МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Образовательный портал «GeekBrains»

Факультет дистанционный

Кафедра разработки ПО

Специальность: разработчик

Специализация: веб-разработка на Java

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ДИПЛОМНОМУ**

**ПРОЕКТУ НА ТЕМУ:**

**«РАЗРАБОТКА МИКРОСЕРВИСНОГО ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ УЧЕТА И ОТОБРАЖЕНИЯ ПОКАЗАНИЙ ПРИБОРОВ УЧЕТА ТЕПЛА»**

Дипломник Карпенко А.О.

(Ф.И.О.)

Руководитель проекта:

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой:

(учен. степень, звание, подпись, Ф.И.О.)

Консультанты:

(учен. степень, звание, должность, подпись, Ф. И.О.)

Консультанты:

(учен. степень, звание, должность, подпись, Ф. И.О.)

Нормоконтролеры:

(учен. степень, звание, должность, подпись, Ф.И.О.)

Дипломный проект защищен с оценкой

Председатель ГЭК

(учен. степень, звание, подпись, Ф.И.О.)

Москва 2024

Реферат

GeekBrains

2024

У

Изм.

Лист

№ докум.

Подпись

Дата

Лист

1

ДП 00.00.ПЗ.

Реферат

Лит.

Листов

1

Разраб.

Карпенко А.О

Провер.

.

Утверд.

Н.контр.

.

Дипломный проект содержит исходный код модулей программы, 42 страницы пояснительной записки, в том числе 1 таблицу, 10 источников литературы.

ВЕБ-ПРИЛОЖЕНИЕ, БЕЗОПАСНОСТЬ, МИКРОСЕРВИС, ЗАВИСИМОСТИ, УЧЕТ ТЕПЛОВОЙ ЭНЕРГИИ, РАЗРАБОТКА, ИНТЕРФЕЙСЫ ПЕРЕДАЧИ

Целью дипломного проекта является разработка веб-приложения на микросервисной архитектуре по регистрации и авторизации пользователей, регистрации устройств учета, хранения и предоставления данных с этих приборов и автоматизации создания отчетов.

В рамках дипломного проекта произведен анализ работы приборов учета тепла, их технических характеристик, возможностей считывания данных.

В рамках исследовательского раздела произведены выбор стека используемых технологий, разработка проекта архитектуры приложения, обсуждение и проектирование его модулей.

В практической части реализован модуль регистрации и авторизации пользователей, организована защита конфиденциальных данных, начата разработка основного модуля для принятия, хранения и предоставления данных с приборов учета в привязке к конкретному авторизованному пользователю, проведено обсуждение модуля автоматизации предоставления отчетов.

Das Referat

GeekBrains

2024

У

Изм.

Лист

№ докум.

Подпись

Дата

Лист

1

ДП 00.00.ПЗ.

Das Referat

Лит.

Листов

1

Разраб.

Карпенко А.О

Провер.

Утверд.

Н.контр.

Das Diplomprojekt enthält den Quellcode der Programmmodule, 42 Seiten erklärenden Zettels, einschließlich 1 Tabelle, 10 Quellen der Literaturquellen.

WEBANWENDUNG, SICHERHEIT, MICROSERVICE, ABHÄNGIGKEITEN, RECHNUNGSLEGUNGEN ZUR THERMISCHEN ENERGIE, ENTWICKLUNG, ÜBERTRAGUNGSSCHNITTSTELLEN

Ziel der Diplomarbeit ist die Entwicklung einer Webanwendung auf einer Microservice-Architektur zur Registrierung und Autorisierung von Benutzern, zur Registrierung von Messgeräten, zur Speicherung und Bereitstellung von Daten dieser Geräte sowie zur Automatisierung der Erstellung von Berichten.

Im Rahmen der Diplomarbeit wurde eine Analyse der Funktionsweise von Wärmemessgeräten, ihrer technischen Eigenschaften und Datenauslesefähigkeiten durchgeführt.

Im Rahmen des Forschungsteils wurde der Stack der verwendeten Technologien ausgewählt, ein Entwurf einer Anwendungsarchitektur entwickelt und deren Module diskutiert und entworfen.

Im praktischen Teil wurde ein Modul zur Registrierung und Autorisierung von Benutzern implementiert, der Schutz vertraulicher Daten organisiert und das Hauptmodul zur Annahme, Speicherung und Bereitstellung von Daten von Messgeräten in Bezug auf einen bestimmten autorisierten Benutzer entwickelt begonnen und eine Diskussion über das Modul zur Automatisierung der Berichtsbereitstellung geführt.

Содержание

GeekBrains

2024

У

Изм.

Лист

№ докум.

Подпись

Дата

Лист

1

ДП 00.00.ПЗ.

Содержание

Лит.

Листов

2

Разраб.

Карпенко А.О

Провер.

Утверд.

Н.контр.

.

[Введение 6](#_Toc162288239)

[1 Spring 9](#_Toc162288240)

[1.1 Цели и задачи Spring 9](#_Toc162288241)

[1.2 Инъекция зависимостей 9](#_Toc162288242)

[1.3 Аспектно - ориентированное программирование 9](#_Toc162288243)

[1.4 Функции Spring 10](#_Toc162288244)

[1.5 Архитектура Spring 10](#_Toc162288245)

[1.6 Контейнер компонентов 10](#_Toc162288246)

[1.7 Контейнеры 11](#_Toc162288247)

[2 Исследование предметной области 12](#_Toc162288248)

[2.1 Теплосчетчик и счетчик воды СКМ-2 12](#_Toc162288249)

[2.1.1 Измерение и учет времени работы 13](#_Toc162288250)

[2.1.2 Индикация, регистрация и хранение параметров 14](#_Toc162288251)

[2.1.3 Выходные интерфейсы 14](#_Toc162288252)

[2.1.4 Устройство и работа 15](#_Toc162288253)

[2.1.5 Передача данных 15](#_Toc162288254)

[2.1.6 Обоснование необходимости разработки приложения 16](#_Toc162288255)

[3 Разработка вэб – приложения 18](#_Toc162288256)

[3.1 Выбор среды разработки 18](#_Toc162288257)

[3.2 Автоматическая сборка проектов 19](#_Toc162288258)

[3.2.1 Описание зависимостей 20](#_Toc162288259)

[3.3 Модуль регистрации и авторизации клиентов 21](#_Toc162288260)

[3.3.1 Основные сущности модуля 22](#_Toc162288261)

[3.3.2 Хранение данных 25](#_Toc162288262)

[3.3.3 Сервис 25](#_Toc162288263)

[3.3.4 Контроллеры 27](#_Toc162288264)

[3.3.5 Настройки безопасности 28](#_Toc162288265)

[3.3.6 Веб-приложение для работы с модулем 29](#_Toc162288266)

[3.3.7 Общие параметры запуска 33](#_Toc162288267)

[3.3 Модуль сбора и регистрации данных с приборов 34](#_Toc162288268)

[3.3.1 Основные сущности модуля 34](#_Toc162288269)

[3.3.2 Хранение данных 35](#_Toc162288270)

[3.3.3 Сервис 36](#_Toc162288271)

[3.3.4 Контроллеры 37](#_Toc162288272)

[3.3.5 Настройки безопасности 38](#_Toc162288273)

[3.3.6 Веб-приложение для работы с модулем 39](#_Toc162288274)

[3.3.7 Общие параметры запуска 39](#_Toc162288275)

[Заключение 41](#_Toc162288276)

[Список использованных источников 42](#_Toc162288277)

Введение

GeekBrains

2024

У

Изм.

