МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

02.03.03 МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ И АДМИНИСТРИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

БАКАЛАВРСКАЯ РАБОТА

на тему Разработка серверной части информационной системы «Умный дом»

Студент Л. Ю. Михайлов _____

Руководитель А. В. Очеповский _____

Допустить к защите
Заведующий кафедрой к.тех.н, доцент, А.В. Очеповский _____

«_____» _____ 2016 г.

Тольятти 2016

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

Институт **математики, физики и информационных технологий**
Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав. кафедрой «Прикладная математика и информатика»
_____ А.В. Очеповский
«____»_____ 2016 г.

ЗАДАНИЕ
на выполнение бакалаврской работы

Студент Михайлов Леонид Юрьевич
1. Тема Разработка серверной части информационной системы «Умный дом»
2. Срок сдачи студентом законченной выпускной квалификационной работы 13.06.2016
3. Исходные данные к выпускной квалификационной работе: показания с датчиков аппаратного устройства, хранение информации в базе данных, требования к функциональным характеристикам: обеспечение канала связи с аппаратным устройством и мобильным клиентом по типу JSON, криптографическая защита для передачи данных.
4. Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов, разделов):
Введение
Глава 1. Анализ информационной системы «Умный дом»
Глава 2. Проектирование серверной части информационной системы «Умный дом»
Глава 3. Реализация серверной части информационной системы «Умный дом»
Заключение
5. Ориентировочный перечень графического и иллюстративного материала: презентация, включающая блок-схемы работы информационной системы, графики, диаграммы.
6. Дата выдачи задания «11» января 2016 г.

Заказчик, менеджер по проектам ООО «Сто линий» _____ С. А. Мальцев
Руководитель выпускной квалификационной работы _____ А. В. Очеповский
Задание принял к исполнению _____ Л. Ю. Михайлов

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав. кафедрой «Прикладная математика и информатика»
_____ А.В. Очеповский

«____»_____ 2016 г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения бакалаврской работы

Студента Михайлова Леонида Юрьевича
по теме Разработка серверной части информационной системы «Умный дом»

Наименование раздела работы	Плановый срок выполнения раздела	Фактический срок выполнения раздела	Отметка о выполнении	Подпись руководителя
Выбор и утверждение темы ВКР	14.01.2016	14.01.2016	Выполнено	
Анализ состояния вопроса	20.02.2016	20.02.2016	Выполнено	
Разработка алгоритма анализа и передачи информации с датчиков	15.03.2016	15.03.2016	Выполнено	
Конструирование базы данных	20.03.2016	20.03.2016	Выполнено	
Программная реализация предложенных решений	29.03.2016	29.03.2016	Выполнено	
Апробация предложенных решений на реальных данных	30.04.2016	30.04.2016	Выполнено	
Оформление текста бакалаврской работы	5.05.2016	5.05.2016	Выполнено	
Подготовка презентации к выступлению	10.06.2016	10.06.2016	Выполнено	

Предварительная защита ВКР	14.06.2016	14.06.2016	Выполнено	
Корректировка ВКР согласно сделанным замечаниям	15.06.2016	15.06.2016	Выполнено	
Проверка ВКР в системе «Антиплагиат.ВУЗ»	16.06.2016	16.06.2016	Выполнено	
Сдача пояснительной записки ВКР и реализованного программного приложения	16.06.2016	16.06.2016	Выполнено	

Руководитель выпускной квалификационной работы _____ А. В. Очеповский

Задание принял к исполнению _____ Л. Ю. Михайлов

Аннотация

Тема данной выпускной квалификационной работы: «Разработка серверной части информационной системы «Умный дом»»

Работа выполнена студентов ТГУ, института математики, информатики, физики и информационных технологий, группы МОб-1201, Михайловым Леонидом Юрьевичем.

Актуальность темы данной работы обусловлена необходимостью систематизации работы информационной системы «Умный дом».

Целью данной работы является разработка серверного приложения для информационной системы «Умный дом».

Для достижения поставленной цели работе решались следующие задачи:

— систематизация знаний о системе «Умный дом»;

— анализ существующих автоматизированных систем контроля информации о помещении;

— проектирование архитектуры серверной части информационной системы «Умный дом»;

— проектирование базы данных;

— реализация серверного приложения для информационной системы «Умный дом».

Работа состоит из введения, трех глав (аналитической, проектной и части реализации), заключения, списка использованной литературы и приложения.

В работе использованы современные системы и технологии проектирования информационных систем, Java, MySQL и др.

Первая глава посвящена анализу предметной области.

Вторая глава посвящена разработке серверной части информационной системы «Умный дом».

В третьей главе предоставлена реализация спроектированной серверной части для информационной системы «Умный дом».

В заключении сформулированы основные выводы, которые были сделаны в процессе исследования и описаны результаты практической реализации работы.

В приложении предоставлены фрагменты программного кода и другие дополнительные материалы.

Объем данной работы 50 страниц, на которых размещены 35 рисунков и 2 таблицы. При написании работы использовалось 26 источников.

Оглавление

Введение

О понятии «Умный дом» или «Smart House» впервые заговорили в Вашингтоне в 70–х годах прошлого века разработчики Института интеллектуального здания. С тех пор система «Умного дома» эволюционировала и представляет собой прогрессирующую концепцию взаимодействия человека со своим жильем. Так ситуация складывается за рубежом.

В России же, для большинства населения данная система остается просто термином. Недостаток информации и понимания о принципе работы подобных комплексов, систем «Умного дома» остается проблемной стороной внедрения подобных комплексов в нашей стране.

Актуальность выпускной квалификационной работы обусловлена отсутствием готовых решений систем автоматизированного управления помещений.

Разрабатываемая информационная система является **новшеством** в сфере работ жилищно–коммунального хозяйства, которое позволяет автоматизировать сбор информации о расходуемых ресурсах.

Объект выпускной квалификационной работы – процесс автоматизации взаимодействия информационной системы «Умный дом».

Предмет выпускной квалификационной работы – серверная часть информационной системы «Умный дом».

Цель выпускной квалификационной работы – разработка серверного приложения для информационной системы «Умный дом».

Для достижения указанной цели в ходе выполнения выпускной квалификационной работы, необходимо выполнить следующие задачи:

— систематизировать знания о системе «Умный дом»;

— проанализировать существующие автоматизированные системы контроля информации о помещении;

— обосновать использование вычислительной техники;

— спроектировать архитектуру серверной части информационной системы «Умный дом»;

— спроектировать базу данных;

— реализовать серверное приложение для информационной системы «Умный дом».

Выпускная квалификационная работа состоит из введения, трех глав, заключения, списка литературы и приложений.

Во введении описывается актуальность и новизна проводимого исследования, формируется цель, и ставятся задачи, которые необходимо решить для достижения цели.

В первой главе описывается анализ существующих информационных систем «Умный дом», формируются требования к информационной системе.

Во второй главе проводится проектирование серверной части информационной системы «Умный дом», включающее в себя проектирование общей архитектуры приложения, проектирование базы данных, а также проектирование архитектуры компонентов серверного приложения.

В третьей главе описывается реализация серверной части информационной системы «Умный дом», ее тестирование и ввод в эксплуатацию.

В заключении приводятся основные выводы по работе, достигнутые в ходе выполнения выпускной квалификационной работы.

Глава 1 Анализ информационной системы «Умный дом»

1.1 Описание информационной системы «Умный дом»

Инженерный комплекс «Умный дом» – это инновационная система, которая позволяет интегрировать имеющиеся коммуникации в одну систему, управляемая искусственным интеллектом, программируемым под индивидуальные потребности клиента [15].

Аналоги инженерного комплекса «Умный дом» распространены по всему миру, но для большей части населения России инновация остается незнакомым термином. Внедрение инженерных комплексов «Умный дом» в нашей стране осложняется такими факторами, как недостаток информации и понимания о принципе работы подобных комплексов и экономическая безграмотность при расчете затрат и выгоды от внедрения [2].

Несмотря на это, инженерный комплекс внедряется в домах россиян частями: устанавливаются световые датчики движения, датчики протечки воды. Также россияне управляют при помощи «Умного дома» системой безопасности своих домов, домашними кинотеатрами и освещением. Инженерный комплекс не просто комфортен для быта клиента, но и позволяет экономить финансовые затраты на коммунальные услуги.

Инженерный комплекс «Умный дом» настраивается под индивидуальные потребности клиента. А потому перед этапом внедрения интеллектуальной системы в архитектуру дома, клиенту необходимо определиться на начальном этапе о желаемом функционале комплекса.

По прогнозам экспертов японско–американской компании по комплексной защите информации Trend Micro, спрос на «Умные дома» будет зависеть от консолидации сил компаний–инсталляторов и архитекторов для проектирования инженерных комплексов. Это вполне логично, поскольку клиенту удобнее «купить систему разом», нежели приобретать услуги экспертов и специалистов по частям.

При долгосрочной совместной работе архитектора и инсталлятора их взаимодействие укрепляется. А потому клиент не будет тратить свое время, пока архитектор и компания–инсталлятор разбираются в специфике работы друг друга, изучая особенности оборудования и технологии. Алгоритм работы «клиент – архитектор+инсталлятор» экономит время внедрения интеллектуальной системы в собственность клиента. Недостаток такого алгоритма в том, что он работает в случае проектирования помещения.

В случае, если планируется внедрение системы в уже построенное здание, то лучше воспользоваться устройством, которое выполнит функцию сбора информации о помещении и передачи её пользователю с целью рационального пользования ресурсами. Такое устройство может содержать различные модули сбора информации, позволяющие расширить функционал и потребительский спрос устройства.

1.2 Архитектура информационной системы «Умный дом»

Под архитектурой системы умного дома следует понимать совокупность модулей необходимых для её полноценной работы. Данные модули должны быть связанны между собой определённым образом, для обеспечения стабильной и корректной работы всей системы [1].

Исходя из того, что система «Умный дом» подразумевает под собой сбор, анализ и организацию передачи данных, данной системе требуется свой сервер, а также база данных [13]. Данные, собранные с датчиков, передаются на сервер и сохраняются в базу данных и в дальнейшем используются по запросу в web–приложении или мобильном клиенте.

На рисунке 1.1 представлена архитектура информационной системы «Умный дом», которая включает в себя разрабатываемую серверную часть.

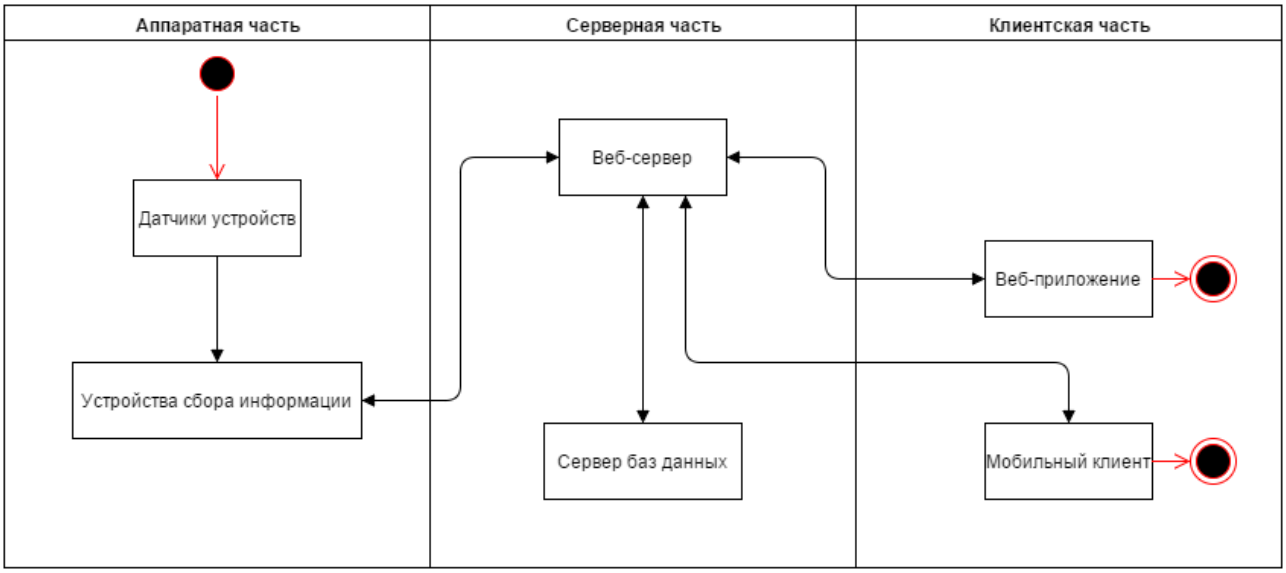

Рисунок 1.1 – Архитектура информационной системы «Умный дом»

Для интеграции данной ИС в структуру Жилищно-коммунального хозяйства (далее просто «ЖКХ») рассмотрим модель системы контроля показаний с электрощитов на рисунке 1.2 и ее декомпозицию на рисунке 1.3.

Рисунок 1.2 – Модель процесса контроля показаний «как есть» в нотации IDEF0

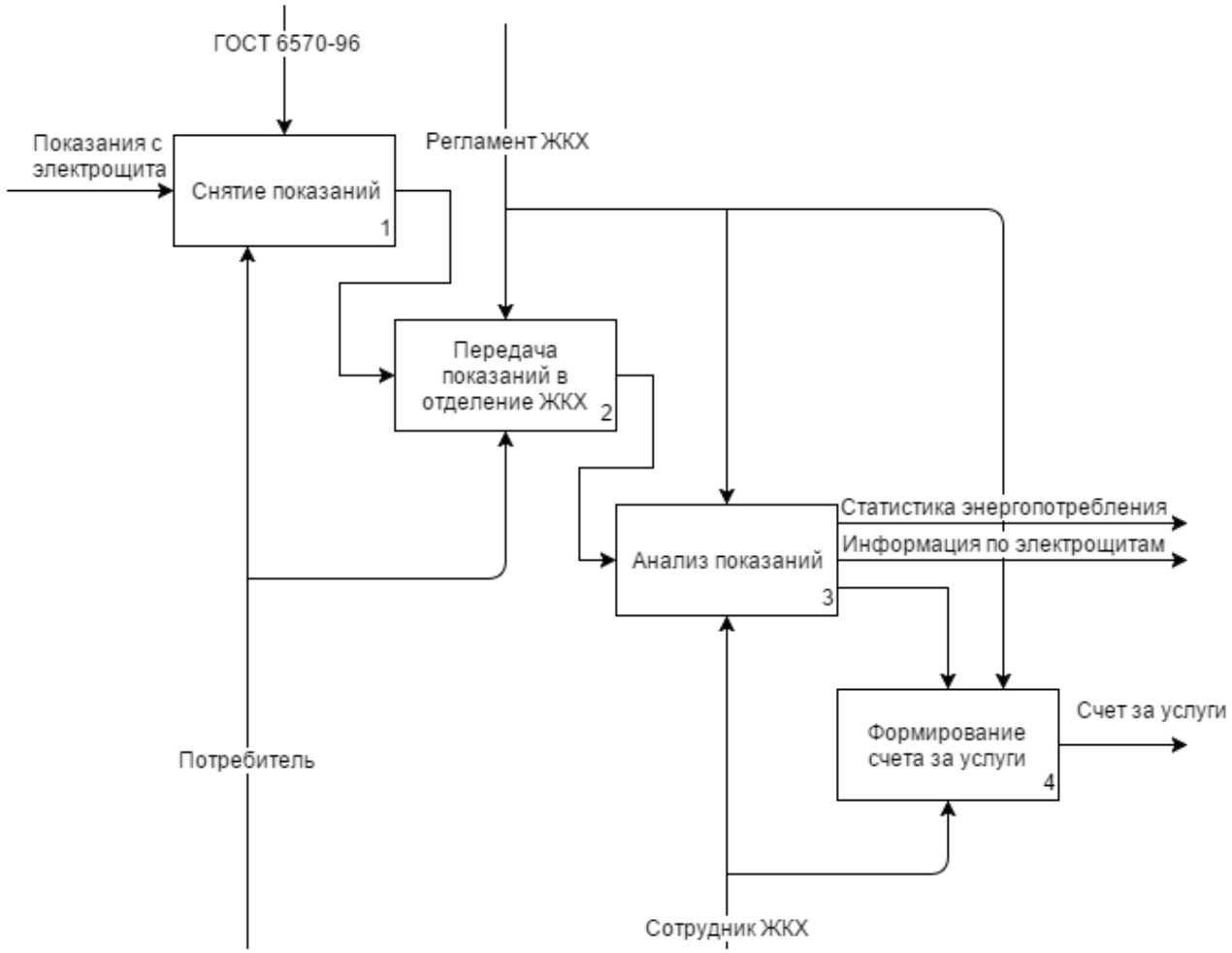

Рисунок 1.3 – Декомпозиция модели процесса контроля показаний «как есть»

Опишем процессы модели на рисунке 1.3:

— снятие показаний с электрощита производит потребитель;

— снятые показания передаются потребителем в отделение ЖКХ;

— сотрудник ЖКХ анализирует показания;

— сотрудник ЖКХ формирует счет за предоставленные услуги.

