2205287\_cgcp: @workspace /fix

**1 – Table Design for City Table**

@workspace Create a Mongo Db schema for following table

cty\_code - Primary key, unique identifier

cty\_description - description of the city and input to the weather API

created\_at -The date and time its created

updated\_at -Modified date and time

**Output :**

To create a MongoDB schema for the given table, you can use the following JSON format:

{

"cty\_code": {

"type": "String",

"required": true,

"unique": true

},

"cty\_description": {

"type": "String",

"required": true

},

"created\_at": {

"type": "Date",

"default": Date.now

},

"updated\_at": {

"type": "Date",

"default": Date.now

}

}

This schema defines four fields for each city document: cty\_code, cty\_description, created\_at, and updated\_at. The cty\_code field is the primary key and a unique identifier for each city. The cty\_description field is the description of the city and the input to the weather API. The created\_at and updated\_at fields are timestamps that indicate when the document was created and modified, respectively. The default value for both fields is the current date and time. You can use the mongoose module to create a model based on this schema and perform CRUD operations on the city collection.

**2 – Table Design for Audit Table.**

@workspace Create a Mongodb Schema for audit table

Aud\_id - Primary key, unique identifier

Aud\_cty\_code - City code that’s been requested for weather/forecast

Aud\_enquiry\_type -Weather or forecast

Aud\_enquiry\_at- The date and time when the request is made for weather/forecast

**Output:**

You can create a Mongodb schema for the audit table as follows:

// Import mongoose module

const mongoose = require('mongoose');

// Define audit schema

const AuditSchema = new mongoose.Schema({

// Aud\_id field is the primary key and a unique identifier

aud\_id: {

type: String,

required: true,

unique: true

},

// Aud\_cty\_code field is the city code that's been requested for weather/forecast

aud\_cty\_code: {

type: String,

required: true,

ref: 'City' // Reference to the city collection

},

// Aud\_enquiry\_type field is the weather or forecast

aud\_enquiry\_type: {

type: String,

required: true,

enum: ['weather', 'forecast'] // Only allow these two values

},

// Aud\_enquiry\_at field is the date and time when the request is made

aud\_enquiry\_at: {

type: Date,

required: true,

default: Date.now // Default value is the current date and time

}

});

// Create a model based on the audit schema

const Audit = mongoose.model('Audit', AuditSchema);

// Export the model

module.exports = Audit;

Workspace: To identify and fix the problem in the provided code, let's first analyze it step-by-step:

1. \*\*Package Declaration\*\*: Correct.

2. \*\*Imports\*\*: Correct.

3. \*\*Class Declaration\*\*: Correct.

4. \*\*Main Method\*\*: Correct.

The provided code appears to be a standard Spring Boot application setup and does not contain any apparent bugs. However, if there is an implicit issue not visible in the provided excerpt, such as a missing dependency or configuration, it cannot be determined from this snippet alone.

Assuming there is no hidden context and the code is correct, here is the same code with no changes needed:

```java

package com.example;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class MySpringBootApp {

public static void main(String[] args) {

SpringApplication.run(MySpringBootApp.class, args);

}

}

```

If there is a specific issue you are encountering, please provide more details so I can assist you further.

2205287\_cgcp: run this spring bot

GitHub Copilot: To run the Spring Boot application, follow these steps:

1. \*\*Ensure you have Maven or Gradle installed\*\*: Spring Boot applications typically use Maven or Gradle for build and dependency management.

2. \*\*Navigate to the project directory\*\*: Open the terminal and navigate to the root directory of your Spring Boot project.

3. \*\*Run the application using Maven or Gradle\*\*:

- If you are using \*\*Maven\*\*, run:

```sh

./mvnw spring-boot:run

```

- If you are using \*\*Gradle\*\*, run:

```sh

./gradlew bootRun

```

Here are the detailed steps:

### Step-by-Step Instructions

1. \*\*Open Terminal\*\*:

- On your Mac, open the Terminal application.

2. \*\*Navigate to Project Directory\*\*:

```sh

cd /path/to/your/project

```

3. \*\*Run the Application\*\*:

- If using \*\*Maven\*\*:

```sh

./mvnw spring-boot:run

```

- If using \*\*Gradle\*\*:

```sh

./gradlew bootRun

```

### Example

Assuming your project is located at `/Users/yourusername/projects/MySpringBootApp`, you would do the following:

```sh

cd /Users/yourusername/projects/MySpringBootApp

./mvnw spring-boot:run

```

or

```sh

cd /Users/yourusername/projects/MySpringBootApp

./gradlew bootRun

```

This will start your Spring Boot application, and you should see output in the terminal indicating that the application has started successfully.

2205287\_cgcp: no such file ./gradelew

GitHub Copilot: It looks like you might be using Maven instead of Gradle. Let's proceed with Maven commands. If you don't have the Maven wrapper (`./mvnw`), you can use the `mvn` command directly.

### Steps to Run the Spring Boot Application with Maven

1. \*\*Open Terminal\*\*:

- On your Mac, open the Terminal application.

