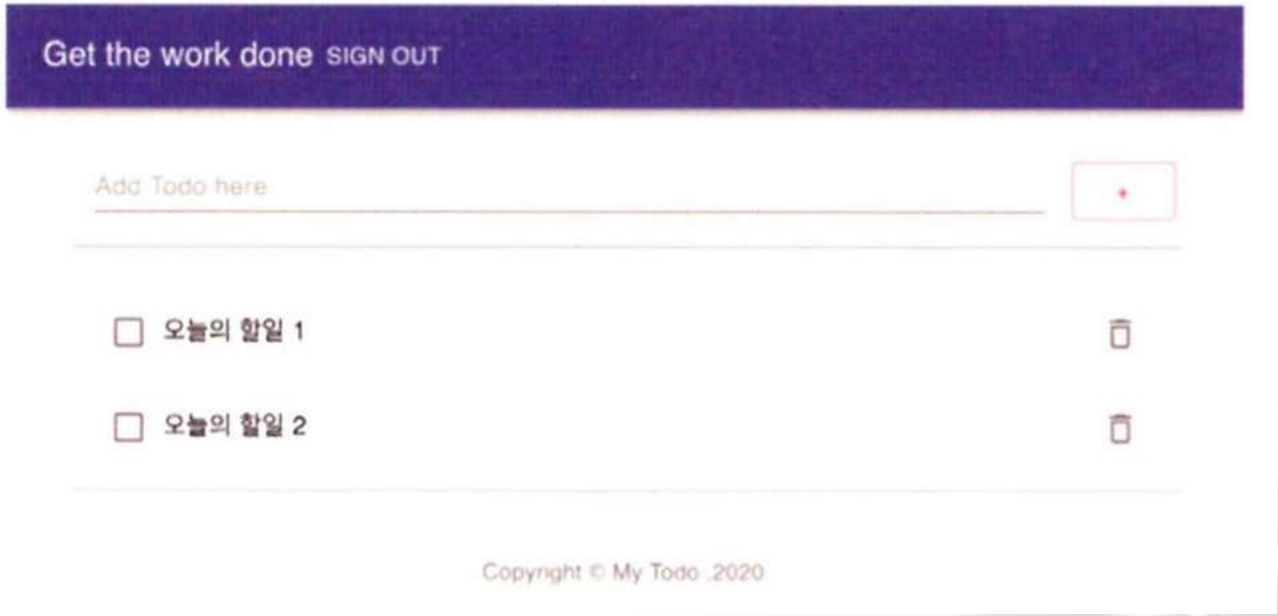
ToDo

개요

- ToDo Application



개요

- ToDo Application
  - ✓ 구현 기능
    - ToDo 생성: + 버튼을 클릭해 ToDo 아이템을 생성
    - ToDo 리스트: 생성된 아이 템 목록을 화면에서 확인
    - ToDo 수정: ToDo 아이템을 체크하거나 내용을 수정
    - ToDo 삭제: ToDo 아이템을 삭제
    - 회원 가입: 사용자는 애플리케이션에 회원 가입
    - 로그인: 계정을 생성한 사용자는 계정으로 로그인 할 수 있음
    - 로그아웃: 로그인 한 사용자는 로그 아웃 할 수 있음

개요

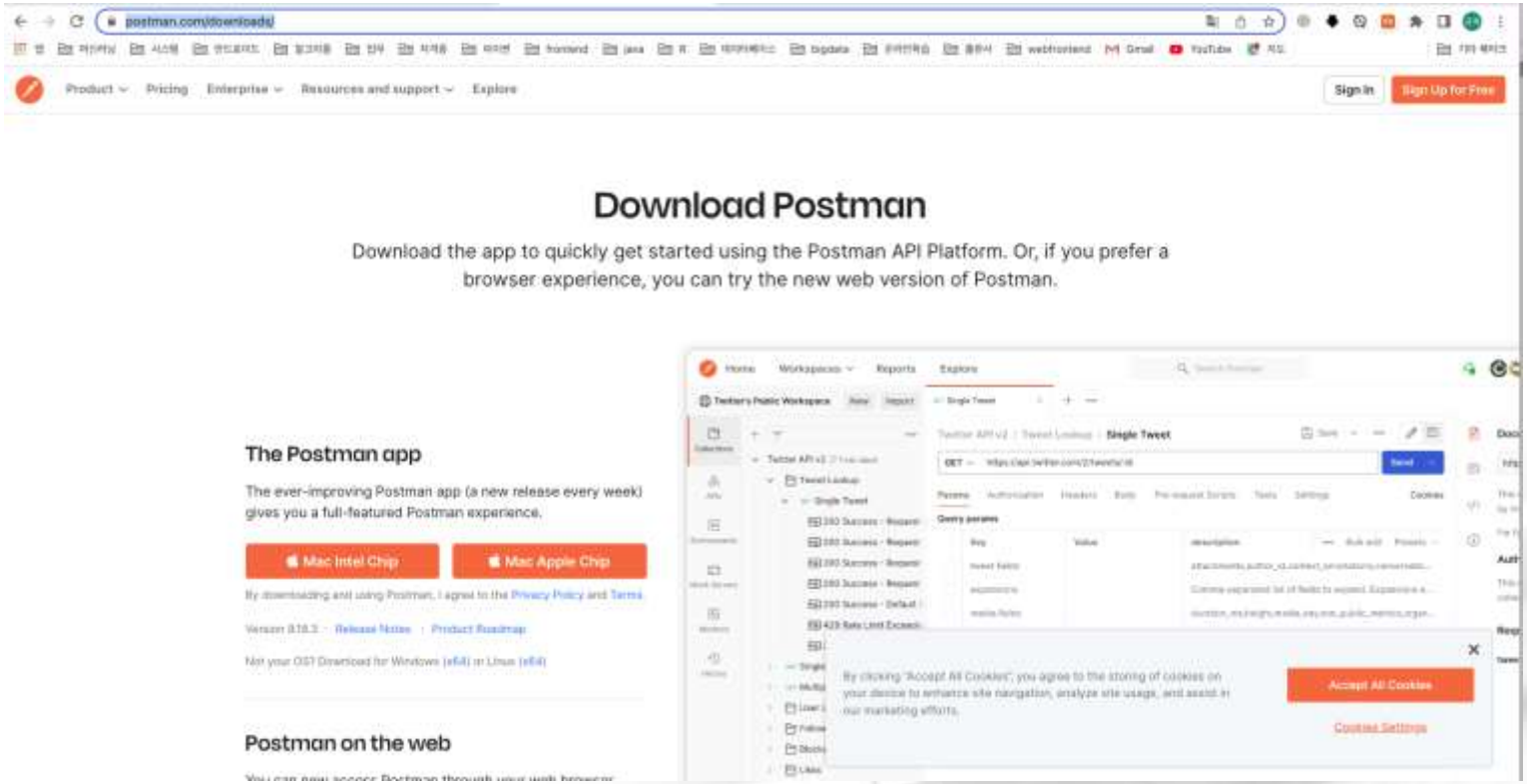
- ToDo Application
  - ✓ 기술
    - HTML/CSS/React.js
      - FrontEnd 애플리케이션 개발에 사용
      - FrontEnd와 BackEnd를 분리(decouple) 해서 개발
    - MariaDB
      - 반 영구적으로 데이터를 저장하기 위한 관계형 데이터베이스
    - Spring Boot
      - BackEnd 애플리케이션 개발에 사용
      - Spring Boot로 REST API를 구현
      - REST API를 구현하고 FrontEnd를 분리하는 것은 마이크로 서비스 아키텍처로 서비스를 확장하는 데 용이

개발 환경

- 개발 환경
  - ✓ Back End
    - JDK 1.11 이상 버전
    - MariaDB
    - IntelliJ: IDE
    - Dbeaver: 관계형 데이터베이스 접속 도구
    - Postman: API Server 테스트 도구
  - ✓ FrontEnd
    - Visual Studio Code: IDE
    - React.js

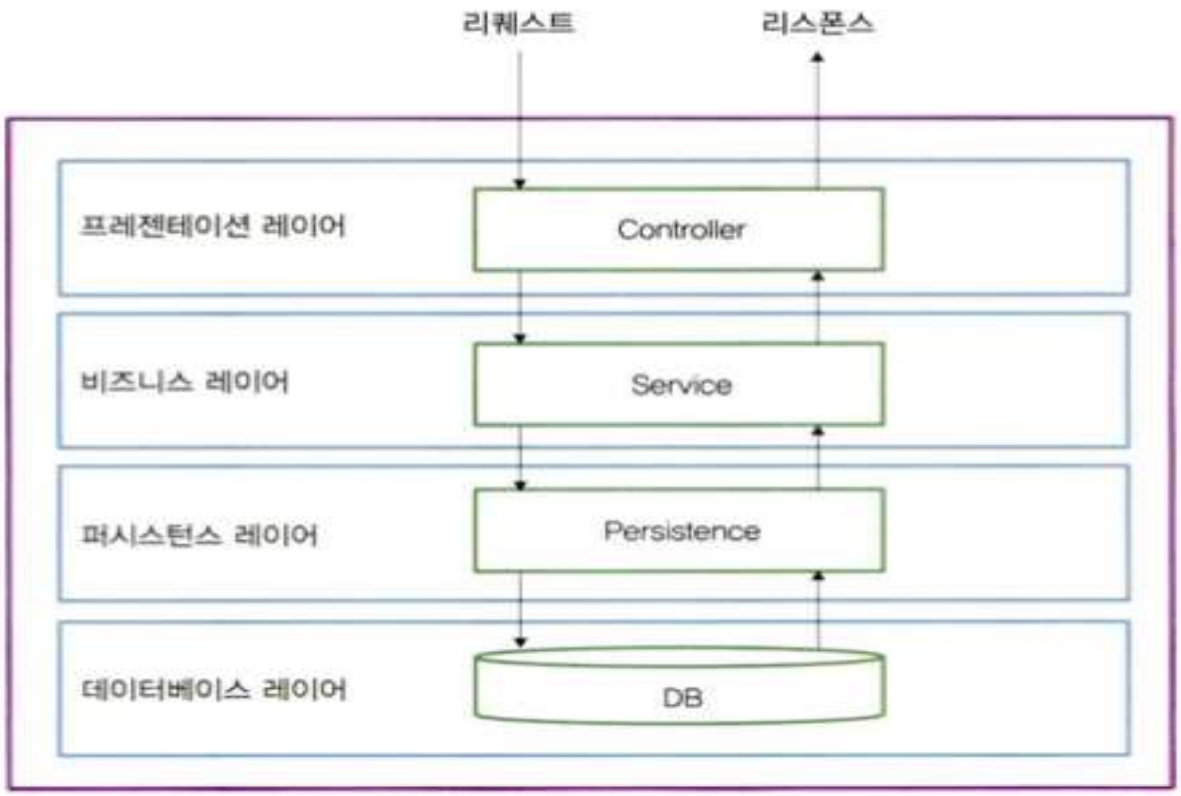
개발 환경

- REST API 테스트를 위한 애플리케이션
  - ✓ https://www.postman.com/downloads/ 에서 다운로드 받아서 설치



