МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Информационные технологии»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

К ВЫПОЛНЕНИЮ КОНТРОЛЬНЫХ РАБОТ ПО ДИСЦИПЛИНЕ «СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ»

Ростов-на-Дону
ДГТУ
2022

УДК 372.8:004

Составитель М.В. Привалов

Методические указания к выполнению контрольных работ по дисциплине «Современные технологии программирования» / сост. М.В. Привалов . – Ростов-на-Дону: Донской государственный технический университет, 2022. – 15 с.

Рассматриваются современные подходы и технологии разработки программного обеспечения для Web-ориентированных и распределённых информационных систем.

Предназначены для обучающихся направления 09.04.02 «Информационные технологии» всех форм обучения.

УДК 372.8:004

Печатается по решению редакционно-издательского совета
Донского государственного технического университета

Ответственный за выпуск зав. кафедрой «Информационные технологии»,
д-р техн. наук, профессор Б.В. Соболь

В печать 12.05.2022.
Формат 60×84/16. Объем 0,9 усл.п.л.
Тираж 50 экз. Заказ № 185.

Издательский центр ДГТУ
Адрес университета и полиграфического предприятия:
344000, г. Ростов-на-Дону, пл. Гагарина,1

СОДЕРЖАНИЕ

КОНТРОЛЬНАЯ РАБОТА №1. ПРИМЕНЕНИЕ JAVA PERSISTENCE API И ORM HIBERNATE ДЛЯ РАБОТЫ С РСУБД В ПРИЛОЖЕНИЯХ НА JAVA

Цель работы: получить навыки по работе с реляционными базами данных в Java. Научиться использовать отображение объектов на реляционную модель с использованием современных ORM-каркасов.

ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Hibernate — это популярный ORM-каркас, цель которого – связать ООП и реляционную базу данных. Работа с Hibernate сокращает время разработки проекта в сравнении с обычным JDBC, так как создаёт связь между таблицами в базе данных (далее – БД) и Java-классами и наоборот. Схематично это можно отразить так (рис. 1):

Рис. 1. Hibernate и реляционная СУБД в Java-приложении

Преимущества использования Hibernate:

– Обеспечивает простой API для записи и получения Java-объектов в/из БД.
– Минимизирует доступ к БД, используя стратегии fetching.
– Не требует сервера приложения.
– Позволяет нам не работать с типами данных языка SQL, а иметь дело с привычными нам типами данных Java (POJO – Plain Old Java Objects).
– Заботится о создании связей между Java-классами и таблицами БД с помощью XML-файлов или аннотаций, не внося изменения в программный код. Если нам необходимо изменить БД, то достаточно лишь внести изменения в XML-файлы.

Hibernate поддерживает все популярные СУБД: MySQL, Oracle, PostgreSQL, Microsoft SQL Server Database, HSQL, DB2 и может работать в связке с такими технологиями, как Maven и J2EE, а также NoSQL СУБД.

Приложение, использующее Hibernate, очень упрощённо имеет следующую архитектуру (рис. 2.):

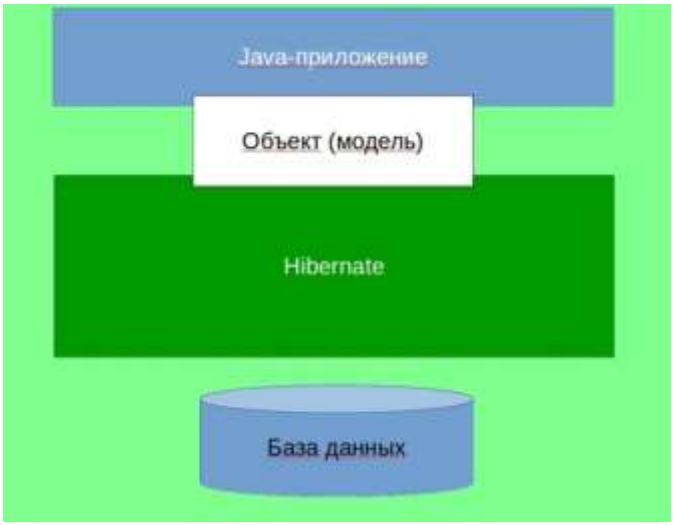

Рис. 2. Упрощённая архитектура Java-приложения, использующего Hibernate

Если мы рассмотрим строение самого Hibernate более подробно, то этот же рисунок будет уже выглядеть следующим образом (рис. 3):