Лист

№ докум.

Подпись

Дата

Лист

1

ДП 00.00.ПЗ.

Введение

Лит.

Листов

3

Разраб.

Карпенко А.О

Провер.

.

Утверд.

Н.контр.

.

В последнее время становятся актуальными вэб - приложения, приходя на смену обычным приложениям. Причин для этого может быть много: санкции, требования. Вэб-приложение же изначально является кроссплаформенным решением и не зависит от операционной системы устройства. Оно работает, как вэб-сайт, поэтому при разработке необходимо учитывать, чтобы приложение одинаково работало с разными браузерами (Google Chrome, Mozilla Firefox, Microsoft Edge, Tor Browser, Opera, Safari) и было адаптивным (могло подстраиваться к разным размерам экрана).

Цель проекта – это разработать бэкенд вэб – приложения для учета показаний и формирования отчетов по показаниям приборов учета тепла. Планируется написание микросервиса для работы с базой пользователей, авторизацией и аутентификацией и микросервиса сбора и предоставления суточных и часовых данных приборов учета в привязке к пользователю и разграничению доступа к данным.

Написание вэб – приложения будет происходить в среде разработки IntelliJ IDEA на языке программирования Java, используя фреймворк Spring Framework, базу данных MySQL или PostgreSQL.

Планируется покрытие тестами всех доступных «ручек» API.

«Java — строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Разработка ведётся сообществом, организованным через Java Community Process;

язык и основные реализующие его технологии распространяются по лицензии GPL. Права на торговую марку принадлежат корпорации Oracle.

Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, для которой существует реализация виртуальной Java-машины. Дата официального выпуска — 23 мая 1995 года. Занимает высокие места в рейтингах популярности языков программирования (2-е место в рейтингах IEEE Spectrum (2020) и TIOBE (2021)).» © [Википедия](https://ru.wikipedia.org/wiki/Java)

«**Spring Framework** (или коротко Spring) — универсальный фреймворк с открытым исходным кодом для Java-платформы. Также существует форк для платформы .NET Framework, названный Spring.NET.

Spring обеспечивает решения многих задач, с которыми сталкиваются Java-разработчики и организации, которые хотят создать информационную систему, основанную на платформе Java. Из-за широкой функциональности трудно определить наиболее значимые структурные элементы, из которых он состоит. Spring не всецело связан с платформой Java Enterprise, несмотря на свою масштабную интеграцию с ней, что является важной причиной его популярности.

Spring может быть рассмотрен как коллекция меньших фреймворков или фреймворков во фреймворке. Большинство этих фреймворков может работать независимо друг от друга, однако они обеспечивают большую функциональность при совместном их использовании.» © [Википедия](https://ru.wikipedia.org/wiki/Spring_Framework)

«**MySQL** — свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.» © [Википедия](https://ru.wikipedia.org/wiki/MySQL)

В дипломной работе я буду выступать в качестве разработчика.

1 Spring

GeekBrains

2024

У

Изм.

Лист

№ докум.

Подпись

Дата

Лист

1

ДП 01.00.ПЗ.

Фрэймворк Spring

Лит.

Листов

3

Разраб.

Карпенко А.О

Провер.

Утверд.

Н.контр.

1.1 Цели и задачи Spring

Любое нетривиальное приложение состоит из множества компонентов, каждый из которых несет ответственность за свою часть общей функциональности приложения, взаимодействуя с другими элементами.

При запуске приложения необходимо выполнять интеграцию компонентов.

Spring предлагает контейнер – контекст приложения Spring – который создаёт и управляет компонентами приложения.

Компоненты (или beans) соединены вместе в контексте приложения Spring, чтобы создать полноценное приложение.

1.2 Инъекция зависимостей

Архитектурный шаблон, подразумевающий подход, при котором вместо того, чтобы компоненты создавали и поддерживали жизненный цикл других компонентов, от которых они зависят, приложение, зависящее от зависимостей, полагается на отдельный объект (контейнер) для создания и поддержки всех компонентов и внедрения их в нужные им компоненты.

Реализуется за счёт: параметризированных конструкторов и публичных свойств.

1.3 Аспектно - ориентированное программирование

АОП – аспектно-ориентированное программирование – архитектурный принцип, способствующий разделению задач в программной системе. Системы состоят из компонентов, каждый из которых имеет высокое зацепление.

1.4 Функции Spring

Spring Framework предоставляет архитектурную структуру приложения.

Бизнес – логика инкапсулируется в набор бизнес – объектов – сервисы.

Сервисы выполняют операции, общие для всех пользовательских интерфейсов.

Приложение содержит разный набор контроллеров и представлений для каждого пользовательского интерфейса, и они используют общие бизнес - объекты для выполнения критических операций.

Контроллеры необходимы только для выполнения операций с конкретным интерфейсом пользователя:

- HTML Views

- JPS

- JSON

и т.д.

1.5 Архитектура Spring

Spring Framework состоит из модулей, которые разделены на категории.

Модули предоставляют все необходимое для разработки корпоративных приложений.

Включаемые модули (Spring Initializer):

Core

Spring MVC

JDBC

и т.д.

1.6 Контейнер компонентов

В приложении на основе Spring Framework объекты приложения содержатся в контейнере Spring.

Контейнер будет:

создавать объекты

связывать их вместе

настраивать и управлять их жизненным циклом

1.7 Контейнеры

Spring Framework состоит из контейнеров, которые можно разделить на два типа:

Фабрики компонентов (интерфейс org.springframework.beans.factory. BeanFactory) являются простейшими контейнерами, предоставляющими базовую поддержку для внедрения зависимостей.

Контекст приложений (интерфейс org.springframework.context. ApplicationContext) основан на понятии фабрики компонентов, реализует прикладные службы фреймворка, такие как:

возможность приема текстовых сообщений из файлов свойств

возможность публиковать события приложений для заинтересованных слушателей

2 Исследование предметной области

GeekBrains

2024

У

Изм.

Лист

№ докум.

Подпись

Дата

Лист

1

ДП 02.00.ПЗ.

Исследование предметной

области

Лит.

Листов

6

Разраб.

Карпенко А.О

Провер.

Утверд.

Н.контр.

2.1 Теплосчетчик и счетчик воды СКМ-2

****

Теплосчетчики и счетчики воды СКМ-2, производитель - ООО "ВОГЕЗЭНЕРГО», г. Минск, Республика Беларусь, зареги­стрированы в государственном реестре средств измерений Республики Беларусь № РБ 03 10 4364 21 и государственном реестре средств измерений Российской Федерации № 76793-19.

Счет­чики соответствуют требованиям ГОСТ EN 1434 – 2018, ГОСТ Р 51649, ТУ BY 101138220.007-2010. Применяемые в счетчиках расходомеры электромагнитные ВИРС-М и ультразвуковые ВИРС-У соответствуют ГОСТ ISO 4064-2017 и ГОСТ EN 1434 – 2018.

