Spring CQRS

기본 설정

- 데이터 업데이트를 위한 MySQL Database 생성 및 접속 확인

기본 설정

- 데이터 읽기를 위한 Mongo Database 생성 및 접속 확인
  - ✓ 서버 설치: https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/#std-label-install-mdb-community-ubuntu
    - gnupg(암호화 소프트웨어) 와 curl(다양한 통신 프로토콜을 이용하여 데이터를 전송하기 위한 라이브러리와 명령 줄 도구를 제공하는 컴퓨터 소프트웨어) 설치
      - sudo apt-get install gnupg curl
    - Mongo DB public key 가져오기
      - curl -fsSL https://pgp.mongodb.com/server-7.0.asc | sudo gpg -o /usr/share/keyrings/mongodb-server-7.0.gpg --dearmor
    - Mongo DB 목록 만들기
      - echo "deb [arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-7.0.gpg] https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-7.0.list deb [arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-7.0.gpg] https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0 multiverse

기본 설정

- ❖ 데이터 읽기를 위한 Mongo Database 생성 및 접속 확인
  - ✓ 서비스 관련 명령
    - 패키지 정보 업데이트
      - ◆ sudo apt-get update
    - Mongo DB 설치
      - ◆ sudo apt-get install -y mongodb-org
    - 서비스 시작
      - ◆ sudo systemctl start mongod
    - 서비스 확인
      - ◆ sudo systemctl status mongod
    - 부팅 시 시작
      - ◆ sudo systemctl enable mongod

기본 설정

- ❖ 데이터 읽기를 위한 Mongo Database 생성 및 접속 확인
  - ✓ 계정 생성 없이 외부 접속 허용
    - 설정 파일 수정: sudo nano /etc/mongod.conf

      net:
        port: 27017
        #bindIp: 127.0.0.1
        bindIp: 0.0.0.0
    - 방화벽에서 27017 포트 개방

      sudo ufw allow 27017
    - EC2의 보안 그룹에서 27017 포트를 외부에서 접속할 수 있도록 인바운드 규칙 수정
    - EC2에서 Mongo DB 서버 재시작

      sudo systemctl restart mongod
    - 접속
      - ◆ Mongo Shell: mongosh --host 3.84.200.63 --port 27017
      - ◆ Compass: mongodb://3.84.200.63:27017

기본 설정

- ❖ 데이터 읽기를 위한 Mongo Database 생성 및 접속 확인
  - ✓ 계정을 이용한 외부 접속
    - ● mongosh 명령으로 셸에 접속
      - ◆ 데이터베이스 접속: use admin
      - ◆ 유저와 비밀번호 설정

```
db.createUser(
  {
    user: "계정",
    pwd: "비밀번호",
    roles: [
      { role: "userAdminAnyDatabase", db: "admin" },
      { role: "readWriteAnyDatabase", db: "admin" }
    ]
  }
)
```

기본 설정

- ❖ 데이터 읽기를 위한 Mongo Database 생성 및 접속 확인
  - ✓ 계정을 이용한 외부 접속
    - 설정 파일에 추가: sudo nano /etc/mongod.conf

      security:

        authorization: 'enabled'
    - 서버 재시작: sudo systemctl restart mongod

기본 설정

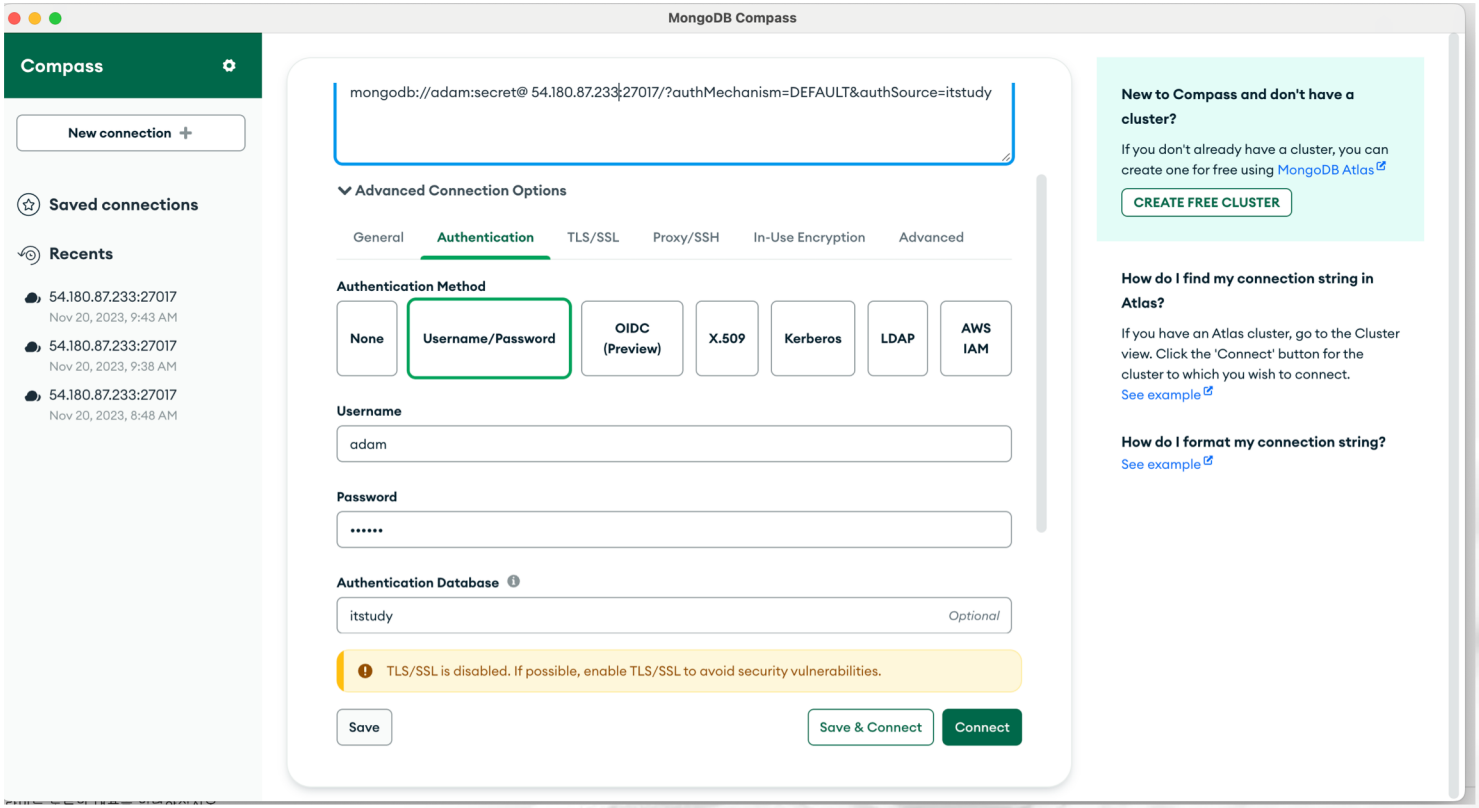
- ❖ 데이터 읽기를 위한 Mongo Database 생성 및 접속 확인
  - ✓ 계정을 이용한 외부 접속
    - ● 접속
      - ◆ Mongo Shell: mongosh -u myUseradmin -p wnddkd --host 3.84.200.63 --port 27017
      - ◆ Compass: mongodb://myUseradmin:wnddkd@3.84.200.63:27017/?authMechanism=DEFAULT&tls=true