Исходя из данной модели выявлены следующие недостатки:

— для непосредственного сбора и передачи показаний необходим потребитель;

— для анализа показаний и формирование счета необходим сотрудник ЖКХ;

— отсутствует организованная система сбора информации;

Данная модель может быть усовершенствована путем внедрения новой информационной системы «Умный дом».

1.3 Формирование требований к информационной системе «Умный дом»

На основе особенностей системы контроля информации были сформированы следующие требования к проектируемой информационной системе:

1. Установка системы «Умный дом» не должно требовать серьезных изменений в архитектуре помещения.
2. Реализовывать систему «Умный дом» необходимо на платформе предоставляющий удобный функционал замены вышедших из строя модулей. Например, на Arduino [8].
3. Выбранное ПО должно обеспечивать легкость встраиваемости новых элементов в систему, или же объединение аналогичных устройств в одну систему.
4. Составляющие систему модули должны быть соотнесены по отношению цена/высокое качество. Это позволит снизить итоговую стоимость проекта.
5. Проект должен реализовывать качественную систему шифрования данных для обеспечения безопасности данных.
6. Реализовать множественное параллельное подключение устройств на базе Arduino к серверу для создания сети «Умных домов» с единым управлением.
7. Серверное приложение должно обрабатывать данные не менее чем с десяти устройств одновременно.

Для устранения недостатков существующей системы было принято решение в пользу создания новой информационной системы.

Таким образом, при успешном выполнении определенных выше требований к проекту «Умный дом», будет спроектировано экономное устройство с единым сервером, успешно заменяющее большинство

аналогичных систем «Умный дом». А если проектировать устройство изначально монолитным, то тем самым можно повысить его потребительские качества. Перспектива работы над проектом заключается в доработке устройства и маркетинговой компании, продвижения и популяризации систем «Умный дом» в российском регионе.

1.3.1 Анализ аналогов систем автоматизированного управления

В 2011 году в международном журнале «Молодой ученый» были опубликованы результаты исследования систем автоматизированного управления домом. Достоинства и недостатки систем представлены в таблице 1.1.

Таблица 1.1 – Анализ существующих систем на предмет удовлетворения требованиям

Автоматизированная система управления	Краткое описание	Достоинства	Недостатки	Удовлетворение составленным требованиям
EIB	«EIB – это европейский стандарт международной ассоциации European Installtion Bus Association, объединяющей десятки ведущих европейских производителей электротехнической продукции» [5].	Автономность и независимость от работоспособности компьютера визуализации, работает и хранит в памяти все сценарии. Компоненты устанавливаются в стандартный электрощит или монтажные коробки. Адаптивность, возможность дополнения.	Высокая цена. Низкая скорость передачи команд (всего 0,3 сек). Низкий уровень помехозащищенности. Реализация недопустима на большом объекте. Низкий ассортимент датчиков движения.	Не удовлетворяет составленным требованиям.

Автоматизированная система управления	Краткое описание	Достоинства	Недостатки	Удовлетворение составленным требованиям
X–10	X–10 – это опробованная технология, которая получила широкое распространение на европейском и американском рынках. X–10 – часто применяемый стандарт в области автоматизации домашнего быта. «X–10 – метод и протокол передачи управляющих сигналов–команд по силовой электропроводке на электронные модули, к которым подключены управляемые электробытовые и осветительные приборы» [5].	Компоненты легко устанавливаются в стандартном электрощите, простое подключение модулей к электрическим розеткам. Простое программирование. Открытый протокол. Простая интеграция в любую систему управления.	Низкая скорость передачи информации. Низкая помехозащищенность. Возможность ложного срабатывания из–за помехи в электросети. Возможен доступ к устройствам X–10 по электросети.	Частично удовлетворяет составленным требованиям.
Crestron	В основе централизованной системы Crestron лежит управление центральными контроллерами и множеством исполнительных командных блоков [3]. Управляющие контроллеры Crestron обладают большим набором встроенных возможностей, высоко производительны и достаточно гибки.	Широкий ассортимент интерфейсов пользователя. Возможность собрать в единый комплекс все системы жизнеобеспечения. Управление из единого центра.	Самая высокая цена. Небольшой выбор дизайнерских решений.	Не удовлетворяет составленным требованиям.

Таким образом, в процессе изучения материала по проекту «Умный дом» был выявлен ряд недостатков, которые на данном этапе снижают популярность

реализации данного проекта в городах России, а также значительно усложняют процесс эксплуатации данной системы.

Проанализировав недостатки существующих систем было принято решение создать новую информационную систему «Умный дом» на основе составленных требований.

Сильной стороной проектируемой нами системы «Умный дом» является:

— **Экономность**. Система «Умный дом» может контролировать расход ресурсов помещения или помочь информацией, о возможности оптимизации данного расхода;

— **Безопасность**. Система «Умный дом» способна также иметь в себе модули, благодаря которым обеспечивается дополнительная защита как самого помещения, так и его пользователей;

— **Централизованность**. Система «Умный дом» может контролировать сразу множество объектов.

Таким образом, путем интеграции создаваемой информационной системы в структуру ЖКХ мы преобразуем существующую систему сбора информации.

1.4 Требования к серверному приложению для информационной системы «Умный дом»

Функциональные требования включают в себя предполагаемое описание поведения системы, что она способна выполнять и как выглядит со стороны.

К функциональным требованиям относится:

— приложение должно получать зашифрованный пакет данных с устройства;

— приложение должно расшифровывать пакет данных;

— приложение должно распределять полученные данные в базу данных;

— приложение должно осуществлять добавление необходимых данных в базу данных по запросу пользователя.

Данные требования были составлены в соответствии с вышеприведенными фактами, исходя из потребностей системы к функционалу.

Нефункциональные требования описывают атрибуты качества, характеристики программного обеспечения. Данные требования включают в себя:

— программа должна быть разработана на Java при использовании фреймворка Spring;

— при разработке допускается использование такой IDE как IntelliJ Idea;

— прием данных должен осуществляться через протокол http, tcp передаваемые пакеты должны содержать данные в формате JSON;

— тестирование программы может проводиться без использования сторонних модулей или серверов.

Данные требования были составлены в соответствии с вышеприведенными фактами, исходя из потребностей системы к функционалу.

Глава 2 Проектирование серверной части информационной системы «Умный дом»

2.1 Общая архитектура системы контроля информации

Для того, чтобы собирать информацию с множества устройств и передавать ее посредством сети Интернет, а также для организации единого сервера для последующей обработки информации было принято решение спроектировать новую информационную систему, с исправлениями недостатков, которая будет иметь вид, как изображено на рисунке 2.1 и рисунке 2.2.

Рисунок 2.1 Модель процесса контроля показаний «как будет» в нотации IDEF0

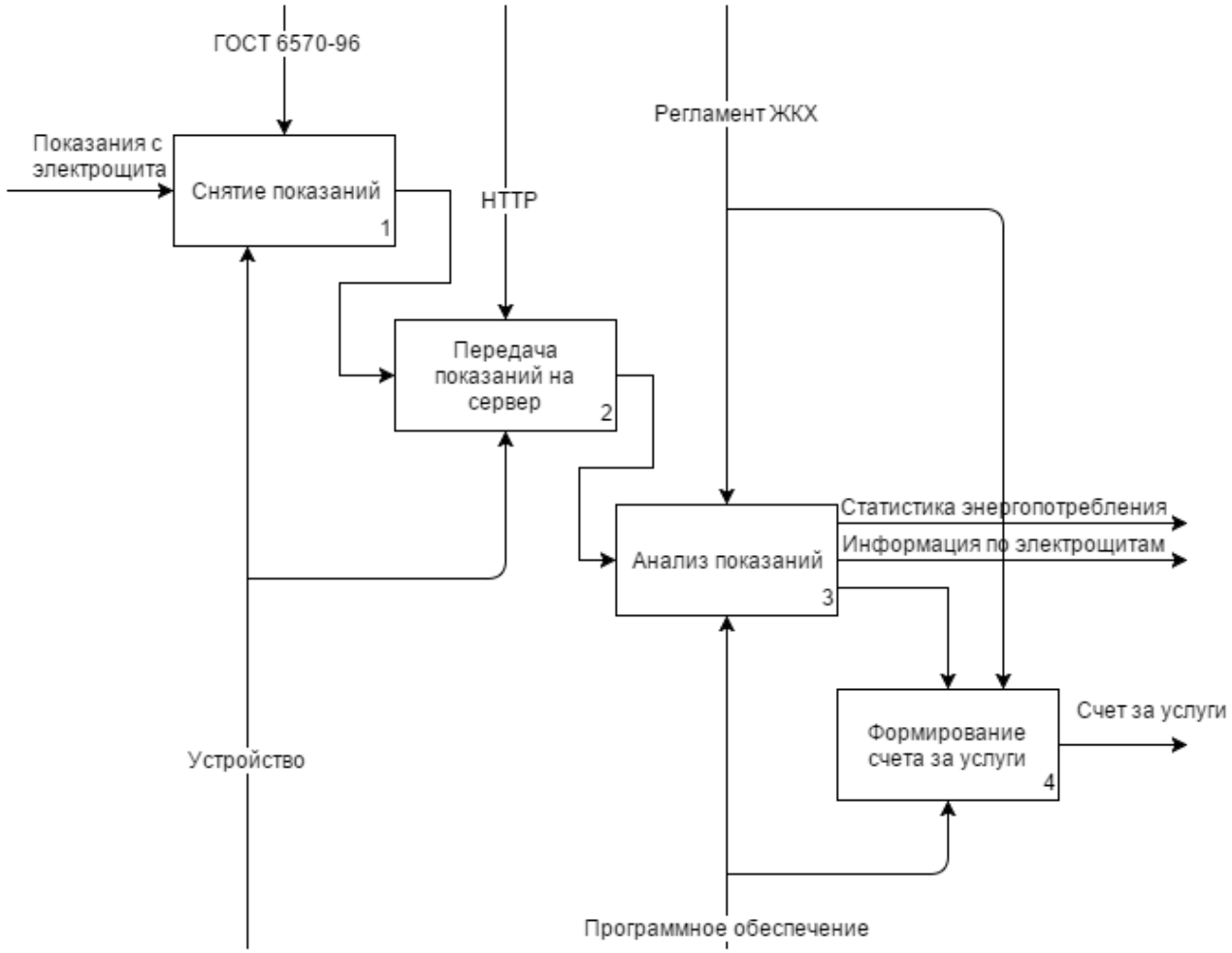

Рисунок 2.2Декомпозиция модели процесса контроля показаний «как будет»

Опишем процессы модели на рисунке 2.2:

— устройство собирает показания с электрощита;

— устройство передает пакет информации с показаниями на сервер;

— программное обеспечение анализирует предоставленную информацию и формирует отчеты;

— программное обеспечение формирует счета за предоставленные услуги.

Рассмотрим взаимодействие подсистем пошагово:

— каждое устройство собирает пакет информации с датчиков и отправляет его на единый web–сервер;

— web–сервер обрабатывает поступающие запросы и заносит информацию в базу данных;

— пользователь запрашивает данные из базы данных;

— web–сервер возвращает пользователю данные.

Таким образом, информационная система «Умный дом», разрабатываемая нами в ходе данной выпускной квалификационной работе должна состоять из сервера, клиента и множества устройств.

Сервер будет выполнять роль промежуточного звена между устройствами размещённых в помещениях и конечным пользователем, дополнительно собирая и анализируя полученную информацию, помогая тем самым сформировать более рациональную схему потребления ресурсов.

Под клиентом в данной схеме понимается устройство или программа получающая обработанную сервером информацию, и предоставляющая эту информацию конечному пользователю.

В качестве устройства мы принимаем электронную плату с набором датчиков и компонентов, обеспечивающих связь с серверной частью. Данное устройство будет собирать данные с помещения и обеспечивать их безопасную передачу на сервер.

Дальнейшее изложение этапов создания конечного приложения будет основываться на итерационном подходе, в основе которого лежит с постепенное усложнение и детализация первоначальной структуры приложения.

В основе разрабатываемой системы лежит архитектура «клиент–сервер», в которой задания или сетевая нагрузка распределены между поставщиками услуг (сервисов), называемых серверами, и заказчиками услуг, называемых клиентами. В качестве среды взаимодействия клиента с сервером используется сеть Интернет, как изображено на рисунке 2.3.

Рисунок 2.3 – Основа архитектуры проектируемой информационной системы

Основными достоинствами архитектуры «клиент–сервер» являются:

— возможность распределить функции вычислительной системы между несколькими независимыми компьютерами в сети. Это позволяет упростить обслуживание вычислительной системы. В частности, замена, ремонт, модернизация или перемещение сервера не затрагивают клиентов;

— все данные хранятся на сервере, который, как правило, защищен гораздо лучше большинства клиентов. На сервере проще обеспечить контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа;

— позволяет объединить различные клиенты. Использовать ресурсы одного сервера часто могут клиенты с разными аппаратными платформами, операционными системами и т.п.;

— позволяет использовать только необходимое количество ресурсов, что снижает затраты на оборудование [14].

Из основных недостатков архитектуры «клиент–сервер» следует выделить:

— в случае использования централизованной системы, неработоспособность основного сервера может сделать неработоспособным все приложение;

— в случае сбоя в работе сервера для устранения неисправностей требуется наличие квалифицированного персонала.

В ходе выбора технической платформы были предложены решения, позволяющие минимизировать вероятность выхода из строя серверной части приложения, а также позволяющее снизить стоимость оборудования до оплаты минимально необходимого уровня производительности.

2.2 Проектирование архитектуры серверного приложения

Поскольку основной целью приложения является предоставление услуг посредством сети Интернет, необходимо включить в его серверную часть веб–сервер – аппаратно–программный комплекс, предназначенный для

обслуживания HTTP–запросов. HTTP – протокол, который служит для передачи гипертекстовых документов в формате HTML. Как правило, такие запросы посылает интернет–браузер клиента. В нашем случае, данный запрос будет приходить на сервер со стороны устройства.

Поскольку информационная система предполагает большое число операций по чтению, записи и изменению значительного объема данных, наиболее удобным вариантом будет включение в серверную часть технологий баз данных.

На рисунке 2.4 отображен процесс детализации первоначальной архитектуры приложения, а точнее его серверной части. Серверная часть вмещает себя веб–сервер и сервер баз данных. В задачи веб–сервера входят:

— получение и ответ на HTTP–запросы;

— перенаправление запросов на необходимое приложение (сайт), как правило, приписанное к определенному домену или поддомену;

— предоставление приложениям доступа к информации по необходимым модулям;

— авторизация и аутентификация пользователей;

— другие функции.

В задачи сервера баз данных входит:

— обслуживание запросов на манипуляции с данными на основе языка SQL;

— обслуживание базы данных;

— обеспечение целостности данных;

— предоставление утилит для административного управления СУБД.

