# LAB 4

# CREATE & CONSUME RESTFUL API

## ➢ CONTENT

- Introduce about Restful API
- Create RESTful web services with Java Spring Boot
- Introduce about ReactJS
- Consume API with ReactJS

## ❖ INTRODUCTION

- *REST architectural style:*
  - REST is a simple way to organize interactions between independent systems. It is simpler than complex mechanisms such as RPC or SOAP
  - Properties: simplicity, modifiability, visibility, portability, reliability
- *RESTful web service:*
  - REST architecture-based web services
  - Lightweight, maintainable, scalable
  - Used to create APIs for web-based application
  - The calling client can perform operations using RESTful service
  - The protocol for REST is HTTP

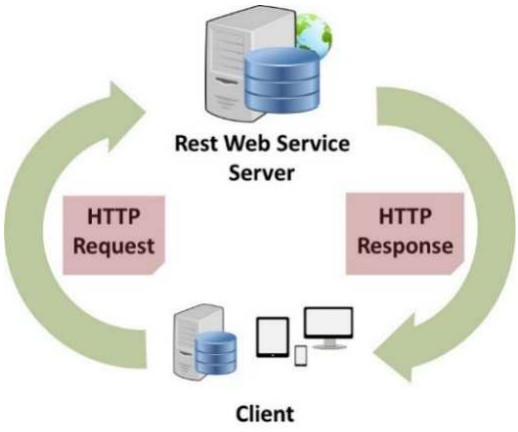



*Figure 1 - RESTful Web Service Architecture*

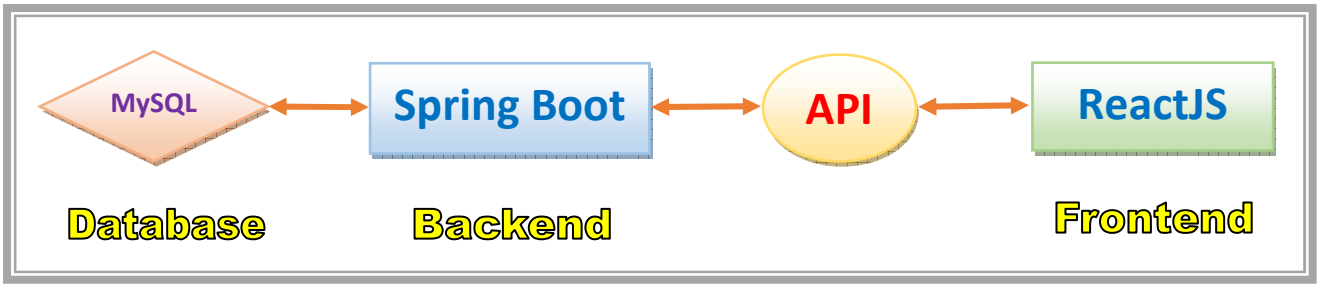
- **@GET, @PUT, @POST, @DELETE**: used to specify the HTTP request type for a method

| | SQL | HTTP Method | HTTP Status code |
|---|---|---|---|
| CREATE | Insert | POST | 201 |
| READ | Select | GET | 200 |
| UPDATE | Update | PUT | 200 |
| DELETE | Delete/ Drop | DELETE | 204 |

*Figure 2 - CRUD (SQL - HTTP)*

- *Formats of APIs:*
  - XML: eXtensible Markup Language
  - JSON: JavaScript Object Notation
- ReactJS is a JavaScript library for building user interfaces. It was developed and is maintained by Facebook and is widely used for building web applications. It allows developers to create reusable UI components and manage the state of their application, making it easier to build complex user interfaces.
- Some alternatives to ReactJS:
  - AngularJS
  - VueJS

- Methods to consume Restful API with ReactJS:
  - *Axios: a promise-based HTTP client*
  - Fetch API: a browser built-in web API
- System architecture diagram:




## ➢ INSTRUCTION

1. Create new Java Spring Boot project with dependencies

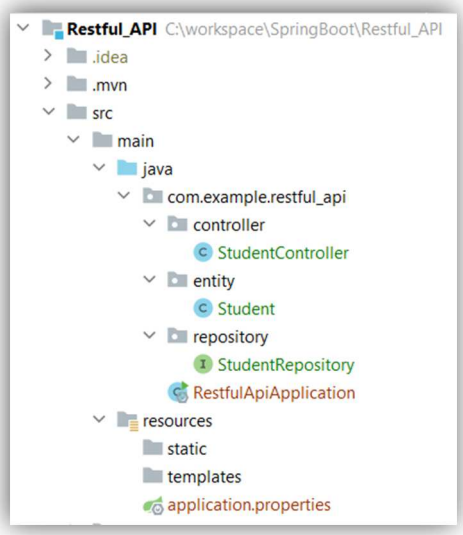



*Figure 3 – Sample project structure*

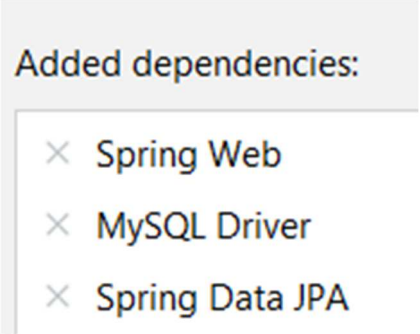



*Figure 4 - Project dependencies*

2. Config MySQL connection, JPA & Hibernate
3. Create Java class for model (entity) which acts as table in database

```java
@Entity
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;
    private String name;
    private int age;

    // auto generated getter & setter
```

*Figure 5 - **Student.java***

4. Create Java interface which extends *JpaRepository*
5. Create Java class for Restful controller to create Restful APIs

```java
@RestController
public class StudentController {
    @Autowired
    StudentRepository studentRepository;

    @GetMapping(value = "/")
    public List<Student> viewStudentList() {
        return studentRepository.findAll();
    }

    @GetMapping(value = "/detail/{id}")
    public Student viewStudentById(
            @PathVariable (value = "id") Long id) {
        return studentRepository.findById(id).get();
    }
```

*Figure 6 - **StudentController.java (1)***

```java
    @PostMapping(value = "/add")
    public Student addStudent(
            @RequestBody Student student) {
        return studentRepository.save(student);
    }

    @PutMapping(value = "/update/{id}")
    public void updateStudent(
            @PathVariable(value = "id") Long id,
            @RequestBody Student student) {
        if (studentRepository.existsById(id)) {
            student.setId(id);
            studentRepository.save(student);
        }
    }

    @DeleteMapping(value = "/delete/{id}")
    public void deleteStudent(
            @PathVariable(value = "id") Long id) {
        if (studentRepository.existsById(id)) {
            Student student = studentRepository.getById(id);
            studentRepository.delete(student);
        }
    }
}
```

*Figure 7 - **StudentController.java (2)***

6. Populate sample data in database using MySQL Workbench or integrated MySQL database in IntelliJ

| id | age | name |
|---|---|---|
| 1 | 18 | Minh |
| 2 | 20 | Hung |
| 3 | 22 | Lien |

