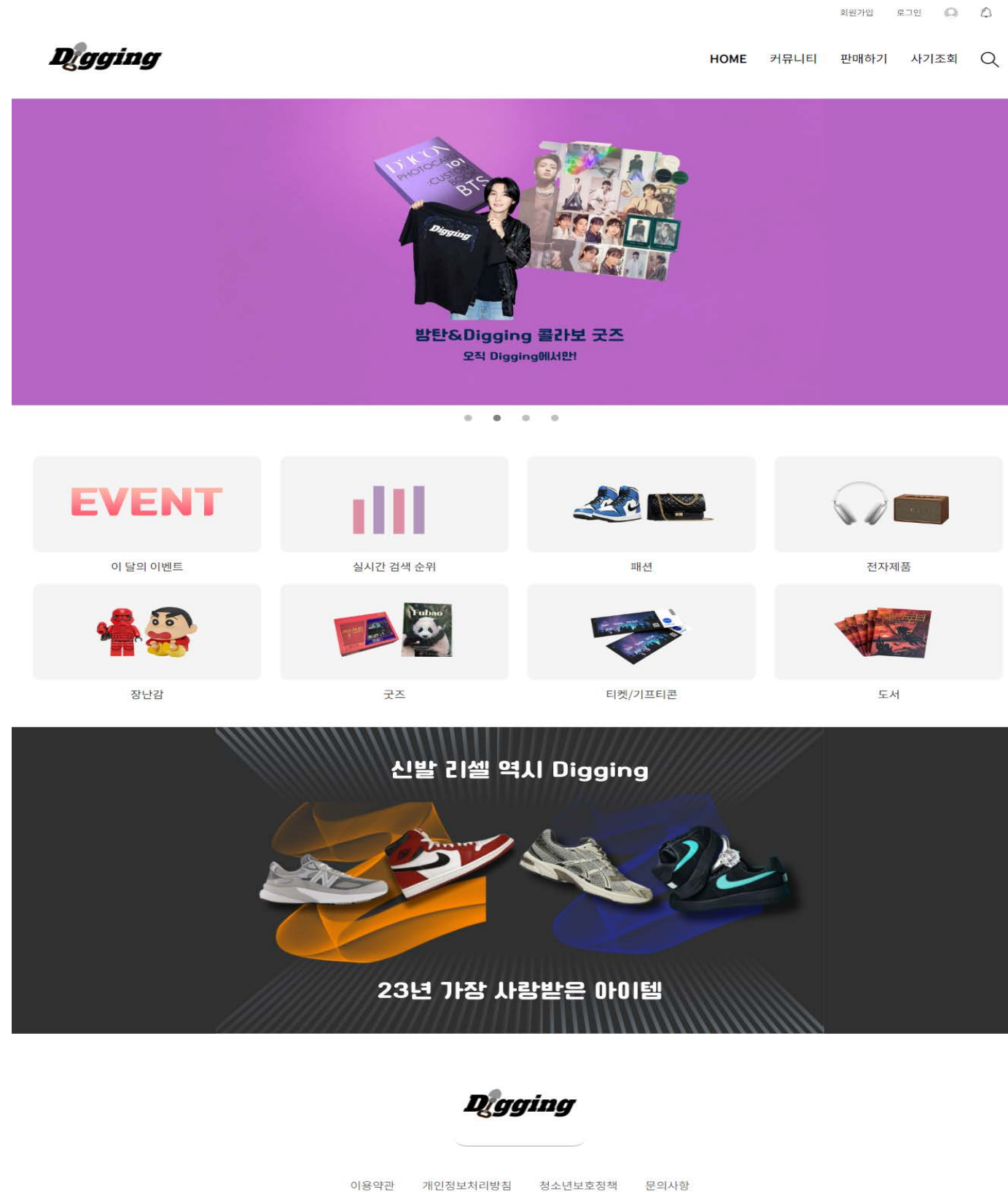


2024.10.29

Digging Project

C2C 거래 및 커뮤니티 사이트

Portfolio site



프로젝트

- 개인 포트폴리오 사이트

개발 인원 및 기간

- 개발 기간 : 2023.11.14 – 2023.12.21
- 개발 인원 : Front 3명 기획 2명

기술 스택

- React.js
- Node.js
- Python
- Cloudtype (배포)

Portfolio site



```
from flask import Flask, jsonify

app = Flask(__name__) #이 랩에 분석 결과를 넣어야 한다.
app.config['JSON_AS_ASCII'] = False # 한글 출력을 활거야

@app.route('/')
def root():
    res = "안녕~!" + "_!"
    return res # "안녕~!" #안녕 대신에 분석결과가 나와야한다.

@app.route('/keywords')
def keywords():
    res = "굿즈"
    return res

@app.route('/keywords/<option>')
def keywords_option(option):
    items = ["규어", "미쿠", "아이유"] # API 설계를 이렇게 할수 있
    res = items[int(option)]
    return res

def my_process(opt):
    items = ["규어", "미쿠", "아이유"]
    res = { "your_request" : opt, "my_response": items[int(opt)] }
    res = jsonify(res)
    return res

@app.route('/keywords_json/<option>') # json으로 출력하려면 impor
def keywords_json_option(option):
    n = int(option)
    res = get_recommendations(n) #int(option) # [('물건', 31), ('스
    res_dict = {}
    for i in range(n):
        res_dict[res[i][0]] = res[i][1]
    print("res_dict:", res_dict)
    return jsonify(res_dict)

app.run(host='localhost', port=5000)
```

프로젝트

- 개인 포트폴리오 사이트

개발 인원 및 기간

- 개발 기간 : 2023.11.14 – 2023.12.21
- 개발 인원 : Front 3명 기획 2명

기술 스택

- React.js
- Node.js
- Python
- Cloudtype (배포)

