MONASH University

FIT5042 Studio 10

Spring boot and Docker

Last updated: 15 Oct 2020

OBJECTIVES

- Deploy a spring boot restful web service
- Understand CRUD operations
- Understand docker-compose
- Test the API's using postman

INSTRUCTIONS

- You are encouraged to use your own laptop or device to do the tasks
- If you do not bring your own device, feel free to use the lab machine.

EFOLIO TASK 10

DESCRIPTION:

- Get familiar with the IntelliJ and Docker
- Understand the SpringBoot framework

WHAT TO SUBMIT (INDIVIDUAL):

- Share and include eFolio URL

EVALUATION CRITERIA:

- Pokemon RESTful service is working properly
- Tidy and structured code
- The result is showing as expected

MONASH University

STUDIO ACTIVITIES

Task 1.1 Docker basics

Environment Setup

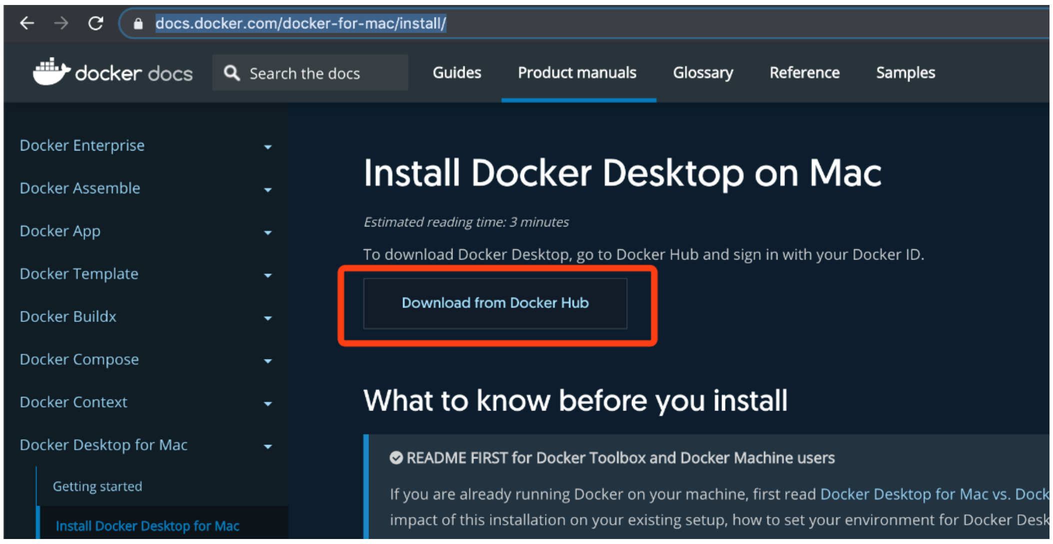
I. Before you start doing this task, you need to install some software and packages to make the environment work for the project. Here, we need Docker, Postman, IntelliJ and Maven. To install the Docker (a set of PAAS products that use virtualisation to deliver software in packages – called **containers**, it allows developers to ship all libraries and dependencies with the application within one package to the client or other teams without configuring it for different environments)

- if you are a Mac user, please go to:

  https://docs.docker.com/docker-for-mac/install/

- if you are a Windows (Pro or Enterprise version) user, please go to:

  https://docs.docker.com/docker-for-windows/install/



Install Docker Desktop on Mac

Estimated reading time: 3 minutes

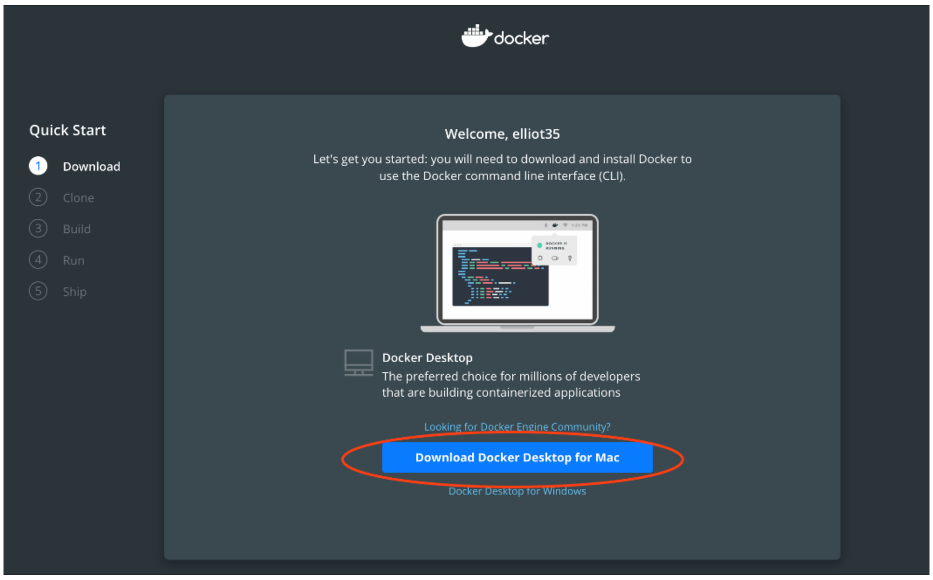
To download Docker Desktop, go to Docker Hub and sign in with your Docker ID.

What to know before you install

README FIRST for Docker Toolbox and Docker Machine users

If you are already running Docker on your machine, first read Docker Desktop for Mac vs. Dock
impact of this installation on your existing setup, how to set your environment for Docker Desk

You might need to login or sign up for downloading Docker, please use your Monash email to sign up (if you haven't been a member of docker) or login (if you've already had an account). After login, you should be able to see the web page as showing below, just click "**Download Docker Desktop for Mac/Windows**" to download the installation package.


docker
Quick Start
1 Download
2 Clone
3 Build
4 Run
5 Ship
Looking for Docker Engine Community?
Download Docker Desktop for Mac
Docker Desktop for Windows

Welcome, elliot35

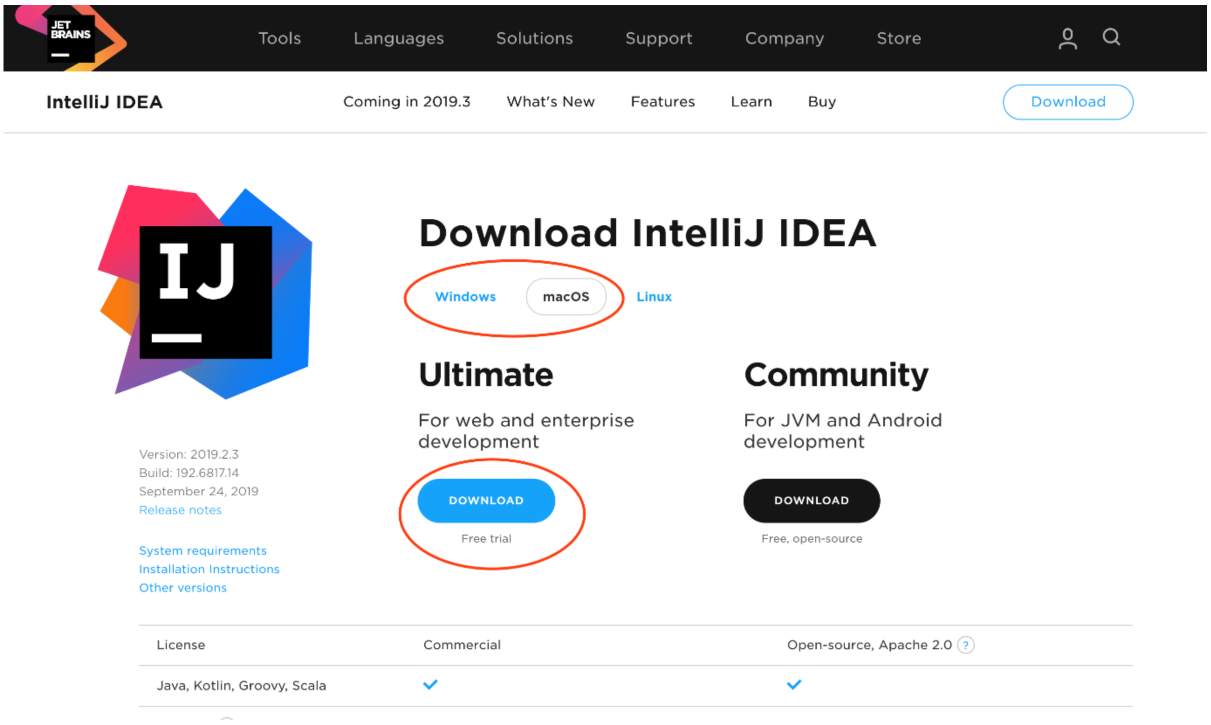
Let's get you started: you will need to download and install Docker to use the Docker command line interface (CLI).