기본 설정

❖ 데이터 읽기를 위한 Mongo Database 생성 및 접속 확인

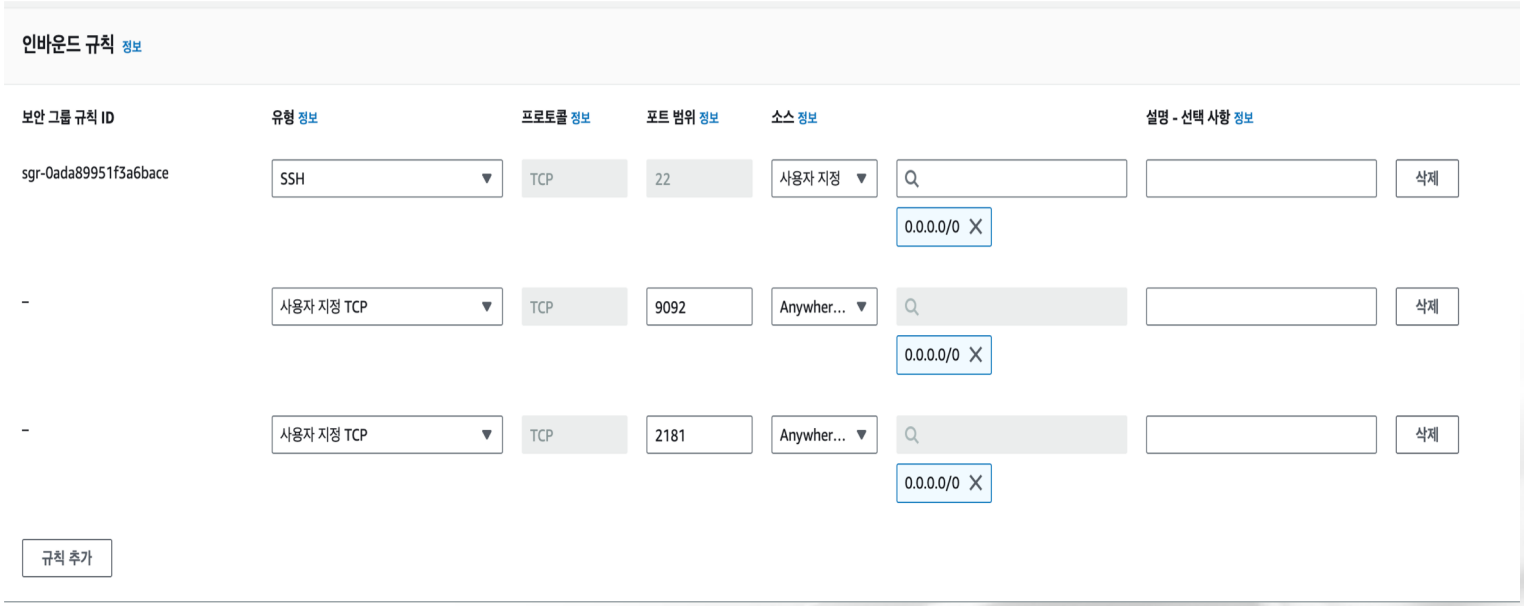
✓ 계정을 이용한 외부 접속

● 접속



기본 설정

- ❖ Kafka 설정
  - ✓ EC2에 Kafka 설치
    - ● EC2 인스턴스 생성
      - ◆ 9092, 2181 번 포트의 InBound 설정에 모든 IP를 허용



      - ◆ EC2 인스턴스에 접속

기본 설정

- ❖ Kafka 설정
  - ✓ EC2에 Kafka 설치
    - ● JDK 설치
      - ◆ sudo apt-get update
      - ◆ sudo apt install –y openjdk-17-jdk
      - ◆ java -version

        openjdk version "17.0.11" 2024-04-16

        OpenJDK Runtime Environment (build 17.0.11+9-Ubuntu-1)

        OpenJDK 64-Bit Server VM (build 17.0.11+9-Ubuntu-1, mixed mode, sharing)