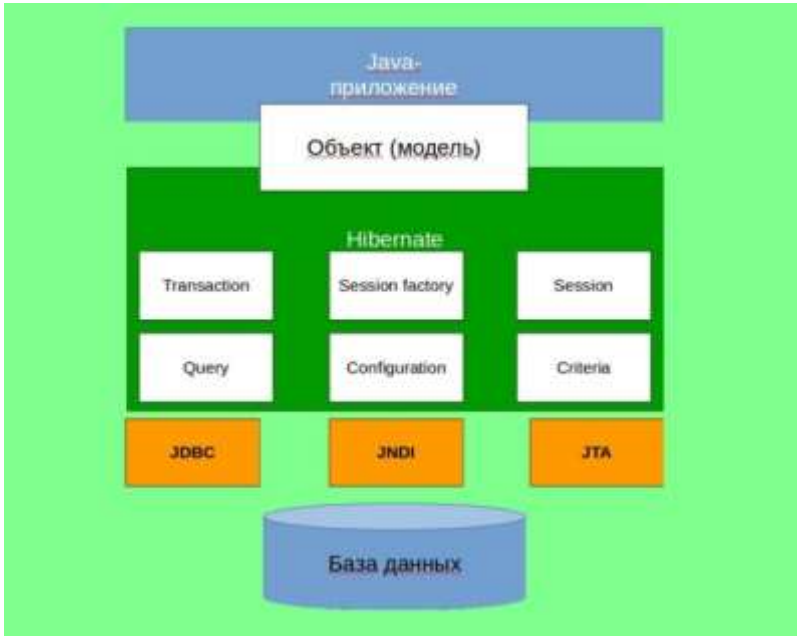

Рис. 3. Детальная архитектура приложения с Hibernate

Hibernate поддерживает такие API, как JDBC, JNDI, JTA.

JDBC обеспечивает простейший уровень абстракции функциональности для реляционных БД. JTA и JNDI, в свою очередь, позволяют Hibernate использовать серверы приложений J2EE.

Transaction. Представляет собой рабочую единицу, изолирующую порцию работы с БД. В Hibernate транзакции обрабатываются менеджером транзакций.

SessionFactory. Самый важный и самый сложный и функционально насыщенный объект, который представляет собой фабрику. Обычно фабрика создаётся в единственном экземпляре, при запуске приложения. Как минимум одна SessionFactory требуется для каждой БД, при этом каждая из них конфигурируется отдельным конфигурационным файлом.

Session. Сессия используется для получения физического соединения с БД. Обычно сессия создаётся при необходимости, а после этого закрывается. Это возможно, потому что эти объекты крайне легковесны. В целом, можно сказать, что создание, чтение, изменение и удаление объектов происходит через объект Session.

Query. Этот объект использует HQL или SQL для чтения/записи данных из/в БД. Экземпляр запроса используется для связывания параметров запроса, ограничения количества результатов, которые будут возвращены, и для выполнения запроса.

Configuration. Этот объект используется для создания объекта SessionFactory и конфигурирует сам Hibernate с помощью конифигурационного XML-файла, который объясняет, как обрабатывать объект Session.

Criteria. Используется для создания и выполнения объектно-ориентированных запросов для получения объектов. Критерии часто используются при реализации поиска информации в БД на основании пользовательских критериев.