2. \*\*Navigate to Project Directory\*\*:

```sh

cd /path/to/your/project

```

3. \*\*Run the Application\*\*:

```sh

mvn spring-boot:run

```

### Example

Assuming your project is located at `/Users/yourusername/projects/MySpringBootApp`, you would do the following:

```sh

cd /Users/yourusername/projects/MySpringBootApp

mvn spring-boot:run

```

This command will compile your project and start the Spring Boot application. You should see output in the terminal indicating that the application has started successfully.

2205287\_cgcp: using gradle only

2205287\_cgcp: regenerate again

GitHub Copilot: Sure, here is the regenerated code for your Spring Boot application:

```java

package com.example;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class MySpringBootApp {

public static void main(String[] args) {

SpringApplication.run(MySpringBootApp.class, args);

}

}

```

To run this Spring Boot application, follow these steps:

1. \*\*Open Terminal\*\*:

- On your Mac, open the Terminal application.

2. \*\*Navigate to Project Directory\*\*:

```sh

cd /path/to/your/project

```

3. \*\*Run the Application\*\*:

```sh

mvn spring-boot:run

```

Replace `/path/to/your/project` with the actual path to your project directory. This will compile and start your Spring Boot application.

2205287\_cgcp: gradle

GitHub Copilot: To run your Spring Boot application using Gradle, you need to ensure that you have a [`build.gradle`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fbuild.gradle%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/build.gradle") file in your project directory. Here is a basic [`build.gradle`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fbuild.gradle%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/build.gradle") file for a Spring Boot application:

### Step 1: Create [`build.gradle`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fbuild.gradle%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/build.gradle")

```groovy

plugins {

id 'org.springframework.boot' version '3.0.0'

id 'io.spring.dependency-management' version '1.0.15.RELEASE'

id 'java'

}

group = 'com.example'

version = '0.0.1-SNAPSHOT'

sourceCompatibility = '17'

repositories {

mavenCentral()

}

dependencies {

implementation 'org.springframework.boot:spring-boot-starter'

testImplementation 'org.springframework.boot:spring-boot-starter-test'

}

test {

useJUnitPlatform()

}

```

### Step 2: Run the Application

1. \*\*Open Terminal\*\*:

- On your Mac, open the Terminal application.

2. \*\*Navigate to Project Directory\*\*:

```sh

cd /path/to/your/project

```

3. \*\*Run the Application\*\*:

```sh

./gradlew bootRun

```

### Example

Assuming your project is located at `/Users/yourusername/projects/MySpringBootApp`, you would do the following:

```sh

cd /Users/yourusername/projects/MySpringBootApp

./gradlew bootRun

```

This will compile and start your Spring Boot application. You should see output in the terminal indicating that the application has started successfully.

2205287\_cgcp: enable public cide

GitHub Copilot: To enable public access to your Spring Boot application, you need to configure it to listen on all network interfaces. By default, Spring Boot applications listen on `localhost` (127.0.0.1), which means they are only accessible from the same machine.

### Step-by-Step Instructions

1. \*\*Open `application.properties` or `application.yml`\*\*:

- If you are using `application.properties`, add the following line:

```properties

server.address=0.0.0.0

```

- If you are using `application.yml`, add the following lines:

```yaml

server:

address: 0.0.0.0

```

2. \*\*Ensure your firewall allows traffic on the specified port\*\*:

- By default, Spring Boot applications run on port 8080. Make sure your firewall settings allow incoming traffic on this port.

3. \*\*Run the Application\*\*:

- Open Terminal.

- Navigate to your project directory.

- Run the application using Gradle:

```sh

./gradlew bootRun

```

### Example

#### `application.properties`

```properties

server.address=0.0.0.0

```

#### `application.yml`

```yaml

server:

address: 0.0.0.0

```

### Running the Application

Assuming your project is located at `/Users/yourusername/projects/MySpringBootApp`, you would do the following:

```sh

cd /Users/yourusername/projects/MySpringBootApp

./gradlew bootRun

```

This configuration will make your Spring Boot application accessible from any device on the same network using the IP address of the machine running the application. For example, if your machine's IP address is `192.168.1.100`, you can access the application at `http://192.168.1.100:8080`.

2205287\_cgcp: create a documentation for your suggestions

GitHub Copilot: ### Documentation for Running a Spring Boot Application with Gradle and Enabling Public Access

This documentation provides step-by-step instructions on how to run a Spring Boot application using Gradle and configure it to be accessible from other devices on the same network.

#### Prerequisites

- Ensure you have Gradle installed.

- Ensure you have a Spring Boot project with a [`build.gradle`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fbuild.gradle%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/build.gradle") file.