기본 설정

- ❖ Kafka 설정
  - ✓ EC2에 Kafka 설치
    - ● 카프카 다운로드 및 설치
      - ◆ 카프카 바이너리 패키지 다운로드 - https://kafka.apache.org/downloads
        - ▪ wget https://archive.apache.org/dist/kafka/3.6.0/kafka_2.13-3.6.0.tgz
      - ◆ 압축 해제
        - ▪ tar xvf kafka_2.13-3.6.0.tgz
      - ◆ 압축 해제 확인
        - ▪ ll

```
total 110640
drwxr-x--- 5 ubuntu ubuntu      4096 Nov  2 05:32 ./
drwxr-xr-x 3 root   root        4096 Nov  2 05:09 ../
-rw-r--r-- 1 ubuntu ubuntu       220 Jan  6  2022 .bash_logout
-rw-r--r-- 1 ubuntu ubuntu      3771 Jan  6  2022 .bashrc
drwx------ 2 ubuntu ubuntu      4096 Nov  2 05:12 .cache/
-rw-r--r-- 1 ubuntu ubuntu       807 Jan  6  2022 .profile
drwx------ 2 ubuntu ubuntu      4096 Nov  2 05:09 .ssh/
-rw-r--r-- 1 ubuntu ubuntu         0 Nov  2
    05:13 .sudo_as_admin_successful
drwxr-xr-x 7 ubuntu ubuntu      4096 Sep 29 05:03 kafka_2.13-3.6.0/
-rw-rw-r-- 1 ubuntu ubuntu 113257079 Oct  4 03:54 kafka_2.13-3.6.0.tgz
```

기본 설정

- ❖ Kafka 설정
  - ✓ EC2에 Kafka 설치
    - ● 카프카 설정
      - ◆ 디렉토리 변경: sudo mv kafka_2.13-3.6.0 /opt/kafka
      - ◆ 환경 변수 추가: nano ~/.bashrc

        export KAFKA_HOME=/opt/kafka

        export PATH=$PATH:$KAFKA_HOME/bin
      - ◆ 변경한 내용 반영

        source ~/.bashrc

기본 설정

- ❖ Kafka 설정
  - ✓ EC2에 Kafka 설치
    - ● 카프카 설정
      - ◆ opt/kafka/config 디렉토리에 있는 server.properties 파일에 추가

```
# The address the socket server listens on. It will get the value returned
    from
# java.net.InetAddress.getCanonicalHostName() if not configured.
#   FORMAT:
#     listeners = listener_name://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://:9092
# Hostname and port the broker will advertise to producers and consumers.
    If not set,
# it uses the value for "listeners" if configured.  Otherwise, it will use the
    value
# returned from java.net.InetAddress.getCanonicalHostName().
advertised.listeners=PLAINTEXT://공인ip:9092

delete.topic.enable=true
auto.create.topics.enable=true
log.retention.minutes=10
```

기본 설정

- Kafka 설정
  - ✓ 주키퍼 실행
    - zookeeper-server-start.sh -daemon /opt/kafka/config/zookeeper.properties
    - jps -vm

기본 설정

- Kafka 설정
  - ✓ 카프카 브로커 실행
    - kafka-server-start.sh -daemon /opt/kafka/config/server.properties
    - jps -m

기본 설정

- ❖ Kafka 설정
  - ✓ 토픽 생성 과 조회
    - ● 토픽 생성
      - ◆ kafka-topics.sh --create --bootstrap-server localhost:9092 --topic cqrs-topic

        Created topic cqrs-topic
    - ● 토픽 리스트 조회
      - ◆ kafka-topics.sh --bootstrap-server localhost:9092 --list

