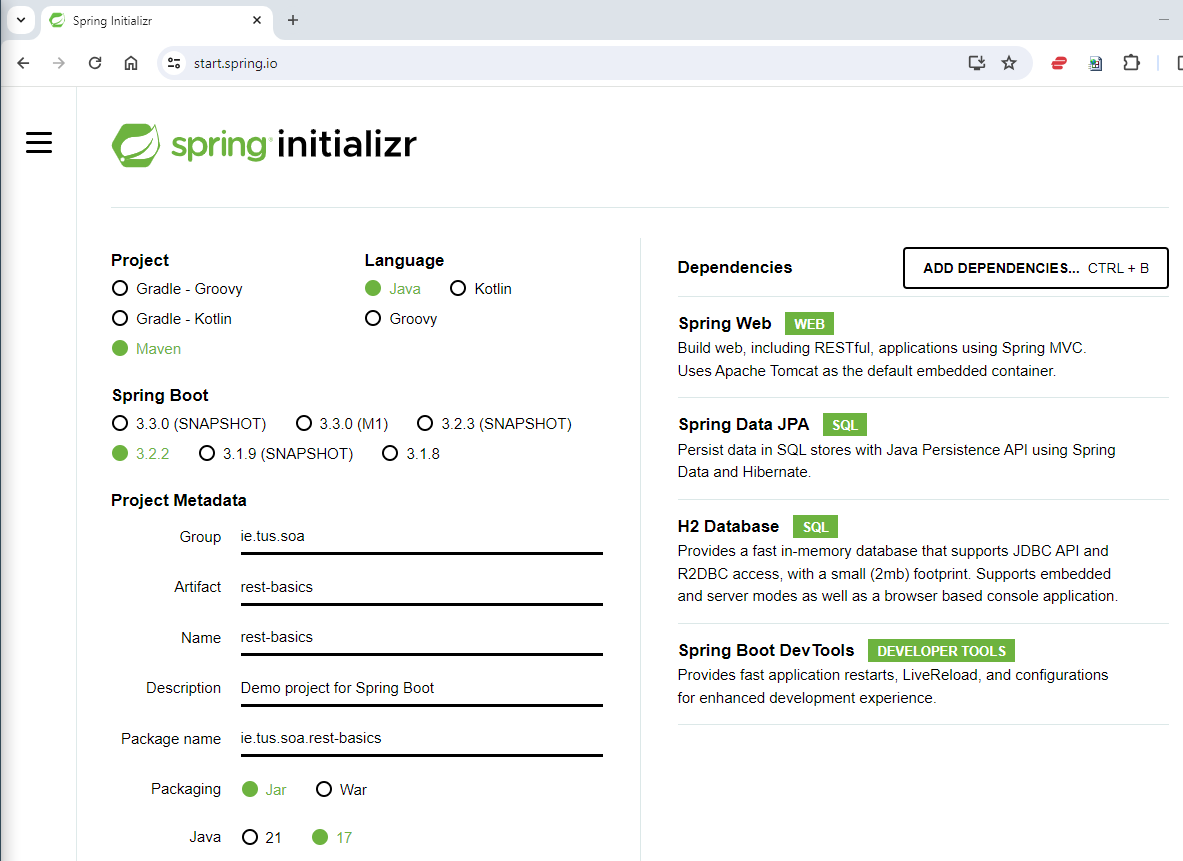
**Lab - Rest Basics**

**~~Step 1: Create a starter project at start.spring.io and import into Eclipse.~~**



Note the dependencies:

**Spring Web**: needed to create RESTful APIs

**Spring Data JPA**: used to access databases

**H2 Database**: in-memory database – handy for development

**Spring Boot Dev Tools**: live reload after code changes

~~Note: Choose the latest Spring Boot version that’s not a snapshot (usually default).~~

~~Click ‘Generate’ to download the project, unzip, and import into Eclipse (Existing Maven Projects).~~