### Step 1: Create or Update [`build.gradle`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fbuild.gradle%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/build.gradle")

Ensure your [`build.gradle`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fbuild.gradle%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/build.gradle") file includes the necessary plugins and dependencies for a Spring Boot application. Here is an example:

```groovy

plugins {

id 'org.springframework.boot' version '3.0.0'

id 'io.spring.dependency-management' version '1.0.15.RELEASE'

id 'java'

}

group = 'com.example'

version = '0.0.1-SNAPSHOT'

sourceCompatibility = '17'

repositories {

mavenCentral()

}

dependencies {

implementation 'org.springframework.boot:spring-boot-starter'

testImplementation 'org.springframework.boot:spring-boot-starter-test'

}

test {

useJUnitPlatform()

}

```

### Step 2: Configure Application for Public Access

To make your Spring Boot application accessible from other devices on the same network, you need to configure it to listen on all network interfaces.

#### Using `application.properties`

Create or update the [`src/main/resources/application.properties`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fsrc%2Fmain%2Fresources%2Fapplication.properties%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/src/main/resources/application.properties") file with the following content:

```properties

server.address=0.0.0.0

```

#### Using `application.yml`

Alternatively, if you prefer YAML configuration, create or update the `src/main/resources/application.yml` file with the following content:

```yaml

server:

address: 0.0.0.0

```

### Step 3: Run the Application

1. \*\*Open Terminal\*\*:

- On your Mac, open the Terminal application.

2. \*\*Navigate to Project Directory\*\*:

```sh

cd /path/to/your/project

```

3. \*\*Run the Application\*\*:

```sh

./gradlew bootRun

```

### Example

Assuming your project is located at `/Users/yourusername/projects/MySpringBootApp`, you would do the following:

```sh

cd /Users/yourusername/projects/MySpringBootApp

./gradlew bootRun

```

### Step 4: Access the Application

Once the application is running, it will be accessible from any device on the same network using the IP address of the machine running the application. For example, if your machine's IP address is `192.168.1.100`, you can access the application at:

```

http://192.168.1.100:8080

```

### Firewall Configuration

Ensure your firewall settings allow incoming traffic on the port your Spring Boot application is running on (default is 8080).

### Conclusion

By following these steps, you can run your Spring Boot application using Gradle and configure it to be accessible from other devices on the same network. This setup is useful for testing and development purposes.

2205287\_cgcp: /track all commands histrory

GitHub Copilot: To track all command history in your terminal, you can use the built-in history functionality of your shell. Here are the steps for different shells commonly used on macOS:

### For Bash Shell

1. \*\*View Command History\*\*:

```sh

history

```

2. \*\*Save Command History to a File\*\*:

```sh

history > command\_history.txt

```

3. \*\*Configure Persistent Command History\*\*:

Add the following lines to your `~/.bash\_profile` or `~/.bashrc` file to ensure that your command history is saved between sessions:

```sh

# Append to the history file, don't overwrite it

shopt -s histappend

# Save multi-line commands as a single command

shopt -s cmdhist

# Increase the size of the history file

HISTFILESIZE=10000

HISTSIZE=10000

# Save the history after each command

PROMPT\_COMMAND="history -a; history -c; history -r; $PROMPT\_COMMAND"

```

### For Zsh Shell

1. \*\*View Command History\*\*:

```sh

history

```

2. \*\*Save Command History to a File\*\*:

```sh

history > command\_history.txt

```

3. \*\*Configure Persistent Command History\*\*:

Add the following lines to your `~/.zshrc` file to ensure that your command history is saved between sessions:

```sh

# Append to the history file, don't overwrite it

setopt APPEND\_HISTORY

# Save multi-line commands as a single command

setopt HIST\_IGNORE\_SPACE

# Increase the size of the history file

HISTFILE=~/.zsh\_history

HISTSIZE=10000

SAVEHIST=10000

# Save the history after each command

precmd() { history -a }

```

### For Fish Shell

1. \*\*View Command History\*\*:

```sh

history

```

2. \*\*Save Command History to a File\*\*:

```sh

history > command\_history.txt

```

3. \*\*Configure Persistent Command History\*\*:

Fish shell automatically saves command history between sessions. You can configure the history settings in the `~/.config/fish/config.fish` file if needed.

### Example: Saving Command History to a File

```sh

# Open Terminal

cd /path/to/your/project

# Run the application

./gradlew bootRun

# Save command history to a file

history > command\_history.txt

```

By following these steps, you can track and save all command history in your terminal.

2205287\_cgcp: track githuub copilot commands

GitHub Copilot: To track the commands issued by GitHub Copilot within Visual Studio Code, you can use the built-in command history feature of the terminal and also keep track of the actions performed by Copilot. Here’s how you can do it:

### Step 1: Enable Command History in Terminal

1. \*\*Open Terminal in VS Code\*\*:

- You can open the terminal by pressing `Ctrl + ` (backtick) or navigating to `View > Terminal`.

2. \*\*View Command History\*\*:

- In the terminal, you can view the command history by typing:

```sh

history

```

3. \*\*Save Command History to a File\*\*:

- To save the command history to a file, you can use:

```sh

history > command\_history.txt

```

### Step 2: Track GitHub Copilot Commands

GitHub Copilot itself does not have a built-in feature to log commands directly. However, you can manually keep track of the commands and actions you perform with Copilot by maintaining a log file.