Portfolio site

회원가입

아이디

이메일

주소

상세주소

판교

× Q

대전광역시 동구 판교1길 ↖

대전광역시 동구 판교2길 ↖

대전광역시 동구 판교3길 ↖

경기도 안성시 판교길 ↖

경기도 평택시 판교길 ↖

경기도 성남시 분당구 판교대장로 ↖

경기도 성남시 분당구 판교동 ↖

경기도 성남시 분당구 판교로 ↖

비밀번호

비밀번호 확인

가입하기

이미 계정이 있으신가요? 로그인

Digging

로그인

아이디

비밀번호

로그인

회원가입

Home

프로젝트

- 개인 포트폴리오 사이트

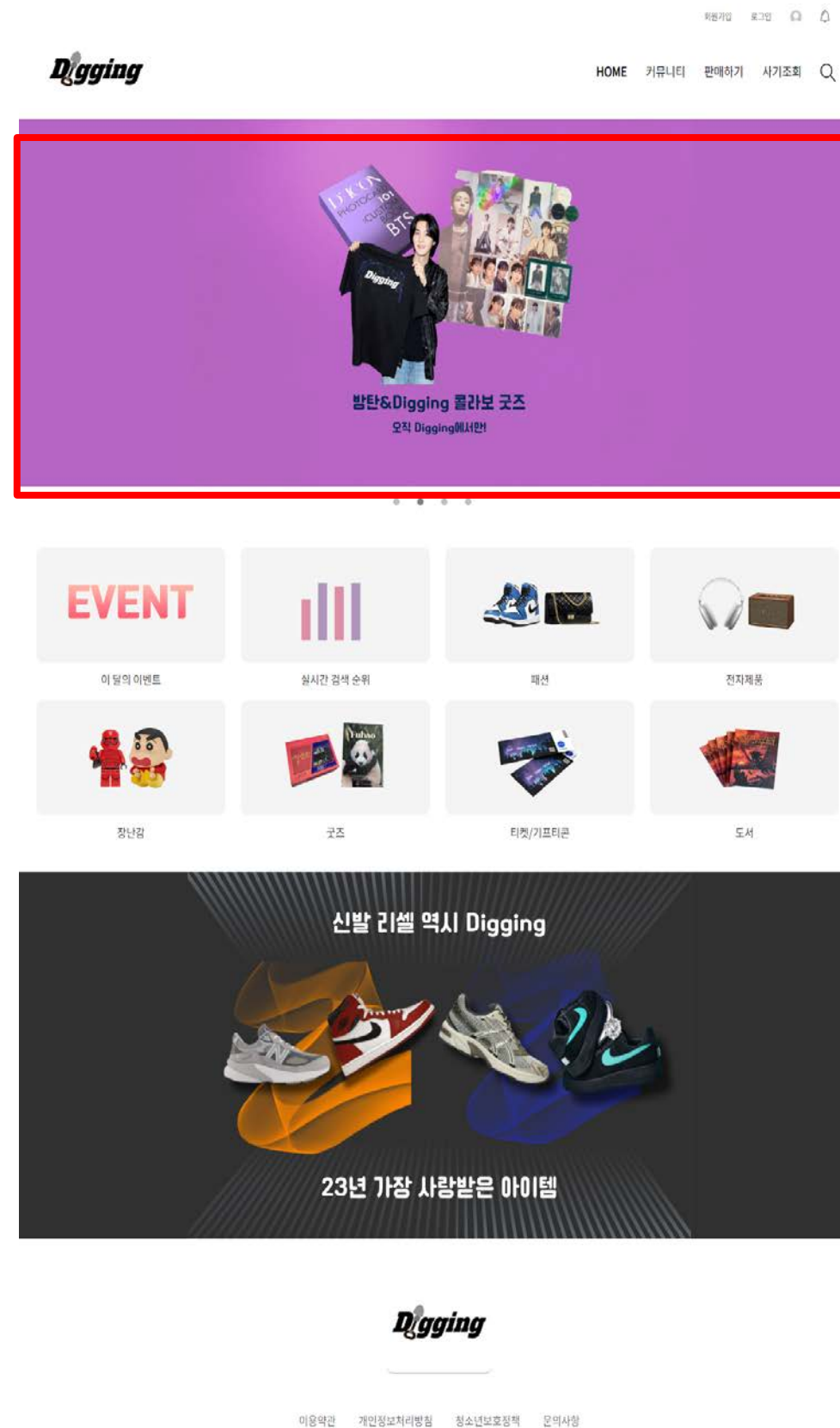
개발 인원 및 기간

- 개발 기간 : 2023.11.14 – 2023.12.21
- 개발 인원 : Front 3명 기획 2명

기술 스택

- React.js
- Node.js
- Python
- Cloudtype (배포)

Portfolio site



```
import React, { useState } from "react";
import { Carousel } from "react-bootstrap";
import "./slide.css";

function Slide({ interval }) {
  const [backgroundIndex, setBackgroundIndex] = useState(0);

  const handleSlideChange = (selectedIndex) => {
    // 슬라이드 변경 시 배경을 변경합니다.
    setBackgroundIndex(selectedIndex);
  };

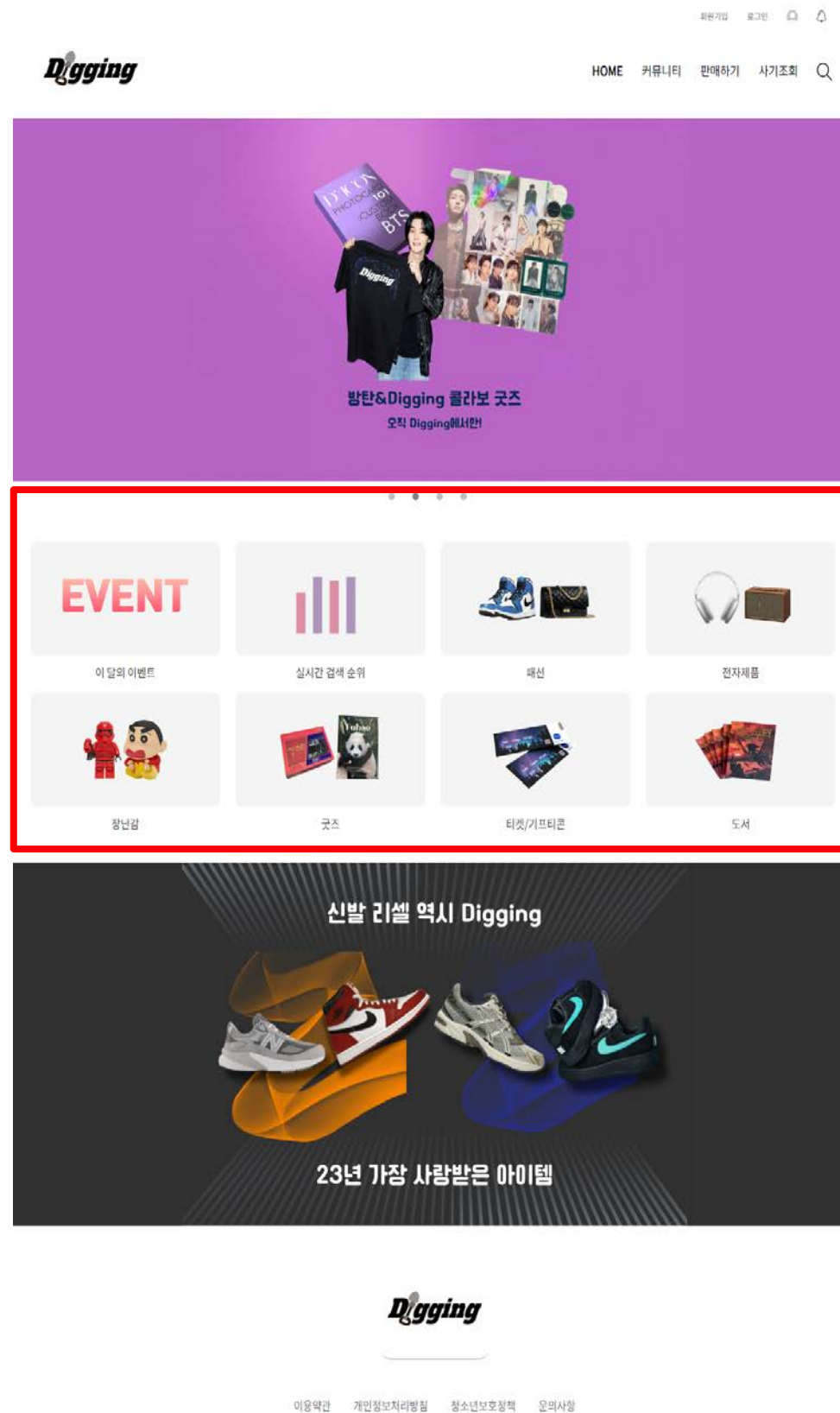
  return (
    <div className="box">
      <div className={`carousel-container background-${backgroundIndex}`}>
        <Carousel
          className="carousel"
          interval={interval}
          onSelect={handleSlideChange}
          slide={false}
        >
          <Carousel.Item>
            
          </Carousel.Item>
          <Carousel.Item>
            
          </Carousel.Item>
          <Carousel.Item>
            
          </Carousel.Item>
          <Carousel.Item>
            
          </Carousel.Item>
          <Carousel.Item>
            
          </Carousel.Item>
          <Carousel.Item>
            
          </Carousel.Item>
          <Carousel.Item>
            
          </Carousel.Item>
          <Carousel.Item>
            
          </Carousel.Item>
        </Carousel>
      </div>
    </div>
  );
}
```