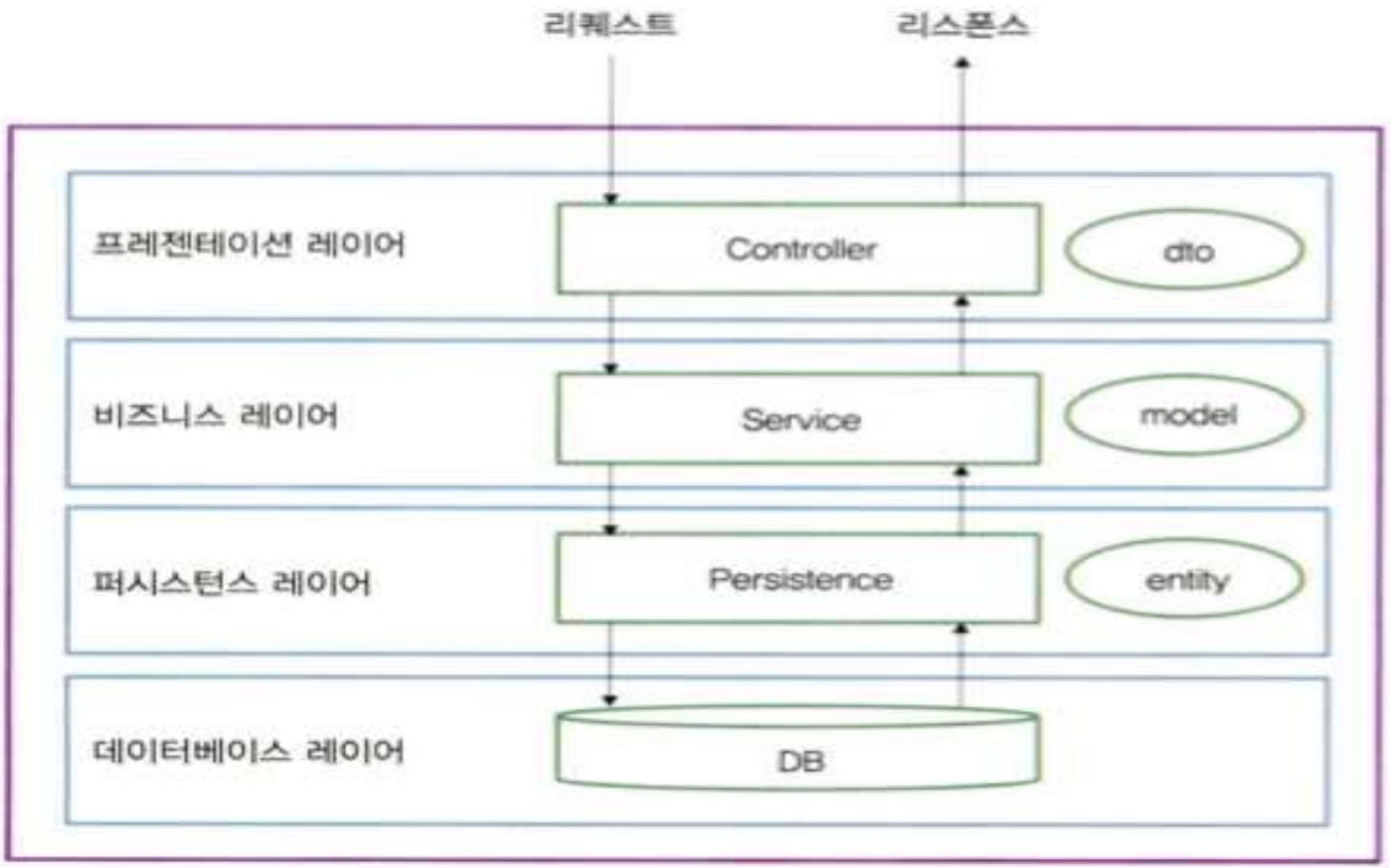
Back End

- Layered Architecture
  - ✓ 애플리케이션을 구성하는 요소들을 수평으로 나눠 관리하는 형태



Back End

- Model, Entity, DTO



Back End

- 데이터를 저장하기 위한 클래스
  - ✓ Entity
    - 관계형 데이터베이스 와 직접 매핑이 되는 데이터의 집합
  - ✓ Model
    - 비지니스 데이터를 담는 역할
  - ✓ DTO(Data Transfer Object)
    - 서비스가 요청을 처리하고 클라이언트로 반환할 때 또는 클라이언트의 데이터를 서비스에 전달할 때 모델 자체를 그대로 사용하는 경우는 거의 없음
    - 데이터를 전달하는 데 사용하는 오브젝트인 Data Transfer Object 로 변환해서 사용
    - 이유
      - 비즈니스 로직을 캡슐화 하기 위해서 – 외부 사용자에게 서비스 내부의 로직, 데이터베이스의 구조 등을 숨기기 위해서
      - 클라이언트가 필요한 정보를 모델이 전부 포함하지 않는 경우가 많고 마찬가지로 서버가 필요한 정보를 클라이언트가 전부 제공하지 않기 때문

Back End

- REST API
  - ✓ REST는 Representational State Transfer의 약자로 아키텍처 스타일(REST API Tutorial)
  - ✓ 아키텍처 스타일은 아키텍처 패턴과는 조금 다른데 아키텍처 패턴은 어떤 반복되는 문제 상황을 해결하는 도구이고 아키텍처 스타일은 반복되는 아키텍처 디자인을 의미
  - ✓ REST 아키텍처 스타일은 6가지 제약조건으로 구성되며 이 가이드 라인을 따르는 API를 RESTful API 라고 함

Back End

- REST API
  - ✓ 6가지 제약 조건
    - Client-Server
      - ◆ 클라이언트-서버라는 것은 리소스를 관리하는 서버가 존재하고 다수의 클라이언트가 리소스를 소비하려고 네트워크를 통해 서버에 접근하는 구조를 의미
      - ◆ 가장 친숙한 것이 바로 웹 애플리케이션
    - Stateless
      - ◆ 상태가 없다는 것은 클라이언트가 서버에 요청을 보낼 때 이전 요청의 영향을 받지 않음을 의미
    - Cacheable Data
      - 서버에서 리소스를 리턴할 때 캐시가 가능한지 아닌지 명시할 수 있어야 함
      - HTTP에서는 cache-control이라는 헤더에 리소스의 캐시 여부를 명시할 수 있음
    - Layered System
      - ◆ 클라이언트가 서버에 요청을 할 때 여러 개의 레이어로 된 서버를 거칠 수 있음
      - ◆ 서버가 인증 서버, 캐싱 서버, 로드 밸런서를 거쳐서 최종적으로 애플리케이션에 도착한다고 가정했을 때 이 사이의 레이어들은 요청과 응답에 어떤 영향을 미치지 않으며 클라이언트는 서버의 레이어 존재 유무를 알지 못함

Back End

- REST API
  - ✓ 6가지 제약 조건
    - A Uniform interface
      - 리소스에 접근할 때 인터페이스가 일관적이어야 한다는 의미
      - Todo 아이템을 가져오려고 http://fsoftwareengineer.com/todo를 사용했는데 Todo 아이템을 업데이트하는 데 http://fsoftwareengineer2.com/todo 사용해야 한다면 이는 일관적인 인터페이스가 아님
      - http://fsoftwareengineer.com/todo는 JSON 형식의 리소스를 리턴했는데 http://fsoftwareengineer.com/account는HTML을 리턴하는 경우 리턴 타입에 일관성이 있다고 할 수 없음
      - 리소스에 접근하는 방식, 요청 형식, 그리고 응답 형식 즉 URI, 요청의 형태와 응답의 형태가 애플리케이션 전반에 걸쳐 일관적이어야 한다는 것이 일관적인 인터페이스 방침
      - 서버가 리턴하는 응답에는 해당 리소스를 수정할 수 있는 충분한 정보가 있어야 하는데 Todo 아이템을 받아왔는데 ID가 없다면 이후 클라이언트는 Todo 아이템을 업데이트하거나 삭제하지 못하게 되는데 이는 리소스를 수정하는 데 충분한 정보가 부족한 것
    - Code-On-Demand
      - 선택 사항으로 클라이언트는 서버에 코드를 요청할 수 있고 서버가 리턴한 코드를 실행할 수 있음

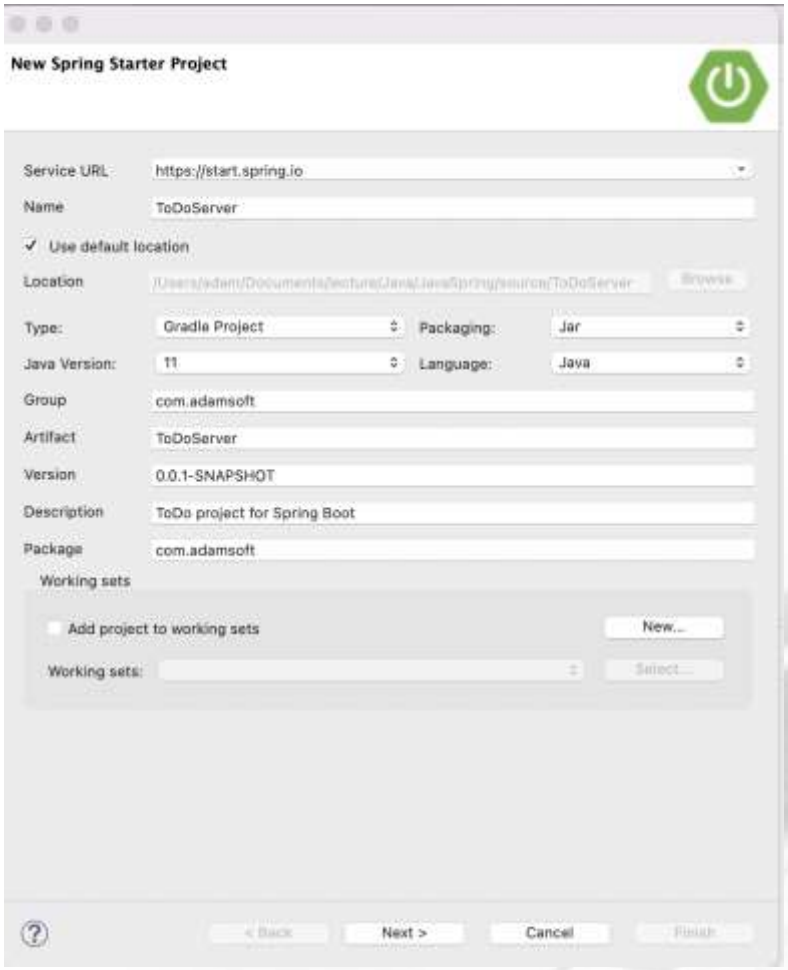
Back End

- MariaDB
  - ✓ 데이터베이스에 접속해서 데이터베이스 생성

    create database adam;

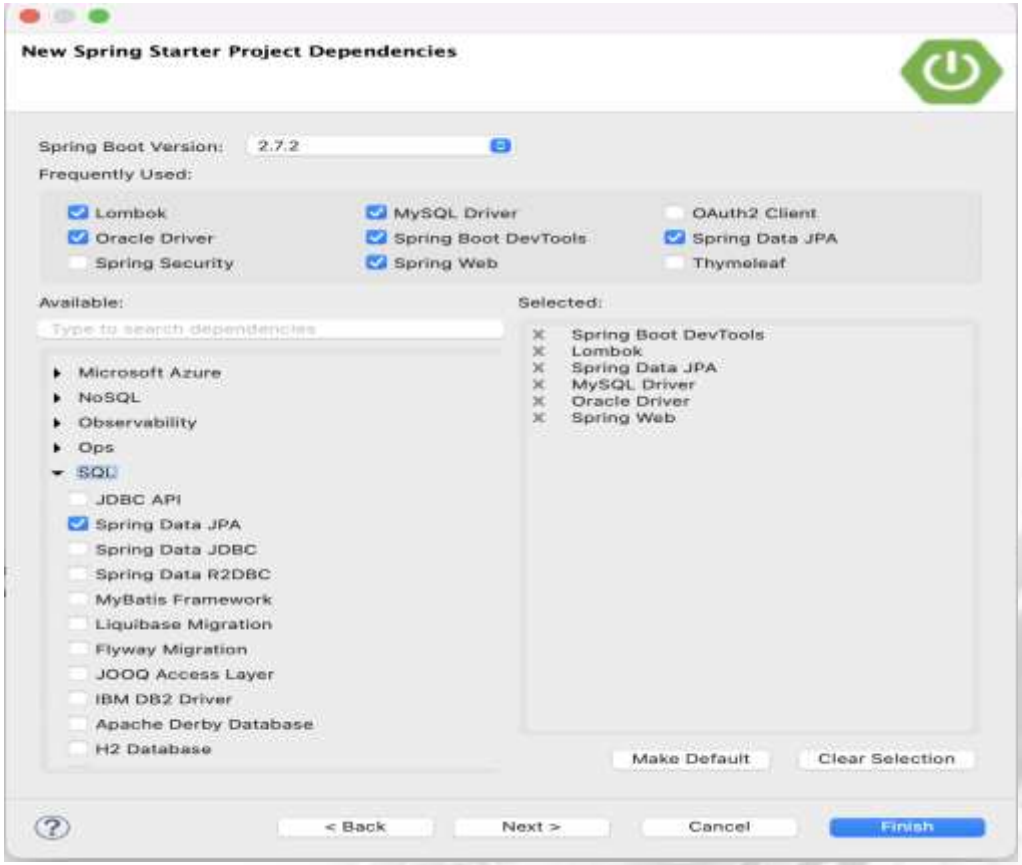
Back End

- Spring Boot 프로젝트 생성
  - ✓ 옵션 설정



Back End

- Spring Boot 프로젝트 생성 - todoserver
  - ✓ 의존성: devtools, web, jpa, lombok, MySQL



Back End

❖ 프로젝트에 application.yml 파일을 추가하고 작성

```
server:
  port: 80

spring:
  datasource:
    url: jdbc:mysql://localhost:3306/adam?useSSL=false&characterEncoding=UTF-
8&serverTimezone=UTC
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: root
    password: wnddkd
  jpa:
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        format_sql: true
        show_sql: true
        use_sql_comments: true
```