### Step 3: Create a Log File for Copilot Commands

1. \*\*Create a Log File\*\*:

- Create a new file in your project directory named `copilot\_commands.log`.

2. \*\*Manually Log Commands\*\*:

- Whenever you use a command or action with GitHub Copilot, manually add an entry to the log file. For example:

```plaintext

[2023-10-01 10:00] Generated Spring Boot application main class

[2023-10-01 10:05] Configured application.properties for public access

[2023-10-01 10:10] Ran application using ./gradlew bootRun

```

### Example: Logging Commands

```plaintext

# copilot\_commands.log

[2023-10-01 10:00] Generated Spring Boot application main class

[2023-10-01 10:05] Configured application.properties for public access

[2023-10-01 10:10] Ran application using ./gradlew bootRun

```

### Automating Command Logging (Optional)

If you want to automate the logging process, you can create a script that appends each command you run to the log file. Here’s an example using a Bash script:

1. \*\*Create a Script\*\*:

- Create a file named `log\_command.sh` with the following content:

```sh

#!/bin/bash

echo "[$(date '+%Y-%m-%d %H:%M')] $@" >> copilot\_commands.log

```

2. \*\*Make the Script Executable\*\*:

```sh

chmod +x log\_command.sh

```

3. \*\*Use the Script to Log Commands\*\*:

- Instead of running commands directly, use the script to log them:

```sh

./log\_command.sh "Ran application using ./gradlew bootRun"

./gradlew bootRun

```

By following these steps, you can effectively track and log the commands and actions performed with GitHub Copilot in Visual Studio Code.

2205287\_cgcp: @workspace Design a Database and table to store various information about cities

Workspace: To design a database and table to store various information about cities, we can use a relational database model. Below is an example of how you might structure this database using SQL.

### Database: `CityInfo`

#### Table: `Cities`

The `Cities` table will store various information about cities, such as their name, country, population, area, and other relevant details.

```sql

CREATE DATABASE CityInfo;

USE CityInfo;

CREATE TABLE Cities (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

country VARCHAR(100) NOT NULL,

population INT,

area FLOAT,

latitude DECIMAL(9, 6),

longitude DECIMAL(9, 6),

timezone VARCHAR(50),

established DATE,

description TEXT

);

```

### Example Data Insertion

Here is an example of how you might insert data into the `Cities` table:

```sql

INSERT INTO Cities (name, country, population, area, latitude, longitude, timezone, established, description)

VALUES

('New York', 'USA', 8419000, 783.8, 40.7128, -74.0060, 'Eastern Time Zone', '1624-01-01', 'The largest city in the United States.'),

('Tokyo', 'Japan', 13929286, 2194.07, 35.6895, 139.6917, 'Japan Standard Time', '1603-01-01', 'The capital city of Japan.'),

('Paris', 'France', 2148327, 105.4, 48.8566, 2.3522, 'Central European Time', '508-01-01', 'The capital city of France, known for its art, fashion, and culture.');

```

### Explanation

- \*\*id\*\*: A unique identifier for each city.

- \*\*name\*\*: The name of the city.

- \*\*country\*\*: The country where the city is located.

- \*\*population\*\*: The population of the city.

- \*\*area\*\*: The area of the city in square kilometers.

- \*\*latitude\*\*: The latitude coordinate of the city.

- \*\*longitude\*\*: The longitude coordinate of the city.

- \*\*timezone\*\*: The timezone of the city.

- \*\*established\*\*: The date when the city was established.

- \*\*description\*\*: A brief description of the city.

This structure allows you to store comprehensive information about various cities in a well-organized manner. You can expand this schema by adding more tables to store related information, such as historical events, landmarks, or economic data.

2205287\_cgcp: @workspace create a mongodb schema version as well

Workspace: To create a MongoDB schema to store various information about cities, we can use a JSON-like structure. MongoDB is a NoSQL database, so it doesn't require a predefined schema, but we can define a structure for our documents.

### Database: `CityInfo`

#### Collection: `cities`

Each document in the `cities` collection will store information about a city, such as its name, country, population, area, and other relevant details.

### Example Document Structure

```json

{

"\_id": ObjectId("60c72b2f9b1d8b3f4c8b4567"),

"name": "New York",

"country": "USA",

"population": 8419000,

"area": 783.8,

"coordinates": {

"latitude": 40.7128,

"longitude": -74.0060

},

"timezone": "Eastern Time Zone",

"established": ISODate("1624-01-01T00:00:00Z"),

"description": "The largest city in the United States."

}

```

### Example Data Insertion

Here is an example of how you might insert data into the `cities` collection using MongoDB:

```javascript

db.cities.insertMany([

{

"name": "New York",

"country": "USA",

"population": 8419000,

"area": 783.8,

"coordinates": {

"latitude": 40.7128,

"longitude": -74.0060

},

"timezone": "Eastern Time Zone",

"established": ISODate("1624-01-01T00:00:00Z"),

"description": "The largest city in the United States."

},

{

"name": "Tokyo",

"country": "Japan",

"population": 13929286,

"area": 2194.07,

"coordinates": {

"latitude": 35.6895,

"longitude": 139.6917

},

"timezone": "Japan Standard Time",

"established": ISODate("1603-01-01T00:00:00Z"),

"description": "The capital city of Japan."

},

{

"name": "Paris",

"country": "France",

"population": 2148327,

"area": 105.4,

"coordinates": {

"latitude": 48.8566,

"longitude": 2.3522

},

"timezone": "Central European Time",

"established": ISODate("0508-01-01T00:00:00Z"),

"description": "The capital city of France, known for its art, fashion, and culture."

}

]);

```

### Explanation

- \*\*\_id\*\*: A unique identifier for each city document, automatically generated by MongoDB.

