другим объектом или порождает дальнейшие действия приложения). Компьютерное зрение невозможно без машинного обучения.

«Машинное обучение - это обучение приложения распознаванию информации или объектов из внешнего мира, анализу данных и принятию решения в зависимости от алгоритма» [10]. Это могут быть мобильные приложения для прогнозирования прибыли от транзакций, автоматического принятия решения о выдаче кредита, распознавания эмоций, мобильных помощников для улучшения качества жизни и т.д. Технологии построения нейронных сетей, которые недавно начали развиваться, имеют большое значение, потому что нет необходимости знать критерии принятия решения для того чтобы получить прогноз или само решение - сеть изучает и находит сами алгоритмы под руководством разработчика. Действие сети аналогично действию мозга. Обученная нейронная сеть может заданным образом реагировать на информацию из вне в соответствии с алгоритмом, полученным в процессе самообучения машины. Таким образом, современные мобильные технологии позволяют создавать мобильные приложения и сервисы, которые предоставляют ответы, решения и прогнозы, которые неизвестны заранее.

«Для разработчиков мобильных приложений более важна классификация мобильных приложений с точки зрения их структуры» [9].

Нативные приложения написаны на языках программирования для конкретной платформы и встроены в операционную систему, работают быстро и правильно, имеют преимущество как в функциональности, так и в скорости в отличие от других типов мобильных приложений. Они дают возможность построить интерфейс и определить, как программа будет работать на этой платформы. Приложения также предоставляют доступ к аппаратному обеспечению устройства: микрофону, камере, акселерометру и т.д., экономят ресурсы платформы и могут работать полностью или частично при отсутствии Интернет-соединения. Правда с другой стороны, они являются дорогостоящими с точки зрения написания приложения, потому как программисту необходимо владеть обширными знаниями в среде разработки, а также потому, что каждая платформа имеет свой собственный язык программирования.

Нативные приложения имеют свои собственные инструменты и языки программирования. Например, чтобы написать программу для ОС Android чаще используются Android Studio и язык программирования Java, для iOS - ObjectiveC, а также популярный язык программирования Swift, для Windows Phone используется Visual Studio и C #.

Но может возникать ситуация, когда появляется необходимость написать приложение в более сжатые сроки, и чтобы приложение сразу было кроссплатформенным, другими словами могло быть запущено на различных платформах. Тогда в этом случае делается ставка на мобильные веб-приложения или гибридные, а для во время написания таких программ используются различные кроссплатформенные мобильные платформы.

Веб-приложения возможно назвать мобильной версией сайта с расширенным интерактивом. Они не публикуются в магазинах приложений, а вместо этого используют браузер для работы. В зависимости от Интернет-соединения скорость таких приложений может различаться, также они обладают следующими преимуществами: низкой стоимостью и быстрым временем внедрения, кроссплатформенные, используют классические веб-технологии: HTML5, JavaScript и CSS.

Гибридные приложения или генераторы мобильных приложений представляют собой что-то среднее между нативными и веб-приложениями. Они включают в себя кроссплатформенность и возможность использовать программное обеспечение для мобильных устройств. Они обычно публикуются в официальные магазины и соответственно устанавливаются через них, у них не полный доступ к аппаратному обеспечению телефона, имеют возможность самостоятельно обновлять информацию, ближе по функциональности и качеству к нативным приложениям, но они дешевле в зависимости от платформы, используемой разработчиком.

На сегодняшний день сфера мобильной разработки предоставляет различные инструменты и платформы, которые упрощают создание мобильного приложения.

Одной из таких платформ является Appcelerator Titanium. С ее помощью вы можете разрабатывать как десктопные приложения, так и мобильные приложения. На данный момент на этой платформе вы можете создавать приложения для основных мобильных операционных систем. Чтобы написания приложение необходимо знать язык JavaScript.

Программа, разработанная на этой платформе, состоит из объектов, которые имеют уникальные свойства и методы. Большой набор объектов из коробки позволяет по максимуму использовать все функции операционной системы.

В приложении, JavaScript общается с Appcelerator Titanium API. Фреймворк позволяет использовать различные компоненты пользовательского интерфейса для построения таких элементов:

- текстовые поля;
- кнопки;
- списки.

Элементы управления мобильной платформы поддерживают точность представления этих объектов. В большинстве возможных случаях код, написанный для одной платформы, может работать на других платформах без каких-либо изменений. Однако не все графические элементы конкретной платформы могут правильно отображаться. В этом случае для каждой конкретной платформы необходимо написать собственный сегмент кода.

Используясь этой платформой, разработчик получает преимущество в скорости по сравнению с написанием приложения на Java или Swift, но производительность приложения снижается, поскольку скорость native-программ выше. Таким образом можно сделать вывод что программист должен выбрать, что скорость или скорость важнее в конкретной задаче.

Платформа Xamarin может быть использована для разработки кроссплатформенных мобильных приложений. Xamarin использует язык C#. Используя эту платформу, вы можете писать одну логику приложения для нескольких платформах одновременно - Android, iOS, Windows Mobile.

Xamarin состоит из нескольких частей:

– Xamarin.IOS;
– Xamarin Studio;
– Xamarin.Android;
– Компиляторы для iOS и Android;
– Плагины для Visual Studio.

Эти части играют важную роль - с их помощью приложения могут отправлять запросы к интерфейсам приложений на устройствах с операционной системой Android или iOS.

Используя эти платформы, разработчик может создавать как отдельные приложения для конкретной операционной системы, так и используя единую логику - кроссплатформенное приложение. Благодаря этому может быть создан визуальный интерфейс, и вы можете связать с ним код, написанный на C#. Это приложение будет работать на всех основных мобильны платформах Android, iOS и Windows Phone.

Это достигается путем использования технологии Xamarin.Forms. Использование Xamarin дает преимущества в следующих случаях:

– приложение содержит большой кусок кроссплатформенного кода;
– необходимо создать приложение за короткий срок, поддержав несколько платформ;
– необходимо прототипировать приложение.

Не следует использовать Xamarin в следующих случаях:

– разработка приложения идет для конкретной платформы;
– GUI приложение;
– должны быть удовлетворены определенные / особые требования стабильности.

Еще одним инструментом для разработки кроссплатформенных мобильных приложений является Corona SDK. Разнообразный и удобный инструментарий движка, с помощью которого вы сможете за короткое время

создать игру или приложение. Для того чтобы разрабатывать на этом фреймворке необходимо знать язык Lua, которые на самом деле является легким в изучении.

В результате вы получите приложение, которое вы сможете запустить на платформах iOS и Android. В Corona SDK есть практически все необходимое для разработки пользовательского интерфейса, например, ползунки, кнопки и т.д. Приложения на этой платформе производительны и быстры, но все же уступают нативным приложениям.

Можно выделить следующие основными недостатки этой платформы:

– документация преимущественно на иностранном языке;
– приложение компилируется на сервере;
– сложная отладка приложения.

Тот или иной подход к созданию мобильных приложений имеет свои преимущества и недостатки, вследствие этого разработчик должен выбирать технологии на основе потребностей и задач, которые должно выполнять это приложение.

2.2 Разработка архитектуры мобильного приложения продажи страховых полисов

Выбор подхода и архитектуры мобильного приложения играет важную роль, так как от этого будет напрямую зависеть успех приложения, его стоимость, стоимость дальнейшей поддержки и легкость внедрения нового функционала.

В ходе исследовательской работы разработана следующая архитектура мобильного приложения.

В качестве платформы для которой будет разрабатываться приложение была выбрана платформа Android, потому как показывает статистика – более 85% рынка мобильных платформ принадлежит Android. Плюс эта платформа не накладывает ограничение на среду разработки, так как это делает IOS, который

позволяет разрабатывать приложение только будучи используя эту же самую платформу.

Бизнес логика приложения и ее представление будет разнесено – весь функционал, связанный с интеграцией со страховой компанией и с последующей коммуникацией с ней, будет происходит на удаленном сервере, а, соответственно весь пользовательский интерфейс находится на мобильном устройстве. UI будет динамически строится в зависимости от того с какими страховыми компаниями с интегрирован бэкенд приложения. Коммуникация между Android-приложением и сервером будет происходить по протоколу HTTP. Такой подход позволит доставлять до пользователя новый функционал за максимально короткий период, все будет сводится к изменениям на стороне сервера и минимальным изменениям со стороны мобильного приложения. Концепция архитектуры можно видеть на рисунке 2.1

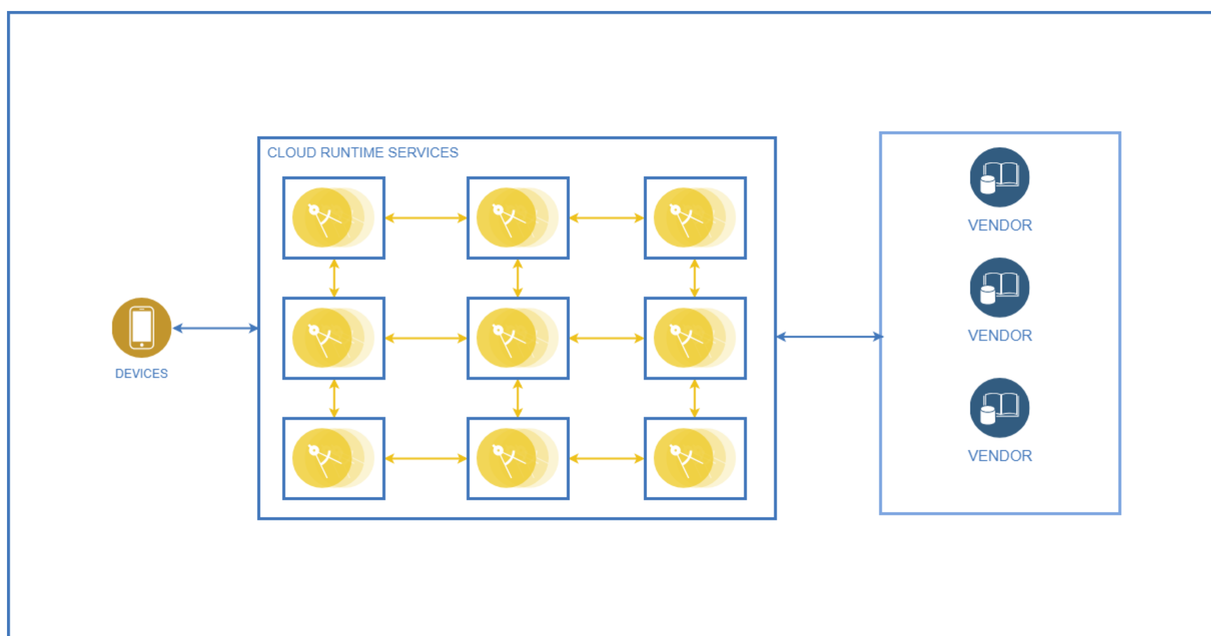

Рисунок 2.13 - Концептуальная архитектура приложения

Для реализации серверной части приложения будут использоваться облачные технологии, так как сейчас они набирают популярность и позволяют абстрагироваться от железной составляющей сервера. Облачные системы - это модели для обеспечения удобного и беспрерывного сетевого доступа к набору

настраиваемых вычислительных ресурсов (таких как системы хранения, приложения, ресурсы памяти, сети, и службы), которые могут быть выделены для той или иной задачи и запущены с минимальными усилиями по управлению и необходимостью взаимодействия с провайдером.

В качестве облачной платформы был выбран Openshift. Red Hat OpenShift Container Platform – это мощная корпоративная платформа для разработки, развертывания и эксплуатации классических и контейнерных приложений в физических, виртуальных и общедоступных облачных средах. Решение основано на хорошо зарекомендовавших себя технологиях с открытым исходным кодом и предлагает эффективную помощь разработчикам приложений и специалистам по эксплуатации ИТ-систем в обновлении прикладных решений, запуске новых сервисов и ускорении процессов разработки. На рисунке 2.2 показана архитектура системы Openshift Container Platform на верхнем уровне абстракции.

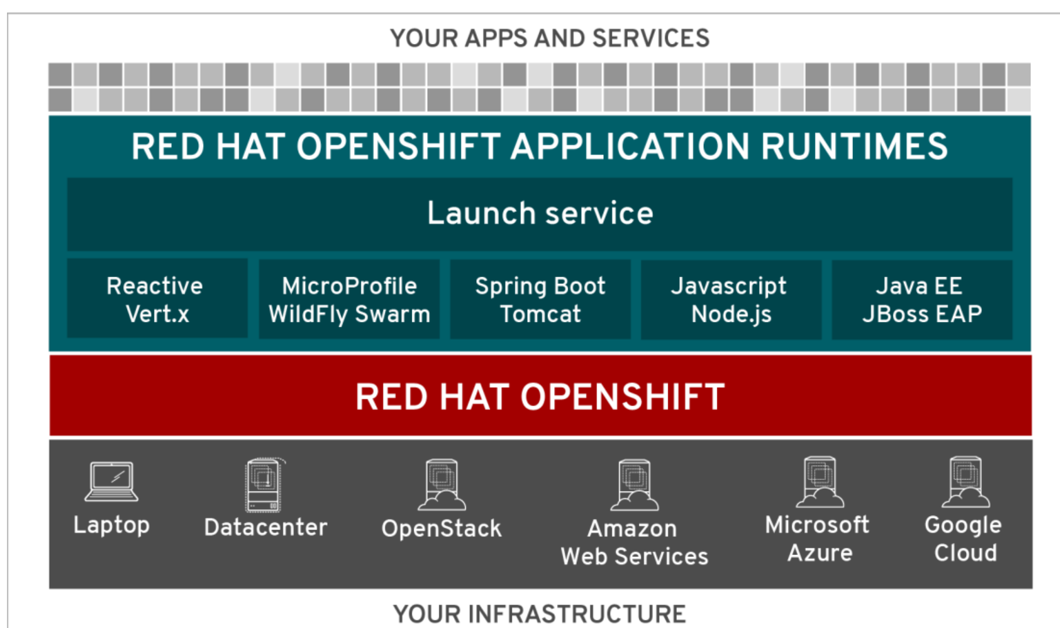

Рисунок 2.14 - Архитектура платформы Openshift

Контейнерная платформа OpenShift обеспечивает оптимальную платформу для подготовки, сборки и развертывания приложений и их компонентов в режиме самообслуживания. Инструменты автоматизации, такие

как встроенное преобразование, значительно упрощают сборку изображений контейнеров в формате докера на основе кода, извлеченного из системы контроля версий. Встроенные инструменты кластеризации, планирования и оркестровки обеспечивают эффективную балансировку нагрузки и автоматическое масштабирование. Функции безопасности полностью исключают риск вмешательства клиентов в другие приложения или хост, а постоянное хранилище подключается напрямую к контейнерам Linux [11].