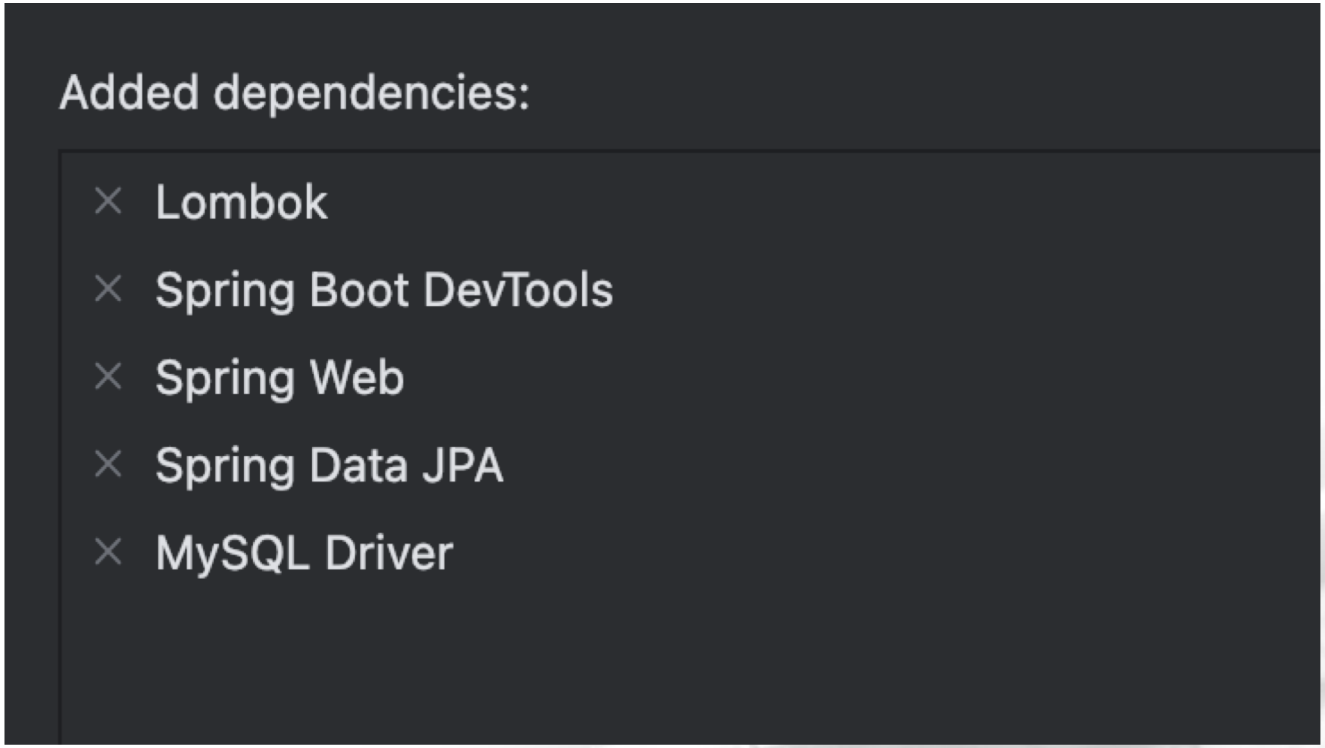
        cqrs-topic

기본 설정

- ❖ Kafka 설정
  - ✓ 메시지 전송 과 확인
    - ● 메시지 전송
      - ◆ kafka-console-producer.sh --bootstrap-server localhost:9092 --topic cqrs-topic
    - ● 메시지 수신
      - ◆ kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic cqrs-topic --from-beginning

구현

- ❖ SpringBoot 데이터 기록 프로젝트
  - ✓ SpringBoot 프로젝트 생성
    - 의존성



구현

- SpringBoot 데이터 기록 프로젝트
  - ✓ application.yml 파일에 기본 설정

```
#웹 서버 실행 포트 설정
server:
  port: 7000

#데이터베이스 접속 정보
spring:
  datasource:
    url: jdbc:mysql://itstudymysql.cesn3uejbkwe.ap-northeast-
2.rds.amazonaws.com:3306/itstudy
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: admin
    password: itstudy1025!

  jpa:
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        format_sql: true
        show_sql: true
```

구현

- SpringBoot 데이터 기록 프로젝트
  - ✓ application.yml 파일에 기본 설정

```
#쿼리에 물음표로 출력되는 바인드 파라미터 출력
logging:
  level:
    org.hibernate.type.descriptor.sql: trace
```

구현

- SpringBoot 데이터 기록 프로젝트
  - ✓ CORS 설정을 위한 클래스 추가 - WebConfig

```
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedOrigins("http://localhost:3000");
    }
}
```

구현

- SpringBoot 데이터 기록 프로젝트
  - ✓ Entity 클래스 추가 - Book

```
package com.example.cqrs_write;

import jakarta.persistence.*;
import lombok.*;
import java.util.Date;

@Entity
@Table(name= "book")
@ToString
@Getter
@Builder
@AllArgsConstructor
@NoArgsConstructor
```

구현

- SpringBoot 데이터 기록 프로젝트
  - ✓ Entity 클래스 추가 - Book

```
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long bid;
    @Column(length = 50, nullable = false)
    private String title;
    @Column(length = 50, nullable = false)
    private String author;
    @Column(length = 50, nullable = false)
    private String category;
    @Column
    private int pages;
    @Column
    private int price;
    @Column
    private Date published_date;
    @Column(length = 50, nullable = false)
    private String description;
}
```

구현

- SpringBoot 데이터 기록 프로젝트
  - ✓ DTO 클래스 추가 - BookDTO

```
import lombok.Data;

@Data
public class BookDTO {
    private String title;
    private String author;
    private String category;
    private int pages;
    private int price;
    private String published_date;
    private String description;
}
```

구현

- SpringBoot 데이터 기록 프로젝트
  - ✓ Repository 인터페이스 추가 - BookRepository

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

}
```

구현

- SpringBoot 데이터 기록 프로젝트
  - ✓ Service 클래스 추가 - BookService

```
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

@Service
@RequiredArgsConstructor
public class BookService {
    private final BookRepository bookRepository;
```

구현

- SpringBoot 데이터 기록 프로젝트
  - ✓ Service 클래스 추가 - BookService

```
public void saveBook(BookDTO bookDTO) {
    try {
        System.out.println(bookDTO);
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd",
Locale.ENGLISH);
        Date published_date = formatter.parse(bookDTO.getPublished_date());
        Book book = Book.builder()
                .title(bookDTO.getTitle())
                .author(bookDTO.getAuthor())
                .category(bookDTO.getCategory())
                .pages(bookDTO.getPages())
                .price(bookDTO.getPrice())
                .published_date(published_date)
                .description(bookDTO.getDescription())
                .build();
        bookRepository.save(book);
    }
    catch(Exception e){
        System.out.println(e.getMessage());
    }
  }
}
```

구현

- SpringBoot 데이터 기록 프로젝트
  - ✓ Controller 클래스 추가 - BookController

```
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequiredArgsConstructor
public class BookController {
    private final  BookService bookService;
    @PostMapping("/cqrs/book")
    public String saveBook(@RequestBody BookDTO bookDTO){
        System.out.println(bookDTO.toString());
        bookService.saveBook(bookDTO);
        return "success";
    }
}
```

구현

- ❖ 클라이언트 프로젝트
  - ✓ 리액트 프로젝트 생성: npx create-react-app react_cqrs
  - ✓ 패키지 설치
    - npm install --save --legacy-peer-deps @material-ui/core
    - npm install --save --legacy-peer-deps @material-ui/icons
    - yarn add axios
  - ✓ 시작: npm run start

구현

❖ 클라이언트 프로젝트

✓ 데이터 삽입을 위한 컴포넌트 생성 – AddBook.jsx

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

function AddBook(props) {
    const [title, setTitle] = useState("");
    const [author, setAuthor] = useState("");
    const [category, setCategory] = useState("");
    const [pages, setPages] = useState("");
    const [price, setPrice] = useState("");
    const [published_date, setPublished_date] = useState("");
    const [description, setDescription] = useState("");
```

