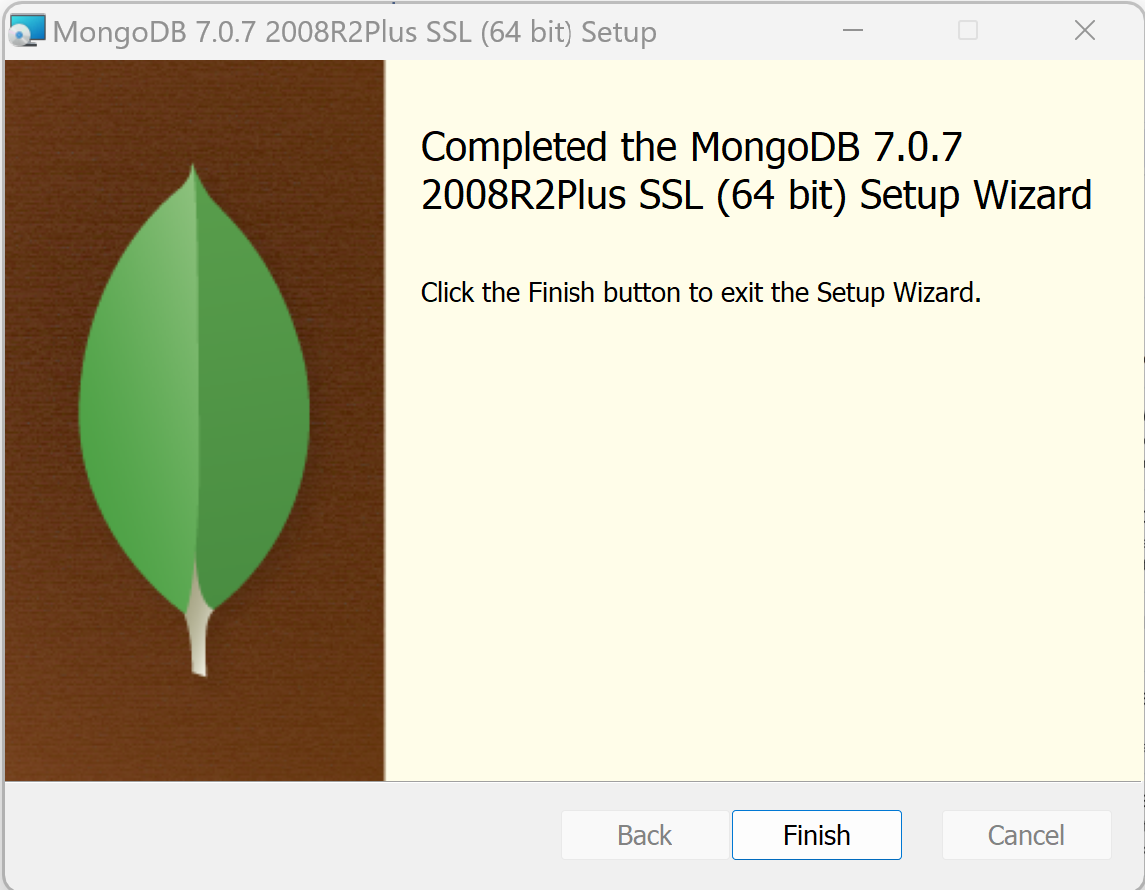
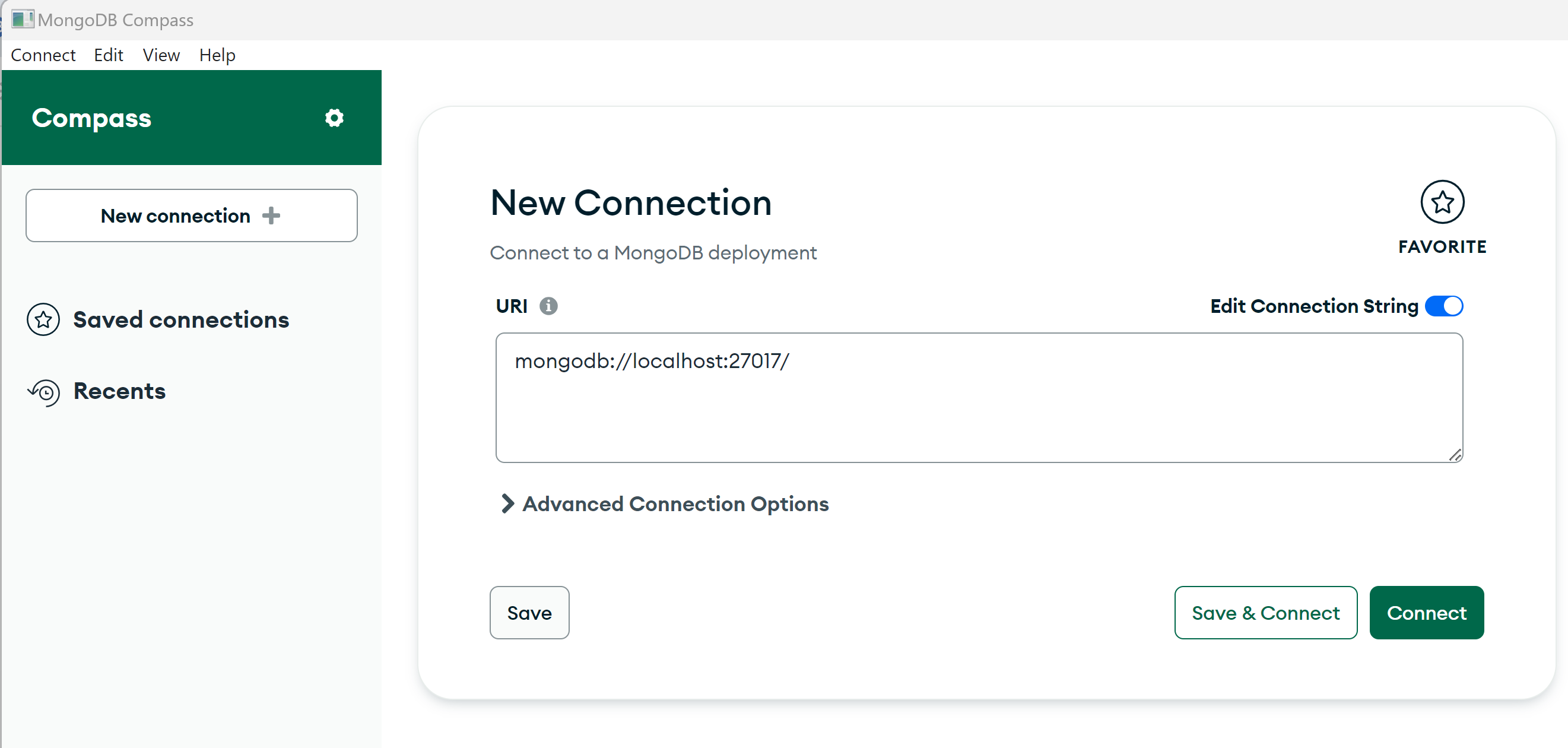
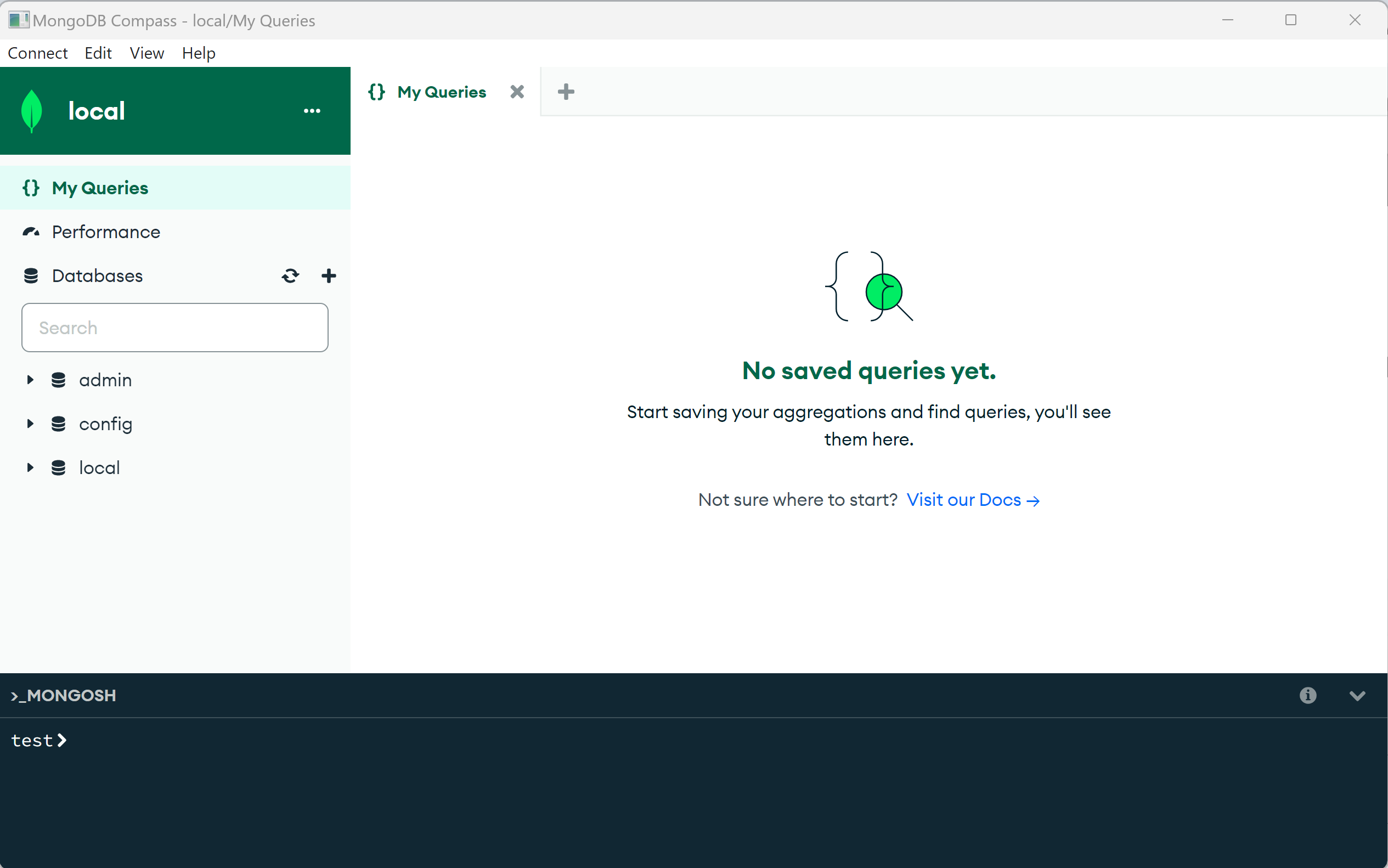
## Install MongoDB