*Figure 8 - Populate sample data in database*

7. Run the web application then test Restful APIs with Postman

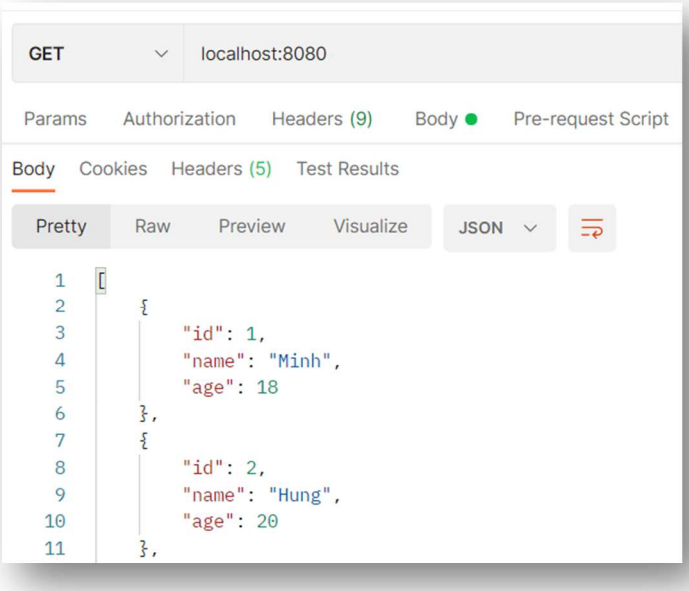



*Figure 9 - View all students (GET method)*

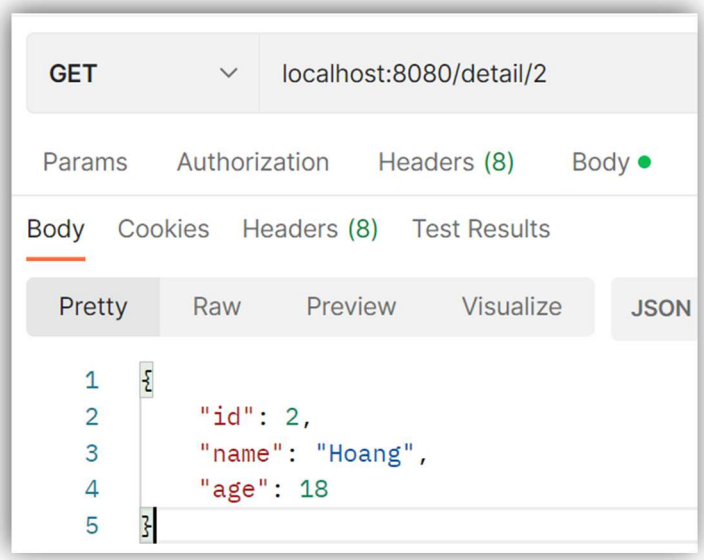



*Figure 10 - View student by ID (GET method)*

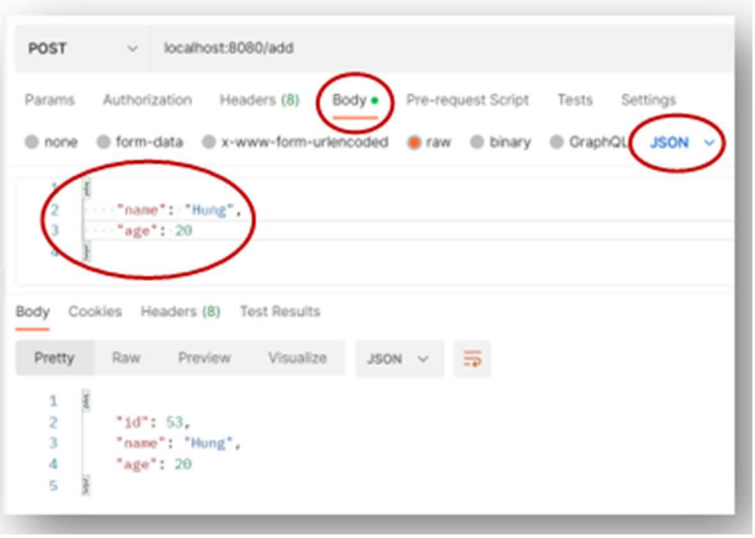



*Figure 11 - Add new student (POST method)*

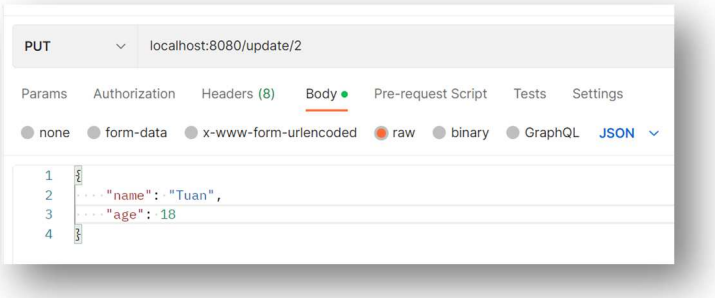



*Figure 12 - Update student (PUT method)*

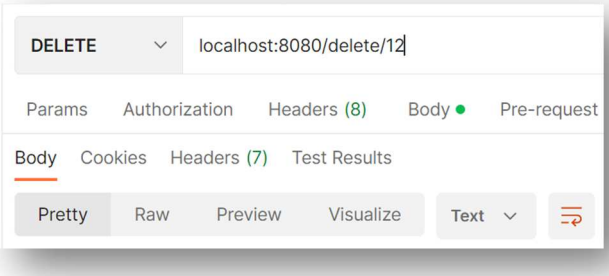



*Figure 13 - Delete student (DELETE method)*

8. Use previous Java Spring Boot project as back-end.

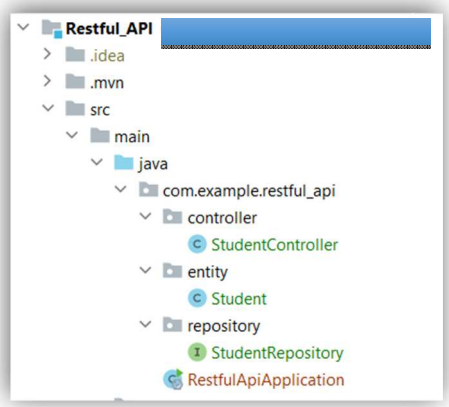



*Figure 14 – Java Spring Boot project structure (back-end)*

9. **Important:** Must enable CORS (Cross-Origin Resource Sharing) either locally for each controller or globally for the whole project.

   **Note:** Without enabled CORS, the back-end & front-end sides can not communicate and exchange data using API.

```java
@SpringBootApplication
public class RestfulApiApplication {

    public static void main(String[] args) {
        SpringApplication.run(RestfulApiApplication.class, args);
    }

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping( pathPattern: "/").allowedOrigins("http://localhost:3000");
            }
        };
    }
}
```

*Figure 15 – Enable CORS globally - **RestfulApiApplication.java***

```java
@CrossOrigin(origins = "http://localhost:3000")
@RestController
public class StudentController {
```

*Figure 16 - Enable CORS locally in each controller - **StudentController.java***

**Note:**

- *Default port for Java Spring Boot is 8080*
- *Default port for ReactJS is 3000*

10. Start the web server to run Java Spring Boot project then leave it on to share APIs to ReactJS. Do not *STOP* this server

**Note:** Both servers for front-end & back-end must run at the same time. ⇨ Open 2 projects in 2 independent IntelliJ windows or open Java Spring Boot project in IntelliJ and ReactJS in Visual Studio Code (or other alternative)

