

차세대 챗봇 Lab I

Lecture-4 리액트 AXIOS

Sungja Choi,
School of Computing, Gachon University
2023-1



학습내용

비동기 HTTP 전송 라이브러리 AXIOS에 대해 학습합니다.

AXIOS

http-proxy-middleware

React-Spring Integration

RestController

RequestMapping, PostMapping, GetMapping

Lecute-3. 리액트 AXIOS

AXIOS 라이브러리를 사용하여 HTTP 비동기 통신을 처리하는
기법을 학습합니다.

[참조]

<https://ko.reactjs.org/>

<https://recoiljs.org/>

Practice

```
>> AXIOS 라이브러리 설치  
npm i axios
```

```
>> http-proxy-middleware  
라이브러리 설치  
npm i  
http-proxy- middleware
```

실습-1 리액트와 스프링 통합

AXIOS 비동기 라이브러리를 사용하여 서버로 부터 유저정보를 가져옵니다.

[실행화면]

GCU React-Spring Integration

ID: 1

Name: Gachon

Date: 2023-03-22T09:11:26.754+00:00

Practice

>> 서버로 부터 전송된 정보를 받기 위해 user 상태변수를 정의하고 setUser 상태함수를 지정

- **useEffect** : 렌더링 위한 혹은 비동기 전송을 위해 사용
- **Axios.post**를 이용해서 서버로 해당 경로를 전송
- **then**은 결과가 올때까지 기다리고 서버로 부터 결과가 전송되면 response 인수로 받음
- **setUser** 메소드를 사용하여 전송된 결과를 user에 저장
- **[]** 은 한번만 렌더링 되도록 하는 useEffect 방식

Front: Coding(1) Axios를 사용하여 Post 요청

```
import React, {useState, useEffect} from 'react';  
import Axios from 'axios';
```

App.js

```
function App() {  
  const [user, setUser] = useState("");  
  useEffect(()=>{  
    Axios.post("/api/users").then((response)=>{  
      if(response.data){  
        setUser(response.data);  
      }else{  
        alert("failed to");  
      }  
    });  
  }, []);  
}
```

Practice

>> 상태변수 user에 저장된 id, name, dob를 화면에 보여줍니다.

서버에 정의된 프로퍼티

```
public class User
```

```
{  
  private Integer id;  
  private String name;  
  private Date dob;  
}
```

서버에서 전송될 때 JSON
형태로 보내집니다.

App.js:10

```
▼ Object  
  dob: "2023-03-22T09:28:59.264"  
  id: 1  
  name: "Gachon"  
  ► [[Prototype]]: Object
```

Front: Coding(2) 결과를 화면에 보여주기 위한

```
return (  
  <div className='App'>  
    <div align = 'left' style = {{margin: '20px'}} >  
      <h2>GCU React-Spring Integration</h2>  
      ID: {user.id} <br/>  
      Name: {user.name} <br />  
      Date: {user.dob} <br />  
    </div>  
  </div>  
);  
}
```

```
export default App;
```

App.js

[REF.]

>>

SOP (Same Origin Policy, 동일출처정책)

- 자바스크립트 엔진 표준 스펙의 보안 규칙
- 하나의 출처(Origin)에서 로드 된 자원(문서나 스크립트)이 일치하지 않는 자원과는 상호작용 하지 못하도록 요청 발생을 제한하는 정책

http://example.com:8042/over/there?name=ferret&page=1#nose

/ _/_ ^_/_ _/_ _/_ ^_/_

| | | | | | |

protocol host port path query string Fragment

(예) **http://localhost:8000**와 **http://localhost:8000/posts**는 같은 출처라서 상호작용이 가능하지만, **http://google.com**에서 **http://localhost:8000**를 호출하면 SOP에 위배됩니다.

[REF.]

>>Cors (Closs-Origin Resource Sharing)

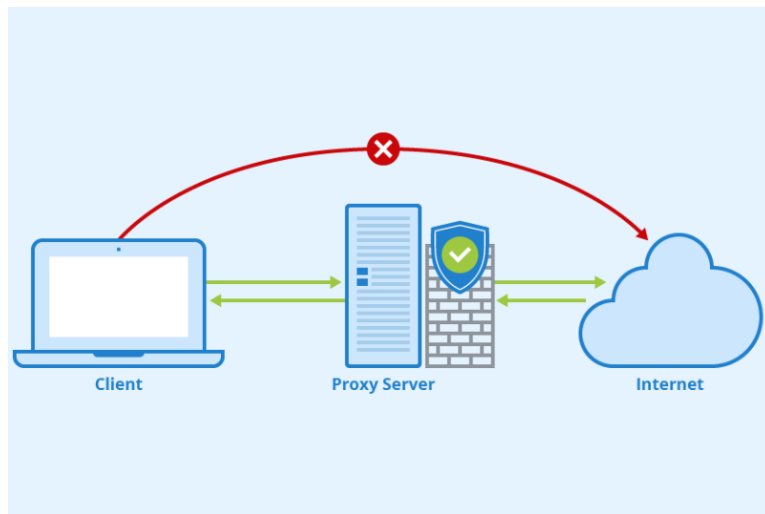
교차출처 리소스 공유:

- 실행중인 웹 어플리케이션이 다른 출처에서 선택한 자원에 접근할 수 있는 권한을 부여하도록 브라우저에게 알려주는 체제.
- 무분별한 리소스 접근을 막는 보안 이슈
- 처음 리소스를 요청한 주소와 다른 주소로 요청하면 Cors 에러 발생

=> Proxy로 해결

PROXY

- Proxy란 유저가 인터넷에 요청을 보낼 때 직접 보내는 것이 아니라 Proxy 서버를 거쳐 최종 목적지까지 전달하게 한다.



Practice

>> 스프링
서버주소(localhost:8080)를
프록시 미들웨어로
생성하고 호스트의 헤더가
변경되도록 설정 (change
Origin)

Front: Coding(3) 프론트엔드-백엔드 연동

setupProxy.js

```
const {createProxyMiddleware} = require("http-proxy-middleware");

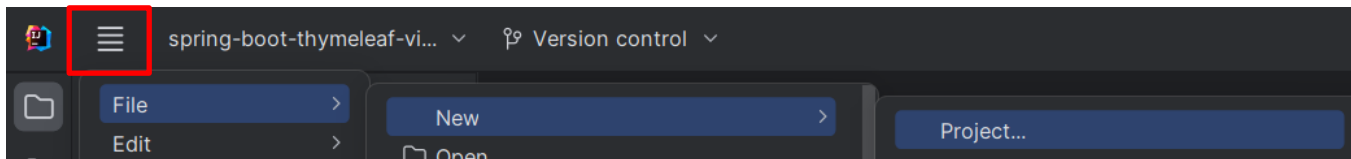
module.exports = function(app){
  app.use(
    "/api", //첫번째 Path (endpoint)
    createProxyMiddleware({
      target:"http://localhost:8080",
      changeOrigin: true,
    })
  );
}
```

Practice

>> IntelliJ IDE 사용
학생버전을 설치하면
Spring Initializer로 스프링
프로젝트를 쉽게 만들 수
있도록 도와줍니다.

백엔드 서버의 스프링 프로젝트 생성

- File – New – Project ...

