

To Do Backend

Postman

- ❖ REST API 테스트를 위한 애플리케이션 설치

- ✓ <https://www.postman.com/downloads/>에서 다운로드 받아서 설치

The screenshot shows two main parts: the Postman download page and the Postman application interface.

Postman Download Page:

- The URL in the browser is [postman.com/downloads/](https://www.postman.com/downloads/).
- The page title is "Download Postman".
- Text: "Download the app to quickly get started using the Postman API Platform. Or, if you prefer a browser experience, you can try the new web version of Postman."
- The Postman app:**
 - The ever-improving Postman app (a new release every week) gives you a full-featured Postman experience.
 - Buttons for "Mac Intel Chip" and "Mac Apple Chip".
 - Text: "By downloading and using Postman, I agree to the [Privacy Policy](#) and [Terms](#)".
 - Text: "Version 9.18.3 · [Release Notes](#) · [Product Roadmap](#)".
 - Text: "Not your OS? Download for Windows (x64) or Linux (x64)".
- Postman on the web:**
 - You can now access Postman through your web browser.

Postman Application Interface:

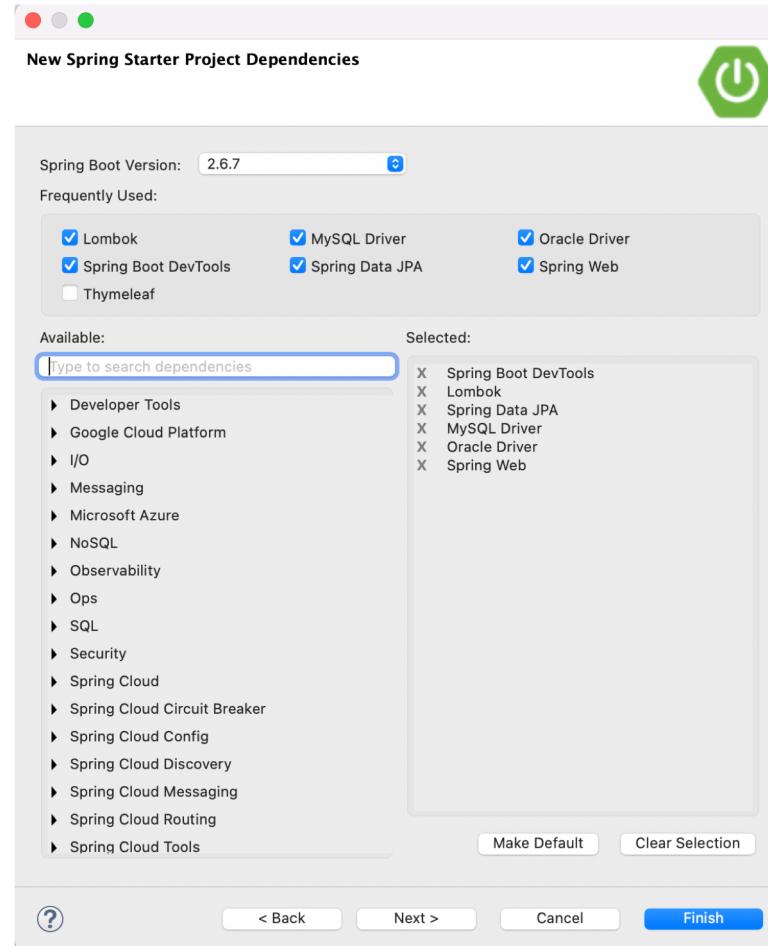
- The interface shows a "Twitter's Public Workspace" with a "Collections" sidebar containing "Twitter API v2", "Tweet Lookup", and "Single Tweet".
- The main panel displays a "Twitter API v2 / Tweet Lookup / Single Tweet" request with a "GET" method and URL <https://api.twitter.com/2/tweets/:id>.
- Request parameters (Query params):

Key	Value
tweet.fields	attachments,author_id,context_annotations,conversation_id,duration_ms,height,media_key,non_public_metrics,organic_results,public_metrics,source,text,timestamp_ms,withheld_in_countries
expansions	
media.fields	
- Buttons: "Save", "Send", "Accept All Cookies", and "Cookies Settings".

ToDo

❖ 프로젝트 생성

- ✓ 의존성: devtools, web, jpa, lombok, mysql, oracle



ToDo

- ❖ application.properties 파일에 기본 설정 추가

```
server.port=80
```

```
#MySQL
```

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/cyberadam?useUnicode=yes&characterEnc  
oding=UTF-8&serverTimezone=UTC  
spring.datasource.username=user00  
spring.datasource.password=user00
```

```
#Oracle
```

```
#spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver  
#spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe  
#spring.datasource.username=system  
#spring.datasource.password=oracle
```

ToDo

- ❖ application.properties 파일에 기본 설정 추가

```
spring.jpa.properties.hibernate.show_sql=true  
spring.jpa.properties.hibernate.format_sql=true  
logging.level.org.hibernate.type.descriptor.sql=trace  
spring.jpa.hibernate.ddl-auto=update
```

```
#Live Reload  
spring.devtools.livereload.enabled=true
```

ToDo

- ❖ 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);
```

```
}
```

```
}
```

ToDo

- ❖ Entity 작업
 - ✓ entity 패키지에 공통된 속성을 소유하는 BaseEntity 생성

```
import java.time.LocalDateTime;
import javax.persistence.Column;
import javax.persistence.EntityListeners;
import javax.persistence.MappedSuperclass;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;
import lombok.Getter;

@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;
}
```

ToDo

- ❖ Entity 작업
 - ✓ entity 패키지에 ToDo 데이터를 위한 Entity Class 생성

```
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
import org.hibernate.annotations.GenericGenerator;  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.EqualsAndHashCode;  
import lombok.NoArgsConstructor;
```

ToDo

- ❖ Entity 작업
 - ✓ entity 패키지에 ToDo 데이터를 위한 Entity Class 생성

```
@Builder  
@Data  
@EqualsAndHashCode(callSuper=false)  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "todo")  
public class ToDoEntity extends BaseEntity{  
    @Id  
    @GeneratedValue(generator="system-uuid")  
    @GenericGenerator(name="system-uuid", strategy="uuid")  
    private Long id;  
  
    @Column(length = 100, nullable = false)  
    private String userId;  
  
    @Column(length = 500, nullable = false)  
    private String title;  
  
    @Column(nullable = false)  
    private boolean done;  
}
```