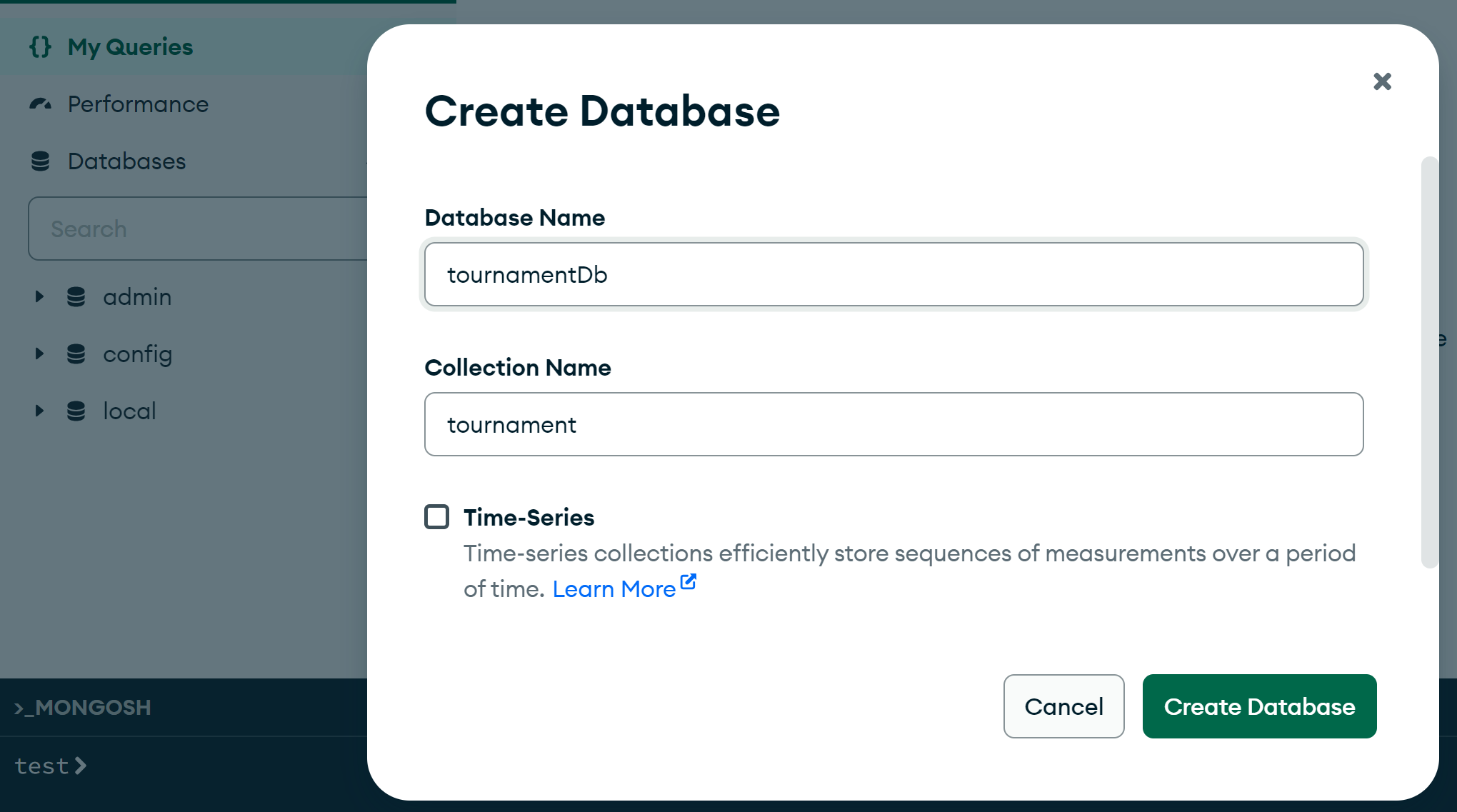
## MongoDB Compass Explorer



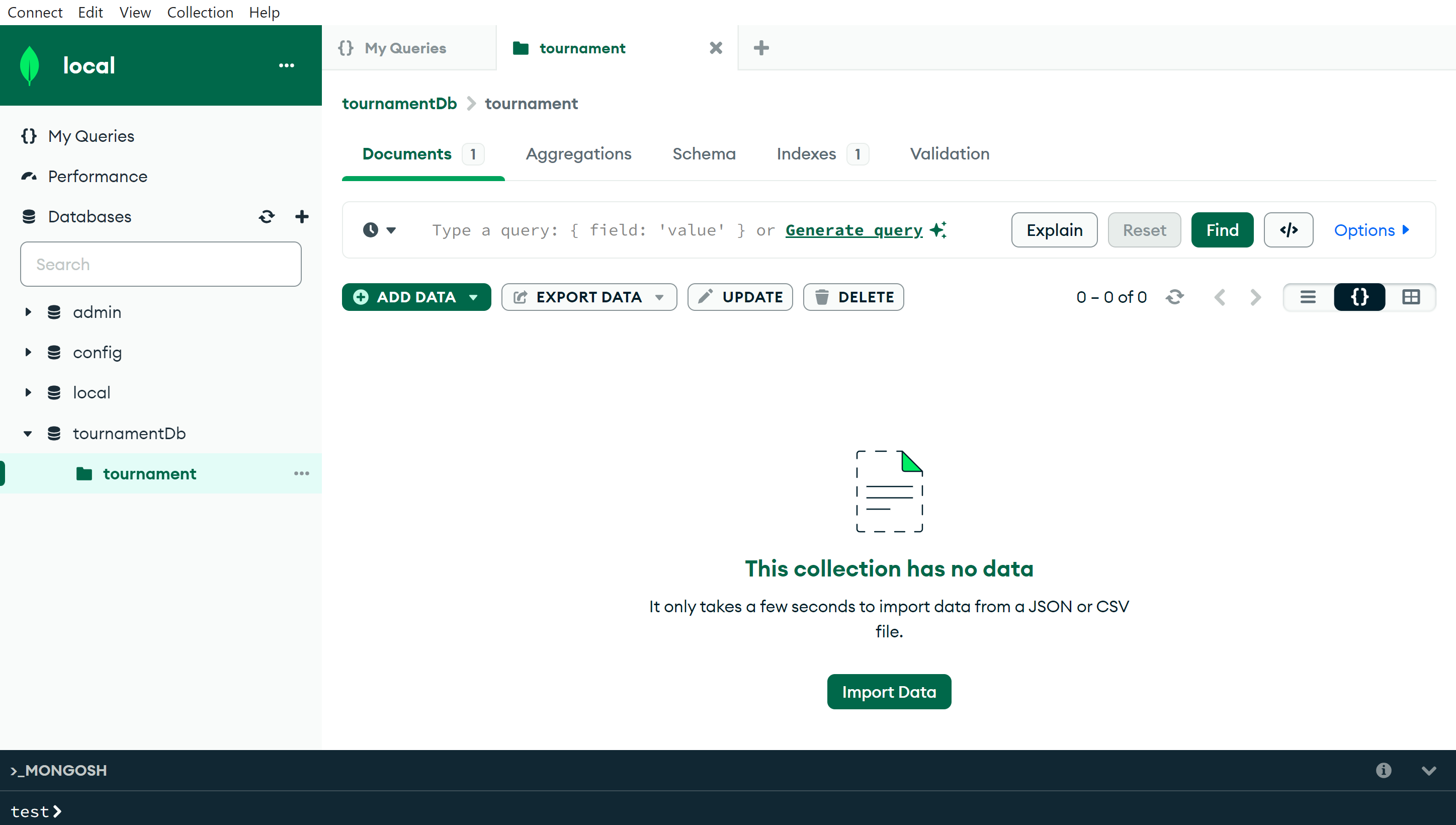
Save connection name = « local »



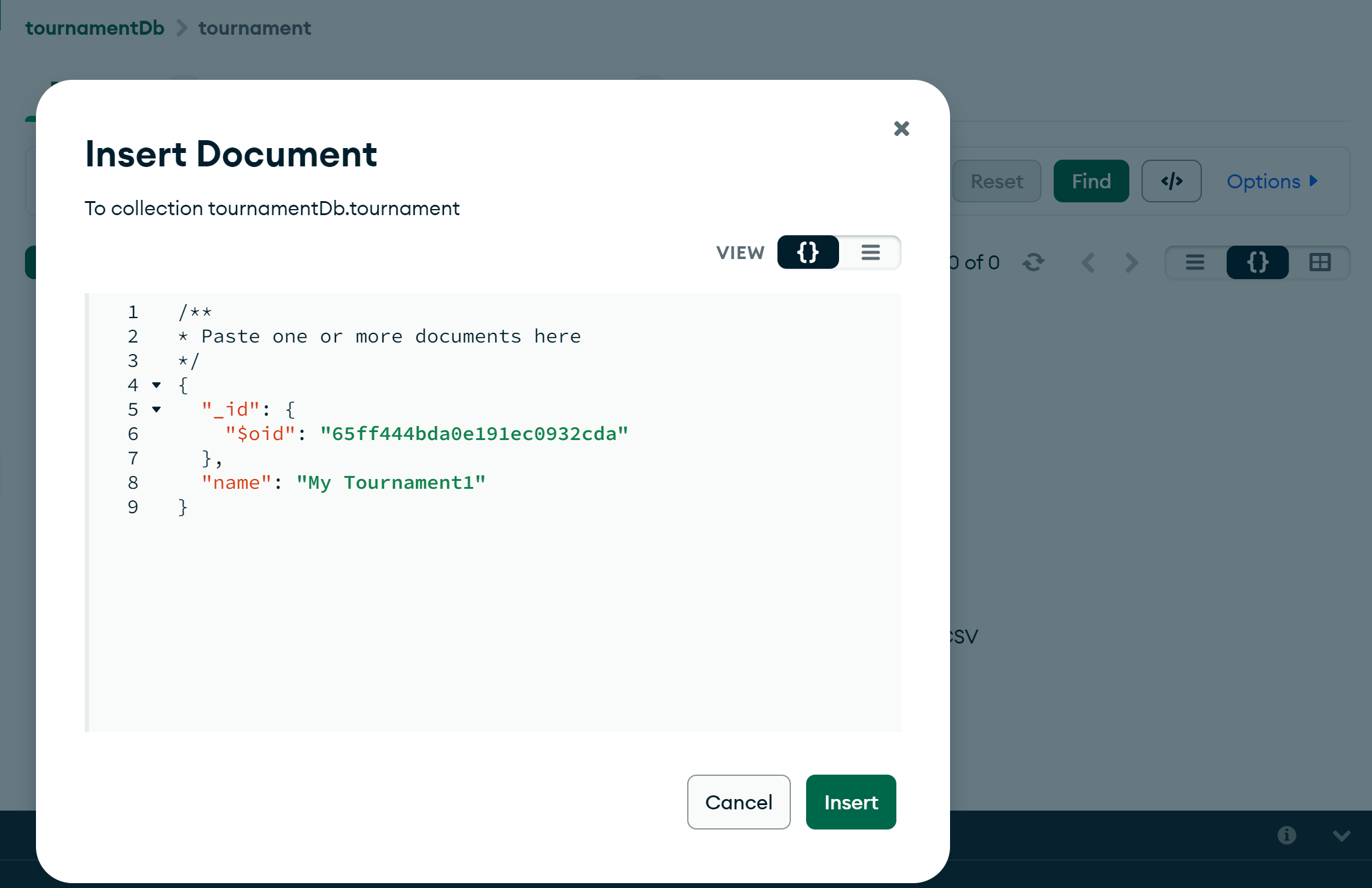
Create DB « tournamentDb », collection : « tournament »



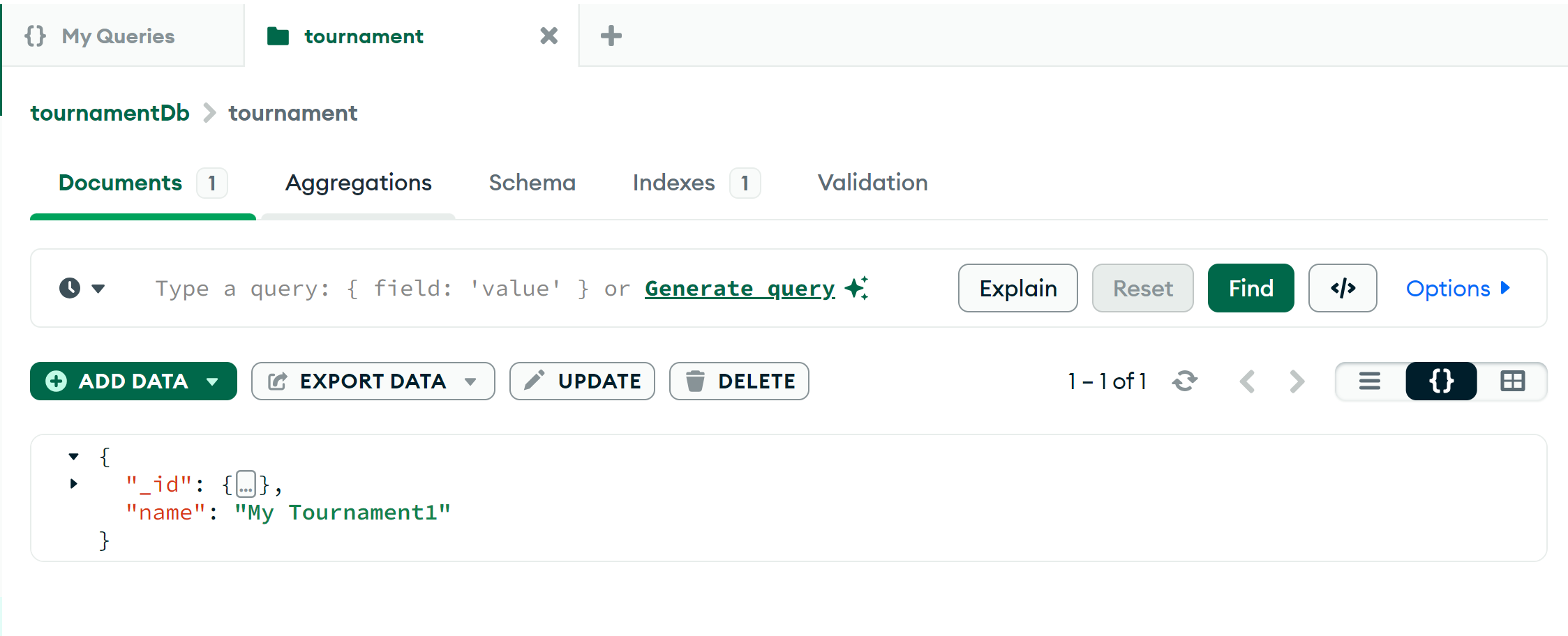
List (empty) document in collection



Create document « tournament », «name » : « My Tournament1 »

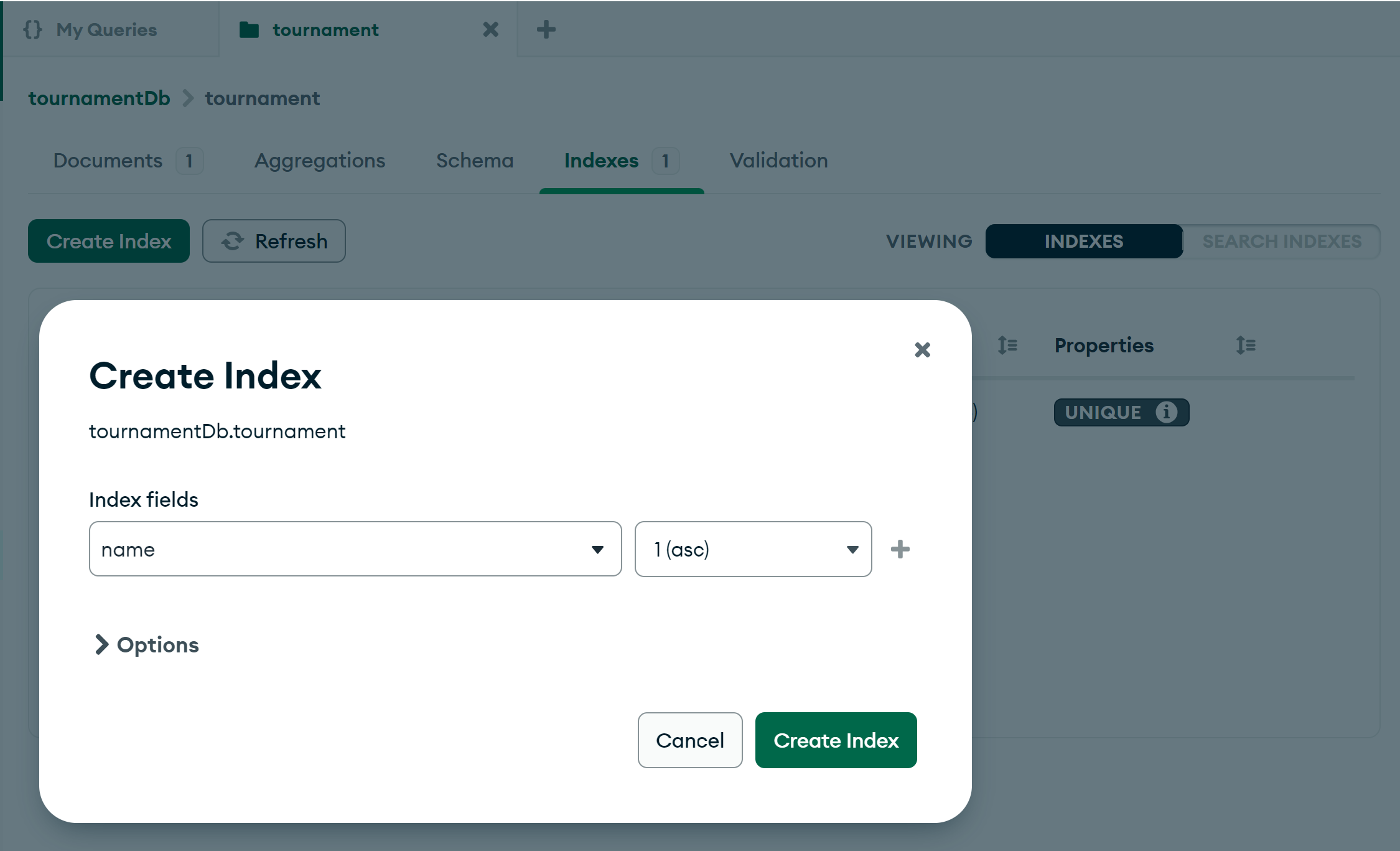


List document(s)