Теплосчетчики и счетчики воды СКМ-2 (далее – счетчики), предназначены для измерения и коммерческого учета:

– тепловой энергии и теплоносителя в системах жидкостного теплоснабжения, холодоснабжения, кондиционирования, параметров теплоносителя;

– объемного и массового расхода холодной и горячей, в том числе питьевой воды, различных теплоносителей, технической воды, сточных вод.

Область применения счетчиков:

– теплоисточники;

– индивидуальные (ИТП) и централизованные (ЦТП) тепловые пункты объектов теплопотребления, зданий;

– системы охлаждения, кондиционирования (чиллеры);

– узлы технического и коммерческого учета воды, растворов, органических и неорганических жидкостей в производственных технологических линиях.

Счетчики могут вычислять количество тепловой энергии и теплоносителя одновременно в двух независимых системах теплоснабжения – СИСТЕМА 1 и СИСТЕМА 2. Счетчики имеют несколько исполнений.

Счетчики СКМ-2 являются составными и состоят из вычислителя (ВБ), преобразователей расхода (расходомеров), термопреобразователей сопротивления, преобразователей давления.

2.1.1 Измерение и учет времени работы

ВБ учитывает время:

- работы при включенном питании;

- простоя (напряжение питания отсутствовало);

- работы в штатном режиме СИСТЕМЫ 1 и СИСТЕМЫ 2;

- работы в нештатных ситуациях СИСТЕМЫ 1 и СИСТЕМЫ 2:

Цена младшего разряда 0,01ч, погрешность измерения ±0,01%

ВБ ведет календарь и учитывает текущее время в течение 5 лет при отсутствии питания от сети

2.1.2 Индикация, регистрация и хранение параметров

ВБ индицирует:

− текущие и итоговые значения измеренных и накопленных параметров для СИСТЕМЫ 1 и СИСТЕМЫ 2;

− архивные данные;

− установленные настроечные параметры ВБ.

ВБ хранит в памяти архивные и итоговые данные, формирует часовые, суточные и месячные отчеты:

– итоговых значений;

– абсолютных и накопленных значений за часы, сутки, месяцы;

– усредненных значений величин за часы, сутки, месяцы;

– нештатных ситуаций за часы, сутки и месяцы.

Объем архива:

– 2048 записей - для хранения среднечасовых значений (одна запись в час);

– 1792 записей - для хранения среднесуточных значений (одна запись в сутки).

2.1.3 Выходные интерфейсы

Имеющиеся в вычислителе выходные интерфейсы предназначенные для передачи информации, перечислены в таблице 1.

Стандартно устанавливаемые интерфейсы - несменяемы.

Опционально устанавливаемые интерфейсы могут быть выбраны и заказаны в соответствии с таблицей 1. Замена их может быть выполнена пользователем, переустановкой модулей в разъемах платы контроллера ВБ. По умолчанию устанавливается модуль RS-485.

Порт USB организован по технологии прямого доступа к памяти и не имеет отношения к сменным модулям интерфейсов.

Доступные протоколы обмена данными – М-Bus, МodBus RTU, ModBus TCP.

Максимальная скорость передачи данных по интерфейсу RS-232 и RS-485 – 115200 кбит/с.

Оптический ИК порт в исходном состоянии закрыт. Открытие порта производится по инструкции. При перерыве в передаче данных более 2 минут порт отключается автоматически.

Таблица 1

|  |  |
| --- | --- |
| Интерфейс | Назначение, выполняемые функции |
| RS-485  (стандартно) | Передача информации на СУ имеющие интерфейс RS-485. Длина линии связи не более 12 м. |
| USB host  (стандартно) | Передача информации на USB Flash накопитель. |
| ИК оптопорт  (стандартно) | Бесконтактная передача информации через оптические устройства считывания (УСО-1, УСО-2). |
| M-bus  (опционально) | Передача информации на СУ имеющие интерфейс М-Bus. Длина линии связи не более 1000 м. |
| RS-232,RS-485  (опционально) | Передача информации на СУ имеющие интерфейс RS-232, RS-485. Длина линии связи не более 1000 м. |
| Ethernet  (опционально) | Передача информации на СУ имеющие интерфейс Ethernet. |
| GSM/GPRS  (опционально) | Передача информации по каналам GSM/GPRS. |

2.1.4 Устройство и работа

Принцип работы счетчика CKM-2 основан на принятии вычислителем сигналов от расходомеров, преобразователей давления и температуры, обработке их, вычислении расхода, количества теплоносителя, тепловой энергии и других параметров.

Объем теплоносителя вычисляется как произведение количества импульсов, полученных от расходомеров, на весовой коэффициент импульса. Тепловая энергия вычисляется в соответствии с формулами. Исполнение счетчика, алгоритм контроля за нештатными ситуациями выбирается при выпуске из производства или на месте установки пользователем.

2.1.5 Передача данных

Передача данных с ВБ на компьютер возможна через интерфейсы (п.2.1.3), радиомодем, модем GSM, Ethernet, адаптер переноса данных ДК-4. С модемами GSM ВБ может работать:

– в режиме CSD (передача данных по открытому каналу);

– в режиме GPRS (пакетная передача данных).

Режим работы выбирается при настройке модема в соответствии с его документацией.

Скорость передачи и паритет устанавливаются одинаковыми для счетчика и считывающего устройства.

При передаче данных с использованием ИК порта порт должен быть включен.

Порядок передачи данных с помощью адаптера ДК-4 описан в руководстве по эксплуатации на ДК-4.

2.1.6 Обоснование необходимости разработки приложения

Имеются приборы учета тепловой энергии «СКМ-2». Для считывания доступны: конфигурация работы прибора и его настроечные параметры, а также архив суточных и часовых данных.

На сегодняшний день приборы отпускаются покупателю и предлагается программа для установки на персональный компьютер для считывания и отображения данных с прибора учета. Показания переносят пользователи в свои программы и формируют на базе этих данных ежемесячные отчеты. Клиентами, использующими приборы в подавляющем большинстве являются ЖКХ и ЖРЭУ, на балансе которых находятся десятки общедомовых приборов учета тепла. Также в последнее время все больше строится многоквартирных домов с установкой индивидуальных приборов учета тепла в каждой квартире, что еще больше увеличивает количество приборов, работы по снятию показаний и составлению сводных отчетов.

Возникла потребность в написании программы в виде веб-сервисов для ведения базы пользователей, их приборов, а также данных с этих приборов и сервиса автоматизированного периодического составления отчетов.

Приложение пока планируется разбить на 3 модуля:

* модуль регистрации и авторизации клиентов;
* модуль сбора и регистрации данных с приборов;
* модуль предоставления отчетов по расписанию.

3 Разработка вэб – приложения

GeekBrains

2024

У

Изм.

Лист

№ докум.

Подпись

Дата

Лист

1

ДП 03.00.ПЗ.

Разработка приложения

Лит.

Листов

23

Разраб.

Карпенко А.О

Провер.

Утверд.

Н.контр.

3.1 Выбор среды разработки

«Интегрированная среда разработки, ИСP (англ. integrated development environment — IDE), также единая среда разработки, ЕСР — комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО).