상세 기여 내용

- Bootstrap을 이용하여 슬라이드가 변경 될때마다 배경색상이 변경되도록 구현
- useState훅을 사용하여 현재 슬라이드의 인덱스를 관리하여 backgroundIndex별로 background-color 변경

```
.background-0 {
  background-color: #e6e6e6;
}
.background-1 {
  background-color: #f0f0f0;
}
.background-2 {
  background-color: #d9d9d9;
}
.background-3 {
  background-color: #c0c0c0;
}
/* 1st background color: rgb(243, 234, 222); */
```


Portfolio site



상세 기여 내용

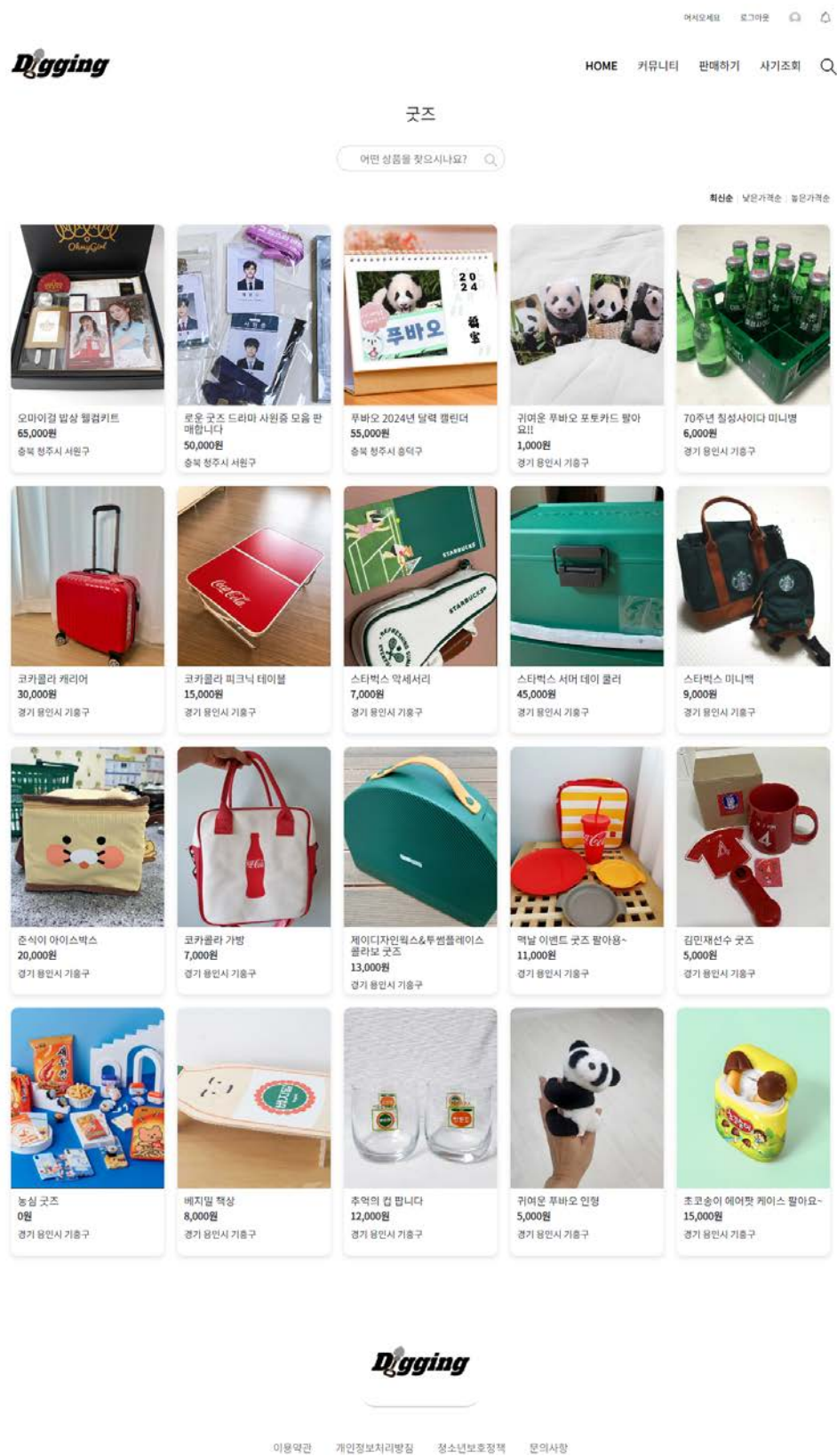
- 외부 데이터를 useState 혹은 사용하여 관리하고, 이를 기반으로 카드를 동적으로 생성
- PopularContent 라는 하위 컴포넌트를 생성하여 코드의 재사용성과 가독성을 높임

```
});
console.log("popularData", popularData);
return (
  <div className="poular-content">
    <div className="content-container">
      {popularData.map((item) => (
        <div className="card-container" key={item.id}>
          <PopularContent
            key={item.id}
            popularData={item}
            categoryUrl={categoryUrl[item.id]}
          />
        </div>
      ))}
    </div>
  </div>
);
}

function PopularContent(props) {
  return (
    <div className="card-box">
      <a href={props.categoryUrl} className="card-link">
        <img
          className="card-image"
          src={props.popularData.img}
          alt="카드 이미지"
        />
      </a>
      <div className="card-content">
        <p>{props.popularData.title}</p>
      </div>
    </div>
  );
}

export default Content;
```

