REST API met Spring Boot, H2 en Swagger-UI

Stap 1

1. Maak een nieuw Maven project via <https://start.spring.io/>

met Java 17

2. Project metadata

Group: nl.cubicated

Artifact: crudapi

Name: crudapi

Description: A rest api for a list of citytrips

Packaging: Jar

3. Met de volgende dependencies:

- spring-boot-starter-data-jpa

- spring-boot-starter-web

- spring-boot-devtools

- h2 // de in-memory SQL database die we gaan gebruiken

4. Generate, download en pak het project uit in een map op de D schijf.

5. Open het project in Eclipse.

Stap 2 : Main class en de model Class

1. voeg het volgende toe in de main class:

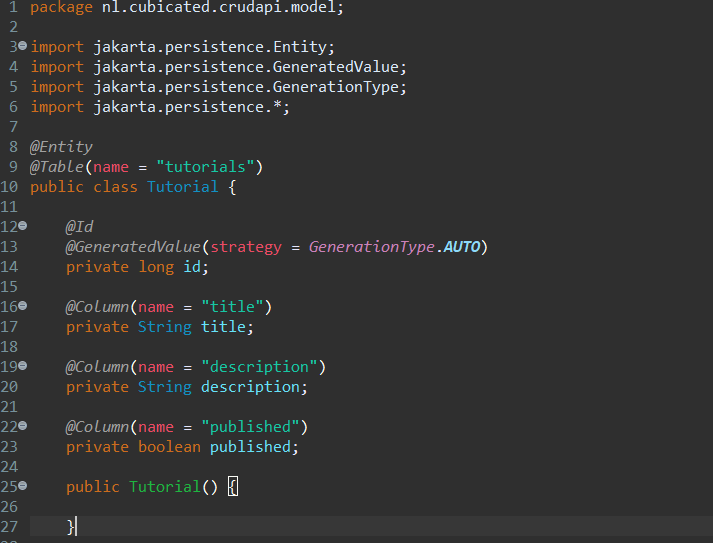
;

}

2. Maak een nieuwe package aan met de naam: nl.cubicated.crudapi.model

3. In deze package maak een nieuwe class aan met de naam: Tutorial

4. In de class Tutorial voeg je de volgende code toe:



vervolg:

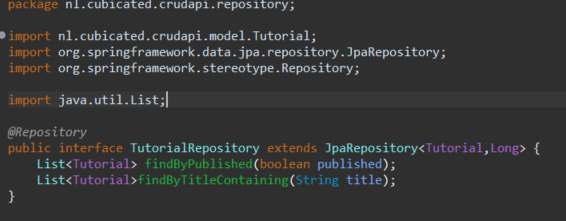


Stap 3 Repository

1. Maak een nieuwe package aan met de naam: nl.cubicated.crudapi.repository.

2. In de repository package maak je een nieuwe interface aan met de naam: TutorialRepository.

3. De volgende code stop je in deze file:

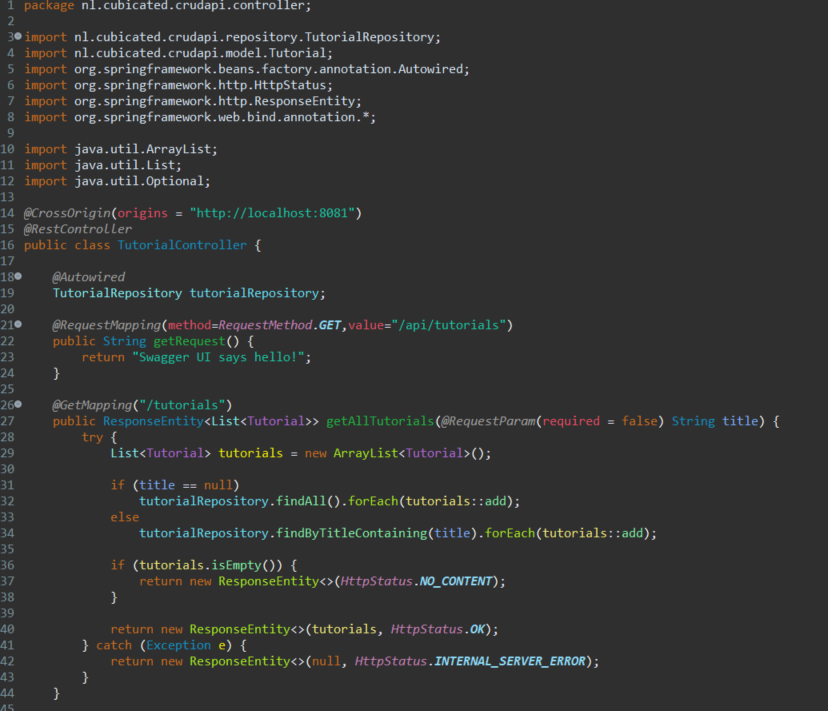


Stap 4 Controller

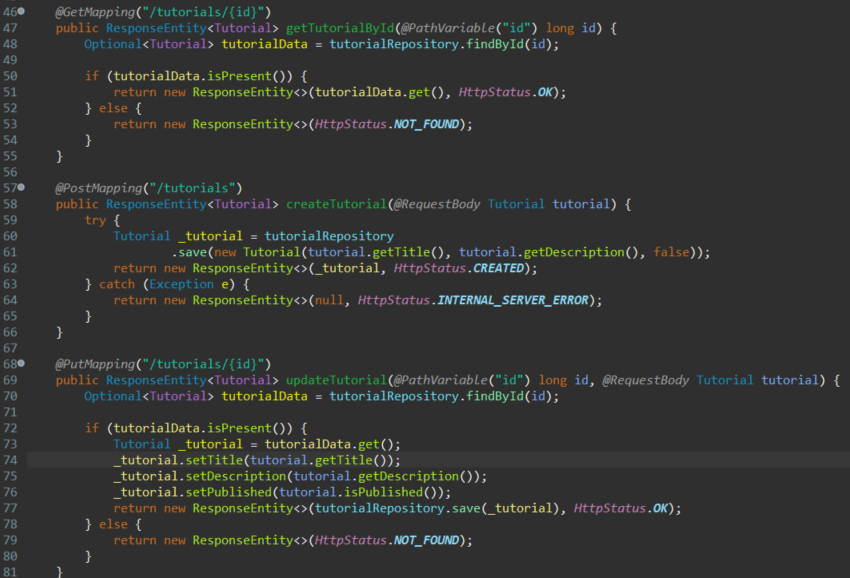
1. Maak een nieuwe package aan met de naam: nl.cubicated.crudapi.controller.

2. In de repository package maak je een nieuwe interface aan met de naam: TutorialController.

3. De volgende code stop je in deze file:



Vervolg 1

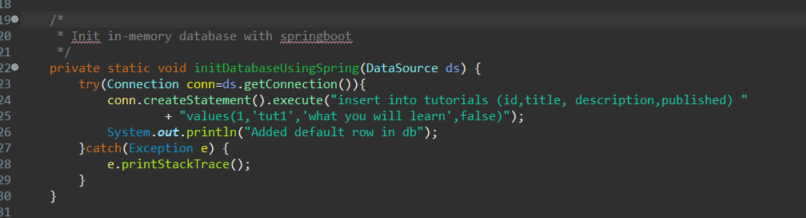


Vervolg 2



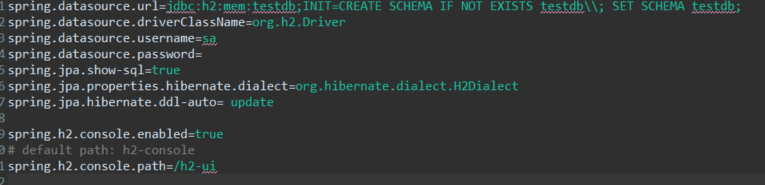
Stap 5: H2 database init data

1. Om te zorgen dat er tenminste al 1 row in de database komt te staan elke keer dat hij opstart is het van belang om deze in de class CrudApplication toe te voegen als volgt:



Stap 6: Application properties

1. Om ervoor te zorgen dat je in de H2 database kunt komen is het van belang om deze properties vast te leggen in de application properties:

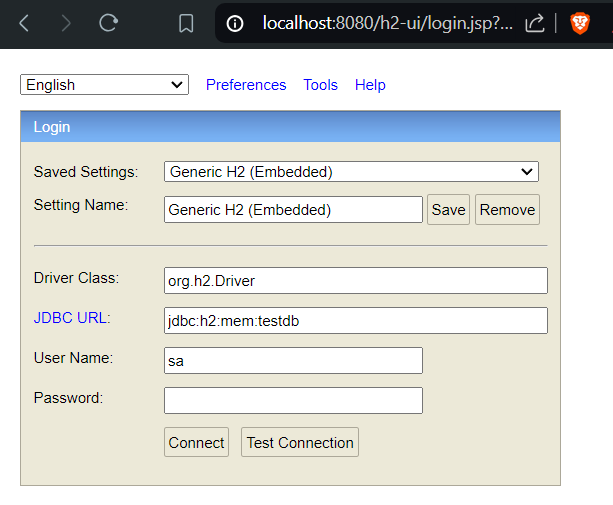
2. Let wel dat er nu geen wachtwoord is ingesteld voor de database. Dit is niet veilig voor production doeleinden!