Необходимые шаги по подключению Hibernate к проекту.

Для работы с Hibernate понадобится, как и в предыдущей работе, подключить необходимые библиотеки и выполнить некоторые настройки. Более детально:

1. Подключить в виде JAR-библиотеки драйвер соответствующей СУБД.
2. Подключить JAR-библиотеку с самим Hibernate (рекомендуется версия 5.x).
3. Выполнить конфигурацию ORM-каркаса в hibernate.cfg.xml.
4. Создать XML-конфигурации сущностей или аннотированные классы.

Для конфигурирования сущностей в настоящее время чаще применяют аннотации. Как сконфигурировать сущности, на которые отражаются таблицы в БД, можно изучить на ресурсе https://javarush.ru/groups/posts/hibernate-java или https://proselyte.net/tutorials/hibernate-tutorial/introduction/

Следует отметить, что первые два пункта требуется выполнять, используя системы и технологии управления зависимостями приложения (Maven, Gradle и им подобные). Как с ними работать, показано ниже.

НАСТРОЙКА, СБОРКА ПРОЕКТОВ И УПРАВЛЕНИЕ ЗАВИСИМОСТЯМИ

Все современные IDE позволяют управлять зависимостями проектов, собрать, запускать и даже развёртывать приложения. Однако, для универсальности, как правило, используют сторонние средства управления зависимостями и сборки проектов. Для Java приложений это Ant, Maven, Gradle.

Apache Maven — каркас для автоматизации управления зависимостями и сборки проектов, специфицированных на XML-языке POM(Project Object Model).

На примере бесплатно распространяемой популярной IDE Eclipse, создание проекта выполняется через меню File->New->Maven project. На первом экране настройки проекта требуется отметить Create simple project, чтобы не выполняль настройку архетипа проекта. После этого потребуется заполнить базовые поля Maven-проекта:

Group Id: наименование отдела или организации (правила аналогичны именованию пакетов)

Artifact Id: название проекта

Далее будет создан новый проект, в котором присутствует файл pom.xml, отвечающий за настройку разрешения зависимостей и сборку. Его содержимое может быть примерно таким (в зависимости от заполнения указанных выше полей):

```
<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  https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.dstu</groupId>
  <artifactId>dbdemo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</project>
```

Структура проекта при этом будет иметь вид, показанный на рис. 4:

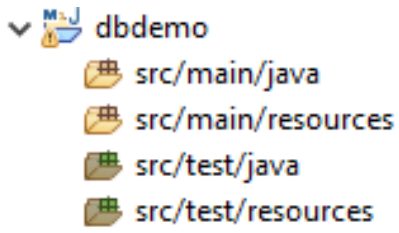

Рис. 4. Структура нового Maven-проекта

Нас будет интересовать исключительно папка src/main/java, в которой будут находиться все исходные тексты нашего приложения.

Для подключения библиотек требуется создать элемент dependencies и каждую библиотеку вписать как отдельный элемент dependency.

Например, в случае Postgresql драйвер JDBC для доступа к данной СУБД может быть добавлен в конфигурацию следующим образом:

```
<dependencies>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <version>9.4.1211</version>
    </dependency>
</dependencies>
```

Для сборки проекта Maven используется команда mvn compile.

После сборки (из IDE или командной строки mvn compile) мы получим в случае успеха следующие сообщения:

```
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------------< org.dstu:dbdemo >--------------------------
[INFO] Building dbdemo 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ dbdemo ---
[WARNING] Using platform encoding (Cp1251 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ dbdemo ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding Cp1251, i.e. build is platform dependent!
[INFO] Compiling 1 source file to C:\Users\max\eclipse-workspace\dbdemo\target\classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.245 s
[INFO] Finished at: 2019-12-22T17:13:08+03:00
[INFO] ------------------------------------------------------------------------
```