Portfolio site



Digging



manager
서울 강남구 가로수길
34일 전

켄틀몬스터 선글라스 팔아요
2,500,000원

켄틀몬스터 선글라스 팔아요 사용감은 없고 새제품입니다.

댓글 0개

내용을 입력해주세요

댓글작성

Digging

상세 기여 내용

- JWT 토큰을 가져와 사용자 정보를 디코드하여 로그인한 사용자에게 대한 정보를 관리하고, Content Card의 사용자 주소 등록 구현
- 판매 목록 및 상세 디테일 CSS 구현

```
useEffect(() => {  
  const token = localStorage.getItem("token");  
  
  if (token) {  
    // 토큰이 존재할 경우 디코드하여 사용자 정보 설정  
    const decoded = jwt_decode(token);  
    setUserInfo(decoded);  
  }  
}, []);
```

```
function TabContent(props, i) {  
  const addressParts = (props.contentData.address || "").split(" ");  
  const processedAddress = addressParts.slice(0, 3).join(" ");  
  
  return (  
    <div className="categoryContents-item">  
      <div className="categoryContents-img-box">  
        <img src={props.contentData.image} alt="contentDataImage" />  
      </div>  
      <div className="text-content">  
        <p className="categoryContents-card-title">{props.contentData.title}</p>  
        <p className="categoryContents-card-price">  
          {Number(props.contentData.price).toLocaleString()}원  
        </p>  
        <p className="categoryContents-card-address">{processedAddress}</p>  
      </div>  
    </div>  
  );  
}  
  
export default ContentItem;
```

Portfolio site

회원가입로그인

Digging

HOME커뮤니티판매하기사기조회

태그

태그를 입력하세요

제목

제목을 입력해주세요

내용

내용을 입력해주세요

가격

₩ 가격을 입력해주세요

파일 선택

선택된 파일 없음

작성하기

Digging

이용약관개인정보처리방침청소년보호정책문의사항

```
import { useNavigate } from "react-router-dom";
import Footer from "../../components/footer/footer";
import { useState, useEffect } from "react";
import ToolBar from "../../components/toolbar/toolbar";
import Select from "react-select";
import makeAnimated from "react-select/animated";
import "./ContentRegister.css";
import jwt_decode from "jwt-decode";
import apiUrl from "../../config";
```

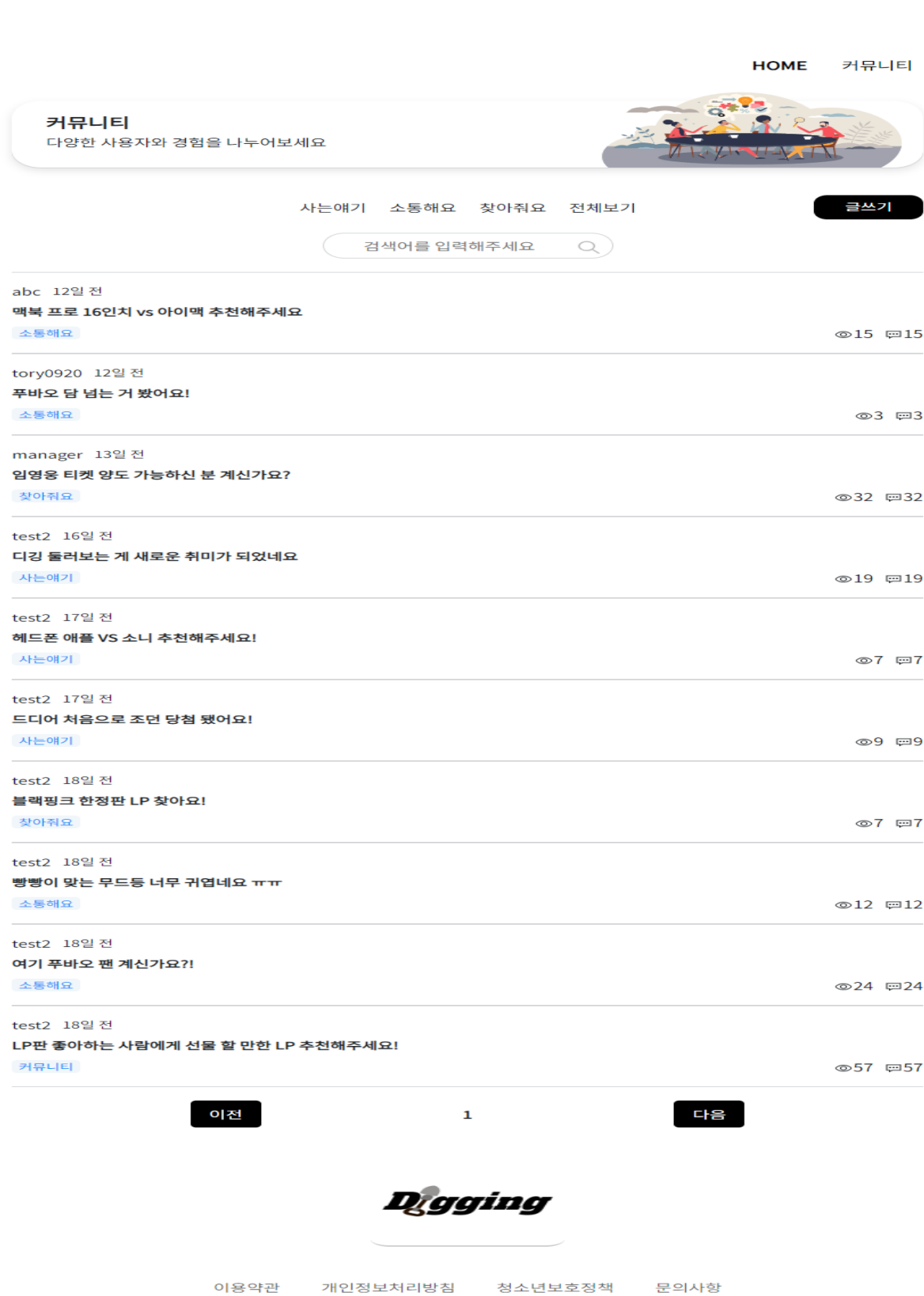
```
function ContentRegister(props) {
  const [userInfo, setUserInfo] = useState(null);
  const [selectedTag, setSelectedTag] = useState(null);
  const navigate = useNavigate();
  const tagOptions = [
    { value: "fashion", label: "패션" },
    { value: "electronic", label: "전자제품" },
    { value: "toy", label: "장난감" },
    { value: "goods", label: "굿즈" },
    { value: "ticket", label: "티켓/기프트콘" },
    { value: "book", label: "도서" },
  ];
}
```