Необходимо будет развернуть один централизованный сервер с установленными решениями Openstack, которые позволяют осуществлять мониторинг и настройку серверов, а также выделять определенный объем памяти для проектов внутри него, не используя всю мощность сервера [11]. Сами проекты будут контролироваться системой Openshift, которая позволяет интегрировать программный код в виде микросервисов и повысить производительность микросервисов с помощью одной кнопки. Сами микросервисы должны быть обработчиками и распределителями запросов, а также ответственными за постоянное хранение в базе данных микросервисов.

Типовым вариантом разработки подобных микросервисов может являться вариант Spring приложения, написанного на языке Java. Он позволит писать высокоуровневый код, не отвлекаясь на детали реализации низкоуровневых взаимодействий, такие как обеспечение транзакционности, взаимодействия с базой данных, многопоточное разделение входных запросов и т.п. Сами входные запросы могут приниматься по протоколу REST, то есть содержащие в себе тело и заголовки HTTP-запроса, с указанием соответствующего метода для дальнейшей обработки [12].

Выводы к главе 2

1) Проанализированы современные методы разработки мобильных приложений. Выявлены следующие подходы: нативные приложения, написанные на языке для конкретной платформы, веб-приложения и гибридные приложения.

2) Выбрана платформа Android, так как является доминирующей на рынке мобильных приложений. Разработана архитектура мобильного приложения, на основе трехуровневой архитектуры. Для реализации серверной части было решено использовать Openshift платформу, которая позволит эффективно управлять серверной частью. Сервер будет содержать сервис по интеграции с корпоративной информационной системой.

ГЛАВА 3 МОДЕЛЬ ИНТЕГРАЦИИ МОБИЛЬНОГО ПРИЛОЖЕНИЯ С КОРПОРАТИВНОЙ ИНФОРМАЦИОННОЙ СИСТЕМОЙ СТРАХОВОЙ КОМПАНИИ

3.1 Разработка модели интеграции мобильного приложения с КИС страховой компании

Архитектура разрабатываемой информационной системы, которая позволит произвести интеграцию кроссплатформенных приложений с корпоративной информационной системой, будет трехуровневой: в качестве клиента будут выступать приложения (Android, iOS и т.д.), а вся логика будет обрабатываться на сервере приложений.

На начальных этапах проектирования приложения была создана диаграмма вариантов использования. Диаграмма вариантов использования описывает функциональное назначение моделируемой системы. Каждый вариант использования последовательность действий, которые должны быть выполнены системой.

Варианты использования представляют собой овал с текстом сценария. Участие актера в системе моделируется линией между актером и сценарием использования. Для изображения границы системы используют рамку вокруг самого варианта использования.

Диаграммы вариантов использования идеально подходят для:

- изображения целей взаимодействия системы с пользователем;
- определение и организация функциональных требований к системе;
- указание контекста и требований системы;
- моделирование основного потока событий в сценарии использования.

Диаграмма вариантов использования системы для покупки страховых полисов представлена на рисунке 3.1. Система содержит актанта «Пользователь».

Пользователю необходимо войти в систему используя свой логин и пароль, иначе если он до этого не пользовался системой, то необходимо зарегистрироваться.

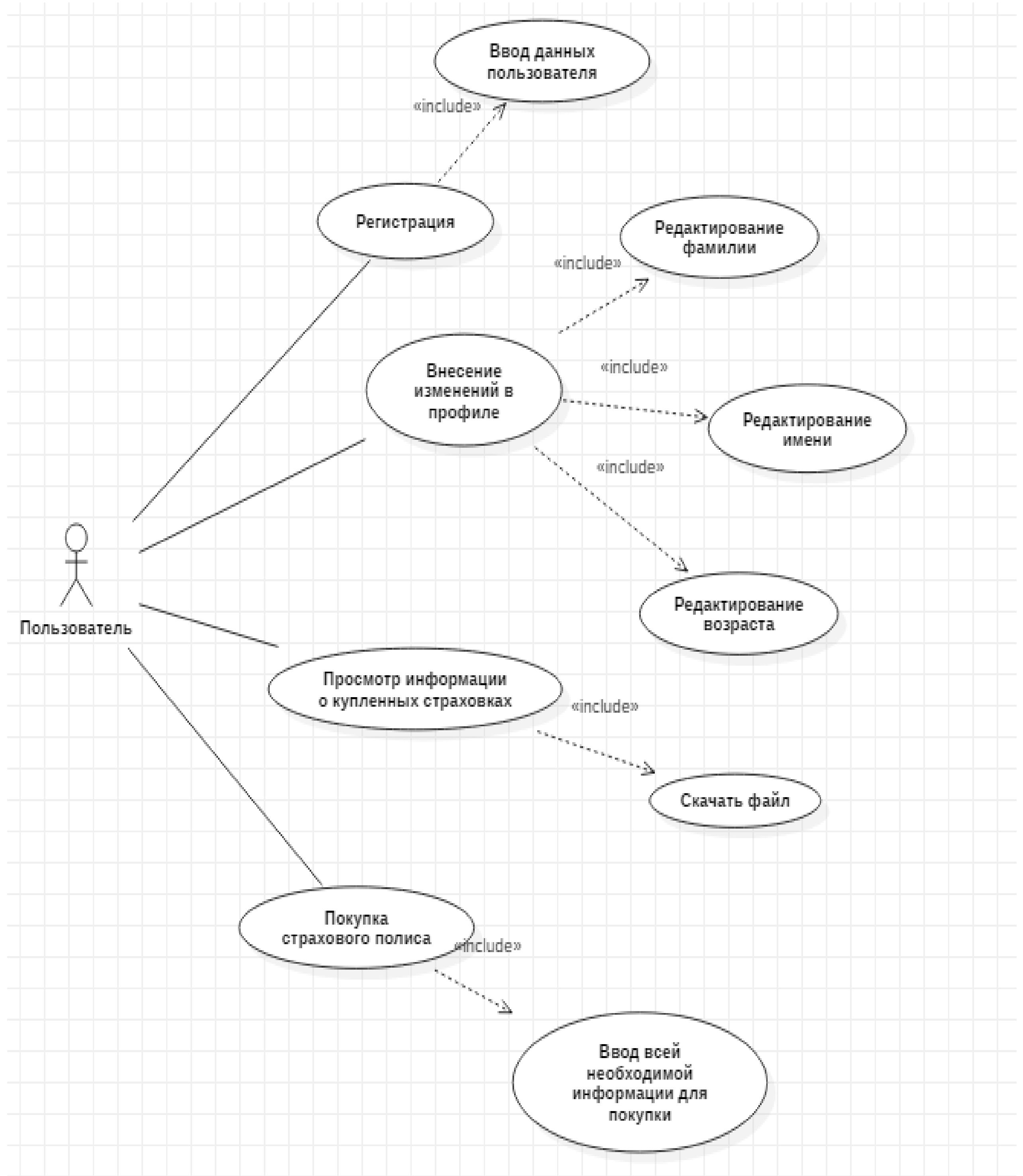

Рисунок 3.1 – Диаграмма вариантов использования

Пользователю предоставляется возможность изменения своих данных таких как: возраст, имя и фамилия. Эти данные используются в варианте использования «Покупка страхового полиса», тем самым это позволяет сэкономить время при покупке полиса. Так же он может просмотреть список уже купленных страховых полисов и купить новый.

Далее была спроектирована диаграмма сущностных классов для того чтобы выделить основные сущности для работы системы.

«Класс-сущность (entityclass) - объекты сущностных классов представляют собой блоки длительно хранимой информации, используемые для организации баз данных и знаний, файловых систем хранения, данных различной логической структуры; в основном в этих классах развит атрибутный раздел, однако имеется небольшое число операций контроля ограничений целостности, как стандартных, так и специфичных для данной предметной области» [13].

Диаграммы классов предлагают ряд преимуществ, что позволяет использовать для следующих задач:

– изображение модели данных для информационных систем, независимо от того, насколько они просты или сложны;

– лучшего понимания программной архитектуры приложения;

– визуально выражения любых специфических потребностей системы;

– создания подробных диаграммы, для выделения классов, которые необходимо реализовать в описанной структуре;

– предоставить независимое от реализации описание типов, используемых в системе, которые впоследствии передаются между ее компонентами.

Диаграмма классов в первую очередь предназначена для разработчиков, чтобы обеспечить концептуальную модель и архитектуру разрабатываемой системы. Как правило, диаграмма классов состоит из более чем одного класса или всех созданных классов для системы.

На рисунке 3.2 представлена диаграмма сущностных классов разрабатываемой системы.

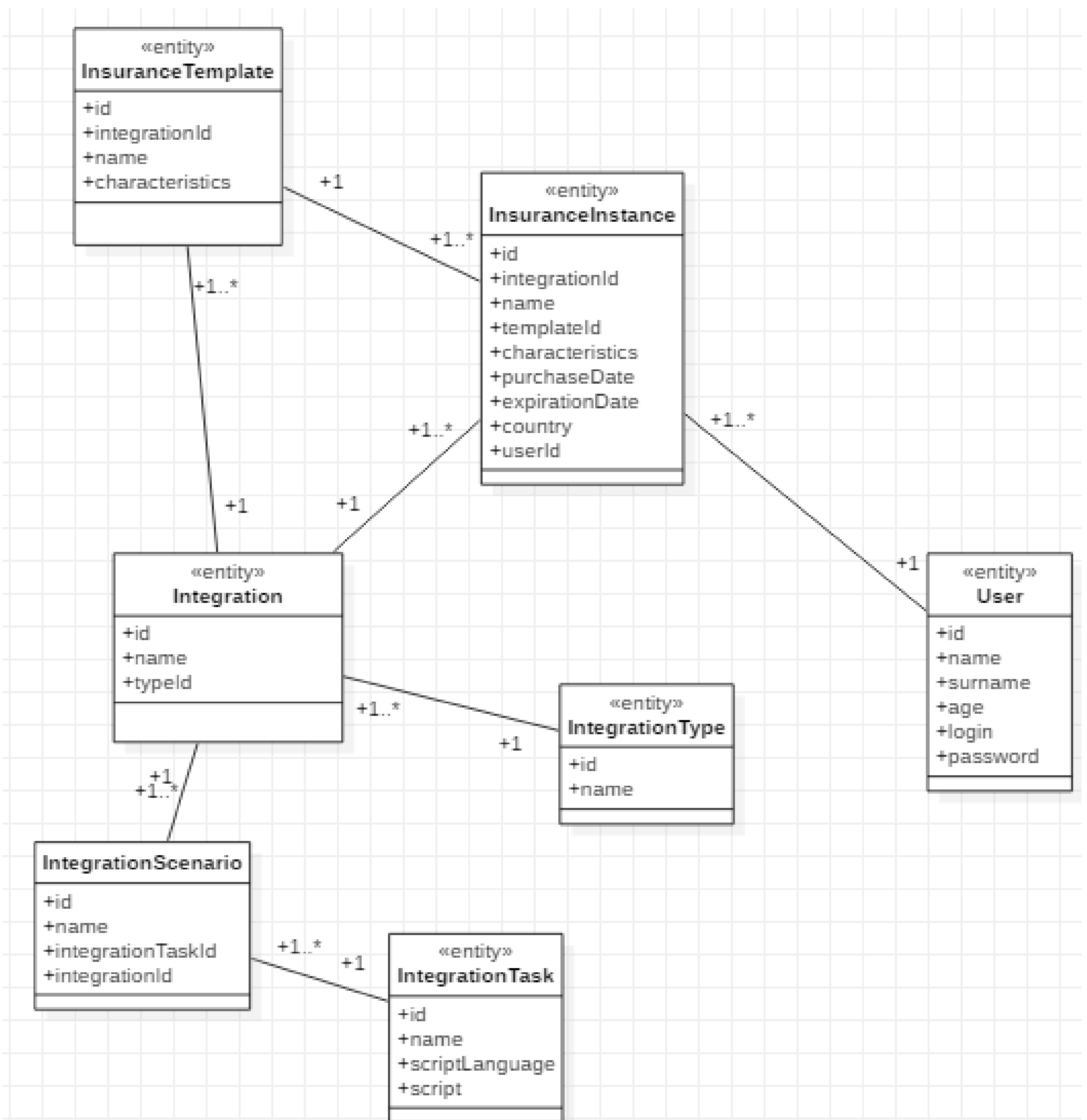

Рисунок 3.2 – Диаграмма сущностных классов

В системе были выделены следующие сущности: «InsuranceTemplate», «InsuranceInstance», «intagration», «IntegrationType», «IntegrationTask», «User».

Сущность «InsuranceTemplate» представляет из себя шаблон покупаемого продукта, который передается на клиента системы и на основе этого шаблона строится страница покупки страхового полиса.

Когда пользователь покупает продукт, создается сущность «InsuranceInstance», в которую записываются все характеристики продукта, которые пользователь выбрал в процессе покупки.