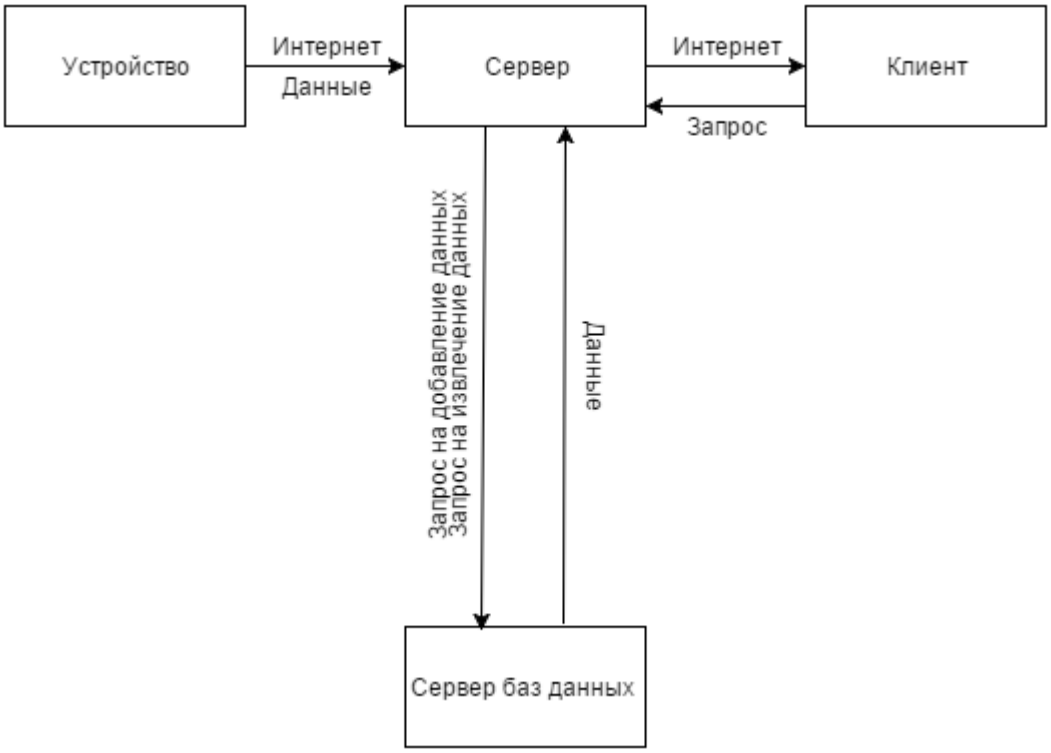

Рисунок 2.4 – Детализация первоначальной архитектуры серверной части информационной системы «Умный дом»

Ввиду необходимости защиты данных было принято решение передавать данные с устройства на сервер в зашифрованном виде с использованием симметричного алгоритма блочного шифрования Advanced Encryption Standard (AES). Этот алгоритм хорошо проанализирован и широко используется. Приложение будет включать в себя класс дешифровки с использованием заранее подготовленного ключа.

Пакет данных, получаемый с устройства сформирован в формате JSON. JSON – легко читаемый как человеком, так и компьютером формат обмена данными. Для внесения информации из такого пакета данных в базу данных требует его разбора. Приложение будет включать в себя класс разбора сообщений JSON формата.

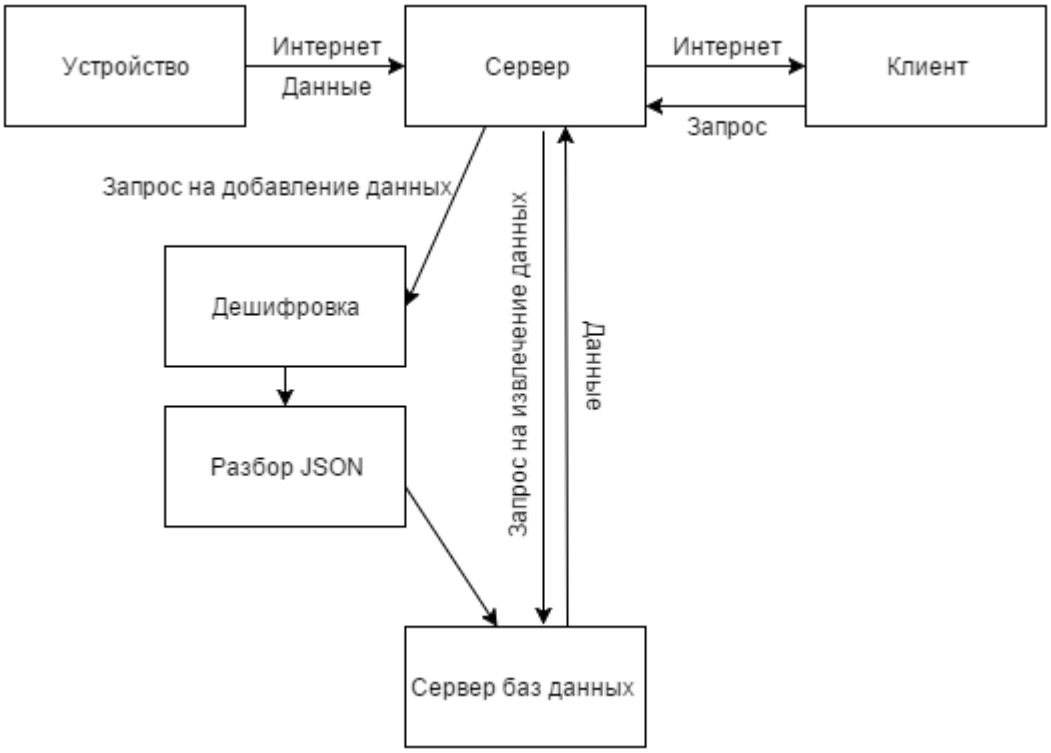

Рисунок 2.5 – Детализация конечной архитектуры серверного приложения информационной системы «Умный дом»

На рисунке 2.6 представлена архитектура серверного приложения ИС «Умный дом». Данная архитектура удовлетворяет всем требованиям, предъявляемым к серверному приложению информационной системы «Умный дом».

2.3 Проектирование базы данных

Создание базы данных включает в себя три этапа проектирования:

— инфологическое проектирование, цель которого состоит в анализе предметной области;

— логическое проектирование, цель которого состоит в преобразовании требований к данным в структуры данных;

— физическое проектирование, цель которого состоит в том, чтобы перенести логическую модель данных в среду СУБД.

Этап физического проектирования базы данных описан в первом параграфе третьей главы данной работы.

2.3.1 Инфологическое проектирование базы данных

Инфологическая модель – «это обобщенное, не привязанное к каким–либо ЭВМ и СУБД, описание предметной области (набор данных, их типов, длин, связей и др.)» [12].

Этапы инфологического проектирования:

— исследование предметной области и изучение ее информационной структуры;

— выявление фрагментов, которые можно охарактеризовать;

— моделирование и интеграция всех представлений.

Для инфологической модели необходимо выполнения условия формализованного описания предметной области. Это нужно для того, чтобы описание могло быть прочитано и усвоено не только специалистами БД. Это описание должно быть ёмким и обобщенным, т.е. не должна иметь привязку к конкретной СУБД.

Инфологическая модель представляет семантическую сторону предметной области модели БД.

Общепринятым стало сокращенное название ER–модель, большинство современных CASE–средств содержат инструментальные средства для описания данных в формализме этой модели. Кроме того, разработаны методы автоматического преобразования проекта БД из ER–модели в реляционную, при этом преобразование выполняется в даталогическую модель, соответствующую конкретной СУБД.

В настоящий момент не существует единой общепринятой системы обозначений для ER–модели и разные CASE–системы используют разные графические нотации, но разобравшись в одной, можно легко понять и другие нотации.

В основе разрабатываемой системы лежит хранилище данных, получаемых с датчиков. Каждый пакет данных содержит следующую информацию:

— имя сенсора;

— уникальный id устройства;

— дата и время снятия показаний;

— показания датчиков.

Сенсоры, фиксирующие показания, находятся на устройстве, которое в свою очередь находится на объекте, принадлежащем пользователю. Поэтому необходимо рассмотреть такие параметры, как:

— объект, принадлежащий пользователю;

— устройство, принадлежащее объекту;

— датчики, установленные в устройстве;

— показания, фиксируемые датчиками.

С разрабатываемой системой будут взаимодействовать следующие группы лиц:

— супервайзер системы;

— администраторы системы;

— пользователи системы.

Супервайзер имеет доступ к административному аккаунту сервера и, соответственно, имеет неограниченные права на любые действия в рамках системы.

Администраторы системы имеют права на выполнение следующих действий:

— добавление и редактирование данных пользователя;

— просмотр отзывов, оставленных на сайте с помощью формы обратной связи.

Пользователи системы могут выполнять следующие действия:

— просматривать информацию по датчикам;

— просматривать контентные страницы сайта;

— оставлять отзывы в форме обратной связи.

Супервайзер может выдавать аккаунты администратора. Пользователем системы является человек, имеющий готовое устройство с уникальным id.

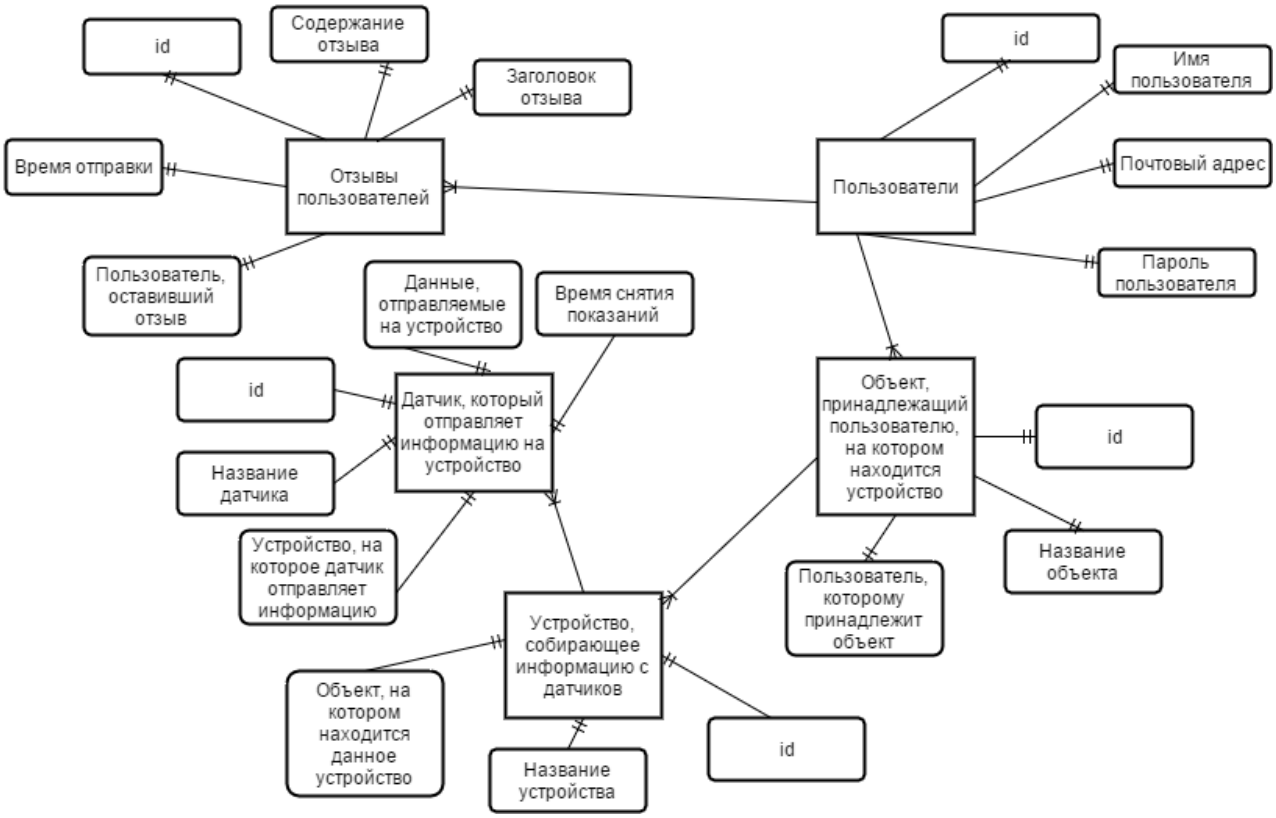

Рисунок 2.6Инфологическая модель базы данных

На рисунке 2.6 предоставлена спроектированная инфологическая модель базы данных.

2.3.2 Логическое проектирование базы данных

Логическое проектирование – этап проектирования базы данных, на котором формируется общая информационная модель предметной области.

На данном этапе производится преобразование требований к данным, показанных на рисунке 2.6, в структуры данных.

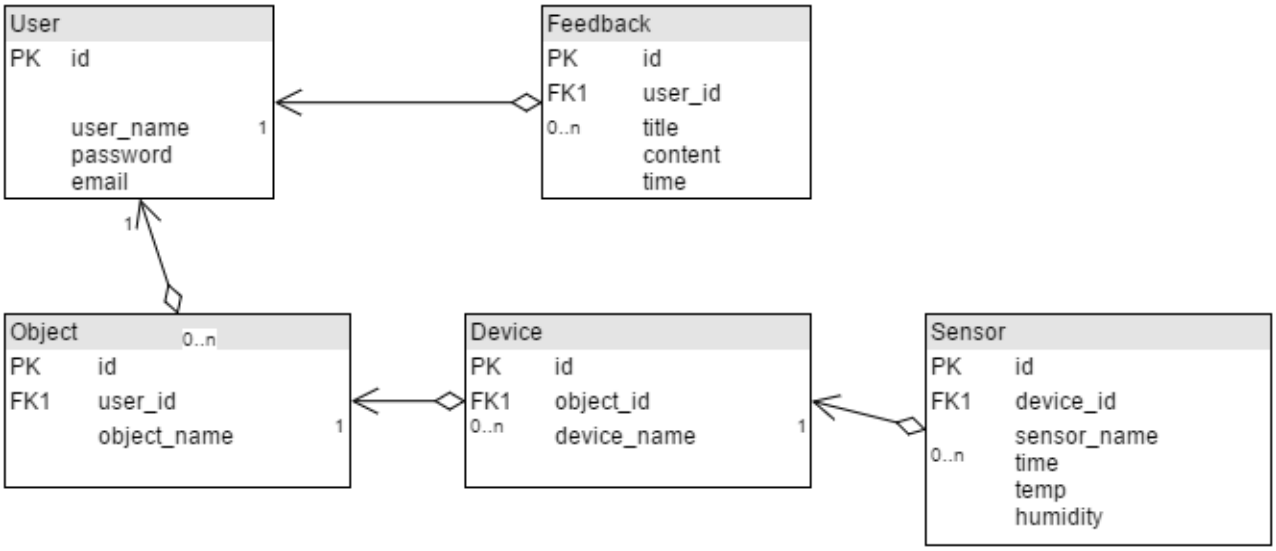

Рисунок 2.7 – Логическая модель базы данных

В таблице 2.2 опишем поля в каждой таблице по отдельности.

Таблица 2.2 – Описание таблиц

Таблица	Описание таблицы	Поле	Описание поля
object	объект, на котором находится устройство	id	первичный ключ таблицы
		object_name	название объекта
		user_id	пользователь объекта
device	устройство, которое собирает и отправляет данные	id	первичный ключ таблицы
		device_name	название устройства
		object_id	объект, на котором находится устройство
sensor	датчик, который фиксирует показания	id	первичный ключ таблицы
		sensor_name	название датчика
		time	время фиксации показаний
		temp	показания температуры, фиксируемые датчиком
		humidity	показания влажности, фиксируемые датчиком
		device_id	устройство, которое собирает показания с этого датчика
feedback	таблица для хранения отзывов	id	первичный ключ таблицы
		title	заголовок отзыва
		content	содержимое отзыва

Таблица	Описание таблицы	Поле	Описание поля
		time	время размещения отзыва
		user_id	пользователь, который отправил отзыв
user	таблица для хранения информации о пользователях	id	первичный ключ таблицы
		user_name	имя пользователя–администратора сайта
		password	пароль для входа
		email	адрес электронной почты пользователя–администратора

Спроектированная база данных удовлетворяет минимальным требованиям разрабатываемой информационной системы «Умный дом» и соответствует третьей нормальной форме.