ToDo

- ❖ Entity 작업
 - ✓ 프로젝트를 실행하고 데이터베이스에 테이블이 생성되는지 확인

ToDo

- ❖ Repository 작업
 - ✓ persistency 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);  
}
```

ToDo

- ❖ Repository 작업
 - ✓ persistency 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);  
}
```

ToDo

❖ 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;
```

ToDo

❖ 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);

}

}

ToDo

- ❖ 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");  
    }  
}
```

ToDo

❖ 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("수정한 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");  
    }  
}
```

ToDo

❖ 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");  
    }  
}
```

ToDo

- ❖ 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 title;
    private boolean done;

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

ToDo

- ❖ DTO 작업
 - ✓ dto 패키지에 ToDoDTO 클래스를 생성하고 작성

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

ToDo

- ❖ DTO 작업
 - ✓ dto 패키지에 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;
}
```

ToDo

- ❖ Service 작업

- ✓ service 패키지에 ToDoService 인터페이스를 생성하고 작성

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

```
public interface ToDoService {  
    public List<ToDoEntity> create(final ToDoEntity entity);  
  
    public List<ToDoEntity> retrieve(final String userId);  
  
    public List<ToDoEntity> update(final ToDoEntity entity);  
  
    public List<ToDoEntity> delete(final ToDoEntity entity);  
}
```

ToDo

❖ Service 작업

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

```
import java.util.List;  
import java.util.Optional;  
  
import org.springframework.stereotype.Service;  
  
import com.adamsoft.entity.ToDoEntity;  
import com.adamsoft.persistence.ToDoRepository;
```

```
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;
```

```
@Slf4j  
@Service  
@RequiredArgsConstructor  
public class ToDoServiceImpl implements ToDoService {  
  
    private final ToDoRepository repository;
```

ToDo

- ❖ Service 작업

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

```
private void validate(final ToDoEntity entity) {  
    if(entity == null) {  
        log.warn("Entity cannot be null.");  
        throw new RuntimeException("Entity cannot be null.");  
    }  
  
    if(entity.getUserId() == null) {  
        log.warn("Unknown user.");  
        throw new RuntimeException("Unknown user.");  
    }  
}
```

ToDo

- ❖ Service 작업

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

```
public List<ToDoEntity> create(final ToDoEntity entity) {  
    validate(entity);  
    repository.save(entity);  
    log.info("Entity Id : {} is saved.", entity.getId());  
    return repository.findByUserId(entity.getUserId());  
}
```

ToDo

- ❖ Service 작업

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

```
public List<ToDoEntity> retrieve(final String userId) {  
    return repository.findByUserId(userId);  
}
```

ToDo

❖ Service 작업

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

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

ToDo

❖ Service 작업

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

```
public List<ToDoEntity> delete(final ToDoEntity entity) {  
    validate(entity);  
  
    try {  
        repository.delete(entity);  
    } catch(Exception e) {  
        log.error("error deleting entity ", entity.getId(), e);  
        throw new RuntimeException("error deleting entity "  
+ entity.getId());  
    }  
    return retrieve(entity.getUserId());  
}  
}
```

ToDo

- ❖ Controller 작업

- ✓ controller 패키지에 TestRestController 클래스를 만들고 테스트를 위한 코드를 작성

```
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

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

@RestController
@RequestMapping("test")
public class TestRestController {

    @GetMapping
    public String TestRestController() {
        return "Hello World!";
    }
}
```

ToDo

❖ Controller 작업

- ✓ controller 패키지에 TestRestController 클래스를 만들고 테스트를 위한 코드를 작성

```
@GetMapping("/testGetMapping")
public String TestRestControllerWithPath() {
    return "Hello World! testGetMapping ";
}
```

```
@GetMapping("/{id}")
public String TestRestControllerWithPathVariables(@PathVariable(required =
false) int id) {
    return "Hello World! ID " + id;
}
```

// /test경로는 이미 존재하므로 /test/testRequestParam으로 지정했다.

```
@GetMapping("/testRequestParam")
public String TestRestControllerRequestParam(@RequestParam(required =
false) int id) {
    return "Hello World! ID " + id;
}
```

ToDo

❖ Controller 작업

- ✓ controller 패키지에 TestRestController 클래스를 만들고 테스트를 위한 코드를 작성

// /test경로는 이미 존재하므로 /test/testRequestBody로 지정했다.

```
@GetMapping("/testRequestBody")
public String TestRestControllerRequestBody(@RequestBody
TestRequestBodyDTO testRequestBodyDTO) {
    return "Hello World! ID " + testRequestBodyDTO.getId() + "
Message : " + testRequestBodyDTO.getMessage();
}
```

@GetMapping("/testResponseBody")

```
public ResponseDTO<String> TestRestControllerResponseBody() {
    List<String> list = new ArrayList<>();
    list.add("Hello World! I'm ResponseDTO");
    ResponseDTO<String> response =
ResponseDTO.<String>builder().data(list).build();
    return response;
}
```

ToDo

- ❖ Controller 작업
 - ✓ controller 패키지에 TestRestController 클래스를 만들고 테스트를 위한 코드를 작성

```
@GetMapping("/testResponseEntity")
public ResponseEntity<?> TestRestControllerResponseEntity() {
    List<String> list = new ArrayList<>();
    list.add("Hello World! I'm ResponseEntity. And you got 400!");
    // http status를 400로 설정.
    ResponseDTO<String> response =
    ResponseDTO.<String>builder().data(list).build();
    return ResponseEntity.badRequest().body(response);
}
```

ToDo

❖ Controller 작업

- ✓ controller 패키지에 ToDoController 클래스를 만들고 요청을 처리할 코드를 작성

```
import kr.co.adamsoft.dto.ResponseDTO;
import kr.co.adamsoft.dto.ToDoDTO;
import kr.co.adamsoft.entity.ToDoEntity;
import kr.co.adamsoft.service.ToDoService;

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;
import java.util.stream.Collectors;
```

ToDo

❖ Controller 작업

- ✓ controller 패키지에 ToDoController 클래스를 만들고 요청을 처리할 코드를 작성

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

    @Autowired
    private ToDoService service;

    @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);
    }
}
```

ToDo

❖ 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.  
  