Каждая сущность «InsuranceInstance» или «InsuranceTemplate» содержит ссылку на сущность «Integration», от которой будет зависеть как тот или иной продукт будет интегрироваться с корпоративной информационной системой.

Сущность «Integration» связан с сущностью «IntegrationType», на основе которой будет определяться каким образом будет выполнятся тот или иной сценарий, это может быть REST запросы или SOAP запросы. Данное решение позволит системе быть вариативной в зависимости от того какие интерфейсы имеет корпоративная информационная система с которой происходит интеграция. «IntegrationScenario» определяет сценарий, который будет выполнятся, например, сценарий покупки или сценарий расчета стоимости. «IntegrationTask» описывает сущность которая содержит выполняемый скипт.

Далее была спроектирована диаграмма компонентов. На данной диаграмме изображаются типы компонентов и зависимости между ними.

Цель диаграммы компонентов - показать взаимосвязь между различными компонентами в системе. Для целей UML 2.0 термин «компонент» относится к модулю классов, которые представляют независимые системы или подсистемы с возможностью взаимодействия с остальной частью системы.

Существует целый подход к разработке, который вращается вокруг компонентов: разработка на основе компонентов. При таком подходе диаграммы компонентов позволяют разработчику идентифицировать различные компоненты и за что будет каждый компонент отвечать.

Чаще всего в подходе объектно-ориентированного программирования диаграмма компонентов позволяет старшему разработчику группировать классы вместе на основе общей цели, чтобы разработчик и другие могли взглянуть на проект разработки программного обеспечения с высокого уровня.

Хотя на первый взгляд диаграммы компонентов могут показаться сложными, они имеют неоценимое значение для разработки вашей системы. Диаграммы компонентов могут помочь вашей команде:

- Представить физическую структуру системы;
- Обратить внимание на компоненты системы и на связь между ними;
- Понять поведение сервиса в части, касающейся интерфейса.

В таблице 3.1 дано краткое описание основных компонентов системы.

Таблица 3.1 - Основные сервисы системы

Сервис	Описание
Insurance Manager Service	Сервис которой отвечает за логику работы с сущностями страховки.
User Service	Сервис для работы с пользователями и отвечает за аутентификацию
Integration Service	Сервис которой отвечает за интеграцию с корпоративной информационной системой
Notification Service	Сервис который отвечает за push-уведомления для пользователей
Insurance DB	База данных которая содержит всю информацию о страховках
User DB	База данных содержащая информацию о пользователях системы
Integration DB	База данных которая содержит данные о интеграциях, сценариях интеграций и тасках.
Notification DB	База данных содержащая уведомления отправляемые пользователям.

Диаграмма компонентов разработанной системы для интеграции с КИС приведена на рисунке 3.3, она отражает компоненты системы-сервисы и связи между ними.

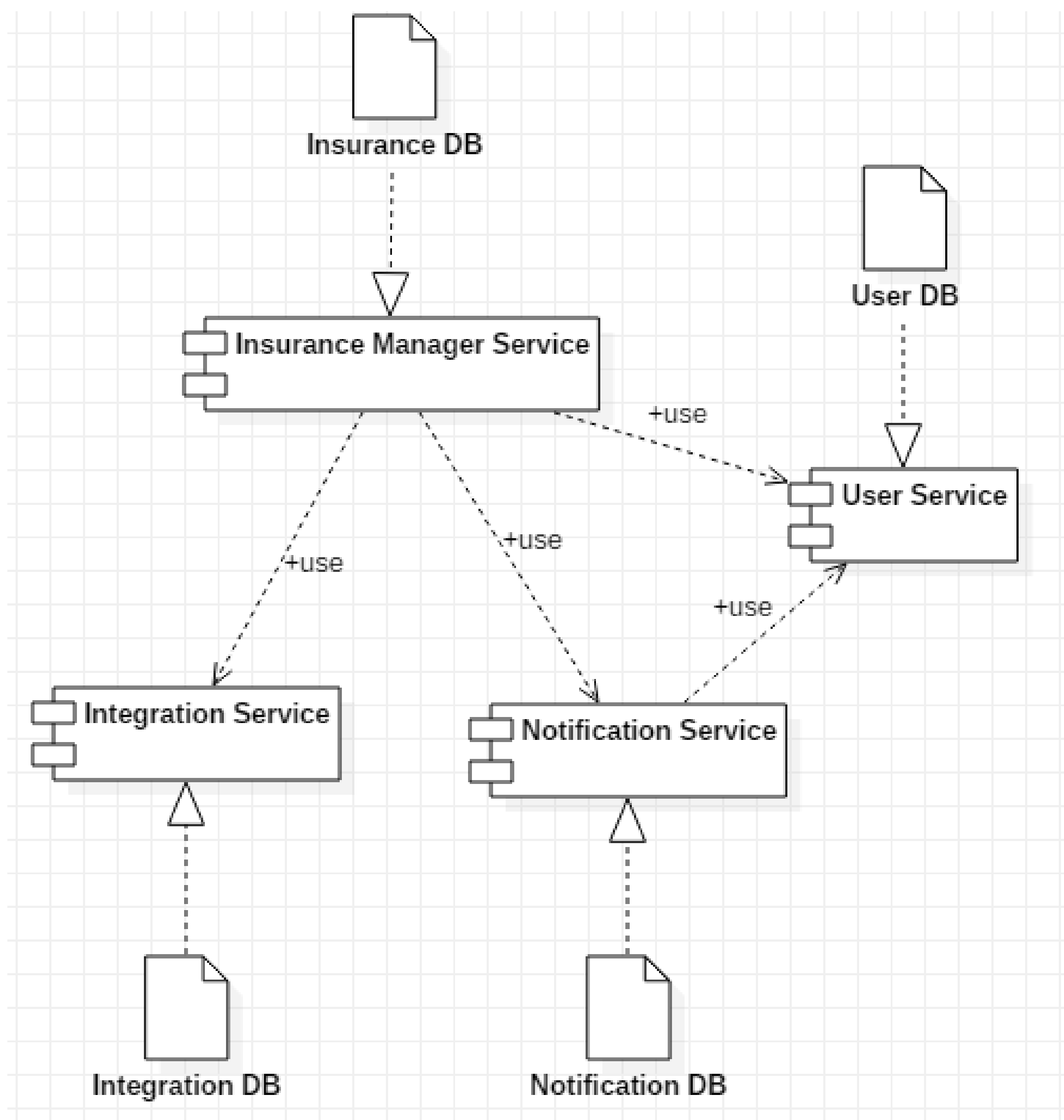

Рисунок 3.3 – Диаграмма компонентов разработанной системы для интеграции с КИС

В качестве языка для разработки серверной части был выбран Java. «Java – язык программирования, разработанный компанией Sun Microsystems» [14]. Приложения Java обычно компилируются в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине (JVM) независимо от компьютерной архитектуры. «Дата официального выпуска - 23 мая 1995 года. Сегодня технология Java предоставляет средства для превращения статических

Web-страниц в интерактивные динамические документы и для создания распределенных не зависящих от платформы приложений» [14].

Часто к недостаткам концепции виртуальной машины относят то, что исполнение байт-кода виртуальной машиной может снижать производительность программ и алгоритмов, реализованных на языке Java. В последнее время был внесен ряд усовершенствований, которые несколько увеличили скорость выполнения программ на Java:

– применение технологии трансляции байт-кода в машинный код непосредственно во время работы программы (JIT-технология) с возможностью сохранения версий класса в машинном коде;

– широкое использование платформенно-ориентированного кода (native-код) в стандартных библиотеках;

– аппаратные средства, обеспечивающие ускоренную обработку байт-кода.

Для реализации системы был выбран язык Java по следующим причинам:

– кроссплатформенный язык;

– автоматическое освобождение памяти;

– объектная ориентированность;

– удобство работы с потоками;

– java virtual machine может запускать Java байт код, скомпилированный из других языков программирования;

– поддержка функционального программирования.

В качестве СУБД выбран MongoDB в силу того, что информациях хранится в JSON документах, это позволяет в дальнейшем без усилий добавлять атрибуты на сущности, а также кластерной масштабируемости этой базы данных. А также отсутствие сложных JOIN запросов и нет необходимости маппинга объектов приложения в объекты БД.

Так же была разработана диаграмма последовательности. Диаграмма последовательности представляет собой тип диаграммы взаимодействия, потому

что она описывает, как и в каком порядке группа объектов работает вместе. Эти диаграммы используются разработчиками программного обеспечения и бизнес-специалистами для понимания требований к новой системе или для документирования существующего процесса. Диаграммы последовательности иногда называют диаграммами событий или сценариями событий.

На рисунке 3.4 представлена диаграмма последовательности для случая расчета стоимости страховки. Такая же последовательность взаимодействия сервисов будет и для других сценариев таких как покупка и т.д.

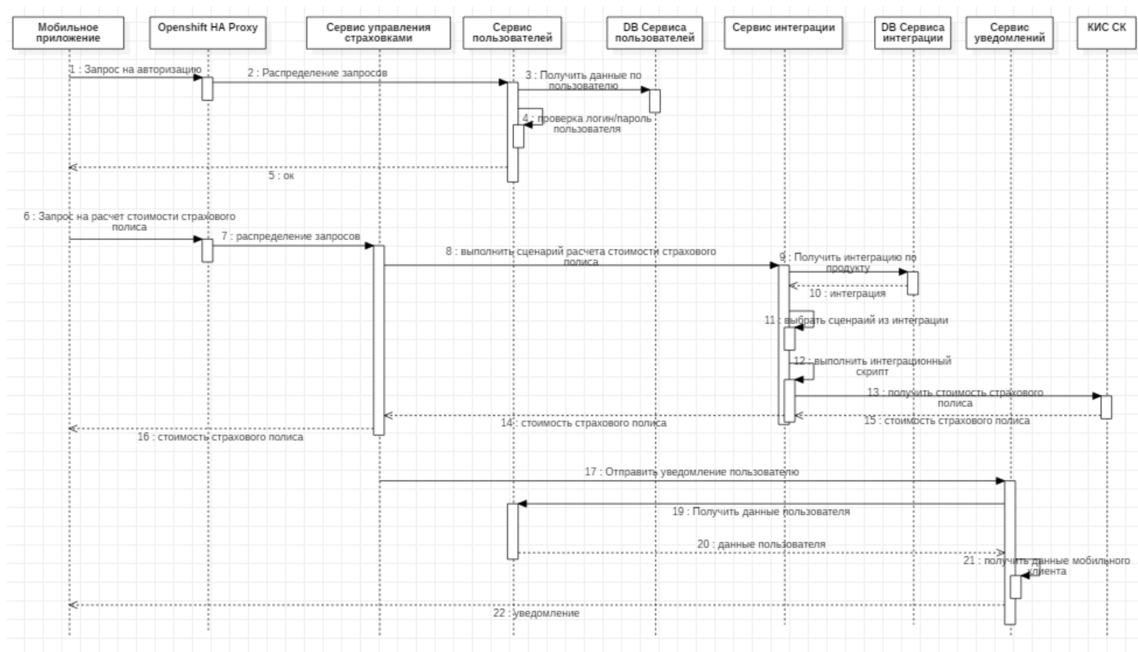

Рисунок 3.4 – Диаграмма последовательности расчета стоимости страхового полиса при выборе различных опций

Для того чтобы сервер мог передать какую-либо информацию клиенту, в качестве бизнес информации можно привести пример индивидуального предложения, необходимо поддержать технологию push-уведомлений. Технология push-уведомлений позволяет доставлять информацию из приложения на мобильное устройство или настольный компьютер без специального запроса из клиентского приложения, что означает, что это приложение не нужно запускать для получения Push-уведомления. С тех пор, как

эта технология была впервые представлена 10 лет назад, появилось несколько платформ для отправки Push-уведомлений.

Была выбрана платформа Firebase. Она содержит в себе сервис Firebase Cloud Messaging (FCM). FCM - это бесплатное кроссплатформенное решение для обмена сообщениями, которое позволяет отправлять push-уведомления клиентским приложениям, не беспокоясь о коде сервера. Используя FCM вместе с Firebase Notification Composer можно создавать уведомления, предназначенные для очень специфических разделов вашей пользовательской базы, часто без необходимости написания какого-либо специального кода [15].

FCM поддерживает два типа сообщений:

– сообщения уведомления. Клиентское приложение будет вести себя по-разному в зависимости от того, находится ли оно в фоновом режиме или на переднем плане, когда оно получает сообщение из FCM. Если приложение работает в фоновом режиме, то Firebase SDK автоматически обработает сообщение и отобразит его в виде уведомления в системном трее устройства;

– сообщения с данными. В отличие от уведомлений, можно использовать сообщения с данными для отправки пользовательских данных в клиентское приложение, но объем такого сообщение ограничен.

На рисунке 3.5 показана консоль сервиса уведомлений.

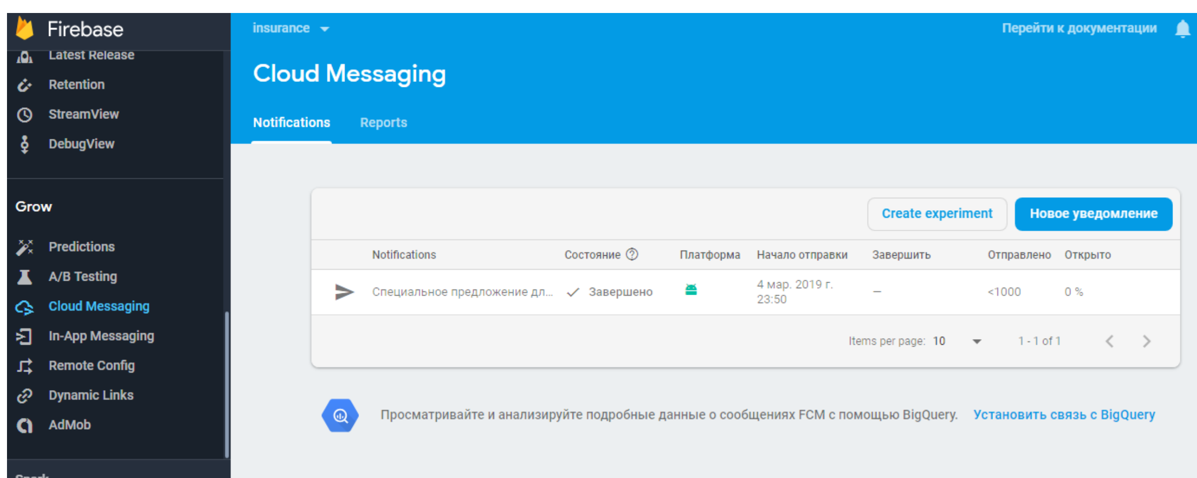

Рисунок 3.5 – Консоль Firebase Cloud Messaging

В разделе уведомлений показаны настроенные уведомления, которые будут отправляться клиентам. Так же эти уведомления могут быть таргетированными, другими словами они будут отправляться определенному кругу пользователей.