2.4 Проектирование архитектуры компонентов серверного приложения

Проектирование общей модели будет выглядеть в виде разработки архитектуры компонентов серверного приложения. Представим необходимые для реализации классы в виде UML диаграммы на рисунке 2.8.

Опишем эти классы:

— GreetingController – класс, обрабатывающий GET и POST запросы;

— MainTemplate – содержит методы операций с базами данных, используя классы *DAO;

— классы *DAO – содержат методы, применяемые к соответствующим названиям классов таблицам;

— классы *RowMapper – определяют выборку результирующего набора данных вследствие запроса SELECT;

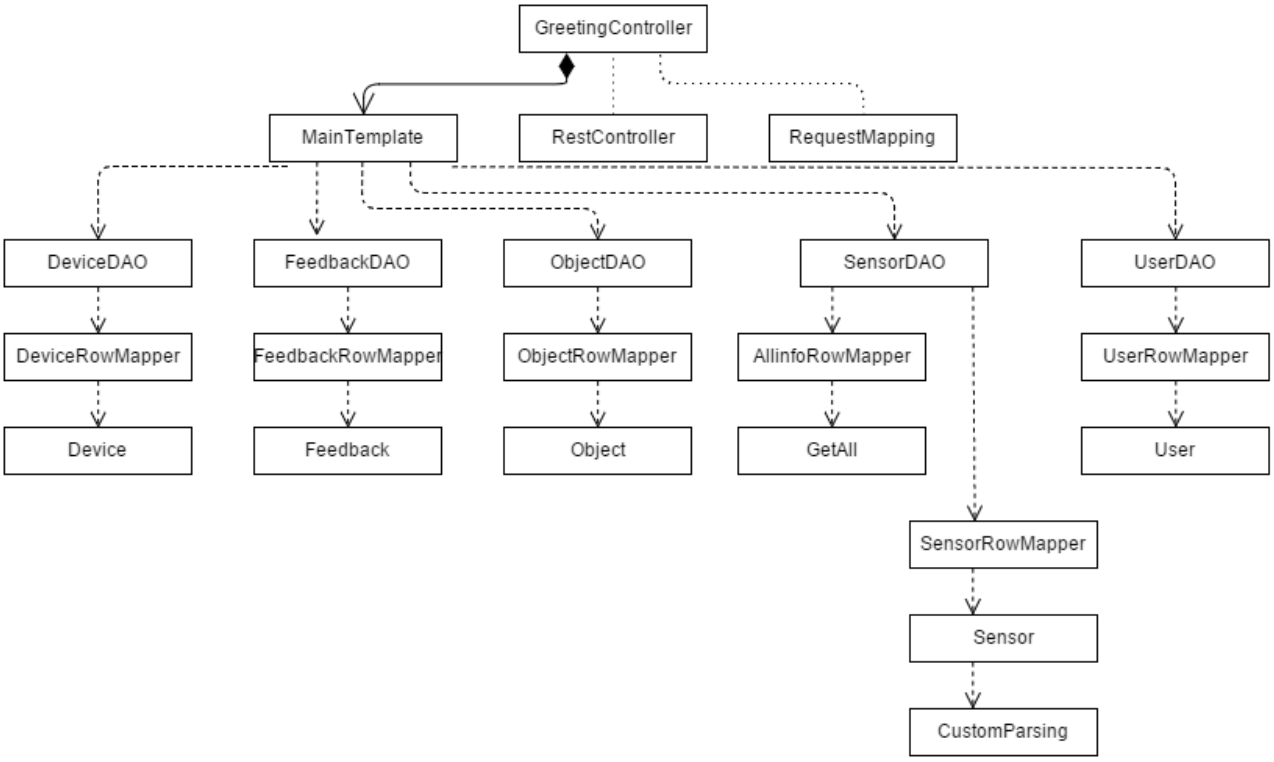

Рисунок 2.8 – UML диаграмма классов

— в классах Device, Feedback, Object, GetAll, Sensor и User будут описаны их параметры, которые идентичны столбцам в соответствующих таблицах, а также их get и set методы;

— класс CustomParsing разбирает JSON сообщение в представлении String на список: ключ–значение;

— @RestController – стандартное средство фреймворка Spring [23], сигнализирующее о том, что данный класс является контроллером и каждый метод может возвращать гипертекст или любые другие данные;

— @RequestMapping – стандартное средство фреймворка Spring, сигнализирующее о том, что каждый метод указанного класса будет обладать параметрами и возможностями, описанными в аннотации, например, получение запросов по определенному адресу с указанными методами передачи данных.

Данный список был сформирован в соответствии с необходимыми функциями приложения, дальнейшее развитие и реализация приложения предусматривается в границах данных классов.

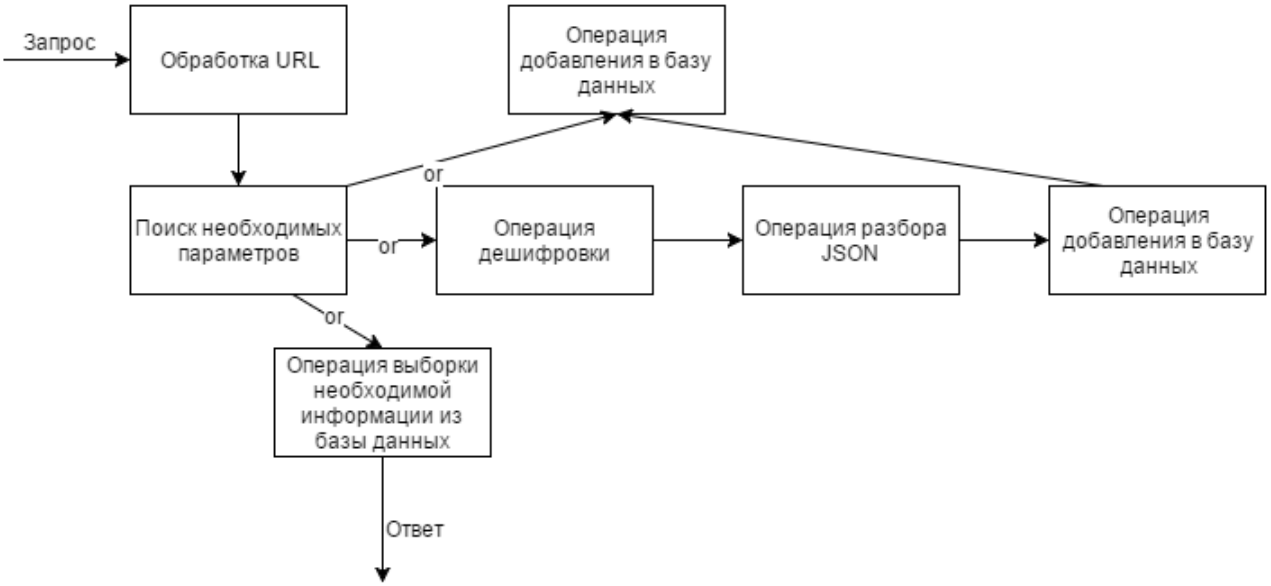

Рисунок 2.9 – Диаграмма высокого уровня абстракции проектируемого приложения

Данная диаграмма на рисунке 2.9 представляет собой процесс операций, начиная с точки «Запрос». После обработки URL происходит поиск необходимых параметров, для определения дальнейший действий:

— ответ с информацией из базы данных;

— добавление в базу данных;

— операций дешифровки, разбора JSON и распределение информации в базу данных.

Создадим пустые тела функций и классов и определим между ними связь.

В первую очередь нам необходим класс, с помощью которого можно принять и отдать данные на обработку для обмена данными с базой данных. Для этого необходимо создать класс GreetingController для распознавания URL-запросов на сервер. Рисунок 2.10 реализует заготовку под GreetingController класс, его методы и параметры.

```
public class GreetingController {
    private ApplicationContext context = new ClassPathXmlApplicationContext("Spring-User.xml");
    MainTemplate mainTemplate = (MainTemplate)context.getBean("userDAO");
    @RequestMapping(name = "/getjson", method = RequestMethod.POST)
    public void getjson(@RequestBody String json){...}
    @RequestMapping("/getallusers")
    public List<User> getallusers() {...}
    @RequestMapping("/getuserbyid")
    public List<User> getuserbyid(int id) {...}
    @RequestMapping("/getallsensors")
    public List<Sensor> getallsensors() {...}
    @RequestMapping("/getallobjects")
    public List<Object> getallobjects() {...}
    @RequestMapping("/getallfeedbacks")
    public List<Feedback> getallfeedbacks() {...}
    @RequestMapping("/getalldevice")
    public List<Device> getalldevice() {...}
    @RequestMapping("/getallbyuser_name")
    public List<GetAll> getallbyuser_name(String user_name) {...}
    @RequestMapping("/insertuser")
    public void insertuser(String user_name, String password, String email) {...}
    @RequestMapping("/insertobject")
    public void insertuser(String object_name, String user_id) {...}
    @RequestMapping("/insertdevice")
    public void insertdevice(String device_name, String object_id) {...}
    @RequestMapping("/insertsensor")
    public void insertsensor(String sensor_name, int device_id, int exampledata1, int exampledata2) {...}
    @RequestMapping("/getencrypt")
    public String getencrypt(String content){...}
    @RequestMapping("/getdecrypt")
    public String getdecrypt(String content){...}
}
```

Рисунок 2.10 – Заготовка класса GreetingController

Необходимо создать класс для манипуляций с запросами к базе данных. Класс MainTemplate использует методы классов *DAO (Data Access Object), которые содержат все необходимые запросы для обмена данными с одноименными классам таблицам. Рисунок 2.11 демонстрирует заготовку для класса MainTemplate, его методы и параметры.

```
public class MainTemplate
{
    private DataSource dataSourcemysql;
    private JdbcTemplate jdbcTemplateObjectMySQL;
    public void insertsensor(String sensor_name, int device_id, int exampledata1, int exampledata2){...}
    public void setDataSourcemysql(DataSource dataSource) {...}
    public List<User> getAllUsers() {...}
    public List<User> getbyid(int id) {...}
    public List<GetAll> getallbyusername(String user_name) {...}
    public List<Sensor> getAllSensors(){...}
    public List<Object> getAllObjects(){...}
    public List<Feedback> getAllFeedbacks(){...}
    public List<Device> getAllDevice(){...}
    public void insertuser(String user_name, String password, String email){...}
    public void insertobject(String object_name, String user_id){...}
    public void insertdevice(String device_name, String object_id){...}
}
```

Рисунок 2.11 – Заготовка класса MainTemplate

Далее требуется создать классы *DAO, которые обрабатывают данные перед их отправкой в базу данных и выполняют соответствующие запросы. Так же они позволяют делать выборки данных при помощи заготовленных SQL запросов. На рисунке 2.12 продемонстрирована заготовка этого класса. Для удобства были созданы, аналогичные таблицам в базе данных, классы для промежуточного хранения и последующей обработки, при помощи *DAO классов, данных. На рисунке 2.13 показана заготовка одного из данных классов.

```
public class UserDAO {
    private JdbcTemplate jdbcTemplateObjectMySQL;
    private SimpleJdbcInsert jdbcInsertMySQL;
    public UserDAO(JdbcTemplate jdbcTemplateMySQL) {...}
    public List<User> findAllMySQL() {...}
    public List<User> findbyid(int id) {...}
    public void insertuser(String user_name, String password, String email) {...}
}
```

Рисунок 2.12 – Заготовка класса UserDAO

```
public class User
{
    int id;
    String user_name;
    String password;
    String email;
    //getter and setter methods
    public int getId() {...}

    public String getEmail() {...}

    public String getPassword() {...}

    public String getUser_name() {...}

    public void setEmail(String email) {...}

    public void setId(int id) {...}

    public void setPassword(String password) {...}

    public void setUser_name(String user_name) {...}
}
```

Рисунок 2.13 – Заготовка класса User

Классы *RowMapper по результатам запроса могут распределять данные из запроса, которые находятся в переменной ResultSet. Далее при помощи

стандартных средств Spring JDBC Template результаты запроса записываются в используемые в качестве промежуточных хранилищах информации классах, описанных ранее. Правила передачи значений из результата запроса в класс промежуточного хранения данных описываются в *RowMapper классах. На рисунке 2.14 продемонстрирована заготовка класса UserRowMapper.

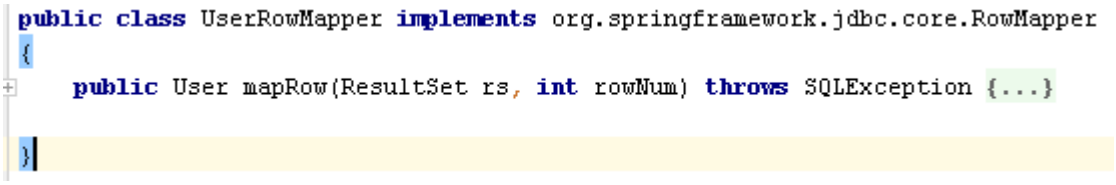

Рисунок 2.14 – Заготовка класса UserRowMapper

В результате конструирования общей архитектуры серверного приложения, проектирования базы данных, а также проектирования архитектуры компонентов, было спроектировано серверное приложение информационной системы «Умный дом», согласно со сформированными в первой главе требованиями.

Глава 3 Реализация серверной части информационной системы «Умный дом»

3.1 Кодирование серверного приложения

В данном параграфе будут описаны этапы кодирования и реализации компонентов серверной части информационной системы «Умный дом», в соответствии с составленными требованиями.

3.1.1 Реализация базы данных

В мире существует множество СУБД, исследование всех не входит в рамки данный выпускной-квалификационной работы.

Мы рассмотрим наиболее популярные из них.

— Oracle Database 11g Standard Edition

Oracle Database 11g единственная СУБД, которая предназначена для распределения вычислительных сред (Grid). Данная система дает клиентам возможность упрощенного управления корпоративной информацией, что положительно сказывается на процессе ведения бизнеса и внедрения инновационных решений.

Oracle Database 11g обеспечивает высокие показатели следующих характеристик: производительность, масштабируемость, доступность, безопасность и удобство управления в сетях распределенными вычислениям на базе стандартных серверов и систем хранения.

СУБД Oracle Database 11g необходима для эффективного развертывания на базе различных типов оборудования: будь то небольшие blade-серверы или мощные симметричные многопроцессорные системы и кластеры.

Данная СУБД обеспечит удобную и эффективную автоматизацию управления и эксплуатации. Oracle Database 11g - идеальное решение для эффективной реализации OLTP приложений, хранилищ данных и систем управления контентом.

Стоимость СУБД— от 100 тысяч рублей.

— Firebird

Firebird (FirebirdSQL) — компактная, кроссплатформенная, свободная система управления базами данных (СУБД). Работает на GNU/Linux, Microsoft Windows и разнообразных Unix платформах.

Преимущества Firebird: многоверсионная архитектура, которая обеспечивает параллельную обработку оперативных и аналитических запросов; компактность (дистрибутив 5МЬ), высокая эффективность и мощная языковая поддержка для хранимых процедур и триггеров.

Среди недостатков: отсутствует кеш результатов запросов и полнотекстовых индексов.

Стоимость — бесплатно.

— MySQL

MySQL—свободная СУБД. Является собственностью Sun Microsystems – компании, которая разрабатывает и поддерживает серверные приложения. MySQL распространяется под лицензией GNU General Public License и под своей собственной коммерческой. Также есть возможность получения СУБД по индивидуальному заказу функционала лицензионных клиентов. Так в первых версиях MySQL появился механизм репликации.