- \*\*name\*\*: The name of the city.

- \*\*country\*\*: The country where the city is located.

- \*\*population\*\*: The population of the city.

- \*\*area\*\*: The area of the city in square kilometers.

- \*\*coordinates\*\*: An embedded document containing the latitude and longitude coordinates of the city.

- \*\*timezone\*\*: The timezone of the city.

- \*\*established\*\*: The date when the city was established, stored as an ISODate.

- \*\*description\*\*: A brief description of the city.

This structure allows you to store comprehensive information about various cities in a flexible and scalable manner using MongoDB.

2205287\_cgcp: @workspace create spring project with rest controller and JPA Method for getting city information from Mongodb

Workspace: To create a Spring Boot project with a REST controller and JPA method for getting city information from MongoDB, follow these steps:

### 1. Project Setup

First, ensure your [`build.gradle`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fbuild.gradle%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/build.gradle") includes the necessary dependencies for Spring Boot, Spring Data JPA, and MongoDB.

#### [`build.gradle`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fbuild.gradle%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/build.gradle")

```gradle

plugins {

id 'org.springframework.boot' version '2.5.4'

id 'io.spring.dependency-management' version '1.0.11.RELEASE'

id 'java'

}

group 'com.example'

version '1.0-SNAPSHOT'

repositories {

mavenCentral()

}

dependencies {

implementation 'org.springframework.boot:spring-boot-starter-web'

implementation 'org.springframework.boot:spring-boot-starter-data-mongodb'

testImplementation 'org.springframework.boot:spring-boot-starter-test'

}

test {

useJUnitPlatform()

}

```

### 2. MongoDB Configuration

Configure MongoDB connection in [`src/main/resources/application.properties`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fsrc%2Fmain%2Fresources%2Fapplication.properties%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/src/main/resources/application.properties").

#### [`application.properties`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fsrc%2Fmain%2Fresources%2Fapplication.properties%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/src/main/resources/application.properties")

```properties

# Spring Boot application properties

# MongoDB configuration

spring.data.mongodb.uri=mongodb://localhost:27017/cityinfo

```

### 3. Create City Entity

Create a `City` entity class in [`src/main/java/com/example/model/City.java`](command:\_github.copilot.openSymbolFromReferences?%5B%22src%2Fmain%2Fjava%2Fcom%2Fexample%2Fmodel%2FCity.java%22%2C%5B%7B%22uri%22%3A%7B%22%24mid%22%3A1%2C%22fsPath%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22external%22%3A%22file%3A%2F%2F%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22scheme%22%3A%22file%22%7D%2C%22pos%22%3A%7B%22line%22%3A44%2C%22character%22%3A97%7D%7D%5D%5D "Go to definition").

#### `City.java`

```java

package com.example.model;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "cities")

public class City {

@Id

private String id;

private String name;

private String country;

private int population;

private double area;

private double latitude;

private double longitude;

private String timezone;

private String established;

private String description;

// Getters and Setters

}

```

### 4. Create City Repository

Create a repository interface for the `City` entity in [`src/main/java/com/example/repository/CityRepository.java`](command:\_github.copilot.openSymbolFromReferences?%5B%22src%2Fmain%2Fjava%2Fcom%2Fexample%2Frepository%2FCityRepository.java%22%2C%5B%7B%22uri%22%3A%7B%22%24mid%22%3A1%2C%22fsPath%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22external%22%3A%22file%3A%2F%2F%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22scheme%22%3A%22file%22%7D%2C%22pos%22%3A%7B%22line%22%3A44%2C%22character%22%3A97%7D%7D%5D%5D "Go to definition").

#### `CityRepository.java`

```java

package com.example.repository;

import com.example.model.City;

import org.springframework.data.mongodb.repository.MongoRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface CityRepository extends MongoRepository<City, String> {

City findByName(String name);

}

```

### 5. Create City Service

Create a service class to handle business logic in [`src/main/java/com/example/service/CityService.java`](command:\_github.copilot.openSymbolFromReferences?%5B%22src%2Fmain%2Fjava%2Fcom%2Fexample%2Fservice%2FCityService.java%22%2C%5B%7B%22uri%22%3A%7B%22%24mid%22%3A1%2C%22fsPath%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22external%22%3A%22file%3A%2F%2F%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22scheme%22%3A%22file%22%7D%2C%22pos%22%3A%7B%22line%22%3A44%2C%22character%22%3A97%7D%7D%5D%5D "Go to definition").