Docker Desktop

The preferred choice for millions of developers that are building containerized applications

II. Download and install the IntelliJ which we are going to use it as our IDE in this case, you can use your Monash account to get access to the Ultimate version.

- if you are a Mac user, please go to: *https://www.jetbrains.com/idea/download/#section=mac*
- if you are a Windows user, please go to: *https://www.jetbrains.com/idea/download/#section=windows*



Download IntelliJ IDEA

Windows macOS Linux

Ultimate	Community
For web and enterprise development	For JVM and Android development
DOWNLOAD	DOWNLOAD
Free trial	Free, open-source

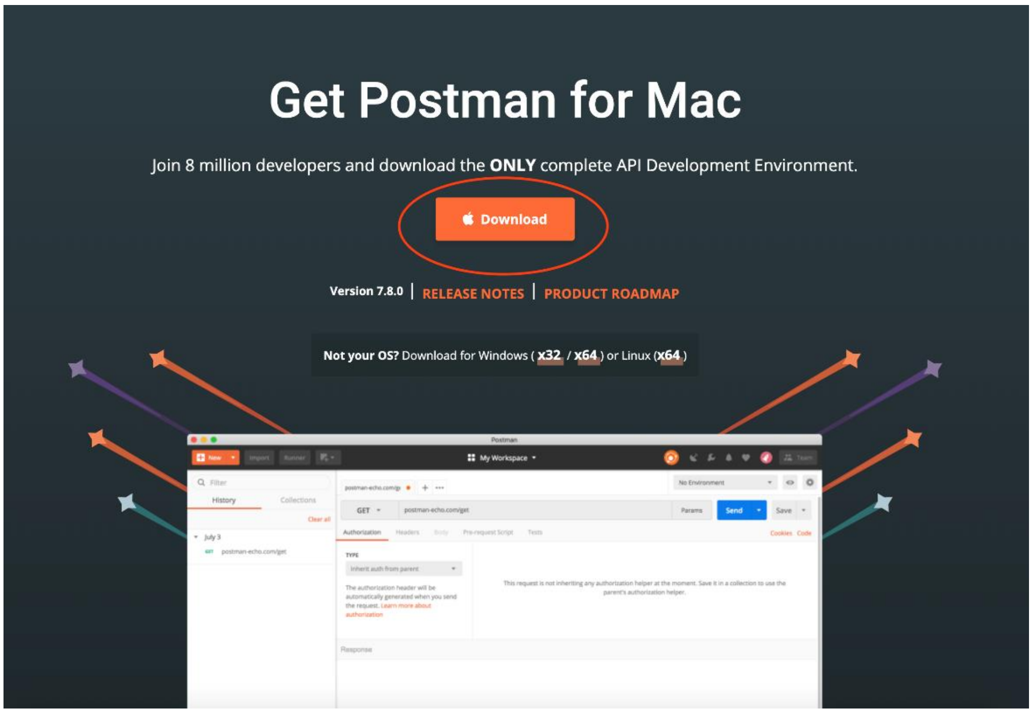
Version: 2019.2.3
Build: 192.6817.14
September 24, 2019
Release notes

System requirements
Installation Instructions
Other versions

License	Commercial	Open-source, Apache 2.0 ?
Java, Kotlin, Groovy, Scala	✓	✓

III. Download and install POSTMAN which we will be using it to test our API

- please go to: *https://www.getpostman.com/downloads/*



Get Postman for Mac

Join 8 million developers and download the **ONLY** complete API Development Environment.

Download

Version 7.8.0 | RELEASE NOTES | PRODUCT ROADMAP

Not your OS? Download for Windows (**x32** / **x64**) or Linux (**x64**)

IV. Since the application that we are going to create is a Java-Maven project, it would be easier to handle if Maven is installed on the computer. To install maven,

- if you are a Mac user, we need to use a brew (package manager) command to install the Maven, if you haven't got brew in your system, please install Homebrew by typing:
    - Once you have the brew, in your terminal, please type:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

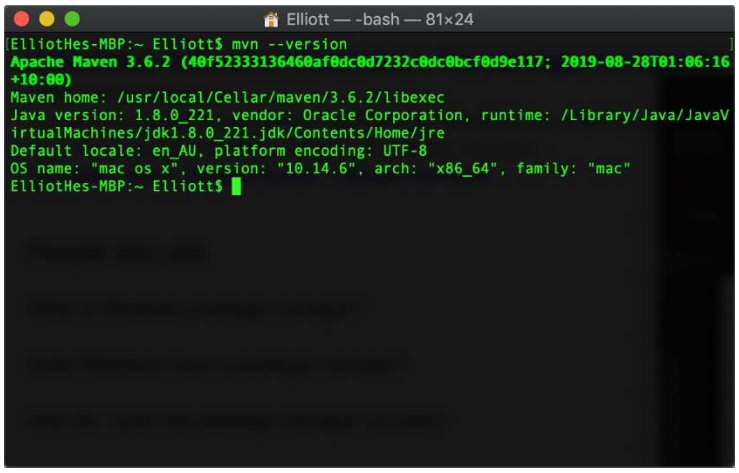
brew install maven

- if you are a Windows user, you need to find the documentation of your own package manager to install the maven packages in your system

After you installed maven, type:

mvn --version or ***mvn -v***

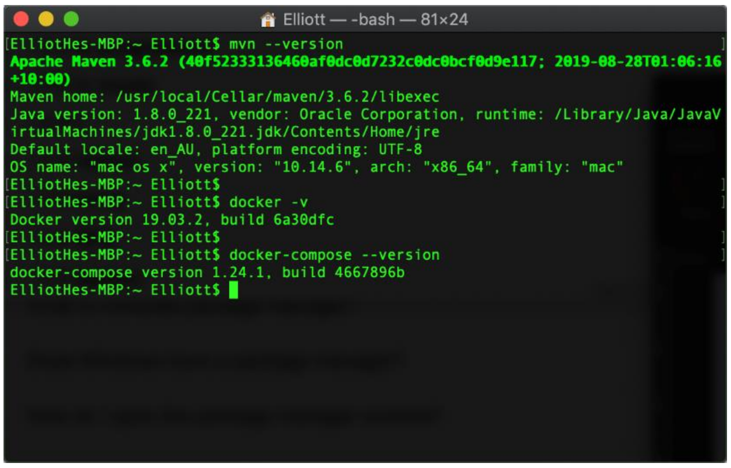
to check whether your maven has been successfully installed. You should be able to see the result showing as below in your terminal once you've done this process.



```
Elliott — -bash — 81×24
ElliotHes-MBP:~ Elliott$ mvn --version
Apache Maven 3.6.2 (40f52333136460af0dc0d7232c0dc0bcf0d9e117; 2019-08-28T01:06:16
+10:00)
Maven home: /usr/local/Cellar/maven/3.6.2/libexec
Java version: 1.8.0_221, vendor: Oracle Corporation, runtime: /Library/Java/JavaV
irtualMachines/jdk1.8.0_221.jdk/Contents/Home/jre
Default locale: en_AU, platform encoding: UTF-8
OS name: "mac os x", version: "10.14.6", arch: "x86_64", family: "mac"
ElliotHes-MBP:~ Elliott$
```