Среда разработки включает в себя:

- текстовый редактор,

- транслятор (компилятор и/или интерпретатор),

- средства автоматизации сборки

Иногда содержит также средства для интеграции с системами управления версиями и разнообразные инструменты для упрощения конструирования графического интерфейса пользователя. Многие современные среды разработки также включают браузер классов, инспектор объектов и диаграмму иерархии классов для использования при объектно-ориентированной разработке ПО. ИСР обычно предназначены для нескольких языков программирования — такие как IntelliJ IDEA, NetBeans, Eclipse, Qt Creator, Geany, Embarcadero RAD Studio, Code::Blocks, Xcode, Dev-C++ или Microsoft Visual Studio, но есть и IDE для одного определённого языка программирования — как, например, Visual Basic или Delphi.» © [Википедия](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8)

Мой выбор пал на IntelliJ IDEA, потому что эта среда разработки мне понравилась больше, чем Visual Studio Code. А программировал на языке Java я только на этих двух IDE.

3.2 Автоматическая сборка проектов

В проекте будет использоваться система автоматической сборки (также известная как сборочный сервер) Maven. И стоит начать с добавления в файл pom.xml зависимостей, благодаря которым сборщик скачает и установит в наш проект необходимые библиотеки:

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-security</**artifactId**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-data-jpa</**artifactId**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.postgresql</**groupId**>  
 <**artifactId**>postgresql</**artifactId**>  
 <**scope**>runtime</**scope**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.projectlombok</**groupId**>  
 <**artifactId**>lombok</**artifactId**>  
 <**version**>1.18.30</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.springframework.hateoas</**groupId**>  
 <**artifactId**>spring-hateoas</**artifactId**>  
 <**version**>0.24.0.RELEASE</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>javax.servlet</**groupId**>  
 <**artifactId**>jstl</**artifactId**>  
 <**version**>1.2</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.springframework.security</**groupId**>  
 <**artifactId**>spring-security-taglibs</**artifactId**>  
 <**version**>5.2.0.RELEASE</**version**>  
</**dependency**>

3.2.1 Описание зависимостей

**Spring Boot** (spring-boot-starter-web) это средство с открытым исходным кодом, которое упрощает использование платформ на основе Java для создания микрослужб и веб-приложений. Обеспечивает возможность создавать автономные приложения Spring, которые можно запускать сразу и которым не нужны заметки, XML-файл конфигурации или написание множества дополнительных строк кода. [©](https://azure.microsoft.com/ru-ru/resources/cloud-computing-dictionary/what-is-java-spring-boot)

**Spring Security** (spring-boot-starter-security) это фреймворк, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для корпоративных приложений, созданных с помощью Spring Framework. [©](https://habr.com/ru/articles/203318/)

**Java Persistence API (**spring-boot-starter-data-jpa) – это стандартная технология, позволяющая "отображать" объекты на реляционные базы данных. POM-модель spring-boot-starter-data-jpa позволяет быстро начать работу. Она обеспечивает следующие ключевые зависимости:

Hibernate одна из самых популярных реализаций JPA.

Spring Data JPA: помогает реализовать репозитории на основе JPA.

Spring ORM: основное средство поддержки ORM из Spring Framework. [©](https://habr.com/ru/articles/203318/)

**PostgreSQL** (postgresql) – это драйвер подключения к используемой нами базе данных.

**Lombok** (lombok) – это библиотека для сокращения кода в классах и расширения функциональности языка Java. Подключается к среде разработки (IDE) или инструменту сборки приложений [Maven](https://blog.skillfactory.ru/glossary/maven/), [Gradle](https://blog.skillfactory.ru/glossary/gradle/) в качестве плагина.

Библиотека получила название в честь индонезийского острова Ломбок, расположенного недалеко от острова Ява. В переводе с индонезийского Lombok означает «перец чили»: по аналогии с приправой, библиотека призвана повысить качество Java-кода.

Принцип работы библиотеки основан на использовании аннотаций, которые:

* многократно сокращают шаблонный код;
* экономят время разработчиков;
* улучшают читаемость кода.

**Spring HATEOAS** (spring-hateoas) - это модуль Spring, который предоставляет функциональность для создания гипермедиа-совместимых RESTful веб-сервисов. Он позволяет создавать RESTful сервисы, которые включают в себя гиперссылки для доступа к связанным ресурсам. Это позволяет клиентам более гибко интерактивно использовать API, избежав жесткой зависимости от схемы сервера и обеспечивает более гибкую навигацию по данным. В общем, Spring HATEOAS помогает создавать более интеллектуальные и динамические RESTful веб-сервисы.

Spring HATEOAS используется для создания RESTful веб-сервисов, которые следуют принципу HATEOAS (Hypermedia as the Engine of Application State). Этот принцип означает, что веб-сервис должен включать в ответ гиперссылки, которые позволяют клиенту обнаружить и использовать доступные ресурсы и операции. Это помогает упростить разработку и управление клиентского кода, поскольку клиент может автоматически обнаруживать и использовать доступные ресурсы, вместо того чтобы знать о них заранее.

**JSP Standard Tag Library** (jstl) - специальная библиотека.

JSP по умолчанию позволяет встраивать код на java в разметку html. Однако иногда использование стандартных способов для ряда операций, например, для вывода на страницу элементов из списка и т.д., может быть несколько громоздким. Чтобы упростить встраивание кода java в JSP была разработана специальная библиотека - JSTL. JSTL (JSP Standard Tag Library) предоставляет теги для базовых задач JSP (цикл, условные выражения и т.д.)

Эта библиотека не является частью инфраструктуры Java EE, поэтому ее необходимо добавлять в проект самостоятельно.

**Библиотека тегов безопасности JSP** (spring-security-taglibs) - предоставляет набор тегов, которые можно использовать для управления доступом к ресурсам в веб-приложении.

3.3 Модуль регистрации и авторизации клиентов

Были изучены различные модели регистрации и авторизации клиентов.

Реализовать удалось простую авторизацию на базе Spring Security.

3.3.1 Основные сущности модуля

Модуль представлен двумя сущностями: User и Role

Класс User реализует класс UserDetails, который предоставляет необходимую информацию для построения объекта Authentication из DAO объектов приложения или других источников данных системы безопасности. Объект UserDetailsсодержит имя пользователя, пароль, флаги: isAccountNonExpired, isAccountNonLocked, isCredentialsNonExpired, isEnabled и Collection — прав (ролей) пользователя.