#### `CityService.java`

```java

package com.example.service;

import com.example.model.City;

import com.example.repository.CityRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class CityService {

@Autowired

private CityRepository cityRepository;

public List<City> getAllCities() {

return cityRepository.findAll();

}

public City getCityByName(String name) {

return cityRepository.findByName(name);

}

}

```

### 6. Create City Controller

Create a REST controller to expose endpoints in [`src/main/java/com/example/controller/CityController.java`](command:\_github.copilot.openSymbolFromReferences?%5B%22src%2Fmain%2Fjava%2Fcom%2Fexample%2Fcontroller%2FCityController.java%22%2C%5B%7B%22uri%22%3A%7B%22%24mid%22%3A1%2C%22fsPath%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22external%22%3A%22file%3A%2F%2F%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22scheme%22%3A%22file%22%7D%2C%22pos%22%3A%7B%22line%22%3A44%2C%22character%22%3A97%7D%7D%5D%5D "Go to definition").

#### `CityController.java`

```java

package com.example.controller;

import com.example.model.City;

import com.example.service.CityService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/cities")

public class CityController {

@Autowired

private CityService cityService;

@GetMapping

public List<City> getAllCities() {

return cityService.getAllCities();

}

@GetMapping("/{name}")

public City getCityByName(@PathVariable String name) {

return cityService.getCityByName(name);

}

}

```

### 7. Main Application Class

Ensure your main application class is set up correctly in [`src/main/java/com/example/MySpringBootApp.java`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fsrc%2Fmain%2Fjava%2Fcom%2Fexample%2FMySpringBootApp.java%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/src/main/java/com/example/MySpringBootApp.java").

#### [`MySpringBootApp.java`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fsrc%2Fmain%2Fjava%2Fcom%2Fexample%2FMySpringBootApp.java%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/src/main/java/com/example/MySpringBootApp.java")

```java

package com.example;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class MySpringBootApp {

public static void main(String[] args) {

SpringApplication.run(MySpringBootApp.class, args);

}

}

```

### 8. Running the Application

To run the application, use the following command:

```sh

gradle bootRun

```

### 9. Testing the Endpoints

You can test the endpoints using tools like Postman or curl.

- Get all cities: [`GET http://localhost:8080/api/cities`](command:\_github.copilot.openSymbolFromReferences?%5B%22GET%20http%3A%2F%2Flocalhost%3A8080%2Fapi%2Fcities%22%2C%5B%7B%22uri%22%3A%7B%22%24mid%22%3A1%2C%22fsPath%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22external%22%3A%22file%3A%2F%2F%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22scheme%22%3A%22file%22%7D%2C%22pos%22%3A%7B%22line%22%3A14%2C%22character%22%3A13%7D%7D%5D%5D "Go to definition")

- Get a city by name: [`GET http://localhost:8080/api/cities/{name}`](command:\_github.copilot.openSymbolFromReferences?%5B%22GET%20http%3A%2F%2Flocalhost%3A8080%2Fapi%2Fcities%2F%7Bname%7D%22%2C%5B%7B%22uri%22%3A%7B%22%24mid%22%3A1%2C%22fsPath%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22external%22%3A%22file%3A%2F%2F%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22scheme%22%3A%22file%22%7D%2C%22pos%22%3A%7B%22line%22%3A14%2C%22character%22%3A13%7D%7D%5D%5D "Go to definition")

This setup provides a basic Spring Boot application with a REST controller and JPA method for getting city information from MongoDB.

2205287\_cgcp: @workspace add the above code and create a new project

Workspace: To create a new Spring Boot project with the provided structure and code, follow these steps:

### 1. Project Structure

Create the following directory structure:

```

my-spring-boot-app/

├── .gradle/

├── build.gradle

├── README.md

├── settings.gradle

├── src/

│ ├── main/

│ │ ├── java/

│ │ │ └── com/

│ │ │ └── example/

│ │ │ └── MySpringBootApp.java

│ │ └── resources/

│ │ └── application.properties

│ └── test/

│ ├── java/

│ │ └── com/

│ │ └── example/

│ │ └── MySpringBootAppTests.java

│ └── resources/

```

### 2. Create [`build.gradle`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fbuild.gradle%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/build.gradle")

Create the [`build.gradle`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fbuild.gradle%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/build.gradle") file with the following content:

```gradle

plugins {

id 'org.springframework.boot' version '2.5.4'

id 'io.spring.dependency-management' version '1.0.11.RELEASE'

id 'java'

}

group 'com.example'

version '1.0-SNAPSHOT'

repositories {

mavenCentral()

}

dependencies {

implementation 'org.springframework.boot:spring-boot-starter-web'

implementation 'org.springframework.boot:spring-boot-starter-data-mongodb'

testImplementation 'org.springframework.boot:spring-boot-starter-test'

}

test {

useJUnitPlatform()

}

