Step-by-Step Skeleton to Create SpringBoot project

# 

# Step 1: Generate Spring (<https://start.spring.io/>)

* Select the following variables:
  + Project: Maven
  + Language: Java
  + Spring Boot: 3.4.0
  + Project Metadata
    - Edit <Group> and <Name> accordingly
    - <Packaging>: Jar
    - <Java>: Select Version
  + Dependencies:
    - Spring Boot DevTools: always install
    - Spring Web: always install
    - Thymeleaf: always install
    - Validation: install if require validation
    - Spring Data Redis (Access+Driver): install if require Redis

# 

# Step 2: Add necessary dependencies to pom.xml

* Obtain from <https://mvnrepository.com/>
* Commonly used:
  + Json P (if you using some Json thing)
    - Usually select Json P Default Provider (<https://mvnrepository.com/artifact/org.glassfish/jakarta.json/2.0.1>)
  + Jedis (if you are linking to Redis DB - this is needed on top of the Redis dependency installed in Step 1)
    - Select Jedis (<https://mvnrepository.com/artifact/redis.clients/jedis/5.2.0>)
* Copy the Maven code into pom.xml in between <dependencies>:
  + E.g.

<!-- https://mvnrepository.com/artifact/redis.clients/jedis -->

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>5.2.0</version>

</dependency>

# Step 3: Add Redis Configuration.java Source Code (For Redis Template)

* Copy boilerplate code for Redis Configuration [i.e. code is always the same] if you are using Redis database

# Step 4: Edit application.properties

* Purpose: to add the ENV variables for 1) Redis (e.g. host, port, database, username, password) and/or 2) API Keys and/or 3) other env variables that you need

# Step 5: Code …

# Step 6: Try Program with Local Redis (if using Redis)

# Step 7: Try Program with cloud Redis

* Either with 1) Redis from Railway or 2) Redis Cloud
* Tips:

1. Set ENV variables to match that of cloud Redis

* Windows:
  + set SPRING\_DATA\_REDIS\_HOST=...
  + set SPRING\_DATA\_REDIS\_PORT=...
  + set SPRING\_DATA\_REDIS\_USERNAME=...
  + set SPRING\_DATA\_REIDS\_PASSWORD=...
    - Note:
      * env variable all uppercase, “.” change to “\_”
      * To see what is currently set in the terminal, use echo %<ENV\_VARIABLE\_NAME>%
* Mac:
  + export SPRING\_DATA\_REDIS\_HOST=

1. To access cloud Redis in your terminal, do

* redis-cli -u redis://default:xxxx@autorack.proxy.rlwy.net:41033
  + Note: part in red is obtained from the cloud Redis
    - For Railway Redis, it is obtained under “variable” → “redis\_public\_url”
  + Note: will not work if you are on NUS\_WIFI (public WIFI) as it is unsecured and railway/redis will not let you view the cloud Redis

# Step 8: Create Single or Multi-Stage Docker and Dockerize locally

* Create the Docker File in your root directory (same folder as pom.xml)
* Obtain base-image from Dockerhub (<https://hub.docker.com/>) → search
  + Java:
    - search “java”
    - under “deprecation notice” select either one (e.g. eclipse-temurin)
    - go to “tags” (beside “overview”)
    - filter “tags” by the java version needed (e.g. 23)
    - choose the one with jdk (instead of jre) and correct system (e.g. windows, linux, amd (for mac))
    - copy the code on the left side (e.g. eclipse-temurin:23.0.1\_11-jdk)
* Get jar file name by creating the jar file locally
  + CMD for windows:
    - mvn package -Dmaven.test.skip=true
  + Go to target file to find the jar file created, and place the jar file name into the ENTRYPOINT (refer to table below)