3.2 Разработка мобильного приложения продажи страховых полисов

В качестве платформы для разработки мобильного приложения была выбрана платформа Android. Операционная система Android была разработана компанией Android, Inc., которая в 2005 году была приобретена компанией Google. В ноябре 2007-го был сформирован консорциум Open Handset Alliance. В задачи этого консорциума входит разработка, сопровождение и развитие Android, внедрение инноваций в мобильных технологиях, а также повышение удобства работы с устройствами Android при одновременном снижении затрат.

Одно из главных преимуществ платформы Android - ее открытость. Операционная система Android построена на основе открытого исходного кода и находится в свободном распространении. Это позволяет разработчикам получить доступ к исходному коду Android и понять, каким образом реализованы свойства и функции приложений. Любой пользователь может принять участие в совершенствовании операционной системы Android. Для этого достаточно отправить отчет об обнаруженных ошибках либо принять участие в одной из дискуссионных групп Open Source Project. В Интернете доступны различные приложения Android с открытым исходным кодом, предлагаемые компанией Google и рядом других производителей [16].

Открытость платформы способствует быстрому обновлению. В отличие от закрытой системы iOS компании Apple, доступной только на устройствах Apple, система Android доступна на устройствах десятков производителей оборудования (OEM, Original Equipment Manufacturer) и телекоммуникационных компаний по всему миру. Все они конкурируют между собой, что идет на пользу конечному потребителю.

По последним тенденциям разработки мобильных страховых приложений разрабатываемый клиент должно содержать следующие функции [17]:

– профиль пользователя;
– уведомления;
– расчет стоимости;
– безопасная покупка;
– поддержка нескольких языков.

Экран авторизации показан на рисунке 3.6, который позволяет пользователю авторизоваться и работать под своей учетной записью. Так же он позволяет перейти на экран регистрации, если у пользователя еще нет аккаунта. Аутентификация происходит через HTTP Basic Authentication.

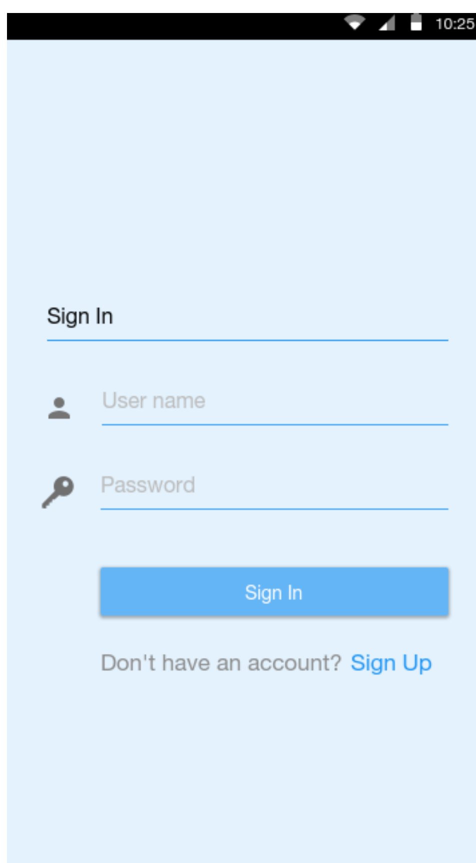

Рисунок 3.6 - Экран авторизации

При регистрации нового аккаунта пользователю необходимо указать логин, пароль и почту, это показано на 3.7. Поля валидируются в соответствии с

их масками, так, например, в поле ввода почты нельзя вводить значения отличные от маски почты.

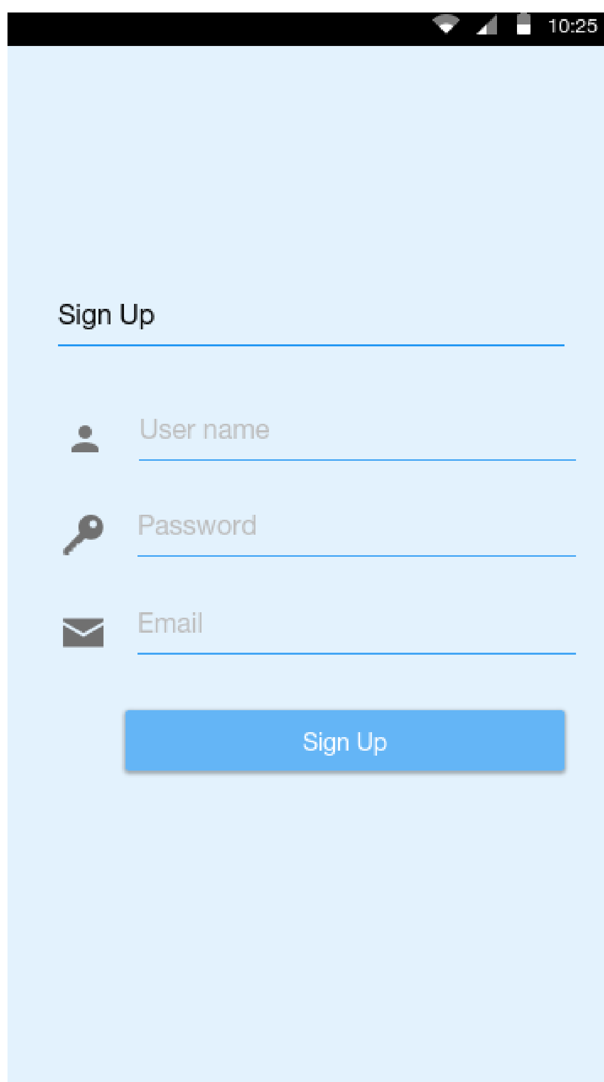

Рисунок 3.7 - Экран регистрации

После регистрации или если у пользователя уже есть аккаунт, ему откроется доступ к главному экрану приложение (рисунок 3.8). Здесь пользователь может выбрать страховую компанию из доступных на данный момент. Так же пользователю доступна поле поиска для быстрого доступа к предложению определенной страховой компании. Информация о доступных страховых предложениях формируется на сервере приложений, в зависимости от присутствия интеграции со страховой компанией. Вся информация о страховом приложение приходит с сервера в виде JSON файла, который формируется в

сервисе страховок и представляет их себя шаблон. При вводе информации в строку поиска, приложение начинает поиск в каталоге после того как введено 3 символа и сравнивает на содержимое.

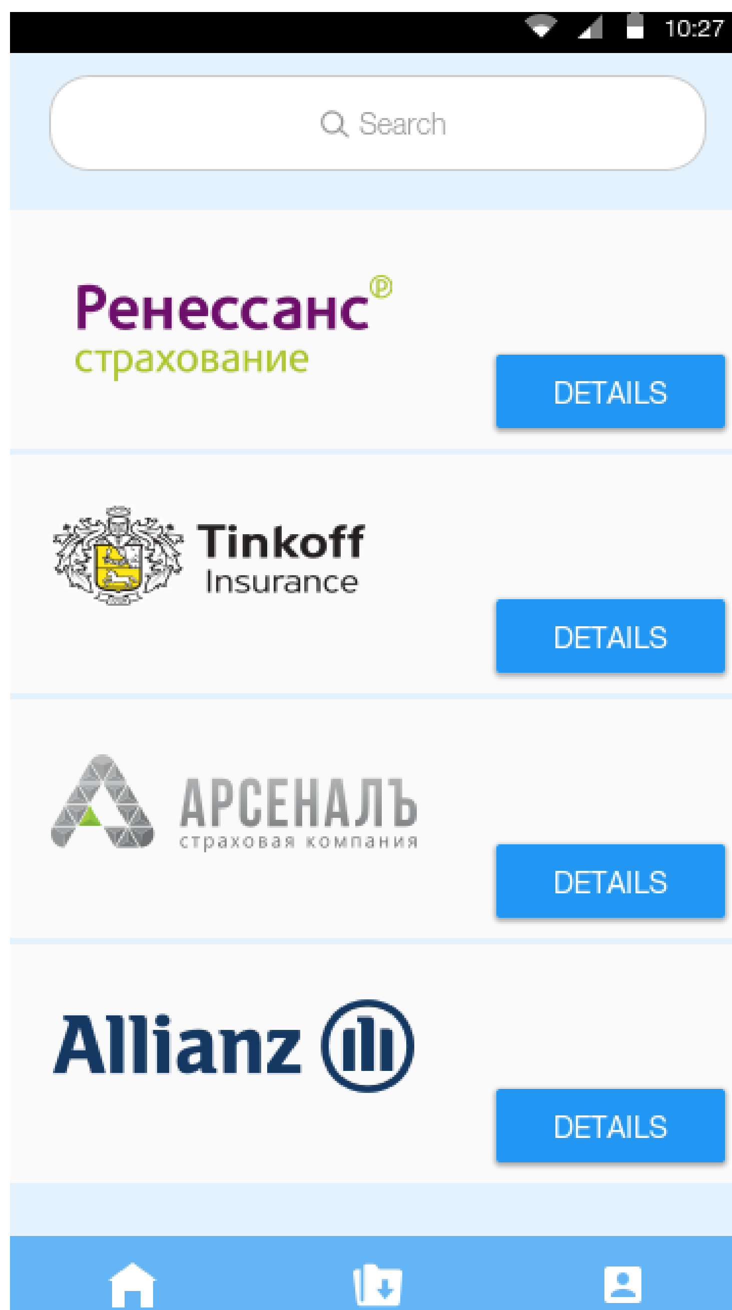

Рисунок 3.8 - Главный экран

Нижняя панель позволяет пользователю перейти в разделы: «Мой профиль» и «Купленные страховки» и «Главный экран». В разделе «Мой профиль» пользователю доступны разделы настроек, информация о приложении и кнопка «Выйти», которая отправит пользователя на экран авторизации, так же

на этом экране в верхней части выводится имя и фамилия пользователя (рисунок 3.9). При навигации в раздел настроек пользователю становится доступен функционал по изменению данных такие как: возраст, фамилию и имя, это видно на рисунке 3.10. При вводе этих данных пользователю стоит учитываться, что они будут использоваться при покупке страхового полиса. Данное решение позволит сохранить время пользователя, что напрямую сказывается на удобстве использования приложения в процессе покупки.

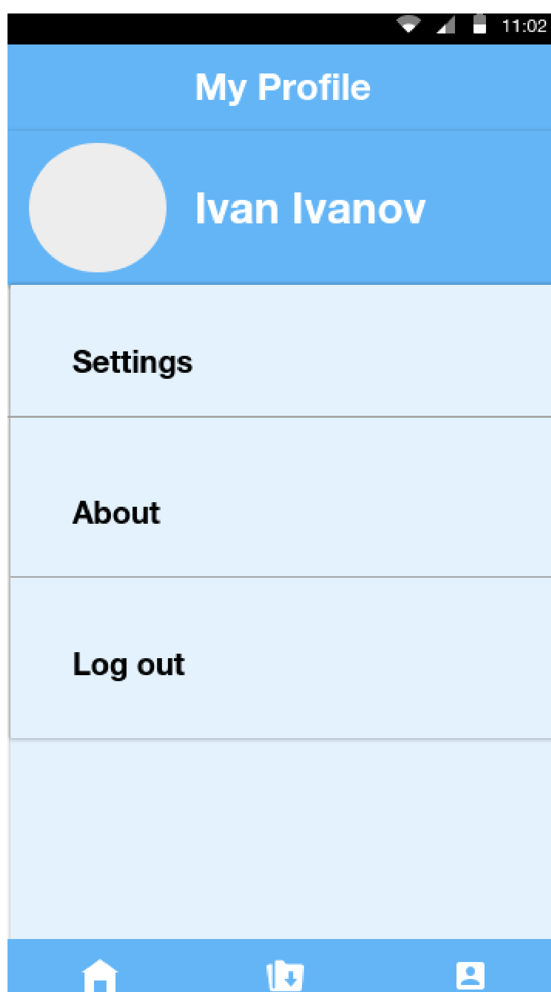

Рисунок 3.9 - Экран «Мой профиль»

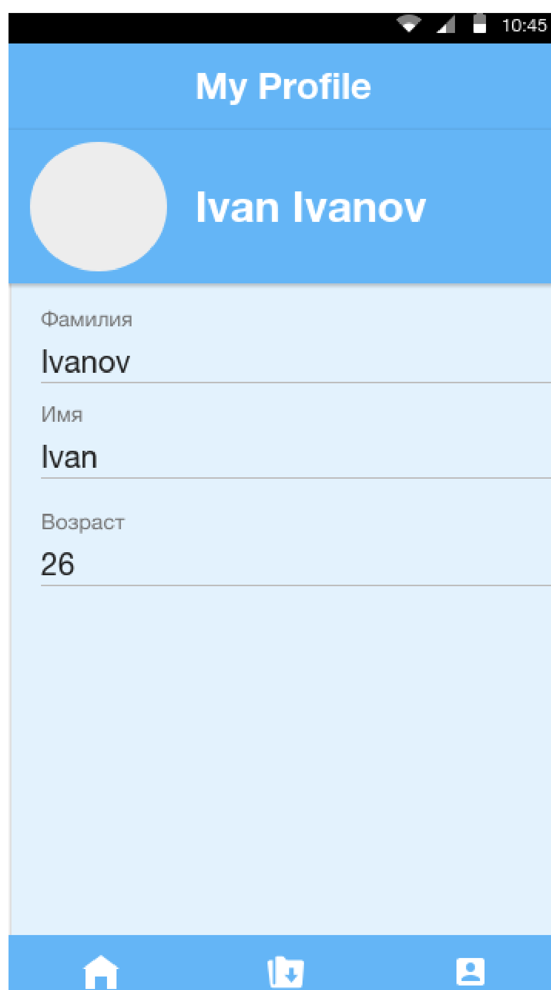

Рисунок 3.10 – Настройки пользователя

В разделе «Купленные страховки» пользователю доступны все его ранее купленные страховки, которые хранятся на сервере приложения и затем при запросе единожды скачиваются на телефон (рисунок 3.11). Здесь можно увидеть номер оформленного страхового полиса и при нажатии на иконку документа, который находится правее номера полиса, документ откроется с помощью выбранного приложения, который зависит от формата документа. Эти документы хранятся во внутренней памяти телефона, в специальной папке. На случай если пользователь удалит эти документы с телефона или удалит приложение, то они останутся на сервере приложений и в последующем у пользователя будет возможность заново скачать эти документы.