        ToDoEntity entity = ToDoDTO.toEntity(dto);  
        entity.setId(null);  
        entity.setUserId(temporaryUserId);  
        List<ToDoEntity> entities = service.create(entity);  
        List<ToDoDTO> dtos =  
        entities.stream().map(ToDoDTO::new).collect(Collectors.toList());  
        ResponseDTO<ToDoDTO> response =  
        ResponseDTO.<ToDoDTO>builder().data(dtos).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);  
    }  
}
```

ToDo

- ❖ Controller 작업

- ✓ controller 패키지에 ToDoController 클래스를 만들고 요청을 처리할 코드를 작성

```
@GetMapping  
public ResponseEntity<?> retrieveTodoList() {  
    String temporaryUserId = "temporary-user"; // temporary user id.  
  
    List<ToDoEntity> entities = service.retrieve(temporaryUserId);  
  
    List<ToDoDTO> dtos =  
entities.stream().map(ToDoDTO::new).collect(Collectors.toList());  
  
    ResponseDTO<ToDoDTO> response =  
ResponseDTO.<ToDoDTO>builder().data(dtos).build();  
  
    return ResponseEntity.ok().body(response);  
}
```

ToDo

- ❖ 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.  
  
    ToDoEntity entity = ToDoDTO.toEntity(dto);  
  
    entity.setUserId(temporaryUserId);  
  
    List<ToDoEntity> entities = service.update(entity);  
  
    List<ToDoDTO> dtos =  
entities.stream().map(ToDoDTO::new).collect(Collectors.toList());  
  
    ResponseDTO<ToDoDTO> response =  
ResponseDTO.<ToDoDTO>builder().data(dtos).build();  
  
    return ResponseEntity.ok().body(response);  
}
```

ToDo

- ❖ 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.  
        ToDoEntity entity = ToDoDTO.toEntity(dto);  
        entity.setUserId(temporaryUserId);  
        List<ToDoEntity> entities = service.delete(entity);  
        List<ToDoDTO> dtos =  
        entities.stream().map(ToDoDTO::new).collect(Collectors.toList());  
        ResponseDTO<ToDoDTO> response =  
        ResponseDTO.<ToDoDTO>builder().data(dtos).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);  
    }  
}
```

ToDo

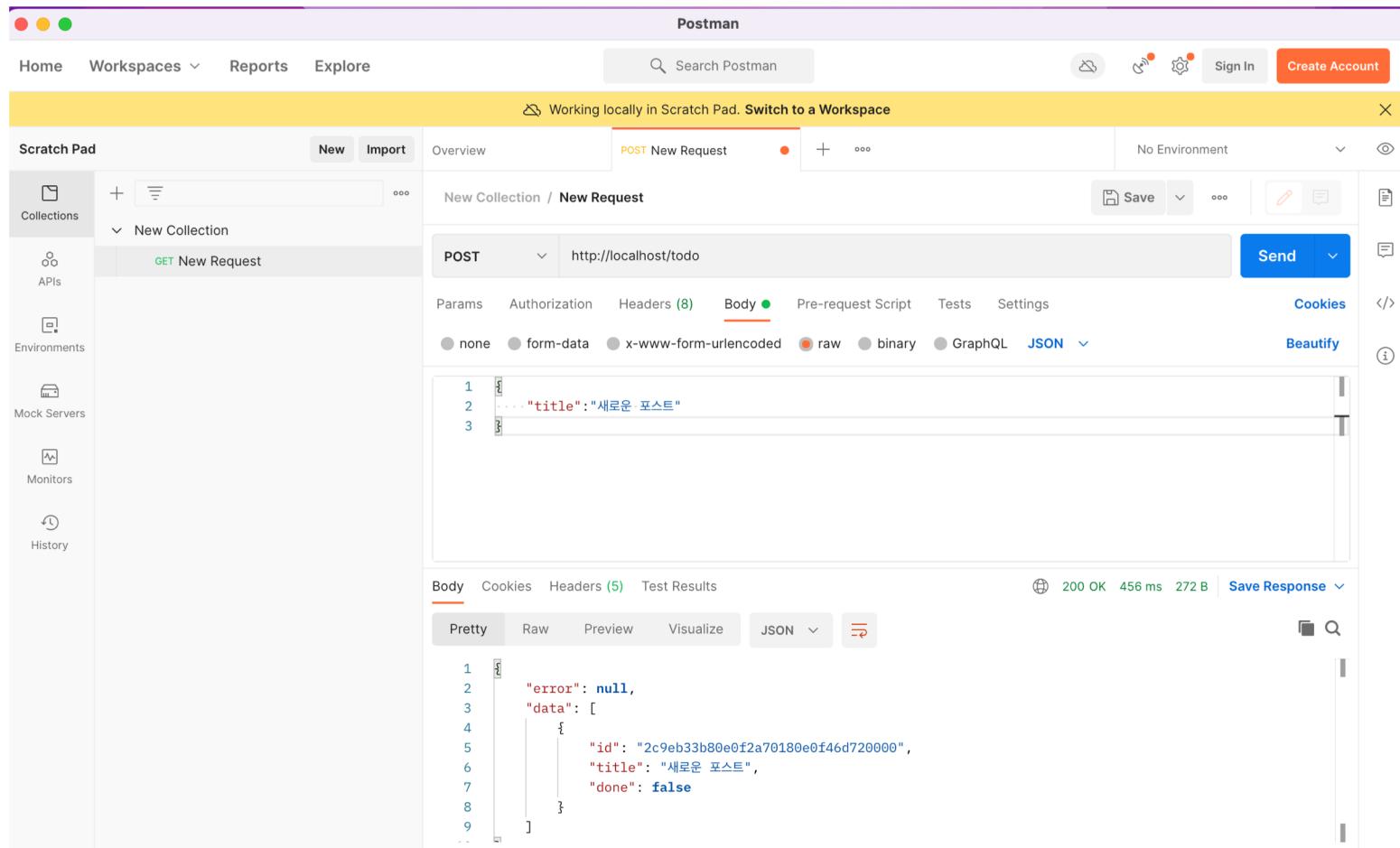
- ❖ Controller 작업

- ✓ controller 패키지에 ToDoController 클래스를 만들고 요청을 처리할 코드를 작성