@Entity  
@Table(name = **"t\_user"**)  
**public class** User **implements** UserDetails {  
 @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 **private** Long **id**;  
 @Size(min=5, message = **"Не меньше 5 знаков"**)  
 **private** String **username**;  
 @Size(min=5, message = **"Не меньше 5 знаков"**)  
 **private** String **password**;  
 @Transient  
 **private** String **passwordConfirm**;  
 @ManyToMany(fetch = FetchType.***EAGER***)  
 **private** Set<Role> **roles**;  
  
 **public** User(){  
 }  
 **public** Long getId() {  
 **return id**;  
 }  
 **public void** setId(Long id) {  
 **this**.**id** = id;  
 }  
  
 @Override  
 **public** String getUsername() {  
 **return username**;  
 }  
 @Override  
 **public boolean** isAccountNonExpired() {  
 **return true**;  
 }

@Override  
 **public boolean** isAccountNonLocked() {  
 **return true**;  
 }  
 @Override  
 **public boolean** isCredentialsNonExpired() {  
 **return true**;  
 }  
 @Override  
 **public boolean** isEnabled() {  
 **return true**;  
 }  
 **public void** setUsername(String username) {  
 **this**.**username** = username;  
 }  
 @Override  
 **public** Collection<? **extends** GrantedAuthority> getAuthorities() {  
 **return** getRoles();  
 }  
 @Override  
 **public** String getPassword() {  
 **return password**;  
 }  
 **public void** setPassword(String password) {  
 **this**.**password** = password;  
 }  
 **public** String getPasswordConfirm() {  
 **return passwordConfirm**;  
 }  
 **public void** setPasswordConfirm(String passwordConfirm) {  
 **this**.**passwordConfirm** = passwordConfirm;  
 }  
 **public** Set<Role> getRoles() {  
 **return roles**;  
 }  
 **public void** setRoles(Set<Role> roles) {  
 **this**.**roles** = roles;  
 }  
}

Класс User реализует класс GrantedAuthority, который отражает разрешения, выданные пользователю в масштабе всего приложения, такие разрешения (как правило называются «роли»), например ROLE\_ANONYMOUS, ROLE\_USER, ROLE\_ADMIN. В нашем случае роли с пользователями связаны отношением многие-ко-многим.

@Entity  
@Table(name = **"t\_role"**)  
**public class** Role **implements** GrantedAuthority {  
 @Id  
 **private** Long **id**;  
 **private** String **name**;  
 @Transient  
 @ManyToMany(mappedBy = **"roles"**)  
 **private** Set<User> **users**;  
 **public** Role() {  
 }  
  
 **public** Role(Long id) {  
 **this**.**id** = id;  
 }  
  
 **public** Role(Long id, String name) {  
 **this**.**id** = id;  
 **this**.**name** = name;  
 }  
  
 **public** Long getId() {  
 **return id**;  
 }  
  
 **public void** setId(Long id) {  
 **this**.**id** = id;  
 }  
  
 **public** String getName() {  
 **return name**;  
 }  
  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
  
 **public** Set<User> getUsers() {  
 **return users**;  
 }  
  
 **public void** setUsers(Set<User> users) {  
 **this**.**users** = users;  
 }  
  
 @Override  
 **public** String getAuthority() {  
 **return** getName();  
 }  
}

3.3.2 Хранение данных

Хранение данных организовано в БД PostgreSQL. Используем интерфейсы, расширяющие JPAReposytory, который в свою очередь наследуется от CrudRepository.

CrudRepository является базовым интерфейсом в Spring Data, который предоставляет базовые методы CRUD.

JpaRepository, с другой стороны, предоставляет дополнительные функции, специфичные для JPA, такие как flush(), deleteInBatch(), deleteAllInBatch(), и т.д.

При этом в UserRepository мы описываем метод, возвращающий пользователя по его имени.

**public interface** RoleRepository **extends** JpaRepository<Role, Long> {}

**public interface** UserRepository **extends** JpaRepository<User, Long> {  
 User findByUsername(String username);  
}

3.3.3 Сервис

Сервисный слой представлен классом UserService, реализующим интерфейс UserDetailService, который используется, чтобы создать UserDetails объект путем реализации единственного метода этого интерфейса.

@Service  
**public class** UserService **implements** UserDetailsService {  
 @PersistenceContext  
 **private** EntityManager **em**;  
 @Autowired  
 UserRepository **userRepository**;  
 @Autowired  
 RoleRepository **roleRepository**;  
 @Autowired  
 BCryptPasswordEncoder **bCryptPasswordEncoder**;

@Override  
 **public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {  
 User user = **userRepository**.findByUsername(username);  
  
 **if** (user == **null**) {  
 **throw new** UsernameNotFoundException(**"User not found"**);  
 }  
 **return** user;  
 }  
 **public** User findUserById(Long userId) {  
 Optional<User> userFromDb = **userRepository**.findById(userId);  
 **return** userFromDb.orElse(**new** User());  
 }  
 **public** User findUserByUsername(String username) {  
 **return userRepository**.findByUsername(username);  
 }  
 **public** List<User> allUsers() {  
 **return userRepository**.findAll();  
 }  
 **public boolean** saveUser(User user) {  
 User userFromDB = **userRepository**.findByUsername(user.getUsername());  
 **if** (userFromDB != **null**) {  
 **return false**;  
 }  
 **if** (**userRepository**.count() == 0){  
 user.setRoles(**new** HashSet<Role>(Arrays.*asList*(  
 **new** Role(1L, **"ROLE\_ADMIN"**),  
 **new** Role(2L, **"ROLE\_USER"**))));  
 }  
 **else** {  
 user.setRoles(Collections.*singleton*(**new** Role(2L, **"ROLE\_USER"**)));  
 }  
 **roleRepository**.saveAll(user.getRoles());  
 user.setPassword(**bCryptPasswordEncoder**.encode(user.getPassword()));  
 **userRepository**.save(user);  
 **return true**;  
 }  
  
 **public boolean** deleteUser(Long userId) {  
 **if** (**userRepository**.findById(userId).isPresent()  
 && !**userRepository**.findById(userId).get().getRoles()

.stream().anyMatch(role -> role.getName().equals(**"ROLE\_ADMIN"**))) {  
 **userRepository**.deleteById(userId);  
 **return true**;  
 }  
 **return false**;  
 }  
  
 **public** List<User> usergtList(Long idMin) {  
 **return em**.createQuery(**"SELECT u FROM User u WHERE u.id > :paramId"**, User.**class**)  
 .setParameter(**"paramId"**, idMin).getResultList();  
 }  
}

3.3.4 Контроллеры

Контроллеры представлены двумя контроллерами RegistrationController и AdminController.

RegistrationController отвечает за регистрацию нового пользователя в системе. При этом первый зарегистрированный пользователь получает роль ADMIN и может удалять других пользователей. Себя при этом он удалить не может. Все последующие пользователи получают роль USER и имеют более ограниченные права. Планируется в дальнейшем расширить используемые роли и больше разделить права доступа.

@Controller  
**public class** RegistrationController {  
  
 @Autowired  
 **private** UserService **userService**;  
  
 @GetMapping(**"/registration"**)  
 **public** String registration(Model model) {  
 model.addAttribute(**"userForm"**, **new** User());  
  
 **return "registration"**;  
 }  
  
 @PostMapping(**"/registration"**)  
 **public** String addUser(@ModelAttribute(**"userForm"**) @Valid User userForm,

BindingResult bindingResult, Model model) {  
  
 **if** (bindingResult.hasErrors()) {  
 **return "registration"**;  
 }  
 **if** (!userForm.getPassword().equals(userForm.getPasswordConfirm())){  
 model.addAttribute(**"passwordError"**, **"Пароли не совпадают"**);  
 **return "registration"**;  
 }  
 **if** (!**userService**.saveUser(userForm)){  
 model.addAttribute(**"usernameError"**, **"Пользователь с таким именем уже существует"**);  
 **return "registration"**;  
 }  
  
 **return "redirect:/"**;  
 }  
}

AdminController отвечает за управление учетными данными пользователя в системе. Доступ к ветке разрешен пользователю с правами ADMIN. Администратор может удалить любого пользователя, кроме себя.