구현

- 클라이언트 프로젝트
  - ✓ 데이터 삽입을 위한 컴포넌트 생성 – AddBook.jsx

```
const onTitleChange = (event) => {
    setTitle(event.target.value);
};
const onAuthorChange = (event) => {
    setAuthor(event.target.value);
};
const onCategoryChange = (event) => {
    setCategory(event.target.value);
};
const onPagesChange = (event) => {
    setPages(event.target.value);
};
const onPriceChange = (event) => {
    setPrice(event.target.value);
};
const onPublished_dateChange = (event) => {
    setPublished_date(event.target.value);
};
const onDescriptionChange = (event) => {
    setDescription(event.target.value);
};
```

구현

- 클라이언트 프로젝트
  - ✓ 데이터 삽입을 위한 컴포넌트 생성 – AddBook.jsx

```
const onSubmit = (event) => {
    event.preventDefault();
    const book = {}
    book.title=title
    book.author = author
    book.category = category
    book.pages = pages
    book.price = price
    book.published_date = published_date
    book.description = description
    props.add(book)
    setTitle("")
    setAuthor("")
    setCategory("")
    setPages("")
    setPrice("")
    setPublished_date("")
    setDescription("")
};
```

구현

- 클라이언트 프로젝트
  - ✓ 데이터 삽입을 위한 컴포넌트 생성 – AddBook.jsx

```
return(
    <Paper style={{ margin: 16, padding: 16 }}>
    <Grid container>
      <Grid xs={6} md={6} item style={{ paddingRight: 16 }}>
        <TextField
          onChange={onTitleChange}
          value = {title}
          placeholder="Add Book Title"
          fullWidth
        />
      </Grid>
      <Grid xs={6} md={6} item style={{ paddingRight: 16 }}>
        <TextField
        onChange={onAuthorChange}
        value = {author}
          placeholder="Add Book Author"
          fullWidth
        />
      </Grid>
```

구현

- 클라이언트 프로젝트
  - ✓ 데이터 삽입을 위한 컴포넌트 생성 – AddBook.jsx

```
<Grid xs={3} md={3} item style={{ paddingRight: 16 }}>
  <TextField
  onChange={onCategoryChange}
  value = {category}
    placeholder="Add Book Category"
    fullWidth
  />
</Grid>
<Grid xs={3} md={3} item style={{ paddingRight: 16 }}>
  <TextField
  onChange={onPagesChange}
  value = {pages}
    placeholder="Add Book Pages"
    fullWidth
  />
</Grid>
```

구현

- 클라이언트 프로젝트
  - ✓ 데이터 삽입을 위한 컴포넌트 생성 – AddBook.jsx

```
<Grid xs={3} md={3} item style={{ paddingRight: 16 }}>
  <TextField
  onChange={onPriceChange}
  value = {price}
    placeholder="Add Book Price"
    fullWidth
  />
</Grid>
<Grid xs={3} md={3} item style={{ paddingRight: 16 }}>
  <TextField
  onChange={onPublished_dateChange}
  value = {published_date}
    placeholder="Add Book Published_Date"
    fullWidth
  />
</Grid>
```

구현

- 클라이언트 프로젝트
  - ✓ 데이터 삽입을 위한 컴포넌트 생성 – AddBook.jsx

```
<Grid xs={11} md={11} item style={{ paddingRight: 16 }}>
  <TextField
  onChange={onDescriptionChange}
  value = {description}
    placeholder="Add Book Description"
    fullWidth
  />
</Grid>
```

구현

- 클라이언트 프로젝트
  - ✓ 데이터 삽입을 위한 컴포넌트 생성 – AddBook.jsx

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

export default AddBook;
```

구현

- 클라이언트 프로젝트
  - ✓ App.js 수정

```
import './App.css';
import {Paper} from "@material-ui/core"
import AddBook from "./AddBook";
import Axios from "axios";

function App() {
  //데이터 추가를 위한 함수
  const add = (book) => {
      console.log("book : ", book);
      Axios.post("http://localhost:7000/cqrs/book", book).then((response) => {
        console.log(response.data)
        if (response.data === "success") {
          alert("저장에 성공했습니다.")
        } else {
          alert("코멘트를 저장하지 못했습니다.");
        }
      });
  };
```

구현

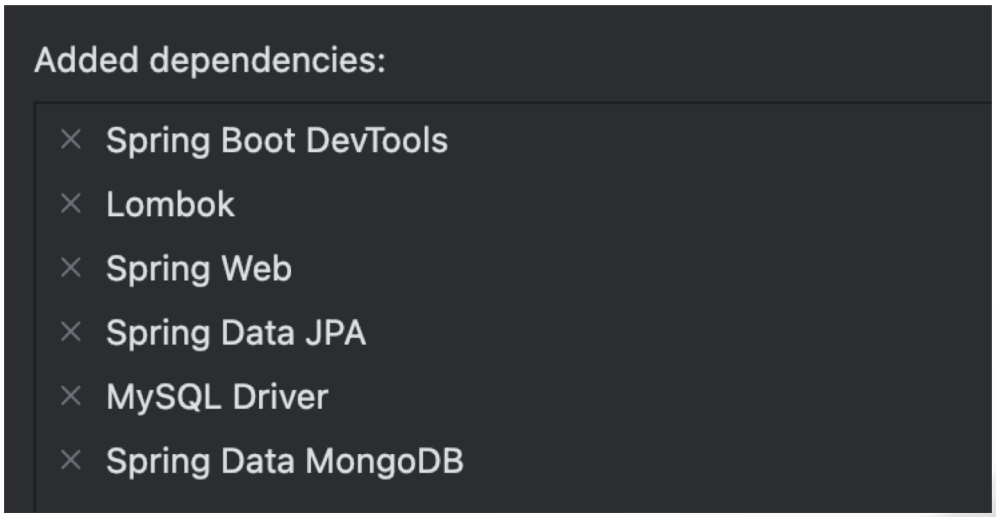
- 클라이언트 프로젝트
  - ✓ App.js 수정