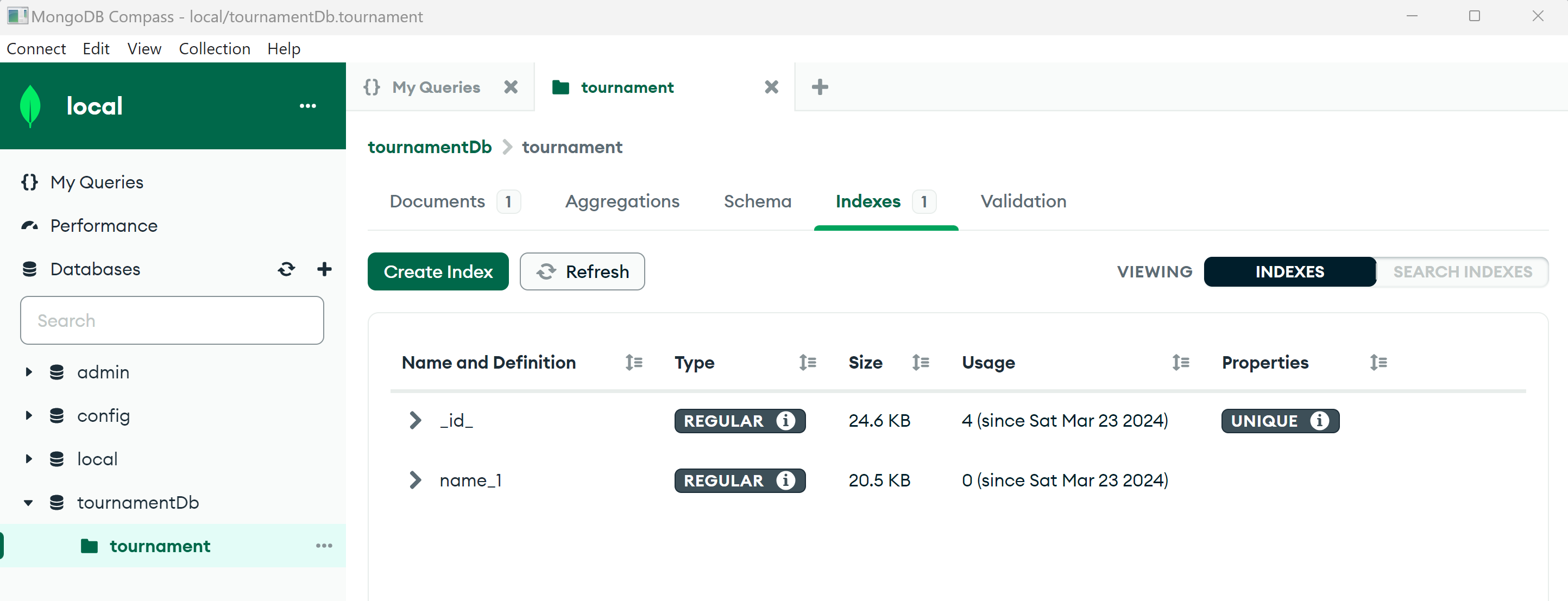


## Index

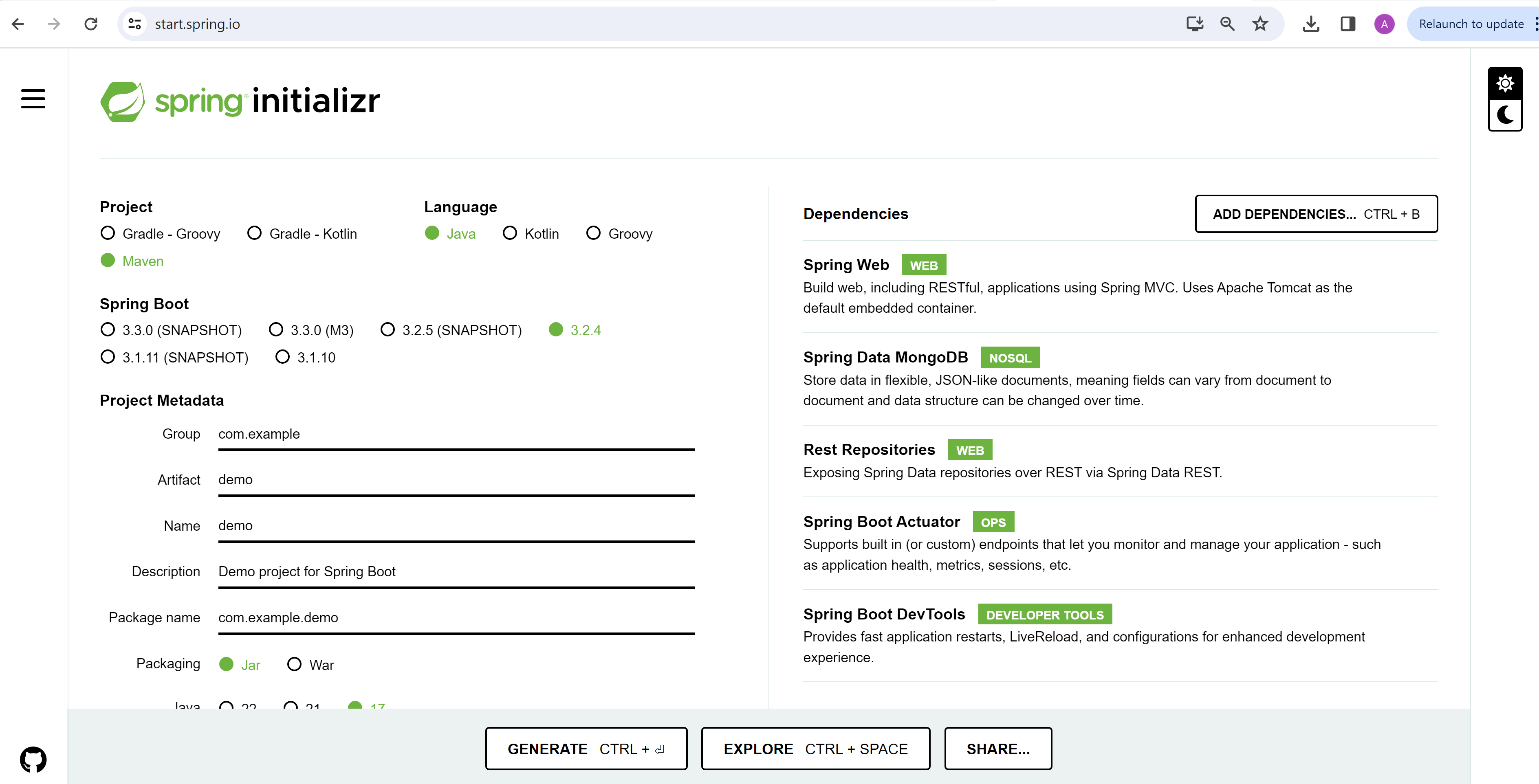
Create Index by « name »



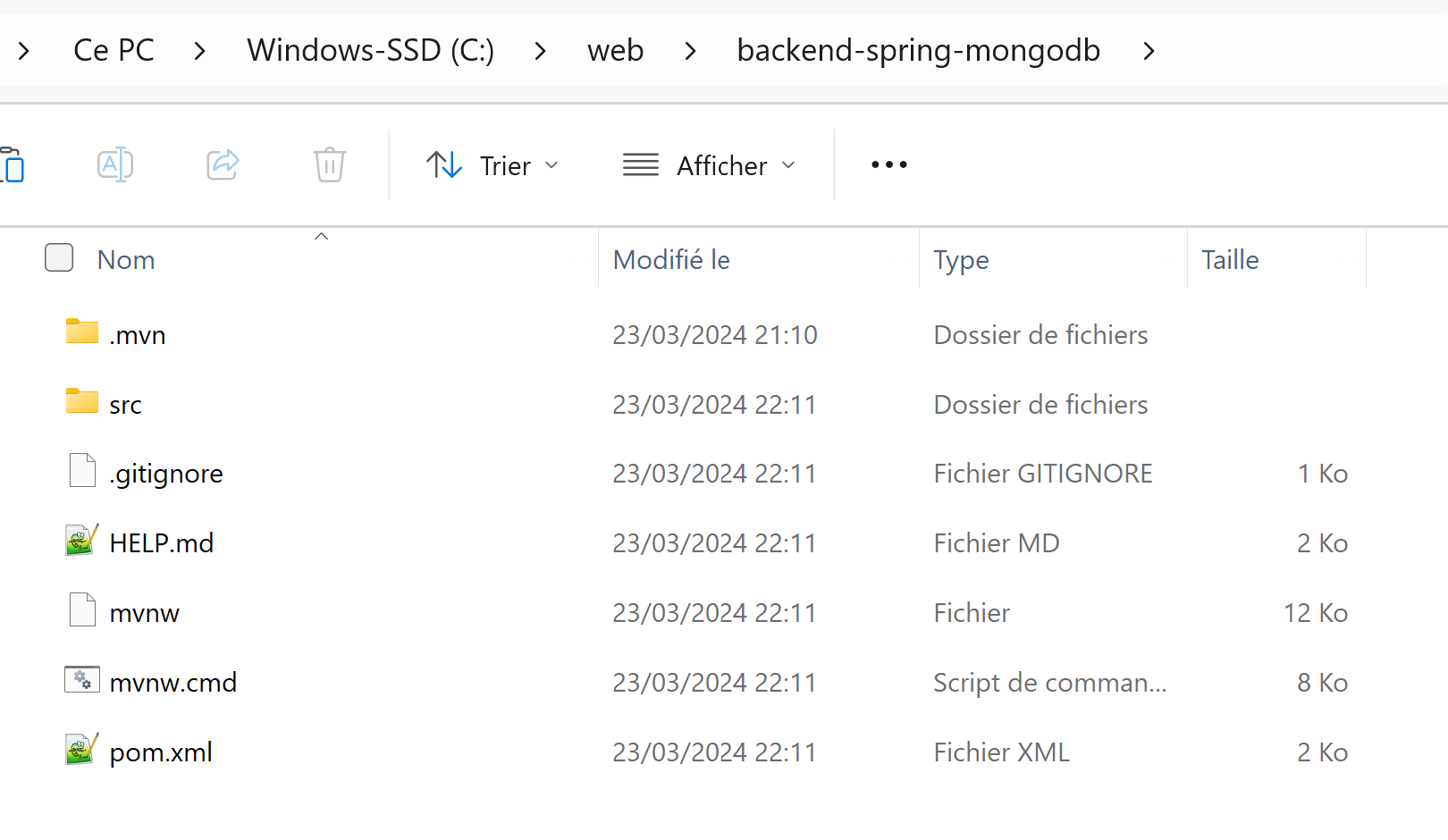
List Indexes (by « id » + by « name»)