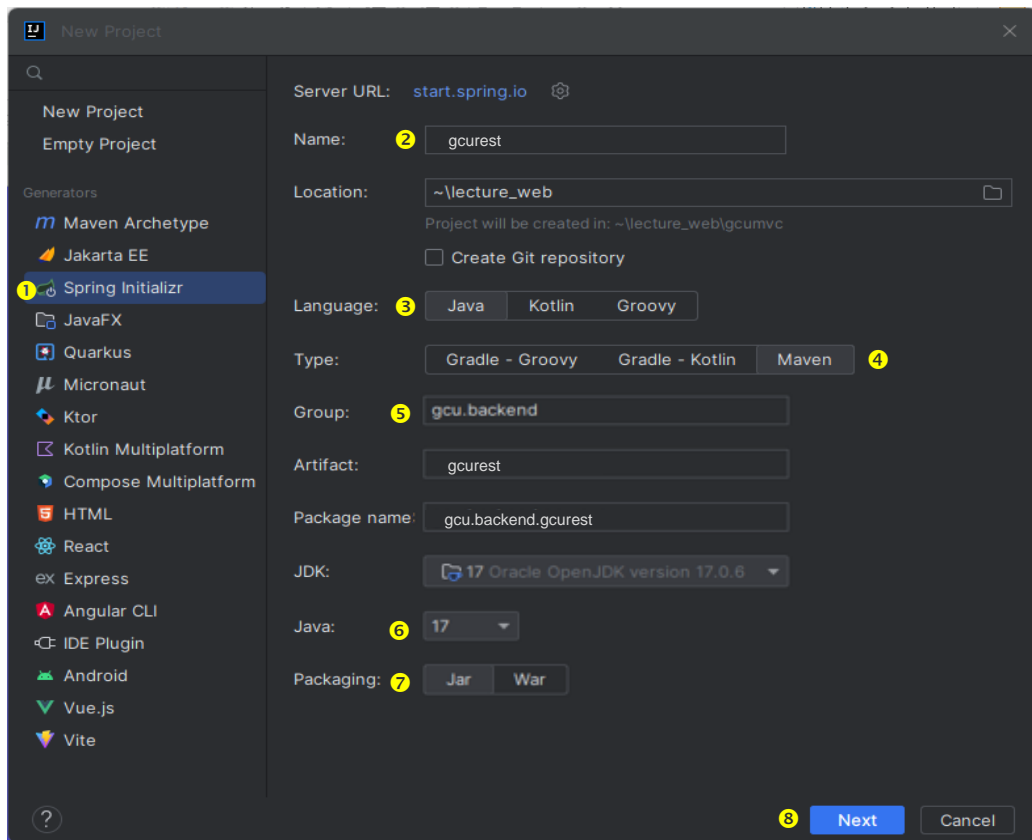


Practice

>> IntelliJ IDE 사용
학생버전을 설치하면
Spring Initializer로 스프링
프로젝트를 쉽게 만들 수
있도록 도와줍니다.

- 1 Spring initializr를 선택
- 2 프로젝트 이름 (gcurest)
- 3 개발언어 선택 (Java)
- 4 빌드 유형 선택 (maven)
- 5 그룹명 (gcu.backend)
- 6 JDK 버전 선택
- 7 배포유형 선택 (Jar)
- 8 Next버튼 클릭

서버 프로젝트 생성



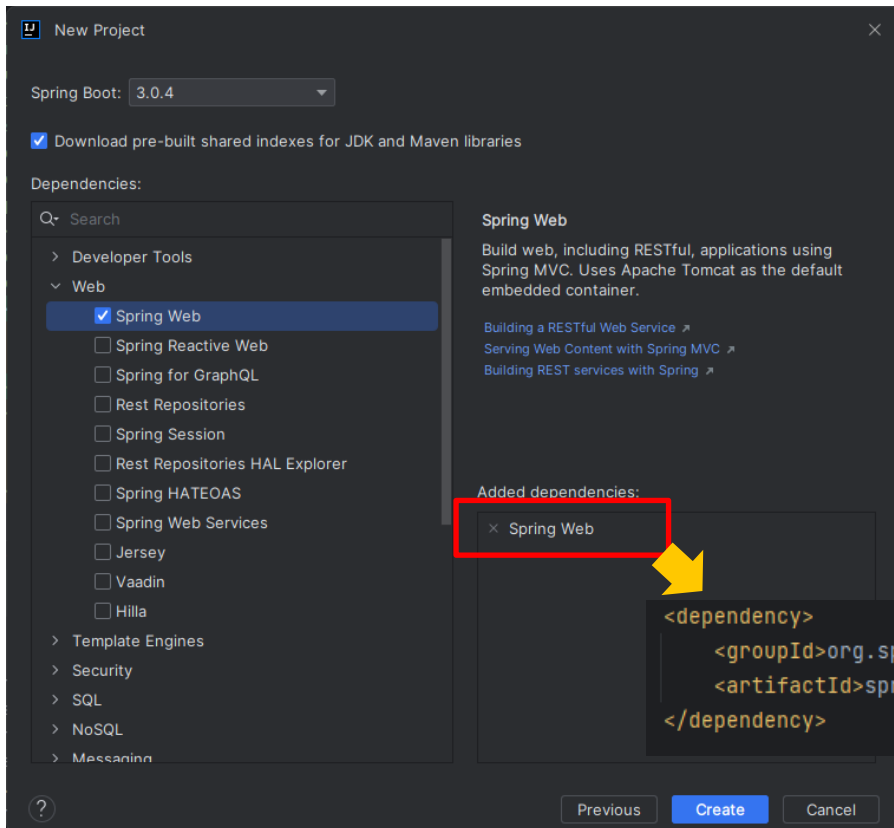
Practice

>> Spring Web

라이브러리를 프로젝트에 추가하여 웹개발에 필요한 모듈을 지원받습니다. 체크한 라이브러리는 pom.xml에 추가됩니다.

[REF.] pom.xml 파일은 프로젝트에서 사용하는 의존성 라이브러리를 관리합니다.

의존성 라이브러리 선택



pom.xml

Practice

>>

@SpringBootApplication

을 통해 백엔드 서버
어플리케이션이
만들어집니다.

[REF.] @ 표시를
어노테이션이라고 하며
표기되는 문구에 따라
기능이 주어집니다.