```
    return (
      <div className="App">
        <Paper style={{ margin: 16 }}>
          <AddBook add = {add}/>
        </Paper>
      </div>
    );
  }

  export default App;
```

구현

- ❖ SpringBoot 데이터 읽기 프로젝트
  - ✓ SpringBoot 프로젝트 생성
    - ● 의존성



구현

- SpringBoot 데이터 읽기 프로젝트
  - ✓ MySQL에 초기 데이터가 없다면 MySQL 설정 부분 과 StartListener 작업 제거

구현

- SpringBoot 데이터 읽기 프로젝트
  - ✓ application.yml

```
#웹 서버 실행 포트 설정
server:
  port: 8000

#데이터베이스 접속 정보
spring:
  datasource:
    url: jdbc:mysql://itstudymysql.cesn3uejbkwe.ap-northeast-
2.rds.amazonaws.com:3306/itstudy
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: admin
    password: itstudy1025!

  jpa:
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        format_sql: true
        show_sql: true
```

구현

- SpringBoot 데이터 읽기 프로젝트
  - ✓ application.yml

```
#쿼리에 물음표로 출력되는 바인드 파라미터 출력
logging:
  level:
    org.hibernate.type.descriptor.sql: trace
```

구현

- SpringBoot 데이터 읽기 프로젝트
  - ✓ Book 클래스

```
import jakarta.persistence.*;
import lombok.*;

import java.util.Date;

@Entity
@Table(name= "book")
@ToString
@Getter
@Builder
@AllArgsConstructor
@NoArgsConstructor
```

구현

- SpringBoot 데이터 읽기 프로젝트
  - ✓ Book 클래스

```
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long bid;
    @Column(length = 50, nullable = false)
    private String title;
    @Column(length = 50, nullable = false)
    private String author;
    @Column(length = 50, nullable = false)
    private String category;
    @Column
    private int pages;
    @Column
    private int price;
    @Column
    private Date published_date;
    @Column(length = 50, nullable = false)
    private String description;
}
```

구현

- SpringBoot 데이터 읽기 프로젝트
  - ✓ BookRepository 인터페이스

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

}
```

구현

- SpringBoot 데이터 읽기 프로젝트
  - ✓ CORS 환경 설정 클래스

```
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedOrigins("http://localhost:3000");
    }
}
```

구현

- SpringBoot 데이터 읽기 프로젝트
  - ✓ 애플리케이션이 시작하자 마자 수행되는 클래스

```
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import lombok.RequiredArgsConstructor;
import org.bson.Document;
import org.springframework.boot.context.event.ApplicationStartedEvent;
import org.springframework.context.ApplicationListener;
import org.springframework.stereotype.Component;
import java.util.List;
```

구현

- SpringBoot 데이터 읽기 프로젝트
  - ✓ 애플리케이션이 시작하자 마자 수행되는 클래스

```
@Component
@RequiredArgsConstructor
public class StartListener implements
ApplicationListener<ApplicationStartedEvent> {
    private final BookRepository bookRepository;

    @Override
    public void onApplicationEvent(ApplicationStartedEvent
applicationStartedEvent) {
        List<Book> books = bookRepository.findAll();
        MongoClient mongoClient =
MongoClients.create("mongodb://adam:wnddkd@43.200.180.22:27017");
        MongoDatabase database = mongoClient.getDatabase("itstudy");
        MongoCollection<Document> mongo_books =
database.getCollection("books");
        mongo_books.drop();
        mongo_books = database.getCollection("books");
```

구현

- SpringBoot 데이터 읽기 프로젝트
  - ✓ 애플리케이션이 시작하자 마자 수행되는 클래스

```
            for(Book book : books) {
                Document mongoBook = new Document();
                mongoBook.append("bid", book.getBid());
                mongoBook.append("title", book.getTitle());
                mongoBook.append("author", book.getAuthor());
                mongoBook.append("category", book.getCategory());
                mongoBook.append("pages", book.getPages());
                mongoBook.append("price", book.getPrice());
                mongoBook.append("published_date",
        book.getPublished_date().toString());
                mongoBook.append("description", book.getDescription());
                mongo_books.insertOne(mongoBook);
            }
            mongoClient.close();
        }
    }
```

구현

- SpringBoot 데이터 읽기 프로젝트
  - ✓ Controller 클래스

```
import com.mongodb.client.*;
import lombok.RequiredArgsConstructor;
import org.bson.Document;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

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

구현

- SpringBoot 데이터 읽기 프로젝트
  - ✓ Controller 클래스

```
@RestController
@RequiredArgsConstructor
public class BookController {
    @GetMapping("/cqrs/book")
    public ResponseEntity<List> getBooks(){
        MongoClient mongoClient =
MongoClients.create("mongodb://adam:wnddkd@43.200.180.22:27017");
        MongoDatabase database = mongoClient.getDatabase("itstudy");
        MongoCollection<Document> mongo_books =
database.getCollection("books");
        // 데이터를 저장해서 리턴할 인스턴스 생성
        List<Document> list = new ArrayList<Document>();
```

구현

- SpringBoot 데이터 읽기 프로젝트
  - ✓ Controller 클래스

```
        try {

            try (MongoCursor<Document> cur = mongo_books.find().iterator())
                                        {
                while (cur.hasNext()) {
                    Document doc = cur.next();
                    list.add(doc);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            mongoClient.close();
        }
        return ResponseEntity.status(HttpStatus.OK).body(list);
    }
}
```

구현

- 클라이언트 프로젝트
  - ✓ app.js 수정

```
import './App.css';
import {Paper} from "@material-ui/core"
import AddBook from "./AddBook";
import Axios from "axios";
import React, {useEffect, useState} from 'react';