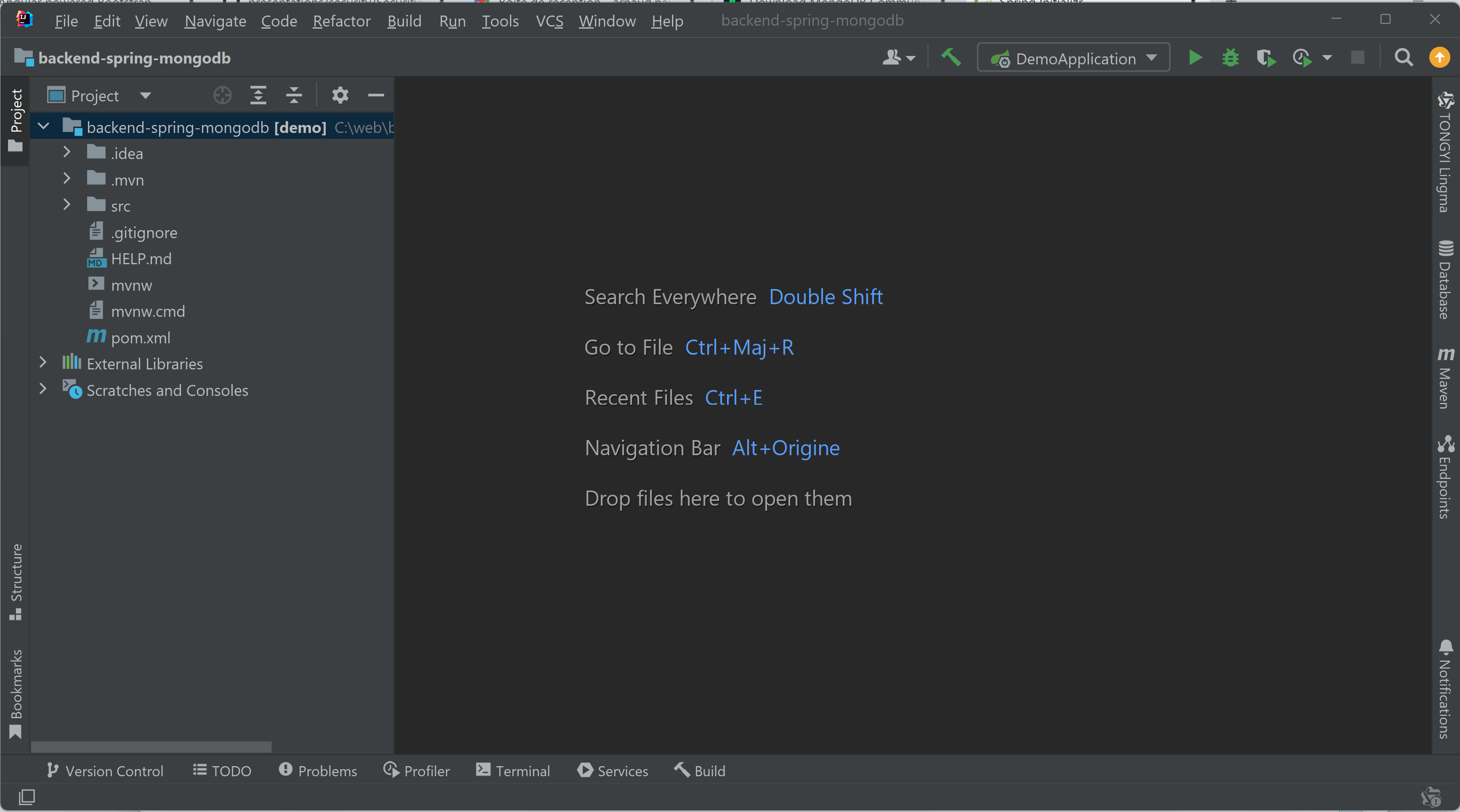
## SpringInitializr



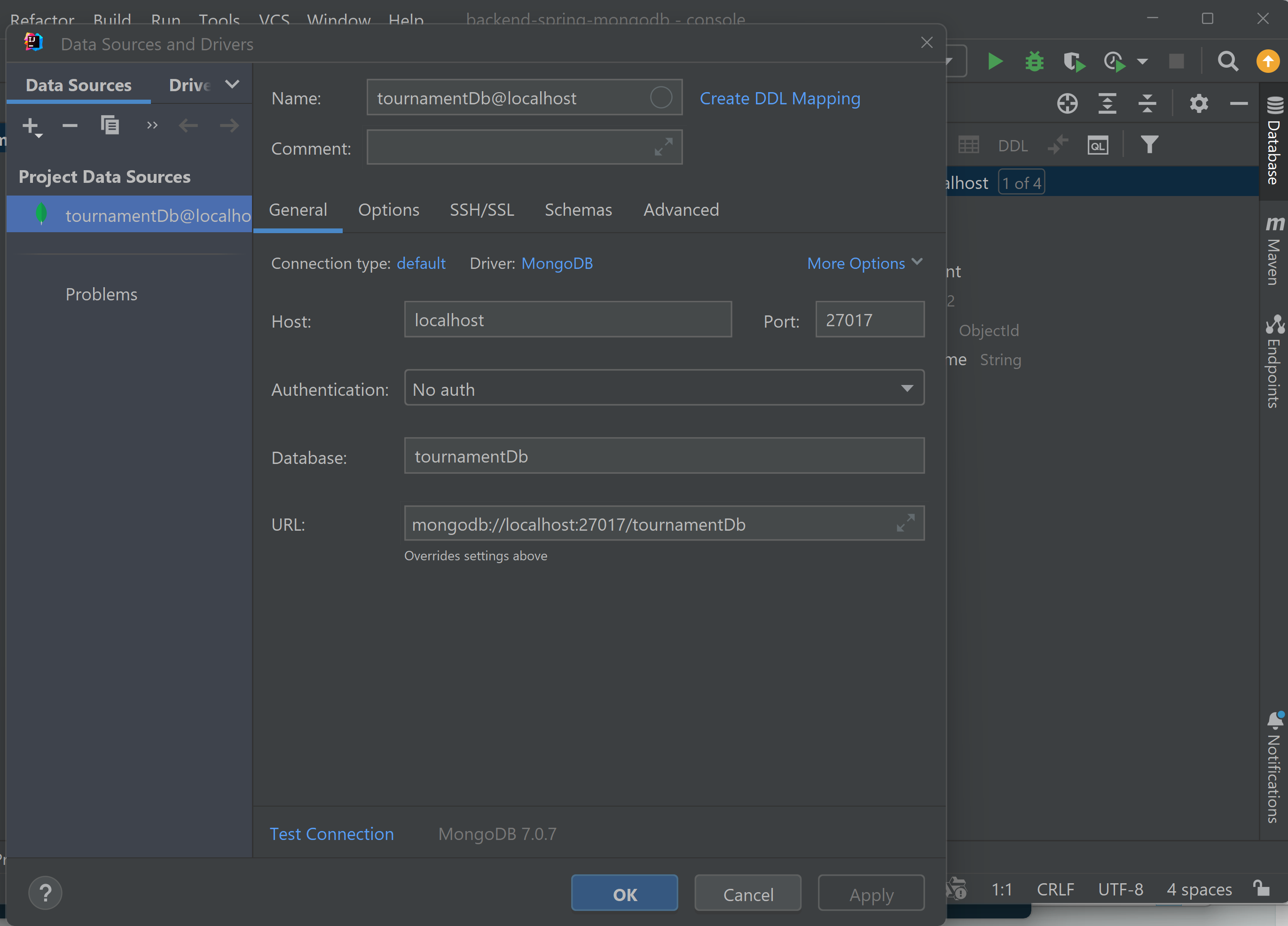
Click “Generate” ( Download zip), then unzip (and rename dir “backend-spring-mondb”)



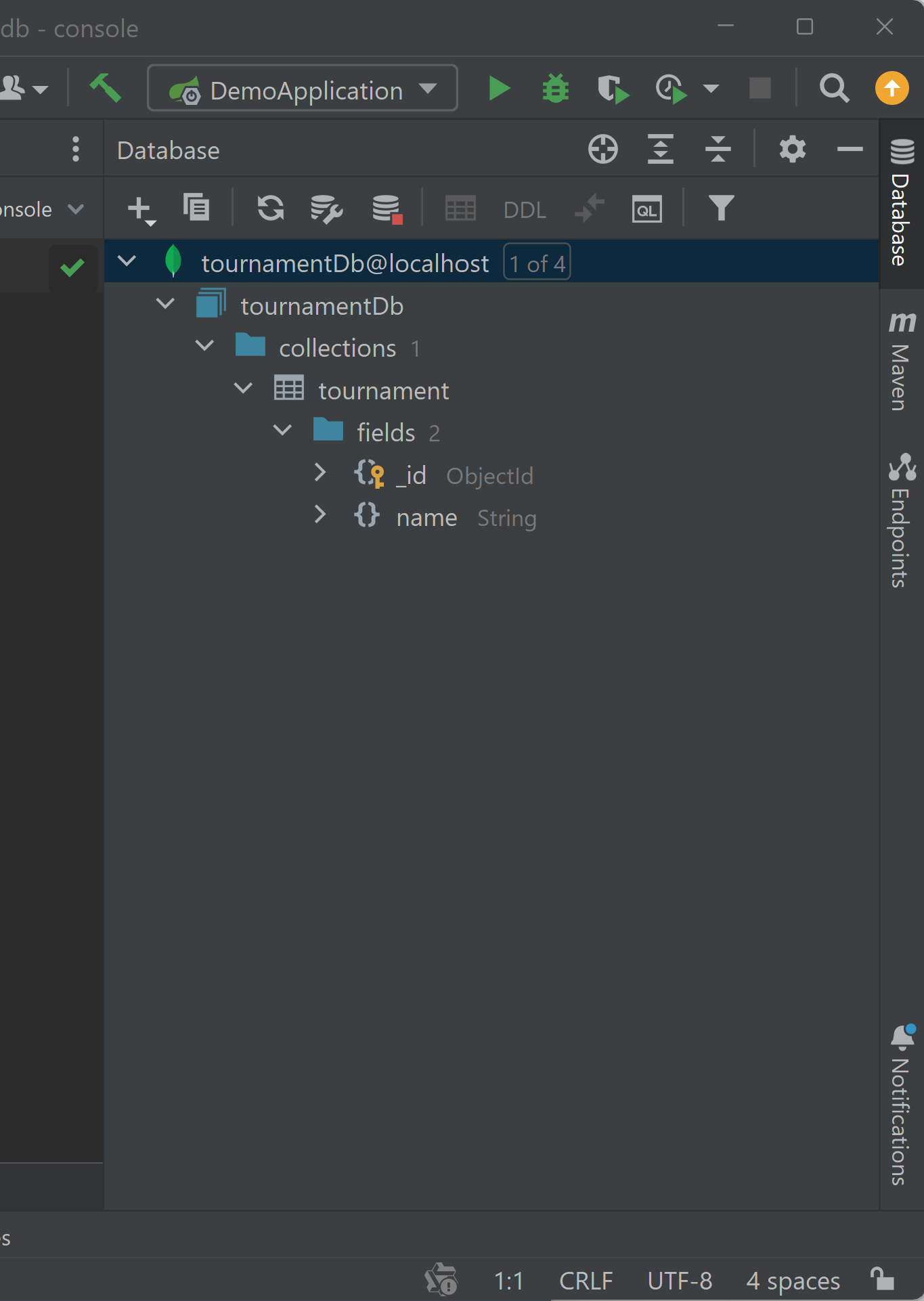
## Import as Maven project in IntelliJ



IntelliJ Database Explorer



Database -> Collection -> schema visible in IntelliJ



## Write Entity class

package com.example.demo.domain;  
  
import org.springframework.data.annotation.Id;  
import org.springframework.data.mongodb.core.mapping.Document;  
  
import java.time.LocalDate;  
  
@Document  
public class Tournament {  
  
 @Id  
 public String id;  
  
 public String name;  
  
 public LocalDate startDate;  
 public LocalDate endDate;  
  
}

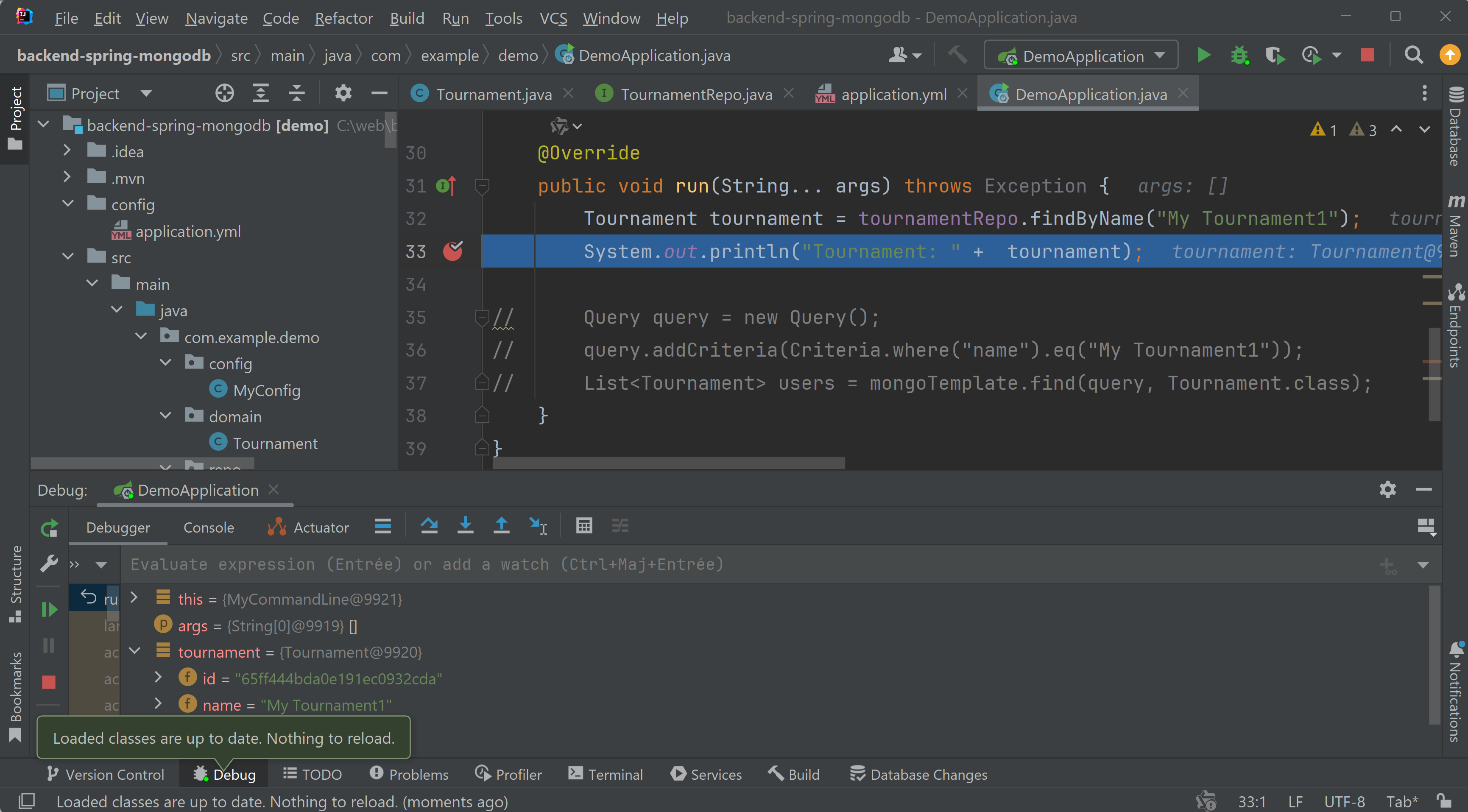
## Write Repository ( = CRUD)

package com.example.demo.repo;  
  
import com.example.demo.domain.Tournament;  
import org.springframework.data.mongodb.repository.MongoRepository;  
  
public interface TournamentRepo extends MongoRepository<Tournament,String> {  
  
 Tournament findByName(String name);  
  