```
@DeleteMapping  
public ResponseEntity<?> deleteTodo(@RequestBody ToDoDTO dto) {  
    try {  
        String temporaryUserId = "temporary-user"; //  
        temporary user id.  
        ToDoEntity entity = ToDoDTO.toEntity(dto);  
        entity.setUserId(temporaryUserId);  
        List<ToDoEntity> entities = service.delete(entity);  
        List<ToDoDTO> dtos =  
        entities.stream().map(ToDoDTO::new).collect(Collectors.toList());  
        ResponseDTO<ToDoDTO> response =  
        ResponseDTO.<ToDoDTO>builder().data(dtos).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);  
    }  
}
```

ToDo

❖ 테스트 - PostMan



ToDo

❖ 테스트 - PostMan

The screenshot shows the Postman application interface. At the top, there is a header bar with a dropdown menu set to "GET", a URL input field containing "http://localhost/todo", and a blue "Send" button. Below the header, there are tabs for "Params", "Authorization", "Headers (6)", "Body", "Pre-request Script", "Tests", and "Settings". The "Body" tab is currently selected, indicated by an orange underline. Under the "Body" tab, there are several radio buttons for different data types: "none", "form-data", "x-www-form-urlencoded", "raw" (which is selected and highlighted in orange), "binary", and "GraphQL". To the right of the body type buttons are "Cookies" and "Beautify" buttons. The main content area shows a large text input field with the number "1" at the top left. Below this, there is a large empty white space. At the bottom of the interface, there is a navigation bar with tabs for "Body", "Cookies", "Headers (5)", and "Test Results", with "Body" being the active tab. To the right of the navigation bar are status indicators: a globe icon, "200 OK", "152 ms", "272 B", and a "Save Response" button. At the very bottom, there are buttons for "Pretty", "Raw", "Preview", "Visualize", and "JSON" (which is selected and highlighted in orange). To the right of these buttons are icons for copy and search.

```
1 {
  "error": null,
  "data": [
    {
      "id": "2c9eb33b80e0f2a70180e0f46d720000",
      "title": "새로운 포스트",
      "done": false
    }
  ]
}
```

ToDo

❖ 테스트 - PostMan

The screenshot shows the Postman application interface for testing a REST API. The request method is set to **PUT**, and the URL is **http://localhost/todo**. The **Body** tab is selected, showing a JSON payload:

```
1 {  
2     "id": "2c9eb33b80e0f2a70180e0f46d720000",  
3     "title": "수정한 포스트",  
4     "done": false  
5 }
```

The response status is **200 OK** with **137 ms** latency and **272 B** size. The response body is:

```
1 {  
2     "error": null,  
3     "data": [  
4         {  
5             "id": "2c9eb33b80e0f2a70180e0f46d720000",  
6             "title": "수정한 포스트",  
7             "done": false  
8         }  
9     ]  
--
```

ToDo

❖ 테스트 - PostMan

The screenshot shows the Postman application interface for testing a REST API. The request URL is `http://localhost/todo`. The method is set to `DELETE`. The `Body` tab is selected, showing the following JSON payload:

```
1 {  
2   ... | ... "id": "2c9eb33b80e0f2a70180e0f46d720000"  
3 }
```

The response tab shows the following details:

- Body tab is selected.
- Response status: `200 OK`
- Response time: `46 ms`
- Response size: `188 B`
- Save Response button is available.

Below the response, the JSON response body is displayed:

```
1 {  
2   "error": null,  
3   "data": []  
4 }
```

To Do Frontend

React.js

- ❖ SPA

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

- ❖ 개발 환경

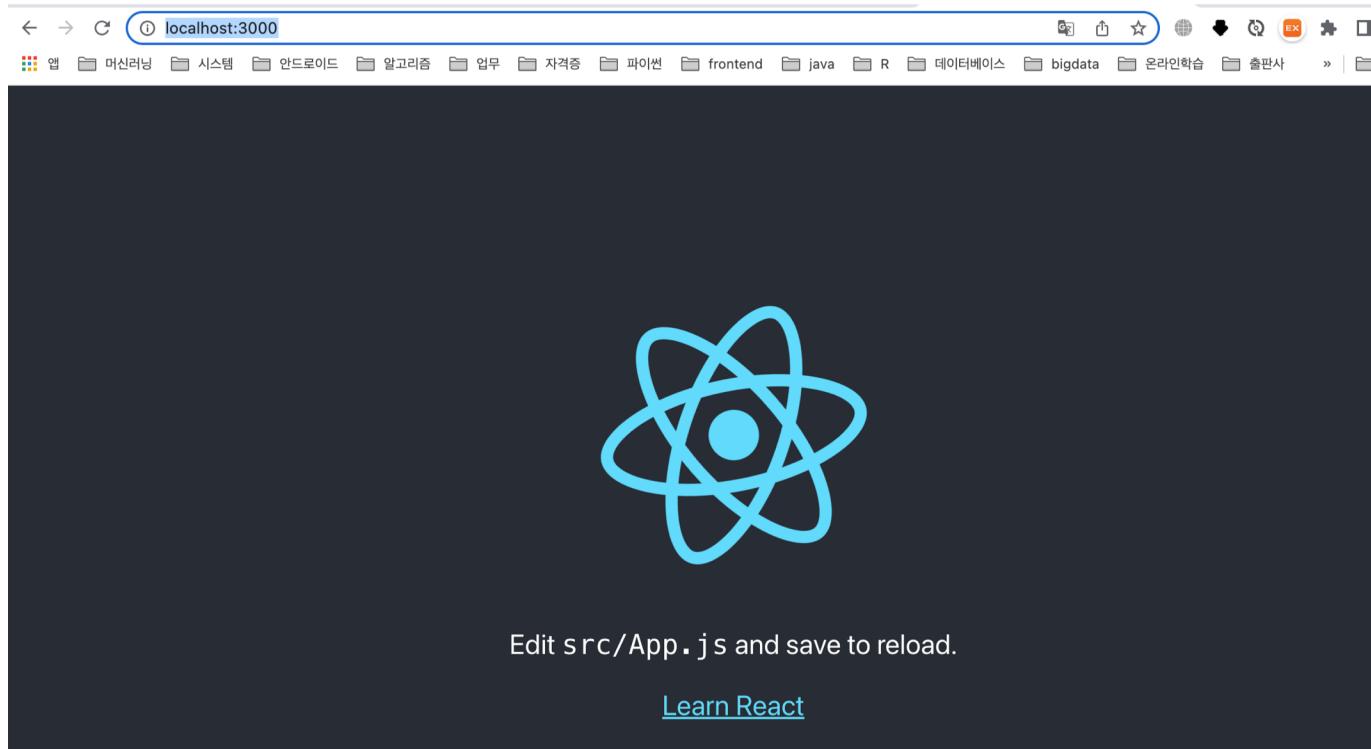
- ✓ node
- ✓ IDE - VS Code

React.ja

- ❖ 프로젝트 생성 및 실행
 - ✓ react 프로젝트 생성 – todo-react-app 이 프로젝트 이름
npx create-react-app todo-react-app
 - ✓ 애플리케이션 실행
npm start

React

- ❖ 프로젝트 생성 및 실행
- ✓ 브라우저 확인



React

- ❖ 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
```