Stap 7: Eerste test

1. Start de applicatie op als Java applicatie in Eclipse

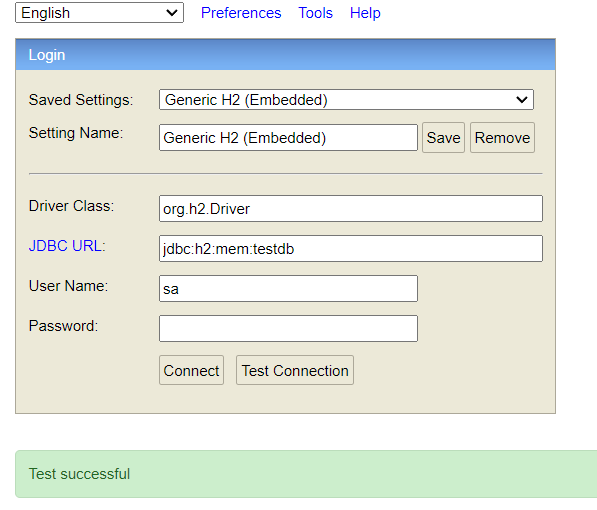
2. Vul de volgende url in :<http://localhost:8080/h2-ui>

Als het goed is zie je dan het volgende:



3. Om de connectie te testen klik je op test connectie.

Als het goed is zie je dan het volgende:



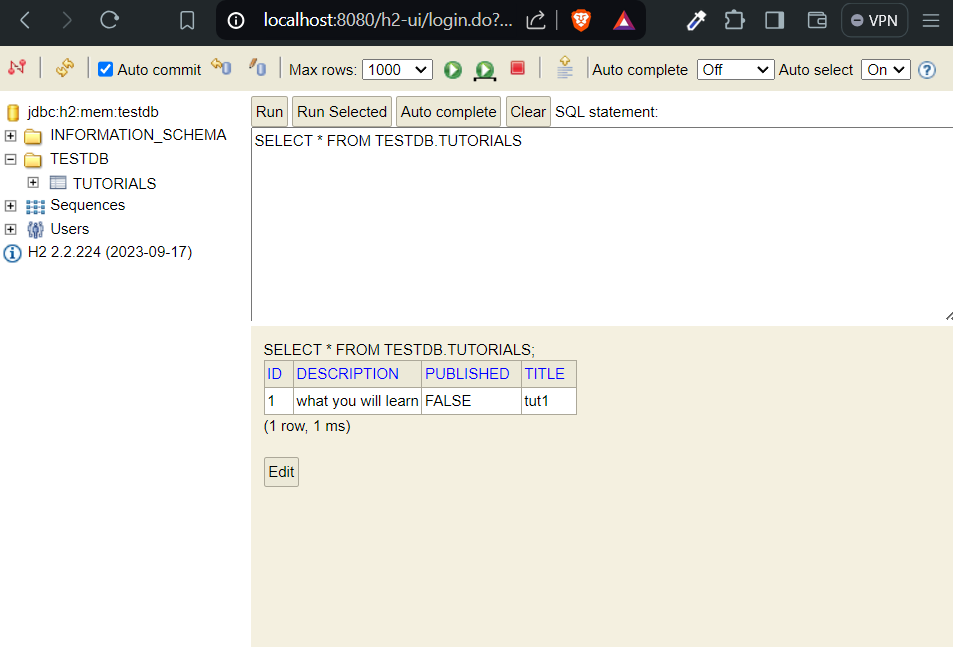
4. Om de database te openen klik je op connect.

De database wordt geopend

5. Aan de linkerkant klik je op TESTDB en dan op TUTORIALS

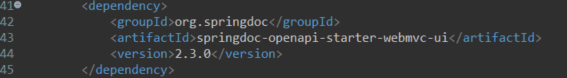
In het rechterveld wordt de SQL statement: SELECT \* FROM TESTDB.TUTORIALS geladen.