}

## Adding config/application.yml

spring:  
 data:  
 mongodb:  
 host: localhost  
 port: 27017  
 database: tournamentDb

## Launching App (Debug)



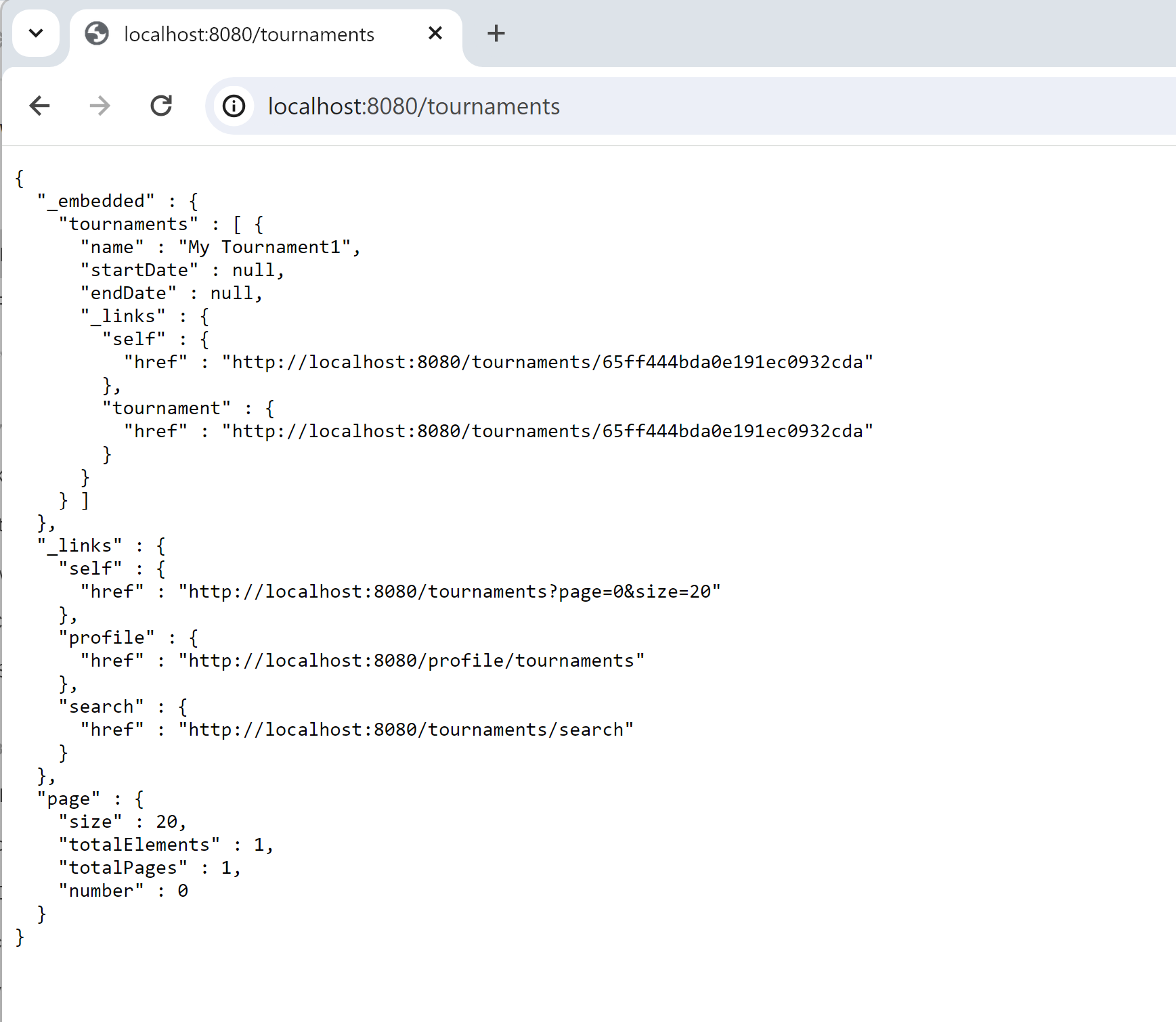
## Adding default CRUD Rest Api (not recommended, mostly for debug)

Edit repository class, add

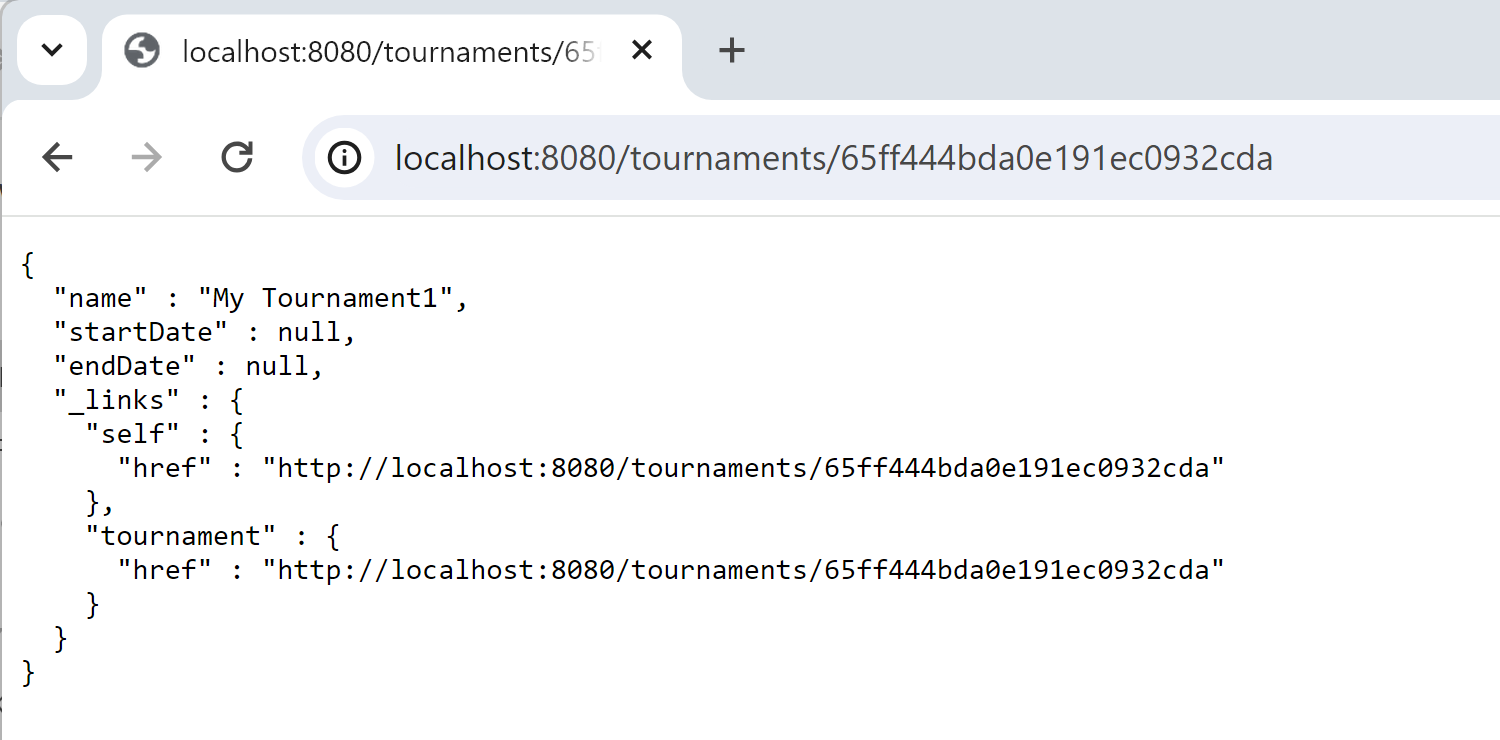
import org.springframework.data.rest.core.annotation.RepositoryRestResource;  
  
@RepositoryRestResource(path = "tournaments")

Relaunch server

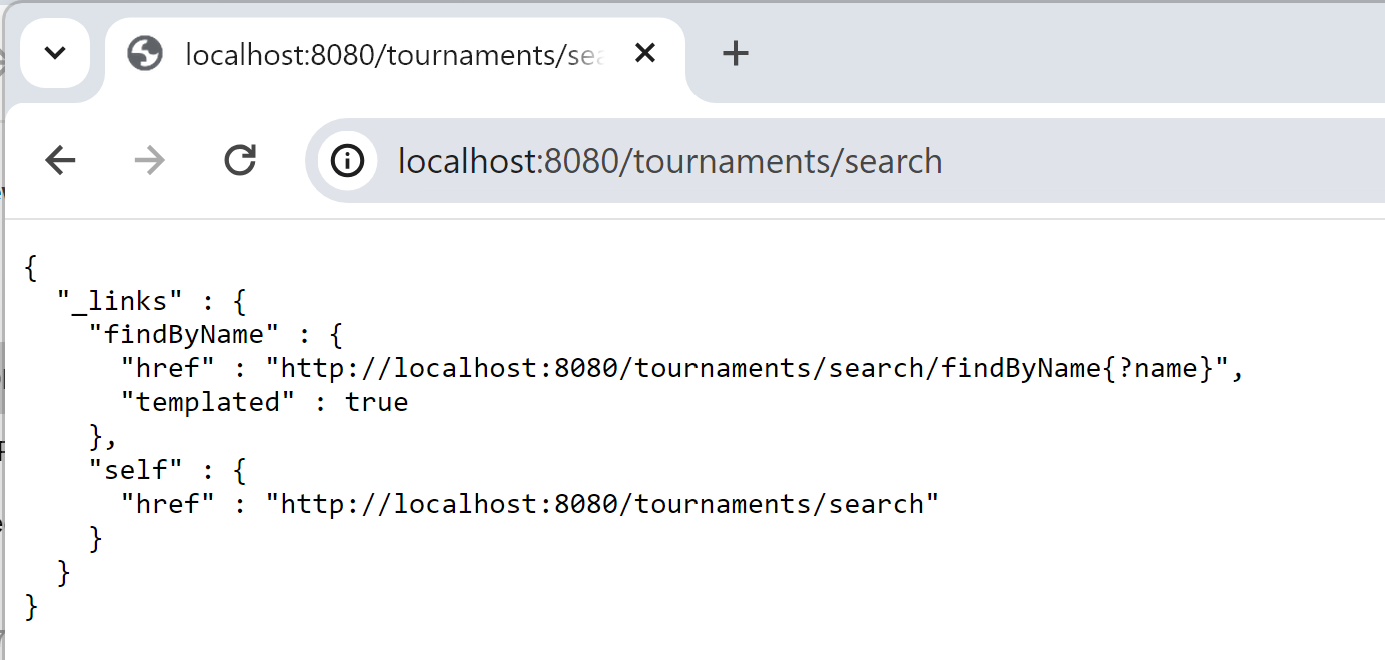
Test find all “GET /tournaments” in Browser



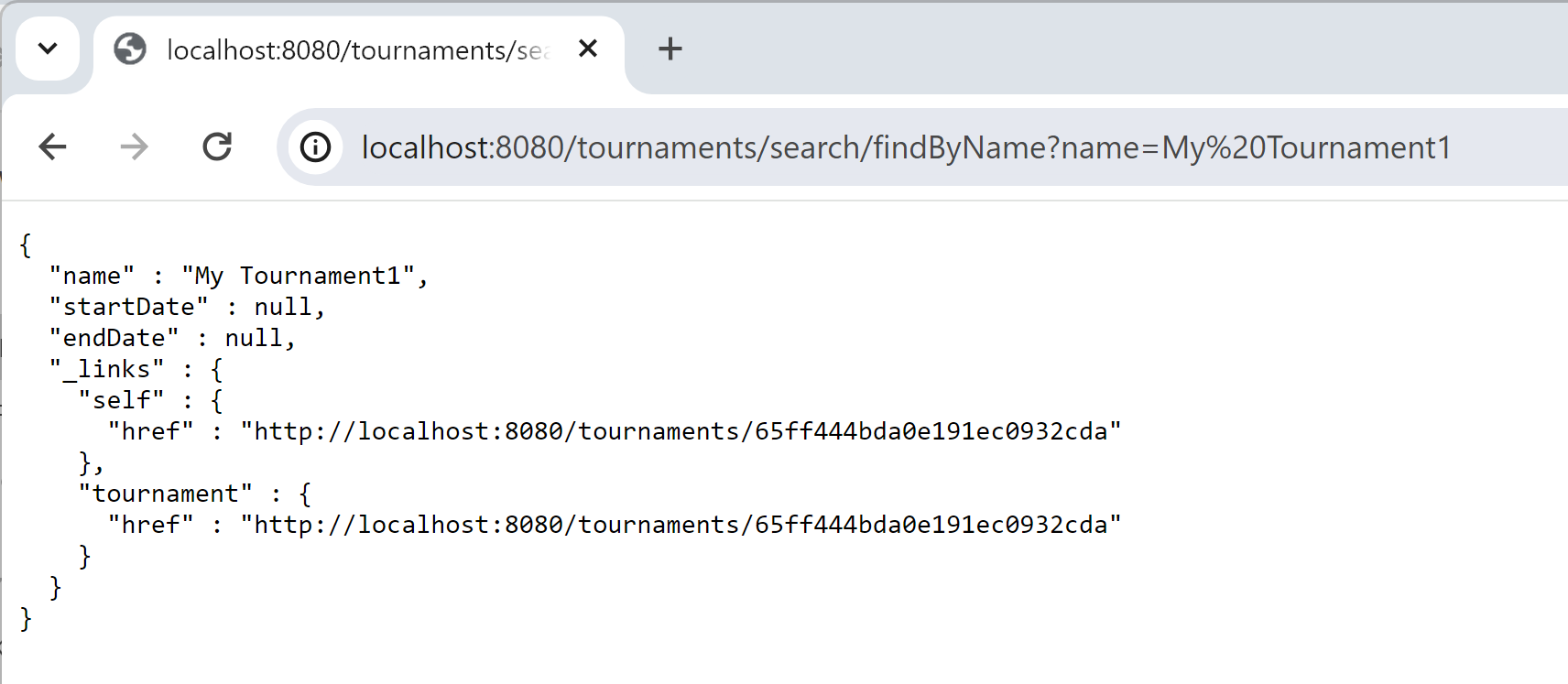
Test find By Id : “GET /tournaments/65ff444bda0e191ec0932cda“