| **Single-Stage** | **Multi-Stage** |
| --- | --- |
| From eclipse-temurin:23.0.1\_11-jdk  # Optional, just creating a folder for the compiled files to be in  WORKDIR /myapp  # Copy necessary files (folders need to give name, individual files can just do “.” to use the same name)  COPY src src  COPY .mvn .mvn  COPY pom.xml .  COPY mvnw .  RUN chmod a+x ./mvnw && ./mvnw package -Dmaven.test.skip=true  # Set the environment variables  ENV SPRING\_DATA\_REDIS\_HOST=localhost  ENV SPRING\_DATA\_REDIS\_PORT=6379  ENV SPRING\_DATA\_REIDS\_USERNAME=  ENV SPRING\_DATA\_REDIS\_PASSWORD=  ENV PORT=5000  # or ENV SERVER\_PORT=5000 (if not on Railway)  EXPOSE ${PORT}  # or EXPOSE ${SERVER\_PORT}  ENTRYPOINT SERVER\_PORT=${PORT} java -jar target/**<jar file name>** | # builder can be any name  From eclipse-temurin:23.0.1\_11-jdk **AS builder**  # Optional, just creating a folder for the compiled files to be in  WORKDIR /compiledDir  COPY src src  COPY .mvn .mvn  COPY pom.xml .  COPY mvnw .  RUN chmod a+x ./mvnw && ./mvnw package -Dmaven.test.skip=true  # Doing MultiStage, do not need ENTRYPOINT  FROM eclipse-temurin:23.0.1\_11-jdk  WORKDIR /myapp  COPY --from=builder /compiledDir/target/**<jar file name of compiled jar>** **<jar name that you want it to be>**  # E.g. COPY --from=builder /compiledDir/target/day18workshopWordDoc-0.0.1-SNAPSHOT.jar day18workshopWordDoc.jar  # Set the environment variables  ENV SPRING\_DATA\_REDIS\_HOST=localhost  ENV SPRING\_DATA\_REDIS\_PORT=6379  ENV SPRING\_DATA\_REIDS\_USERNAME=  ENV SPRING\_DATA\_REDIS\_PASSWORD=  ENV PORT=5000  ENV PORT=3000  EXPOSE ${PORT}  # or EXPOSE ${SERVER\_PORT}  ENTRYPOINT java -jar **<jar name that you want it to be>** |

* Note: no “space” between the ENV VAR and the “=” and the VALUE
* Build the image locally
  + docker image build -t **<username>**/**<image\_name>**:v1.0.0 .
* Run the container (with the ENV variables set if needed)
  + docker run -d -p 12345:5000 -e SERVER\_PORT=3000 -e SPRING\_DATA\_REDIS\_HOST=localhost.net
    - Note:
      * 12345 is the port that will be used to access in the future (i.e. localhost:12345)
      * 5000 is the port that you configured in the Dockerfile or your application.properties
* Access with the port set (e.g. 12345 in the above example)

# Step 9: Link to Railway (with cloud Redis + Docker)

* Create “empty service” in Railway
* Set the ENV Variables in Railway service (if needed)
  + E.g. Redis configuration (host, port, password, username, database)
* CMD prompt:
  + railway login
  + railway link
  + railway up

Commonly Used Code Chunks

# Logger

Logger logger = Logger.getLogger(**<Class\_Name>**.class.getName());

logger.info(“message”);

logger.warning(“message”);

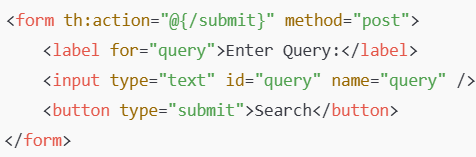
logger.severe(“message”);

# Thymeleaf (Various codes)

* th:if & th:unless (conditionals)
  + lt (less than)
  + le (less than or equals to)
  + gt (greater than)
  + ge (greater than or equals to)
  + Example:
    - th:if=”${currInt le 12}” → display only if currInt is less than or equals to 12
* th:text
  + To display dynamic text
* th:each=” indiv\_item : ${collection\_name}” (iteration)
  + Works for Array, List, Map.Entry (for Maps)
  + Possible information from iteration
    - Index - the current index
    - Count - collection size
    - Odd, even, first, last - boolean properties
* th:action or th:href (for generating URL)
  + Use link expression
  + **th:href=”@{**/weather/**{city}(city = ${java\_city}, units=${java\_units})}”**
    - units=${java\_units} is going to be displayed as query strings (i.e. /weather/?units=metrics) as it does not match any path variable
* Others

| **Binding Errors**  th:if=”${#fields.hasErrors('<'fieldName'>')}”  #fields.errors('fieldName') ← this can be iterated (i.e. th:each “err : #fields.errors(‘fieldname’);  #fields.hasGlobalErrors()  #fields.globalErrors() |
| --- |
| **List**  th:if="${#lists.isEmpty(items)}"  #lists.size(list)  #lists.contains(list, element) |
| **Map**  th:if="${#maps.isEmpty(map)}"  #maps.containsKey(map, key)  #maps.containsValue(map, value)  #maps.size(map) |
| **Array**  #arrays.isEmpty(array)  #arrays.size(array)  #arrays.contains(array, element) |
| **String**  #strings.isEmpty(str)  #strings.contains(str, 'substring') |