Стоимость простой лицензии MySQL Enterprise на год — 600 у.е., или 18 тысяч рублей.

Стоимость лицензии на MySQL Community edition — бесплатно.

Если сравнивать описанные выше системы управления базами данных, то Oracle является неприемлемым с точки зрения стоимости лицензии.

СУБД FireBird по сравнению с MySQL похожи по функционалу. Но у MySQL есть преимущество перед FireBird ввиду поддержки кеширования запросов.

Также у FireBird меньше сообщество в сети Интернет. Статистика представлена на рисунке 3.1. А от этого, как следствие, увеличиваются временные затраты на поиск и решение возможных проблем.

У MySQL двойное лицензирование. Данная СУБД может распространяться по лицензии GPL, но тогда разработчики должны обеспечить свободный доступ к коду своей базы данных. Либо, можно воспользоваться коммерческой лицензией, которая не требует свободного доступа к коду, и также обеспечивает сервисную поддержку.

Это означает, что в рамках разрабатываемого приложения возможно использовать бесплатную версию MySQL Community Edition.

MySQL имеет API для языков Delphi, C, C++, Эйфель, Java, Лисп, Perl, PHP, Python, Ruby, Smalltalk и Tcl, библиотеки для языков платформы .NET, а также обеспечивает поддержку для JDBC посредством JDBC-драйвера.

Сравним динамику популярности двух наиболее подходящих СУБД на рисунке 3.1.

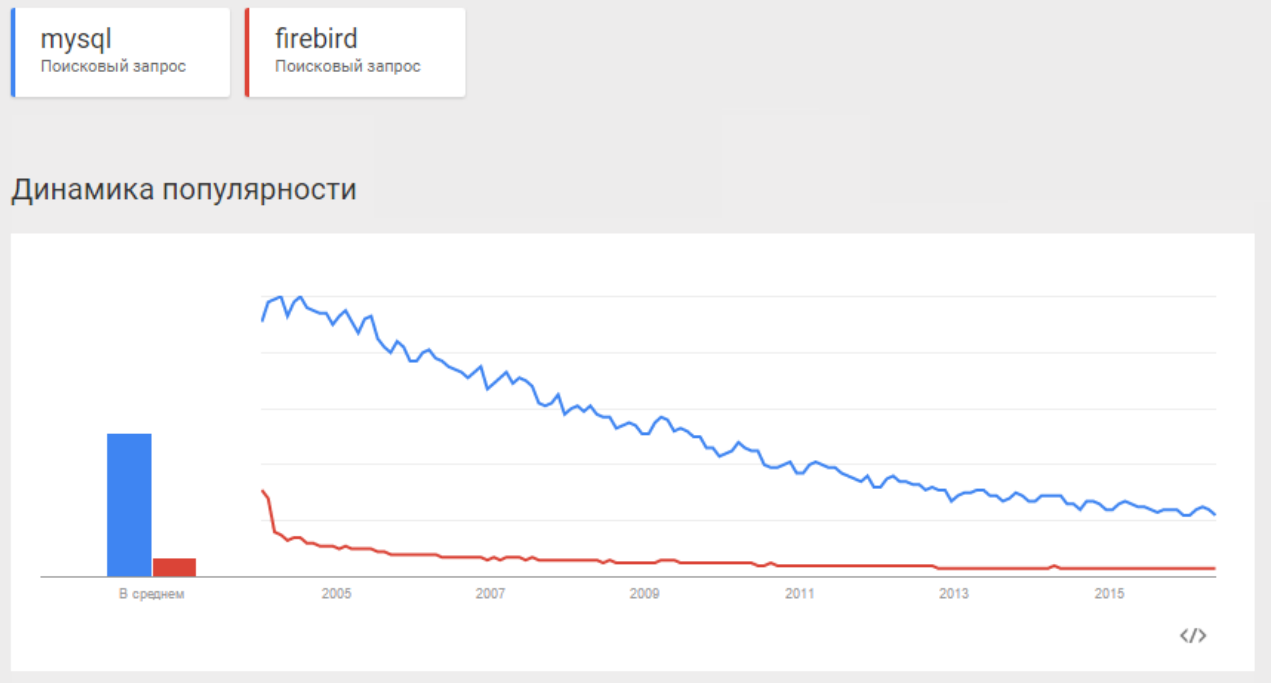

Рисунок 3.1Сравнение мирового поискового трафика по запросам «mysql» и «firebird»

Для разработки серверного приложения ИС «Умный дом» мы остановились на бесплатной версии СУБД MySQL Community Edition.

Реализация базы данных была совершена посредством системы управления базами данных MySQL workbench несколькими этапами.

На первом этапе была создана схема под названием arduinomydb. Ее вид изображен на рисунке 3.2.

Рисунок 3.2 – Схема базы данных

На втором этапе были созданы таблицы, поля и установлены параметры полей, посредством функционала инструмента визуального проектирования баз данныхMySQL workbench, который изображен на рисунке 3.3.

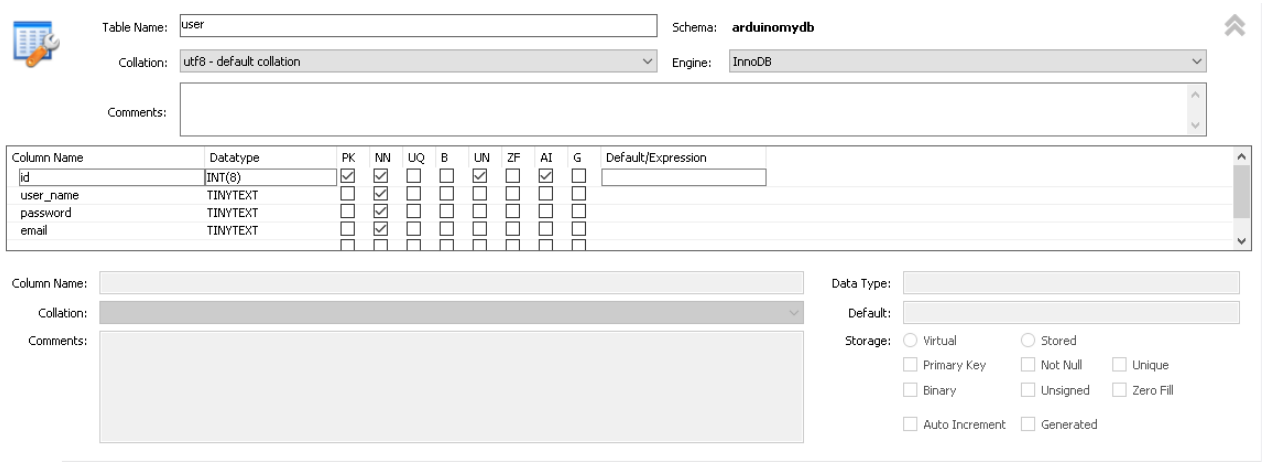

Рисунок 3.3 – Функционал создания таблиц MySQL workbench

На третьем этапе были установлены связи между таблицами, при помощи внешних ключей. Функционал создания связей между таблицами СУБД MySQL workbench показан на рисунке 3.4.

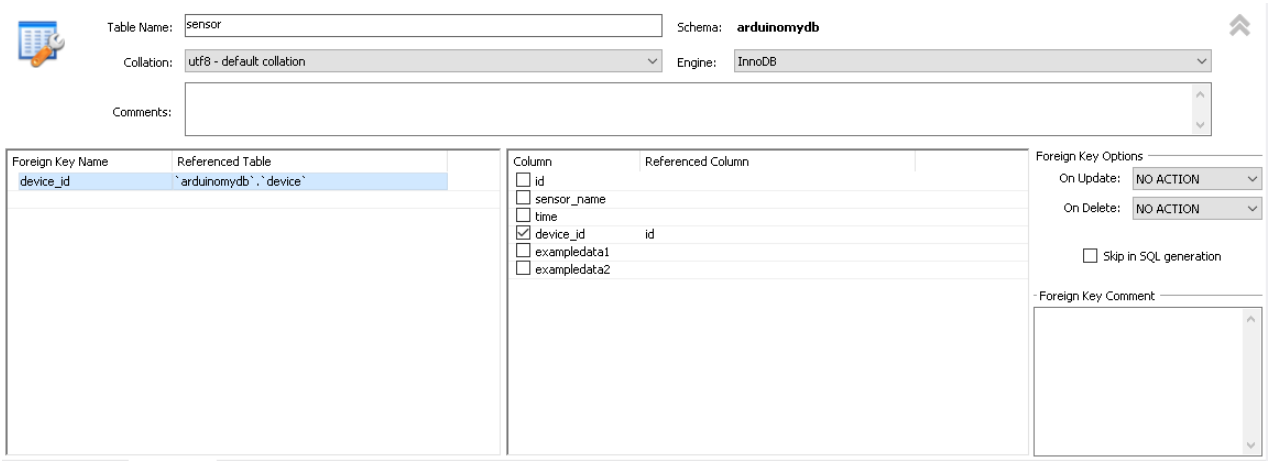

Рисунок 3.4 – Функционал создания связей между таблицами MySQL workbench

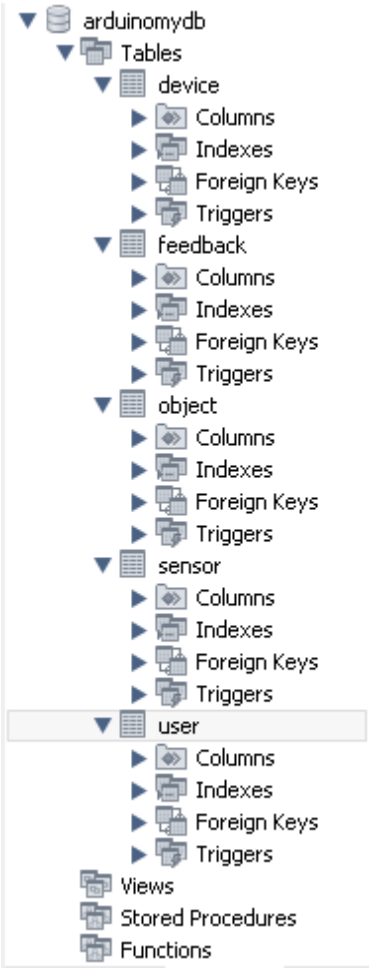

Рисунок 3.5 – Готовый вид схемы, созданной в MySQL workbench

В результате создания всех таблиц, а также создания связей между ними, мы получили готовую схему, соответствующую составленным требованиям, ее конечный вид представлен на рисунке 3.5.

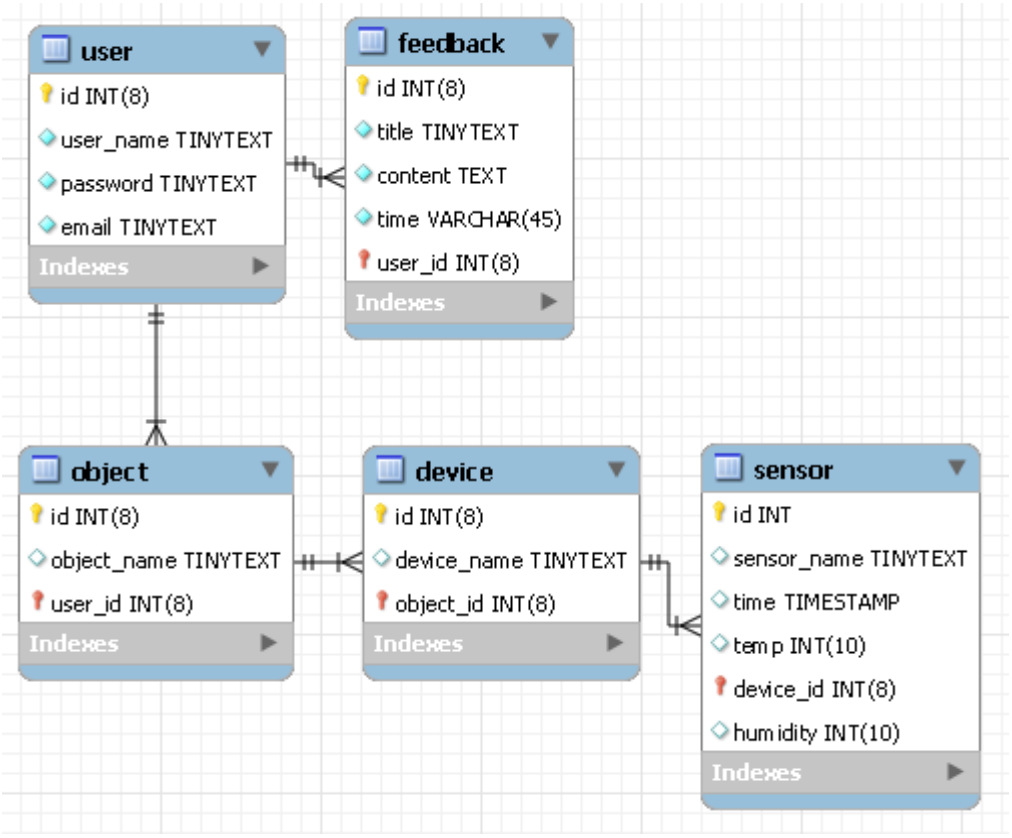

Рисунок 3.6 – Физическая модель базы данных

На рисунке 3.6 предоставлена физическая модель базы данных.

3.1.2 Реализация компонентов серверного приложения

Для серверного приложения будет использоваться Java Development Kit. Java Development Kit, сокращенно JDK — бесплатно распространяемый компанией Oracle Corporation (ранее Sun Microsystems) комплект разработчика приложений на языке Java. Данный комплект включает компилятор Java (javac), стандартные библиотеки классов Java, документацию, утилиты и исполнительную систему Java (JRE). Интегрированная среда разработки на Java в состав JDK не входит, поэтому мы будем использовать такую IDE, как IntelliJ Idea. Так же был использован фреймворк Spring и сборщик проектов Maven

На первом этапе была создана структура каталогов, которая имеет вид, показанный на рисунке 3.7.

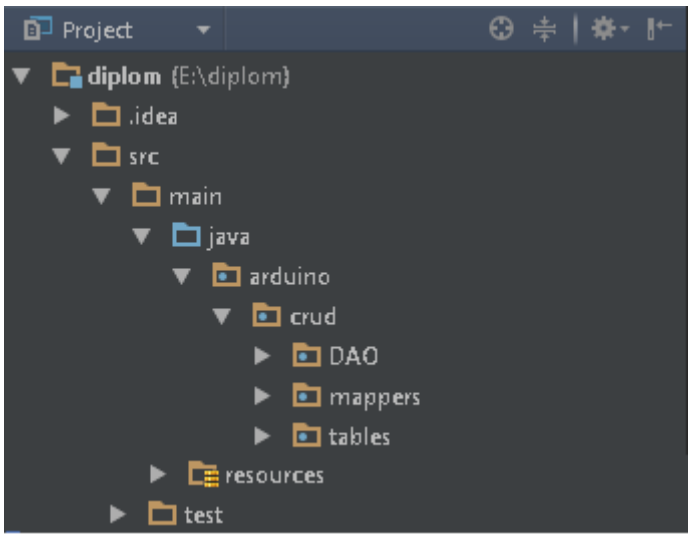

Рисунок 3.7 – Структура каталогов проекта в IDE IntelliJ IDEA

На втором этапе мы создаем сборку Maven, для этого нам необходимо создать определение Maven–проекта, который формируется как XML–файл с названием pom.xml в корневом каталоге проекта. Помимо всего прочего, файл pom.xml определяет имя проекта, версию, а также зависимости от сторонних библиотек.

На третьем этапе мы создаем классы, соответствующие составленным требованиям. После этого структура каталогов будет выглядеть, как показано на рисунке 3.8.