Back End

❖ 프로젝트에 application.yml 파일을 추가하고 작성

```
#Live Reload
devtools:
  livereload:
    enabled: true

logging:
  level:
    org.hibernate.type.descriptor.sql: trace
```

Back End

- Entity 작업
  - ✓ ToDoApplication.java 파일에서 JPA 감시 옵션을 설정

```
import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@EnableJpaAuditing
@SpringBootApplication
public class ToDoApplication {
        public static void main(String[] args) {
                SpringApplication.run(ToDoApplication.class, args);
        }
}
```

Back End

- Entity 작업
  - ✓ model 패키지에 공통된 속성을 소유하는 BaseEntity 생성

```
@MappedSuperclass
@EntityListeners(value = { AuditingEntityListener.class })
@Getter
abstract class BaseEntity {
    @CreatedDate
    @Column(name = "regdate", updatable = false)
    private LocalDateTime regDate;

    @LastModifiedDate
    @Column(name ="moddate" )
    private LocalDateTime modDate;
}
```

Back End

- Entity 작업
  - ✓ model 패키지에 ToDo 데이터를 위한 Entity Class 생성

```
import jakarta.persistence.*;

import lombok.*;
import org.hibernate.annotations.GenericGenerator;

@Builder
@Data
@EqualsAndHashCode(callSuper=false)
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "todo")
```

Back End

- Entity 작업
  - ✓ model 패키지에 ToDo 데이터를 위한 Entity Class 생성

```
public class ToDoEntity extends BaseEntity{
    @Id
    @GeneratedValue(generator="system-uuid")
    @GenericGenerator(name="system-uuid", strategy="uuid")
    private String id;

    @Column(length = 100, nullable = false)
    private String userId;

    @Column(length = 500, nullable = false)
    private String title;

    @Column(nullable = false)
    private boolean done;
}
```

Back End

- Entity 작업
  - ✓ 프로젝트를 실행하고 데이터베이스에 테이블이 생성되는지 확인

Back End

- Repository 작업
  - ✓ persistence 패키지에 ToDoRepository 인터페이스를 생성하고 작성

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.List;

@Repository
public interface ToDoRepository extends JpaRepository<ToDoEntity, String>{
        List<ToDoEntity> findByUserId(String userId);
}
```

Back End

- Repository 작업
  - ✓ src/main/test 디렉토리 안에 ToDoRepository를 테스트 하기 위한 클래스를 만들고 테스트

```
import java.util.List;
import java.util.Optional;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
public class RepositoryTest {

        @Autowired
        ToDoRepository toDoRepository;
```

Back End

- Repository 작업
  - ✓ src/main/test 디렉토리 안에 ToDoRepository를 테스트 하기 위한 클래스를 만들고 테스트

```
//삽입 확인
@Test
public void testInsert(){
        ToDoEntity toDoEntity =
ToDoEntity.builder().userId("adam").title("첫번째 데이터").build();
        toDoRepository.save(toDoEntity);
}

//UserId에 해당하는 데이터 전부 가져오기
//@Test
public void testSelect(){
        List<ToDoEntity> list = toDoRepository.findByUserId("adam");
        for(ToDoEntity toDo : list) {
                System.out.println(toDo);
        }
}
```

Back End

- Repository 작업
  - ✓ src/main/test 디렉토리 안에 ToDoRepository를 테스트 하기 위한 클래스를 만들고 테스트

```
//id에 해당하는 데이터 1개 가져오기
@Test
public void testDetail(){
        Optional<ToDoEntity> optional =
toDoRepository.findById("40288a8480deec5a0180deec5f640000");
        if(optional.isPresent()) {
                System.out.println(optional.get());
        }else {
                System.out.println("데이터 없음");
        }
}
```

Back End

- Repository 작업
  - ✓ src/main/test 디렉토리 안에 ToDoRepository를 테스트 하기 위한 클래스를 만들고 테스트

```
//id에 해당하는 데이터 수정
@Test
public void testUpdate(){
        ToDoEntity toDoEntity =
ToDoEntity.builder().id("40288a8480deec5a0180deec5f640000").userId("adam").title("수
정한 데이터").build();
        toDoRepository.save(toDoEntity);

        Optional<ToDoEntity> optional =
toDoRepository.findById("40288a8480deec5a0180deec5f640000");
        if(optional.isPresent()) {
                System.out.println(optional.get());
        }else {
                System.out.println("데이터 없음");
        }
}
```

Back End

- Repository 작업
  - ✓ src/main/test 디렉토리 안에 ToDoRepository를 테스트 하기 위한 클래스를 만들고 테스트

```
        //데이터 삭제
        @Test
        public void testDelete(){
                ToDoEntity toDoEntity =
ToDoEntity.builder().id("40288a8480deec5a0180deec5f640000").build();
                toDoRepository.delete(toDoEntity);

                Optional<ToDoEntity> optional =
toDoRepository.findById("40288a8480deec5a0180deec5f640000");
                if(optional.isPresent()) {
                        System.out.println(optional.get());
                }else {
                        System.out.println("데이터 없음");
                }
        }
}
```

Back End

- DTO 작업
  - ✓ dto 패키지에 ToDoDTO 클래스를 생성하고 작성

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@NoArgsConstructor
@AllArgsConstructor
@Data
public class ToDoDTO {
    private String id;
    private String userId;
    private String title;
    private boolean done;
```

Back End

- DTO 작업
  - ✓ dto 패키지에 ToDoDTO 클래스를 생성하고 작성

```
    public ToDoDTO(final ToDoEntity entity) {
        this.id = entity.getId();
        this.userId = entity.getUserId();
        this.title = entity.getTitle();
        this.done = entity.isDone();
    }

    public static ToDoEntity toEntity(final ToDoDTO dto) {
        return ToDoEntity.builder()
                .id(dto.getId())
                .userId(dto.getUserId())
                .title(dto.getTitle())
                .done(dto.isDone())
                .build();
    }
}
```

Back End

- DTO 작업
  - ✓ dto 패키지에 응답에 사용할 ResponseDTO 클래스를 생성하고 작성

```
import java.util.List;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Builder
@NoArgsConstructor
@AllArgsConstructor
@Data
public class ResponseDTO<T> {
    private String error;
    private List<T> data;
}
```

Back End

- Service 작업
  - ✓ service 패키지에 ToDoService 인터페이스를 생성하고 작성

```
import java.util.List;

public interface ToDoService {
    public String testService();

    public List<ToDoDTO> create(final ToDoDTO dto);

    public List<ToDoDTO> retrieve(final String userId);

    public List<ToDoDTO> update(final ToDoDTO dto);

    public List<ToDoDTO> delete(final ToDoDTO dto);
}
```

Back End

- Service 작업
  - ✓ service 패키지에 ToDoService 인터페이스를 implements 한 ToDoServiceImpl 클래스를 생성하고 작성

```
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;
import org.springframework.stereotype.Service;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;

@Slf4j
@Service
@RequiredArgsConstructor
public class ToDoServiceImpl implements ToDoService {

    private final ToDoRepository repository;

    public String testService() {
        return "Test Service";
    }
```

Back End

- Service 작업
  - ✓ service 패키지에 ToDoService 인터페이스를 implements 한 ToDoServiceImpl 클래스를 생성하고 작성

```
private void validate(final ToDoDTO dto) {
    if(dto == null) {
        log.warn("DTO cannot be null.");
        throw new RuntimeException("DTO cannot be null.");
    }

    if(dto.getUserId() == null) {
        log.warn("Unknown user.");
        throw new RuntimeException("Unknown user.");
    }
}
```

Back End

- Service 작업
  - ✓ service 패키지에 ToDoService 인터페이스를 implements 한 ToDoServiceImpl 클래스를 생성하고 작성

```
public List<ToDoDTO> create(final ToDoDTO dto) {
    validate(dto);
    ToDoEntity entity = ToDoDTO.toEntity(dto);
    repository.save(entity);
    log.info("Entity Id : {} is saved.", entity.getId());
    List<ToDoDTO> list =
repository.findByUserId(entity.getUserId()).stream().map(toDoEntity ->
        new ToDoDTO(toDoEntity)).collect(Collectors.toList());
    return list;
}
```

Back End

- Service 작업
  - ✓ service 패키지에 ToDoService 인터페이스를 implements 한 ToDoServiceImpl 클래스를 생성하고 작성

```
public List<ToDoDTO> retrieve(final String userId) {
    List<ToDoDTO> list = repository.findByUserId(userId).stream().map(entity -> new
ToDoDTO(entity)).collect(Collectors.toList());
    return list;
}
```

Back End

- Service 작업
  - ✓ service 패키지에 ToDoService 인터페이스를 implements 한 ToDoServiceImpl 클래스를 생성하고 작성

```
public List<ToDoDTO> update(final ToDoDTO dto) {
    validate(dto);
    final Optional<ToDoEntity> original = repository.findById(dto.getId());
    original.ifPresent(todo -> {
        todo.setTitle(dto.getTitle());
        todo.setDone(dto.isDone());
        repository.save(todo);
    });

    return retrieve(dto.getUserId());
}
```

Back End

- Service 작업
  - ✓ service 패키지에 ToDoService 인터페이스를 implements 한 ToDoServiceImpl 클래스를 생성하고 작성

```
    public List<ToDoDTO> delete(final ToDoDTO dto) {
        validate(dto);

        try {
            repository.delete(ToDoDTO.toEntity(dto));
        } catch(Exception e) {
            log.error("error deleting entity ", dto.getId(), e);
            throw new RuntimeException("error deleting entity " + dto.getId());
        }
        return retrieve(dto.getUserId());
    }
}
```

Back End

- Controller 작업
  - ✓ controller 패키지에 ToDoController 클래스를 만들고 요청을 처리할 코드를 작성

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.ArrayList;
import java.util.List;

@RestController
@RequestMapping("todo")
public class ToDoController {

    @Autowired
    private ToDoService service;
```

Back End

- Controller 작업
  - ✓ controller 패키지에 ToDoController 클래스를 만들고 요청을 처리할 코드를 작성

```
@GetMapping("/test")
 public ResponseEntity<?> testTodo() {
        String str = service.testService(); // 테스트 서비스 사용
        List<String> list = new ArrayList<>();
        list.add(str);
        ResponseDTO<String> response =
ResponseDTO.<String>builder().data(list).build();
        // ResponseEntity.ok(response) 를 사용해도 상관 없음
        return ResponseEntity.ok().body(response);
    }
```

Back End

- Controller 작업
  - ✓ controller 패키지에 ToDoController 클래스를 만들고 요청을 처리할 코드를 작성

```
@PostMapping
    public ResponseEntity<?> createTodo(@RequestBody ToDoDTO dto) {
        try {
            String temporaryUserId = "temporary-user"; // temporary user id.
            dto.setUserId(temporaryUserId);
            List<ToDoDTO> list = service.create(dto);
            ResponseDTO<ToDoDTO> response =
ResponseDTO.<ToDoDTO>builder().data(list).build();
            return ResponseEntity.ok().body(response);
        } catch (Exception e) {
            String error = e.getMessage();
            ResponseDTO<ToDoDTO> response =
ResponseDTO.<ToDoDTO>builder().error(error).build();
            return ResponseEntity.badRequest().body(response);
        }
    }
```

Back End

- Controller 작업
  - ✓ controller 패키지에 ToDoController 클래스를 만들고 요청을 처리할 코드를 작성

```
@GetMapping
  public ResponseEntity<?> retrieveTodoList() {
      String temporaryUserId = "temporary-user"; // temporary user id.
      List<ToDoDTO> list = service.retrieve(temporaryUserId);
      ResponseDTO<ToDoDTO> response =
ResponseDTO.<ToDoDTO>builder().data(list).build();
      return ResponseEntity.ok().body(response);
  }
```

Back End

- Controller 작업
  - ✓ controller 패키지에 ToDoController 클래스를 만들고 요청을 처리할 코드를 작성