@Controller  
**public class** AdminController {  
 @Autowired  
 **private** UserService **userService**;  
  
 @GetMapping(**"/admin"**)  
 **public** String userList(Model model) {  
 model.addAttribute(**"allUsers"**, **userService**.allUsers());  
 **return "admin"**;  
 }  
  
 @PostMapping(**"/admin"**)  
 **public** String deleteUser(@RequestParam(required = **true**, defaultValue = **""** ) Long userId, @RequestParam(required = **true**, defaultValue = **""** ) String action, Model model) {  
 **if** (action.equals(**"delete"**)){  
 **userService**.deleteUser(userId);  
 }  
 **return "redirect:/admin"**;  
 }  
  
 @GetMapping(**"/admin/gt/{userId}"**)  
 **public** String gtUser(@PathVariable(**"userId"**) Long userId, Model model) {  
 model.addAttribute(**"allUsers"**, **userService**.usergtList(userId));  
 **return "admin"**;  
 }  
}

3.3.5 Настройки безопасности

Настройки безопасности пока представлены классом WebSecurityConfig расширающим WebSecurityConfigurerAdapter. Понимаю, что такой подход сейчас устарел и требует замены на современные, но в попытках написать что-то актуальное и стоящее провел много времени и пока это единственный вариант, который удалось довести до корректного запуска и работы.

@Configuration  
@EnableWebSecurity  
**public class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {  
 @Autowired  
 UserService **userService**;  
  
 @Bean  
 **public** BCryptPasswordEncoder bCryptPasswordEncoder() {  
 **return new** BCryptPasswordEncoder();  
 }  
@Override  
 **protected void** configure(HttpSecurity httpSecurity) **throws** Exception {  
 httpSecurity  
 .csrf()  
 .disable()  
 .authorizeRequests()  
 *//Доступ только для не зарегистрированных пользователей  
 //.antMatchers("/registration").not().fullyAuthenticated()  
 //Доступ только для пользователей с ролью Администратор* .antMatchers(**"/admin/\*\*"**).hasRole(**"ADMIN"**)  
 .antMatchers(**"/devices"**).hasRole(**"USER"**)  
 *//Доступ разрешен всем пользователей* .antMatchers(**"/"**, **"/resources/\*\*"**, **"/registration"**).permitAll()  
 *//Все остальные страницы требуют аутентификации* .anyRequest().authenticated()  
 .and()  
 *//Настройка для входа в систему* .formLogin()  
 .loginPage(**"/login"**)  
 *//Перенарпавление на главную страницу после успешного входа* .defaultSuccessUrl(**"/"**)  
 .permitAll()  
 .and()  
 .logout()  
 .permitAll()  
 .logoutSuccessUrl(**"/"**);  
 }  
  
 @Autowired  
 **protected void** configureGlobal(AuthenticationManagerBuilder auth) **throws** Exception {  
 auth.userDetailsService(**userService**).passwordEncoder(bCryptPasswordEncoder());  
 }  
}

3.3.6 Веб-приложение для работы с модулем

Для проверки работы модуля пока организовал несколько страниц. На страницах организована чистая функциональность, без приукрашательств. Организован переход между страницами, перенаправление в зависимости от успешности регистрации и авторизации. Организована логика взаимодействия с пользователем. Обработаны некоторые ошибки.

Стартовая страница

<%@ taglib prefix=**"c"** uri=**"http://java.sun.com/jsp/jstl/core"** %>  
<%@ taglib prefix=**"sec"** uri=**"http://www.springframework.org/security/tags"** %>  
<%@ page language=**"java"** contentType=**"text/html; charset=utf-8"** pageEncoding=**"utf-8"** %>  
  
<!DOCTYPE HTML>  
<html>  
<head>  
 <title>Главная</title>  
 <meta http-equiv=**"Content-Type"** content=**"text/html; charset=UTF-8"**/>  
 <link rel=**"stylesheet"** type=**"text/css"** href=**"${contextPath}/resources/css/style.css"**>  
</head>  
<body>  
<div>  
 <sec:authorize access=**"!isAuthenticated()"**>  
 <h3>Пользователь не авторизован</h3>  
 <h4><a href=**"/login"**>Войти</a></h4>  
 </sec:authorize>  
 <sec:authorize access=**"isAuthenticated()"**>  
 <h3>Авторизован пользователь: ${pageContext.request.userPrincipal.name}</h3>  
 <h4><a href=**"/logout"**>Выйти</a></h4>  
 </sec:authorize>  
 <h4><a href=**"/registration"**>Зарегистрировать нового пользователя</a></h4>  
 <h4><a href=**"/devices"**>Приборы (только зарегистрированные пользователи)</a></h4>  
 <h4><a href=**"/admin"**>Пользователи (только админ)</a></h4>  
</div>  
</body>  
</html>

Страница авторизации

<%@ taglib prefix=**"sec"** uri=**"http://www.springframework.org/security/tags"** %>  
<%@ page language=**"java"** contentType=**"text/html; charset=utf-8"** pageEncoding=**"utf-8"** %>  
  
<!DOCTYPE html>  
<html>  
<head>  
 <meta charset=**"utf-8"**>  
 <title>Log in with your account</title>  
</head>  
  
<body>  
<sec:authorize access=**"isAuthenticated()"**>  
 <% response.sendRedirect(**"/"**); %>  
</sec:authorize>  
<div>  
 <form method=**"POST"** action=**"/login"**>  
 <h2>Вход в систему</h2>  
 <div>  
 <input name=**"username"** type=**"text"** placeholder=**"Username"** autofocus=**"true"**/>  
 <input name=**"password"** type=**"password"** placeholder=**"Password"**/>  
 <button type=**"submit"**>Log In</button>  
 <h4><a href=**"/registration"**>Зарегистрироваться</a></h4>  
 </div>  
 </form>  
</div>  
</body>  
</html>

Страница регистрации нового пользователя

<%@ taglib prefix=**"form"** uri=**"http://www.springframework.org/tags/form"** %>  
<%@ taglib prefix=**"c"** uri=**"http://java.sun.com/jsp/jstl/core"** %>  
<%@ page language=**"java"** contentType=**"text/html; charset=utf-8"** pageEncoding=**"utf-8"** %>  
  
<!DOCTYPE html>  
<html>  
<head>  
 <meta charset=**"utf-8"**>  
 <title>Регистрация</title>  
</head>  
<body>  
<div>  
 <form:form method=**"POST"** modelAttribute=**"userForm"**>  
 <h2>Регистрация</h2>  
 <div>  
 <form:input type=**"text"** path=**"username"** placeholder=**"Username"** autofocus=**"true"**></form:input>  
 <form:errors path=**"username"**></form:errors>  
 ${usernameError}  
 </div>  
 <div>  
 <form:input type=**"password"** path=**"password"** placeholder=**"Password"**></form:input>  
 </div>  
 <div>  
 <form:input type=**"password"** path=**"passwordConfirm"** placeholder=**"Confirm your password"**></form:input>  
 <form:errors path=**"password"**></form:errors>  
 ${passwordError}  
 </div>  
 <button type=**"submit"**>Зарегистрироваться</button>  
 </form:form>  
 <a href=**"/"**>Главная</a>  
</div>  
</body>  
</html>