V. Before we create the project, make sure our environment has been correctly set up, in your terminal, type:

- ***mvn --version*** to check your maven has been installed
- ***docker -v*** to check your docker has been installed
- ***docker-compose --version*** to check your docker-compose has been installed

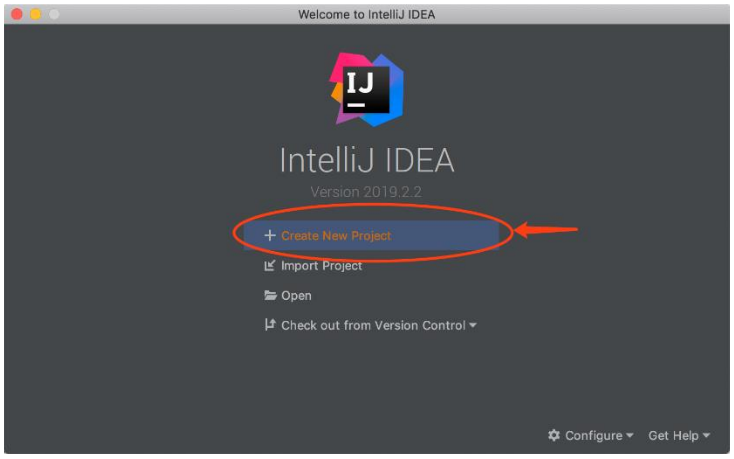


```
Elliott — -bash — 81×24
ElliotHes-MBP:~ Elliott$ mvn --version
Apache Maven 3.6.2 (40f52333136460af0dc0d7232c0dc0bcf0d9e117; 2019-08-28T01:06:16
+10:00)
Maven home: /usr/local/Cellar/maven/3.6.2/libexec
Java version: 1.8.0_221, vendor: Oracle Corporation, runtime: /Library/Java/JavaV
irtualMachines/jdk1.8.0_221.jdk/Contents/Home/jre
Default locale: en_AU, platform encoding: UTF-8
OS name: "mac os x", version: "10.14.6", arch: "x86_64", family: "mac"
ElliotHes-MBP:~ Elliott$
ElliotHes-MBP:~ Elliott$ docker -v
Docker version 19.03.2, build 6a30dfc
ElliotHes-MBP:~ Elliott$
ElliotHes-MBP:~ Elliott$ docker-compose --version
docker-compose version 1.24.1, build 4667896b
ElliotHes-MBP:~ Elliott$
```

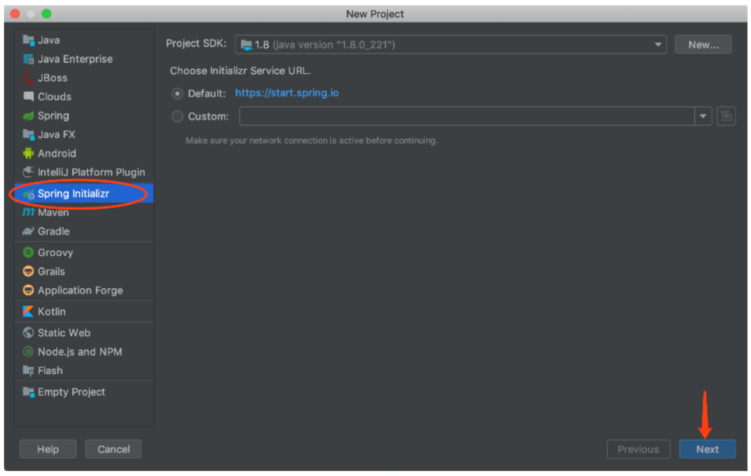
If you can see the same result as I do, you are ready to go!

Use SpringBoot to create Pokémon API from scratch

1. Open your IntelliJ IDEA, click "Create New Project":



2. Next, choose "Spring Initializr" since we are going to create a SpringBoot application.



Project SDK: 1.8 (java version "1.8.0_221")

Choose Initializr Service URL.

Default: https://start.spring.io

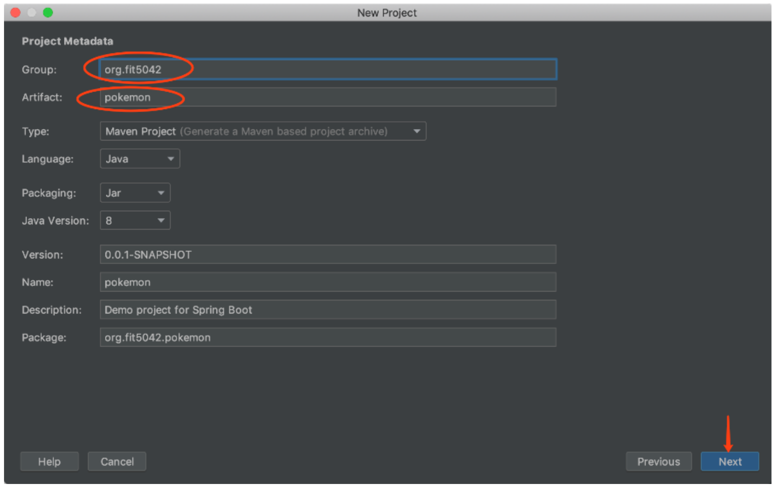
Custom:

Make sure your network connection is active before continuing.

3. Then, in the Project Metadata section, let's change our

- "Group" to "org.fit5042"
- "Artifact" to "pokemon"

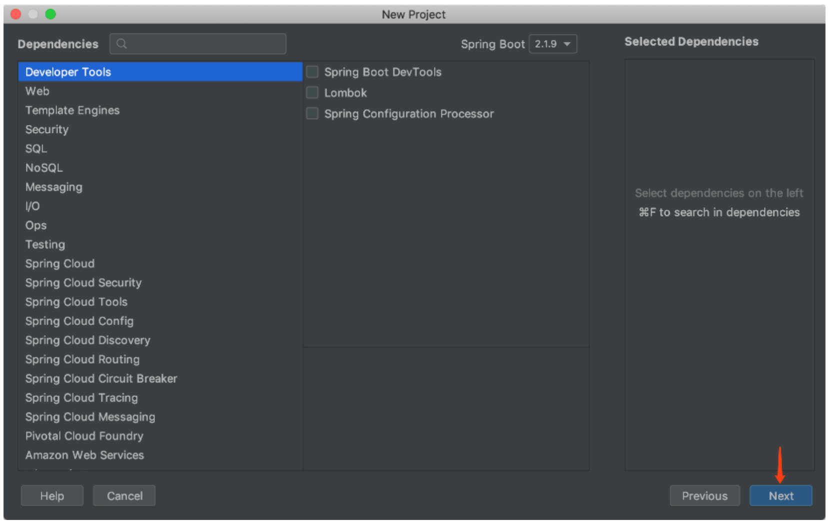
Here, you can consider the "Group" as the package root, and the "Artifact" represents project id.