**~~Step 2: Create a basic RESTful API~~**

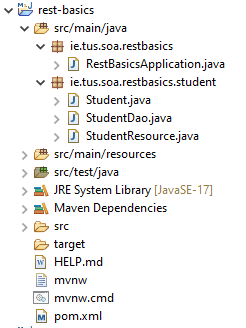
~~In this step, we’ll create a RESTful API for a Student object which will comprise the following classes:~~

**~~Student~~**~~: Plain Old Java Object (POJO) to hold student details, aka ‘Student bean’.~~

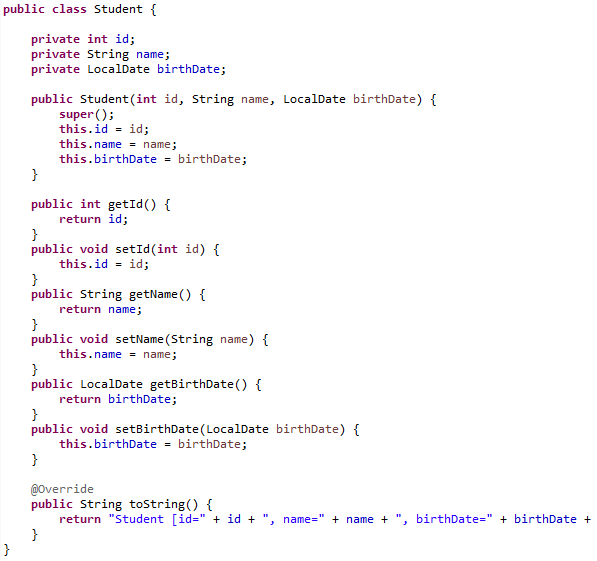
**~~StudentDao~~**~~: Student Data Access Object (DAO); used to access a database to do CRUD operations. For now, we’ll use a Map in place of a database.~~

**~~StudentResource~~**~~: This class is the RestController – it contains the mappings for GET, POST, DELETE etc.~~

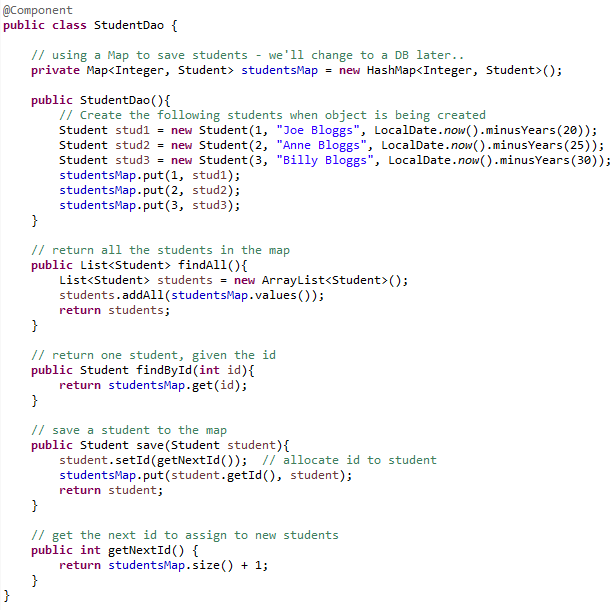
Create the classes in the package **ie.tus.soa.rest-basics.student** as follows:



~~Complete the~~ ***~~Student~~*** ~~class - write the private member variables and use Eclipse to generate the~~ **~~constructor~~**~~, the~~ **~~getters~~** ~~and~~ **~~setters~~**~~, and the~~ **~~toString~~** ~~methods.~~