```
@PutMapping
public ResponseEntity<?> updateTodo(@RequestBody ToDoDTO dto) {
    String temporaryUserId = "temporary-user"; // temporary user id.
    dto.setUserId(temporaryUserId);
    List<ToDoDTO> list = service.update(dto);
    ResponseDTO<ToDoDTO> response =
ResponseDTO.<ToDoDTO>builder().data(list).build();
    return ResponseEntity.ok().body(response);
}
```

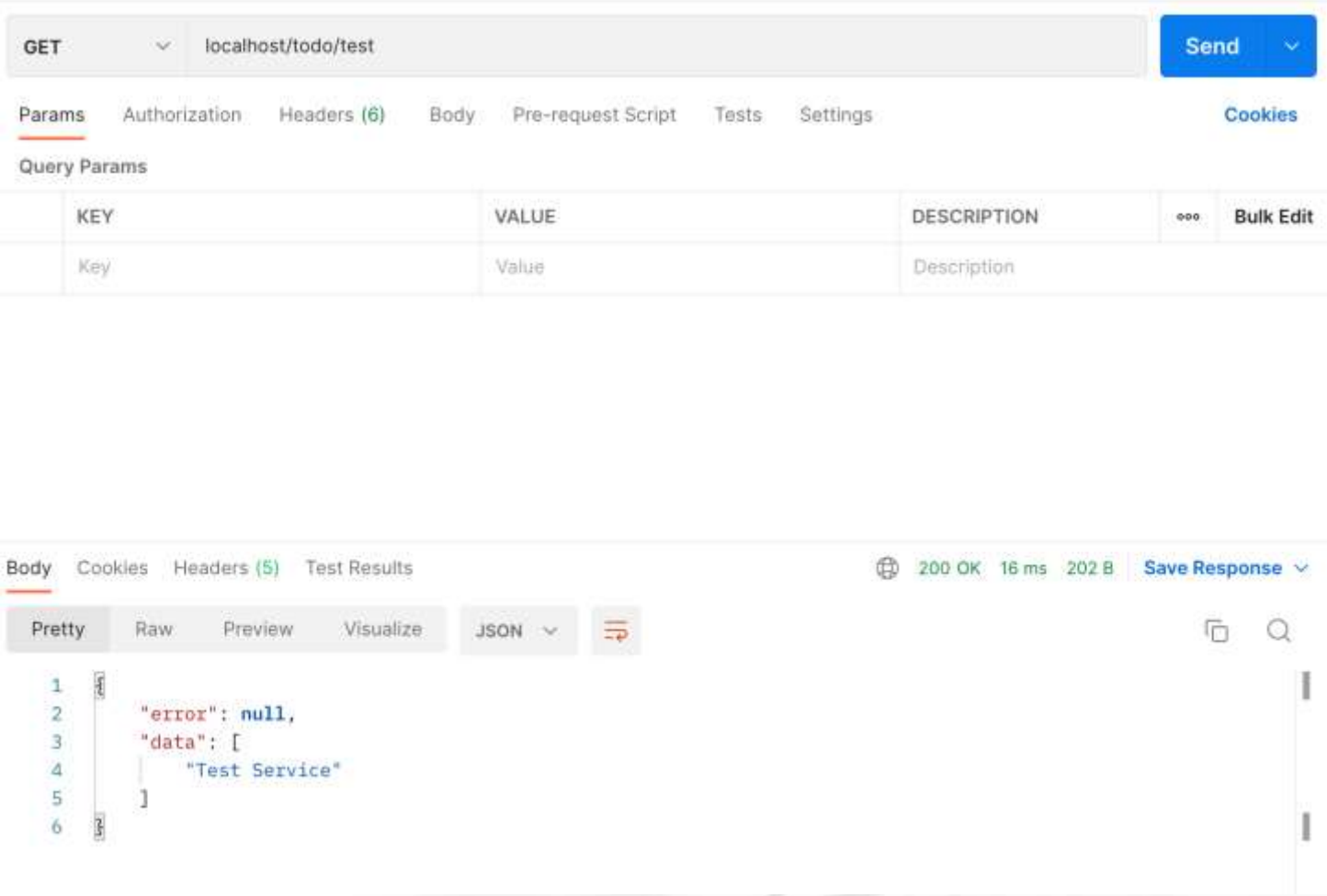
Back End

- Controller 작업
  - ✓ controller 패키지에 ToDoController 클래스를 만들고 요청을 처리할 코드를 작성