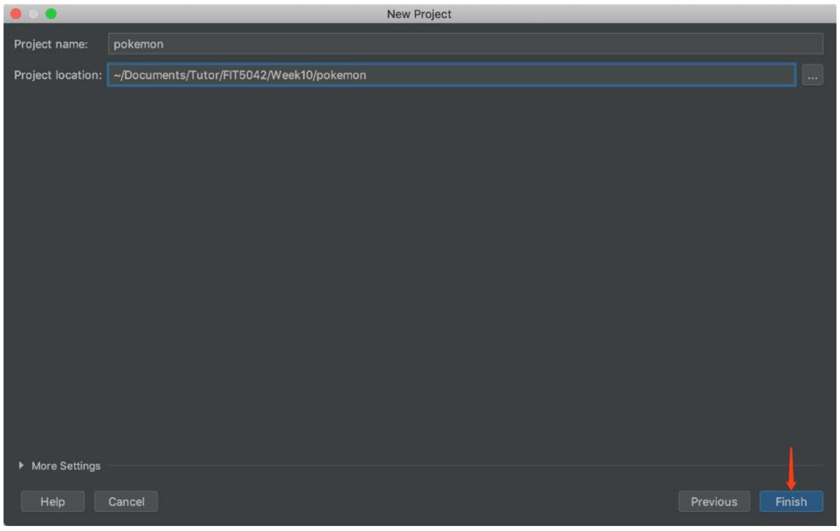
Project Metadata

Group:	org.fit5042
Artifact:	pokemon
Type:	Maven Project (Generate a Maven based project archive)
Language:	Java
Packaging:	Jar
Java Version:	8
Version:	0.0.1-SNAPSHOT
Name:	pokemon
Description:	Demo project for Spring Boot
Package:	org.fit5042.pokemon

4. Leave the dependencies section as default since we will install all the dependencies after some further configurations.



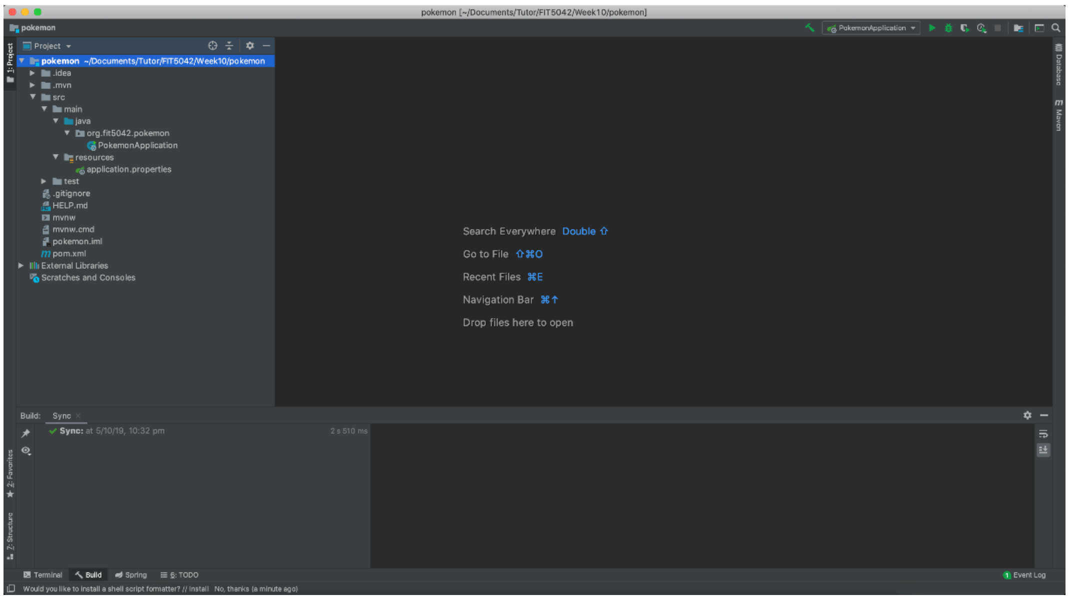
5. Choose a proper location to be your project folder and click "Finish"


New Project

Project name: pokemon

Project location: ~/Documents/Tutor/FIT5042/Week10/pokemon

6. After we created the project, we should be able to see the window showing as below:


pokemon [~/Documents/Tutor/FIT5042/Week10/pokemon]

```
pokemon ~/Documents/Tutor/FIT5042/Week10/pokemon
  .idea
  .mvn
  src
    main
      java
        org.fit5042.pokemon
          PokemonApplication
      resources
        application.properties
    test
  .gitignore
  HELP.md
  mvnw
  mvnw.cmd
  pokemon.iml
  pom.xml
External Libraries
Scratches and Consoles
```

7. Let's first modify the *pom.xml* file. All the dependencies, project details and properties will be declared in this file.

Add these two properties within the *properties* tag

```
<properties>

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

   <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

   <java.version>1.8</java.version>

</properties>
```

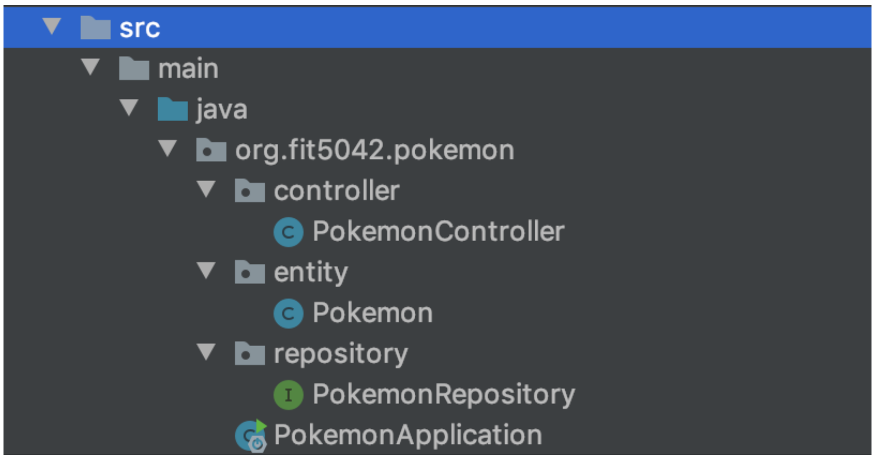
Add the Spring Data JPA dependency, Spring Web dependency, and MySQL JDBC connector dependency within the *dependencies* tag since these are the required dependencies

```
<dependency>

   <groupId>org.springframework.boot</groupId>

   <artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>



<dependency>

   <groupId>org.springframework.boot</groupId>

   <artifactId>spring-boot-starter-web</artifactId>

</dependency>



<dependency>

   <groupId>mysql</groupId>

   <artifactId>mysql-connector-java</artifactId>

   <scope>runtime</scope>

</dependency>
```

8. Next, under */src/main/java/org.fit5042.pokemon* folder, let's create some necessary packages and classes.

- create a *PokemonController* Java Class under *controller* package
- create a *Pokemon* Java Class under *entity* package
- create a *PokemonRepository* Java Interface under *repository* package