* Creating dynamic URL based on User-input in the form
  + Requires backend processing (can’t do it directly in thymeleaf in the form
  + Technically in this case, th:action=”@(/submit)” not needed → can use action=”/submit”

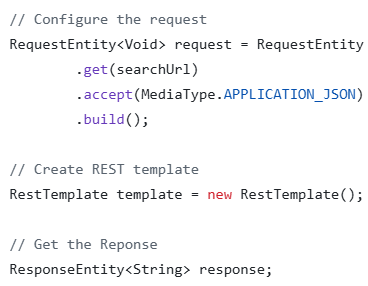




# Sending Request to API

* Configure the request with RequestEntity<(type\_of\_content\_of\_payload)>
  + If <String>, indicates payload is Json. <Void> means no payload
* Create REST template with RestTemplate template = new RestTemplate();
* Get the response with template.exchange(request\_entity, (class\_of\_expected\_response))
  + E.g. String.class if we are expecting a Json or String response

1. Get Request (.get + RequestEntity<Void>)







1. Post Request (.post + RequestEntity<String>)



# Redis Commands (in CMD and in Java)

| **Operation** | **In CMD (Terminal)** | **In Java** |
| --- | --- | --- |
| **Value Operations** | | |
| Get value | GET key | template.opsForValue().get(“key”); |
| Set value | SET key value | template.opsForValue().set(“key”, “value”); |
| Delete key | DEL key | template.delete(“key”); |
| Check if key exists | EXISTS key | template.hasKey(“key”); |
| Get all keys | keys \* | template.keys(“\*”); |
| Increment value | INCR key | template.opsForValue().increment(“key”); |
| Decrement value | DECR key | template.opsForValue().decrement(“key”); |
| Append to string (appends value to existing value of key if value is string) | APPEND key value | template.opsForValue().append(“key”, “value”); |
| Expire key | EXPIRE key seconds | template.expire(“key”, Duration.ofSeconds(seconds)); |
| Returns the expiration time in seconds of a key | TTL key | template.getExpire(“key”); |
| **Hash Operations** | | |
| Set hash field | HSET key field value | template.opsForHash().put(“key”, “field”, “value”); |
| Get hash field | HGET key field | template.opsForHash().get(“key”, “field”); |
| Delete hash field | HDEL key field | template.opsForHash().delete(“key”, “field”); |
| Get all fields | HGETALL key | template.opsForHash().entries(“key”); |
| Check hash field | HEXISTS key field | template.opsForHash().hasKey(“key”, “field”); |
| Increment hash field | HINCRBYkey field value | template.opsForHash().increment(“key”, “field”, “value”); |
| **List Operations** | | |
| Push to List (Left) | LPUSH key value | template.opsForList().leftPush(“key”, “value”); |
| Push to List (Right) | RPUSH key value | template.opsForList().rightPush(“key”, “value”); |
| Pop from List (Left) | LPOP key | template.opsForList().leftPop(“key”); |
| Pop from List (Right) | RPOP key | template.opsForList().rightPop(“key”); |
| Get List Range | LRANGE key start end | template.opsForList().range(“key”, start, end); |
| ^ Get all items in list | LRANGE key 0 -1 | template.opsForList().range(“key”, 0, -1); |
| Get List Length | LLEN key | template.opsForList().size(“key”); |
| **Set Operations** | | |
| Add to Set | SADD key value | template.opsForSet().add(“key”, “value”); |
| Remove from Set | SREM key value | template.opsForSet().remove(“key”, “value”); |
| Get all members | SMEMBERS key | template.opsForSet().members(“key”); |
| Check member | SISMEMBER key value | template.opsForSet().isMember(“key”, “value”); |
| Get Set Size | SCARD key | template.opsForSet().size(“key”); |