```

### 3. Create [`README.md`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/README.md")

Create the [`README.md`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/README.md") file with the following content:

```md

# My Spring Boot App

This is a Spring Boot application that demonstrates the usage of JUnit tests.

## Prerequisites

- Java 8 or higher

- Gradle

## Getting Started

1. Clone the repository:

```shell

git clone https://github.com/your-username/my-spring-boot-app.git

```

2. Build the project:

```shell

cd my-spring-boot-app

gradle build

```

3. Run the application:

```shell

gradle bootRun

```

4. Access the application:

Open your web browser and navigate to [http://localhost:8080](http://localhost:8080).

## Running Tests

To run the JUnit tests, execute the following command:

```shell

gradle test

```

## Configuration

The application can be configured by modifying the [`application.properties`](command:\_github.copilot.openSymbolFromReferences?%5B%22application.properties%22%2C%5B%7B%22uri%22%3A%7B%22%24mid%22%3A1%2C%22fsPath%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22external%22%3A%22file%3A%2F%2F%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22scheme%22%3A%22file%22%7D%2C%22pos%22%3A%7B%22line%22%3A44%2C%22character%22%3A52%7D%7D%5D%5D "Go to definition") file located in the [`src/main/resources`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fsrc%2Fmain%2Fresources%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/src/main/resources") directory.

## License

This project is licensed under the MIT License. See the LICENSE file for details.

```

### 4. Create [`settings.gradle`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fsettings.gradle%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/settings.gradle")

Create the [`settings.gradle`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fsettings.gradle%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/settings.gradle") file with the following content:

```gradle

// This file is intentionally left blank.

```

### 5. Create [`MySpringBootApp.java`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fsrc%2Fmain%2Fjava%2Fcom%2Fexample%2FMySpringBootApp.java%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/src/main/java/com/example/MySpringBootApp.java")

Create the [`MySpringBootApp.java`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fsrc%2Fmain%2Fjava%2Fcom%2Fexample%2FMySpringBootApp.java%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/src/main/java/com/example/MySpringBootApp.java") file in [`src/main/java/com/example/`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fsrc%2Fmain%2Fjava%2Fcom%2Fexample%2F%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/src/main/java/com/example/") with the following content:

```java

package com.example;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class MySpringBootApp {

public static void main(String[] args) {

SpringApplication.run(MySpringBootApp.class, args);

}

}

```

### 6. Create [`application.properties`](command:\_github.copilot.openSymbolFromReferences?%5B%22application.properties%22%2C%5B%7B%22uri%22%3A%7B%22%24mid%22%3A1%2C%22fsPath%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22external%22%3A%22file%3A%2F%2F%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22scheme%22%3A%22file%22%7D%2C%22pos%22%3A%7B%22line%22%3A44%2C%22character%22%3A52%7D%7D%5D%5D "Go to definition")

Create the [`application.properties`](command:\_github.copilot.openSymbolFromReferences?%5B%22application.properties%22%2C%5B%7B%22uri%22%3A%7B%22%24mid%22%3A1%2C%22fsPath%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22external%22%3A%22file%3A%2F%2F%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22scheme%22%3A%22file%22%7D%2C%22pos%22%3A%7B%22line%22%3A44%2C%22character%22%3A52%7D%7D%5D%5D "Go to definition") file in [`src/main/resources/`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fsrc%2Fmain%2Fresources%2F%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/src/main/resources/") with the following content:

```properties

# Spring Boot application properties

# MongoDB configuration

spring.data.mongodb.uri=mongodb://localhost:27017/cityinfo

```

### 7. Create [`MySpringBootAppTests.java`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fsrc%2Ftest%2Fjava%2Fcom%2Fexample%2FMySpringBootAppTests.java%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/src/test/java/com/example/MySpringBootAppTests.java")

Create the [`MySpringBootAppTests.java`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fsrc%2Ftest%2Fjava%2Fcom%2Fexample%2FMySpringBootAppTests.java%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/src/test/java/com/example/MySpringBootAppTests.java") file in [`src/test/java/com/example/`](command:\_github.copilot.openRelativePath?%5B%7B%22scheme%22%3A%22file%22%2C%22authority%22%3A%22%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2Fsrc%2Ftest%2Fjava%2Fcom%2Fexample%2F%22%2C%22query%22%3A%22%22%2C%22fragment%22%3A%22%22%7D%5D "/Users/2205287/Documents/Hackathon/my-spring-boot-app/src/test/java/com/example/") with the following content:

```java

package com.example;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

class MySpringBootAppTests {

@Test

void contextLoads() {

// Add your test logic here

}

}

```

### 8. Create City Entity

Create the `City.java` file in [`src/main/java/com/example/model/`](command:\_github.copilot.openSymbolFromReferences?%5B%22src%2Fmain%2Fjava%2Fcom%2Fexample%2Fmodel%2F%22%2C%5B%7B%22uri%22%3A%7B%22%24mid%22%3A1%2C%22fsPath%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22external%22%3A%22file%3A%2F%2F%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22scheme%22%3A%22file%22%7D%2C%22pos%22%3A%7B%22line%22%3A44%2C%22character%22%3A97%7D%7D%5D%5D "Go to definition") with the following content:

```java

package com.example.model;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "cities")

public class City {

@Id

private String id;

private String name;

private String country;

private int population;

private double area;

private double latitude;

private double longitude;

private String timezone;

private String established;

private String description;

// Getters and Setters

}