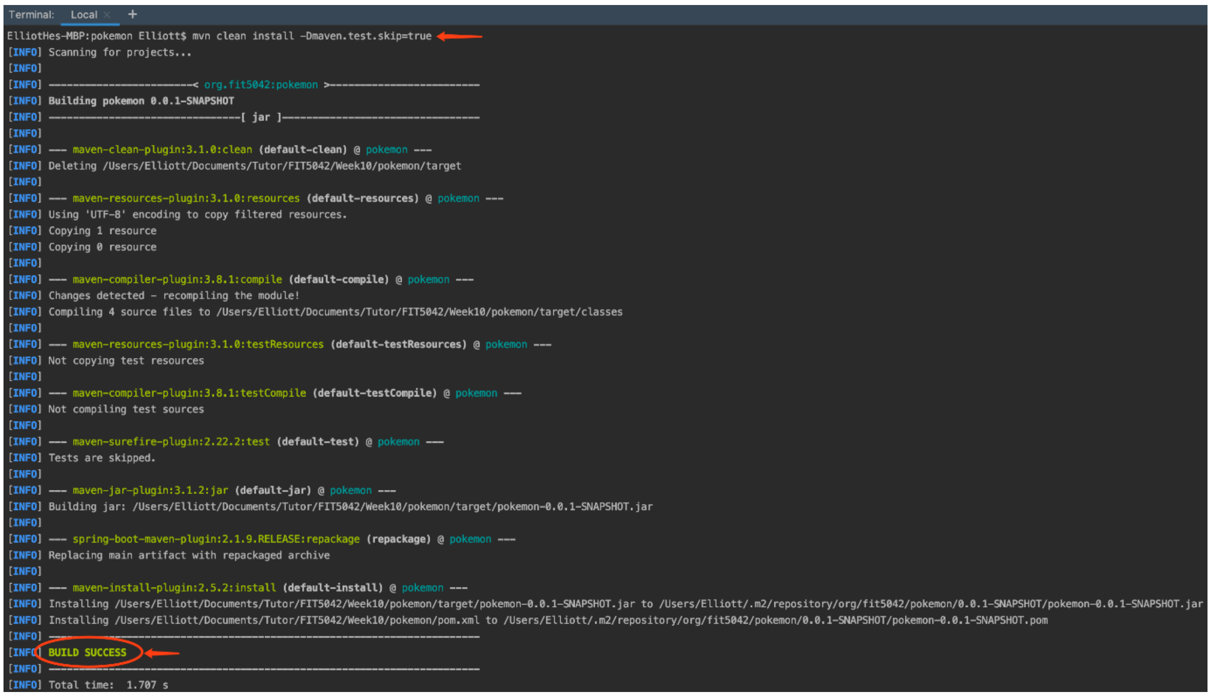
After you created these classes and packages, you should be able to see the structure as showing in the snapshot



```
▼ src
    ▼ main
        ▼ java
            ▼ org.fit5042.pokemon
                ▼ controller
                      PokemonController
                ▼ entity
                      Pokemon
                ▼ repository
                      PokemonRepository
                  PokemonApplication
```

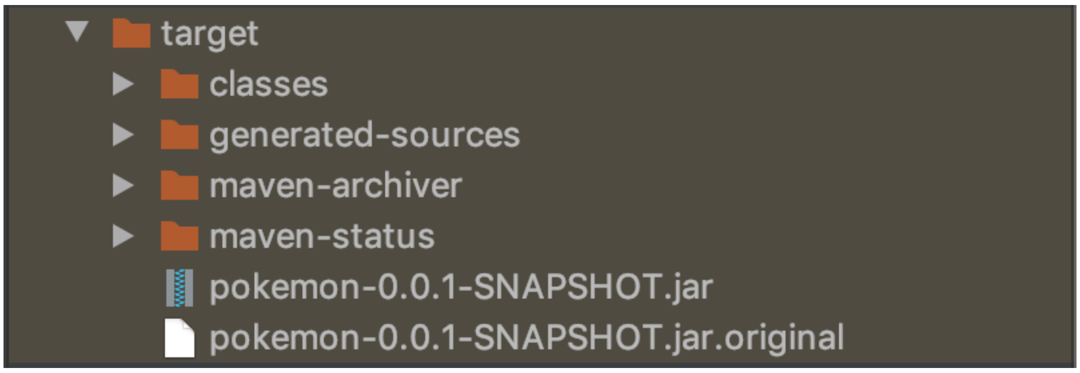
9. Now, before we go to the coding part, let’s install all the dependencies and build the jar file first by typing

mvn clean install -Dmaven.test.skip=true



```
ElliotHes-MBP:pokemon Elliott$ mvn clean install -Dmaven.test.skip=true
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< org.fit5042:pokemon >-----------------------
[INFO] Building pokemon 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ pokemon ---
[INFO] Deleting /Users/Elliott/Documents/Tutor/FIT5042/Week10/pokemon/target
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ pokemon ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ pokemon ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 4 source files to /Users/Elliott/Documents/Tutor/FIT5042/Week10/pokemon/target/classes
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:testResources (default-testResources) @ pokemon ---
[INFO] Not copying test resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ pokemon ---
[INFO] Not compiling test sources
[INFO]
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ pokemon ---
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-jar-plugin:3.1.2:jar (default-jar) @ pokemon ---
[INFO] Building jar: /Users/Elliott/Documents/Tutor/FIT5042/Week10/pokemon/target/pokemon-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.1.9.RELEASE:repackage (repackage) @ pokemon ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ pokemon ---
[INFO] Installing /Users/Elliott/Documents/Tutor/FIT5042/Week10/pokemon/target/pokemon-0.0.1-SNAPSHOT.jar to /Users/Elliott/.m2/repository/org/fit5042/pokemon/0.0.1-SNAPSHOT/pokemon-0.0.1-SNAPSHOT.jar
[INFO] Installing /Users/Elliott/Documents/Tutor/FIT5042/Week10/pokemon/pom.xml to /Users/Elliott/.m2/repository/org/fit5042/pokemon/0.0.1-SNAPSHOT/pokemon-0.0.1-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.707 s
```

After you see the successful message as showing above, you should be able to see all dependencies has been included in the External Libraries. Additionally, you can now see the target folder under the project root.



```
▼ target
    ▶ classes
    ▶ generated-sources
    ▶ maven-archiver
    ▶ maven-status
      pokemon-0.0.1-SNAPSHOT.jar
      pokemon-0.0.1-SNAPSHOT.jar.original
```

We are going to use the jar file here to create our Docker image later.

10. Go to *Pokemon* entity class (under *entity* package) and copy the code below:

```
package org.fit5042.pokemon.entity;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

import javax.persistence.*;
import java.io.Serializable;

@Entity
@Table(name = "pokemon")
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
public class Pokemon implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String name;

    private String power;

    public String getPower() {
        return power;
    }

    public void setPower(String power) {
        this.power = power;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

You might find a lot of errors when you first copy and paste the code, please press Command button and hover your mouse over the *@Entity* to let the IntelliJ to solve the issue.

11. Next, go to the *PokemonRepository* interface (under *repository* package) and paste the code showing as below:

```
package org.fit5042.pokemon.repository;

import org.fit5042.pokemon.entity.Pokemon;
import org.springframework.data.jpa.repository.JpaRepository;

public interface PokemonRepository extends JpaRepository<Pokemon, Long> { }
```

12. Go to the *PokemonController* class (under *controller* package), copy and paste the code below:

```java
package org.fit5042.pokemon.controller;

import org.fit5042.pokemon.entity.Pokemon;
import org.fit5042.pokemon.repository.PokemonRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api")
public class PokemonController {