```
@DeleteMapping
    public ResponseEntity<?> deleteTodo(@RequestBody ToDoDTO dto) {
        try {
            String temporaryUserId = "temporary-user"; // temporary user id.
            dto.setUserId(temporaryUserId);
            List<ToDoDTO> list = service.delete(dto);
            ResponseDTO<ToDoDTO> response =
ResponseDTO.<ToDoDTO>builder().data(list).build();
            return ResponseEntity.ok().body(response);
        } catch (Exception e) {
            String error = e.getMessage();
            ResponseDTO<ToDoDTO> response =
ResponseDTO.<ToDoDTO>builder().error(error).build();
            return ResponseEntity.badRequest().body(response);
        }
    }
}
```

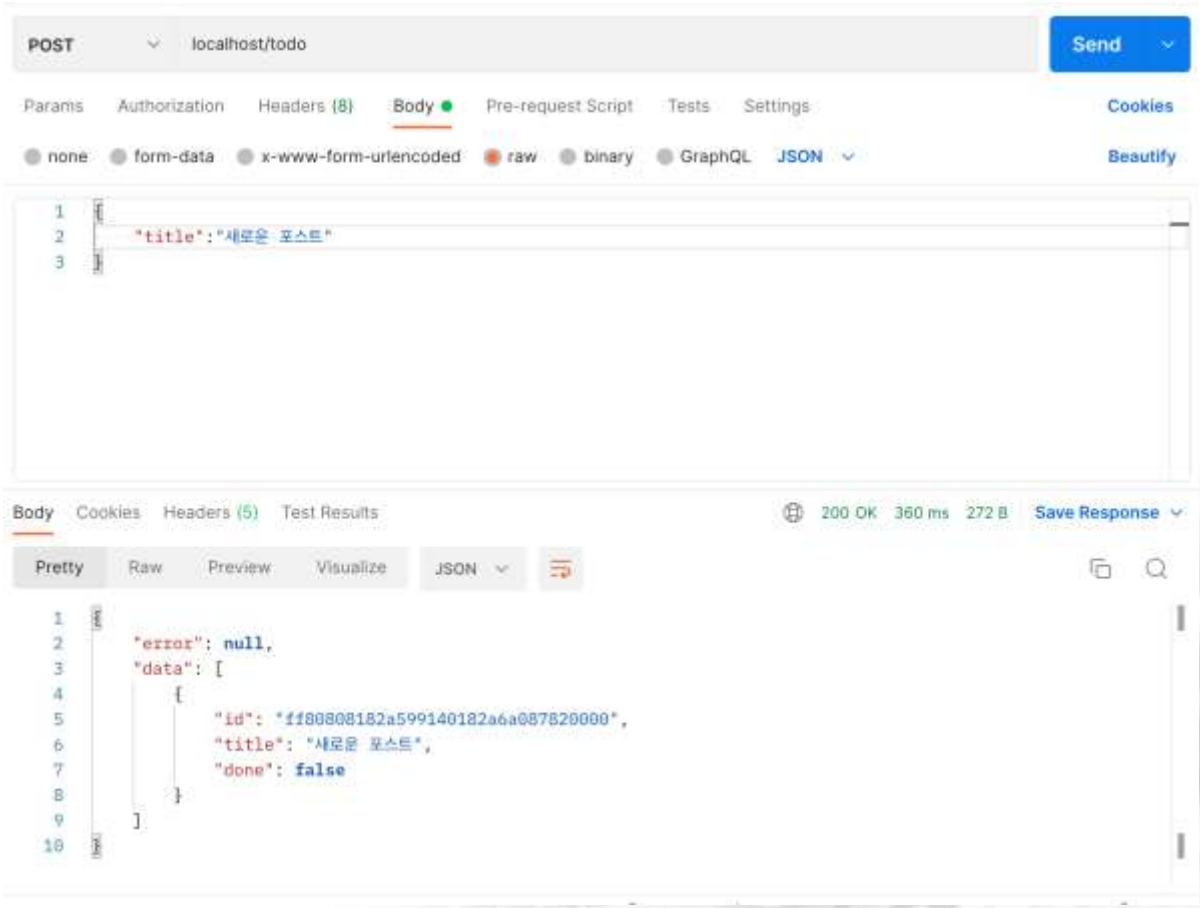
Back End

- 테스트 – PostMan
  - ✓ test



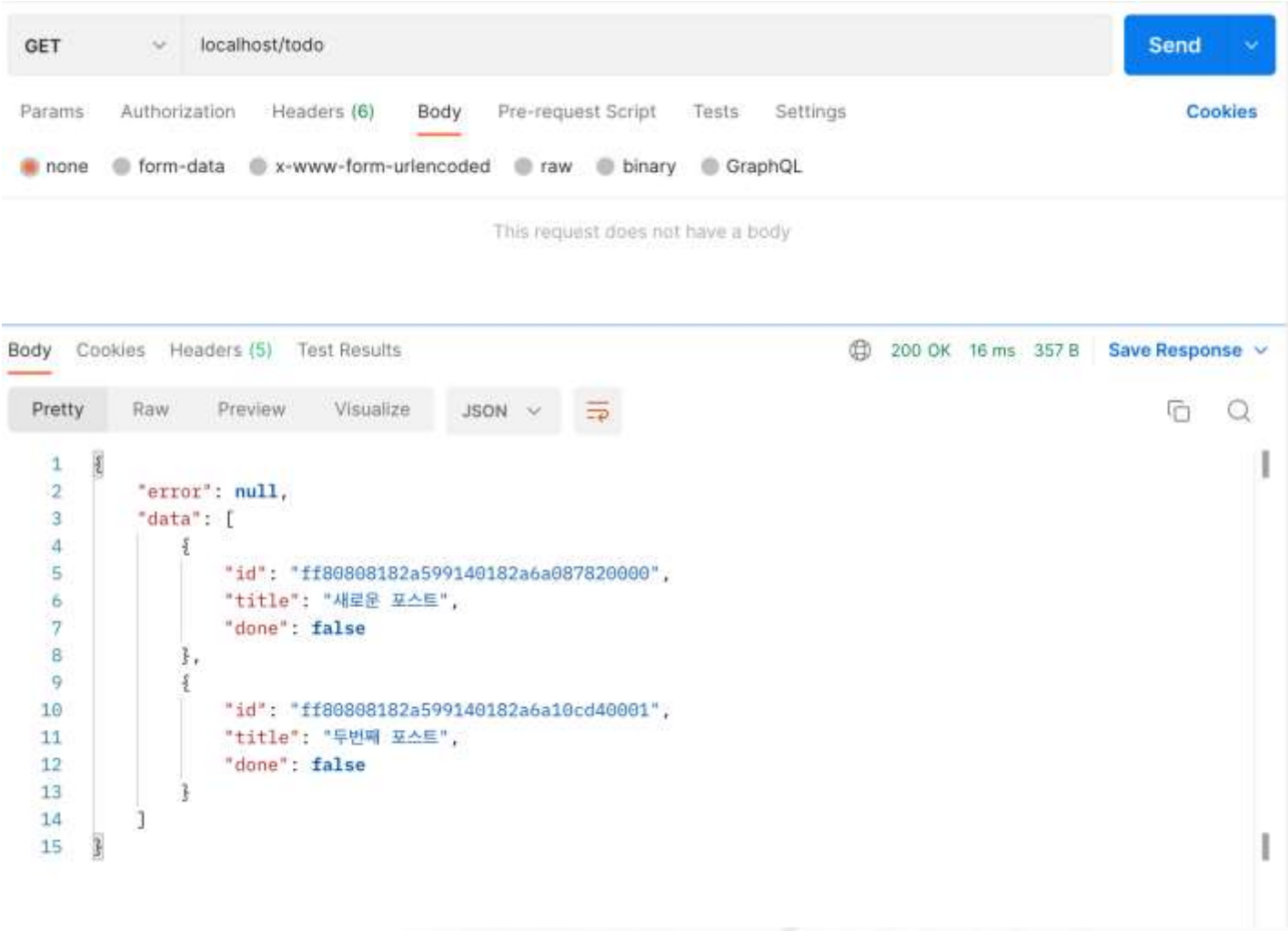
Back End

- 테스트 – PostMan
  - ✓ 데이터 삽입



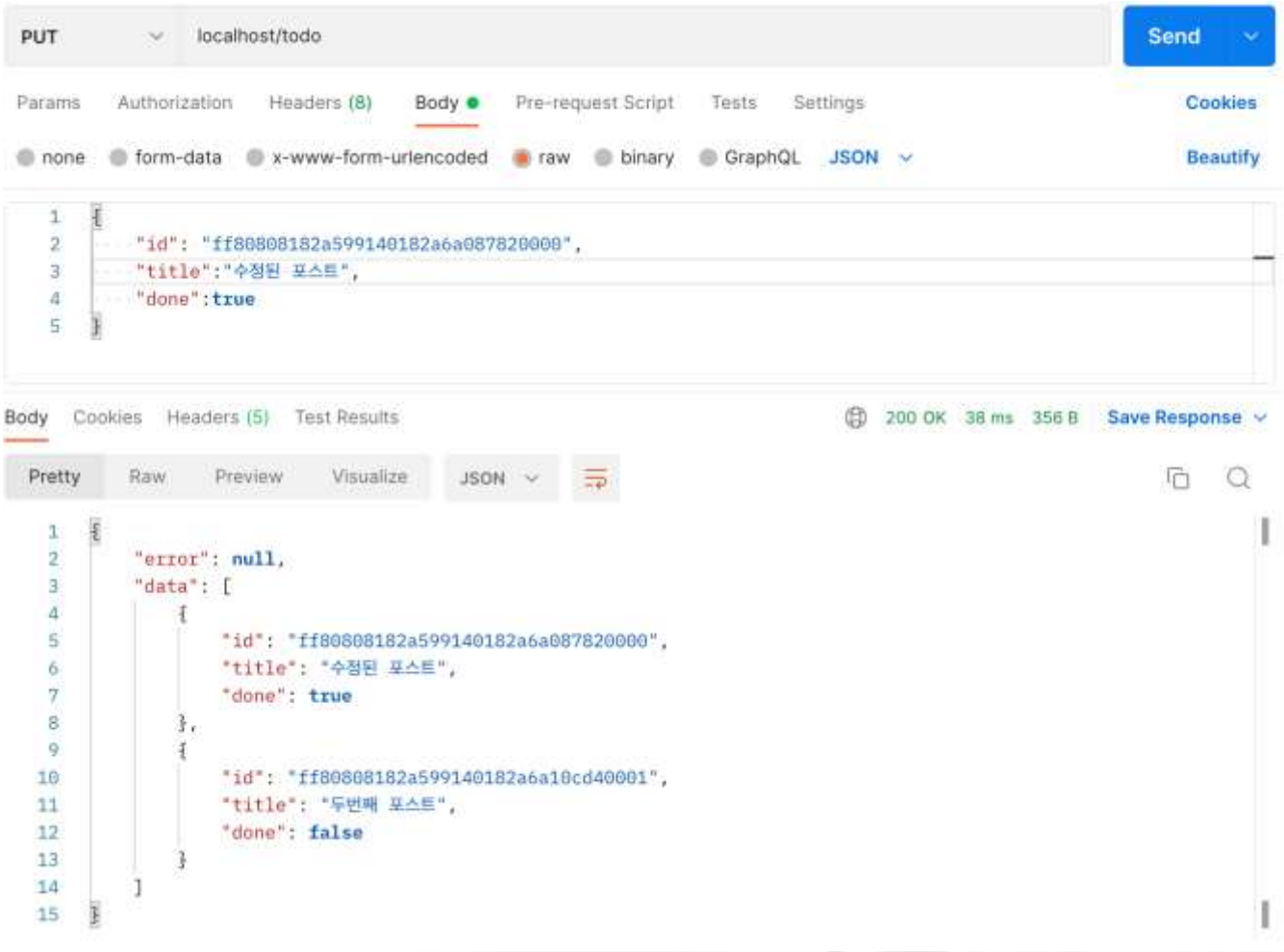
Back End

- 테스트 – PostMan
  - ✓ 데이터 조회



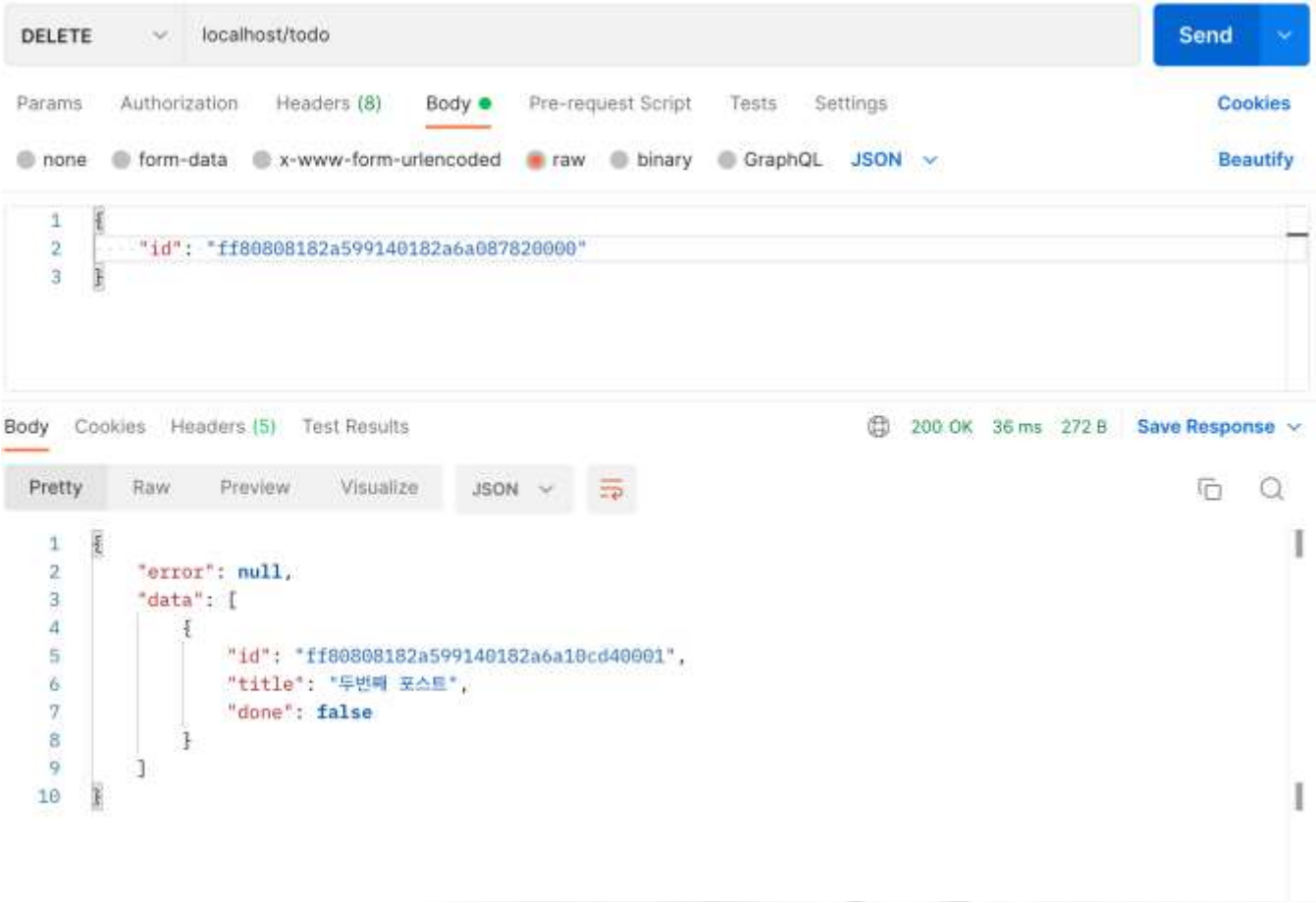
Back End

- 테스트 – PostMan
  - ✓ 데이터 수정



Back End

- 테스트 – PostMan
  - ✓ 데이터 삭제



Front End

- SPA
  - ✓ React.js나 Angularjs, Vuejs는 대중적인 SPA 라이브러리/프레임워크
  - ✓ Single Page Application의 약자로 한 번 웹 페이지를 로딩하면 사용자가 임의로 새로 고침 하지 않는 이상 페이지를 새로 로딩하지 않는 애플리케이션

Front End

- 개발 환경
  - ✓ Node 설치
  - ✓ IDE - VS Code

Front End

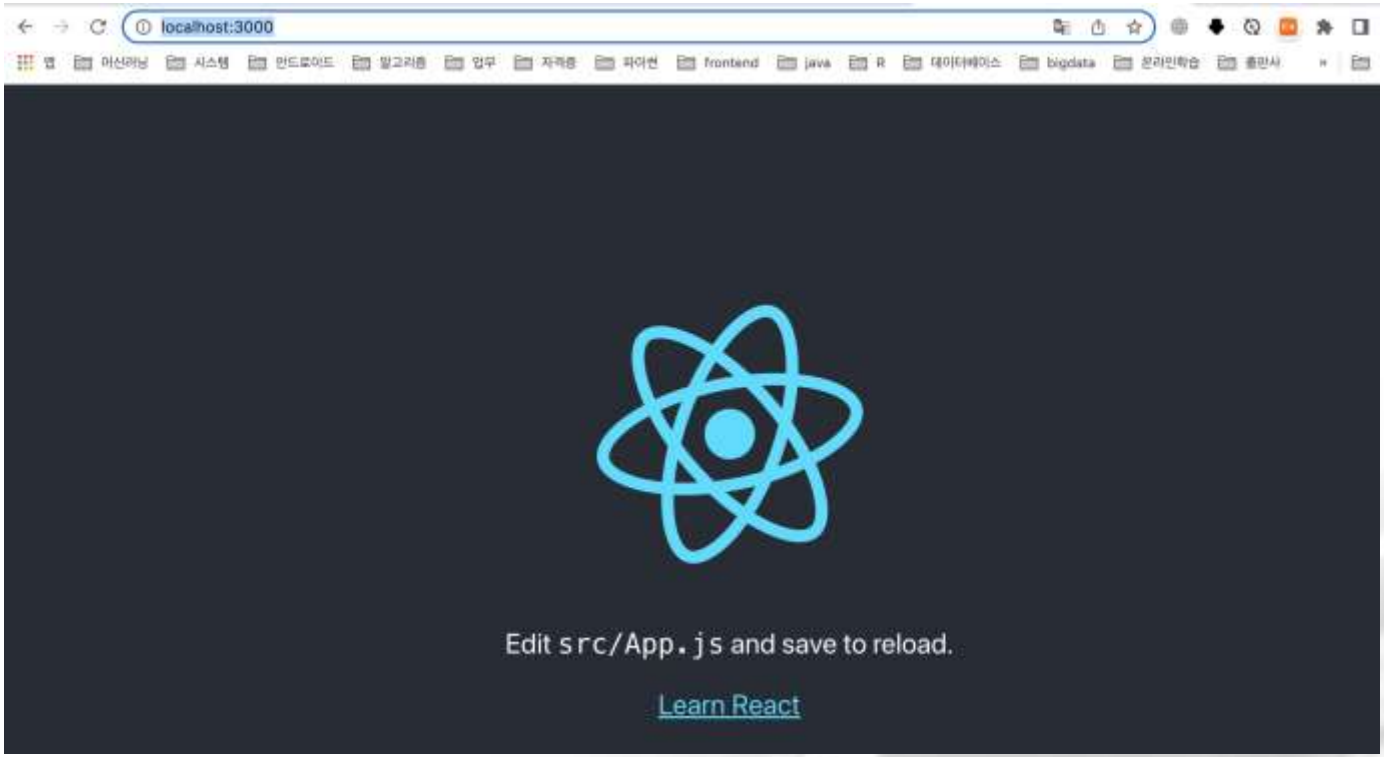
- 프로젝트 생성 및 실행
  - ✓ react 프로젝트 생성: todo-react-app

    npx create-react-app todo-react-app
  - ✓ 애플리케이션 실행

    npm start

Front End

- 프로젝트 생성 및 실행
  - ✓ 브라우저 확인



Front End

- UI 개발을 빠르게 하기 위한 material-ui 패키지
  - ✓ https://mui.com/
  - ✓ 설치

```
npm install --save --legacy-peer-deps @material-ui/core
npm install --save --legacy-peer-deps @material-ui/icons
```

Front End

- ToDo 목록 보기
  - ✓ src 디렉토리에 ToDo.jsx 파일을 추가하고 작성

```
import React from "react";

class ToDo extends React.Component {

  render() {
    return (
        <div className="ToDo">
            <input type="checkbox" id="todo0" name="todo0" value="todo0" />
            <lable for="todo0">ToDo 컴포넌트 만들기</lable>
        </div>
    );
  }
}

export default ToDo;
```

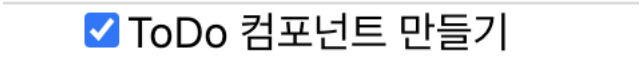
Front End

- ToDo 목록 보기
  - ✓ App.js 파일을 수정

```
import React from "react";
import ToDo from "./ToDo";
import './App.css';

function App() {
  return (
    <div className="App">
      <ToDo/>
    </div>
  );
}

export default App;
```

Front End

- ToDo 목록 보기
  - ✓ 브라우저 확인



Front End

- ToDo 목록 보기
  - ✓ ToDo.jsx 파일 수정 – Props 와 State