~~Complete the~~ ***~~StudentDao~~*** ~~class as follows:~~

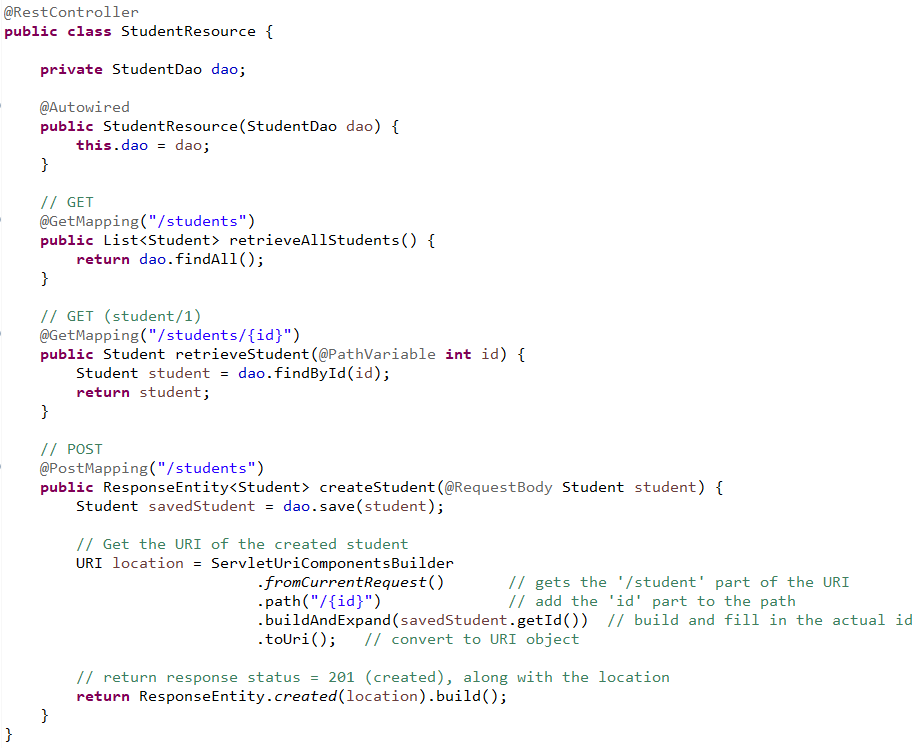


**~~Note:~~**

~~We’re using this class to mimic a class that communicates with a database. We use the~~ ***~~java.util.Map~~*** ~~data structure to save the~~ ***~~Student~~*** ~~objects. You’ll need to add more methods to this class to~~ *~~edit~~* ~~and~~ *~~delete~~* ~~students - more later.~~

~~This class is marked with the~~ **~~@Component~~** ~~annotation. This means we want Spring to manage the class. This is~~ *~~@Autowired~~* ~~into the~~ ***~~StudentResource~~*** ~~class below.~~

Complete the ***StudentResource*** class as follows:



**~~Notes:~~**

~~This class is marked with the~~ ***~~@RestController~~*** ~~annotation; Spring uses this to pick up the URI mappings to the methods defined e.g. when a GET request comes in on~~ ***~~/students~~***~~, the~~ ***~~retrieveAllStudents()~~*** ~~method is called.~~