Чтобы не получать предупреждений о платформенно-зависимой сборке, желательно явно указать кодировку исходных файлов и текстовых ресурсов:

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```

```
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------------< org.dstu:dbdemo >--------------------------
[INFO] Building dbdemo 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ dbdemo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ dbdemo ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\max\eclipse-workspace\dbdemo\target\classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.290 s
[INFO] Finished at: 2019-12-22T17:14:44+03:00
[INFO] ------------------------------------------------------------------------
```

ПОРЯДОК ВЫПОЛНЕНИЯ КОНТРОЛЬНОЙ РАБОТЫ

При выполнении работы создайте программу, которая получает на вход CSV файл с информацией об объектах согласно варианту. Исходный файл нужно организовать самостоятельно, согласно определённым самостоятельно свойствам сущностей. Программа должна считывать эти данные и помещать их в БД. При этом вся работа с БД должна вестись посредством объектно-реляционного отображения с применением Hibernate.

Требования к программе.

− Конфигурация зависимостей должна быть выполнена посредством Maven.
− Входной файл задается в качестве параметра при запуске программы.
− Каждый объект должен содержать 3 - 5 свойств.
− Отражение таблиц на объектную модель должно быть задано с применением аннотаций.
− Реализуйте DAO для доступа к соответствующим сущностям.

− Реализуйте и продемонстрируйте работу следующих запросов:

- запроса на вставку новых данных (внесите в БД данные из CSV файла);
- запроса на вывод всех объектов из БД;
- запроса на изменение одного и нескольких объектов в БД;
- запроса на выборку данных по условию (требуется задать условие, задействующее все таблицы, а сам запрос – в виде HQL - https://proselyte.net/tutorials/hibernate-tutorial/hibernate-query-language/).

ВАРИАНТЫ ЗАДАНИЙ

1) студент, преподаватель, группа;
2) телевизор, монитор, категория товаров;
3) авторучка, карандаш, отдел канцтоваров;
4) катер, водный мотоцикл, причал;
5) диван, кровать, комната;
6) самолет, корабль, (аэро)порт;
7) штатный сотрудник, совместитель, кафедра;
8) эллипс, многоугольник, презентация;
9) книга, журнал, полка;
10) зачет, экзамен, предмет;
11) город, село, регион;
12) млекопитающее, птица, зоопарк;
13) детская коляска, автокресло, гараж;
14) цифровая и обычная фоторамка, фото;
15) кондиционер, обогреватель;
16) квадрат, прямоугольник, рисунок;
17) стол, стул, офис;
18) ноутбук, планшет, установленная операционная система;
19) HDD, SDD, сервер;
20) клавиатура, мышь, компьютер;
21) беговая дорожка, степпер, зал;
22) мобильный телефон, смартфон, приложение.

СОДЕРЖАНИЕ ОТЧЁТА

В отчёте по контрольной работе приведите следующее:

1. Краткие сведения о выбранной РСУБД
2. Схему данных построенной согласно заданию физической модели данных.
3. Конфигурационные файлы Hibernate, приложения (если есть) и Maven.
4. Результаты работы, демонстрирующие работу всех запросов, оговоренных в задании.
5. Код программы (рекомендуется вынести в приложение).

КОНТРОЛЬНАЯ РАБОТА №2. ПРОЕКТИРОВАНИЕ И СОЗДАНИЕ WEB-СЛУЖБ С ИСПОЛЬЗОВАНИЕМ АРХИТЕКТУРНОГО ПАТТЕРНА REST

Цель работы**:** научиться реализовывать Web-службы и их клиенты для многозвенных и распределённых систем, построенных с использованием архитектурного паттерна REST.

ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ.