```
import React, {useState} from "react";

const ToDo = (props) => {
    // 새로운 state 변수를 선언하고, count라 부르겠습니다.
    const [item, setItem] = useState(props.item);

    return (
        <div className="ToDo">
        <input
            type="checkbox"
            id={item.id}
            name={item.id}
            checked={item.done}/>
        <label id={item.id}>{item.title}</label>
    </div>
    );
  }

  export default ToDo;
```

Front End

- ToDo 목록 보기
  - ✓ App.js 파일 수정 – Props 와 State

```
import React, {useState} from "react";
import ToDo from "./ToDo";
import './App.css';

function App() {
  const [item, setItem] = useState({ id: 0, title: "Hello React", done: true});
  return (
    <div className="App">
      <ToDo item={item} />
    </div>
  );
}

export default App;
```

Front End

- ToDo 목록 보기
  - ✓ 브라우저 확인 – Props 와 State

Front End

- ToDo 목록 보기
  - ✓ App.js 파일 수정 – ToDo 리스트

```
import React, {useState} from "react";
import ToDo from "./ToDo";
import './App.css';

function App() {
  const [items, setItems] = useState([{ id: 0, title: "Java", done: true}, { id: 1, title:
"Python", done: true}]);

  return (
    <div className="App">
      {items.map((item, idx) => (
        <ToDo item={item} key={item.id} />
      ))}
    </div>
  );
}

export default App;
```

Front End

- ToDo 목록 보기
  - ✓ 브라우저 확인 – ToDo 리스트

Front End

- ToDo 목록 보기
  - ✓ ToDo.jsx 파일 수정 – material ui를 이용한 디자인

```
import React, {useState} from "react";

import {
    ListItem,
    ListItemText,
    InputBase,
    Checkbox,
} from "@material-ui/core";
```

Front End

- ToDo 목록 보기
  - ✓ ToDo.jsx 파일 수정 – material ui를 이용한 디자인

```
const ToDo = (props) => {
    const [item, setItem] = useState(props.item);
    return (
        <ListItem>
        <Checkbox checked={item.done} />
        <ListItemText>
            <InputBase
                inputProps={{ "aria-label": "naked" }}
                type="text"
                id={item.id}
                name={item.id}
                value={item.title}
                multiline={true}
                fullWidth={true}
            />
        </ListItemText>
    </ListItem>
    );
}
export default ToDo;
```

Front End

- ToDo 목록 보기
  - ✓ App.js 파일 수정 – material ui를 이용한 디자인

```
import React, {useState} from "react";
import ToDo from "./ToDo";
import './App.css';
import {Paper, List} from "@material-ui/core"
```

Front End

- ToDo 목록 보기
  - ✓ App.js 파일 수정 – material ui를 이용한 디자인

```
function App() {
  const [items, setItems] = useState([{ id: 0, title: "Java", done: true}, { id: 1, title:
"Python", done: true}]);
  return (
    <div>
     {
       items.length > 0  ?
       (<Paper style={{ margin: 16 }}>
       <List>
         {items.map((item, idx) => (
           <ToDo item={item} />
         ))}
       </List>
       </Paper>)
       :null
     }
    </div>
  );
}
export default App;
```

Front End

- ToDo 목록 보기
  - ✓ 웹 브라우저에서 확인 – material ui를 이용한 디자인

Front End

- ToDo 추가
  - ✓ AddToDo.jsx 파일을 추가하고 작성

```
import React, {useState} from "react"
import { TextField, Paper, Button, Grid } from "@material-ui/core";

const AddTodo = (props) =>  {
  const [item, setItem] = useState({ title: "" } )

    return (
      <Paper style={{ margin: 16, padding: 16 }}>
        <Grid container>
          <Grid xs={11} md={11} item style={{ paddingRight: 16 }}>
            <TextField
              placeholder="Add Todo here"
              fullWidth
            />
          </Grid>
```

Front End

- ToDo 추가
  - ✓ AddToDo.jsx 파일을 추가하고 작성

```
            <Grid xs={1} md={1} item>
              <Button
                fullWidth
                color="secondary"
                letiant="outlined"
              >
                +
              </Button>
            </Grid>
          </Grid>
        </Paper>
      );
    }

    export default AddTodo;
```

Front End

- ToDo 추가
  - ✓ App.js 파일 수정

```
import React, {useState} from "react";
import ToDo from "./ToDo";
import AddToDo from "./AddToDo";
import './App.css';
import {Paper, List, Container} from "@material-ui/core"

function App() {
  const [items, setItems] = useState([{ id: 0, title: "Java", done: true}, { id: 1, title:
"Python", done: true}]);
```

Front End

- ToDo 추가
  - ✓ App.js 파일 수정

```
return (
  <div>
    <Container maxWidth="md">
    <AddToDo/>
    {
      items.length > 0  ?
      (<Paper style={{ margin: 16 }}>
      <List>
        {items.map((item, idx) => (
          <ToDo item={item} />
        ))}
      </List>
      </Paper>)
      :null
    }
    </Container>
  </div>
);
}
export default App;
```

Front End

- ToDo 추가
  - ✓ 브라우저 확인

Front End

- ToDo 추가
  - ✓ App.js 파일 수정 – 이벤트 처리

```
import React, {useState} from "react";
import ToDo from "./ToDo";
import AddToDo from "./AddToDo";
import './App.css';
import {Paper, List, Container} from "@material-ui/core"

function App() {
  const [items, setItems] = useState([{ id: 0, title: "Java", done: true}, { id: 1, title:
"Python", done: true}]);

    //데이터 추가를 위한 함수
    const add = (item) => {
      item.id = "ID-" + items.length; // key를 위한 id추가
      item.done = false; // done 초기화
      setItems(items.concat(item)); // 업데이트는 반드시 this.setState로 해야됨.
      console.log("items : ", items);
    };
```

Front End

- ToDo 추가
  - ✓ App.js 파일 수정 – 이벤트 처리

```
return (
  <div>
    <Container maxWidth="md">
    <AddToDo add={add}/>
    {
      items.length > 0  ?
      (<Paper style={{ margin: 16 }}>
      <List>
        {items.map((item, idx) => (
          <ToDo item={item} />
        ))}
      </List>
      </Paper>)
      :null
    }
    </Container>
  </div>
);
}

export default App;
```

Front End

- ToDo 추가
  - ✓ AddToDo.jsx 파일 수정 – 이벤트 처리

```
import React, {useState} from "react"
import { TextField, Paper, Button, Grid } from "@material-ui/core";

const AddTodo = (props) =>  {
    const [title, setTitle] = useState('');

    const onChangeTitle = (e) => {
        setTitle(e.target.value);
    };

    //+버튼을 눌렀을 때 호출
    const onButtonClick = () => {
        props.add({"title":title});
        setTitle("");
    };

    //Enter를 눌렀을 때 처리
    const enterKeyEventHandler = (e) => {
        if (e.key === "Enter") {
            onButtonClick();
        }
    };
```

Front End

- ToDo 추가
  - ✓ AddToDo.jsx 파일 수정 – 이벤트 처리

```
return (
  <Paper style={{ margin: 16, padding: 16 }}>
    <Grid container>
      <Grid xs={11} md={11} item style={{ paddingRight: 16 }}>
        <TextField
          placeholder="Add Todo here"
          fullWidth
          onChange={onChangeTitle}
          value={title}
          onKeyPress={enterKeyEventHandler}

        />
      </Grid>
```

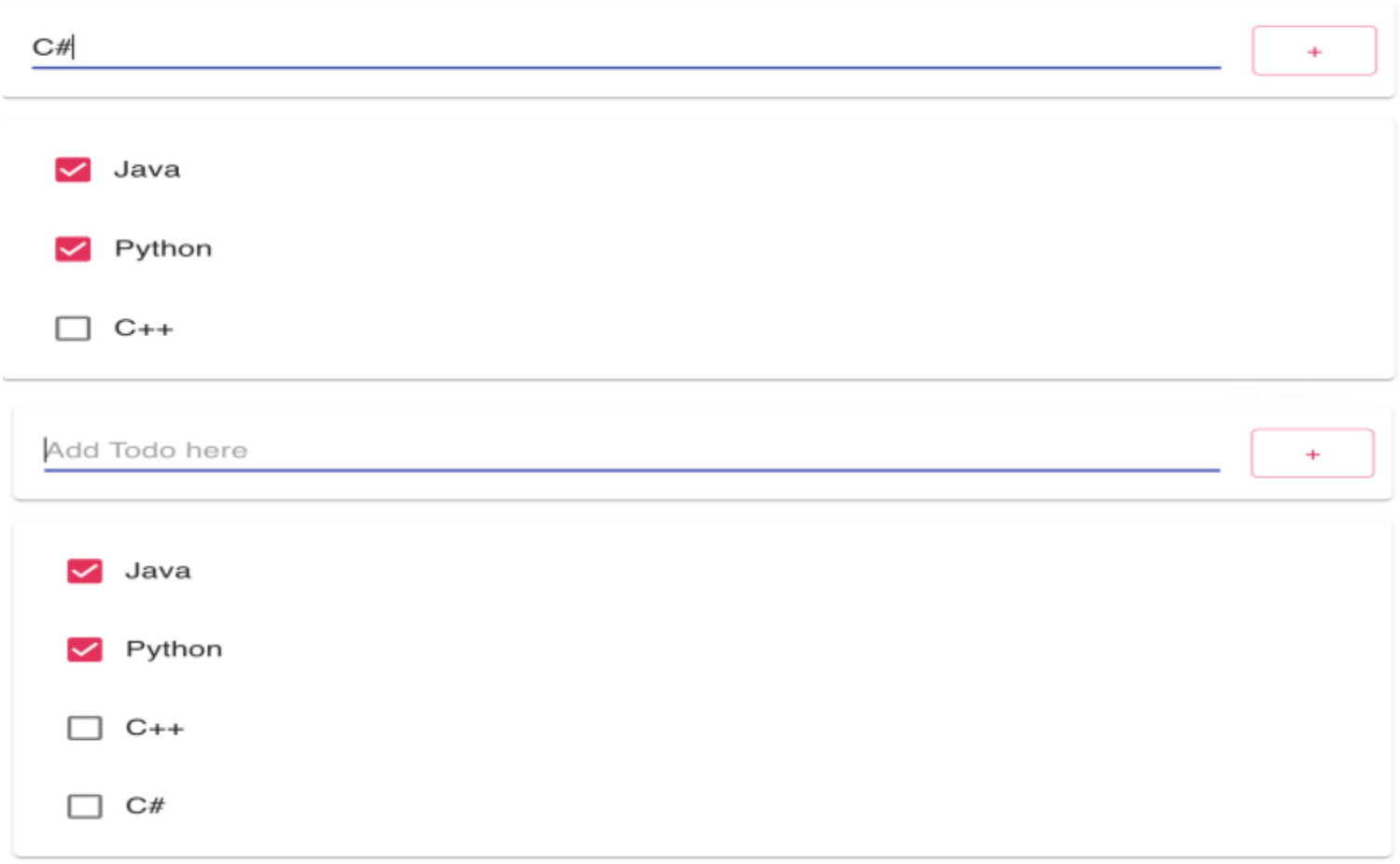
Front End

- ToDo 추가
  - ✓ AddToDo.jsx 파일 수정 – 이벤트 처리

```
                <Grid xs={1} md={1} item>
                  <Button
                    fullWidth
                    color="secondary"
                    letiant="outlined"
                    onClick={onButtonClick}

                  >
                    +
                  </Button>
                </Grid>
              </Grid>
            </Paper>
        );
    }

    export default AddTodo;
```

Front End

- ToDo 추가
  - ✓ 브라우저 확인 – 이벤트 처리



Front End

- ToDo 삭제
  - ✓ ToDo.jsx 파일 수정 – 삭제 아이콘 출력

```
import React from "react";

import {
    ListItem,
    ListItemText,
    InputBase,
    Checkbox,
    ListItemSecondaryAction,
    IconButton
} from "@material-ui/core";

import DeleteOutlined from "@material-ui/icons/DeleteOutlined";
```

Front End

- ToDo 삭제
  - ✓ ToDo.jsx 파일 수정 – 삭제 아이콘 출력

```
import React, {useState} from "react";

import {
    ListItem,
    ListItemText,
    InputBase,
    Checkbox,
    ListItemSecondaryAction,
    IconButton
} from "@material-ui/core";

import DeleteOutlined from "@material-ui/icons/DeleteOutlined";
```

Front End

- ToDo 삭제
  - ✓ ToDo.jsx 파일 수정 – 삭제 아이콘 출력