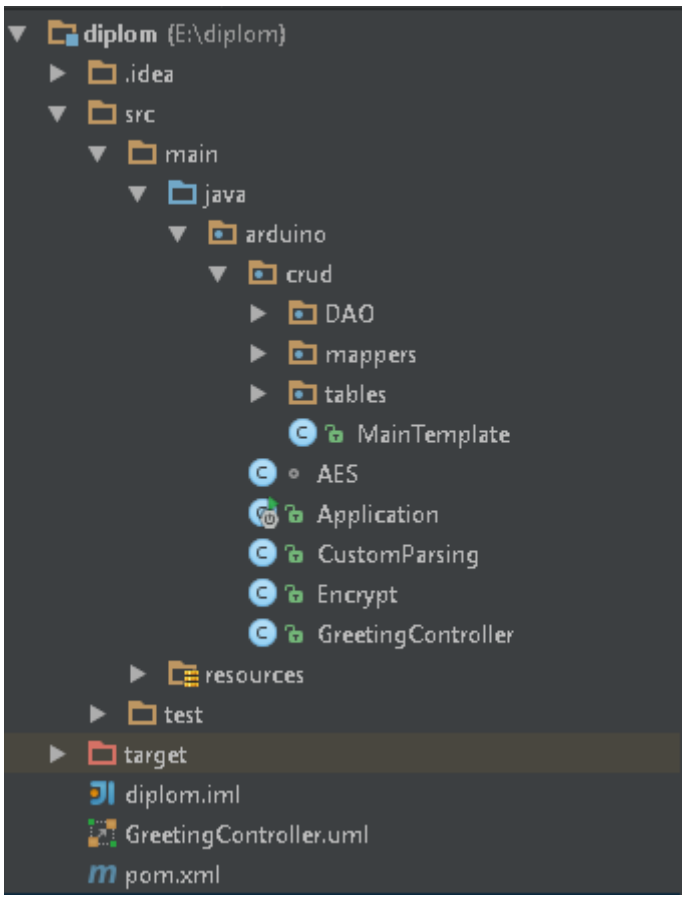

Рисунок 3.8 – Структура каталогов проекта с классами и pom.xml

В каталогах DAO mappers и tables хранятся классы с соответствующими названиями *DAO, *RowMapper и названиями таблиц базы данных.

На четвертом этапе мы программируем класс GreetingController. Этот класс обрабатывает GET и POST запросы. На рисунке 3.9 представлен частичный код реализации данного класса. Полный код данного класса предоставлен в приложении.

```
public class GreetingController {

    private ApplicationContext context = new ClassPathXmlApplicationContext("Spring-User.xml");
    MainTemplate mainTemplate = (MainTemplate)context.getBean("userDAO");

    @RequestMapping(name = "/getjson", method = RequestMethod.POST)
    public void getjson(@RequestBody String json){
        Sensor sensor = CustomParsing.parseSensors(json);
        mainTemplate.insertsensor(
                sensor.getSensor_name(),
                sensor.getDevice_id(),
                sensor.getExampledata1(),
                sensor.getExampledata2());
    }

    @RequestMapping("/getallusers")
    public List<User> getallusers() { return mainTemplate.getAllUsers(); }

    @RequestMapping("/getuserbyid")
    public List<User> getuserbyid(int id) { return mainTemplate.getbyid(id); }

    @RequestMapping("/getallsensors")
    public List<Sensor> getallsensors() { return mainTemplate.getAllSensors(); }

    @RequestMapping("/getallobjects")
    public List<Object> getallobjects() { return mainTemplate.getAllObjects(); }
```

Рисунок 3.9 – Частичный код реализации класса GreetingController

На пятом этапе мы программируем класс MainTemplate. Этот класс содержит методы операций с базами данных, используя классы *DAO. На рисунке 3.10 представлен частичный код реализации данного класса. Полный код данного класса предоставлен в приложении.

```
public class MainTemplate
{
    private DataSource dataSourcemysql;
    private JdbcTemplate jdbcTemplateObjectMySQL;

    public void insertsensor(String sensor_name, int device_id, int exampledata1, int exampledata2){
        SensorDAO sensorDAO = new SensorDAO(this.jdbcTemplateObjectMySQL);
        sensorDAO.insertsensor(sensor_name, device_id, exampledata1, exampledata2);
    }

    public void setDataSourcemysql(DataSource dataSource) {
        this.jdbcTemplateObjectMySQL = new JdbcTemplate(dataSource);
    }

    public List<User> getAllUsers() {
        UserDAO userDAO = new UserDAO(this.jdbcTemplateObjectMySQL);
        return userDAO.findAllMySQL();
    }

    public List<User> getbyid(int id) {
        UserDAO userDAO = new UserDAO(this.jdbcTemplateObjectMySQL);
        return userDAO.findbyid(id);
    }

    public List<GetAll> getallbyusername(String user_name) {
        SensorDAO sensorDAO = new SensorDAO(this.jdbcTemplateObjectMySQL);
        return sensorDAO.findallbyname(user_name);
    }
```

Рисунок 3.10 – Частичный код реализации класса MainTemplate

На шестом этапе мы программируем классы *DAO. Эти классы содержат методы, применяемые к соответствующим названиям классов таблицам. На рисунке 3.11 представлен код реализации класса UserDAO. Ввиду того, что методы данных классов содержат практически идентичный код, нет необходимости описывать все классы *DAO в данном параграфе. Полный код данных классов предоставлен в приложении.

```
public class UserDAO {

    private JdbcTemplate jdbcTemplateObjectMySQL;
    private SimpleJdbcInsert jdbcInsertMySQL;

    public UserDAO(JdbcTemplate jdbcTemplateMySQL) {
        this.jdbcTemplateObjectMySQL = jdbcTemplateMySQL;
        this.jdbcInsertMySQL = new SimpleJdbcInsert(jdbcTemplateMySQL).withTableName("user");
    }

    public List<User> findAllMySQL() {
        return this.jdbcTemplateObjectMySQL.query("select * from `user`", new UserRowMapper());
    }

    public List<User> findbyid(int id) {
        return this.jdbcTemplateObjectMySQL.query("select * from `user` WHERE id =" +id, new UserRowMapper());
    }

    public void insertuser(String user_name, String password, String email) {
        SqlParameterSource parameters = new MapSqlParameterSource()
                .addValue("user_name", user_name)
                .addValue("password", password)
                .addValue("email", email);
        this.jdbcInsertMySQL.execute(parameters);
    }
}
```

Рисунок 3.11 – Код реализации класса UserDAO

На седьмом этапе мы программируем классы *RowMapper.Эти классы определяют выборку результирующего набора данных вследствие запроса SELECT. На рисунке 3.12 представлен код реализации класса UserRowMapper. Ввиду того, что методы данных классов содержат практически идентичный код, нет необходимости описывать все классы *RowMapper в данном параграфе. Полный код данных классов предоставлен в приложении.

```
public class UserRowMapper implements org.springframework.jdbc.core.RowMapper
{
    public User mapRow(ResultSet rs, int rowNum) throws SQLException {
        User user = new User();
        user.setId(rs.getInt("id"));
        user.setUser_name(rs.getString("user_name"));
        user.setPassword(rs.getString("password"));
        user.setEmail(rs.getString("email"));
        return user;
    }

}
```

Рисунок 3.12 – Код реализации класса UserRowMapper

На восьмом этапе мы программируем классы Device, Feedback, Object, GetAll, Sensor и User. В этих классах описаны их параметры, которые идентичны столбцам в соответствующих таблицах, а также их get и set методы.

На рисунке 3.13 представлен код реализации класса User. Ввиду того, что методы данных классов содержат практически идентичный код, нет необходимости описывать все классы этого типа в данном параграфе. Полный код данных классов предоставлен в приложении.

```
public class User
{
    int id;
    String user_name;
    String password;
    String email;
    //getter and setter methods
    public int getId() { return id; }

    public String getEmail() { return email; }

    public String getPassword() { return password; }

    public String getUser_name() { return user_name; }

    public void setEmail(String email) { this.email = email; }

    public void setId(int id) { this.id = id; }

    public void setPassword(String password) { this.password = password; }

    public void setUser_name(String user_name) { this.user_name = user_name; }
}
```

Рисунок 3.13 – Код реализации класса User

На девятом этапе мы программируем класс CustomParsing.Этот класс разбирает JSON сообщение в представлении String на список: ключ–значение. На рисунке 3.14 предоставлен код реализации класса CustomParsing.

```
public class CustomParsing {
    static Sensor parseSensors(String json){
        Sensor sensor = new Sensor();
        BasicJsonParser basicJsonParser = new BasicJsonParser();
        Map<String, Object> parseMap = basicJsonParser.parseMap(json);
        sensor.setSensor_name(parseMap.get("sensor_name").toString());
        sensor.setDevice_id(Integer.parseInt(parseMap.get("device_id").toString()));
        sensor.setExampledata1(Integer.parseInt(parseMap.get("exampledata1").toString()));
        sensor.setExampledata2(Integer.parseInt(parseMap.get("exampledata2").toString()));
        return sensor;
    }
}
```

Рисунок 3.14 – Код реализации класса CustomParsing

В результате создания и программирования всех классов, мы получили готовое серверное приложение для информационной системы «Умный дом», соответствующее составленным требованиям.

3.2 Тестирование серверного приложения

Для того, чтобы начать непосредственно тестирование, необходимо построить приложение Spring через сборщика Maven.

Тестирование созданного серверного приложения проводилось посредством страницы, показанной на рисунке 3.15, формата HTML, с поддержкой JavaSqript, описывающей в себе различные GET запросы.

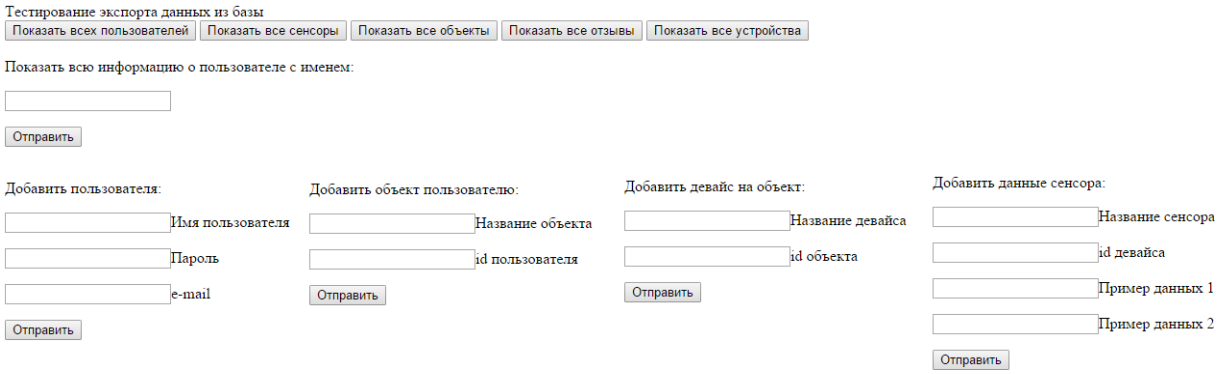

Рисунок 3.15 – Тестовая страница index.html

На первом этапе проверим экспорт информации из базы данных. Поочередно кликнем на кнопки «Показать всех пользователей», «Показать все сенсоры», «Показать все объекты», «Показать все отзывы» и «Показать все устройства». Результат выполнения запроса мы видим на рисунке 3.16.

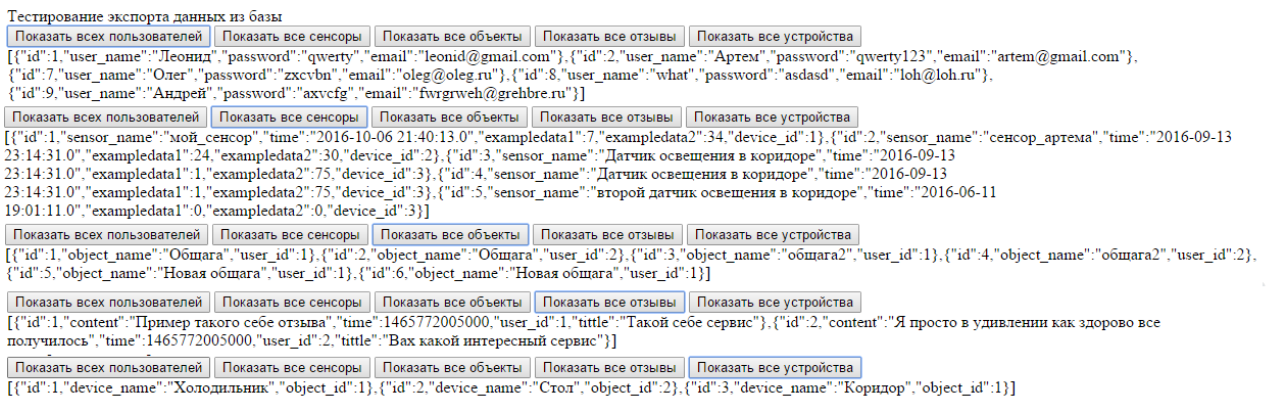

Рисунок 3.16 – Тест экспорта данных из базы

Все запросы были успешно выполнены. Предоставлена вся запрашиваемая информация.

На втором этапе проверим вывод всей информации о пользователе, его объектах, его устройствах и его сенсорах. Воспользуемся формой «Показать всю информацию о пользователе с именем:» и, в качестве примера, введем в это поле имя «Леонид».

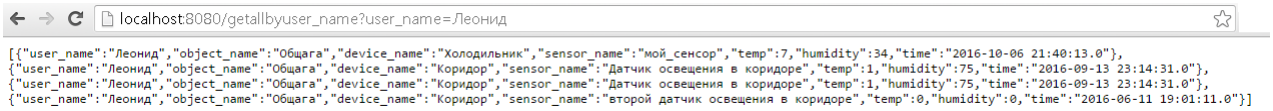

Рисунок 3.17 – Тест экспорта информации о конкретном пользователе

Рисунок 3.17 свидетельствует о том, что запрос был успешно выполнен. Предоставлена вся информация, принадлежащая пользователю с вводимым именем.

На третьем этапе проверим импорт данных в базу данных. Для этого воспользуемся формами «Добавить пользователя», «Добавить объект пользователю», «Добавить девайс на объект» и «Добавить данные сенсора». Введем в поля всю необходимую информацию поочередно в каждой форме и проверим добавилась ли информация в базу данных, используя кнопки «Показать всех пользователей», «Показать все сенсоры», «Показать все объекты», «Показать все отзывы» и «Показать все устройства». На рисунке 3.18 мы видим полученную в результате запроса информацию.

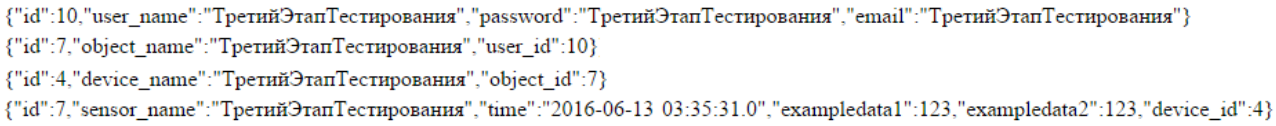

Рисунок 3.18 – Информация, полученная путем ввода и запроса информации из базы данных

Все запросы были успешно выполнены. Успешно получена вся запрашиваемая информация.

3.3 Развертывание серверного приложения для информационной системы «Умный дом»

Перед развертыванием серверного приложения необходимо определиться в выборе технической платформы. Это может быть собственная серверная машина, либо арендованный хостинг для нашего приложения.

Серверное приложение, написанное с помощью фреймворка Spring [26] может быть развернуто двумя способами:

— путем копирования всех файлов из корневого каталога среды разработки на серверную машину запуска исполняемого jar приложения arduino-1.0-SNAPSHOT из папки target в корневом каталоге, так же потребуется создать дамп базы данных и разместить его на серверной машине;

— путем предварительной сборки приложения и копирования его на арендованный хостинг при помощи протокола FTP, так же потребуется создать дамп реализованной базы данных и перенести его на хостинг при помощи внутреннего интерфейса хостинга, так же актуален вариант аренды отдельного хостинга специально для базы данных, для этого необходимо предварительно указать в настройках системы адрес снимаемого сервера.