상세 기여 내용

- React-select 라이브러리를 사용하여 태그 선택 기능 구현하였고, 태그 옵션 배열을 정의해 사용자가 선택할 수 있는 태그로 구성

```
>
<label>
  태그
  <Select
    className="contentRegister-custom-select"
    components={animatedComponents}
    isMulti={false}
    theme={({theme}) => ({
      ...theme,
      borderRadius: 4,
      colors: {
        ...theme.colors,
        primary25: "primary25",
        primary: "black",
      },
    })}
    options={tagOptions}
    value={selectedTag}
    onChange={handleTagChange}
    placeholder="태그를 입력하세요"
  />
</label>
<div id="contentRegister_titleInput">
```


Portfolio site



상세 기여 내용

- useState혹을 사용하여 게시판 데이터, 페이지네이션, 검색어, 등을 관리하는 로컬 상태 관리 구현
- Fetch API를 활용하여 서버 엔드포인트에서 게시판 데이터를 비동기적으로 가져오도록 구현 이 과정에서 검색 기능을 포함
- 사용자가 특정 태그에 따라 게시글을 조회할 수 있는 필터링 구현
- 사용자가 키워드를 기반으로 게시글을 검색할 수 있는 검색 입력 기능 구현

Portfolio site

Client

```
function Board() {
  const [boardData, setBoardData] = useState([]);
  const [page, setPage] = useState(1);
  const navigate = useNavigate();
  const token = localStorage.getItem("token");
  const [searchTerm, setSearchTerm] = useState(""); // 검색어 상태 추가
  const [tag, setTag] = useState(""); // 검색어 상태 추가

  const fetchData = async (pageNumber, selectedTag) => {
    try {
      const response = await fetch(
        `${apiUrl}/board?page=${pageNumber}&tag=${selectedTag}`
      );
      if (!response.ok) {
        throw new Error("서버 응답 에러");
      }

      const data = await response.json();
      console.log("data", data);
      let sortedData = data.result.sort(
        (a, b) => new Date(b.date) - new Date(a.date)
      );

      setBoardData(sortedData);
    } catch (error) {
      console.error("데이터를 가져오는 중 에러 발생:", error);
    }
  };
};
```

Server

```
app.get("/board", async (req, res) => {
  try {
    const page = parseInt(req.query.page) || 1;
    const pageSize = 10;
    const skip = (page - 1) * pageSize;
    const { searchTerm, tag } = req.query;
    console.log(tag);
    console.log(searchTerm);
    // Build the query based on whether a search term is provided
    let query = {};
    if (searchTerm) {
      query.title = { $regex: new RegExp(searchTerm, "i") };
    }

    console.log(tag);
    if (tag) {
      query.tag = tag;
    }

    const result = await db
      .collection("board")
      .find(query)
      .skip(skip)
      .limit(pageSize)
      .toArray();

    res.json({ result });
  } catch (error) {
    console.error("게시판 데이터를 가져오는 중 에러 발생:", error);
    res.status(500).json({
      message: "게시판 데이터를 가져오는 중 에러가 발생했습니다.",
    });
  }
});
```

Client

1. Fetch API를 사용하여 서버의 /board 엔드포인트에 GET요청을 보내며 요청은 페이지 번호와 선택된 태그를 쿼리 파라미터로 포함
2. 응답이 성공적이면 JSON으로 변환 후 게시글 데이터를 날짜 기준으로 정렬하여 상태저장

Server

1. 게시글 목록을 가져오는 GET요청을 처리, 쿼리 파라미터로 page와 tag를 받아 게시글을 필터링 하고 정렬하여 클라이언트에 응답

Portfolio site

Client

```
const handleSearchButtonClick = async () => {
  try {
    const response = await fetch(`${apiUrl}/board?searchTerm=${searchTerm}`);
    if (!response.ok) {
      throw new Error("서버 응답 에러");
    }

    const data = await response.json();
    setBoardData(data.result);
  } catch (error) {
    console.error("데이터를 가져오는 중 에러 발생:", error);
  }
};
```

Server

```
app.get("/board", async (req, res) => {
  try {
    const page = parseInt(req.query.page) || 1;
    const pageSize = 10;
    const skip = (page - 1) * pageSize;
    const { searchTerm, tag } = req.query;
    console.log(tag);
    console.log(searchTerm);
    // Build the query based on whether a search term is provided
    let query = {};
    if (searchTerm) {
      query.title = { $regex: new RegExp(searchTerm, "i") };
    }

    console.log(tag);
    if (tag) {
      query.tag = tag;
    }

    const result = await db
      .collection("board")
      .find(query)
      .skip(skip)
      .limit(pageSize)
      .toArray();

    res.json({ result });
  } catch (error) {
    console.error("게시판 데이터를 가져오는 중 에러 발생:", error);
    res.status(500).json({
      message: "게시판 데이터를 가져오는 중 에러가 발생했습니다.",
    });
  }
});
```

Client



1. 사용자가 입력한 검색어를 쿼리 파라미터로 포함하여 /board API 호출
2. 응답이 성공적일때 JSON형태로 변환하여 boardData 상태 업데이트