```
const ToDo = (props) => {
    const [item, setItem] = useState(props.item);
 return (
    <ListItem>
    <Checkbox checked={item.done} />
    <ListItemText>
        <InputBase
            inputProps={{ "aria-label": "naked" }}
            type="text"
            id={item.id}
            name={item.id}
            value={item.title}
            multiline={true}
            fullWidth={true}
        />
    </ListItemText>
```

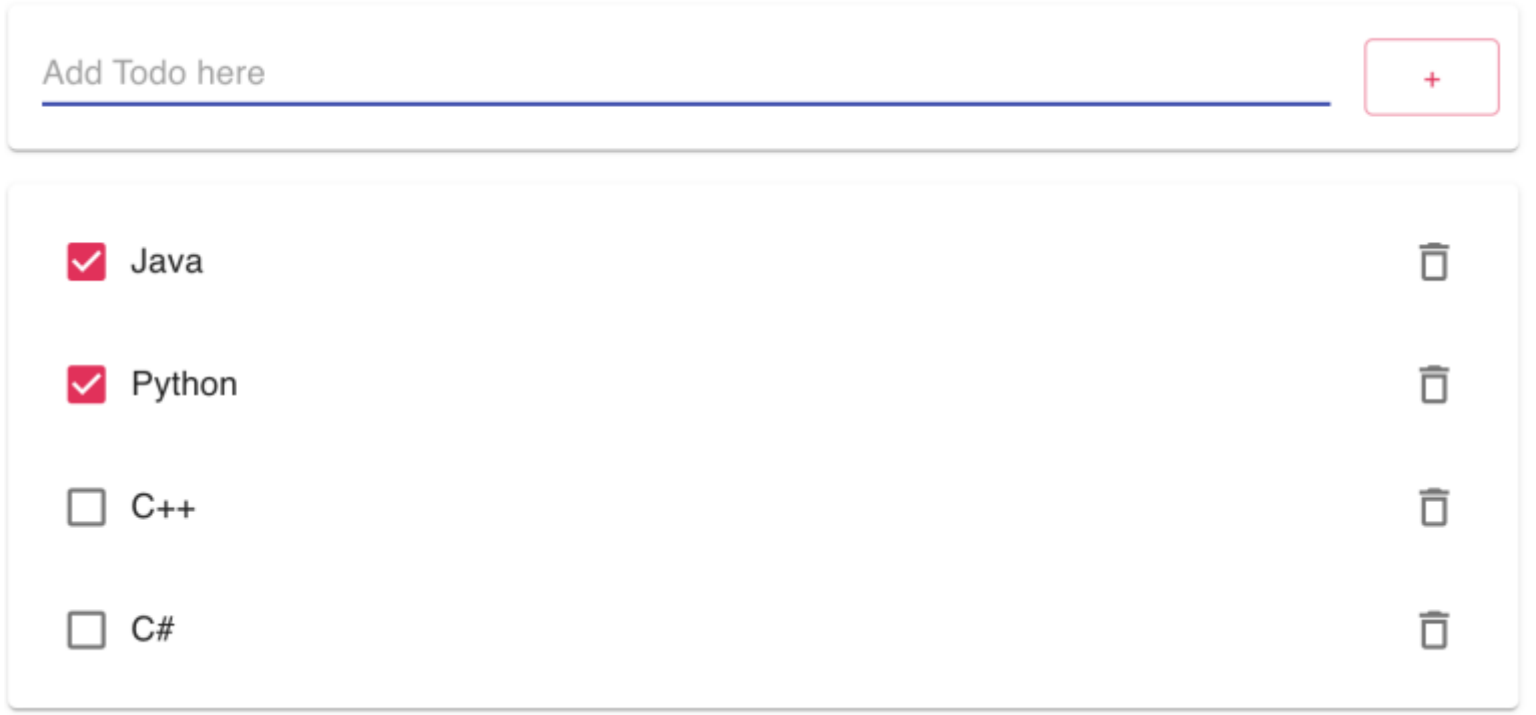
Front End

- ToDo 삭제
  - ✓ ToDo.jsx 파일 수정 – 삭제 아이콘 출력

```
            <ListItemSecondaryAction>
              <IconButton aria-label="Delete Todo">
                <DeleteOutlined />
              </IconButton>
            </ListItemSecondaryAction>

          </ListItem>
        );
      }
      export default ToDo;
```

Front End

- ToDo 삭제
  - ✓ 브라우저에서 확인 – 삭제 아이콘 출력



Front End

- ToDo 삭제
  - ✓ App.js 파일 수정 – 삭제 구현

```
import React, {useState} from "react";
import ToDo from "./ToDo";
import AddToDo from "./AddToDo";
import './App.css';
import {Paper, List, Container} from "@material-ui/core"

function App() {
  const [items, setItems] = useState([{ id: 0, title: "Java", done: true}, { id: 1, title:
"Python", done: true}]);

    //데이터 추가를 위한 함수
    const add = (item) => {
      item.id = "ID-" + items.length; // key를 위한 id추가
      item.done = false; // done 초기화
      setItems(items.concat(item)); // 업데이트는 반드시 this.setState로 해야됨.
      console.log("items : ", items);
    };

    const remove = (id) => {
      setItems(items.filter((e) => e.id !== id));
    };
```

Front End

- ToDo 삭제
  - ✓ App.js 파일 수정 – 삭제 구현

```
return (
  <div>
    <Container maxWidth="md">
    <AddToDo add={add}/>
    {
      items.length > 0  ?
      (<Paper style={{ margin: 16 }}>
      <List>
        {items.map((item, idx) => (
          <ToDo item={item} remove={remove} />
        ))}
      </List>
      </Paper>)
      :null
    }
    </Container>
  </div>
);
}
export default App;
```

Front End

❖ ToDo 삭제

✓ ToDo.jsx 파일 수정 – 삭제 구현

```
import React, {useState} from "react";

import {
    ListItem,
    ListItemText,
    InputBase,
    Checkbox,
    ListItemSecondaryAction,
    IconButton
} from "@material-ui/core";

import DeleteOutlined from "@material-ui/icons/DeleteOutlined";


const ToDo = (props) => {
    const [item, setItem] = useState(props.item);

    const deleteEventHandler = (id) => {
        props.remove(id);
    };
```

Front End

- ToDo 삭제
  - ✓ ToDo.jsx 파일 수정 – 삭제 구현

```
return (
    <ListItem>
    <Checkbox checked={item.done} />
    <ListItemText>
        <InputBase
            inputProps={{ "aria-label": "naked" }}
            type="text"
            id={item.id}
            name={item.id}
            value={item.title}
            multiline={true}
            fullWidth={true}
        />
    </ListItemText>
```

Front End

- ToDo 삭제
  - ✓ ToDo.jsx 파일 수정 – 삭제 구현

```
            <ListItemSecondaryAction>
              <IconButton aria-label="Delete Todo" onClick={() =>
deleteEventHandler(item.id)}>
                <DeleteOutlined />
              </IconButton>
            </ListItemSecondaryAction>

          </ListItem>
        );
      }
      export default ToDo;
```

Front End

- ToDo 삭제
  - ✓ 웹 브라우저에서 확인 – 삭제 구현

Front End

- ToDo 초기화
  - ✓ App.js 파일 수정 – 기본 데이터 삭제

```
const [items, setItems] = useState([]);
```

Front End

- ToDo 수정
  - ✓ 요구 사항
    - Todo 컴포넌트에 readonly 플래그가 있어 readonly가 true인 경우 아이템 수정이 불가능하고 false인 경우 아이템을 수정할 수 있음
    - 사용자가 어떤 아이템의 title을 클릭하면 해당 input field는 수정할 수 있는 상태인 readonly를 false인 상태로 변경
    - 사용자가 Enter키 또는 Retrun키를 누르면 readonly가 true인 상태로 전환됨
    - 체크박스 클릭 시 item.done 값을 전환

Front End

- ToDo 수정
  - ✓ ToDo.jsx 수정

```
import React, {useState} from "react";

import {
    ListItem,
    ListItemText,
    InputBase,
    Checkbox,
    ListItemSecondaryAction,
    IconButton
} from "@material-ui/core";

import DeleteOutlined from "@material-ui/icons/DeleteOutlined";
```

Front End

- ToDo 수정
  - ✓ ToDo.jsx 수정

```
const ToDo = (props) => {
    const [item, setItem] = useState(props.item);
    const [readOnly, setReadOnly] = useState(false);

    const deleteEventHandler = (id) => {
        props.remove(id);
    };

    const offReadOnlyMode = () => {
        setReadOnly(false)
    };

    const enterKeyEventHandler = (e) => {
        if (e.key === "Enter") {
            setReadOnly(true)
        }
    };
```

Front End

- ToDo 수정
  - ✓ ToDo.jsx 수정

```
const editEventHandler = (e) => {
    setItem({...item, title:e.target.value });
};

const checkboxEventHandler = (e) => {
    setItem({...item, done:!item.done});
};
```

Front End

- ToDo 수정
  - ✓ ToDo.jsx 수정

```
return (
    <ListItem>
    <Checkbox checked={item.done} onChange={checkboxEventHandler} />
    <ListItemText>
        <InputBase
            inputProps={{ "aria-label": "naked", readOnly: readOnly,          }}
            type="text"
            id={item.id}
            name={item.id}
            value={item.title}
            multiline={true}
            fullWidth={true}
            onClick={offReadOnlyMode}
            onChange={editEventHandler}
            onKeyPress={enterKeyEventHandler}
        />
    </ListItemText>
```

Front End

- ToDo 수정
  - ✓ ToDo.jsx 수정

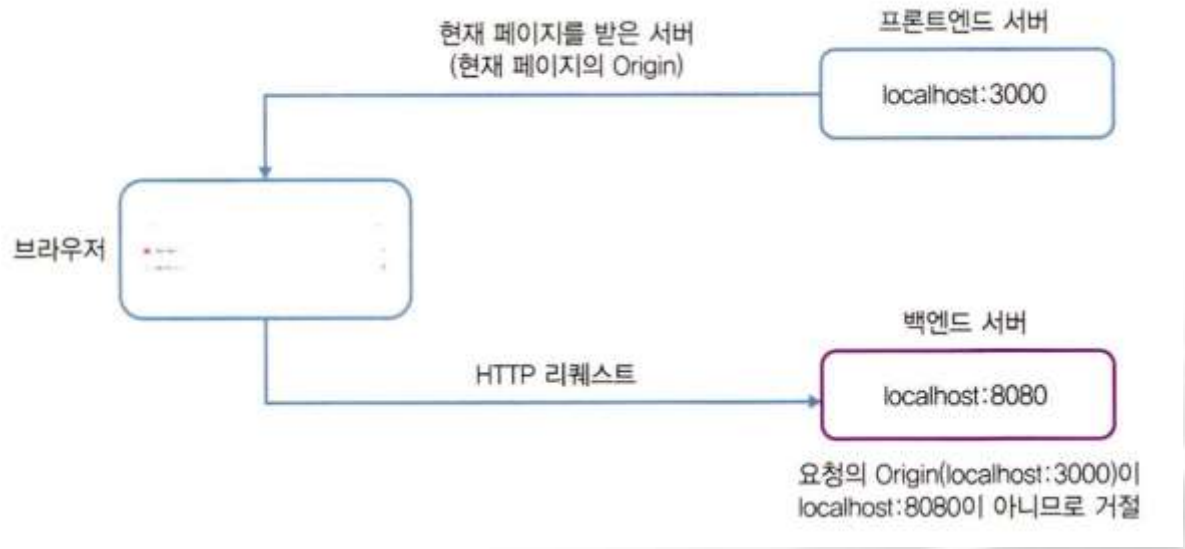
```
        <ListItemSecondaryAction>
          <IconButton aria-label="Delete Todo" onClick={() =>
deleteEventHandler(item.id)}>
            <DeleteOutlined />
          </IconButton>
        </ListItemSecondaryAction>

      </ListItem>
    );
}
export default ToDo;
```

Front End

- ToDo 수정
  - ✓ 웹 브라우저에서 확인

CORS

- CORS
  - ✓ Cross-Origin Resource Sharing 의 약자(https://en.wikipe dia.org/wiki/Cross-origin_resource_sharing)로 처음 리소스를 제공한 도메인(Origin)이 현재 요청하려는 도메인과 다르더라도 요청을 허락해 주는 웹 보안 방침



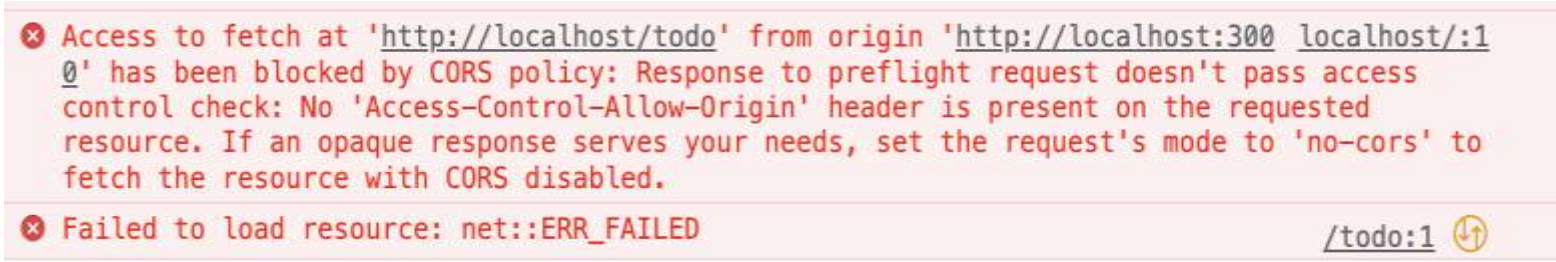
CORS

- CORS
  - ✓ FrontEnd Project 의 App.js 파일의 컴포넌트에 추가