All Searches from repository queries



Search by name



## Custom Rest API (business API code)

Internal rules for code:

1/ all rest controller methods should have 1 line of code, by delegating to a @Service class

2/ PUT and POST rest controller methods should have 1 @RequestBody param using a “RequestDTO” class, and return a “ResponseDTO” class

3/ all service methods should consist of 3 steps:

// step 1/3: unmarshall, check inputs  
…

// step 2/3: business code  
…

// step 3/3: marshall output (DTO, not internal entity)

…

Create a Rest controller

package com.example.demo.rest;  
  
import com.example.demo.rest.dto.TournamentCreateRequestDTO;  
import com.example.demo.rest.dto.TournamentCreateResponseDTO;  
import com.example.demo.service.TournamentService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
@RequestMapping("/api/tournament")  
public class TournamentRestController {  
  
 @Autowired  
 private TournamentService tournamentService;  
  
 @PostMapping()  
 public TournamentCreateResponseDTO createTournament(  
 @RequestBody TournamentCreateRequestDTO req) {  
 return tournamentService.createTournament(req);  
 }  
}

Create the corresponding Service class

package com.example.demo.service;  
  
import com.example.demo.domain.Tournament;  
import com.example.demo.repo.TournamentRepo;  
import com.example.demo.rest.dto.TournamentCreateRequestDTO;  
import com.example.demo.rest.dto.TournamentCreateResponseDTO;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.time.LocalDate;  
import java.time.LocalDateTime;  
import java.util.Objects;  
  
@Service  
@Transactional  
public class TournamentService {  
  
 @Autowired  
 private TournamentRepo tournamentRepo;  
  
 public TournamentCreateResponseDTO createTournament(  
 TournamentCreateRequestDTO req) {  
 // step 1/3: unmarshall, check inputs  
 String name = Objects.*requireNonNull*(req.name);  
 if (name.length() < 3) {  
 throw new IllegalArgumentException("name too short");  
 }  
 Tournament alreadyFound = tournamentRepo.findByName(name);  
 if (alreadyFound != null) {  
 throw new IllegalArgumentException("name already used");  
 }  
  
 // step 2/3: business code  
 // OK, create  
 Tournament entity = new Tournament();  
 entity.name = name;  
 entity.createdAt = LocalDateTime.*now*();  
 entity.createdBy = "<<currentUser>>"; // security not impl yet  
 entity = tournamentRepo.save(entity);  
  
 // step 3/3: marshall output (DTO, not internal entity)  
 TournamentCreateResponseDTO res = toDto(entity);  
 return res;  
 }  
  
 private TournamentCreateResponseDTO toDto(Tournament src) {  
 TournamentCreateResponseDTO res =  
 new TournamentCreateResponseDTO(  
 src.id, src.name, src.createdAt, src.createdBy);  
 return res;  
 }  
}

## Test using http client : Curl

$ curl -H "content-type: application/json" http://localhost:8080/api/tournament -d '{"name":"super to urnament"}'

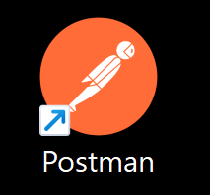
{"id":"65ff5a96e9a12719c3847643","name":"super tournament","createdDate":"2024-03-23T23:41:26.9612767","createdBy":"<<currentUser>>"}

Relaunch test … assume document NOT inserted twice with same name

$ curl -H "content-type: application/json" http://localhost:8080/api/tournament -d '{"name":"super to urnament"}'

{"timestamp":"2024-03-23T22:43:09.970+00:00","status":500,"error":"Internal Server Error","trace":"java.lang.IllegalArgumentException: name already used\r\n\tat com.example.demo.service.TournamentService.createTournament(TournamentService.java:31)\r\n\tat …

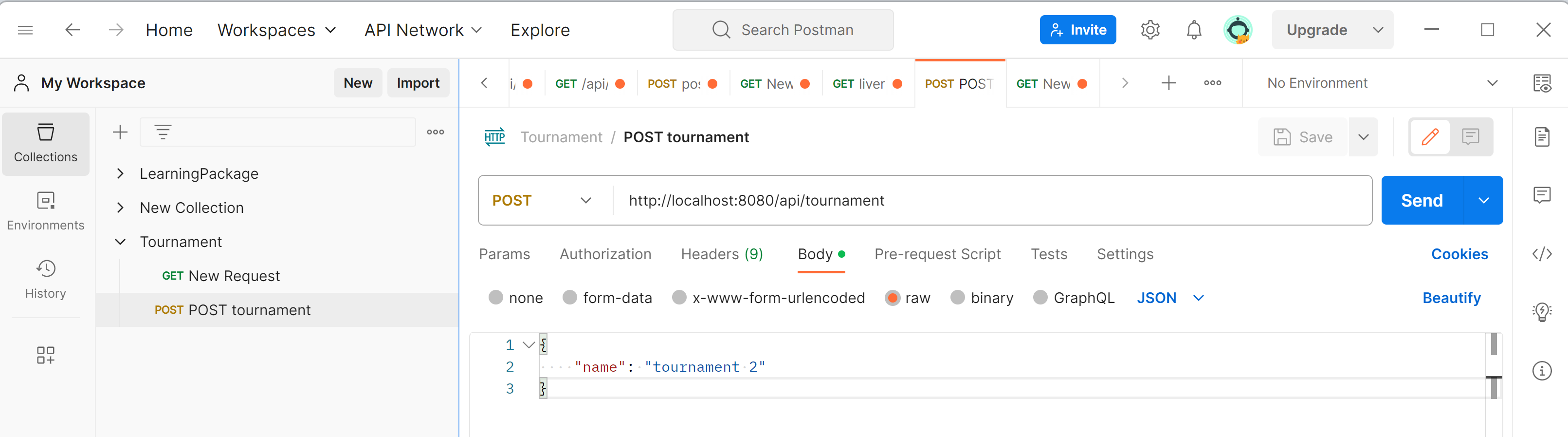
## Test using Postman



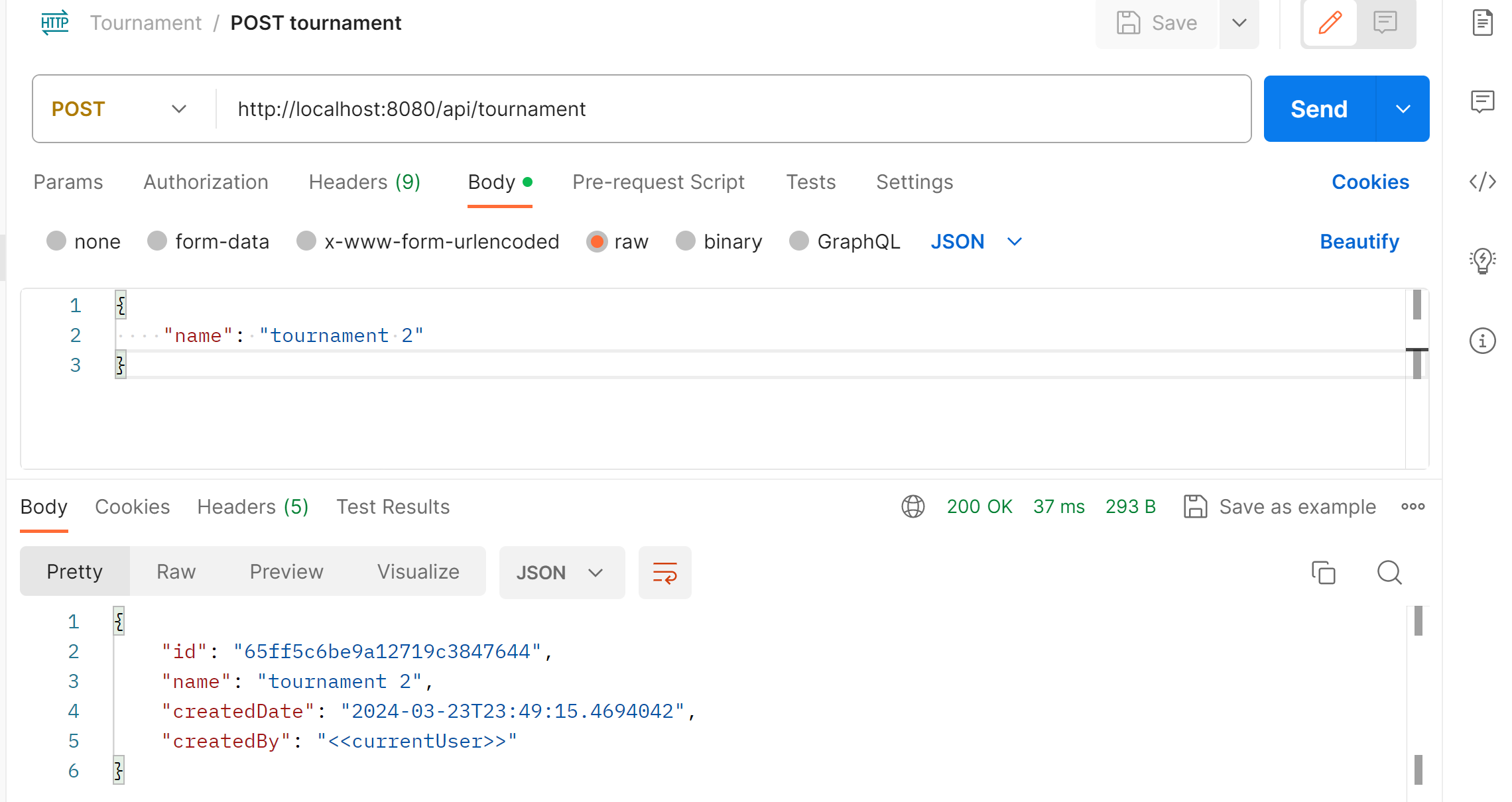
Create “Tournament” rest collection

Then “add request”

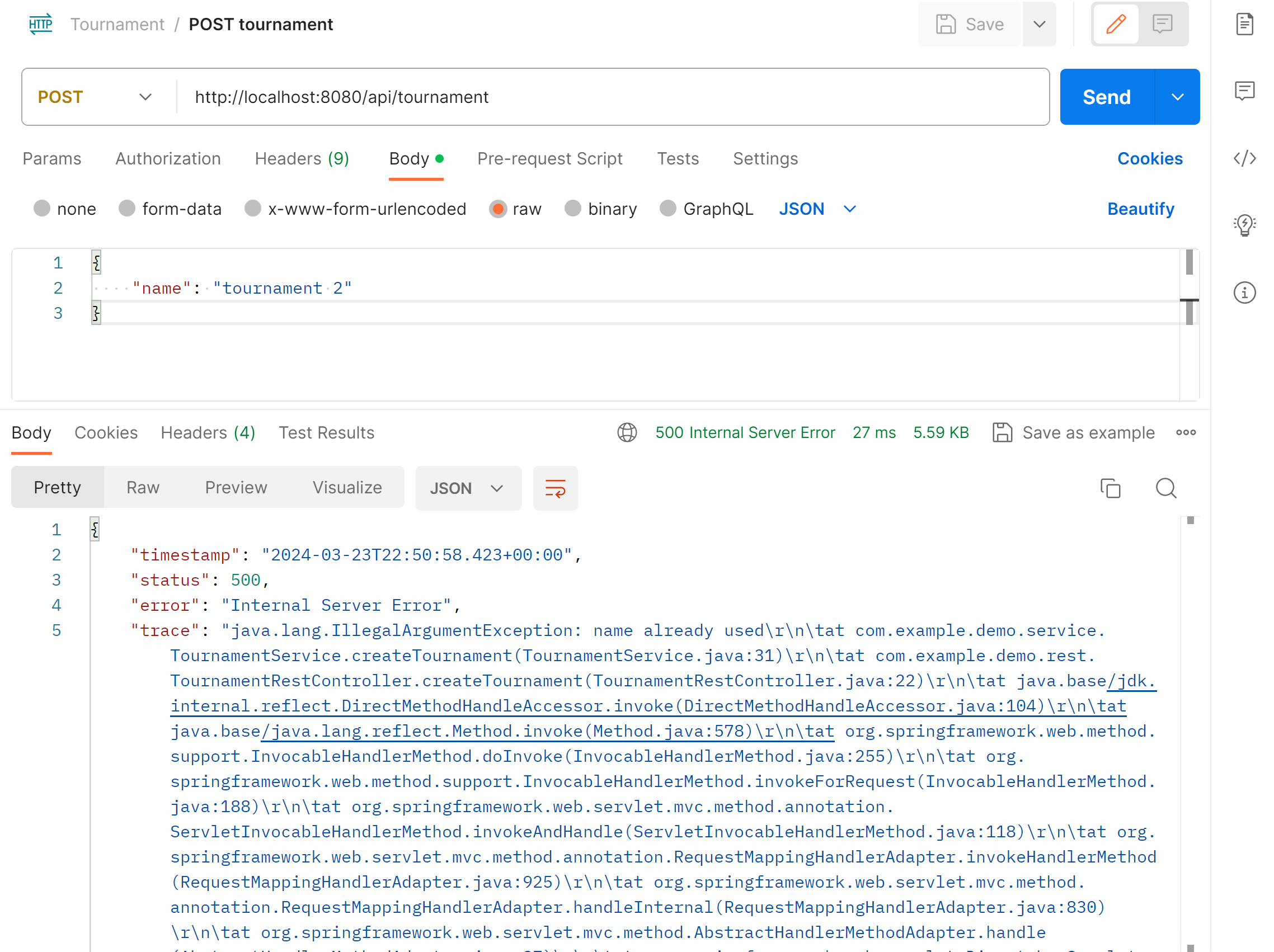
Edit, to use POST, url, body with type json