ToDo

- ❖ ToDo 목록 보기

- ✓ src 디렉토리에 ToDo.js 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;
```

ToDo

- ❖ 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;
```

ToDo

- ❖ ToDo 목록 보기
 - ✓ 브라우저 확인

ToDo 컴포넌트 만들기

ToDo

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

```
import React from "react";

class ToDo extends React.Component {
  constructor(props) {
    //props 오브젝트 초기화
    super(props);
    this.state = {item:props.item};
  }
}
```

ToDo

- ❖ ToDo 목록 보기

- ✓ ToDo.js 파일 수정 – Props 와 State

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

export default ToDo;
```

ToDo

❖ ToDo 목록 보기

- ✓ App.js 파일 수정 – Props 와 State

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

class App extends React.Component{
    constructor(props) {
        super(props);
        this.state = {item: { id: 0, title: "Hello React", done: true}};
    }

    render() {
        return (
            <div className="App">
                <ToDo item={this.state.item} />
            </div>
        );
    }
}

export default App;
```

ToDo

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

Hello React

ToDo

❖ ToDo 목록 보기

✓ App.js 파일 수정 – ToDo 리스트

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

class App extends React.Component{
  constructor(props) {
    super(props);
    this.state = {items: [{ id: 0, title: "Java", done: true}, { id: 1, title: "Python", done: true}]};
  }

  render() {
    var todoltems = this.state.items.map((item, idx) => (
      <ToDo item={item} key={item.id} />
    ));
  }
}
```

ToDo

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

```
return (  
  <div className="App">  
    {todoltems}  
  </div>  
);  
}  
  
export default App;
```

ToDo

- ❖ ToDo 목록 보기
 - ✓ 브라우저 확인 – ToDo 리스트

-
- Java
 - Python

ToDo

❖ ToDo 목록 보기

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

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

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

```
class ToDo extends React.Component {  
  constructor(props) {  
    //props 오브젝트 초기화  
    super(props);  
    this.state = { item: props.item };  
  }
```

ToDo

- ❖ ToDo 목록 보기

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

```
render() {
    const item = this.state.item;
    return (
        <ListItemIcon>
            <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>
        </ListItemIcon>
    );
}

export default ToDo;
```

ToDo

❖ ToDo 목록 보기

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

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

```
class App extends React.Component{
  constructor(props) {
    super(props);
    this.state = {items: [{ id: 0, title: "Java", done: true}, { id: 1, title: "Python", done: true}]}
  }
}
```

ToDo

❖ ToDo 목록 보기

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

```
render() {
  var todoltems = this.state.items.length > 0 && (
    <Paper style={{ margin: 16 }}>
      <List>
        {this.state.items.map((item, idx) => (
          <ToDo item={item} />
        ))}
      </List>
    </Paper>
  );
  return (
    <div className="App">
      {todoltems}
    </div>
  );
}
}

export default App;
```

ToDo

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

Java

Python

ToDo

❖ ToDo 추가

- ✓ AddToDo.js 파일을 추가하고 작성

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

class AddTodo extends React.Component {
  constructor(props) {
    super(props);
    this.state = { item: { title: "" } };

  }
}
```

ToDo

- ❖ ToDo 추가
 - ✓ AddToDo.js 파일을 추가하고 작성

```
render() {  
  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>  
    </Grid>
```

ToDo

❖ ToDo 추가

- ✓ AddToDo.js 파일을 추가하고 작성

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

export default AddTodo;
```

ToDo

❖ ToDo 추가

✓ App.js 파일 수정

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

class App extends React.Component{
  constructor(props) {
    super(props);
    this.state = {items: [{ id: 0, title: "Java", done: true}, { id: 1, title: "Python", done: true}]}
  }
}
```

ToDo

❖ ToDo 추가

✓ App.js 파일 수정

```
render() {  
  var todoltems = this.state.items.length > 0 && (  
    <Paper style={{ margin: 16 }}>  
      <List>  
        {this.state.items.map((item, idx) => (  
          <ToDo item={item} key={item.id} delete={this.delete} />  
        ))}  
      </List>  
    </Paper>  
);
```

ToDo

- ❖ ToDo 추가

- ✓ App.js 파일 수정

```
return (
  <div className="App">
    <Container maxWidth="md">
      <AddToDo/>
      <div className="ToDoList">{todoItems}</div>
    </Container>
  </div>
);
}

export default App;
```

ToDo

- ❖ ToDo 추가
 - ✓ 브라우저 확인

Add Todo here

+

Java

Python

ToDo

❖ ToDo 추가

- ✓ Addjs 파일 수정 – 이벤트 처리

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

class App extends React.Component{
    constructor(props) {
        super(props);
        this.state = {items: [{ id: 0, title: "Java", done: true}, { id: 1, title: "Python", done: true}]};
    }

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

ToDo

❖ ToDo 추가

- ✓ Addjs 파일 수정 – 이벤트 처리

```
render() {
  var todolItems = this.state.items.length > 0 && (
    <Paper style={{ margin: 16 }}>
      <List>
        {this.state.items.map((item, idx) => (
          <ToDo item={item} key={item.id} delete={this.delete} />
        ))}
      </List>
    </Paper>
  );
}
```

ToDo

❖ ToDo 추가

- ✓ Addjs 파일 수정 – 이벤트 처리

```
return (
  <div className="App">
    <Container maxWidth="md">
      <AddToDo add={this.add}/>
      <div className="ToDoList">{todoItems}</div>
    </Container>
  </div>
);
}
}

export default App;
```

ToDo

- ❖ ToDo 추가
 - ✓ AddToDo.js 파일 수정 – 이벤트 처리

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

class AddTodo extends React.Component {
  constructor(props) {
    super(props);
    this.state = { item: { title: "" } };
    this.add = props.add;
  }
}
```

ToDo

❖ ToDo 추가

✓ AddToDo.js 파일 수정 – 이벤트 처리