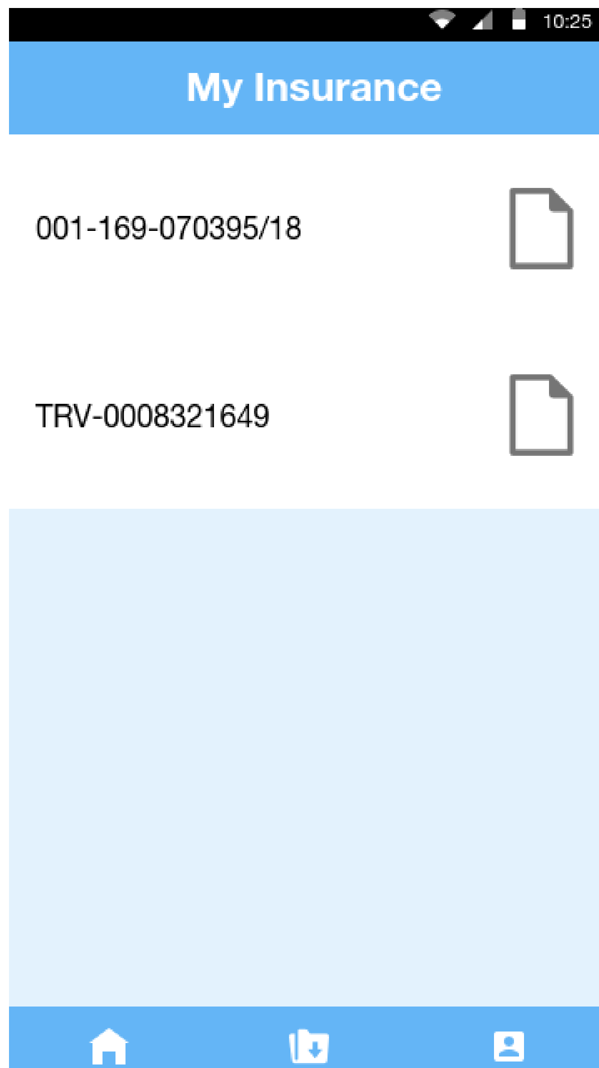

Рисунок 3.11 - Раздел купленных страховок

Когда пользователь определился с компанией, у которой он будет покупать страховку на главной странице приложения, он нажимает на кнопку «Детали». На этом экране пользователь вводит необходимые данные для покупки страховки (рисунок 3.12) такие как:

- страна посещения;
- даты действия страховки;
- сумма страхования;
- дополнительные опции страхования.

Пользователю также доступно изменения валюты путем переключателя рядом с областью выбора суммы страхования.

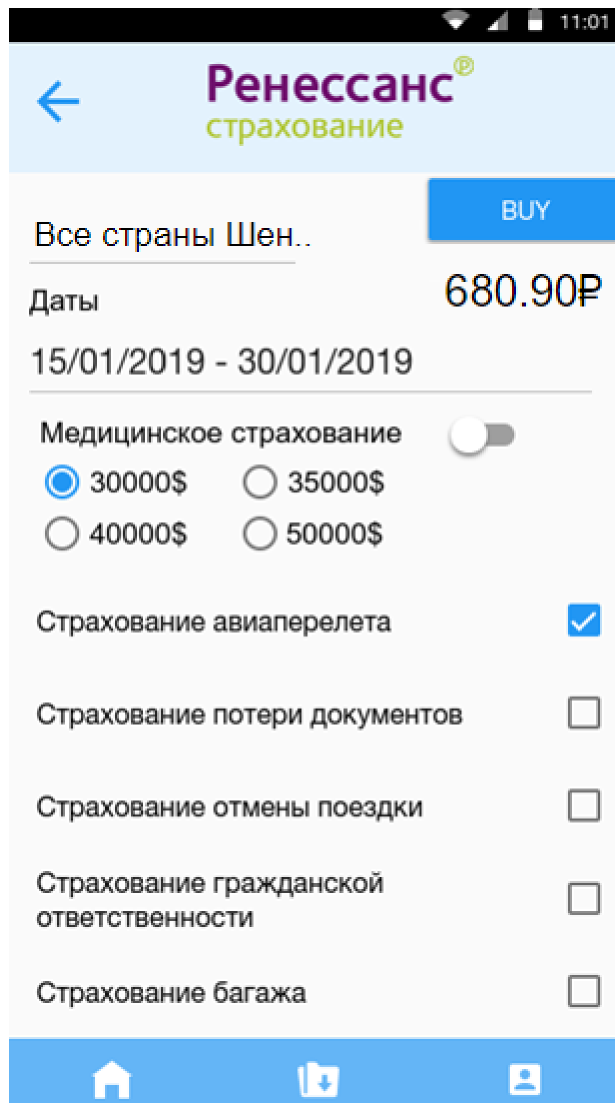

Рисунок 3.12 - Экран покупки страховки

Поля ввода страны посещения и даты поездки являются типовыми и повторяются в любом предложении не зависимо от страховой компании. Дополнительные параметры выводятся в зависимости от характеристик, которые хранятся на шаблоне страхового предложения на сервере. Все эти характеристики влияют на конечную стоимость продукта и выбранные позиции затем при покупке отправятся на сервер вместе с основной информацией, на основе которой сформируется запрос в интегрируемую информационную систему.

Выводы к главе 3

1) Были проанализированы основные сценарии использования мобильного приложения для покупки страховых полисов и спроектирована диаграмма «Вариантов использования».

2) Спроектирована диаграма «Сущностных классов», которая отражает основные сущности системы, диаграмма «Компонентов», которая показывает взаимосвязь основных сервисов, а так же представлена диаграмма «Последовательности», которая отражает модель интеграции мобильного клиента с КИС СК через разработанную информационную систему.

3) Представлены основные экраны и их описание разработанного мобильного приложения для продажи страховых полисов.

ГЛАВА 4 АПРОБАЦИЯ МОДЕЛИ ИНТЕГРАЦИИ МОБИЛЬНОГО ПРИЛОЖЕНИЯ С КОРПОРАТИВНОЙ ИНФОРМАЦИОННОЙ СИСТЕМОЙ СТРАХОВОЙ КОМПАНИИ

4.1 Процесс внедрения системы для интеграции мобильного приложения с корпоративной информационной системой страховой компании

Процесс внедрения информационной системы для интеграции мобильного клиента с корпоративной информационной системой страховой компании предусматривает развертывание системы на веб-сервере для возможности доступа к ее функционалу через интернет.

Развертывание компонентов системы показано на рисунке 4.1.

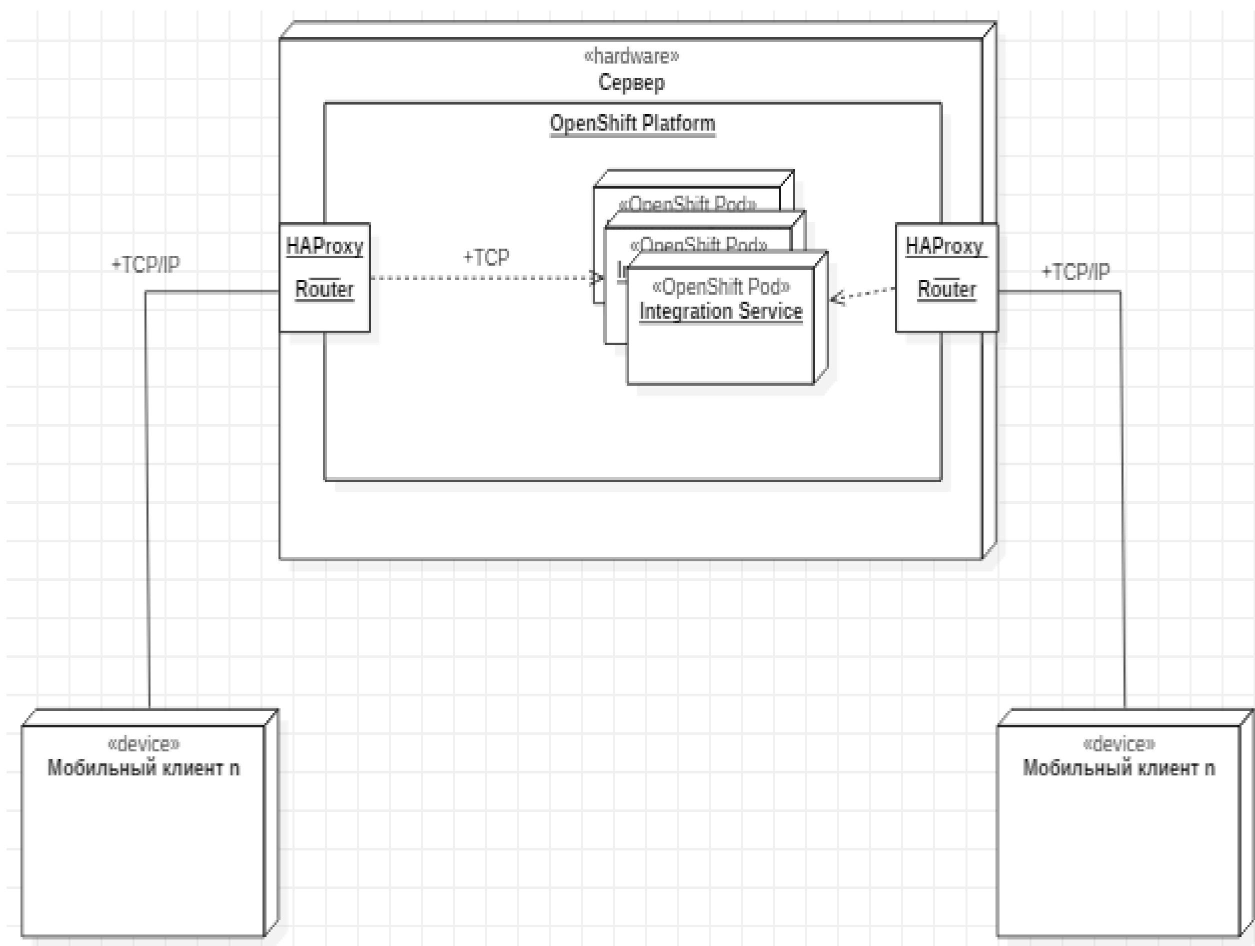

Рисунок 4.1 – Диаграмма развертывания

Диаграмма развертывания - это тип диаграммы UML, который показывает архитектуру выполнения системы, включая такие узлы, как аппаратные или программные среды выполнения, и промежуточное программное обеспечение, соединяющее их.

Диаграммы развертывания обычно используются для визуализации физического оборудования и программного обеспечения системы. Используя его, вы можете понять, как система будет физически развернута на оборудовании. Диаграммы развертывания помогают моделировать топологию оборудования системы по сравнению с другими типами диаграмм UML, которые в основном описывают логические компоненты системы [19].

Разработанная информационная система имеет следующие узлы: сервер и мобильные приложения.

На сервере системы развернуто решение OpenShift Platform, где подняты поды для каждого микросервиса, каждый из который представляет собой Docker образ. Docker образ - это файл, состоящий из нескольких слоев, используемый для выполнения кода в контейнере Docker. Образ по существу построен из инструкций для полной и исполняемой версии приложения, которое опирается на ядро операционной системы. Когда Docker запускает образ, оно становится одним или несколькими экземплярами этого контейнера. Docker - это программная платформа для виртуализации на уровне операционной системы с открытым исходным кодом, в первую очередь разработанная для Linux и Windows. Docker использует функции изоляции ресурсов ядра ОС, такие как cgroups в Linux, для запуска нескольких независимых контейнеров в одной ОС. Контейнер, который перемещается из одной среды Docker в другую с той же ОС, будет работать без изменений, поскольку образ включает в себя все зависимости, необходимые для выполнения кода.

Каждый сервис представлен на вкладке Pods системы Openshift. На данном окне представлено состояние сервиса и его имя. Скриншот вкладки представлен на рисунке 4.2.