Launch request: click “Send”



Relaunch (expecting error… name already used)



## Adding support for OpenAPI (Swagger)

Step 1: edit pom.xml, add

<dependency>  
 <groupId>org.springdoc</groupId>  
 <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>  
 <version>2.1.0</version>  
</dependency>

WARN …. This used to work with springboot 2.\*, but not on 3.\*

<!--  
<dependency>  
 <groupId>org.springdoc</groupId>  
 <artifactId>springdoc-openapi-ui</artifactId>  
 <version>1.7.0</version>  
</dependency>  
-->

Step 2:

Edit main application class, add

@OpenAPIDefinition(  
 info = @Info(title = "Tournament App API", version = "1.0",  
 description = "Rest API using OpenAPI 3 for a tutorial Tournament application"))

Step 3:

Edit your Rest Controller, add annotation @OpenAPIDefinition to class

import io.swagger.v3.oas.annotations.OpenAPIDefinition;

@OpenAPIDefinition(  
 // tags = { Tag("Tournament") }  
)

And optionally add annotation @Operation to methods:

import io.swagger.v3.oas.annotations.Operation;  
import io.swagger.v3.oas.annotations.responses.ApiResponse;  
import io.swagger.v3.oas.annotations.responses.ApiResponses;

@Operation(summary = "Create a new (unique by name) tournament")  
@ApiResponses(value = {  
 @ApiResponse(responseCode = "200", description = "Successful operation"),  
 @ApiResponse(responseCode = "500", description = "name already used"),  
})

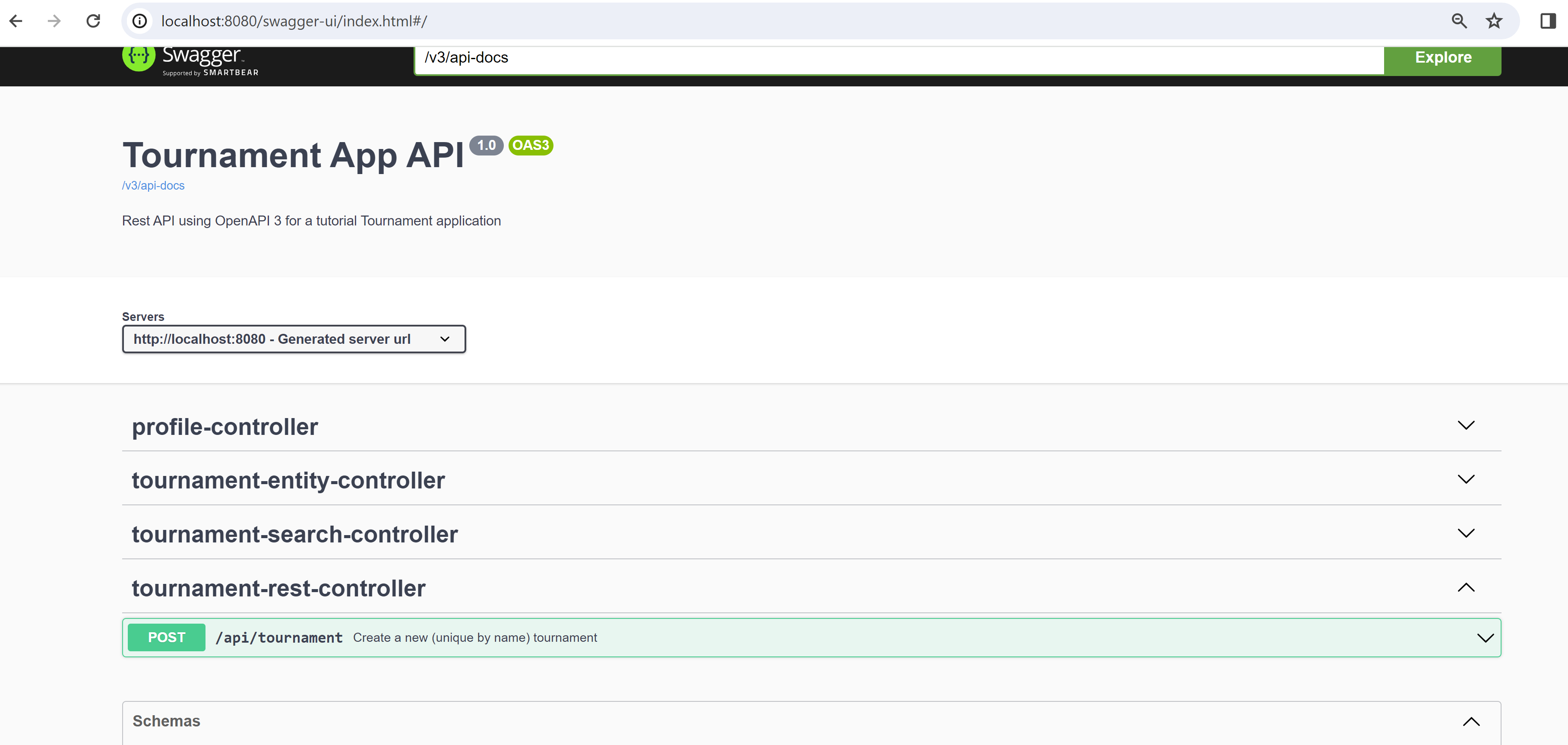
## Helper static Html page

Add file “src/main/resources/static/index.html”

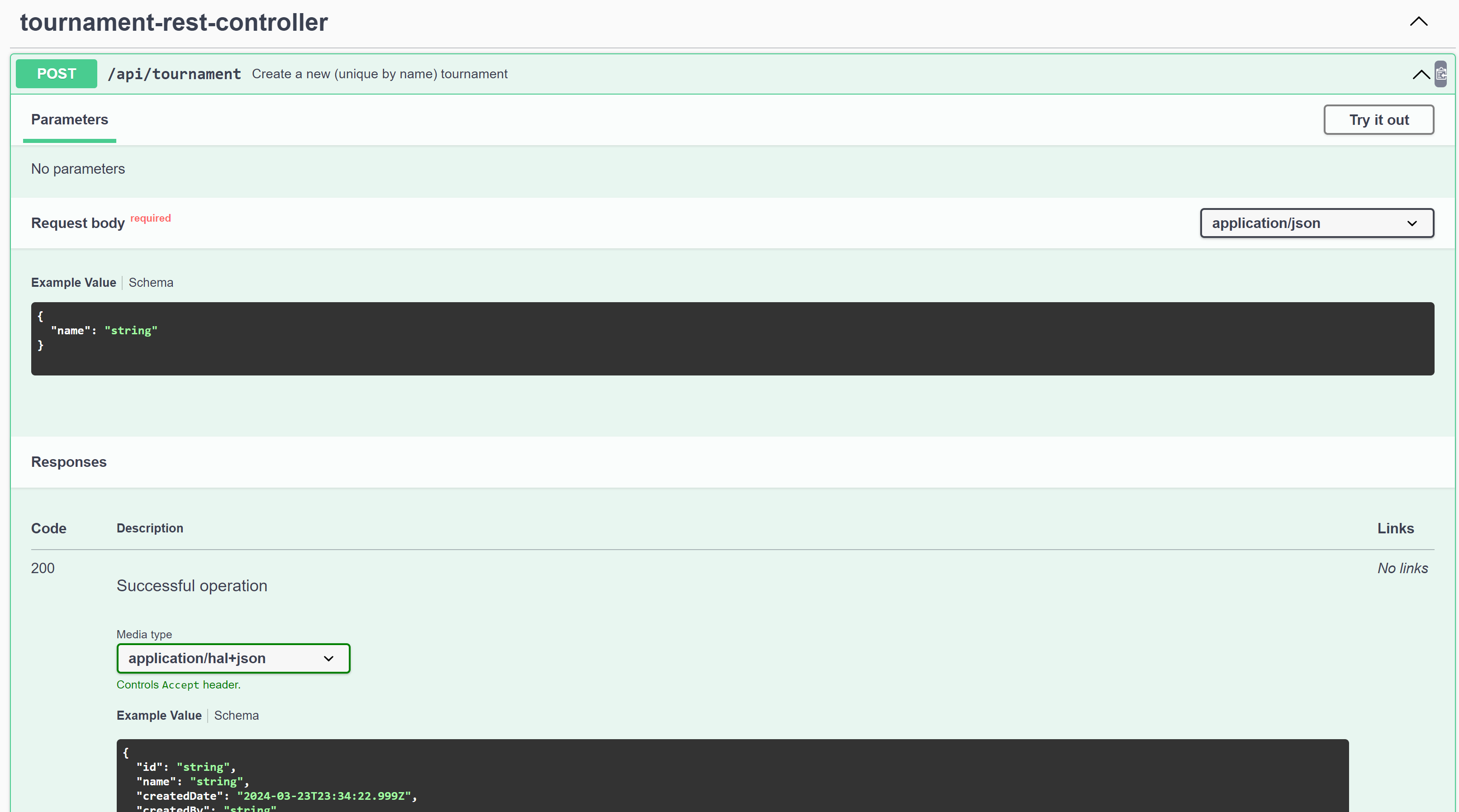
<html>  
<body>  
  
<H1>Test Tournament AApp (Springboot, Rest, MongoDB, OpenAPI)</H1>  
  
<A href="/swagger-ui.html">/swagger-ui.html</A>  
<br/>  
<A href="/v3/api-docs">/v3/api-docs</A>  
<br/>  
  
</body>

Relaunch, open <http://localhost:8080/swagger-ui.html>

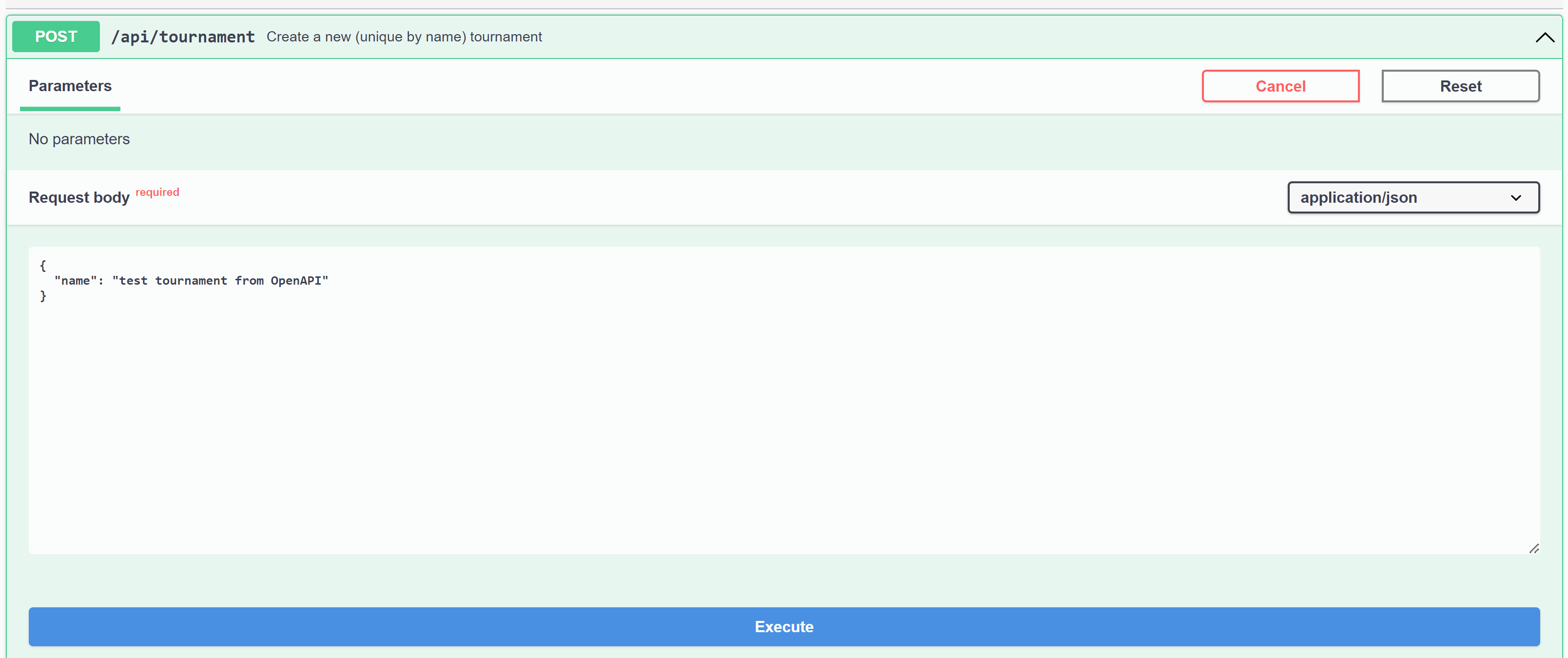
( This is redirected to <http://localhost:8080/swagger-ui/index.html> )