```
//Input 의 내용이 변경될 때 호출  
onInputChange = (e) => {  
    const thisItem = this.state.item;  
    thisItem.title = e.target.value;  
    this.setState({ item: thisItem });  
    console.log(thisItem);  
};  
  
//+버튼을 눌렀을 때 호출  
onButtonClick = () => {  
    this.add(this.state.item);  
    this.setState({ item: { title: "" } });  
};  
  
//Enter를 눌렀을 때 처리  
enterKeyEventHandler = (e) => {  
    if (e.key === "Enter") {  
        this.onButtonClick();  
    }  
};
```

ToDo

❖ ToDo 추가

- ✓ AddToDo.js 파일 수정 – 이벤트 처리

```
render() {
  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={this.onInputChange}
            value={this.state.item.title}
            onKeyPress={this.enterKeyEventHandler}>
          />
        </Grid>
```

ToDo

❖ ToDo 추가

- ✓ AddToDo.js 파일 수정 – 이벤트 처리

```
<Grid xs={1} md={1} item>
  <Button
    fullWidth
    color="secondary"
    variant="outlined"
    onClick={this.onButtonClick}
    >
    +
    </Button>
  </Grid>
</Grid>
</Paper>
);
}
}

export default AddTodo;
```

ToDo

❖ ToDo 추가

- ✓ 브라우저 확인 – 이벤트 처리

C#

+

Java

Python

C++

Add Todo here

+

Java

Python

C++

C#

ToDo

❖ ToDo 삭제

- ✓ ToDo.js 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";
```

ToDo

❖ ToDo 삭제

- ✓ ToDo.js 파일 수정 – 삭제 아이콘 출력

```
class ToDo extends React.Component {  
  constructor(props) {  
    //props 오브젝트 초기화  
    super(props);  
    this.state = { item: props.item };  
  }  
}
```

ToDo

❖ ToDo 삭제

- ✓ ToDo.js 파일 수정 – 삭제 아이콘 출력

```
render() {
  const item = this.state.item;
  return (
    <ListItemIcon>
      <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>
    )
```

ToDo

- ❖ ToDo 삭제

- ✓ ToDo.js 파일 수정 – 삭제 아이콘 출력

```
<ListItemIconSecondaryAction>
  <IconButton aria-label="Delete Todo">
    <DeleteOutlined />
  </IconButton>
</ListItemIconSecondaryAction>
</ListItemIcon>
);
}
}

export default ToDo;
```

ToDo

- ❖ ToDo 삭제
 - ✓ 브라우저에서 확인 – 삭제 아이콘 출력

Add Todo here

Java ✖

Python ✖

C++ ✖

C# ✖

ToDo

❖ ToDo 삭제

✓ App.js 파일 수정 – 삭제 구현

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

class App extends React.Component{
    constructor(props) {
        super(props);
        this.state = {items: [{ id: 0, title: "Java", done: true}, { id: 1, title: "Python", done: true}]};
    }

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

ToDo

- ❖ ToDo 삭제

- ✓ App.js 파일 수정 – 삭제 구현

```
delete = (item) => {
  const thisItems = this.state.items;
  console.log("Before Update Items : ", this.state.items);
  const newItems = thisItems.filter((e) => e.id !== item.id); // 해당 id 걸러내기
  this.setState({ items: newItems }, () => {
    // 디버깅 콜백
    console.log("Update Items : ", this.state.items);
  });
};
```

ToDo

❖ ToDo 삭제

- ✓ App.js 파일 수정 – 삭제 구현

```
render() {
  var todolItems = this.state.items.length > 0 && (
    <Paper style={{ margin: 16 }}>
      <List>
        {this.state.items.map((item, idx) => (
          <ToDo item={item} key={item.id} delete={this.delete}/>
        ))}
      </List>
    </Paper>
  );
}
```

ToDo

❖ ToDo 삭제

✓ App.js 파일 수정 – 삭제 구현

```
return (
  <div className="App">
    <Container maxWidth="md">
      <AddToDo add={this.add}/>
      <div className="ToDoList">{todoItems}</div>
    </Container>
  </div>
);
}
}

export default App;
```

ToDo

❖ ToDo 삭제

- ✓ ToDo.js 파일 수정 – 삭제 구현

```
import React from "react";

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

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

ToDo

❖ ToDo 삭제

- ✓ ToDo.js 파일 수정 – 삭제 구현

```
class ToDo extends React.Component {  
  constructor(props) {  
    //props 오브젝트 초기화  
    super(props);  
    this.state = { item: props.item };  
    this.delete = props.delete;  
  }  
  
  deleteEventHandler = () => {  
    this.delete(this.state.item);  
  };
```

ToDo

❖ ToDo 삭제

- ✓ ToDo.js 파일 수정 – 삭제 구현

```
render() {
  const item = this.state.item;
  return (
    <ListIem>
      <Checkbox checked={item.done} />
      <ListIemText>
        <InputBase
          inputProps={{ "aria-label": "naked" }}
          type="text"
          id={item.id}
          name={item.id}
          value={item.title}
          multiline={true}
          fullWidth={true}
        />
      </ListIemText>
```

ToDo

- ❖ ToDo 삭제
 - ✓ ToDo.js 파일 수정 – 삭제 구현

```
<ListItemIconSecondaryAction>
    <IconButton aria-label="Delete Todo"
    onClick={this.deleteEventHandler}>
        <DeleteOutlined />
    </IconButton>
</ListItemIconSecondaryAction>
</ListItemIcon>
);
}
}

export default ToDo;
```

ToDo

- ❖ ToDo 삭제
 - ✓ 웹 브라우저에서 확인 – 삭제 구현

Add Todo here

+

Python

C#

刪除

Add Todo here

+

Python

刪除

ToDo

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

```
class App extends React.Component{  
  constructor(props) {  
    super(props);  
    this.state = {items: []};  
  }  
}
```

ToDo

❖ 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

ToDo

- ❖ ToDo 수정

- ✓ ToDo.js 수정

```
import React from "react";

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

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

class ToDo extends React.Component {
  constructor(props) {
    super(props);
    this.state = { item: props.item, readOnly: true };
    this.delete = props.delete;
  }
}
```

ToDo

- ❖ ToDo 수정

- ✓ ToDo.js 수정