Страница администрирования учетных записей

<%@ taglib prefix=**"c"** uri=**"http://java.sun.com/jsp/jstl/core"** %>  
<%@ page language=**"java"** contentType=**"text/html; charset=utf-8"** pageEncoding=**"utf-8"** %>  
  
<!DOCTYPE html>  
<html>  
<head>  
 <meta charset=**"utf-8"**>  
 <title>Log in with your account</title>  
 <link rel=**"stylesheet"** type=**"text/css"** href=**"${contextPath}/resources/css/style.css"**>  
</head>  
  
<body>  
<div>  
 <table>  
 <thead>  
 <th>ID</th>  
 <th>UserName</th>  
 <th>Password</th>  
 <th>Roles</th>  
 </thead>  
 <c:forEach items=**"${allUsers}"** var=**"user"**>  
 <tr>  
 <td>${user.id}</td>  
 <td>${user.username}</td>  
 <td>${user.password}</td>  
 <td>  
 <c:forEach items=**"${user.roles}"** var=**"role"**>${role.name}; </c:forEach>  
 </td>  
 <td>  
 <form action=**"${pageContext.request.contextPath}/admin"** method=**"post"**>  
 <input type=**"hidden"** name=**"userId"** value=**"${user.id}"**/>  
 <input type=**"hidden"** name=**"action"** value=**"delete"**/>  
 <button type=**"submit"**>Delete</button>  
 </form>  
  
 </td>  
  
 </tr>  
 </c:forEach>  
 </table>  
 <a href=**"/"**>Главная</a>  
</div>  
</body>  
</html>

3.3.7 Общие параметры запуска

Для запуска приложения используются следующие настройки описанные в application.properties

**spring.datasource.url**=**jdbc:postgresql://localhost:5432/HeatDB  
spring.datasource.username**=**postgres  
spring.datasource.password**=**postgres  
spring.jpa.show-sql**=**true  
spring.jpa.generate-ddl**=**false  
spring.jpa.hibernate.ddl-auto**=**update  
spring.jpa.properties.hibernate.jdbc.lob.non\_contextual\_creation**=**true  
  
spring.mvc.view.prefix** = **/WEB-INF/jsp/  
spring.mvc.view.suffix** = **.jsp**

База данных развернута в контейнере Docker и запускается через .bat файл

docker-compose.exe up -d

Настройки запуска прописаны в docker-compose.yml

**version**: **'3'  
services**:  
 **postgres**:  
 **image**: postgres  
 **ports**:  
 - **"5432:5432"  
 environment**:  
 - POSTGRES\_PASSWORD=postgres  
 - POSTGRES\_USER=postgres  
 - POSTGRES\_DB=HeatDB  
 **volumes**:  
 - ./database:/var/lib/postgresql/data

3.4 Модуль сбора и регистрации данных с приборов

Данные с приборов можно разделить на две большие группы:

- данные настройки и конфигурации прибора;

- часовые и суточные данные учета тепла.

Приборы являются сложными устройствами, способными вести учет тепловой энергии сразу в двух системах, используя для вычислений до 7 датчиков температуры, до 6 датчиков давления, до 6 датчиков расхода. При этом каждый из подключенных датчиков конфигурируется в приборе несколькими параметрами.

Часовые и суточные данные также представлены множеством значений, в зависимости от режима работы прибора (штатный, с ошибкой/ами по каналам датчиков, с отключенным электропитанием и т.д.) При этом накопление данных в постоянной памяти ведется в течение 5-ти лет, а хранение после отключения – в течение 10-ти.

3.4.1 Основные сущности модуля

Проработка архитектуры и формата хранения еще на этапе проектирования, поэтому классы прибора и часовых данных заменены на классы-заглушки.

Модуль представлен двумя сущностями: Device и Data.

@Entity  
@Table(name = **"t\_devices"**)  
@Getter  
@Setter  
@EqualsAndHashCode(of = **"id"**)  
**public class** Device {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.***AUTO***)  
 @Column(name = **"ID"**, updatable = **false**, nullable = **false**)  
 **private** Long **id** = **null**;  
  
 @Column(name = **"number"**)  
 **private** String **number**;  
  
 @Column(name = **"name"**, nullable = **false**)  
 **private** String **name**;  
  
 @ManyToOne(cascade = CascadeType.***ALL***, fetch = FetchType.***LAZY***)  
 @JsonBackReference  
 **private** User **user**;  
  
 @OneToMany(cascade = CascadeType.***ALL***, mappedBy = **"device"**, fetch = FetchType.***LAZY***,

orphanRemoval = **true**)  
 @JsonManagedReference  
 **private** Set<Data> **data** = **new** HashSet<>();  
  
 **public void** setDeviceData(Set<Data> data) {  
 **this**.**data**.clear();  
 **if** (data != **null**) {  
 **this**.**data**.addAll(data);  
 }  
 }  
}

@Entity  
@Table(name = **"t\_data"**)  
@Getter  
@Setter  
@EqualsAndHashCode(of = **"id"**)  
**public class** Data **implements** Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.***AUTO***)  
 @Column(name = **"id"**, updatable = **false**, nullable = **false**)  
 **private** Long **id** = **null**;  
  
 @Column(name = **"data"**, updatable = **false**, nullable = **false**)  
 **private** String **data** = **null**;  
  
  
 @ManyToOne(fetch = FetchType.***LAZY***)  
 @JsonBackReference  
 **private** Device **device**;  
}

3.4.2 Хранение данных

Хранение данных организовано в БД PostgreSQL. Используем интерфейсы, расширяющие JPAReposytory.

**public interface** DeviceRepository **extends** JpaRepository<Device, Long> {  
 List<Device> findAllByUser(User user);  
 Device findByNumber(String number);  
}

**public interface** DataRepository **extends** JpaRepository<Data, Long> {  
}

3.4.3 Сервис

Сервисный слой представлен интерфейсом DeviceService и классом DeviceServiceImpl его реализующим.

Интерфейс DeviceService введен для создания архитектурной границы.

**public interface** DeviceService {  
 Device get(Long id);  
 Device get(String number);  
 List<Device> getAll();  
 List<Device> findAllByUser(User user);  
 **void** create(Device device);  
}

Пока реализовано добавление прибора пользователем, отображение всех приборов пользователя, отображение конкретного прибора по номеру.

@Service  
**public class** DeviceServiceImpl **implements** DeviceService{  
 **private** DeviceRepository **deviceRepository**;  
  
 **private** DataRepository **dataRepository**;  
  
 **public** DeviceServiceImpl(DataRepository dataRepository, DeviceRepository deviceRepository) {  
 **this**.**dataRepository** = dataRepository;  
 **this**.**deviceRepository** = deviceRepository;  
 }  
  
 @Override  
 **public** Device get(Long id) {  
 **return null**;  
 }  
  
 @Override  
 **public** Device get(String number) {  
 **return deviceRepository**.findByNumber(number);  
 }  
  
 @Override  
 **public** List<Device> getAll() {  
 **return null**;  
 }  
 @Override  
 **public** List<Device> findAllByUser(User user){  
 **return deviceRepository**.findAllByUser(user);  
 }  
  
 @Override  
 **public void** create(Device device) {  
 **deviceRepository**.save(device);  
 System.***out***.println(device);  
 }  
}

3.4.4 Контроллеры

Контроллер представлен пока одним контроллером DeviceController.