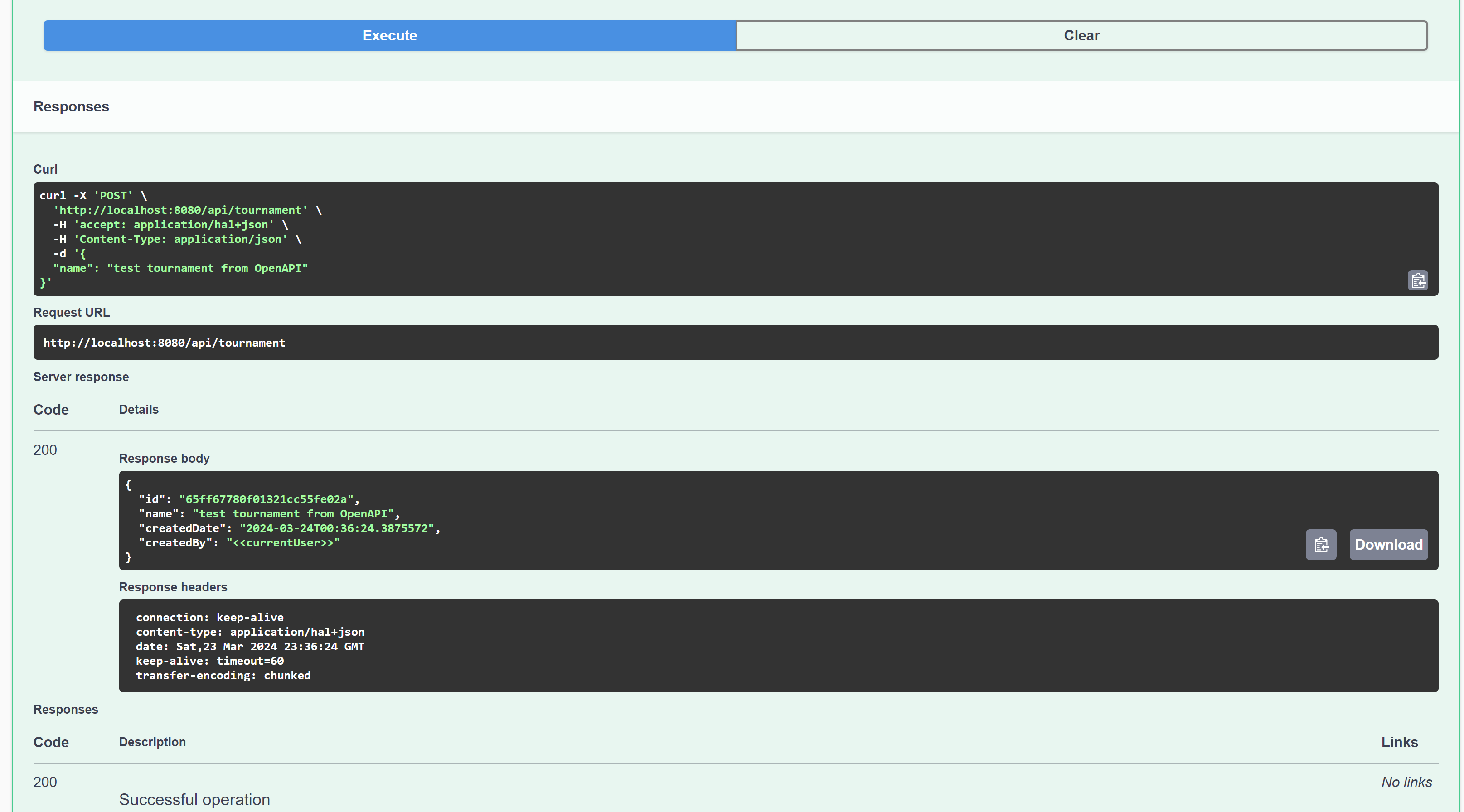
Expand custom method “POST /api/tournament”



Then click on “Try it out”, and fill request body



Click on “Execute”



JSON open-api doc

Open <http://localhost:8080/v3/api-docs>



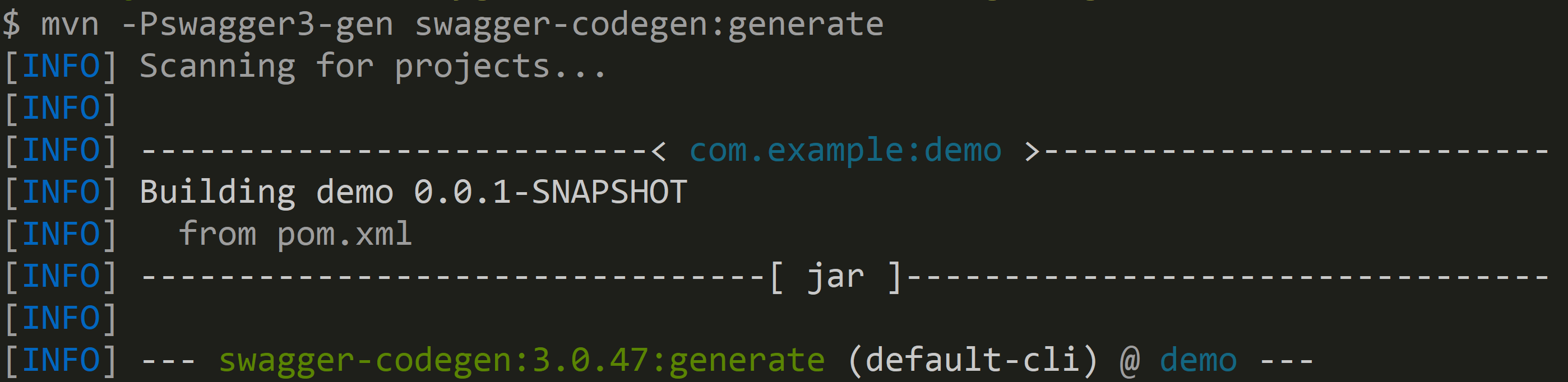
## Generate Client code from OpenAPI for Angular

Edit pom.xml, add

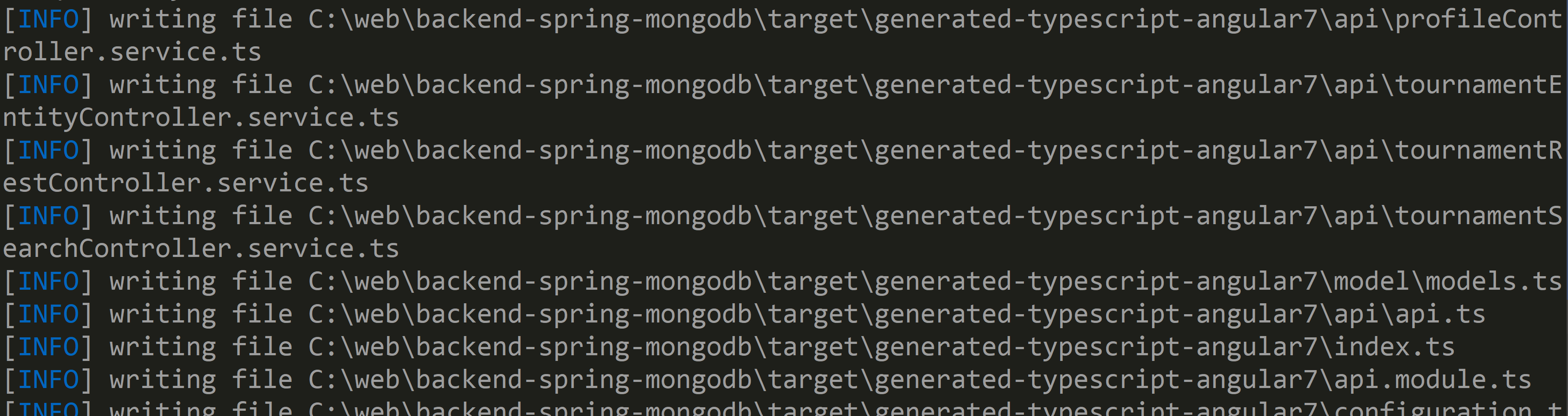
<profiles>  
 <profile>  
 <id>swagger3-gen</id>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>io.swagger.codegen.v3</groupId>  
 <artifactId>swagger-codegen-maven-plugin</artifactId>  
 <version>3.0.47</version>  
 <configuration>  
 <inputSpec>http://localhost:8080/v3/api-docs</inputSpec>  
 <language>typescript-angular</language>  
 <output>${basedir}/target/generated-typescript-angular7</output>  
 <configOptions>  
 <ngVersion>17.0.0</ngVersion>  
 </configOptions>  
 </configuration>  
 <executions>  
 <execution>  
 <id>generate-swagger-typescript-angular-7</id>  
 <phase>generate-sources</phase>  
 <goals>  
 <goal>generate</goal>  
 </goals>  
 </execution>  
 </executions>  
 </plugin>  
 </plugins>  
 </build>  
 </profile>  
  
</profiles>

Then execute in terminal

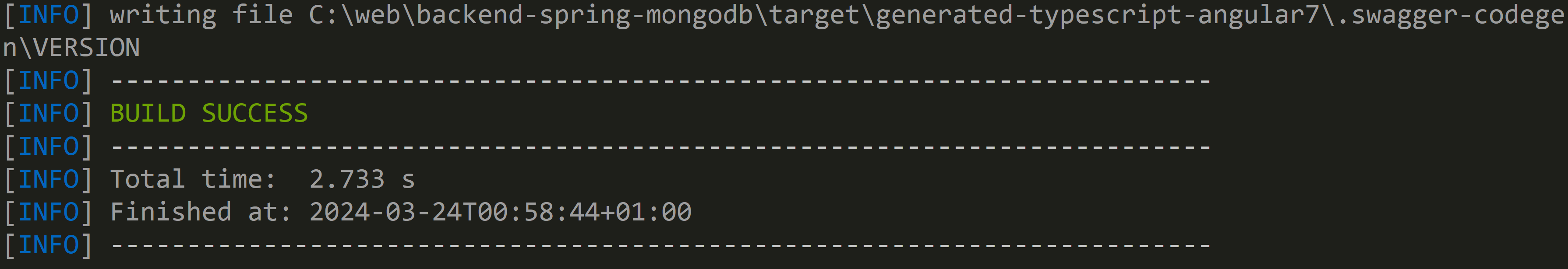
mvn -Pswagger3-gen swagger-codegen:generate



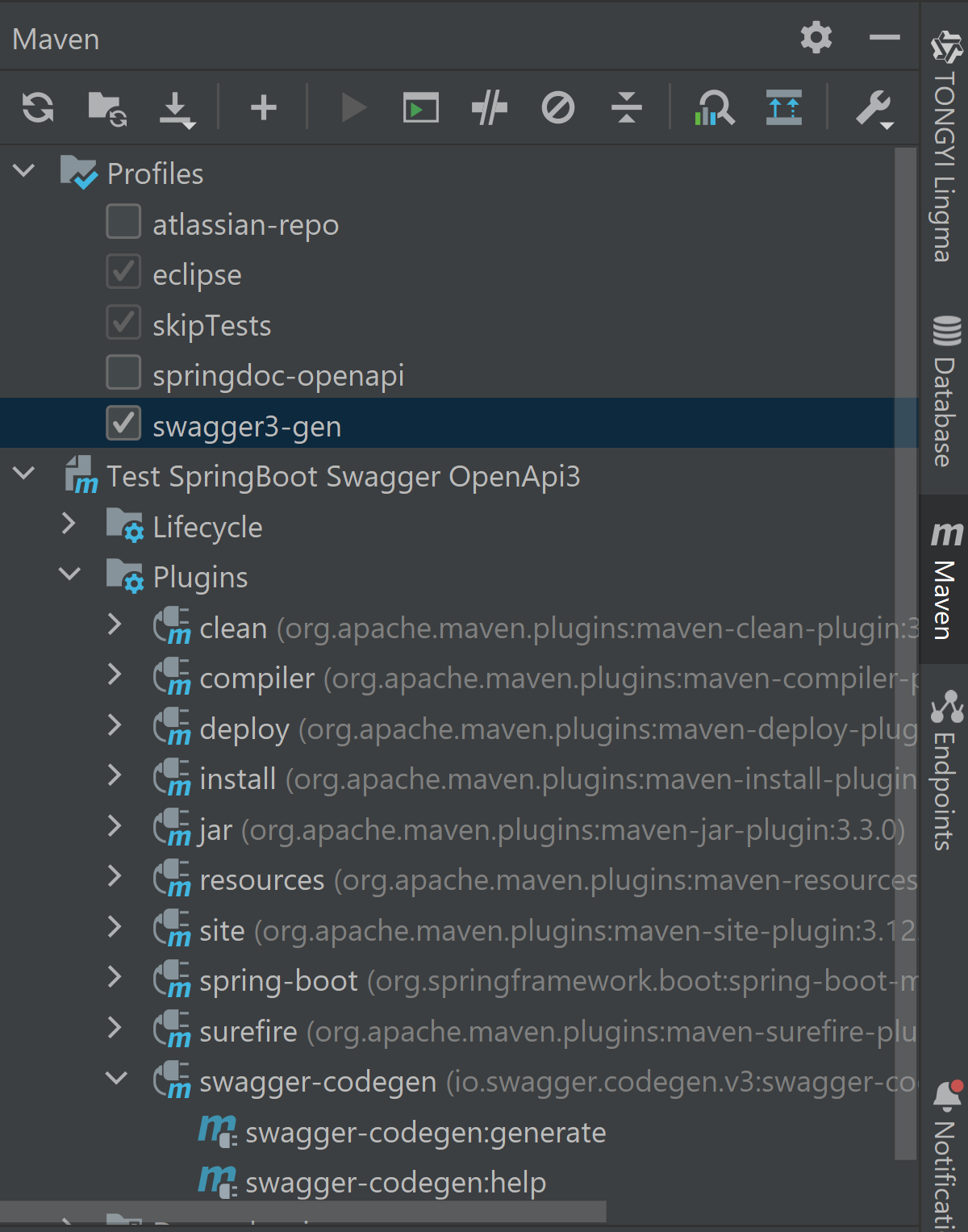
…



…



Alternatively, excute maven goal from IntelliJ



Browsing source code of plugin for supported version of Angular

<https://github.com/swagger-api/swagger-codegen-generators/blob/master/src/main/java/io/swagger/codegen/v3/generators/typescript/TypeScriptAngularClientCodegen.java#L232>