```

### 9. Create City Repository

Create the `CityRepository.java` file in [`src/main/java/com/example/repository/`](command:\_github.copilot.openSymbolFromReferences?%5B%22src%2Fmain%2Fjava%2Fcom%2Fexample%2Frepository%2F%22%2C%5B%7B%22uri%22%3A%7B%22%24mid%22%3A1%2C%22fsPath%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22external%22%3A%22file%3A%2F%2F%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22scheme%22%3A%22file%22%7D%2C%22pos%22%3A%7B%22line%22%3A44%2C%22character%22%3A97%7D%7D%5D%5D "Go to definition") with the following content:

```java

package com.example.repository;

import com.example.model.City;

import org.springframework.data.mongodb.repository.MongoRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface CityRepository extends MongoRepository<City, String> {

City findByName(String name);

}

```

### 10. Create City Service

Create the `CityService.java` file in [`src/main/java/com/example/service/`](command:\_github.copilot.openSymbolFromReferences?%5B%22src%2Fmain%2Fjava%2Fcom%2Fexample%2Fservice%2F%22%2C%5B%7B%22uri%22%3A%7B%22%24mid%22%3A1%2C%22fsPath%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22external%22%3A%22file%3A%2F%2F%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22scheme%22%3A%22file%22%7D%2C%22pos%22%3A%7B%22line%22%3A44%2C%22character%22%3A97%7D%7D%5D%5D "Go to definition") with the following content:

```java

package com.example.service;

import com.example.model.City;

import com.example.repository.CityRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class CityService {

@Autowired

private CityRepository cityRepository;

public List<City> getAllCities() {

return cityRepository.findAll();

}

public City getCityByName(String name) {

return cityRepository.findByName(name);

}

}

```

### 11. Create City Controller

Create the `CityController.java` file in [`src/main/java/com/example/controller/`](command:\_github.copilot.openSymbolFromReferences?%5B%22src%2Fmain%2Fjava%2Fcom%2Fexample%2Fcontroller%2F%22%2C%5B%7B%22uri%22%3A%7B%22%24mid%22%3A1%2C%22fsPath%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22external%22%3A%22file%3A%2F%2F%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22scheme%22%3A%22file%22%7D%2C%22pos%22%3A%7B%22line%22%3A44%2C%22character%22%3A97%7D%7D%5D%5D "Go to definition") with the following content:

```java

package com.example.controller;

import com.example.model.City;

import com.example.service.CityService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/cities")

public class CityController {

@Autowired

private CityService cityService;

@GetMapping

public List<City> getAllCities() {

return cityService.getAllCities();

}

@GetMapping("/{name}")

public City getCityByName(@PathVariable String name) {

return cityService.getCityByName(name);

}

}

```

### 12. Running the Application

To run the application, use the following command:

```sh

gradle bootRun

```

### 13. Testing the Endpoints

You can test the endpoints using tools like Postman or curl.

- Get all cities: [`GET http://localhost:8080/api/cities`](command:\_github.copilot.openSymbolFromReferences?%5B%22GET%20http%3A%2F%2Flocalhost%3A8080%2Fapi%2Fcities%22%2C%5B%7B%22uri%22%3A%7B%22%24mid%22%3A1%2C%22fsPath%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22external%22%3A%22file%3A%2F%2F%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22scheme%22%3A%22file%22%7D%2C%22pos%22%3A%7B%22line%22%3A14%2C%22character%22%3A13%7D%7D%5D%5D "Go to definition")

- Get a city by name: [`GET http://localhost:8080/api/cities/{name}`](command:\_github.copilot.openSymbolFromReferences?%5B%22GET%20http%3A%2F%2Flocalhost%3A8080%2Fapi%2Fcities%2F%7Bname%7D%22%2C%5B%7B%22uri%22%3A%7B%22%24mid%22%3A1%2C%22fsPath%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22external%22%3A%22file%3A%2F%2F%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22path%22%3A%22%2FUsers%2F2205287%2FDocuments%2FHackathon%2Fmy-spring-boot-app%2FREADME.md%22%2C%22scheme%22%3A%22file%22%7D%2C%22pos%22%3A%7B%22line%22%3A14%2C%22character%22%3A13%7D%7D%5D%5D "Go to definition")

This setup provides a basic Spring Boot application with a REST controller and JPA method for getting city information from MongoDB.

**Prompt :**

**@workspace /new React framework code Create responsive UI to display weather information by accepting cities as dropdown from DB (Pull Cities from API Call). Based on city, weather information will be displayed**

**2 pages or sections: 1 for current weather and 1 for forecast**

**Current Weather info fields (min 3): Temperature, wind speed and rain forecast etc**

**Forecast weather: hourly or in a day, few hours break up of forecasting info with appropriate icons or images.**

**Validation of data should be done.**