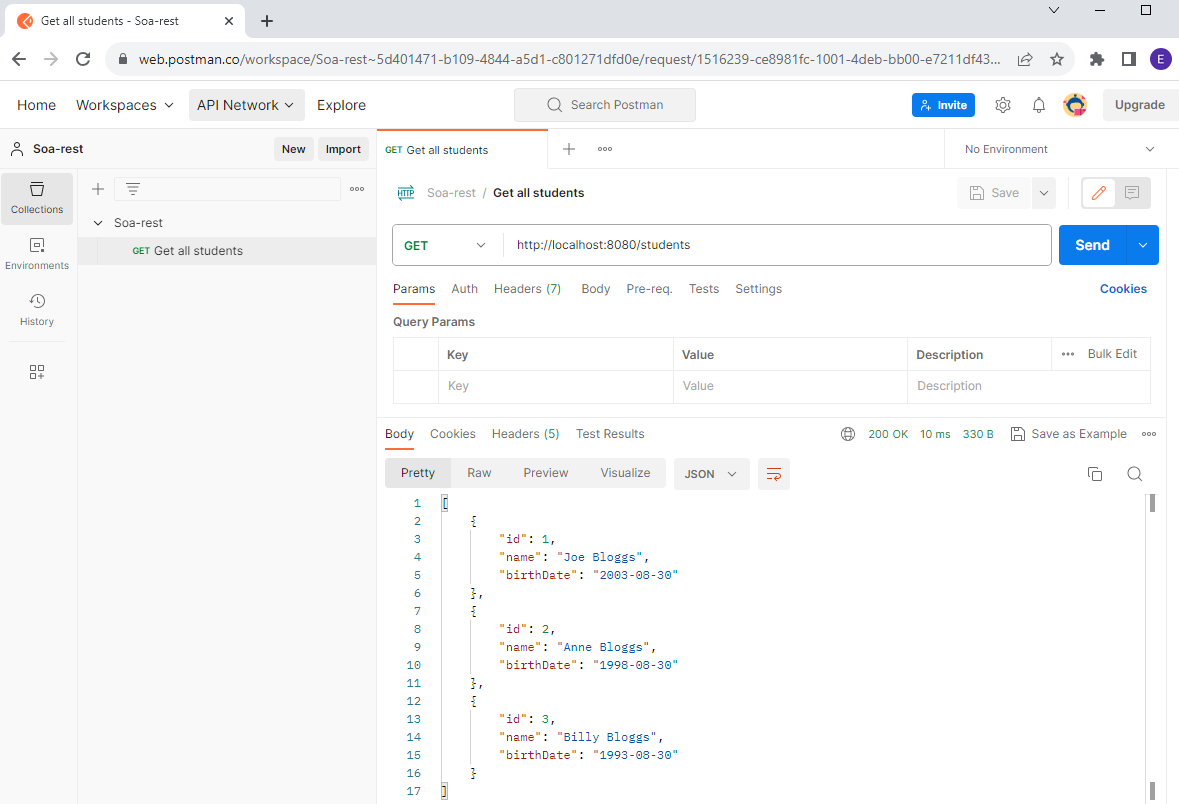
~~Note the~~ ***~~@PathVariable~~*** ~~in the~~ ***~~retrieveStudent()~~*** ~~method; this is used to get the id variable from the URI. E.g. if the user types in~~ ***~~/students/123~~***~~, id will be returned as 123.~~

***~~@RequestBody~~*** ~~is used in the POST method to pass a student object from the client. The POST method returns a~~ *~~201~~* ~~(created) status code, along with a path to where the new student is created.~~

**~~Step 3 – Test the API using Postman~~**

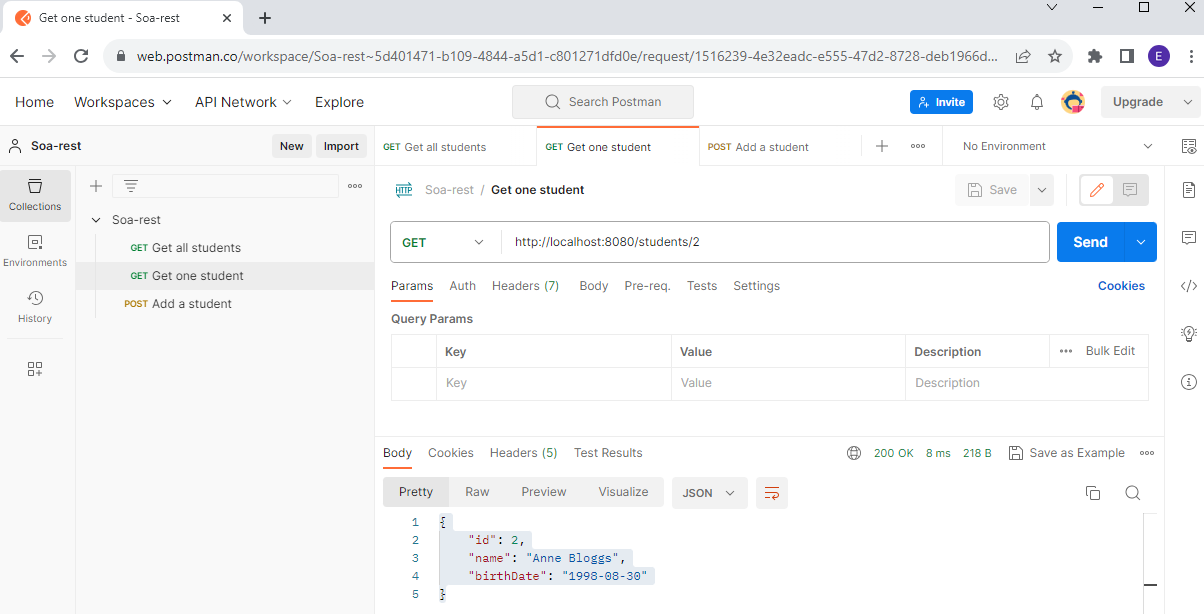
~~The browser only allows us to see the GET response. In order to test all the verbs, we’ll use Postman.~~