```
React.useEffect(()=>{
  const requestoptions = {
    method: "GET",
    headers: { "Content-Type": "application/json" },
  };

  fetch("http://192.168.0.128/todo", requestoptions)
    .then((response) => response.json())
    .then(
    (response) => {
      setItems(response.data);
  },
    (error) => {
      this.setState({
        error
      });
  }
  );

 }, [])
```

CORS

- CORS
  - ✓ 브라우저의 검사 창에서 확인 – 에러 발생



Spring Boot Application - CORS

- CORS 설정
  - ✓ Controller에 추가
    - Controller의 요청 처리 메서드 위에 @CrossOrigin 를 추가해서 설정 가능
    - Controller 클래스 상단에 @CrossOrigin(origins = "http://example.com", maxAge = 3600) 형태로 설정

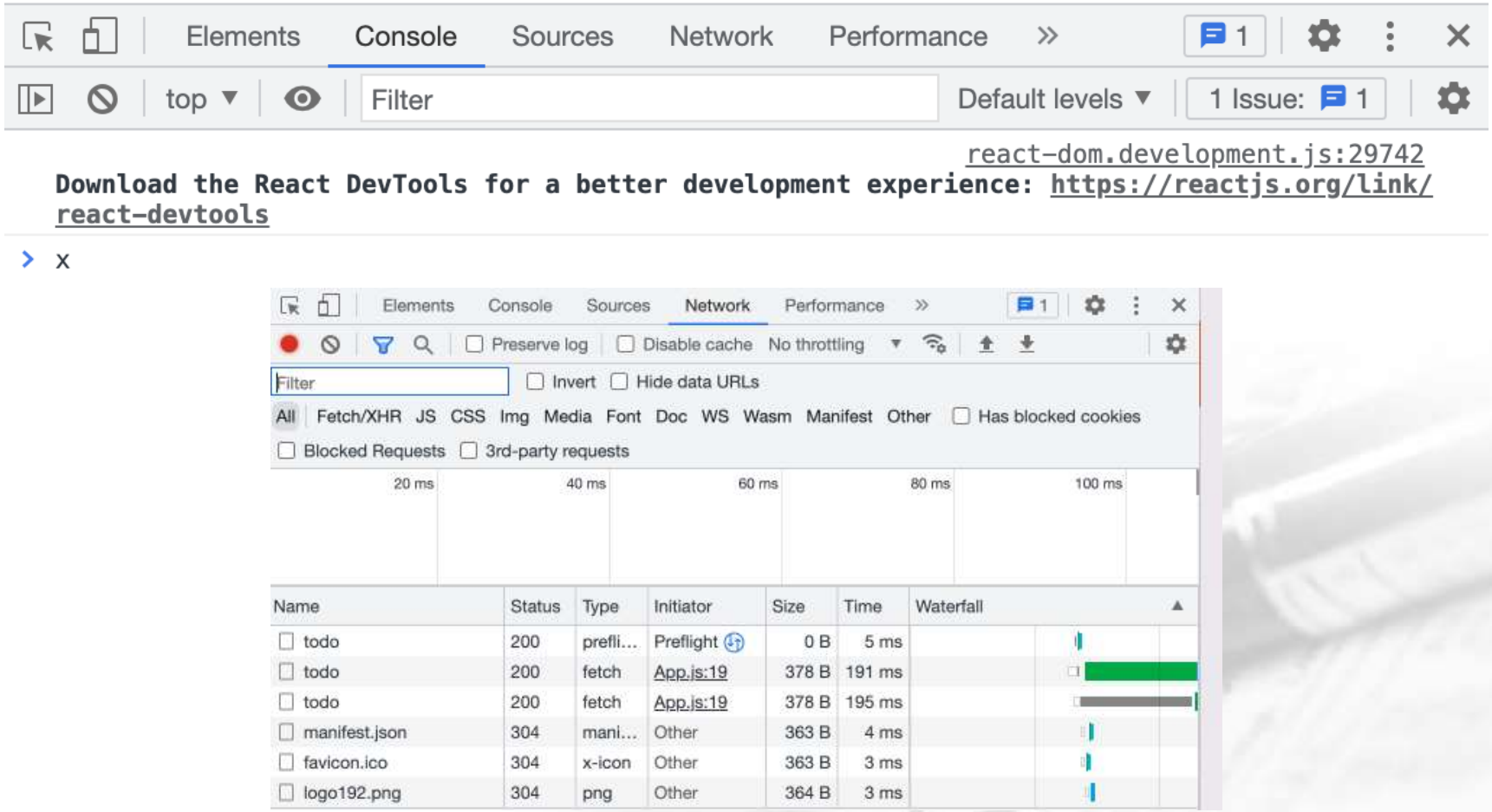
Spring Boot Application - CORS

- CORS 설정
  - ✓ CORS 설정을 위한 환경 설정 클래스를 추가 – config.WebMvcConfig

```
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
@Configuration // 스프링 빈으로 등록
public class WebMvcConfig implements WebMvcConfigurer {
        private final long MAX_AGE_SECS = 3600;

        @Override
        public void addCorsMappings(CorsRegistry registry) {
                // 모든 경로에 대해
                registry.addMapping("/**")
                // Origin이 http:localhost:3000에 대해
                .allowedOrigins("http://192.168.0.128:3000")
                // GET, POST, PUT, PATCH, DELETE, OPTIONS 메서드를 허용한다.
                .allowedMethods("GET", "POST", "PUT", "PATCH", "DELETE",
"OPTIONS")
                .allowedHeaders("*")
                .allowCredentials(true)
                .maxAge(MAX_AGE_SECS);
        }
}
```

Spring Boot Application - CORS

- CORS 설정
  - ✓ 브라우저를 새로 고침을 수행하고 확인



Front End

- Back-End 의 URL을 저장할 환경 설정 파일을 생성 – app-config.js

```
let backendHost;
backendHost = "http://192.168.0.128";
export const API_BASE_URL = `${backendHost}`;
```

Front End

- BackEnd의 데이터를 가져오는 함수를 별도의 파일에 생성 – service/ApiService.js

```
import { API_BASE_URL } from "../app-config";

export function call(api, method, request) {
  let options = {
    headers: new Headers({
      "Content-Type": "application/json",
    }),
    url: API_BASE_URL + api,
    method: method,
  };

  if (request) {
    // GET method
    options.body = JSON.stringify(request);
  }
```

Front End

- BackEnd의 데이터를 가져오는 함수를 별도의 파일에 생성 – service/ApiService.js

```
return fetch(options.url, options).then((response) =>
  response.json().then((json) => {
    if (!response.ok) {
      // response.ok가 true이면 정상적인 리스폰스를 받은것, 아니면 에러 리스폰스를 받은것.
      return Promise.reject(json);
    }
    return json;
  })
);
}
```

Front End

❖ App.js 파일의 내용 수정 – 삽입, 삭제, 가져오기 적용

```
import React, {useState, useEffect} from "react";
import ToDo from "./ToDo";
import AddToDo from "./AddToDo";
import './App.css';
import {Paper, List, Container} from "@material-ui/core"
import { call } from "./service/ApiService";

function App() {
  const [items, setItems] = useState([]);

  useEffect(()=>{
    call("/todo", "GET", null).then((response) =>
    setItems(response.data)
  );
  }, [])
```

Front End

- App.js 파일의 내용 수정 – 삽입, 삭제, 가져오기 적용

```
//데이터 추가를 위한 함수
const add = (item) => {
  console.log(item);
  call("/todo", "POST", item).then((response) =>
    setItems(response.data)
  );
};

const remove = (id) => {
  let item = {"id": id};
  call("/todo", "DELETE", item).then((response) =>
    setItems(response.data)
  );
};
```

Front End

- App.js 파일의 내용 수정 – 삽입, 삭제, 가져오기 적용

```
    return (

      <div>
        <Container maxWidth="md">
        <AddToDo add={add}/>
        {
          items.length > 0  ?
          (<Paper style={{ margin: 16 }}>
          <List>
            {items.map((item, idx) => (
              <ToDo item={item} remove={remove} key={idx} />
            ))}
          </List>
          </Paper>)
          :null
        }
        </Container>
      </div>
    );
  }
  export default App;
```

Front End

❖ 브라우저 와 데이터베이스에서 확인

123 done	ABC title	ABC user_id
0	Java	temporary-user
0	Python	temporary-user

Front End

- App.js 파일의 내용 수정 – 수정 적용

```
import React, {useState, useEffect} from "react";
import ToDo from "./ToDo";
import AddToDo from "./AddToDo";
import './App.css';
import {Paper, List, Container} from "@material-ui/core"
import { call } from "./service/ApiService";

function App() {
  const [items, setItems] = useState([]);

  useEffect(()=>{
    call("/todo", "GET", null).then((response) =>
    setItems(response.data)
  );
  }, [])
```

Front End

- App.js 파일의 내용 수정 – 수정 적용

```
//데이터 추가를 위한 함수
const add = (item) => {
  console.log(item);
  call("/todo", "POST", item).then((response) =>
    setItems(response.data)
  );
};

const remove = (id) => {
  let item = {"id": id};
  call("/todo", "DELETE", item).then((response) =>
    setItems(response.data)
  );
};

const update = (item) => {
  call("/todo", "PUT", item).then((response) =>
  setItems(response.data)
  );
};
```

Front End

❖ App.js 파일의 내용 수정 – 수정 적용

```
    return (

      <div>
        <Container maxWidth="md">
        <AddToDo add={add}/>
        {
          items.length > 0  ?
          (<Paper style={{ margin: 16 }}>
          <List>
            {items.map((item, idx) => (
              <ToDo item={item} remove={remove} key={idx} update={update}/>
            ))}
          </List>
          </Paper>)
          :null
        }
        </Container>
      </div>
    );
  }

  export default App;
```

Front End

❖ ToDo.jsx 파일의 내용 수정 – 수정 적용

```
import React, {useState} from "react";

import {
    ListItem,
    ListItemText,
    InputBase,
    Checkbox,
    ListItemSecondaryAction,
    IconButton
} from "@material-ui/core";

import DeleteOutlined from "@material-ui/icons/DeleteOutlined";
```

Front End

- ToDo.jsx 파일의 내용 수정 – 수정 적용

```
const ToDo = (props) => {
    const [item, setItem] = useState(props.item);
    const [readOnly, setReadOnly] = useState(false);

    const deleteEventHandler = (id) => {
        props.remove(id);
    };

    const offReadOnlyMode = () => {
        setReadOnly(false)
    };
```

Front End

- ToDo.jsx 파일의 내용 수정 – 수정 적용

```
const enterKeyEventHandler = (e) => {
    if (e.key === "Enter") {
        setReadOnly(true)
        props.update(item);
    }
};

 const checkboxEventHandler = (e) => {
    setItem({...item, done:!item.done});
    props.update(item);
};
```

Front End

- ToDo.jsx 파일의 내용 수정 – 수정 적용

```
return (
    <ListItem>
    <Checkbox checked={item.done} onChange={checkboxEventHandler} />
    <ListItemText>
        <InputBase
            inputProps={{ "aria-label": "naked", readOnly: readOnly,          }}
            type="text"
            id={item.id}
            name={item.id}
            value={item.title}
            multiline={true}
            fullWidth={true}
            onClick={offReadOnlyMode}
            onChange={editEventHandler}
            onKeyPress={enterKeyEventHandler}
        />
    </ListItemText>
```

Front End

- ToDo.jsx 파일의 내용 수정 – 수정 적용

```
        <ListItemSecondaryAction>
          <IconButton aria-label="Delete Todo" onClick={() => deleteEventHandler(item.id)}>
            <DeleteOutlined />
          </IconButton>
        </ListItemSecondaryAction>

      </ListItem>
    );
  }
  export default ToDo;
```