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

  useEffect(() => {
    console.log("화면이 렌더링 된 이후에 바로 수행이 됨: componentDidMount()과 동일
");
    Axios.get("http://127.0.0.1:8000/cqrs/book/").then((response) => {
        //console.log(response.data)
        if (response.data) {
          setItems(response.data)
        } else {
          alert("읽기 실패");
        }
      });
  }, []);
```

구현

- 클라이언트 프로젝트
  - ✓ app.js 수정

```
//데이터 추가를 위한 함수
const add = (book) => {
    console.log("book : ", book);
    Axios.post("http://127.0.0.1:7000/cqrs/book/", book).then((response) => {
      console.log(response.data)
      if (response.data.bid) {
        alert("저장에 성공했습니다.")
      } else {
        alert("코멘트를 저장하지 못했습니다.");
      }
    });
};
```

구현

- 클라이언트 프로젝트
  - ✓ app.js 수정

```
return (
  <div className="App">
    <Paper style={{ margin: 16 }}>
      <AddBook add = {add}/>
    </Paper>
      {items.map((item, index) => (
        <p key = {index}>
          {item.title}
        </p>
      ))}
  </div>
 );
}
export default App;
```

구현

- ❖ 카프카 연결
  - ✓ DataWrite 프로젝트 수정
    - ● build.gradle 파일에 의존성 추가

```
implementation 'org.springframework.kafka:spring-kafka'
```

구현

- ❖ 카프카 연결
  - ✓ DataWrite 프로젝트 수정
    - ● application.yml 파일에 카프카 설정 추가

```
spring:
    kafka:
      bootstrap-servers: 43.201.249.166:9092
      consumer:
        group-id: adamsoft
        auto-offset-reset: earliest
        key-deserializer:
org.apache.kafka.common.serialization.StringDeserializer
        value-deserializer:
org.apache.kafka.common.serialization.StringDeserializer
      producer:
        key-serializer: org.apache.kafka.common.serialization.StringSerializer
        value-serializer: org.apache.kafka.common.serialization.StringSerializer
```

구현

- ❖ 카프카 연결
  - ✓ DataWrite 프로젝트 수정
    - ● 카프카 환경 설정 클래스 추가

```
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.StringSerializer;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.core.DefaultKafkaProducerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.core.ProducerFactory;

import java.util.HashMap;
import java.util.Map;
```

구현

- ❖ 카프카 연결
  - ✓ DataWrite 프로젝트 수정
    - ● 카프카 환경 설정 클래스 추가

```
@Configuration
public class KafkaConfiguration {
    @Value("${spring.kafka.bootstrap-servers}")
    private String bootstrapServers;

    @Bean
    public ProducerFactory<String, String> producerFactory() {

        Map<String,Object> configs = new HashMap<>();
        configs.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
bootstrapServers);
        configs.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);
        configs.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);
        return new DefaultKafkaProducerFactory(configs);
    }
```

구현

- ❖ 카프카 연결
  - ✓ DataWrite 프로젝트 수정
    - ● 카프카 환경 설정 클래스 추가

```
    @Bean
    public KafkaTemplate<String, String> kafkaTemplate() {
        return new KafkaTemplate<>(producerFactory());
    }
}
```

구현

- ❖ 카프카 연결
  - ✓ DataWrite 프로젝트 수정
    - ● BookDTO 클래스 수정

```
import lombok.Data;

@Data
public class BookDTO {
    private Long bid;
    private String title;
    private String author;
    private String category;
    private int pages;
    private int price;
    private String published_date;
    private String description;
}
```

구현

- 카프카 연결
  - ✓ DataWrite 프로젝트 수정
    - 카프카 메시지 전송 클래스 추가

```
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;
```

구현

- ❖ 카프카 연결
  - ✓ DataWrite 프로젝트 수정
    - ● 카프카 메시지 전송 클래스 추가

```
@Service
@RequiredArgsConstructor
public class KafkaProducer {
    private static final String TOPIC = "cqrs-topic";
    @Autowired
    private  KafkaTemplate<String, String> kafkaTemplate;
    private final Logger log = LoggerFactory.getLogger(getClass());
```

구현

- ❖ 카프카 연결
  - ✓ DataWrite 프로젝트 수정
    - ● 카프카 메시지 전송 클래스 추가

```
public void sendMessage(BookDTO bookDTO) {
        String message =
                "{₩"bid₩":" + "₩"" + bookDTO.getBid() + "₩""
                        + ", ₩"title₩":" + "₩"" + bookDTO.getTitle() +  "₩""
                        + ", ₩"author₩":" + "₩"" + bookDTO.getAuthor() +  "₩""
                        + ", ₩"category₩":" + "₩"" + bookDTO.getCategory() +  "₩""
                        + ", ₩"pages₩":" + "₩"" + bookDTO.getPages() +  "₩""
                        + ", ₩"price₩":" + "₩"" + bookDTO.getPrice() +  "₩""
                        + ", ₩"published_date₩":" + "₩"" +
bookDTO.getPublished_date().toString() +  "₩""
                        + ", ₩"description₩":" + "₩"" + bookDTO.getDescription() +
"₩""
                + "}";
        System.out.println(message);
        this.kafkaTemplate.send(TOPIC, message);
    }
}
```

구현

- ❖ 카프카 연결
  - ✓ DataWrite 프로젝트 수정
    - ● Service 클래스 수정

```
@Service
@RequiredArgsConstructor
public class BookService {
    private final BookRepository bookRepository;
    private final KafkaProducer kafkaProducer;

    public void saveBook(BookDTO bookDTO) {
        try {
            System.out.println(bookDTO);
            SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd",
Locale.ENGLISH);
            Date published_date = formatter.parse(bookDTO.getPublished_date());
```

구현

- ❖ 카프카 연결
  - ✓ DataWrite 프로젝트 수정
    - ● Service 클래스 수정