    @Autowired
    private PokemonRepository pokemonRepository;

    @PostMapping("/pokemons")
    public Pokemon create(@RequestBody Pokemon pokemon)
    {
        return pokemonRepository.save(pokemon);
    }

    @RequestMapping(value = "/pokemons", method = RequestMethod.GET)
//    @GetMapping("/pokemons")
    public List<Pokemon> findAll()
    {
        return pokemonRepository.findAll();
    }

    @PutMapping("/pokemons/{pokemon_id}")
    public Pokemon update(@PathVariable("pokemon_id") Long pokemonId, @RequestBody Pokemon pokemonObject)
    {
        Pokemon pokemon = pokemonRepository.getOne(pokemonId);
        pokemon.setName(pokemonObject.getName());
        pokemon.setPower(pokemonObject.getPower());
        return pokemonRepository.save(pokemon);
    }

    @RequestMapping(value = "/pokemons/{pokemon_id}", method = RequestMethod.DELETE)
//    @DeleteMapping("/pokemons/{pokemon_id}")
    public List<Pokemon> delete(@PathVariable("pokemon_id") Long pokemonId)
    {
        pokemonRepository.deleteById(pokemonId);
        return pokemonRepository.findAll();
    }

    @GetMapping("/pokemons/{pokemon_id}")
    public Pokemon findByPokemonId(@PathVariable("pokemon_id") Long pokemonId)
    {
        return pokemonRepository.getOne(pokemonId);
    }
}
```

Here, you might notice that there are two ways for implementing the HTTP Request injection, you can either use *@[Get/Put/Delete/Post]Mapping("[url]")* or use *@RequestMapping* with the URL declared in the *value* property and HTTP method declared in the *method* property. These two dependency injection will give the same result, however, *@RequestMapping* might give you more control compare to *@[HTTP method]Mapping*.

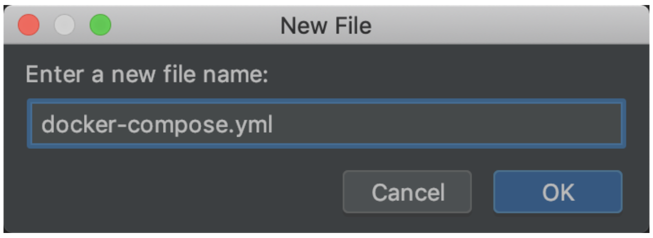
13. Go to *application.properties* file (under */src/main/resources* folder) and paste the code

```
## Spring DATA SOURCE Configurations
spring.datasource.url = jdbc:mysql://mysql-docker-
container:3306/spring_app_db?allowPublicKeyRetrieval=true&useSSL=false
spring.datasource.username = fit5042
spring.datasource.password = fit5042

## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update
spring.jackson.serialization.FAIL_ON_EMPTY_BEANS=false
```

14. Now, let's run the instruction below again in our terminal to build our project with the latest configurations:

mvn clean install -Dmaven.test.skip=true

15. Under the project root folder, let's create a new file called "*docker-compose.yml*".



16. Copy and paste the code below into the *docker-compose.yml* file:

```
version: '3'

services:
  mysql-docker-container:
    image: mysql:latest
    restart: always
    command: --default-authentication-plugin=mysql_native_password
    environment:
      - MYSQL_ROOT_PASSWORD=root123
      - MYSQL_DATABASE=spring_app_db
      - MYSQL_USER=fit5042
      - MYSQL_PASSWORD=fit5042
    volumes:
      - /data/mysql
    networks:
      - mynetwork
  spring-boot-jpa-app:
    image: spring-boot-jpa-image
    build:
      context: ./
      dockerfile: Dockerfile
    environment:
      - WAIT_HOSTS=mysql-docker-container:3306
    ports:
      - 8087:8080
    volumes:
      - /data/spring-boot-app
    networks:
      - mynetwork
networks:
  mynetwork:
```

Here, we configure these environment variables for our database with the username as “fit5042” and password as “fit5042”. As you might remember, we have also configured the database info in our *application.properties* file in step 13, to enable our application to link with the database properly.

17. Next, under project root folder, let's create a new file named "Dockerfile". This file can be considered as a docker script file to enable Docker to build the project based on all the command-line instructions which have been declared in the "Dockerfile".