В результате выполненных действий с IDE IntelliJ IDEA и MySQL workbench, было разработано и протестировано серверное приложение для информационной системы «Умный дом», а также были предложены варианты его развертывания.

Заключение

В данной выпускной квалификационной работе был выделен объект исследования, проанализирована предметная область, определены цели и задачи работы. Проведен обзор программных средств, позволяющих реализовать серверное приложения для информационной системы «Умный дом».

Анализ программных средств показал, что для решения задач данной выпускной квалификационной работы требуется разработать свое приложение, а не использовать готовые варианты. Поэтому был произведен выбор в пользу Java и MySQL. С помощью MySQL workbench разработана архитектура базы данных. Определены необходимые поля базы данных, их структура и тип.

Была спроектирована база данных, состоящая из 5 таблиц, в которых отражены основные атрибуты – соответствующие поля, необходимые для автоматизации и учета информации с датчиков. Была спроектирована и реализована архитектура компонентов серверного приложения информационной системы «Умный дом». Реализован алгоритм, который принимает данные в зашифрованном виде, дешифрует эти данные, разбирает сообщение формата JSON на параметры и распределяет полученную информацию в базу данных.

Таким образом, мы получили готовое к развертыванию и работе серверное приложение для информационной системы «Умный дом».

Список используемой литературы

Периодические издания

1. Авдеев, А. С. Разработка систем автоматизации жилых и офисных помещений «Умный Дом» [Текст] // Сборник научных трудов студентов «Катановские чтения» – 2014». – 2014. – С. 142–143.

2. Авдеев, А. С. Основные проблемы программирования систем «Умного Дома» [Текст] / А. С. Авдеев, А. И. Герасимова // Перспективы науки. –2014. – С. 62–65.

3. Атрощенко, В. А. К вопросу формирования данных систем управления умного дома [Текст] / В. А. Атрощенко, С.Е Кошевая, М.В. Серикова // Современные проблемы науки и образования. – 2014. – № 5.; URL: http://www.science–education.ru/ru/article/view?id=15067

4. Веллинг, Л. Разработка Web–приложений с помощью PHP и MySQL [Текст] / Л. Веллинг, Л. Томпсон. – 3–е изд. – М.: «Вильямс». – 2013. – 880 с.

5. Ву Т. З. Анализ систем автоматизированного управления умным домом [Текст] // Молодой ученый. – 2011. – №4. Т.1. – С. 28–31.

6. МакГрат, М. Программирование на Java для начинающих [Текст] / М. МакГрат. – М.: Эксмо. – 2016. – 192 с.

7. Маклафлин, Б. PHP и MySQL Исчерпывающее руководство [Текст] / перевод О. Сивченко – СПб: Веб–технологии, – 2014 – С. 114 – 158.

8. Пономаренко, В. И. Использование платформы Arduino в измерениях и физическом эксперименте [Текст] / В. И. Пономаренко, А. С. Карасев // ПНД. –2014. –Т. 22. –No. 4. – С. 77–90.

9. Шалыто, А. А., Шопырин Д. Г. Синхронное программирование [Текст] //Информационно–управляющие системы. – 2014 – С. 35–42.

10. Янк, К. PHP и MySQL. От новичка к профессионалу [Текст] / Я. Кевин, – СПб.: Веб–технологии. – 2013 – С. 243 – 270.

Учебники и учебные пособия

11. Информационно–вычислительные сети : учебное пособие / Д. А. Капустин, В. Е. Дементьев. – Ульяновск : УлГТУ. – 2011. – 141 с.

12. Чулюков, В. А. Информационные системы (конспект лекций). [Текст] / А. В. Чулюков. – Воронеж: ВГПУ, 2012. – 57 с.

Электронные ресурсы

13. «Умный дом» как сочетание стиля, функциональности и комфорта. Наука и технологии России [Электронный ресурс] Режим доступа: http://strf.ru.

14. Байгозин, Д. В. Концепция интегрированной системы автоматизации проектирования и моделирования инженерного оборудования зданий [Электронный ресурс] / Д. В. Байгозин, Г. Б. Захарова, В. Ю. Плотников // Научный электронный архив. URL: http://econf.rae.ru/article/4810

15. Кадырова Л. Ш. УМНЫЙ ДОМ: идеология или технология. [Электронный ресурс]: Международный научно–исследовательский журнал. Режим доступа: http://research–joumal.org/featured/arch/umnyj–dom–ideologiya–ili–texnologiya/

16. Компания Nginx, Inc. [Офиц. сайт]. URL: nginx.org/ru

17. Компания Apache HTTP Server Project [Офиц. сайт]. URL: httpd.apache.org

Литература на иностранных языках

18. Baron Schwartz, Peter Zaitsev, Vadim Tkachenko High Performance MySQL, 3rd Edition Optimization, Backups, and Replication. – California, USA, 2012 – pp. 88 – 96.

19. Chadwick, J. ASP.NET MVC 4: Developing Real–World Applications with ASP.NET MVC./ J. Chadwick, T. Snyder, H. Panda – M.: Williams. – 2013. – 432 p.

20. Williams, N. S. Professional Java for Web Applications [Text] / N. S. Williams – Indianapolis, Indiana, : John Wiley & Sons, Inc. – 2014. – 865 с.

21. Pro Spring MVC: With Web Flow (Expert's Voice in Spring) [Text] / M. Deinum, K. Serneels, C. Yates and other, – New York : Apress. – 2012. – 596 p.

22. Sattari, H. Spring Web Services 2 Cookbook Paperback [Text] / H. Sattari, . Kunjumohamed. – UK : Packt Publishing. – 2012.–322 p.

23. *S*pring Data [Text] / M. Pollack, O. Gierke, T. Risberg and other, – California: O'Reilly Media. – 1st edition. – 316 p.

24. Schwartz, B. High Performance MySQL [Text] / B. Schwartz, V. Tkachenko V. – 3rd edition optimization. – California (USA) : Backups, and Replication. – 2012 – pp. 88 – 96.

25. Walls, Cr. Spring in Action [Text] / Cr. Walls, – New York : Manning Publications. – 3rd edition. – 2011. – 424 p.

26. Wheeler, W. Spring in Practice [Text] / W. Wheeler, J. White. – New York : Manning Publications. – 1st edition. – 2013. – 560 p.

Приложение

Листинг кода разработанного приложения

Код реализации класса GreetingController

```
      @RestController
@RequestMapping(name = "", method = RequestMethod.GET)
public class GreetingController {
    private ApplicationContext context = new
ClassPathXmlApplicationContext("Spring-User.xml");
    MainTemplate mainTemplate = (MainTemplate)context.getBean("userDAO");
    @RequestMapping(name = "/getjson", method = RequestMethod.POST)
    public void getjson(@RequestBody String json){
        Sensor sensor = CustomParsing.parseSensors(json);
        mainTemplate.insertsensor(
                sensor.getSensor_name(),
                sensor.getDevice_id(),
                sensor.getExampledata1(),
                sensor.getExampledata2());
    }
    @RequestMapping("/getallusers")
    public List<User> getallusers() {
        return mainTemplate.getAllUsers();
    }
    @RequestMapping("/getuserbyid")
    public List<User> getuserbyid(int id) {
        return mainTemplate.getbyid(id);
    }
    @RequestMapping("/getallsensors")
    public List<Sensor> getallsensors() {
        return mainTemplate.getAllSensors();
    }
    @RequestMapping("/getallobjects")
    public List<Object> getallobjects() {
        return mainTemplate.getAllObjects();
    }
    @RequestMapping("/getallfeedbacks")
    public List<Feedback> getallfeedbacks() {
        return mainTemplate.getAllFeedbacks();
    }
    @RequestMapping("/getalldevice")
    public List<Device> getalldevice() {
        return mainTemplate.getAllDevice();
    }
    @RequestMapping("/getallbyuser_name")
    public List<GetAll> getallbyuser_name(String user_name) {
        return mainTemplate.getallbyusername(user_name);
    }
    @RequestMapping("/insertuser")
    public void insertuser(String user_name, String password, String email) {
        mainTemplate.insertuser(user_name, password, email);
    }
    @RequestMapping("/insertobject")
    public void insertuser(String object_name, String user_id) {
        mainTemplate.insertobject(object_name, user_id);
    }
    @RequestMapping("/insertdevice")
    public void insertdevice(String device_name, String object_id) {
        mainTemplate.insertdevice(device_name, object_id);
    }
    @RequestMapping("/insertsensor")
    public void insertsensor(String sensor_name, int device_id, int
```

```
exampledata1, int exampledata2) {
        mainTemplate.insertsensor(sensor_name, device_id, exampledata1,
exampledata2);
```

Продолжение приложения

```
    }
    @RequestMapping("/getencrypt")
    public String getencrypt(String content){
        try {
            return "<!DOCTYPE html>\n" +
                    "<html lang=\"en\">\n" +
                    "<head>\n" +
                    "    <meta charset=\"UTF-8\">\n" +
                    "    <title></title>\n" +
                    "    <script>\n" +
                    "        onload=function(){\n" +
                    "            document.getElementById(\"container\").value =
\"tut podgruzochka\";\n" +
                    "        };\n" +
                    "        function httpGet(theUrl)\n" +
                    "        {\n" +
                    "            var xmlHttp = new XMLHttpRequest();\n" +
                    "            xmlHttp.open( \"GET\", theUrl, false ); //
false for synchronous request\n" +
                    "            xmlHttp.send( null );\n" +
                    "            //console.log(xmlHttp.response);\n" +
                    "
document.getElementById(\"container\").innerText = xmlHttp.response;\n" +
                    "        }\n" +
                    "    </script>\n" +
                    "</head>\n" +
                    "<body>\n" +
                    "<a id=\"container\">\n" +
                    "\n" +
                    "<a href=\"/getdecrypt?content=" + Encrypt.encrypt(content)
+"\">НАЖМИ НА МЕНЯ</a>\n" +
                    "</body>\n" +
                    "</html>";
        }
        catch (GeneralSecurityException e){
            return e.getMessage();
        }
    }
    @RequestMapping("/getdecrypt")
    public String getdecrypt(String content){
        try {
            return Encrypt.decrypt(content);
        }
        catch (GeneralSecurityException e){
            return e.getMessage();
        }
    }
}
```

Код реализации класса MainTemplate

```
public class MainTemplate
{
   private DataSource dataSourcemysql;
   private JdbcTemplate jdbcTemplateObjectMySQL;
    public void insertsensor(String sensor_name, int device_id, int
exampledata1, int exampledata2){
        SensorDAO sensorDAO = new SensorDAO(this.jdbcTemplateObjectMySQL);
```

```
        sensorDAO.insertsensor(sensor_name, device_id, exampledata1,
exampledata2);
    }
```

Продолжение приложения

```
   public void setDataSourcemysql(DataSource dataSource) {
      this.jdbcTemplateObjectMySQL = new JdbcTemplate(dataSource);
   }
   public List<User> getAllUsers() {
      UserDAO userDAO = new UserDAO(this.jdbcTemplateObjectMySQL);
      return userDAO.findAllMySQL();
   }
    public List<User> getbyid(int id) {
        UserDAO userDAO = new UserDAO(this.jdbcTemplateObjectMySQL);
        return userDAO.findbyid(id);
    }
    public List<GetAll> getallbyusername(String user_name) {
        SensorDAO sensorDAO = new SensorDAO(this.jdbcTemplateObjectMySQL);
        return sensorDAO.findallbyname(user_name);
    }
    public List<Sensor> getAllSensors(){
        SensorDAO sensorDAO = new SensorDAO(this.jdbcTemplateObjectMySQL);
        return sensorDAO.findAllMySQL();
    }
    public List<Object> getAllObjects(){
        ObjectDAO objectDAO = new ObjectDAO(this.jdbcTemplateObjectMySQL);
        return objectDAO.findAllMySQL();
    }
    public List<Feedback> getAllFeedbacks(){
        FeedbackDAO feedbackDAO = new FeedbackDAO(this.jdbcTemplateObjectMySQL);
        return feedbackDAO.findAllMySQL();
    }
    public List<Device> getAllDevice(){
        DeviceDAO deviceDAO = new DeviceDAO(this.jdbcTemplateObjectMySQL);
        return deviceDAO.findAllMySQL();
    }
    public void insertuser(String user_name, String password, String email){
        UserDAO userDAO = new UserDAO(this.jdbcTemplateObjectMySQL);
        userDAO.insertuser(user_name, password, email);
    }
    public void insertobject(String object_name, String user_id){
        ObjectDAO objectDAO = new ObjectDAO(this.jdbcTemplateObjectMySQL);
        objectDAO.insertobject(object_name, user_id);
    }
    public void insertdevice(String device_name, String object_id){
        DeviceDAO deviceDAO = new DeviceDAO(this.jdbcTemplateObjectMySQL);
        deviceDAO.insertdevice(device_name, object_id);
    }
}
```

Код реализации класса UserDAO

```
      public class UserDAO {
    private JdbcTemplate jdbcTemplateObjectMySQL;
    private SimpleJdbcInsert jdbcInsertMySQL;
    public UserDAO(JdbcTemplate jdbcTemplateMySQL) {
        this.jdbcTemplateObjectMySQL = jdbcTemplateMySQL;
        this.jdbcInsertMySQL = new
SimpleJdbcInsert(jdbcTemplateMySQL).withTableName("user");
    }
    public List<User> findAllMySQL() {
        return this.jdbcTemplateObjectMySQL.query("select * from `user`", new
UserRowMapper());
    }
```

```
    public List<User> findbyid(int id) {
        return this.jdbcTemplateObjectMySQL.query("select * from `user` WHERE id
=" +id, new UserRowMapper());
```

Продолжение приложения

```
    }
    public void insertuser(String user_name, String password, String email) {
        SqlParameterSource parameters = new MapSqlParameterSource()
                .addValue("user_name", user_name)
                .addValue("password", password)
                .addValue("email", email);
        this.jdbcInsertMySQL.execute(parameters);
    }
}
```

Код реализации класса SensorDAO

```
  public class SensorDAO {
    private JdbcTemplate jdbcTemplateObjectMySQL;
    private SimpleJdbcInsert jdbcInsertMySQL;
    public SensorDAO(JdbcTemplate jdbcTemplateMySQL) {
        this.jdbcTemplateObjectMySQL = jdbcTemplateMySQL;
        this.jdbcInsertMySQL = new
SimpleJdbcInsert(jdbcTemplateMySQL).withTableName("sensor");
    }
    public List<Sensor> findAllMySQL() {
        return this.jdbcTemplateObjectMySQL.query("select * from sensor", new
SensorRowMapper());
    }
    public List<GetAll> findallbyname(String user_name) {
        return this.jdbcTemplateObjectMySQL.query("SELECT " +
                        "* " +
                        "from sensor s " +
                        "inner join device d on d.id=s.device_id " +
                        "INNER join object o on o.id=d.object_id " +
                        "inner join `user` u on u.id=o.user_id " +
                        "where u.`user_name` like '%"+user_name+"%';",
                new AllinfoRowMapper());
    }
    public void insertsensor(String sensor_name, int device_id, int
exampledata1, int exampledata2) {
        SqlParameterSource parameters = new MapSqlParameterSource()
                .addValue("sensor_name", sensor_name)
                .addValue("device_id", device_id)
                .addValue("exampledata1", exampledata1)
                .addValue("exampledata2", exampledata2);
        this.jdbcInsertMySQL.execute(parameters);
    }
}
```

Код реализации класса ObjectDAO

```
  public class ObjectDAO {
    private JdbcTemplate jdbcTemplateObjectMySQL;
    private SimpleJdbcInsert jdbcInsertMySQL;
    public ObjectDAO(JdbcTemplate jdbcTemplateMySQL) {
        this.jdbcTemplateObjectMySQL = jdbcTemplateMySQL;
        this.jdbcInsertMySQL = new
SimpleJdbcInsert(jdbcTemplateMySQL).withTableName("object");
    }
    public List<Object> findAllMySQL() {
        return this.jdbcTemplateObjectMySQL.query("select * from object", new
```

```
ObjectRowMapper());
    }
```