(예) @Getter

private String name;

코드를 구현하지 않아도
name 프러퍼티에 대해 Get
메소드가 자동으로 만들어
집니다.

스프링 어플리케이션 확인

```
package gcu.backend.gcurest;                                     GcurestApplication.java

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class GcurestApplication {

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

}
```

Practice

>> User 클래스는 정수형 id, 문자열 name, Date형 dob를 멤버로 가집니다. 인수를 모두 전달받는 생성자를 구현합니다.

Back: Coding(1) User 클래스 정의

```
package gcu.backend.gcurest;
```

User.java

```
import java.util.Date;
```

```
public class User
```

```
{
```

```
    private Integer id;
```

```
    private String name;
```

```
    private Date dob;
```

```
    public User(Integer id, String name, Date dob) {
```

```
        this.id = id;
```

```
        this.name = name;
```

```
        this.dob = dob;
```

```
    }
```

```
}
```

Practice

>> **@RestController**는
HTTP 요청을 처리하도록
도와주는
어노테이션입니다. 해당
예제에서는 POST 요청을
전송합니다.

>> **@PostMapping**
("api/users")
HTTP POST 요청이
들어오면 user() 메소드를
수행합니다. 괄호안의
패스(/api/users)는
전송경로를 나타냅니다.

Back: Coding(2) HTTP 요청 처리를 위한 코드

```
package gcu.backend.gcurest;                                     UserRest.java

import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.Date;

@RestController
public class UserRest{

    @PostMapping("/api/users")
    public User user() {

        System.out.println("UserApiController start...");
        User user = new User(1, "Gachon", new Date());

        return user;
    }
}
```

Practice

>> 터미널에서 메이븐 실행명령을 통해 서버 프로젝트를 실행시킵니다.
./mvnw spring-boot:run

[REF.] 그라들 빌드 사용
./gradlew build

실행

```
PS C:\Users\user\spring-workspace\gcurest> ./mvnw spring-boot:run
```



Practice

>> User의 Setter/Getter를
구현합니다.

Back: Coding(3) User Setter/Getter 추가

User.java

```
public Integer getId() {  
    return id;  
}
```

```
public void setId(Integer id) {  
    this.id = id;  
}
```

```
public String getName() {  
    return name;  
}
```

```
public void setName(String name) {  
    this.name = name;  
}
```

```
public Date getDob() {  
    return dob;  
}
```

```
public void setDob(Date dob) {  
    this.dob = dob;  
}
```

Practice

>> @GetMapping을 사용하여 HTTP GET 요청을 처리합니다. localhost:8080/api/getuser URI로 식별됩니다. 매핑 메소드는 get() 함수이고 User 객체를 리턴합니다.

Back: Coding(4) GetMapping 추가

UserRest.java

```
@GetMapping("/api/getuser")
public User get() {

    System.out.println("UserApiController start...");
    User user = new User(2, "Kakao", new Date());

    return user;
}
```

Practice

>> 비동기의 HTTP GET

요청을 프론트엔드에서
localhost:3000으로
보냅니다. 서버로 연결되는
경로는 /api/getuser 입니다.

Front: Axios Get 추가

App.js

```
Axios.get("/api/getuser").then((response)=>{  
  if(response.data){  
    console.log(response.data);  
    setUser(response.data);  
  }else{  
    alert("failed to");  
  }  
});
```


Practice

프론트 엔드와 백엔드 실행

>> VSCode

```
PS C:\Users\user\frontend-workspace\hello> npm start
```

>> IntelliJ



```
PS C:\Users\user\spring-workspace\gcurest> ./mvnw spring-boot:run
```

Practice

>> localhost:3000을
실행시키면 8080포트의
톰캣서버와 연동하여
결과를 리턴 받고 화면에
보여줍니다.