DeviceController отвечает за регистрацию нового устройства в системе, выдачу информации по конкретному устройству, выдачу всех устройств пользователя.

Данный контроллер организован как REST-контроллер, возвращающий данные в формате JSON.

Контролер работает пока сразу с двумя сервисами, и сервисом устройств, и сервисом пользователей. В дальнейшем планируется выделить работу с устройствами в отдельный микросервис.

@RestController  
@RequestMapping(**"/devices"**)  
**public class** DeviceController {  
  
 **private** DeviceService **deviceService**;  
 **private** UserService **userService**;  
  
 **public** DeviceController(DeviceService deviceService, UserService userService) {  
 **this**.**deviceService** = deviceService;  
 **this**.**userService** = userService;  
 }  
  
 @GetMapping()  
 **public** List<Device> getDevices(Principal principal) {  
 **return deviceService**.findAllByUser(**userService**.findUserByUsername(principal.getName()));  
 }

@RequestMapping(value = **"/{deviceNumber}"**, method = RequestMethod.***GET***,

produces = MediaType.***APPLICATION\_JSON\_VALUE***)  
 @ResponseStatus(value = HttpStatus.***OK***)  
 **public** @ResponseBody  
 Device get(@PathVariable String deviceNumber) {  
 **return deviceService**.get(deviceNumber);  
 }  
 @RequestMapping(value = **"/"**, method = RequestMethod.***POST***,

produces = MediaType.***APPLICATION\_JSON\_VALUE***)  
 @ResponseStatus(value = HttpStatus.***OK***)  
 **public** ResponseEntity<?> create(Principal principal, @RequestBody Device device) {  
 device.setUser(**userService**.findUserByUsername(**"admin"**));  
 **deviceService**.create(device);  
 HttpHeaders headers = **new** HttpHeaders();  
 ControllerLinkBuilder linkBuilder = *linkTo*(*methodOn*(DeviceController.**class**).get(device.getNumber()));  
 headers.setLocation(linkBuilder.toUri());  
 **return new** ResponseEntity<>(headers, HttpStatus.***CREATED***);  
 }  
}

3.4.5 Настройки безопасности

Настройки безопасности пока представлены классом WebSecurityConfig расширающим WebSecurityConfigurerAdapter. Доступ к ветке /devices разрешен, авторизованным пользователям с ролью USER.

@Configuration  
@EnableWebSecurity  
**public class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {  
 @Autowired  
 UserService **userService**;  
  
 @Bean  
 **public** BCryptPasswordEncoder bCryptPasswordEncoder() {  
 **return new** BCryptPasswordEncoder();  
 }  
@Override  
 **protected void** configure(HttpSecurity httpSecurity) **throws** Exception {  
 httpSecurity  
 .csrf()  
 .disable()  
 .authorizeRequests()  
 *//Доступ только для пользователей с ролью Администратор* .antMatchers(**"/admin/\*\*"**).hasRole(**"ADMIN"**)  
 .antMatchers(**"/devices"**).hasRole(**"USER"**)  
 *//Доступ разрешен всем пользователей* .antMatchers(**"/"**, **"/resources/\*\*"**, **"/registration"**).permitAll()  
 *//Все остальные страницы требуют аутентификации* .anyRequest().authenticated()  
 .and()  
 *//Настройка для входа в систему* .formLogin()  
 .loginPage(**"/login"**)  
 *//Перенарпавление на главную страницу после успешного входа* .defaultSuccessUrl(**"/"**)  
 .permitAll()  
 .and()  
 .logout()  
 .permitAll()  
 .logoutSuccessUrl(**"/"**);  
 }  
  
 @Autowired  
 **protected void** configureGlobal(AuthenticationManagerBuilder auth) **throws** Exception {  
 auth.userDetailsService(**userService**).passwordEncoder(bCryptPasswordEncoder());  
 }  
}

3.4.6 Веб-приложение для работы с модулем

Веб-часть доступа к модулю еще только проектируется. Возможно, сервис так и останется в виде REST-сервиса, для принятия, хранения и предоставления данных об устройствах, а визуальная часть будет предусмотрена в сервисе предоставления отчетов.

3.4.7 Общие параметры запуска

Для запуска модуля пока используются совместные с модулем пользователей настройки приложения описанные в application.properties.

**spring.datasource.url**=**jdbc:postgresql://localhost:5432/HeatDB  
spring.datasource.username**=**postgres  
spring.datasource.password**=**postgres  
spring.jpa.show-sql**=**true  
spring.jpa.generate-ddl**=**false  
spring.jpa.hibernate.ddl-auto**=**update  
spring.jpa.properties.hibernate.jdbc.lob.non\_contextual\_creation**=**true  
  
spring.mvc.view.prefix** = **/WEB-INF/jsp/  
spring.mvc.view.suffix** = **.jsp**

Таблицы базы данных в БД пользователей. БД развернута в контейнере Docker и запускается через .bat файл

docker-compose.exe up -d

Настройки запуска прописаны в docker-compose.yml

**version**: **'3'  
services**:  
 **postgres**:  
 **image**: postgres  
 **ports**:  
 - **"5432:5432"  
 environment**:  
 - POSTGRES\_PASSWORD=postgres  
 - POSTGRES\_USER=postgres  
 - POSTGRES\_DB=HeatDB  
 **volumes**:  
 - ./database:/var/lib/postgresql/data

Заключение

В результате проектирования вэб – приложения был создан сервис регистрации и авторизации пользователей, начата разработка сервиса сбора, хранения и предоставления данных с приборов учета, начато обсуждение сервиса по автоматизированному представлению отчетов учета тепловой энергии по расписанию.

Чтобы собрать проект и запустить его как сайт, был использован сборщик Maven. Собранный проект представляет из себя приложение с расширением .jar.

В дальнейшем планирую дальше заниматься разработкой этого вэб – приложение.

Ссылка на github: https://github.com/karpenka-aliaksandr/JAVA\_HeatReader.git

Список использованных источников

GeekBrains

2024

У

Изм.

Лист

№ докум.

Подпись

Дата

Лист

1

ДП 00.00.ПЗ.

Список использованных

источников

Лит.

Листов

1

Разраб.

Карпенко А.О

Провер.

Утверд.

Н.контр.

.

1. <https://tproger.ru/articles/pishem-java-veb-prilozhenie-na-sovremennom-steke-s-nulja-do-mikroservisnoj-arhitektury-chast-1>
2. <https://stackoverflow.com/>
3. <https://ru.wikipedia.org/>
4. <https://mvnrepository.com/>
5. <https://habr.com/>
6. <https://azure.microsoft.com/ru-ru/>
7. <https://www.nic.ru/>
8. <https://javarush.com/>
9. <https://github.com/>
10. <https://cp.beget.com/>