1. ~~Install Postman from the Software Center (or use the web version; for this you’ll have to install the Desktop Agent to communicate with the localhost).~~
2. ~~Start Postman and create a new~~ ***~~Collection~~***~~.~~
3. ~~Create a new~~ ***~~Request~~*** ~~in the collection.~~
4. ~~Give the request a name e.g. “Get all Students”~~
5. ~~Fill in the location of the resource you want to access~~
6. ~~Click ‘Send’ to send the request~~

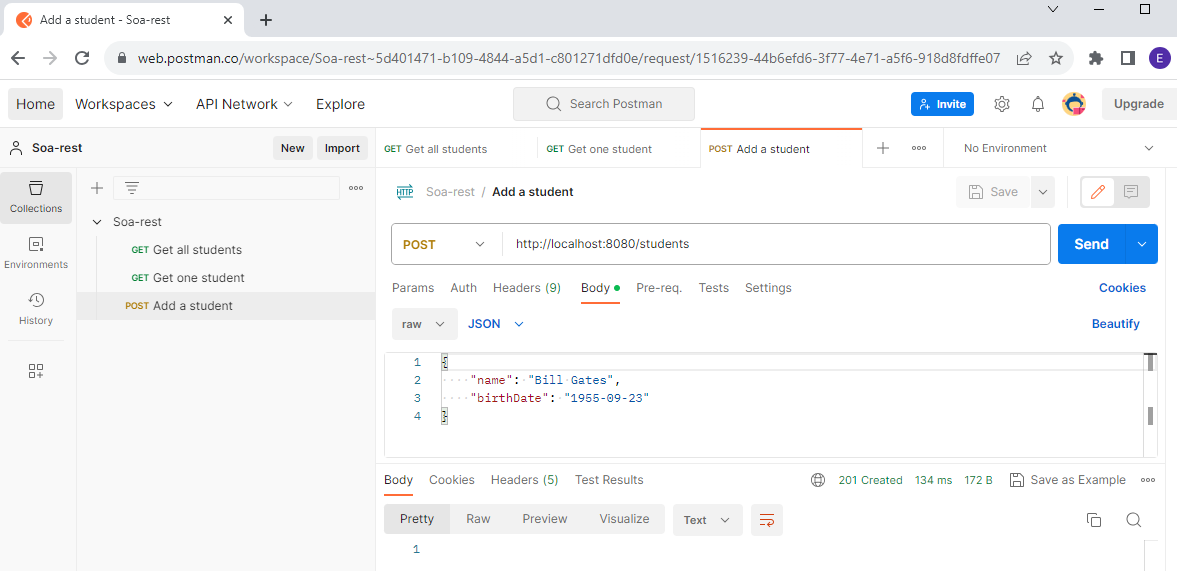


~~Next, make new requests to get one student and another to save a student – shown on next page.~~

**~~Get one student: /students/2~~**



**~~Add a student (POST):~~**



~~Make sure to add~~ **~~all~~** ~~the requests – this will speed up testing later; we’ll use the same Student example when we add the database in the next lab.~~

**Step 4: Expand the API to include the following:**

1. ~~Add appropriate methods to delete~~ ***~~one~~*** ~~student~~
   1. ~~Need to add a deleteStudent method in StudentDao. The method should return true if the student is present and is deleted successfully. It should return false if the student is not present.~~