```
                Book book = Book.builder()
                        .title(bookDTO.getTitle())
                        .author(bookDTO.getAuthor())
                        .category(bookDTO.getCategory())
                        .pages(bookDTO.getPages())
                        .price(bookDTO.getPrice())
                        .published_date(published_date)
                        .description(bookDTO.getDescription())
                        .build();
                bookRepository.save(book);
                bookDTO.setBid(book.getBid());
                kafkaProducer.sendMessage(bookDTO);
            }
            catch(Exception e){
                System.out.println(e.getMessage());
            }
        }
    }
```

구현

- ❖ 카프카 연결
  - ✓ DataRead 프로젝트 수정
    - ● build.gradle 파일에 의존성 추가

```
implementation 'org.springframework.kafka:spring-kafka'
implementation 'org.json:json:20190722'
```

구현

- ❖ 카프카 연결
  - ✓ DataRead 프로젝트 수정
    - ● application.yml 파일에 카프카 설정 추가

```
spring:
    kafka:
      bootstrap-servers: 43.201.249.166:9092
      consumer:
        group-id: adamsoft
        auto-offset-reset: earliest
        key-deserializer:
org.apache.kafka.common.serialization.StringDeserializer
        value-deserializer:
org.apache.kafka.common.serialization.StringDeserializer
      producer:
        key-serializer: org.apache.kafka.common.serialization.StringSerializer
        value-serializer: org.apache.kafka.common.serialization.StringSerializer
```

구현

- ❖ 카프카 연결
  - ✓ DataRead 프로젝트 수정
    - ● 카프카 환경 설정 클래스 추가

```
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.StringSerializer;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.core.DefaultKafkaProducerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.core.ProducerFactory;

import java.util.HashMap;
import java.util.Map;
```

구현

- ❖ 카프카 연결
  - ✓ DataRead 프로젝트 수정
    - ● 카프카 환경 설정 클래스 추가

```
@Configuration
public class KafkaConfiguration {
    @Value("${spring.kafka.bootstrap-servers}")
    private String bootstrapServers;

    @Bean
    public ProducerFactory<String, String> producerFactory() {

        Map<String,Object> configs = new HashMap<>();
        configs.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
bootstrapServers);
        configs.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);
        configs.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);
        return new DefaultKafkaProducerFactory(configs);
    }
```

구현

- ❖ 카프카 연결
  - ✓ DataRead 프로젝트 수정
    - ● 카프카 환경 설정 클래스 추가

```
    @Bean
    public KafkaTemplate<String, String> kafkaTemplate() {
        return new KafkaTemplate<>(producerFactory());
    }
}
```

구현

- ❖ 카프카 연결
  - ✓ DataRead 프로젝트 수정
    - ● KafkaConsumer 클래스 추가

```
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import org.bson.Document;
import org.json.JSONObject;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Service;

import java.io.IOException;
```

구현

- ❖ 카프카 연결
  - ✓ DataRead 프로젝트 수정
    - ● KafkaConsumer 클래스 추가

```
@Service
public class KafkaConsumer {

    @KafkaListener(topics = "cqrs-topic", groupId = "adamsoft")
    public void consume(String message) throws IOException {
        System.out.println(message);
        JSONObject messageObj = new JSONObject(message);

        MongoClient mongoClient =
MongoClients.create("mongodb://adam:wnddkd@43.200.180.22:27017");
        MongoDatabase database = mongoClient.getDatabase("itstudy");
        MongoCollection<Document> mongo_books =
database.getCollection("books");
       mongo_books = database.getCollection("books");
```

구현

- ❖ 카프카 연결
  - ✓ DataRead 프로젝트 수정
    - ● KafkaConsumer 클래스 추가

```
            Document mongoBook = new Document();
            mongoBook.append("bid", messageObj.getLong("bid"));
            mongoBook.append("title", messageObj.getString("title"));
            mongoBook.append("author", messageObj.getString("author"));
            mongoBook.append("category", messageObj.getString("category"));
            mongoBook.append("pages", messageObj.getInt("pages"));
            mongoBook.append("price", messageObj.getInt("price"));
            mongoBook.append("published_date",
        messageObj.getString("published_date"));
            mongoBook.append("description", messageObj.getString("description"));
            mongo_books.insertOne(mongoBook);

            mongoClient.close();
        }
    }
```

구현

❖ 카프카 연결

✓ 클라이언트 프로젝트의 app.js 수정

```
import './App.css';
import {Paper} from "@material-ui/core"
import AddBook from "./AddBook";
import Axios from "axios";
import React, {useEffect, useState} from 'react';

function App() {
  const [items, setItems] = useState([]);
  const [data, setData] = useState(0);
```

구현

- 카프카 연결
  - ✓ 클라이언트 프로젝트의 app.js 수정

```
useEffect(() => {
    console.log("화면이 렌더링 된 이후에 바로 수행이 됨: componentDidMount()과 동일
");
    Axios.get("http://localhost:8000/cqrs/book").then((response) => {
        //console.log(response.data)
        if (response.data) {
          setItems(response.data)
        } else {
          alert("읽기 실패");
        }
      });
  }, [data]);
```

구현

- 카프카 연결
  - ✓ 클라이언트 프로젝트의 app.js 수정

```
//데이터 추가를 위한 함수
  const add = (book) => {
    console.log("book : ", book);
    Axios.post("http://localhost:7000/cqrs/book", book).then((response) => {
      console.log(response.data)
      if (response.data === "success") {
        alert("저장에 성공했습니다.")
        setData(1)
      } else {
        alert("코멘트를 저장하지 못했습니다.");
      }
    });
};
```

구현

- 카프카 연결
  - ✓ 클라이언트 프로젝트의 app.js 수정

```
    return (
      <div className="App">
        <Paper style={{ margin: 16 }}>
          <AddBook add = {add}/>
        </Paper>
          {items.map((item, index) => (
            <p key = {index}>
              {item.title}
            </p>
          ))}
      </div>
    );
  }
  export default App;
```