Server


1. 쿼리 파라미터에서 searchTerm을 가지고와 사용자가 입력한 검색어가 있을 경우 정규표현식을 사용한 쿼리 조건을 설정

Portfolio site

1234년 로그인 

 HOME 커뮤니티 판매하기 사기조회 

태그

태그를 선택하세요 


제목

제목을 입력해주세요

내용

내용을 입력해주세요

글쓰기



```
const handleSubmit = async (e) => {
  e.preventDefault();
  console.log(
    "Received views:",
    boardData.views,
    "Type:",
    typeof boardData.views
  );

  const numericViews = parseInt(boardData.views);
  try {
    const res = await fetch(`${apiUrl}/board`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ ...boardData, views: numericViews }),
    });

    if (res.ok) {
      console.log("글 전송 ok.");

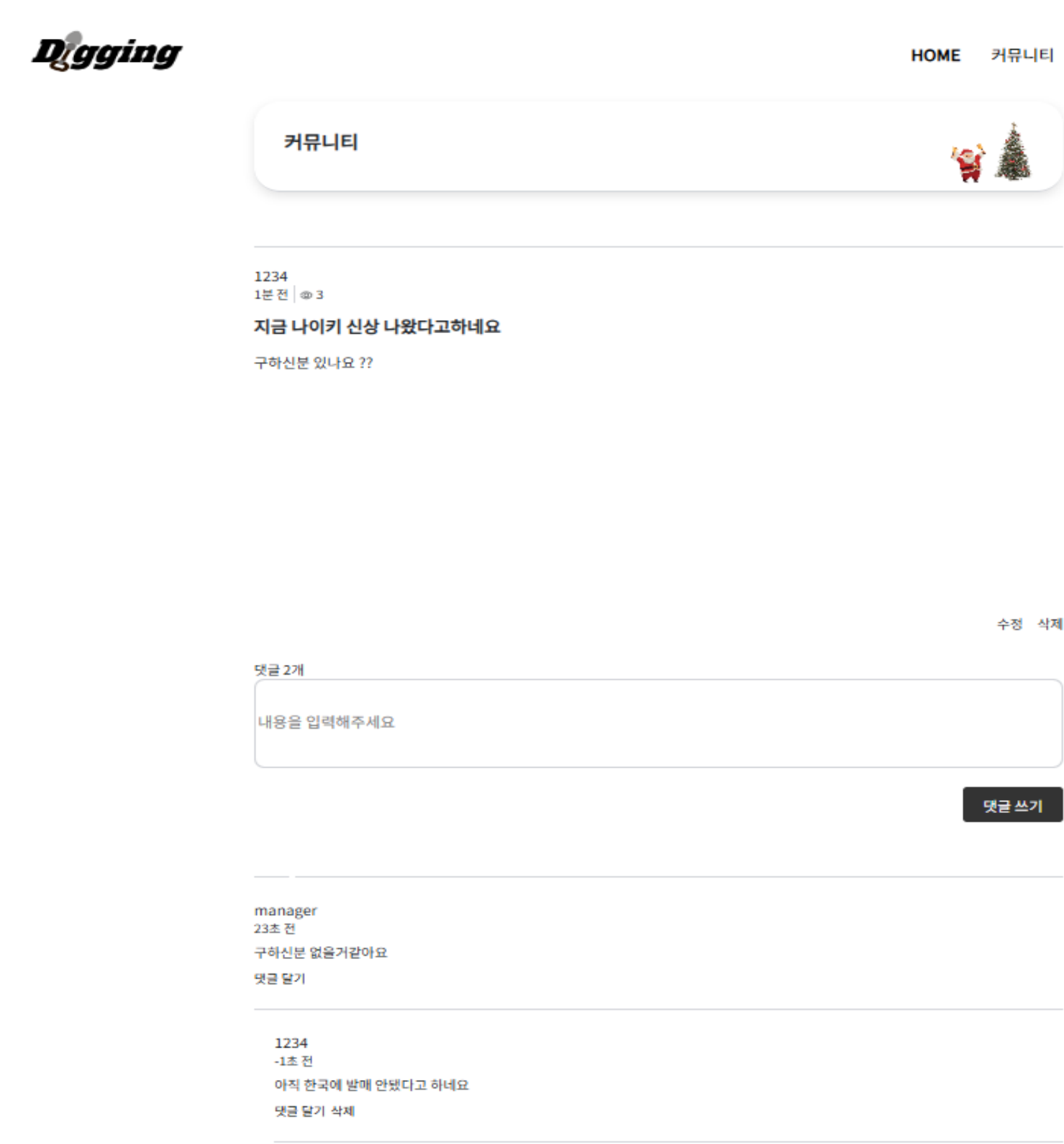
      if (onPostSubmit) {
        onPostSubmit();
      }
      navigate("/board");
    } else {
      console.error("글 전송 실패:", res.status);
    }
  } catch (error) {
    console.error("오류 발생:", error.message);
  }

  // 폼 초기화
  setBoardData({
    id: "",
    number: "",
    title: "",
    content: "",
    writer: "",
    views: "0",
    date: "",
  });
};
```

상세 기여 내용

- 사용자의 JWT 토큰을 통하여 게시글 작성에 필요한 기본 정보를 가져오고, useState를 활용하여 게시글 데이터의 상태를 관리
- React-select 라이브러리를 사용하여 태그 선택 UI 구현 및 사용자가 선택한 태그를 상태에 반영하는 기능 구현
- 사용자가 작성한 게시글 데이터를 서버에 POST 요청으로 전송하는 기능 구현, 전송 성공 시 게시판 페이지로 리디렉션하는 로직 추가

Portfolio site



상세 기여 내용

- useParms 혹은 사용하여 URL에서 postId를 추출하고, 이 postId를 사용하여 해당 게시글의 세부 정보 가져와 게시글 상세 정보 조회 기능
- fetchPostDetail 함수에서 API를 호출하여 게시글과 관련된 댓글을 가져오며, 댓글 및 삭제 기능 구현
- isAuthor 변수를 사용하여 게시글 작성자 및 관리자를 식별하여 게시글 수정 및 삭제 기능 구현
- 게시글 작성 일자 및 조회수 기능 구현