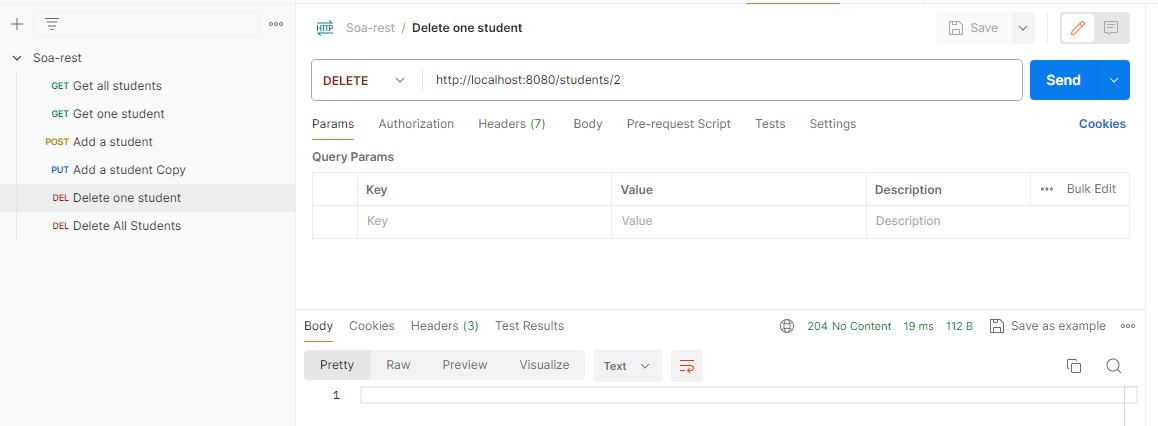
~~(Hint: use the remove() method of the Map object – it will return null if the id you try to delete does not exist)~~

* 1. ~~Need to add a @DeleteMapping mapping to StudentResource~~
  2. ~~Can return a ResponseEntity<Void> to return status codes. Use the value returned from the deleteStudent method to make the following decision:~~
     1. ~~204 (No content) if delete is successful~~
     2. ~~404 (Not found) if resource does not exist~~
  3. ~~Test the new functionality by adding a new request in Postman~~

1. ~~Add appropriate methods to delete~~ ***~~all~~*** ~~students~~
   1. ~~Need to add a deleteAllStudents method in StudentDao.~~ 
      1. ~~deleteAllStudents returns void.~~
      2. ~~Use the clear() method of the Map object to remove all students from the Map~~
   2. ~~Need to add another @DeleteMapping mapping to StudentResource~~
   3. ~~Return error code of 204 (No content)~~
   4. ~~Test the new functionality by adding a new request in Postman~~
2. Add appropriate methods to edit a student
   1. ~~Need to add an editStudent method in StudentDao (similar to saveStudent but don’t need to getNextId(). Remember with PUT we know the id)~~
   2. ~~Need to add a @PutMapping to StudentResource~~
   3. ~~Newly created/edited student should be returned to the user (use ResponseEntity.ok() )~~
   4. ~~Test the new functionality by adding a new request in Postman~~

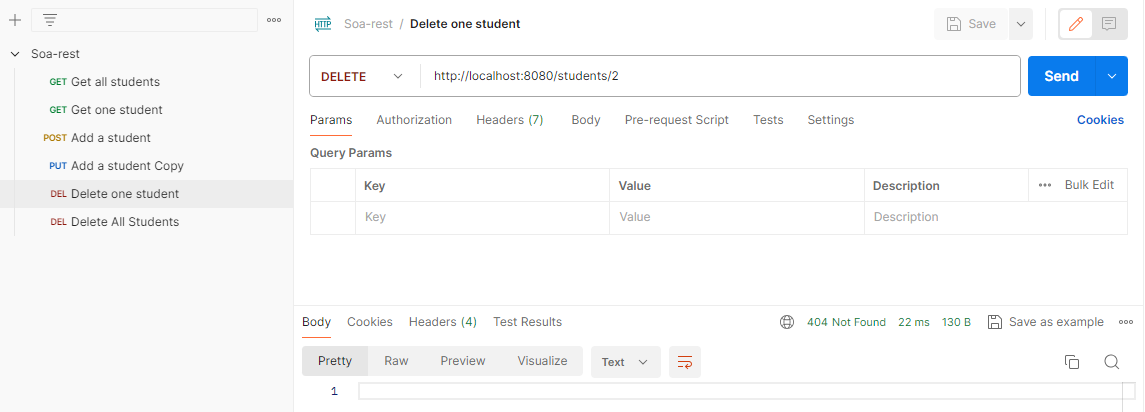
**Examples of how the Postman tests should work:**

