Database Setup

TestcontainerGreetingsApplication startup will instantiate all containers including a new container from image *postgres:latest.* The databases external port is automatically map to the containers internal Postgres standard port (5432).

The shared external port is different each time and available either in Docker Desktop > Containers view under Ports columns or will be included in the container list under PORTS column when printed from `docker ps` command.

cmd: `docker ps`

```
CONTAINER ID   IMAGE        STATUS          PORTS
eba22c98d081   postgres:latest   Up 18 minutes   0.0.0.0:49842->5432/tcp
```

Testcontainers' *Greeting API Service* already has the port mapping wired into its environment so noting the shared port is mainly for external debugging or shell scripting outside of any request to a Greetings Service endpoint.

Database Properties (./src/main/resources/application.properties)

Testcontainers' configures *postgres:latest* with the following environmental properties to determine behavior and capablities. Properties defined in `application.properties` file are pulled in at startup, before *run* command is issued.

Platform Specific: On MacOS, it was necessary to include the *update* dynamic linking libraries in order to change the initial database structure. Namely automatically adding the table 'greeted' was an issue.

```
spring.jpa.hibernate.ddl-auto=update
spring.jpa.generate-ddl=true

spring.datasource.driverClassName=org.postgresql.Driver
spring.datasource.url=jdbc:postgres://localhost:5432/test
spring.datasource.username=test
spring.datasource.password=test
spring.jpa.show-sql=true
```

Note `spring.datasource.*` defines the same *database, user* and *password* 'test'. This is for convenience and doesn't have to be the case. But for this project, 'test' is used for quick deployment.

Confirm JDBC Driver : Postgres Daemon Startup

The following output confirms an instances of *postgres:latest* is up running and ready for request.

```
[greetings] [main] tc.postgres:latest : Creating container for image: postgres:latest
[greetings] [main] tc.postgres:latest : Container postgres:latest started in PT1.946065S
[greetings] [main] tc.postgres:latest : Container is started
                (JDBC URL: jdbc:postgresql://127.0.0.1:49842/test?loggerLevel=OFF)
```

Note: *jdbc:postgrest* is listening on it local loopback for 'test' request.

Greetings API Service Request

The following are two brief example of a POST & GET request/responses that's tested and included in this project. The example uses cURL, but endpoints be called from a web UI using AJAX or some other RESTful API mechanism.

NOTE: DB_PORT for public endpoints should be '8080'. *Spring* - *Testcontainers* handles the internal route and mapping request for backend capabilities.

Create Greeted Name
Endpoint: /api/greetings

Expects request *data{ }* to include one parameter, username with the value of the new user *name*. A request is passed to the internal service manager (prefixed with Default*) which takes the raw data and formats it into a *RequestEntity* that *JpaRepository* uses to perform some CRUD operations.

cmd:
```
curl -v -X "POST" -H "Content-Type: application/json" -d
'{"username":"Victor"}' "http://localhost:${DB_PORT}/api/greetings"
```

The name *Victor* will be inserted at the next record with a sequential 'id' automatically included.
  Table: greeted
  Record: id: 1, username: Victor

The request/response chain of responsibility:

```
 GreetingController.createGreeting( request )          // handled inout request, response
   -> CreateGreetingRequest request                    // defines record properties and attributes
       -> GreetingService.createGreeting(request)      // service interface with required method
           -> DefaultGreetingService.createGreeting(request)    // internal service operations
   <- returns response status (OK) and code 201 - created
```

Under the hood the transaction is captured in the IDE debug log:

```
    GC.createGreeting working...
    Greeted setting username with Victor
        Hibernate: select nextval('greeted_seq')
        Hibernate: insert into greeted (username,id) values (?,?)
    - created new record for Victor
```

Get Existing Greeted Name
Endpoint: /api/greetings/{username}

This GET request returns the Greeted record *(greeted.username, greeted.id)* if the name already exists. *Greetings Service* calls GreetingController.responseGreeting(*username*) when a URI request parameter is detected.

The response can be the returned *name* indicating it already exists, or Empty Row when no record was found. If not found, logic could be added to re-route the request to *createGreeting(request)* in order to create a new greeted name.

cmd: `curl -X "GET" "http://localhost:${DB_PORT}/api/greetings/Victor"`
found: GC found name Victor in db!
empty: curl: (52) Empty reply from server

When the user name isn't in the database, it can be safely inserted as a unique value in the database.

Under the hood the transaction is captured in the IDE debug log:

```
GC query username:Victor
GC.responseGreeting( ) working...
    Hibernate: select g1_0.id,g1_0.username from greeted g1_0 where g1_0.username=?
    Greeted initialized! - user name is in repository! With record id
    [ com.demo.greetings.domain.internal.Greeted@328b336c ]
- GC found name Victor in db!
```

Troubleshooting Database Containers

If after Testcontainers (including postgres:latest) has successfully started and are available, but a GET or POST request returns an error, the following steps can help identify the problem.

1. Confirm *postgres:lastest* is up, running and accepting request. There are two options.

a. from Docker Desktop > Containers list, click the postgrest:latest container name to see a detailed view of the running container - including the startup log. Look for the following:

```
date / time UTC [1] LOG:  database system is ready to accept connections
```

b. use `psql` to query the any database on the server. If the server does not accepted connection ('could not connect') scan the IDE Build/Run log for any errors or warnings related to the configuration and confirm application.properties has the correct directives as listed in the Application Properties section (above).

cmd: `PGPASSWORD=test psql -h $DB_HOST -U $DB_USER -p $DB_PORT -c "select 1"`

2. Confirm *postgres:latest* has the expected database, schema, and table. This should be the case if the process picked up the V1_greeted_tables.sql under the project resources/db/migration/ directory. If not, it may be necessary to manually create the table on the first install.

cmd:
```
PGPASSWORD=test psql -h $DB_HOST -U $DB_USER -p $DB_PORT -d $DB_BASE \
-c "select table_schema, table_name from information_schema.tables where \
table_schema in ('public');"
```
output:
```
    table_schema |  table_name
    --------------+-----------------------
    public       | flyway_schema_history
    public       | greeted
    (2 rows)
```

3. OpenSSL Issue on POST process throws an *OpenSSL SSL_connect: SSL_ERROR_SYSCALL* error, it means that the container (or cluster) doesn't have the required SSL certificate to validate the inbound connection from the *host:port*.

If cURL request included the -v verbose flag (-v), the following output provides more context.
```
    * Connected to localhost (::1) port 49842
    * ALPN: curl offers h2,http/1.1
    * (304) (OUT), TLS handshake, Client hello (1):
    *  CAfile: /usr/local/etc/openssl/cert.pem
    *  CApath: none
    * LibreSSL SSL_connect: SSL_ERROR_SYSCALL in connection to localhost:49842
    * Closing connection
```

a. Simple option and the one used for this project is to edit the file /HOME/.testcontainers.properties by adding the line `checks.disable=true`

b. Alternatively, the following instructions step through adding a certificate for localhost. The problem here though is that the port changes with each new instance. so that certificate would have to be recreated and pushed each time. But for completeness, there are the steps.

[Done]