```
deleteEventHandler = () => {
    this.delete(this.state.item);
};

offReadOnlyMode = () => {
    console.log("Event!", this.state.readOnly);
    this.setState({ readOnly: false }, () => {
        console.log("ReadOnly? ", this.state.readOnly);
    });
};

enterKeyEventHandler = (e) => {
    if (e.key === "Enter") {
        this.setState({ readOnly: true });
    }
};
```

ToDo

- ❖ ToDo 수정
 - ✓ ToDo.js 수정

```
editEventHandler = (e) => {
  const thisItem = this.state.item;
  thisItem.title = e.target.value;
  this.setState({ item: thisItem });
};
```

```
checkboxEventHandler = (e) => {
  const thisItem = this.state.item;
  thisItem.done = !thisItem.done;
  this.setState({ item: thisItem });
};
```

ToDo

- ❖ ToDo 수정

- ✓ ToDo.js 수정

```
render() {  
    const item = this.state.item;  
    return (  
        <ListItem>  
            <Checkbox checked={item.done} onChange={this.checkboxEventHandler}>  
        />  
    );  
}
```

ToDo

- ❖ ToDo 수정
- ✓ ToDo.js 수정

```
<ListItemText>
  <InputBase
    inputProps={{
      "aria-label": "naked",
      readOnly: this.state.readOnly,
    }}
    type="text"
    id={item.id}
    name={item.id}
    value={item.title}
    fullWidth={true}
    onClick={this.offReadOnlyMode}
    onChange={this.editEventHandler}
    onKeyPress={this.enterKeyEventHandler}
  />
</ListItemText>
```

ToDo

- ❖ ToDo 수정
- ✓ ToDo.js 수정

```
<ListItemIconSecondaryAction>
  <IconButton
    aria-label="Delete Todo"
    onClick={this.deleteEventHandler}
  >
    <DeleteOutlined />
  </IconButton>
</ListItemIconSecondaryAction>
</ListItemIcon>
);
}
}

export default ToDo;
```

ToDo

- ❖ ToDo 수정
 - ✓ 웹 브라우저에서 확인

Add Todo here

+

Java 는 플랫폼으로서의 역할로 변신



Python

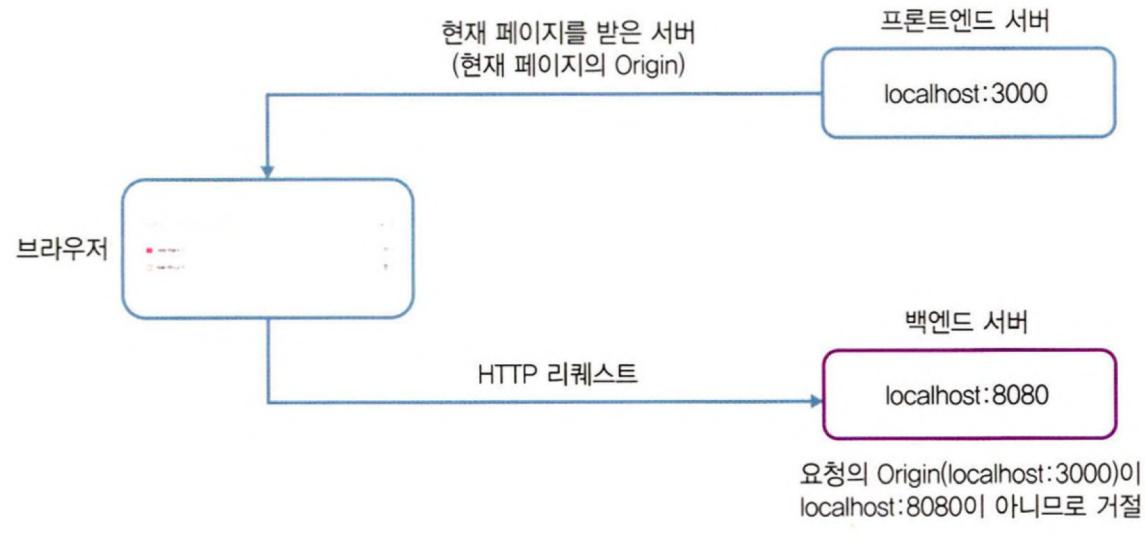


서비스 통합

CORS

❖ CORS

- ✓ Cross-Origin Resource Sharing 의 약자(https://en.wikipedia.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

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

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

CORS

❖ CORS

- ✓ 브라우저의 검사 창에서 확인 – 에러 발생

- ✗ Access to fetch at '<http://localhost/todo>' from origin '<http://localhost:3000> localhost:1' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.
- ✗ Failed to load resource: net::ERR_FAILED [/todo:1](#) 

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://localhost: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

The screenshot shows the Network tab of the Chrome DevTools. At the top, there is a message: "Download the React DevTools for a better development experience: <https://reactjs.org/link/react-devtools>". Below the message, the Network tab has several sections: "Preserve log", "Disable cache", "No throttling", "Filter" (with "Invert" and "Hide data URLs" checkboxes), and a list of request types: All, Fetch/XHR, JS, CSS, Img, Media, Font, Doc, WS, Wasm, Manifest, Other, and "Has blocked cookies". There is also a checkbox for "Blocked Requests" and another for "3rd-party requests". The main area displays a timeline with time markers at 20 ms, 40 ms, 60 ms, 80 ms, and 100 ms. Below the timeline is a table of network requests:

Name	Status	Type	Initiator	Size	Time	Waterfall
todo	200	prefligh...	Preflight ↗	0 B	5 ms	
todo	200	fetch	App.js:19	378 B	191 ms	
todo	200	fetch	App.js:19	378 B	195 ms	
manifest.json	304	mani...	Other	363 B	4 ms	
favicon.ico	304	x-icon	Other	363 B	3 ms	
logo192.png	304	png	Other	364 B	3 ms	

JavaScript Data Request

- ❖ ajax

- ✓ 콜백을 이용한 XMLHttpRequest 처리

```
var oReq = new XMLHttpRequest();
oReq.open("GET , "http://localhost:/todo );
oReq.send();

oReq.addEventListener('load', function () {
    // callback 함수
    console.log(oReq.response);
};
```

JavaScript Data Request

❖ ajax

- ✓ Promise를 이용한 XMLHttpRequest를 HTTP 요청