6. Klik op Run en alle rijen worden geladen, maar aangezien we er maar 1 hebben toegevoegd de Main class, zal dit er maar 1 zijn.



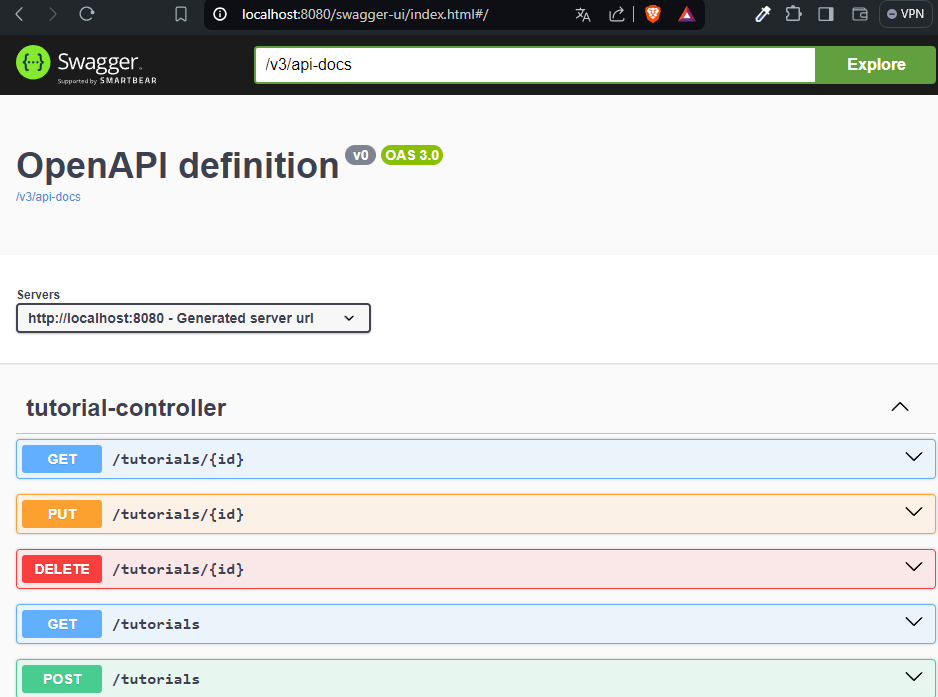
Stap 8: Opzetten van REST API test met Swagger-UI

1. Ga terug naar Eclipse naar de pom.xml en voeg de volgende dependency toe om in Spring Boot 3 te kunnen werken met Swagger-UI.

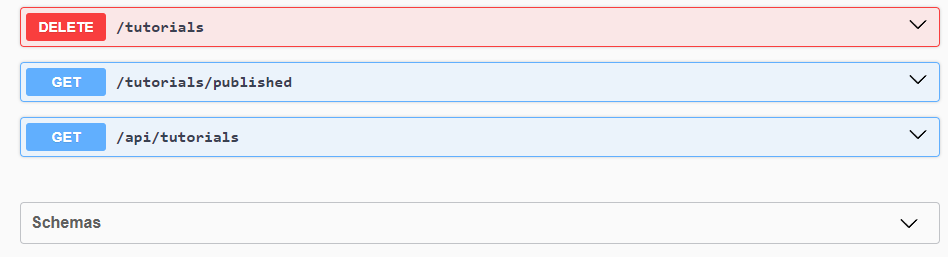
2. Ga in de browser naar de volgende url:

<http://localhost:8080/swagger-ui/>

3. Als het goed is kom je op de volgende pagina:



Vervolg:

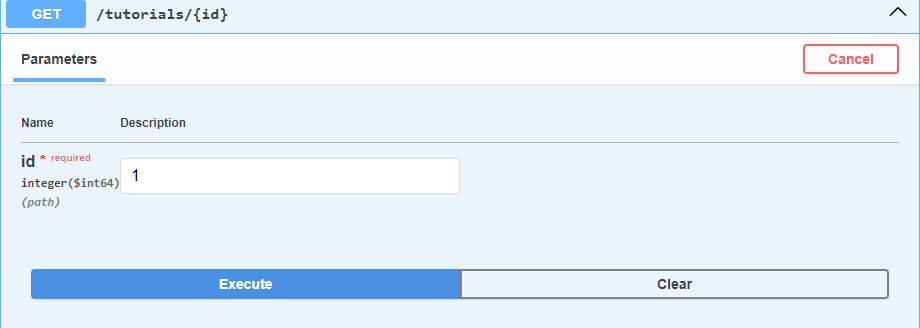


Stap 9: Testen van REST API met Swagger UI

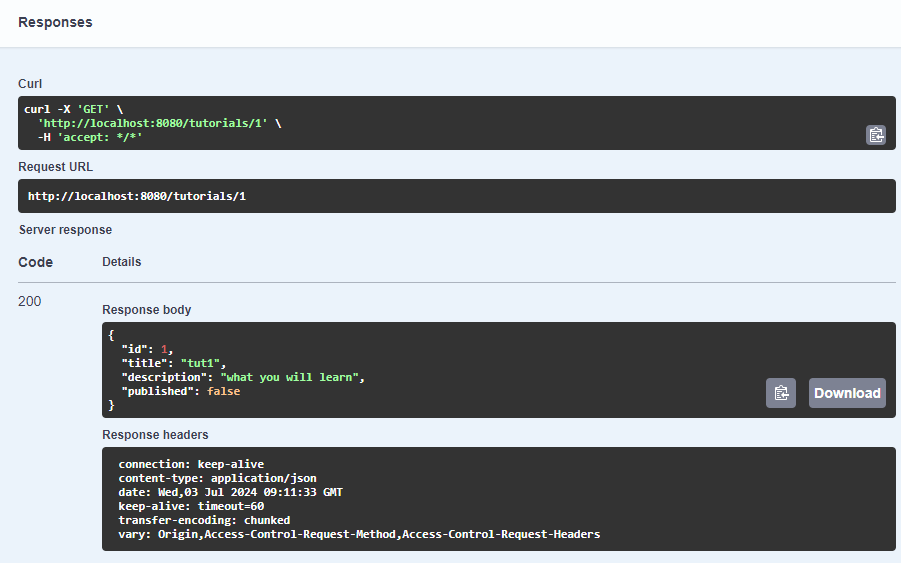
1. In de vorige stappen hebben we de applicatie gemaakt en de database gelinkt. Om daadwerkelijk de REST API te testen met Swagger UI is het van belang dat deze (vanuit Eclipse) is opgestart.

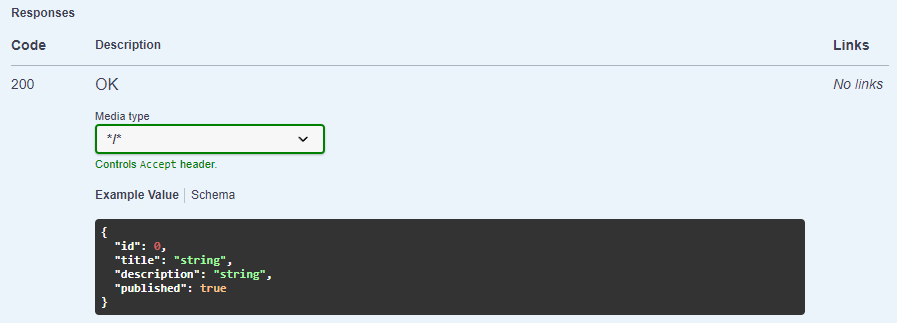
GET ID testen

2. In de swagger-ui ga je naar GET / tutorials/{id} en klik je op execute



3. De volgende response wordt dan geladen:





Uitleg Mappings:

@RequestMapping … /api/tutorials

- Dit is het eerste wat het programma tegenkomt bij het testen straks met swagger.

- Het path /api/tutorials is de bijbehorende URL

- In Swagger komt deze requestmapping overeen met de GET/api/tutorials path

@GetMapping(“/tutorials”)

- Is een url voor het weergeven van alle tutorials welke in de h2 database staan.

@GetMapping(“/tutorials/{id}”)

- Is een url voor het weergeven van een tutorial op basis van de ingegeven id

@PostMapping(“/tutorials”)

- Is een url waarbij men een nieuwe row kan aanmaken met values voor de

fields: id(long), description(String) en published(boolean)

@PutMapping(“/tutorials/{id}”)

- Is een url waarbij men een bestaande row kan updaten op basis van het ingegeven id.

@DeleteMapping(“/tutorials/{id}”)

- Is een url waarbij men een bestaande row kan deleten op basis van het ingegeven id.

@DeleteMapping(“/tutorials”)

- Is een url waarbij men alle rows in de tabel tutorials kan deleten.

@GetMapping(“/tutorials/published”)

- Is een url waarbij men een bepaalde row of rows kan vinden op basis van publicatiedatum.