11. Create a new ReactJS project as front-end, which consumes Restful API

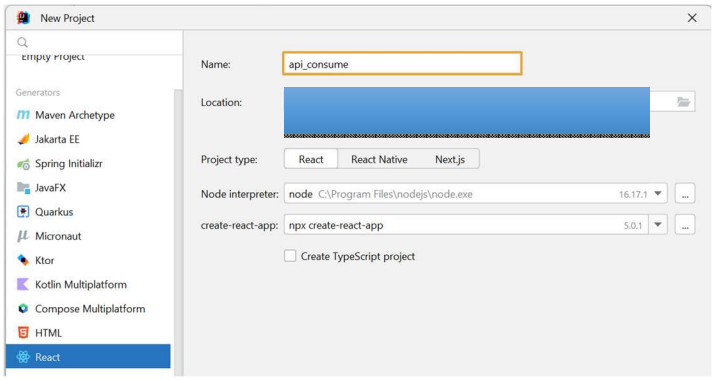



*Figure 17 - Create new ReactJS project*

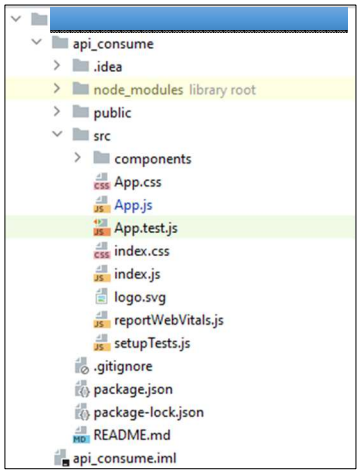



*Figure 18 - ReactJS project structure (front-end)*

12. Install Axios library (module/package) by running below command in integrated terminal at bottom left corner

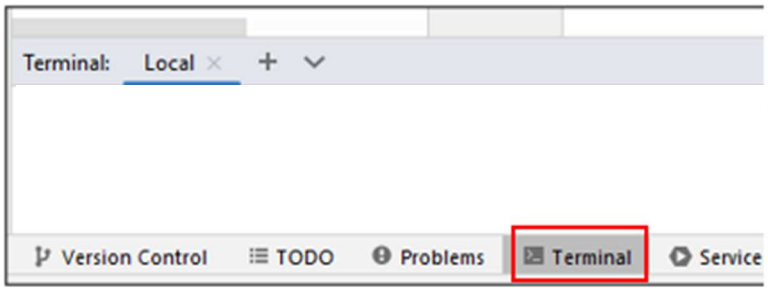



*Figure 19 - Set default Terminal*

```
npm install axios
```

*Figure 20 - Axios module installation*

13. Create **components** package in **src** folder of ReactJS project

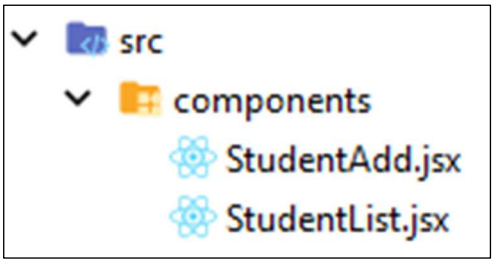



*Figure 21 - Create ReactJS components*

14. (Optional) Use Bootstrap CSS framework to decorate the web user interface

```
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
          rel="stylesheet" integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7cmDA6j6gD"
```

*Figure 22 – **public/index.html***

## 15. Create component **StudentList** to fetch student data from API

```
import React from 'react';
import axios from 'axios';

export default class StudentList extends React.Component {
    state = {
        students: []
    }

    url = "http://localhost:8080/"

    componentDidMount() {
        this.fetchStudentList();
    }

    componentDidUpdate(prevProps : Readonly<P> , prevState : Readonly<S> , snapshot : SS ) {
        if (this.props.reloadList !== prevProps.reloadList) {
            this.fetchStudentList();
        }
    }

    fetchStudentList = () => {
        axios.get(this.url).then((res : AxiosResponse<any> ) => {
            const students = res.data;
            this.setState( state: { students });
        });
    };
```

*Figure 23 - **StudentList.jsx (1)***

```
render() {
    return (
        <div className="container text-center mt-3">
            <table className="table table-primary">
                <thead>
                    <tr>
                        <th colSpan="4" className="h3 text text-danger bg-warning">STUDENT LIST</th>
                    </tr>
                    <tr className="h5 text text-success">
                        <th>Student Id</th>
                        <th>Student Name</th>
                        <th>Student Age</th>
                    </tr>
                </thead>
                <tbody>
                    {
                        this.state.students
                            .map(student =>
                                <tr key={student.id}>
                                    <td> {student.id} </td>
                                    <td> {student.name} </td>
                                    <td> {student.age} </td>
                                </tr>
                            )
                    }
                </tbody>
            </table>
        </div>
    )
}
```

*Figure 24 - **StudentList.jsx (2)***

## 16. Create module **StudentAdd** to add new student using form. Data from form will be passed to Spring Boot by API then be added to database.

```
import React from "react";
import axios from "axios";

export default class StudentAdd extends React.Component {
    state = {
        name: '',
        age: ''
    }

    url = "http://localhost:8080/"

    handleChange = event => {
        this.setState( state: {
            [event.target.id]: event.target.value
        });
    }

    handleSubmit = event => {
        event.preventDefault();

        event.target.reset();
        this.setState( state: {
            name: '',
            age: '',
        });

        const student = {
            name: this.state.name,
            age: this.state.age
        };

        axios.post( url: this.url + 'add', student)
            .then(res => {
                console.log(res);
                this.props.reloadStudentList();
            })
    }
```

*Figure 25 - **StudentAdd.jsx (1)***

```
render() {
    return (
        <div className="container text-center mt-3 mb-5">
            <h3 className="bg-warning text-primary p-2">ADD NEW STUDENT</h3>
            <form className="form card p-3 bg-light" onSubmit={this.handleSubmit} >
                <label className="form-label h5 text-success"> Student Name </label>
                <input className="form-control" type="text" id="name" minLength="3"
                       maxLength="20" required onChange={this.handleChange} />
                <label className="form-label h5 text-success"> Student Age </label>
                <input className="form-control" type="number" id="age" min="18" max="25"
                       required onChange={this.handleChange}/>
                <div className="text-center">
                    <button className="btn btn-primary mt-3 col-md-3" type="submit">
                        Add
                    </button>
                </div>
            </form>
        </div>
    );
}
```