Основным недостатком SOAP XML служб является использование XML в качестве формата сообщений, передаваемых между клиентом и службой. Данный формат имеет очень высокие накладные расходы, связанные с наличием, помимо данных, открывающих и закрывающих тегов. Разбор этих данных требует заметного количества ресурсов процессора и объёмов ОЗУ. Службы WCF тоже не лишены данного недостатка: эта технология значительно более продвинута, по сравнению со старой ASP .NET, при этом она также может использовать для представления данных JSON помимо XML формата. Тем не менее, стандартные коммуникации опираются на тот же SOAP. Плюс, технология является проприетарной.

Альтернативой, которая очень популярна в настоящее время, является применение архитектурного паттерна REST. Он подразумевает применение стандартизированного кодирования URL ресурсов и применение JSON в качестве формата передачи данных.

Примеры URL:

/user/?id – запрос пользователя с идентификатором id

/user/list – запрос полного списка пользователей

/user/search – запрос списка пользователей по критерию

/user/save – сохранение пользователя (подразумевает данные в теле POST)

Реализовать такую службу на Java можно несколькими способами. Самые очевидные из них:

1. Реализация без использования сторонних фреймворков.
2. Реализация с использованием Jersey.
3. Реализация на основе Spring Framework.

СПОСОБЫ РЕАЛИЗАЦИИ RESTFUL СЛУЖБ В JAVA. СТРУКТУРА ПРОЕКТОВ И УПРАВЛЕНИЕ ЗАВИСИМОСТЯМИ.

1. Реализация RESTful службы стандартными средствами платформы Java

Предположим, что мы хотим реализовать службу, поддерживающую методы GET и POST, а также способную возвращать список пользователей по запросу /user/list

1.1 Создадим сервер и контекст. Для этого воспользуемся классом HttpServer.

```
HttpServer httpServer = HttpServer.create(new InetSocketAddress(PORT), 0);
httpServer.createContext("/user/list", httpExchange -> {
   // код обработки запроса
});
```

1.2 Проверим тип запроса. Для GET реализуем вывод списка пользователей, а в случае POST выбросим клиенту ошибку с кодом 405 (Method not allowed)

```
if ("GET".equals(httpExchange.getRequestMethod())) {
    // обработка GET
} else {
  httpExchange.sendResponseHeaders(NOT_ALLOWED, -1);
}
```

1.3 Для реализации выдачи списка пользователей требуется выполнить 3 действия:

a. Задать заголовки (как минимум, указать Content-Type). Для этого надо получить заголовки методом getResponseHeaders() и задать требуемые, используя методы интерфейса Map. После этого надо выдать их с кодом возврата сервера (если нормально, то это код 200), используя метод sendResponseHeaders(200, 0).

b. Получить представление списка пользователей в формате JSON (подразумевается, что у вас есть POJO класс, представляющий пользователя, допустим, User, а также список пользователей). Для этого можно использовать JSON simple, но лучше – библиотеку Jackson.

c. JSON в виде массива символов записать в тело ответа. Тело ответа является потоком вывода, поэтому проблем с этим быть не должно: его нужно получить методом getResponseBody(), записать результат и закрыть.

d. Указать серверу пустой исполнитель с помощью setExecutor(null); и запустить сервер методом start().

Пояснения.

Чтобы подключить библиотеку Jackson, требуется в pom.xml добавить зависимость:

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.11.3</version>
</dependency>
```

Версия может отличаться.

Для конвертирования списка в JSON потребуется объект класса ObjectMapper из этой библиотеки. В случае успешного запуска должен быть открыт сокет на порту PORT (см. пример создания сервера выше), а тестовый REST-клиент по запросу /user/list должен выдать фрагмент JSON (рис. 5).

Рис. 5. Результат тестирования службы

Неудобство данного метода: в случае наличия параметров службы, их придётся вручную разбирать, анализируя запрос, который можно получить посредством getRequestURI().getRawQuery().

2. Реализация RESTful службы с помощью проекта Jersey

Более продвинутый проект для реализации конечных точек REST – это проект Jersey. Его преимущества:

- отображение классов и методов на пути в URI
- отображение параметров запроса на аргументы методов класса
- управление путями, параметрами и рядом заголовков с помощью аннотаций

Подключение Jersey к проекту.

Для подключения к проекту нам понадобится добавить следующие зависимости в pom.xml:

```
<dependency>
  <groupId>org.glassfish.jersey.containers</groupId>
  <artifactId>jersey-container-servlet</artifactId>
  <version>2.33</version>
</dependency>
<dependency>
  <groupId>org.glassfish.jersey.inject</groupId>
  <artifactId>jersey-hk2</artifactId>
  <version>2.26</version>
</dependency>
```

```
<dependency>
   <groupId>org.glassfish.jersey.media</groupId>
   <artifactId>jersey-media-json-jackson</artifactId>
   <version>2.33</version>
</dependency>
```

Первая обеспечивает доступ к контейнеру, вторая – инъекцию зависимостей, третья – сериализацию и десериализацию данных в JSON.

Далее пишем код нашей RESTful службы. Простейший пример показан ниже:

```
@Path("/message")
public class JerseyService
{
   @GET
   public String getMsg()
   {
       return "Hello World !! - Jersey 2";
   }
}
```

Но чтобы служба заработала, она должна быть частью Web-приложения. При этом Jersey должен быть обработчиком URI, путь к которому определяет доступ к конечным точкам REST. Для этого требуется в web.xml добавить следующую конфигурацию:

```
<servlet>
      <servlet-name>jersey-serlvet</servlet-name>
      <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
      <init-param>
          <param-name>jersey.config.server.provider.packages</param-name>
          <param-value>com.evil.service</param-value>
      </init-param>
      <load-on-startup>1</load-on-startup>
   </servlet>
     <servlet-mapping>
      <servlet-name>jersey-serlvet</servlet-name>
      <url-pattern>/rest/*</url-pattern>
   </servlet-mapping>
```

В этой конфигурации com.evil.service – это пакет Java, содержащий классы, предоставляющие конечные точки REST для взаимодействия. Путь /rest/ - это путь в контексте приложения к этим конечным точкам. Так, в нашем примере, путь к ресурсу в контексте приложения будет /rest/message/

Приложение развёртываем в Tomcat, после чего конечные точки становятся доступны.

Аннотации @Produces и @Consumes позволяют управлять тем, что выдаёт служба (обычный текст, JSON, XML и т.п.).

Для отображения параметров на аргументы используйте их имена в фигурных скобках:

```
@Path("/fetch/{id}")
public Response fetch(@PathParam("id") String id) { }
```

Особенности возврата generic-коллекций.

Как правило, для нормальной разработки потребуется возвращать списки сущностей, то есть, пользоваться generic-коллекциями (List, Set, Map и т.п.). Но из-за особенностей Java при сериализации мы потеряем информацию о параметре типа и получим ошибку в Jersey. Чтобы

этого не было, требуется воспользоваться обёрткой GenericEntity перед тем, как возвращать результат клиенту:

```
GenericEntity<List<User>> entity = new GenericEntity<List<User>>(users) {};
return Response.ok(entity).build();
```

Порядок выполнения контрольной работы

В соответствии с индивидуальным заданием требуется:

1. Определить функциональные границы Web-службы.
2. Определить контракт данных и функциональный контракт службы.
3. Создать REST-сервис с контрактом для обмена информацией сервера и клиента с применением обратной связи.
4. Продемонстрировать работу службы.

С помощью стандартных средств платформы или фреймворка Jersey реализовать вышеуказанный сервис и развернуть его в Web-контейнере. Обязательно предусмотреть в службе следующие 3 метода:

a) Возвращающий полный список сущностей (например, пользователей, клиентов, воздушных судов, заказов и т.п., согласно варианту) по запросу GET

b) Возвращающий одну сущность по её Id, переданному запросом GET

c) Добавляющий одну сущность по запросу POST и возвращающий её Id

Продемонстрировать работу сервиса с помощью тестового REST-клиента.

ВАРИАНТЫ ИНДИВИДУАЛЬНЫХ ЗАДАНИЙ

1. Спроектировать компьютерную подсистему учета Интернет-услуг
2. Спроектировать подсистему учета ресурса воздушных судов
3. Спроектировать компьютерную подсистему начисления и учета страховых выплат клиентам в условиях страховой компании
4. Спроектировать подсистему учета и анализа экономической деятельности в условиях агентства недвижимости
5. Спроектировать подсистему формирования и обработки договоров добровольного страхования наземного автотранспорта
6. Спроектировать подсистему обработки торговых агентов
7. Спроектировать подсистему планирования и учета чартеров
8. Спроектировать подсистему формирования и учета счетов-заказов для туристических агентств
9. Спроектировать подсистему бронирования билетов в кинотеатр
10. Спроектировать подсистему расчета себестоимости для рыбного хозяйства
11. Спроектировать подсистему учета потребительских кредитов
12. Спроектировать подсистему реализации и движения горюче-смазочных материалов в условиях сети АЗС
13. Спроектировать подсистему учета продаж товаров
14. Спроектировать подсистему учета сбора и реализации зерновых культур
15. Спроектировать подсистему учета убытков автогражданской ответственности
16. Спроектировать подсистему учета выплат за услуги газоснабжения для населения
17. Спроектировать подсистему расчета арендной платы в условиях коммунального предприятия
18. Спроектировать подсистему учета движения грузового подвижного железнодорожного

транспорта в условиях металлургического завода

19. Спроектировать подсистему учета сырья и материалов
20. Спроектировать подсистему приема электронных коммунальных платежей в условиях банка
21. Спроектировать подсистему учета и планирования ремонтных работ
22. Спроектировать подсистему учета технического состояния компьютерного оборудования

СОДЕРЖАНИЕ ОТЧЁТА

В отчёте следует привести:

1. Описание функциональных границ (контекста) службы.
2. Краткое описание контракта данных и контракта службы:

– Описание назначения и ключевых свойств DTO-объектов (или аналогов).
– Описание основных операций в контракте службы.

3. Характеристики конечной точки службы (endpoint).
4. Пример работы службы.
5. Конфигурационный файл Maven для сборки приложения.
6. Полный код программ со службой и клиентом (рекомендуется вынести в приложения)

ЛИТЕРАТУРА

1. Гуськова, О.И. Объектно ориентированное программирование в Java : учебное пособие / О. И. Гуськова. - Москва : МПГУ, 2018. - 240 с. - ISBN 978-5-4263-0648-6. - Текст : электронный. - URL: https://znanium.com/catalog/product/1020593
2. Прохоренок, Н. А. Основы Java: Самоучитель: учебное пособие / Н.А. Прохоренок. – СПб.: БХВ-Петербург, 2017. - 704 с. ISBN 978-5-9775-3785-8. - Текст : электронный. - URL: https://znanium.com/catalog/product/978545
3. Будилов, В. А. Интернет-программирование на Java: Пособие / В.А. Будилов. – СПб.: БХВ-Петербург, 2014. - 698 с. ISBN 978-5-9775-1931-1. - Текст : электронный. - URL: https://znanium.com/catalog/product/940239
4. Руководство по Hibernate / Записки задумчивого программиста – Proselyte. – Текст: электронный. – URL: https://proselyte.net/tutorials/hibernate-tutorial/introduction/
5. Создаем RESTful web service на Java с использованием Eclipse + Jersey + Glassfish3 /Хабр. – Текст: электронный. – URL: https://habr.com/ru/post/150176/
6. Lars Vogel, REST with Java (JAX-RS) using Jersey - Tutorial / Vogella GmbH. – Текст: электронный. – URL: https://www.vogella.com/tutorials/REST/article.html