Portfolio site

Client

```
const fetchPostDetail = async () => {
  if (token) {
    // 토큰이 존재할 경우 디코딩하여 사용자 정보 설정
    const decoded = jwt_decode(token);
    setUserInfo(decoded);
  }
  // get 요청 해서 받아오는 부분
  try {
    const response = await fetch(`${apiUrl}/board_detail/${postId}`);
    if (response.ok) {
      const data = await response.json();
      setComments(data.comments);
      setPostDetail(data.post);
    } else {
      const errorMessage = await response.text();
      console.error(
        `디테일 에러. Status: ${response.status}, Message: ${errorMessage}`
      );
    }
  } catch (error) {
    console.error("Error fetching post detail:", error.message);
  }
};

const handleCommentSubmit = async () => {
  fetchPostDetail();
};

useEffect(() => {
  fetchPostDetail();
}, [postId]);

const postDate = postDetail ? postDetail.date : null;
const formattedTime = postDate ? formatTimeAgo(postDate) : null;
```

Server

```
app.get("/board_detail/:postId", async (req, res) => {
  const postId = req.params.postId;
  try {
    // 클라이언트에서 전달한 postId를 사용하여 해당 게시물을 찾을
    const post = await db
      .collection("board")
      .findOne({ _id: new ObjectId(postId) });

    const comments = await db
      .collection("comment123")
      .find({ postId: postId })
      .toArray();

    if (!post) {
      return res.status(404).json({ message: "Post not found" });
    }

    // 게시물과 댓글을 하나의 응답 객체로 합침
    const responseData = {
      post: post,
      comments: comments,
    };

    res.json(responseData);
  } catch (error) {
    console.error("Error fetching post detail:", error);
    res.status(500).json({ message: "Internal server error" });
  }
});
```

Client

1. 토큰이 있는경우 디코딩 하여 사용자 정보를 설정
2. fetch로 서버에 GET요청을 보내 postId에 해당하는 게시물과 댓글을 불러온다
3. 응답 상태를 확인하고 정상이면 데이터를 파싱하여 data에 저장

Server

1. Client에서 전달한 postId를 사용하여 MongoDB에서 특정 게시물과 댓글들을 가져와 하나의 JSON객체로 클라이언트에 반환
2. 게시물이나 댓글을 찾지 못한 경우 오류 전송

Portfolio site

Client

```
const handleDelete = async () => {
  try {
    console.log("handle-postId", postId);
    const response = await fetch(`${apiUrl}/board_detail/${postId}`, {
      method: "DELETE",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        action: "deletePost",
      }),
    });

    if (response.ok) {
      navigate("/board");
    } else {
      const errorMessage = await response.text();
      console.error(
        `삭제하지 못했습니다. Status: ${response.status}, Message: ${errorMessage}`
      );
    }
  } catch (error) {
    console.error("삭제 에러 post:", error.message);
  }
};
```

Server

```
app.delete("/board_detail/:postId", async (req, res) => {
  const { postId } = req.params;
  const { action, commentId } = req.body;

  try {
    if (action === "deleteComment") {
      console.log("test");
      // 댓글 삭제 처리
      const result = await db.collection("comment123").deleteOne({
        _id: new ObjectId(commentId),
        postId: postId,
      });

      if (result.deletedCount === 1) {
        res.json({ message: "댓글 삭제 성공" });
      } else {
        res.status(404).json({ error: "댓글을 찾을 수 없습니다." });
      }
    } else if (action === "deletePost") {
      // 게시물 삭제 처리
      const result = await db.collection("board").deleteOne({
        _id: new ObjectId(postId),
      });

      if (result.deletedCount === 1) {
        res.json({ message: "게시물 삭제 성공" });
      } else {
        res.status(404).json({ error: "게시물을 찾을 수 없습니다." });
      }
    } else {
      res.status(400).json({ error: "잘못된 요청입니다." });
    }
  } catch (error) {
    console.error("삭제 에러:", error.message);
    res.status(500).json({ error: "서버 에러" });
  }
});
```