Продолжение приложения

```
    public void insertobject(String object_name, String user_id) {
        SqlParameterSource parameters = new MapSqlParameterSource()


                .addValue("object_name", object_name)
                .addValue("user_id", user_id);
        this.jdbcInsertMySQL.execute(parameters);
    }
}
```

Код реализации класса FeedbackDAO

```
      public class FeedbackDAO {
    private JdbcTemplate jdbcTemplateObjectMySQL;
    private SimpleJdbcInsert jdbcInsertMySQL;
    public FeedbackDAO(JdbcTemplate jdbcTemplateMySQL) {
        this.jdbcTemplateObjectMySQL = jdbcTemplateMySQL;
        this.jdbcInsertMySQL = new
SimpleJdbcInsert(jdbcTemplateMySQL).withTableName("feedback");
    }
    public List<Feedback> findAllMySQL() {
        return this.jdbcTemplateObjectMySQL.query("select * from feedback", new
FeedbackRowMapper());
    }
}
```

Код реализации класса DeviceDAO

```
      public class DeviceDAO {

    private JdbcTemplate jdbcTemplateObjectMySQL;
    private SimpleJdbcInsert jdbcInsertMySQL;

    public DeviceDAO(JdbcTemplate jdbcTemplateMySQL) {
        this.jdbcTemplateObjectMySQL = jdbcTemplateMySQL;
        this.jdbcInsertMySQL = new
SimpleJdbcInsert(jdbcTemplateMySQL).withTableName("device");
    }
    public List<Device> findAllMySQL() {
        return this.jdbcTemplateObjectMySQL.query("select * from device", new
DeviceRowMapper());
    }
    public void insertdevice(String device_name, String object_id) {
        SqlParameterSource parameters = new MapSqlParameterSource()
                .addValue("device_name", device_name)
                .addValue("object_id", object_id);
        this.jdbcInsertMySQL.execute(parameters);
    }
}
```

Код реализации класса UserRowMapper

```
      public class UserRowMapper implements
org.springframework.jdbc.core.RowMapper
{
    public User mapRow(ResultSet rs, int rowNum) throws SQLException {
        User user = new User();
        user.setId(rs.getInt("id"));
```

```
        user.setUser_name(rs.getString("user_name"));
        user.setPassword(rs.getString("password"));
```

Продолжение приложения

```
        user.setEmail(rs.getString("email"));
        return user;
    }
}
```

Код реализации класса SensorRowMapper

```
      public class SensorRowMapper implements
org.springframework.jdbc.core.RowMapper
{
    public Sensor mapRow(ResultSet rs, int rowNum) throws SQLException {
        Sensor sensor = new Sensor();
        sensor.setId(rs.getInt("id"));
        sensor.setSensor_name(rs.getString("sensor_name"));
        sensor.setExampledata1(rs.getInt("exampledata1"));
        sensor.setExampledata2(rs.getInt("exampledata2"));
        sensor.setTime(rs.getTimestamp("time").toString());
        sensor.setDevice_id(rs.getInt("device_id"));
        return sensor;
    }
}
```

Код реализации класса ObjectRowMapper

```
      public class ObjectRowMapper implements
org.springframework.jdbc.core.RowMapper
{
    public Object mapRow(ResultSet rs, int rowNum) throws SQLException {
        Object object = new Object();
        object.setId(rs.getInt("id"));
        object.setObject_name(rs.getString("object_name"));
        object.setUser_id(rs.getInt("user_id"));
        return object;
    }
}
```

Код реализации класса FeedbackRowMapper

```
      public class FeedbackRowMapper implements
org.springframework.jdbc.core.RowMapper
{
    public Feedback mapRow(ResultSet rs, int rowNum) throws SQLException {
        Feedback feedback = new Feedback();
        feedback.setId(rs.getInt("id"));
        feedback.setTitle(rs.getString("title"));
        feedback.setTime(rs.getTimestamp("time"));
        feedback.setUser_id(rs.getInt("user_id"));
        feedback.setContent(rs.getString("content"));
        return feedback;
    }
}
```

Код реализации класса DeviceRowMapper

```
      public class DeviceRowMapper implements
org.springframework.jdbc.core.RowMapper
{
    public Device mapRow(ResultSet rs, int rowNum) throws SQLException {
```

```
        Device device = new Device();
        device.setId(rs.getInt("id"));
```



```
        device.setDevice_name(rs.getString("device_name"));
        device.setObject_id(rs.getInt("object_id"));
        return device;
    }
}
```

Код реализации класса AllinfoRowMapper

```
      public class AllinfoRowMapper implements
org.springframework.jdbc.core.RowMapper
{
    public GetAll mapRow(ResultSet rs, int rowNum) throws SQLException {
        GetAll getall = new GetAll();
        getall.setUser_name(rs.getString("user_name"));
        getall.setObject_name(rs.getString("object_name"));
        getall.setDevice_name(rs.getString("device_name"));
        getall.setSensor_name(rs.getString("sensor_name"));
        getall.setTemp(rs.getInt("exampledata1"));
        getall.setHumidity(rs.getInt("exampledata2"));
        getall.setTime(rs.getTimestamp("time").toString());
        return getall;
    }
}
```

Код реализации класса User

```
      public class User
{
   int id;
   String user_name;
   String password;
   String email;
   //getter and setter methods
   public int getId() {
      return id;
   }
   public String getEmail() {
      return email;
   }
   public String getPassword() {
      return password;
   }
   public String getUser_name() {
      return user_name;
   }
   public void setEmail(String email) {
      this.email = email;
   }
   public void setId(int id) {
      this.id = id;
   }
   public void setPassword(String password) {
      this.password = password;
   }
   public void setUser_name(String user_name) {
      this.user_name = user_name;
   }
}
```

Код реализации класса Sensor

```
public class Sensor
{
    int id;
    String sensor_name;
    String time;
    int exampledata1;
    int exampledata2;
    int device_id;
    //getter and setter methods
    public int getId() {
        return id;
    }
    public String getSensor_name() {
        return sensor_name;
    }
    public String getTime() {
        return time;
    }
    public int getExampledata1() {
        return exampledata1;
    }
    public int getExampledata2(){
        return exampledata2;
    }
    public int getDevice_id(){
        return device_id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public void setSensor_name(String sensor_name) {
        this.sensor_name = sensor_name;
    }
    public void setTime(String time) {
        this.time = time;
    }
    public void setExampledata1(int exampledata1) {
        this.exampledata1 = exampledata1;
    }
    public void setExampledata2(int exampledata2) {
        this.exampledata2 = exampledata2;
    }
    public void setDevice_id(int device_id) {
        this.device_id = device_id;
    }
}
```

Код реализации класса Object

```
public class Object
{
    int id;
    String object_name;
    int user_id;
    //getter and setter methods
    public int getId() {
        return id;
    }
    public String getObject_name() {
```

```
        return object_name;
    }
    public int getUser_id() {
        return user_id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public void setObject_name(String object_name) {
        this.object_name = object_name;
    }
    public void setUser_id(int user_id) {
        this.user_id = user_id;
    }
}
```

Код реализации класса GetAll

```
  public class GetAll
{
    String user_name;
    String object_name;
    String device_name;
    String sensor_name;
    int temp;
    int humidity;
    String time;
    //getter and setter methods
    public String getUser_name() {
        return user_name;
    }
    public String getObject_name() {
        return object_name;
    }
    public String getDevice_name() {
        return device_name;
    }
    public String getSensor_name() {
        return sensor_name;
    }
    public int getTemp() {
        return temp;
    }
    public int getHumidity() {
        return humidity;
    }
    public String getTime() {
        return time;
    }
    public void setUser_name(String user_name) {
        this.user_name = user_name;
    }
    public void setObject_name(String object_name) {
        this.object_name = object_name;
    }
    public void setDevice_name(String device_name) {
        this.device_name = device_name;
    }
    public void setSensor_name(String sensor_name) {
        this.sensor_name = sensor_name;
    }
    public void setTemp(int temp) {
```

```
        this.temp = temp;
    }
    public void setHumidity(int humidity) {
        this.humidity = humidity;
    }
    public void setTime(String time) {
        this.time = time;
    }
}
```

Код реализации класса Feedback

```
  public class Feedback
{
    int id;
    String title;
    String content;
    Timestamp time;
    int user_id;
    //getter and setter methods
    public int getId() {
        return id;
    }
    public String getTittle() {
        return title;
    }
    public Timestamp getTime() {
        return time;
    }
    public int getUser_id(){ return  user_id;}
    public String getContent() {
        return content;
    }
    public void setId(int id) {
        this.id = id;
    }
    public void setTitle(String title) {
        this.title= title;
    }
    public void setTime(Timestamp time) {
        this.time = time;
    }
    public void setUser_id(int user_id) {
        this.user_id = user_id;
    }
    public void setContent(String content) {
        this.content = content;
    }
}
```

Код реализации класса Device

```
  public class Device
{
    int id;
    String device_name;
    int object_id;
    //getter and setter methods
    public int getId() {
        return id;
    }
    public String getDevice_name() {
        return device_name;
```

```
    }
    public int getObject_id() {
        return object_id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public void setDevice_name(String device_name) {
        this.device_name = device_name;
    }
    public void setObject_id(int object_id) {
        this.object_id = object_id;
    }
}
```

Код реализации класса CustomParsing

```
  public class CustomParsing {
    static Sensor parseSensors(String json){
        Sensor sensor = new Sensor();
        BasicJsonParser basicJsonParser = new BasicJsonParser();
        Map<String, Object> parseMap = basicJsonParser.parseMap(json);
        sensor.setSensor_name(parseMap.get("sensor_name").toString());

sensor.setDevice_id(Integer.parseInt(parseMap.get("device_id").toString()));

sensor.setExampledata1(Integer.parseInt(parseMap.get("exampledata1").toString())
);

sensor.setExampledata2(Integer.parseInt(parseMap.get("exampledata2").toString())
);
        return sensor;
    }
}
```

Код файла pom.xml

```
  <?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>project</groupId>
    <artifactId>arduino</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <path-spring>org.springframework</path-spring>
        <path-spring-boot>org.springframework.boot</path-spring-boot>
        <version-spring>4.2.6.RELEASE</version-spring>
        <version-spring-boot>1.3.5.RELEASE</version-spring-boot>
        <java.version>1.8</java.version>
    </properties>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.3.5.RELEASE</version>
    </parent>

    <dependencies>
        <dependency>
```

Продолжение приложения

```
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.10</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.6.2</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-websocket</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-websocket</artifactId>
            <version>4.2.5.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-messaging</artifactId>
        </dependency>
        <dependency>
            <groupId>${path-spring-boot}</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <version>${version-spring-boot}</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <!--version>6.0.2</version not compatible version with 5.1.73 MySQL
Database-->
            <version>5.1.9</version>
        </dependency>
        <dependency>
            <groupId>commons-codec</groupId>
            <artifactId>commons-codec</artifactId>
            <version>1.10</version>
        </dependency>
        <dependency>
            <groupId>${path-spring-boot}</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <version>${version-spring-boot}</version>
        </dependency>
        <dependency>
            <groupId>${path-spring}</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>${version-spring}</version>
        </dependency>
        <dependency>
            <groupId>${path-spring}</groupId>
            <artifactId>spring-websocket</artifactId>
            <version>${version-spring}</version>
        </dependency>
        <dependency>
            <groupId>${path-spring}</groupId>
            <artifactId>spring-context</artifactId>
            <version>${version-spring}</version>
        </dependency>
        <dependency>
```

Продолжение приложения

```
            <groupId>${path-spring}</groupId>
            <artifactId>spring-aop</artifactId>
            <version>${version-spring}</version>
        </dependency>
        <dependency>
            <groupId>${path-spring}</groupId>
            <artifactId>spring-beans</artifactId>
            <version>${version-spring}</version>
        </dependency>
        <dependency>
            <groupId>${path-spring}</groupId>
            <artifactId>spring-core</artifactId>
            <version>${version-spring}</version>
        </dependency>
        <dependency>
            <groupId>${path-spring}</groupId>
            <artifactId>spring-expression</artifactId>
            <version>${version-spring}</version>
        </dependency>
        <dependency>
            <groupId>${path-spring}</groupId>
            <artifactId>spring-web</artifactId>
            <version>${version-spring}</version>
        </dependency>
        <dependency>
            <groupId>${path-spring-boot}</groupId>
            <artifactId>spring-boot</artifactId>
            <version>${version-spring-boot}</version>
        </dependency>
        <dependency>
            <groupId>${path-spring-boot}</groupId>
            <artifactId>spring-boot-autoconfigure</artifactId>
            <version>${version-spring-boot}</version>
        </dependency>
        <dependency>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
            <version>1.2</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>${path-spring-boot}</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <version>${version-spring-boot}</version>
            </plugin>
        </plugins>
    </build>
    <repositories>
        <repository>
            <id>maven-repository</id>
            <url>https://repo1.maven.org/maven2/</url>
        </repository>
        <repository>
            <id>spring-releases</id>
            <url>https://repo.spring.io/libs-release</url>
        </repository>
    </repositories>
    <pluginRepositories>
        <pluginRepository>
```

Продолжение приложения

```
            <id>maven-repository</id>
            <url>https://repo1.maven.org/maven2/</url>
        </pluginRepository>
        <pluginRepository>
            <id>spring-releases</id>
            <url>https://repo.spring.io/libs-release</url>
        </pluginRepository>
    </pluginRepositories>
</project>
```

Код страницы для тестирования index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title></title>
    <script>
        onload=function(){
            document.getElementById("container").value = "tut podgruzochka";
        };
        function httpGet(theUrl)
        {
            var xmlHttp = new XMLHttpRequest();
            xmlHttp.open( "GET", theUrl, false ); // false for synchronous
request

            xmlHttp.send( null );

            //console.log(xmlHttp.response);
            document.getElementById("container").innerText = xmlHttp.response;
        }
    </script>
</head>
<body>
Тестирование экспорта данных из базы<br>
<button onclick="httpGet('http://localhost:8080/getallusers')">Показать всех
пользователей</button>
<button onclick="httpGet('/getallsensors')">Показать все сенсоры</button>
<button onclick="httpGet('/getallobjects')">Показать все объекты</button>
<button onclick="httpGet('/getallfeedbacks')">Показать все отзывы</button>
<button onclick="httpGet('/getalldevice')">Показать все устройства</button><br>
<div id="container">
    <a href=""></a>
</div><br>
Показать всю информацию о пользователе с именем:<br>
<form action="/getallbyuser_name">
    <p><input name="user_name"></p>
    <p><input type="submit"></p>
</form><br>
Добавить пользователя:<br>
<form action="/insertuser">
    <p><input name="user_name">Имя пользователя</p>
    <p><input name="password">Пароль</p>
    <p><input name="email">e-mail</p>
    <p><input type="submit"></p>
</form><br>
Добавить объект пользователю:<br>
<form action="/insertobject">
    <p><input name="object_name">Название объекта</p>
    <p><input name="user_id">id пользователя</p>
    <p><input type="submit"></p>
```

Продолжение приложения

```
</form><br>
Добавить девайс на объект:<br>
<form action="/insertdevice">
    <p><input name="device_name">Название девайса</p>
    <p><input name="object_id">id объекта</p>
    <p><input type="submit"></p>
</form><br>
</form><br>
Добавить данные сенсора:<br>
<form action="/insertsensor">
    <p><input name="sensor_name">Название сенсора</p>
    <p><input name="device_id">id девайса</p>
    <p><input name="exampledata1">Пример данных 1</p>
    <p><input name="exampledata2">Пример данных 2</p>
    <p><input type="submit"></p>
</form><br>
</body>
</html>
```