CET2041 - Backend Development with Java - Practicum 02

Learning Objectives:

- Learning how to use Maven as a build tool.
- Using JAX RS to implement REST endpoints.
- Using JPA with Hibernate to interface with databases.
- Applying Design Patterns to an application.
- Learning to research for 3rd party libraries.

Deliverables:

- Submit a zipped file containing the necessary source code as `CET2041_P02_<Your_Name>.zip` (e.g. `CET2041_P02_John_Doe.zip`).
- Note that a non-working submission will result in a zero.

Background

The Head of Human Resources of DigiCorp decided to commission a simple internal webpage where the HR team can manage and store information about the company staff. They decided to engage the IT department to build a Full Stack application for them. As the star intern of the company, you are assigned the task of building the REST interface and the database interface to link up with the front-end, done by another colleague.

Task

The frontend website is **not required** to be developed for this practicum. You are to implement the backend application using JAX RS, JPA with Hibernate, Maven and a 3rd party JSON library (chosen from the given list below) using Java.

- Jackson - Comes with the Jersey Media bunch of libraries that is normally required for JAX RS to work.
- GSON - From Google.
- json-simple - Open sourced, simple & lightweight JSON library
- Flexjson - Lightweight library for serializing and deserializing Java objects.
- JSON-lib - Java library based on the work by Douglas Crockford.

study the database first.

The database used for this practicum is the **Employees Sample Database** from MySQL. You are to read and familiarize yourself with the documentation to understand how the database is structured and the installation procedures to setup the database in MariaDB.

Your application **must be buildable using Maven, published using Apache Tomcat and tested using Postman**.

Do take note of the following requirements:

need to check for duplicated entries?

1. There **must be** at least 4 endpoints implemented based on the following:
    1. An endpoint for getting **all the names and department numbers** of all departments in the company. Does not consume any user input.
    2. An endpoint for returning the **full employee record** given the employee number. User inputs an employee number.

this is for 1 employee

this is for as many employees as possible and it is 20 records at a time and what is counted as a page?

    3. An endpoint for getting **all employee records** (info to return: employee number, first name, last name and hire date) from a given department number and a optional page number that defaults to 1. The results are to be **paged** and each page has a **maximum of 20 records**. User inputs a department number and an optional page number to retrieve the data. **Note** that in terms of computing, page numbers start from 0 to $n-1$ but from the users point of view, pages start from 1 to n.

does the company promote people but not increase their salary?

so will need employee number, salary, salary from date, salary to date, title, title from_date, title to_date, need to also check if promoted to manager, if yes, must insert into the dept_manager, if no, insert into the dept_emp. if salary field is empty then dont insert into salaries table.

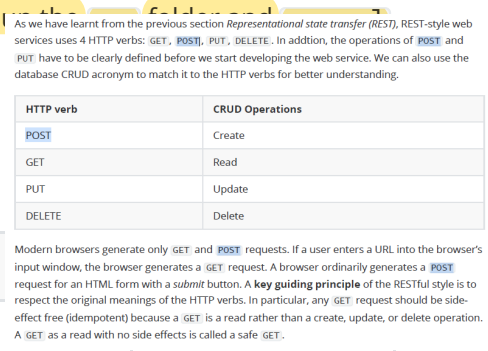
    4. An endpoint that is used for **employee promotion**. Consumes either a JSON or Plain text string of the data required for employee promotion. You are to decide which set of information is required to be sent for this task. At the minimum, promotions are to be carried out 1 at a time.
2. All data **returned** from the server is in the form of a **JSON string** using the **appropriate HTTP status codes**.
3. All related files are to be stored in **related folders**.
4. Safe to assume that user input data is always cleaned (aka numerical inputs will always be numerical) but **does not mean that they are always valid**.
5. The package file **produced** by the Maven build pipeline is a `.war` file.
6. Comments are **still required** but not gradable as it should be used to convey information to yourself and to **justify** your way of implementation.

refer jpa part 2 page 10 to know where to put em and emf or refer to page 13

mportant notes

1. Before submission, **clean** your Maven project before zipping [illegible] file.
2. Test your completed program extensively before submitting.

or the maximum allocation of marks, refer to the table below.


As we have learnt from the previous section *Representational state transfer (REST)*, REST-style web services uses 4 HTTP verbs: `GET`, `POST`, `PUT`, `DELETE`. In addtion, the operations of `POST` and `PUT` have to be clearly defined before we start developing the web service. We can also use the database CRUD acronym to match it to the HTTP verbs for better understanding.

HTTP verb	CRUD Operations
POST	Create
GET	Read
PUT	Update
DELETE	Delete

Modern browsers generate only `GET` and `POST` requests. If a user enters a URL into the browser's input window, the browser generates a `GET` request. A browser ordinarily generates a `POST` request for an HTML form with a *submit* button. A **key guiding principle** of the RESTful style is to respect the original meanings of the HTTP verbs. In particular, any `GET` request should be side-effect free (idempotent) because a `GET` is a read rather than a create, update, or delete operation. A `GET` as a read with no side effects is called a safe `GET`.

Description	
Successful implementation of JPA with Hibernate.	
Successful implementation of JAX RS.	25
Chosen and used a 3rd party JSON library.	15
Applied some design patterns correctly.	15
Properly tested, zero exceptions in server logs.	10