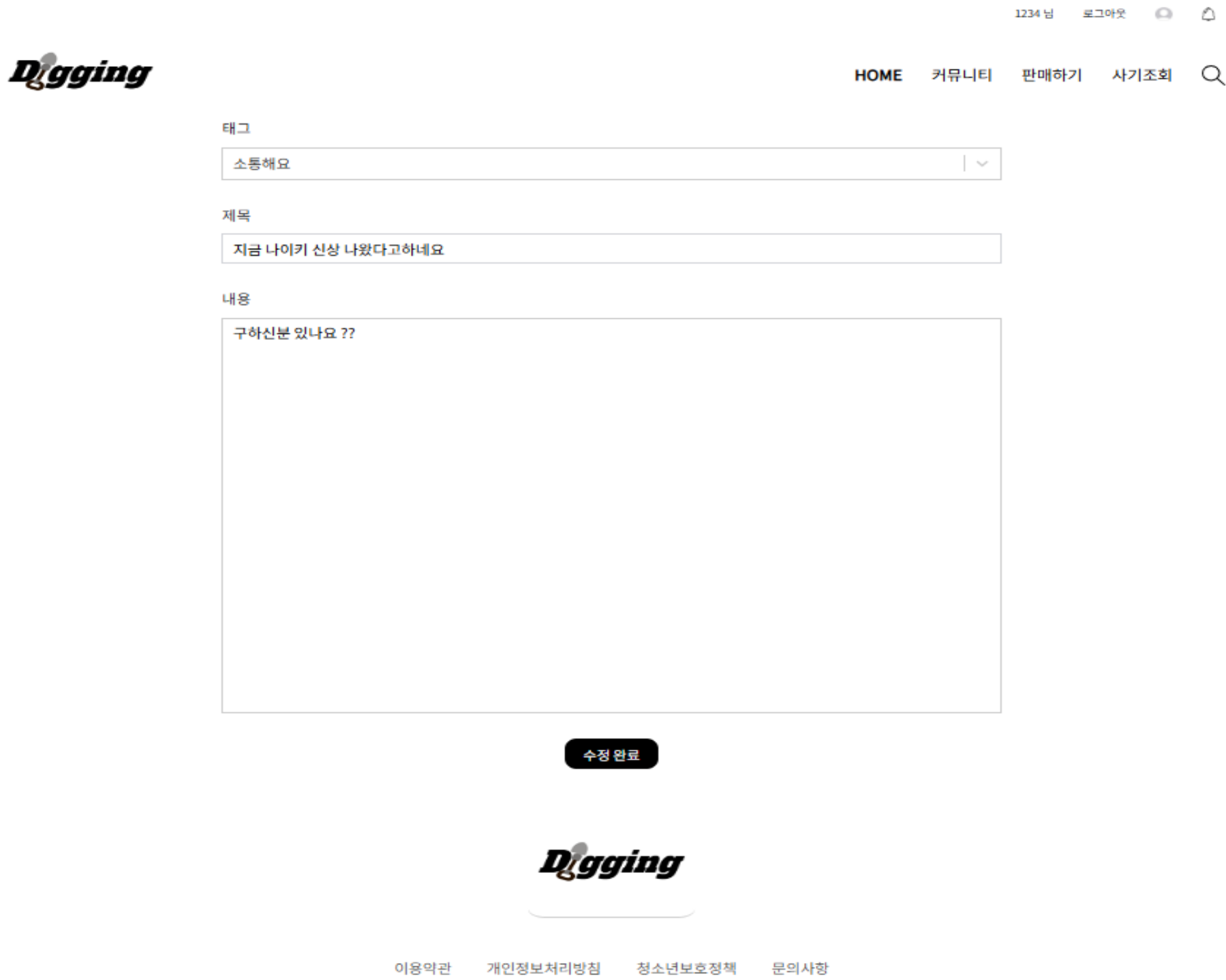
Client

1. Board_detail/\${postId} 경로에 DELETE 요청
2. HTTP 메소드 요청 타입 설정
3. Response.ok의 상태에 따라 성공 실패 여부 판단

Server

1. URL 파라미터에서 postId를 추출하여 요청된 게시물이나 댓글과 연결
2. action이 "deleteComment"인 경우 commentId와 postId 기준으로 댓글을 삭제
3. Action이 "deletePost"인 경우 게시물을 삭제

Portfolio site



상세 기여 내용

- useParms 혹은 사용하여 URL에서 postId를 추출하고, 이 postId를 사용하여 해당 게시글의 세부 정보 가져와 상세 정보 조회
- HTTP 메서드를 이용하여 게시글 수정

Portfolio site

Client

```
useEffect(() => {
  const fetchPostDetail = async () => {
    try {
      const response = await fetch(`${apiUrl}/board_detail/${postId}`);
      if (response.ok) {
        const data = await response.json();

        // 받아온 tag를 tagOptions에서 매칭되는 객체로 찾을
        const matchedTag = tagOptions.find(
          (tag) => tag.value === data.post.tag
        );

        setPostDetail({
          title: data.post.title,
          content: data.post.content,
          tag: matchedTag, // 매칭된 태그 객체를 설정
        });
      } else {
        const errorMessage = await response.text();
        console.error(
          `게시글 데이터를 불러오지 못했습니다. Status: ${response.status}, Message: ${errorMessage}`
        );
      }
    } catch (error) {
      console.error("Error fetching post detail:", error.message);
    }
  };

  fetchPostDetail();
}, [postId]);
```

Server

```
app.get("/board_detail/:postId", async (req, res) => {
  const postId = req.params.postId;
  try {
    // 클라이언트에서 전달한 postId를 사용하여 해당 게시물을 찾을
    const post = await db
      .collection("board")
      .findOne({ _id: new ObjectId(postId) });

    const comments = await db
      .collection("comment123")
      .find({ postId: postId })
      .toArray();

    if (!post) {
      return res.status(404).json({ message: "Post not found" });
    }

    // 게시물과 댓글을 하나의 응답 객체로 합침
    const responseData = {
      post: post,
      comments: comments,
    };

    res.json(responseData);
  } catch (error) {
    console.error("Error fetching post detail:", error);
    res.status(500).json({ message: "Internal server error" });
  }
});
```

Client

1. fetch로 서버에 GET요청을 보내 postId에 해당하는 게시글 정보를 불러온다
2. response.ok의 상태에 따라 성공 실패 여부 판단

Server

1. URL 파라미터에서 postId를 추출하여 요청된 게시물을 가져와 하나의 JSON객체로 클라이언트에 반환
2. 게시글을 찾지 못한 경우 오류 전송

Portfolio site

Client

```
const handleFormSubmit = async (e) => {
  e.preventDefault();

  try {
    const response = await fetch(`${apiUrl}/board_edit/${postId}`, {
      method: "PUT",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        title: postDetail.title,
        content: postDetail.content,
        tag: postDetail.tag?.value, // 태그의 value를 전송
      }),
    });

    if (response.ok) {
      // 수정이 성공하면 상세 페이지로 이동
      navigate(`/board_detail/${postId}`);
    } else {
      const errorMessage = await response.text();
      console.error(
        `수정 실패. Status: ${response.status}, Message: ${errorMessage}`
      );
    }
  } catch (error) {
    console.error("Error editing post:", error.message);
  }
};
```

Server

```
app.put("/board_edit/:postId", async (req, res) => {
  const { postId } = req.params;
  const updatedData = req.body;
  console.log("postId", postId);
  console.log("updatedData", updatedData);
  try {
    await db.collection("board").updateOne(
      { _id: new ObjectId(postId) },
      {
        $set: {
          title: updatedData.title,
          content: updatedData.content,
          tag: updatedData.tag,
        },
      }
    );

    res.json({ message: "ok" });
  } catch (error) {
    console.error("수정 에러:", error.message);
    res.status(500).json({ error: "서버 에러" });
  }
});
```

Client

1. fetch로 서버에 PUT요청을 보내 postId에 해당하는 게시글 정보에 수정 요청
2. response.ok의 상태에 따라 성공 실패 여부 판단

Server

1. URL 파라미터에서 postId를 추출하여 요청된 특정 게시글을 식별
2. MongoDB의 board 컬렉션에서 특정 게시글을 업데이트
3. 게시글을 찾지 못한 경우 오류 전송

Portfolio site

개선 점

[로딩 상태 관리]

데이터를 불러오는 동안 로딩 스피너나 상태 메시지를 보여주어 사용자 경험을 향상한다.

[상태 관리 개선]

상태 변수를 여러 번 수정하는 방식이 사용 되는 경우가 있는데 상태를 단일 함수로 묶어 처리하여 가독성을 높인다

[상태 코드 및 응답 메시지 개선]

다양한 종류의 오류를 구분할 수 있도록 세분화된 상태 코드를 반환한다.

500 – 데이터베이스 연결오류

400 – 클라이언트의 요청한 자원이 존재하지 않을 경우

[유효성 검사]

사용자가 입력한 데이터에 대한 유효성 검사를 추가하여, 빈 제목이나 내용이 제출되지 않도록 작성

[코드 스타일 가이드 수립]

팀 전체가 동일한 환경에서 작업하도록 프로젝트에 대한 공식적인 코드 스타일 가이드를 수립한다.

Portfolio site

C2C 중고거래 및 커뮤니티 사이트