Pods Learn More

Filter by label | Add

Name	Status	Containers Ready	Containe
insurance-manager-service	Running	1/1	0
integration-service	Running	1/1	0
user-service	Running	1/1	0
norification-service	Running	1/1	0

Рисунок 4.2 – Список сервисов системы

После того как система развернута ее необходимо протестировать с помощью функциональное тестирование. Функциональное тестирование в основном используется для проверки того, что часть программного обеспечения выдает результат, который требуется конечному пользователю или бизнесу. Как правило, функциональное тестирование включает оценку и сравнение каждой функции программного обеспечения с требованиями бизнеса. Программное обеспечение тестируется путем предоставления ему некоторых входных данных, чтобы можно было оценить выход, чтобы увидеть, как оно соответствует, связано или изменяется по сравнению с его базовыми требованиями [21]. Кроме того, функциональное тестирование также проверяет программное обеспечение на удобство использования. Некоторые методы функционального тестирования включают интеграционное тестирование, тестирование белого ящика, тестирование черного ящика, модульное тестирование (unit тестирование), приемочное тестирование и т.д. Система тестировалась с помощью модульного тестирования, интеграционного тестирования и системного тестирования. Последовательность тестирование показана на рисунке 4.3.

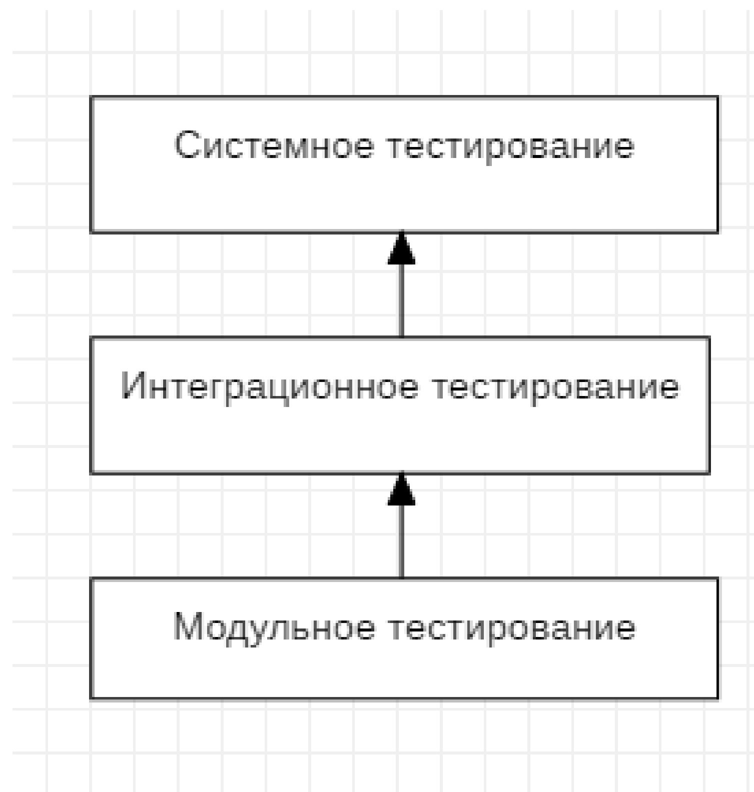

Рисунок 4.3 – Этапы тестирования

Модульное тестирование - это уровень тестирования программного обеспечения, на котором тестируются отдельные модули / компоненты программного обеспечения [21]. Цель состоит в том, чтобы проверить, что каждая единица программного обеспечения работает так, как задумано. Единица является самой маленькой тестируемой частью любого программного обеспечения, в данном случае минимальной частью является Java-класс. В рамках модульного тестирования были покрыты тестами большая часть классов. Это позволит на этапе компиляция отлавливать ошибки, которые были внесены после исправлений или после разработки нового функционала.

Следующим этапом было разработать интеграционные тесты. На этом уровне тестирования программного обеспечения отдельные модули объединяются и тестируются в группе. Целью данного уровня тестирования является выявление ошибок во взаимодействии между интегрированными модулями. В данном случае точкой входа для тестирования являлись эндпоинты REST-контроллеров каждого микросервиса. Тестирование каждого эндпоинта

позволило отловить несколько ошибок при работе с базой данных. В качестве платформы для написания интеграционных тестов была выбрана Arquillian.

Arquillian - это платформа для интеграционного тестирования, которая была разработана специально для Java EE приложений. Главным его преимуществом является легкость и быстрота написания тестов, как если бы это были модульные тесты [22].

Последним этапом было проведение системного тестирование, в рамках которого уже проверялось межсервисное взаимодействие. Оно включало в себя следующие:

– тестирование полностью интегрированных приложений, включая внешние периферийные устройства, чтобы проверить, как компоненты взаимодействуют друг с другом и с системой в целом. Это также называется End to End тестированием;

– проверка что приложение выдает желаемый результат на выходе.

Необходимо было проверить следующие сценарии:

– регистрация пользователя;

– аутентификация пользователя;

– корректное создание шаблонов страховок;

– создание интеграции со сценариями и интеграционными тасками;

– проверка покупки нового продукта, что все полученные характеристики получены и правильно обработаны;

– проверка правильности выполнения сценариев интеграции, а именно компиляции и выполнения интеграционных тасок.

В результате системного тестирования были выявлены ошибки в сценарии выполнения интеграционных скриптов, а именно, не всегда компилировались скрипты для выполнения. Ошибка была найдена в механизме компиляции скриптов в сервисе интеграция. Причиной данной ошибки стала ошибка в реализации.

Далее было решено провести нагрузочное тестирование. Нагрузочное тестирование проверяет оперативность, стабильность, масштабируемость, надежность, скорость и использование ресурсов вашего программного обеспечения и инфраструктуры. Различные типы тестов производительности предоставляют вам разные данные для анализа. Нагрузочный тест был написан на JMeter. В результате тестирования было выявлено долгое выполнение интеграционных сценариев при большой загрузке сервера приложений. График времени ответа на запрос представлен на рисунке 4.4.

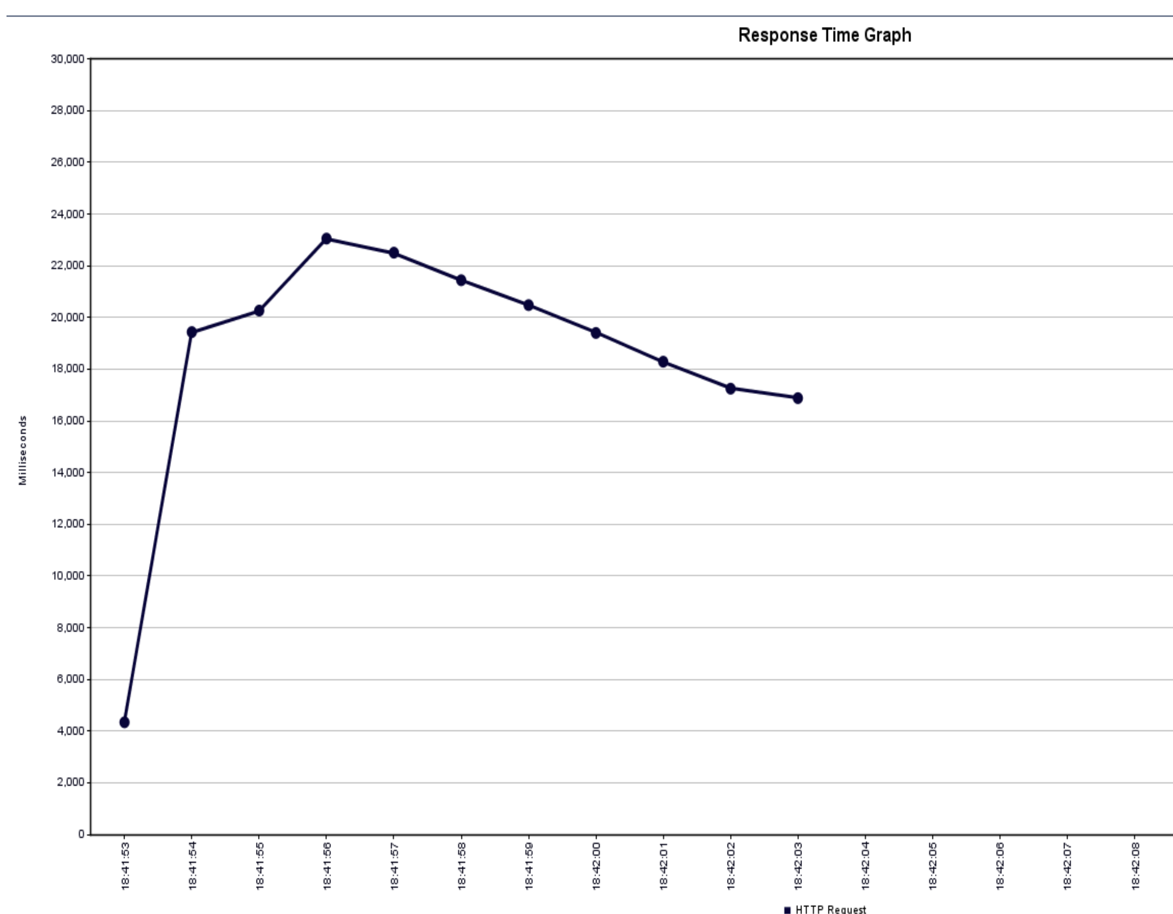

Рисунок 4.4 – График времени ответа

Эта проблема была связана с тем, что интеграционные скрипты каждый раз компилировались и не важно менялись ли они ранее. Для решения этой проблемы был реализован механизм кеширование, для этого была подключена библиотека Google Guava в которую входит модуль Guava Cache.

Guava Cache - это инкрементный кеш, в том смысле, что когда запрашивается объект из кеша, он проверяет, имеет ли он уже соответствующее значение для предоставленного ключа. Если это так, он просто возвращает его (при условии, что срок его действия не истек). Если у него еще нет значения, он использует CacheLoader для извлечения значения, а затем сохраняет значение в кэше и возвращает его. Таким образом, кеш увеличивается по мере запроса новых значений [23].

После того как был внедрен механизм кеширования, был снова проведен нагрузочный тест чтобы потвердеть что механизм кеширование работает, и причина проблемы была поставлена правильно. На рисунке 4.5 представлен тот же самый график только после внесения изменений.

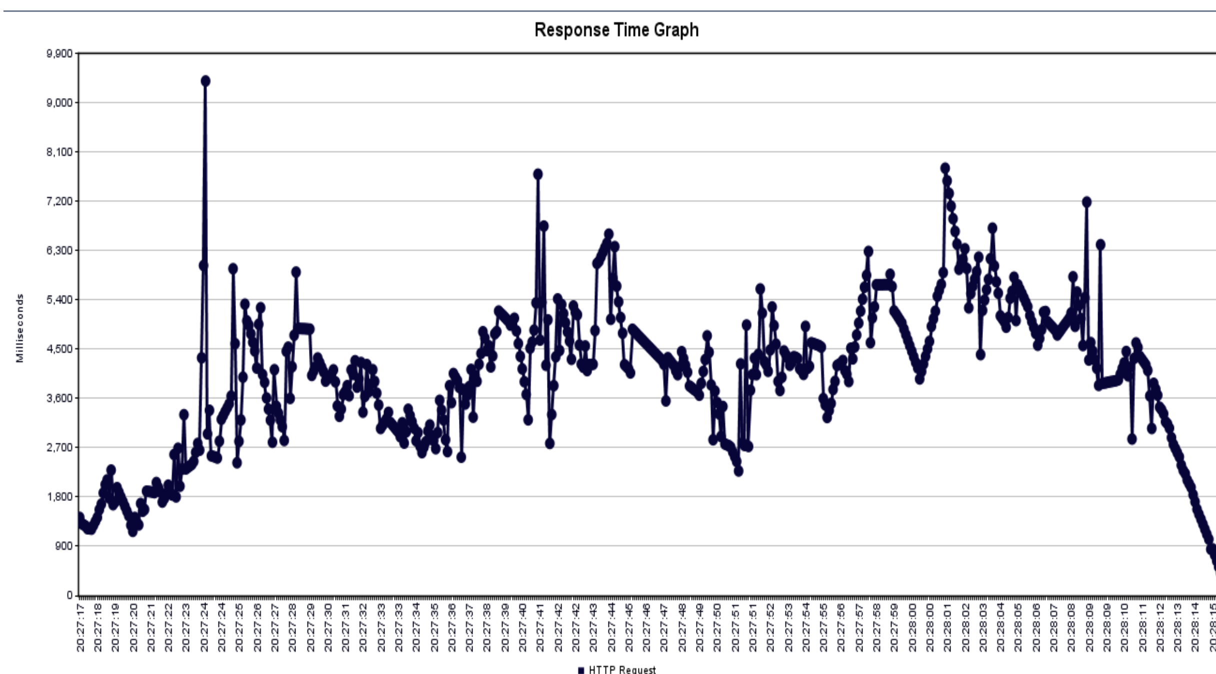

Рисунок 4.5 - График времени ответа после исправлений

Из данного графика можно сделать вывод что среднее время ответа удалось уменьшить в 4 раза и тем самым была достигнута поставленная цель на данном этапе.

4.2 Процесс интеграции мобильного приложения с корпоративной информационной системой страховой компании

В рамках процесса интеграции сначала разработчику необходимо создать шаблон страхового предложения, который будет отображаться в мобильном клиенте и позволит пользователю купить его. Так как на данном этапе пока что нет UI интерфейса для сервиса Insurance Manager Service, который позволил бы в более удобной форме сконфигурировать этот шаблон и сохранить его в базе микросервиса, это было сделано вручную и шаблон был залит на сервер используя Postman (Рисунок 4.6).