```
FROM java:8
VOLUME /tmp
EXPOSE 8080
ADD https://github.com/ufoscout/docker-compose-wait/releases/download/2.4.0/wait /wait
ADD target/pokemon-0.0.1-SNAPSHOT.jar pokemon-0.0.1-SNAPSHOT.jar
RUN pwd && ls
RUN chmod +x /wait
CMD /wait && java -jar pokemon-0.0.1-SNAPSHOT.jar
```

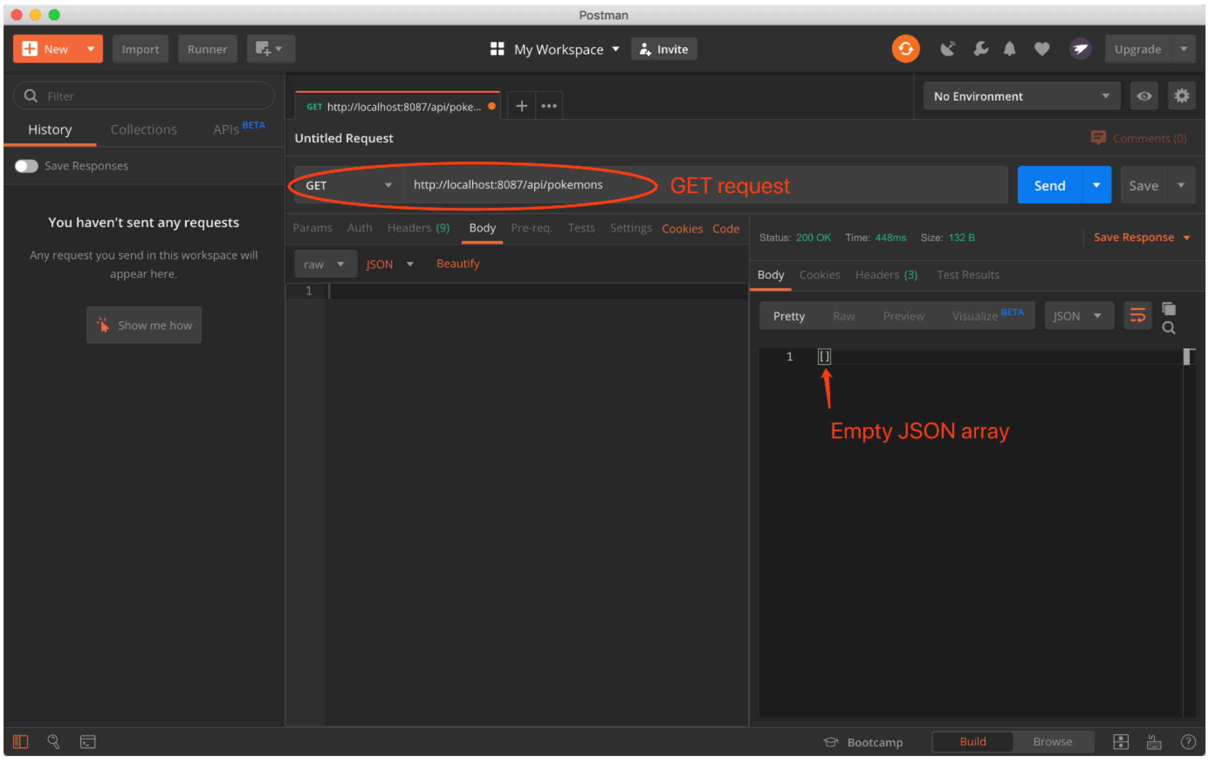
18. Till this stage, we have completed all the coding part. Now let's run the command in the terminal to let docker build the image for us:

docker-compose build

19. In order to run our web service application, run the instruction below to bring our image into the container:

docker-compose up

20. Now, let's test our API. Open the Postman, and type "http://localhost:8087/api/pokemons" and send a GET request. You should be able to see an empty JSON array since there's nothing in the database yet.



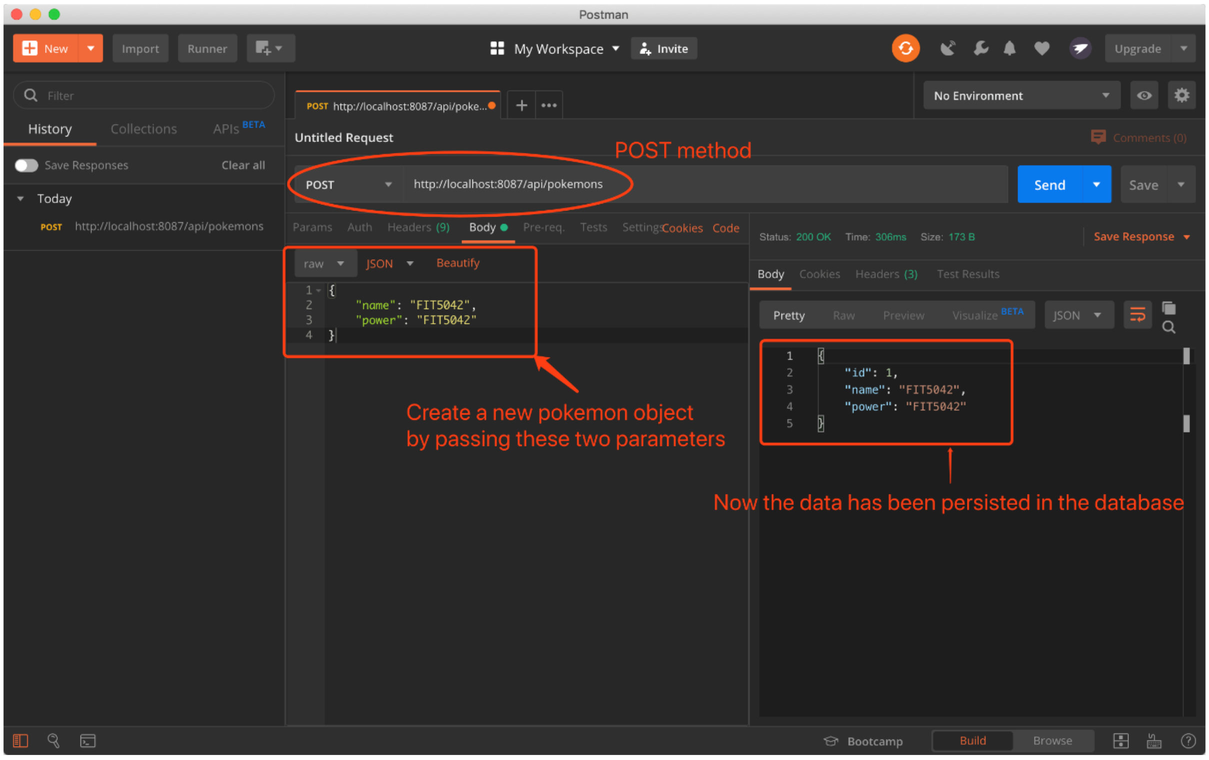
Postman — Untitled Request

GET http://localhost:8087/api/pokemons

Status: 200 OK Time: 448ms Size: 132 B

```
1 []
```

21. Let’s do a POST method to create a new pokemon and save it into the database



Untitled Request

POST method

```
POST http://localhost:8087/api/pokemons
```

```
{
    "name": "FIT5042",
    "power": "FIT5042"
}
```

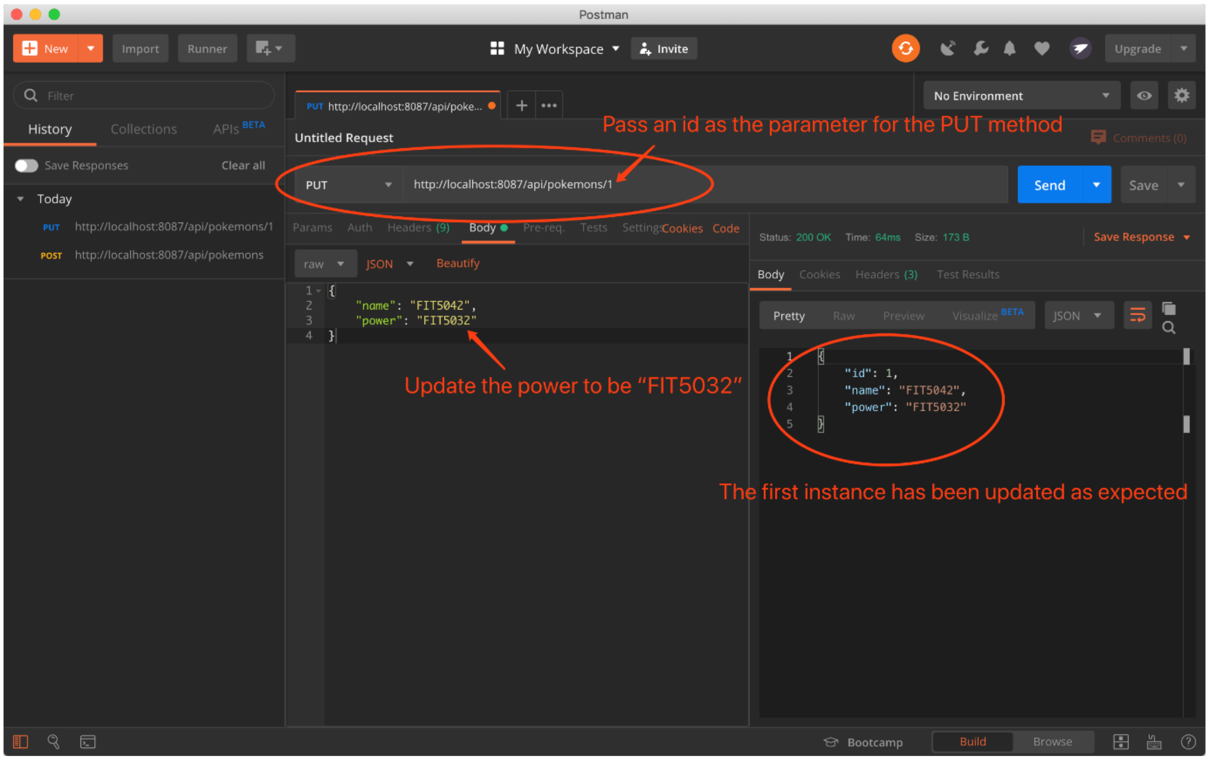
Create a new pokemon object by passing these two parameters

Status: 200 OK Time: 306ms Size: 173 B

```
{
    "id": 1,
    "name": "FIT5042",
    "power": "FIT5042"
}
```

Now the data has been persisted in the database

22. Now, try a PUT method by changing the power property to “FIT5032” for the first pokemon



Untitled Request

Pass an id as the parameter for the PUT method

```
PUT http://localhost:8087/api/pokemons/1
```

History:

```
PUT  http://localhost:8087/api/pokemons/1
POST http://localhost:8087/api/pokemons
```

```
{
    "name": "FIT5042",
    "power": "FIT5032"
}
```

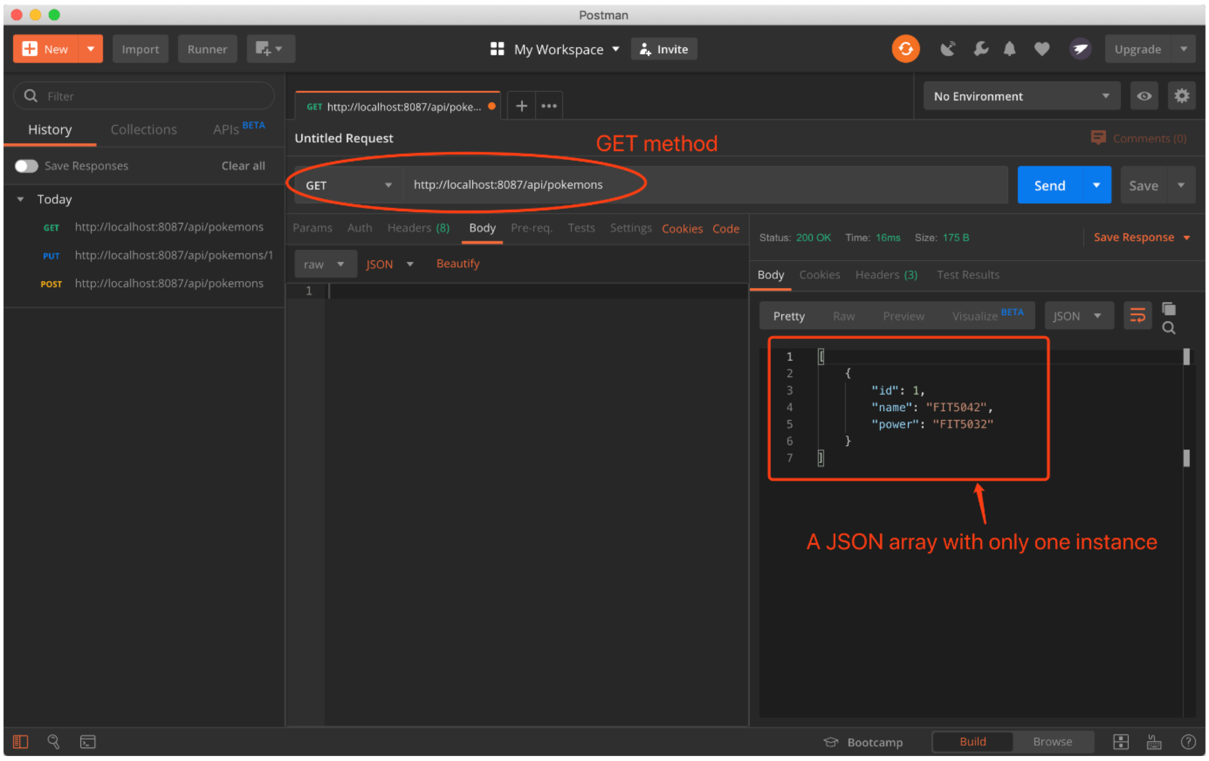
Update the power to be “FIT5032”

Status: 200 OK Time: 64ms Size: 173 B

```
{
    "id": 1,
    "name": "FIT5042",
    "power": "FIT5032"
}
```

The first instance has been updated as expected

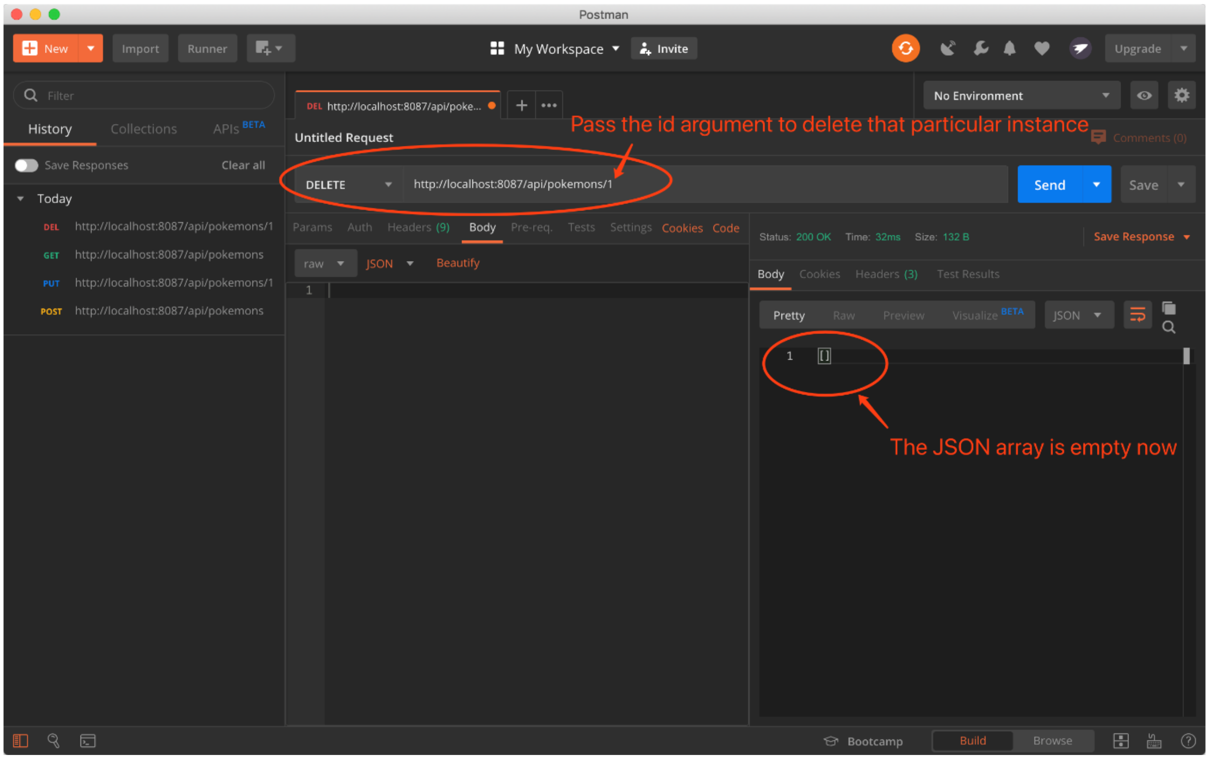
23. Next, let's do another GET request to check the latest database. Since we only have one instance in the database, the JSON array only contains one item.



GET method
GET
http://localhost:8087/api/pokemons
Send
Status: 200 OK Time: 16ms Size: 175 B
A JSON array with only one instance

```
[
    {
        "id": 1,
        "name": "FIT5042",
        "power": "FIT5032"
    }
]
```

24. Finally, let's try the DELETE request



Pass the id argument to delete that particular instance
DELETE
http://localhost:8087/api/pokemons/1
Send
Status: 200 OK Time: 32ms Size: 132 B
The JSON array is empty now

```
[]
```

25. Congratulations! Now, you have successfully finished this SpringBoot RESTful web service project with Docker container.

You have now completed this exercise.

You may improve upon the basic requirements if you desire.

CONCLUSION

At the completion of this tutorial, you would have gained an understanding of

- Deploy a springboot restful web service
- Understand CRUD operations
- Understand docker-compose
- Test the API's using postman