# **Pollinate – Technical Questions**

# Section 1

The Scope of this project is to call an API which will push the data to the database. The API is written in Java and built in Spring boot.

Table used to store the data is built in MS SQL.

|  |  |
| --- | --- |
| **Components** | **Language/ Database Type** |
| Database | MSSQL |
| API | Java/Spring boot |

## Detailed Design

The API built as part of the project inserts the timestamp into the Product table which is stored in the MS SQL database.

The product table is replicated in the API as a model class where the column names are defined as the variables in the class.

The API has two options to the user:

* GET the data from the Product table by doing a get call to the database.
* POST the data to the product table by doing a post call and passing 2 Parameters in the request which are:
  + ID: ID of the product which needs to be inserted
  + Timestamp: Timestamp which is passed to the API

Below is the diagrammatical representation of the same:

Graphical user interface, text, application

Description automatically generated

Classes\Interface in API:

|  |  |  |
| --- | --- | --- |
| **Classes** | **Type** | **Description** |
| Product | Model | A Java view of the class for the table which exists in Database |
| UserRepository | Java Interface | An interface which extends JPA's CRUD repository to do database operations |
| MainController | Java Controller class | A Java class to help build the URI's for the project which the user is going to hit in the application |
| PollinateApplication | Java Main Class | A main class to run the Spring boot Application |

Created the below table Product with table structure as mentioned in the MSSQL data:

Graphical user interface, text, application

Description automatically generated

## URI

|  |  |  |
| --- | --- | --- |
| **Item** | **URI** | **Parameters** |
| API -> Database | GET http://localhost:8080/demo/all |  |
| API -> Database | POST http://localhost:8080/demo/app | ID |
| Timestamp |

## How to Run the API

The API is built to be executable on your device if the following pre-requisites are met.

* Nothing running on the port 8080 as this is the port where the API is being executed.
* A database built in your local environment whose details are included in the application.properties file of the code

To run the API a user can either use a Postman or a curl command

## Postman

To execute via postman the user needs to make one of the calls mentioned above in the URI section of the design.

To post the data to the database the user needs to mention two columns in the param section of the API. An example for the same is attached below:

Graphical user interface, text, application, email

Description automatically generated

## Curl

To use the curl command the user can execute the below command:

curl -X POST <http://localhost:8080/demo/app?id=2&timeStamp=2021-11-30>

Text

Description automatically generated

## MSSQL

Below screenshot shows the data in the database

Graphical user interface, text

Description automatically generated

Proof of concept along with ReadMe document capturing all the relevant steps to be followed for execution in uploaded successfully in the below Github repository

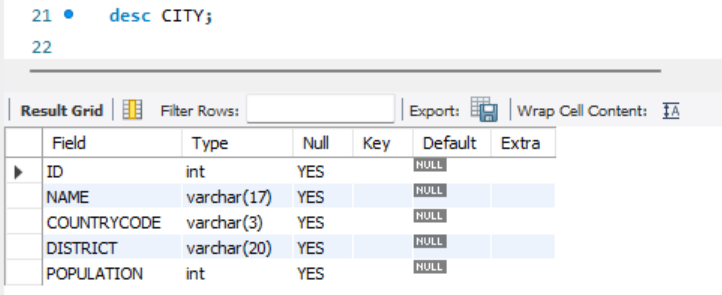
Github repository:

<https://github.com/chethuaries19/pollinate.git>

# Section 2

## Query all columns for a city in CITY with the *ID* 1661.

Created table with name City and Attributes, datatypes as mentioned



Inserted below values into the table

Graphical user interface, text, application

Description automatically generated

Query to fetch all the columns of city with ID=1661

Graphical user interface, text, application, email

Description automatically generated

## Write a query that prints a list of employee names (i.e.: the *name* attribute) for employees in Employee having a salary greater than per month who have been employees for less than months. Sort your result by ascending *employee\_id*

Created table with name EMPLOYEE and Attributes, datatypes as mentioned

Graphical user interface, application

Description automatically generated

Inserted below values into the table

Table

Description automatically generated with low confidence

To print a list of employee names (i.e.: the *name* attribute) for employees in Employee having a salary greater than per month who have been employees for less than months

/TO Print EMPLOYEE names WITH SALARY >some value AND MONTHS< some value- have considered salary>2000 and Months<10 as per the output mentioned in the question sheet

Graphical user interface, text, application, email

Description automatically generated

## 3.1. Query an *alphabetically ordered* list of all names in OCCUPATIONS, immediately followed by the first letter of each profession as a parenthetical (i.e.: enclosed in parentheses). For example: AnActorName(A), ADoctorName(D), AProfessorName(P), and ASingerName(S).

Created table with name OCCUPATIONS and Attributes, datatypes as mentioned

Graphical user interface, text, application

Description automatically generated

Inserted below values into the table

Graphical user interface

Description automatically generated

/\*Query to order the name alphabetically in specified format\*/ Graphical user interface, text, application

Description automatically generated

## 3.2. Query the number of ocurrences of each occupation in OCCUPATIONS. Sort the occurrences in *ascending order*, and output them in the following format:



where [occupation\_count] is the number of occurrences of an occupation in OCCUPATIONS and [occupation] is the *lowercase* occupation name. If more than one *Occupation* has the same [occupation\_count], they should be ordered alphabetically.

Graphical user interface, text, application, email

Description automatically generated

## 4. *P(R)* represents a pattern drawn by Julia in *R* rows. The following pattern represents *P(5)*:

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

Write a query to print the pattern *P(20)*.

Procedure to create the above pattern

A picture containing table

Description automatically generated

Or by using information\_schema.Tables

Graphical user interface, text

Description automatically generated

## *5. P(R)* represents a pattern drawn by Julia in *R* rows. The following pattern represents *P(5)*:

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

Write a query to print the pattern *P(20)*.

Procedure to create the above pattern

Table

Description automatically generated

Or by using information\_schema.Tables

Text

Description automatically generated with medium confidence