**Deleting one student:**

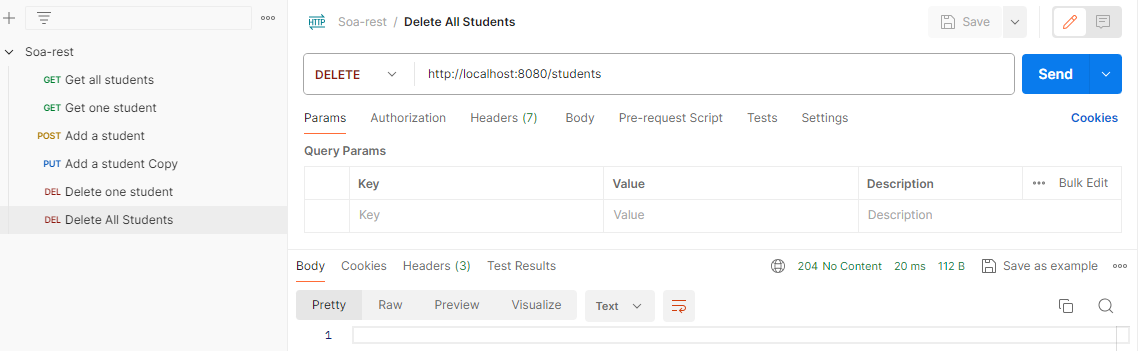


~~Check to make sure the student is deleted by running ‘Get all Students’ again.~~

~~Now try running the same command again, and make sure you get a ‘404 Not Found’ (as the student is already deleted and doesn’t exist):~~

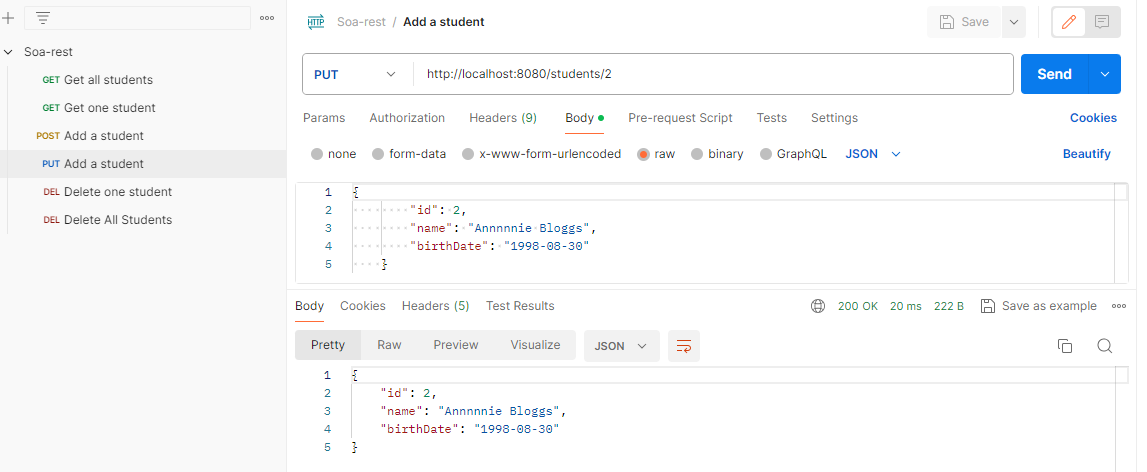


**~~Deleting all students:~~**



~~Check to make sure all students are deleted by running ‘Get all Students’ again.~~

**PUT – Editing a Student:**



**Appendix**

**Response Status codes for Http**

**404** - Not found

**500** – Server exception

**400** – bad request

**200** – success

**201** – created

**204** – no content

**401** – unauthorized (when authentication fails)