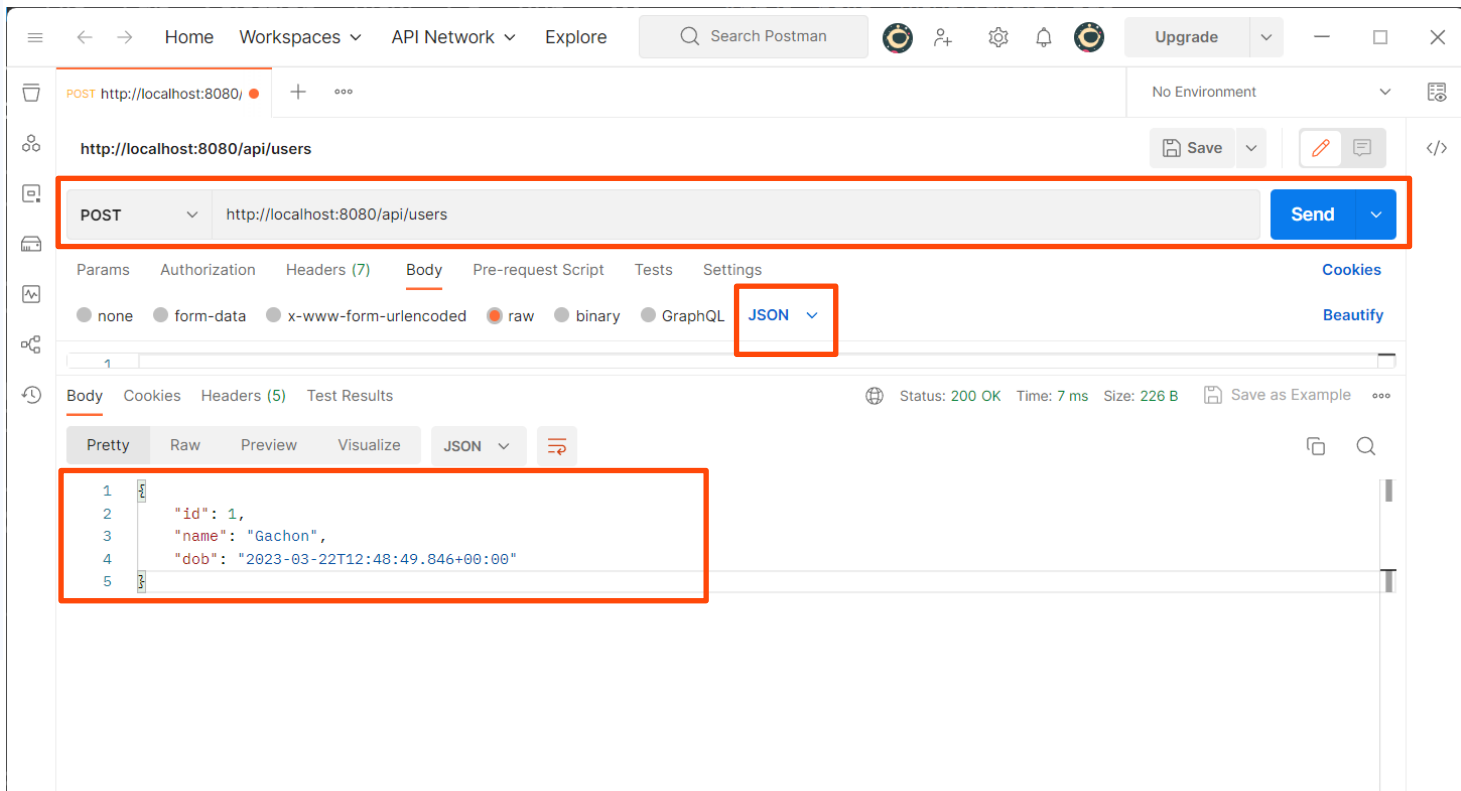
GET Method 실행



Practice

POST Method 실행

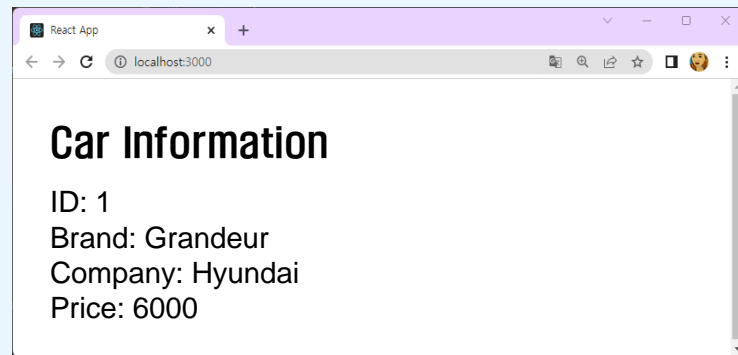
>> POSTMAN을 사용해서
POST 요청을
실행해봅니다.



수업시간 내 진행하시기 바랍니다.

Car Information

- 자동차 정보의 프론트-백 엔드 서비스 구현
- 프론트엔드에서 POST, GET 요청을 보내서 결과 출력
- 프론트엔드의 코드 작성 및 백엔드의 코드 작성
- UI를 다양하게 구성하고 기능도 역량에 맞도록 확장해 봅니다.
- 사이버캠퍼스 제출
- 평가(10점)



CONNECT.

SOLVE.

CREATE. 