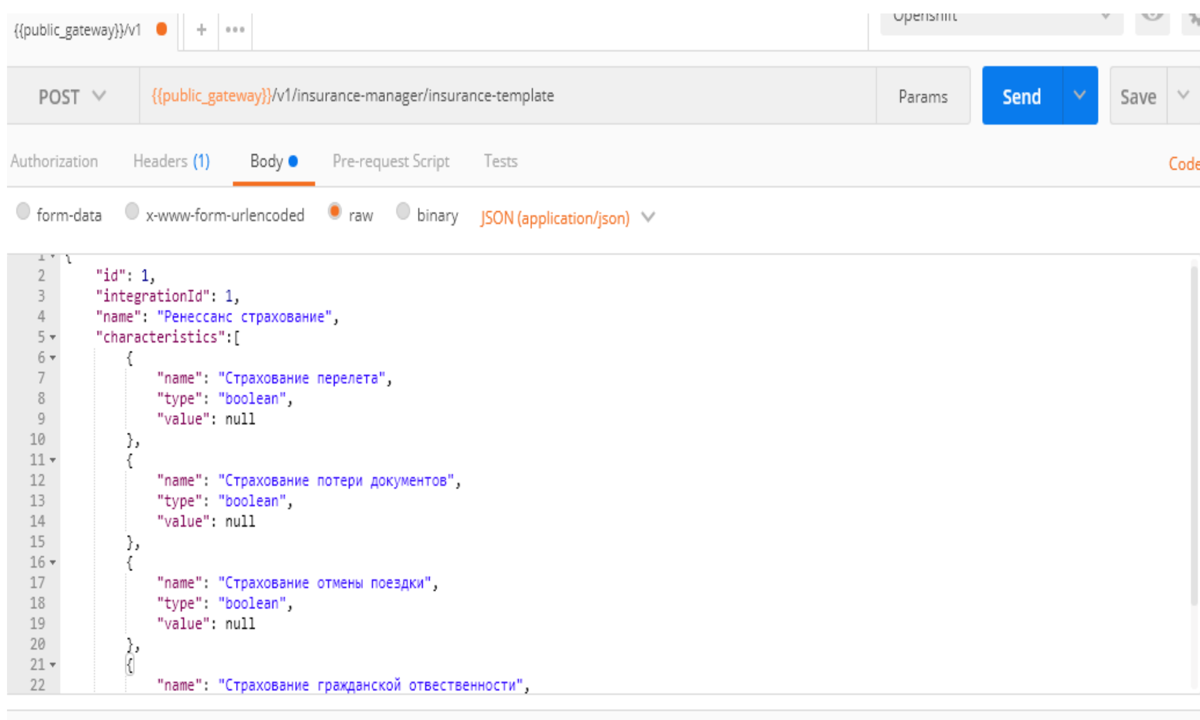

Рисунок 4.6 – Создание шаблона страхового предложения

После того как шаблон создан необходимо создать интеграцию для данного предложения. В рамках создания интеграции необходимо создать сценарии и интеграционные таски. Созданная интеграция представлена на рисунке 4.7. Тип интеграции с id равным 1 уже есть в базе и является интеграция через Groovy скрипты.

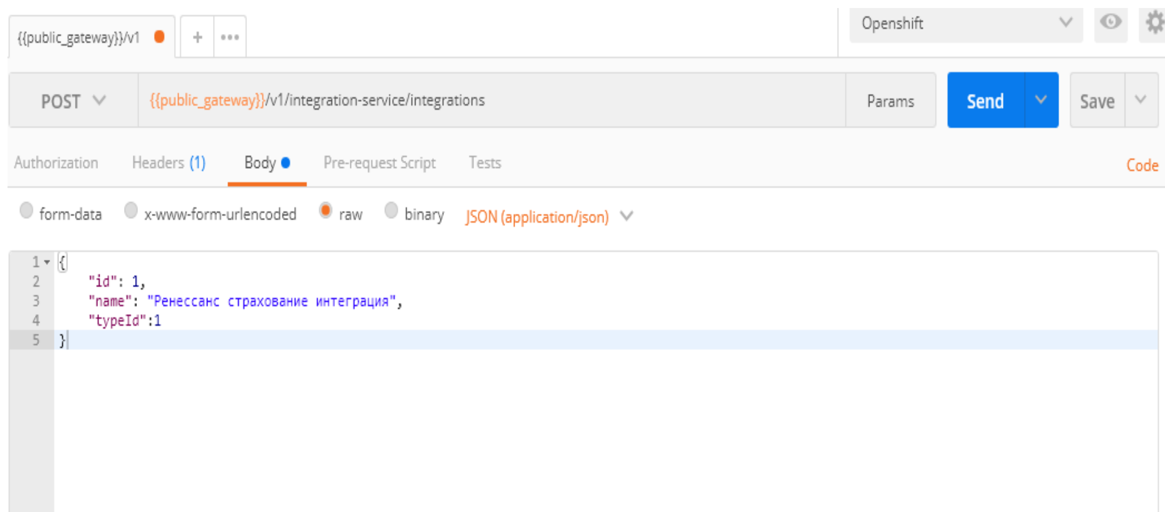

Рисунок 4.7 – Создание интеграции

Были созданы так же сценарии для интеграция такие как «Покупка», «Расчет стоимости полиса» и т.д. Написаны интеграционные скрипты на Groovy. Был выбран этот язык, т.к. его проще использовать разработчикам, которые работали с Java и еще одна причина, это потому что вся система написана на Java. Этот язык является одновременно и статически и динамически типизированным что позволяет использовать его как программный язык, так и скриптовый. Написанные скрипты представлены на рисунке 4.8 и на рисунке 4.9.

```
19 def executeScenario(context) {
20
21     info("Poll Task Started")
22
23     info("Purchase: $context.insuranceInstance")
24
25
26     AbstractIntegrationAPI integrationAPI = AbstractIntegrationAPI.init(context)
27     try {
28         integrationAPI.configRetry(context)
29         integrationAPI.authorize(context)
30
31         if (context.insuranceInstance.characteristics.subscriptionId) {
32             info(". SubscriptionId:" + context.offeringInstance.characteristics.subscriptionId);
33             try {
34                 def subscription = integrationAPI.getSubscription(context.insuranceInstance.characteristics.subscriptionId
35                 fillSubscriptionInfo(context, subscription)
36             } catch (RetryException e) {
37                 error("Unable to resolve subscription by subscriptionId: ${context.insuranceInstance.characteristics.subsc
38                 context.ScriptStatus = "In Progress";
39                 return context;
40             }
41
42             def status = integrationAPI.getProvisioningStatus(context, context.customer.characteristics.customerId, contex
43             if ('success'.equalsIgnoreCase(status)) {
44                 context.ScriptStatus = "OK";
45             } else {
46                 context.ScriptStatus = "In Progress";
47             }
48             return context;
49         }
50
51         def retryTemplate = retryTemplate(context)
52         return retryTemplate.execute({ ctx ->
53             def offer = integrationAPI.getOffering(context, context.insuranceInstance);
54             propagateOfferInfo(context, offer)
55             if (context.insuranceInstance.characteristics.categoryKey == "Education_Key" && integrationAPI.checkCustomerQu
56                 integrationAPI.defineCustomerQualification(context, context.customer.characteristics.customerId)
57             }
58             if (!context.offeringInstance.characteristics.subscriptionId) {
59                 String customerId = context.customer.characteristics.customerId
```

Рисунок 4.8 – Скрипт покупки

```
def requiredUrls() {
    return [];
}

def requiredParameters() {
    return [];
}

def executeScenario(context) {

    AbstractIntegrationAPI integrationAPI = AbstractIntegrationAPI.init(context)
    integrationAPI.configRetry(context)

    info("Resume: $context.insuranceInstance;")

    try {
        integrationAPI.authorize(context)
        calculateInsurancePrice(context, integrationAPI);
        context.ScriptStatus = "OK";
    }
    catch (InternalScriptException ex) {
        integrationAPI.setFailure(context, 'Price_calculation', ex)
        //In order for an error-handler to handle and save this exception
        throw new InternalScriptException(ex.message);
    }

    info("Resume result: $context.ScriptStatus")
}

def calculateInsurancePrice(context, AbstractIntegrationAPI integrationAPI) {
    if (context.insuranceInstance.characteristics.id != null) {
        integrationAPI.calculatePrice(context)
    } else {
        context.lineItems.each() { lineItem ->
```

Рисунок 4.9 – Скрипт расчета цены

Написанные интеграционные скрипты были сохранены в базу данных и привязаны к соответствующему сценарию интеграции. Разработанная информационная система интеграции мобильного приложения продажи страховых полисов с корпоративной информационной системой позволит написать интеграцию за короткий период времени, это позволит снизить трудозатраты разработчиков необходимые для интеграции с той или иной системой.

Подсчитаем потенциальные трудозатраты по интеграции с корпоративной информационной системой. Их можно свести к таблице 4.1.

Таблица 4.1 - Тредозатраты на интеграцию с системой

	Разработчик	Старший-разработчик
1	2	3
Создание шаблона	1 час	30 мин
Создание интеграции	10 мин	4 мин

Продолжение таблицы 4.1

1	2	3
Реализация интеграционных скриптов (1 шт)	17 часов	9 часов
Отладка решения	5 часов	2 часа

На основе данных таблицы 4.1 можно сделать вывод что максимальное время необходимое на интеграцию одного сценария равно 24 часам, что является коротким периодом времени между решением интегрироваться с системой и выходом в пользование.

Так как система может быть развернута в облаке, то сравним издержки при классической инфраструктуре и облачной. В таблице 4.2 представлено сравнение издержек.

Таблица 4.2 - Виды расходов для различных инфраструктур

Виды расходов	Собственная инфраструктура	Облачная инфраструктура
1	2	3
Разовые издержки		
Оборудование	Высокие	Нет
Утилизация, демонтаж оборудования	Нет	Средние/Низкие
Обучение персонала	Средние	Низкие
Лицензия на ПО	Высокие	Нет
Периодические издержки		
Аренда облачных сервисов	Нет	Средние
Заработная плата сотрудникам	Высокие	Низкие
Техническая поддержка и обслуживание	Средние	Низкие
Аренда помещения под серверную аппаратную часть	Средние	Нет

По данной таблице можно отметить экономическую эффективность данной архитектуры в том, что она может быть развернут в облаке и быть в качестве платформы для интеграции, это позволит сэкономить серверных ресурсах для компаний, которые захотят воспользоваться данным решением

4.3 Расчет надежности разработанной информационной системы

Одним из критериев качества разрабатываемой информационной системы является степень ее надежности. Существует немало примеров, когда разработанная система сбоила и это приводило к большим потерям компании. Для этого необходимо произвести оценку надежности разработанной системы.

Существующие модели оценки надежности системы объединены в три большие группы:

- Непрерывные динамические модели (вовремя тестирование ошибки не исправляют);
- Дискретные динамические модели (ошибки устраняются);
- Статистические модели (не учитывается время появления сбоя).

Остановим свой выбор на дискретных динамических моделях, так как они позволяют в процессе тестирования устранять ошибки. В качестве такой модели выберем модель Шумана [24]. Она основана на том, что оценка надежности информационной системы проводится на основании проведенного тестирования. Тестирование проводится поэтапно. По завершению этапа все выявленные проблемы исправляются и начинается следующий этап тестирования.

Было проведено 4 этапа тестирования. Первый этап продолжался 7 часов, затем второй – 5 часов, третий – 8 часов и последний – 3 часа. На этих этапах было выявлено 3, 2, 2, 0 ошибок. Полученные результаты можно привести к таблице 4.3.

Таблица 4.3 – Результаты тестирования

Этап (k)	Продолжительность (t), ч	Количество сбоев (m)
1	7	3
2	5	2
3	8	2
4	3	0

Общее время тестирование будет равно T = 23 часа.

Общее число обнаруженных и исправленных ошибок n = 7.

Количество ошибок, исправленных к началу (i + 1) этапа равно n0 = 0; n1 = 0; n2 = 3; n3 = 5; n4 = 7.

Функция надежности по модели Шумана описывается следующей формулой:

$$R_i(t) = e^{-\lambda_i t}, \tag{4.1}$$

где λ – интенсивность появления ошибок, которая вычисляется по формуле

$$\lambda = (N - n) * C, \tag{4.2}$$

где N – первоначальное количество ошибок;

C – коэффициент пропорциональности, который равен

$$C = \frac{\sum_{i=1}^{k} \frac{m_i}{N - n_{i-1}}}{\sum_{i=1}^{k} t_i}, \tag{4.3}$$

Чтобы найти первоначальное количество ошибок N обратимся к следующей формуле

$$\sum_{i=1}^{k} m_i \frac{\sum_{i=1}^{k} t_i}{\sum_{i=1}^{k} \frac{m_i}{N - n_{i-1}}} = \sum_{i=1}^{k} (N - n_{i-1}) t_i, \tag{4.4}$$

Из данного уравнения путем подбора найдем N = 8.

Теперь зная значение N найдем коэффициент C по формуле 4.3, и он будет равен 0,05.

Рассчитав коэффициент C получаем что интенсивность появления ошибок равна

$$\lambda = (8 - (3 + 2 + 2)) * 0.05 = 0.05$$

Выведем функцию надежности

$$R(t) = e^{-0.05t}$$

По полученным результатам информационная система является надежной и может эксплуатироваться. Так как система использует платформу Openshift, которая из коробки предоставляет механизмы скалирования, автохилинга и т.д., что позволяет еще больше повысить надежность информационной системы в целом.

Выводы к главе 4

1) Описан процесс внедрения системы для интеграции мобильного приложения с корпоративной информационной системой страховой компании, а также описана интеграция с ней. В рамках это была построена диаграмма развертывания системы.

2) Проведено полное функциональное тестирование системы, что позволило выявить проблемы и исправить их.

3) По результатам тестирования был проведен расчет надежности разработанной информационной системы по Модели Шумана, который показал ее надежность.

ЗАКЛЮЧЕНИЕ

В ходе проведения исследования и написания диссертации был проведен анализ существующих методов интеграции мобильных приложений с корпоративными системами и разработан метода интеграции, который обеспечит высокую эффективность реализации продаж страховых услуг.

Для достижения результата были выполнены поставленные задачи:

– Проанализированы подходы к интеграции мобильных приложений с корпоративными информационными системами;

– Проанализированы современные мобильные приложения по продаже страховых полисов;

– Разработана модель интеграции мобильного приложения по продаже страховых полисов;

– Подтверждена эффективность предлагаемой модели на практике.