*Figure 26 - **StudentAdd.jsx (2)***

## 17. Config file **App.js** to add these 2 components

```
import './App.css';
import StudentList from "./components/StudentList";
import StudentAdd from "./components/StudentAdd";
import {useState} from "react";

function App() {
    const[reloadList, setReloadList] = useState( initialState: false);

    const handleReloadList = () => {
        setReloadList(!reloadList);
    };
      return (
        <div className="container text-center card mt-3">
            <div className="row">
                <div className="col">
                    <StudentAdd reloadStudentList={handleReloadList}/>
                </div>
                <div className="col">
                    <StudentList reloadList={reloadList}/>
                </div>
            </div>
        </div>
      );
}

export default App;
```

*Figure 27 - **App.js***

## 18. Start the ReactJS project with **Shift + F10** or typing "**npm start**" in Terminal.

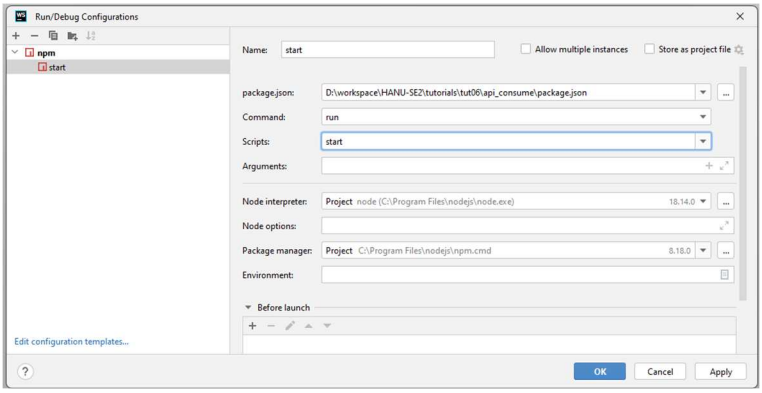



*Figure 28 - Add npm configuration*

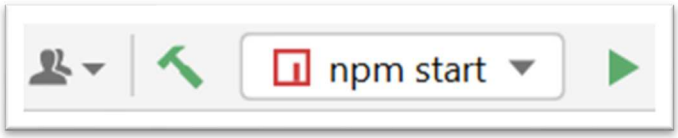



*Figure 29 - Start the ReactJS project*

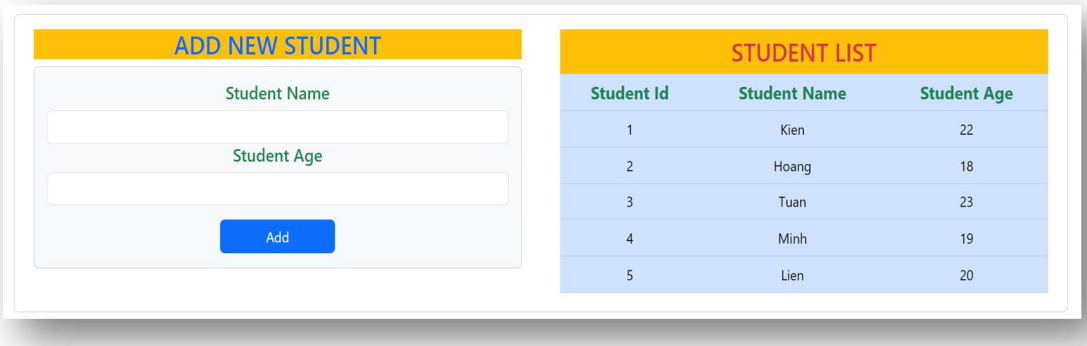



*Figure 30 – Current result*


*Figure 31 - Final result*

## ➢ TO-DO

- Add data validation for Entity & Controller then re-test with Postman
- Create similar Restful APIs for other Spring Boot projects
- Create UPDATE & DELETE components to complete CRUD features
- Create other components to consume remained APIs