```
function exampleFunction() {
    return new Promise((resolve, reject) => {
        var oReq = new XMLHttpRequest();
        oReq.open( , http://localhost/todo );
        oReq.onload = function () {
            resolve(oReq. response); // Resolve 상태
        };
        oReq.onerror = function () {
            reject(oReq. response); // Reject 상태
        };
        oReq.send(); // Pending 상태
    });
}

exampleFunction()
    .then((r) => console.log(r))
    .catch((e) => console.log(e));
```

JavaScript Data Request

- ❖ Fetch API
 - ✓ 요청

```
fetch("localhost/todo") // GET 메서드를 이용해 보냄
  .then(response => {
    // response 수신 시 하고 싶은 작업
  })
  .catch(e => {
    // 에러가 났을 때 하고 싶은 작업
  })
```

JavaScript Data Request

- ❖ Fetch API

- ✓ 매개변수 전달

```
options = {  
    method: POST ,  
    headers: [  
        ["Content-Type , "application/json ]  
    ],  
    body: JSON.stringify(data)  
};  
  
fetch("localhost/todo", options)  
    .then(response => {  
        // response 수신 시 하고 싶은 작업  
    })  
    .catch(e => {  
        // 에러가 났을 때 하고 싶은 작업  
    })
```

ToDo

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

```
let backendHost;  
  
const hostname = window && window.location && window.location.hostname;  
  
if (hostname === "localhost") {  
  backendHost = "http://localhost";  
}  
  
export const API_BASE_URL = `${backendHost}`;
```

ToDo

- ❖ BackEnd의 데이터를 가져오는 함수를 별도의 파일에 생성 – src/service/Api-Service.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);
    }
    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;
        })
    );
}
```

ToDo

- ❖ app.js 파일의 내용 수정 – 삽입, 삭제, 가져오기 적용

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

ToDo

- ❖ app.js 파일의 내용 수정 – 삽입, 삭제, 가져오기 적용

```
class App extends React.Component{
  constructor(props) {
    super(props);
    this.state = {items: []};
  }

  componentDidMount() {
    call("/todo", "GET", null).then((response) =>
      this.setState({ items: response.data })
    );
  }

  add = (item) => {
    call("/todo", "POST", item).then((response) =>
      this.setState({ items: response.data })
    );
  };

  delete = (item) => {
    call("/todo", "DELETE", item).then((response) =>
      this.setState({ items: response.data })
    );
  };
}
```

ToDo

- ❖ app.js 파일의 내용 수정 – 삽입, 삭제, 가져오기 적용

```
render() {
  var todoltems = this.state.items.length > 0 && (
    <Paper style={{ margin: 16 }}>
      <List>
        {this.state.items.map((item, idx) => (
          <ToDo item={item} key={item.id} delete={this.delete}/>
        ))}
      </List>
    </Paper>
  );
}

return (
  <div className="App">
    <Container maxWidth="md">
      <AddToDo add={this.add}/>
      <div className="ToDoList">{todoltems}</div>
    </Container>
  </div>
);
}

export default App;
```

ToDo

- ❖ 브라우저 와 데이터베이스에서 확인

Add Todo here

- Java
- Python

123 done	T \uparrow \downarrow	ABC title	T \uparrow \downarrow	ABC user_id	T \uparrow \downarrow
0	Java			temporary-user	
0	Python			temporary-user	

ToDo

- ❖ app.js 파일의 내용 수정 – 수정 적용

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

class App extends React.Component{
  constructor(props) {
    super(props);
    this.state = {items: []};
  }

  componentDidMount() {
    call("/todo", "GET", null).then((response) =>
      this.setState({ items: response.data })
    );
  }
}
```

ToDo

- ❖ app.js 파일의 내용 수정 – 수정 적용

```
add = (item) => {
  call("/todo", "POST", item).then((response) =>
    this.setState({ items: response.data })
  );
};

delete = (item) => {
  call("/todo", "DELETE", item).then((response) =>
    this.setState({ items: response.data })
  );
};

update = (item) => {
  call("/todo", "PUT", item).then((response) =>
    this.setState({ items: response.data })
  );
};
```

ToDo

- ❖ app.js 파일의 내용 수정 – 수정 적용

```
render() {
  var todoltems = this.state.items.length > 0 && (
    <Paper style={{ margin: 16 }}>
      <List>
        {this.state.items.map((item, idx) => (
          <ToDo item={item}
            key={item.id}
            delete={this.delete}
            update={this.update}/>
        ))}
      </List>
    </Paper>
  );
}
```

ToDo

- ❖ app.js 파일의 내용 수정 – 수정 적용

```
return (  
  <div className="App">  
    <Container maxWidth="md">  
      <AddToDo add={this.add}/>  
      <div className="ToDoList">{todoItems}</div>  
    </Container>  
  </div>  
)  
}  
}  
  
export default App;
```

ToDo

- ❖ ToDo.js 파일의 내용 수정 – 수정 적용

```
import React from "react";

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

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

class ToDo extends React.Component {
  constructor(props) {
    super(props);
    this.state = { item: props.item, readOnly: true };
    this.delete = props.delete;
    this.update = props.update;
  }
}
```

ToDo

- ❖ ToDo.js 파일의 내용 수정 – 수정 적용

```
deleteEventHandler = () => {
    this.delete(this.state.item);
};
```

```
offReadOnlyMode = () => {
    console.log("Event!", this.state.readOnly);
    this.setState({ readOnly: false }, () => {
        console.log("ReadOnly? ", this.state.readOnly);
    });
};
```

```
enterKeyEventHandler = (e) => {
    if (e.key === "Enter") {
        this.setState({ readOnly: true });
        this.update(this.state.item);
    }
};
```

ToDo

- ❖ ToDo.js 파일의 내용 수정 – 수정 적용

```
editEventHandler = (e) => {
  const thisItem = this.state.item;
  thisItem.title = e.target.value;
  this.setState({ item: thisItem });
};

checkboxEventHandler = (e) => {
  const thisItem = this.state.item;
  thisItem.done = !thisItem.done;
  this.setState({ item: thisItem });
  this.update(this.state.item);
};
```

ToDo

- ❖ ToDo.js 파일의 내용 수정 – 수정 적용

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

ToDo

- ❖ ToDo.js 파일의 내용 수정 – 수정 적용

```
<ListItemIconSecondaryAction>
  <IconButton
    aria-label="Delete Todo"
    onClick={this.deleteEventHandler}
  >
    <DeleteOutlined />
  </IconButton>
</ListItemIconSecondaryAction>
</ListItemIcon>
);
}
}

export default ToDo;
```