Описанная в работе функциональность платформы для интеграции с корпоративной информационной системой страховой компании позволит произвести интеграцию с кроссплатформенными приложениями такими как Android, iOS и т.д. Она позволит снизить временные издержки на интеграцию, а также поможет принести прибыль с мобильных продаж страховых полисов.

В разработанная система построена микросервисной архитектуре с использованием современных контейнерных технологиях таких каких Openshift, который содержит удобные встроенные инструменты, которые обеспечивают эффективную балансировку нагрузки и автоматическое масштабирование, что позволяет достичь большой степени надежности системы в целом.

Разработка информационной системы собственными силами предприятия дает возможность легкого и быстрого добавления нового функционала, необходимого для производства, позволяет с наименьшими затратами адаптироваться к современным требованиям рынка.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Онлайн-договоры в страховании. // Synerdocs — сервис обмена электронными документами [Электронный ресурс]. – Режим доступа: https://www.synerdocs.ru/6261782.aspx (дата обращения 23.02.2019).

2. Яковлев В.П. Корпоративные информационные системы: конспект лекций / В.П. Яковлев; СПбГТУРП. – СПб., 2015. − 117 с.

3. Корпоративная информационная система: определение и структура. Современные подходы к построению корпоративных информационных систем. // Корпоративные информационные системы [Электронный ресурс]: – Режим доступа: http://e-educ.ru/ism14.html (дата обращения 23.02.2019).

4. Методы и средства интеграции информационных систем в рамках единого информационного пространства. [Электронный ресурс]. – Режим доступа: http://lab18.ipu.ru/projects/conf2012/1/7.htm (дата обращения 23.02.2019).

5. Дьяченко Д.Г. Унификация системного программного обеспечения на основе технологии виртуализации / Д.Г. Дьяченко // Известия ТулГУ. Технические науки. 2012. №5. [Электронный ресурс]. – Режим доступа: https://cyberleninka.ru/article/n/unifikatsiya-sistemnogo-programmnogo-obespecheniya-na-osnove-tehnologii-virtualizatsii (дата обращения 23.02.2019).

6. Сервис-ориентированная архитектура (SOA) // Хабр [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/mailru/blog/342526/ (дата обращения 23.02.2019).

7. Интеграция бизнес-процессов и сервис-ориентированная архитектура // Решения и технологии [Электронный ресурс]. – Режим доступа: https://www.osp.ru/data/134/081/1237/soa2.pdf (дата обращения 23.02.2019).

8. История мобильного интернета // Хабр [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/yota/blog/352450/ (дата обращения 24.02.2019).

9. What are the popular types and categories of apps // ThinkMobiles [Электронный ресурс]. – Режим доступа: https://thinkmobiles.com/blog/best-augmented-reality-apps/ (дата обращения 24.02.2019).

10. Машинное обучение и мобильная разработка // Хабр [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/oleg-bunin/blog/416477/ (дата обращения 24.02.2019).

11. Знакомство с OpenStack: архитектура, функции, взаимодействия // IBM [Электронный ресурс]. – Режим доступа: https://www.ibm.com/developerworks/ru/library/cl-openstack-overview/index.html (дата обращения 24.02.2019).

12. Всегда ли нужны Docker, микросервисы и реактивное программирование // ITnan [Электронный ресурс]. – Режим доступа: http://itnan.ru/post.php?c=1&p=436346 (дата обращения 24.02.2019).

13. Гради Буч, Язык UML Руководство пользователя / Гради Буч, Джеймс Рамбо, Ивар Якобсон; Москва. – ДМК., 2006. − 483 с.

14. Брюс Эккель, Философия Java / Брюс Эккель; Питер. – Классика Computer Science., 2019. − 1168 с.

15. Send notifications across platforms for free // Firebase [Электронный ресурс]. - Режим доступа: https://firebase.google.com/products/cloud-messaging/?hl=ru (дата обращения 01.03.2019).

16. Пол Дейтел, Android для разработчиков / Пол Дейтел, Харви Дейтел, Александр Уолд; Санкт-Петербург. – Питер., 2016. − 512 с.

17. How Cover, The Insurance App, Has Raised $16M in Series B Funding With These 3 Simple Strategies // Space Technologies [Электронный ресурс]. - Режим доступа: https://www.spaceotechnologies.com/on-demand-insurance-app-development-strategies-cover-startups/

18. HTTP basic authentication // IBM [Электронный ресурс]. - Режим доступа: https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.1.0/com.ibm.cics.ts.internet.doc/topics/dfhtl2a.html (дата обращения 01.03.2019).

19. Как и для чего использовать Docker // Hexlet Guides [Электронный ресурс]. - Режим доступа: https://guides.hexlet.io/docker/ (дата обращения 02.03.2019).

20. Functional Testing // Software Testing Fundamentals [Электронный ресурс]. - Режим доступа: http://softwaretestingfundamentals.com/functional-testing/ (дата обращения 02.03.2019).

21. Arquillian documentation // Arquillian [Электронный ресурс]. - Режим доступа: http://arquillian.org/docs/ (дата обращения 02.03.2019).

22. Introducing the Google Guava Cache // jayway [Электронный ресурс]. - Режим доступа: https://blog.jayway.com/2012/04/16/introducing-the-google-guava-cache/ (дата обращения 02.03.2019).

23. Колчанов, В.Д. Экономическая эффективность внедрения информационных технологий / Колчанов В.Д., Кобко Л.И.; Учеб.пособие. - Москва 2006. - с. 177.

24. Белик, А.Г. Качество и надежность программных систем / Белик А.Г., Цыганенко В.Н.; Учеб.пособие. - Омск 2018. - с. 80.

25.Чистый код с Google Guava // Хабр [Электронный ресурс]. - Режим доступа: https://habr.com/ru/post/244347/ (дата обращения 02.03.2019).

26. Василенко Н. В., Макаров В. А. Оценка надежности программного обеспечения // Вестник НовГУ. 2005. №30. [Электронный ресурс]. – Режим доступа: https://cyberleninka.ru/article/n/otsenka-nadezhnosti-programmnogo-obespecheniya (дата обращения: 02.03.2019).

27. Оценка надежности программного обеспечения // Dev Harmony [Электронный ресурс]. - Режим доступа: http://kirnosenko.com/2011/03/07/software-reliability-estimation/ (дата обращения 02.03.2019).

28. IEEE Recommended Practice on Software Reliability // IEEEXplore [Электронный ресурс]. - Режим доступа: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7827907&isnumber=7827906 (дата обращения 02.03.2019).

29. Автоматизация тестирования Java EE веб-сервисов с помощью SoapUI и Arquillian // Хабр [Электронный ресурс]. - Режим доступа: https://habr.com/ru/post/267301/ (дата обращения 02.03.2019).

30. Core Concepts Routes // Red Hat Openshift [Электронный ресурс]. - Режим доступа: https://docs.openshift.com/enterprise/3.0/architecture/core_concepts/routes.html (дата обращения 03.03.2019).

ПРИЛОЖЕНИЕ А

Свидетельства о публикации

PUBLISHING HOUSE «PROBLEMS OF SCIENCE»

Проблемы науки

СВИДЕТЕЛЬСТВО РОСКОМНАДЗОРА О РЕГИСТРАЦИИ СМИ ПИ № ФС77 - 62929 ОТ 31.08.2015 Г. ISSN 2413-2101

СПРАВКА
О ПРИНЯТИИ СТАТЬИ К ПУБЛИКАЦИИ
№ ПН-7777 ОТ «21» ФЕВРАЛЯ 2019 г.

НАСТОЯЩАЯ СПРАВКА ВЫДАНА ДЛЯ ПРЕДЪЯВЛЕНИЯ ПО МЕСТУ ТРЕБОВАНИЯ И ПОДТВЕРЖДАЕТ ФАКТ ОФИЦИАЛЬНОЙ ПУБЛИКАЦИИ

СТАТЬЯ
АНАЛИЗ СПОСОБОВ ИНТЕГРИРОВАНИЯ КОРПОРАТИВНЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

АВТОР(ы)
Неклюдов Сергей Владимирович

Статья принята к публикации в журнал № 2 (38), 2019 год.
Территория распространения: Российская Федерация, зарубежные страны.
САЙТ ЖУРНАЛА: HTTP://SCIENCEPROBLEMS.RU

Заведующая редакцией

Ефимова А.В.

PUBLISHING HOUSE «PROBLEMS OF SCIENCE»

Academy

СВИДЕТЕЛЬСТВО РОСКОМНАДЗОРА О РЕГИСТРАЦИИ СМИ: ПИ № ФС 77 – 62019 ОТ 05.06.2015 г. ISSN 2412-8236

СПРАВКА
О ПРИНЯТИИ СТАТЬИ К ПУБЛИКАЦИИ

№ АС-8014 ОТ «6» МАРТА 2019 г.

НАСТОЯЩАЯ СПРАВКА ВЫДАНА ДЛЯ ПРЕДЪЯВЛЕНИЯ ПО МЕСТУ ТРЕБОВАНИЯ И ПОДТВЕРЖДАЕТ ФАКТ ОФИЦИАЛЬНОЙ ПУБЛИКАЦИИ

СТАТЬЯ

ИНТЕГРАЦИЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ С КОРПОРАТИВНОЙ ИНФОРМАЦИОННОЙ СИСТЕМОЙ СТРАХОВОЙ КОМПАНИИ

АВТОР(ы)

Неклюдов Сергей Владимирович

Статья принята к публикации в журнал № 3 (42), 2019 год.
Территория распространения: Российская Федерация, зарубежные страны.
САЙТ ЖУРНАЛА: HTTPS://ACADEMICJOURNAL.RU

Заведующая редакцией

Ефимова А.В.

ПРИЛОЖЕНИЕ Б

Фрагмент программного кода мобильного приложения

```
@Component
@Slf4j
public class GroovyScriptCompiler implements ScriptCompiler
{
    private GroovyShellFactory groovyShellFactory;
    private final AllowableClassesHolder allowedClasses;
    private final LibraryLoader libraryLoader;
    private final AstBuilder astBuilder;

    @Autowired
    public GroovyScriptCompiler(GroovyShellFactory groovyShellFactory,
AllowableClassesHolder allowedClasses, LibraryLoader libraryLoader) {
        this.groovyShellFactory = groovyShellFactory;
        this.allowedClasses = allowedClasses;
        this.libraryLoader = libraryLoader;
        this.astBuilder = new AstBuilder();
    }

    @Override
    public ParsedScript parseScript(Script script) {
        List<ASTNode> astNodes;
        try {
            astNodes = astBuilder.buildFromString(CompilePhase.CONVERSION,
script.getSourceCode());
        } catch (MultipleCompilationErrorsException e) {
            LOGGER.error("Cannot parse script", e);
            throw new CompilationException(e, MALFORMED_SCRIPT,
e.getMessage());
        } catch (Exception e) {
            LOGGER.error("Unable parse script", e);
            throw new CompilationException(e, SCRIPT_PARSING_ERROR,
e.getMessage());
        }
        return new GroovyParsedScript(astNodes, script);
    }

    @Override
    public GroovyCompiledScript compile(Script script)
    {
        String sourceCode = script.getSourceCode();
        ScriptLibraries dependencies = script.getDependencies();
```

```
        GroovyShell shell = groovyShellFactory.build();

        try (ScriptSandbox sandbox = new ScriptSandbox(libraryLoader,
allowedClasses)) {

            libraryLoader.reloadLibrariesIfNeeded(dependencies);
            GroovyScriptBase groovyScript = (GroovyScriptBase)
shell.parse(sourceCode);

            return new GroovyCompiledScript(groovyScript, script);

        } catch (CompilationFailedException e) {
            LOGGER.error("Cannot compileByClassName the script", e);
            throw new CompilationException(e, MALFORMED_SCRIPT,
e.getMessage());
        } catch (SecurityException e) {
            LOGGER.error("Script violates security rules", e);
            throw new CompilationException(e,
SCRIPT_VIOLATES_SECURITY_RULES, e.getMessage());
        } catch (Exception e) {
            LOGGER.error("Unable to compileByClassName script class", e);
            throw new CompilationException(e, SCRIPT_EXECUTION_ERROR);
        }
    }

    @Override
    public Class compileClass(Script script)
    {
        String sourceCode = script.getSourceCode();
        ScriptLibraries dependencies = script.getDependencies();

        GroovyShell shell = groovyShellFactory.build();

        try (ScriptSandbox sandbox = new ScriptSandbox(libraryLoader,
allowedClasses)) {

            libraryLoader.reloadLibrariesIfNeeded(dependencies);
            return shell.getClassLoader().parseClass(sourceCode);

        } catch (CompilationFailedException e) {
            LOGGER.error("Cannot compileByClassName the script", e);
            throw new CompilationException(e, MALFORMED_SCRIPT,
e.getMessage());
```

```
        } catch (SecurityException e) {
            LOGGER.error("Script violates security rules", e);
            throw new CompilationException(e,
SCRIPT_VIOLATES_SECURITY_RULES, e.getMessage());
        } catch (Exception e) {
            LOGGER.error("Unable to compileByClassName script class", e);
            throw new CompilationException(e, SCRIPT_EXECUTION_ERROR);
        }
    }

    @Override
    public void cleanupMetadata(CompiledScript compiledScript) {
        if (!isNull(compiledScript)) {
            GroovyCompiledScript groovyScript =
(GroovyCompiledScript)compiledScript;
            GroovyScriptBase nativeScript = groovyScript.getCompiledGroovyScript();
            final GroovyClassLoader groovyClassLoader = (GroovyClassLoader)
nativeScript.getClass().getClassLoader();
            InvokerHelper.removeClass(nativeScript.getClass());
            groovyClassLoader.clearCache();
        } else {
            LOGGER.info("Cannot cleanup metadata of null script");
        }
    }
}
```