26. FE 포스트조회기능 구현하기

26.1 포스트 읽기 페이지 구현하기

26.1.1 PostViewer UI 준비

components\post\PostViewer.js

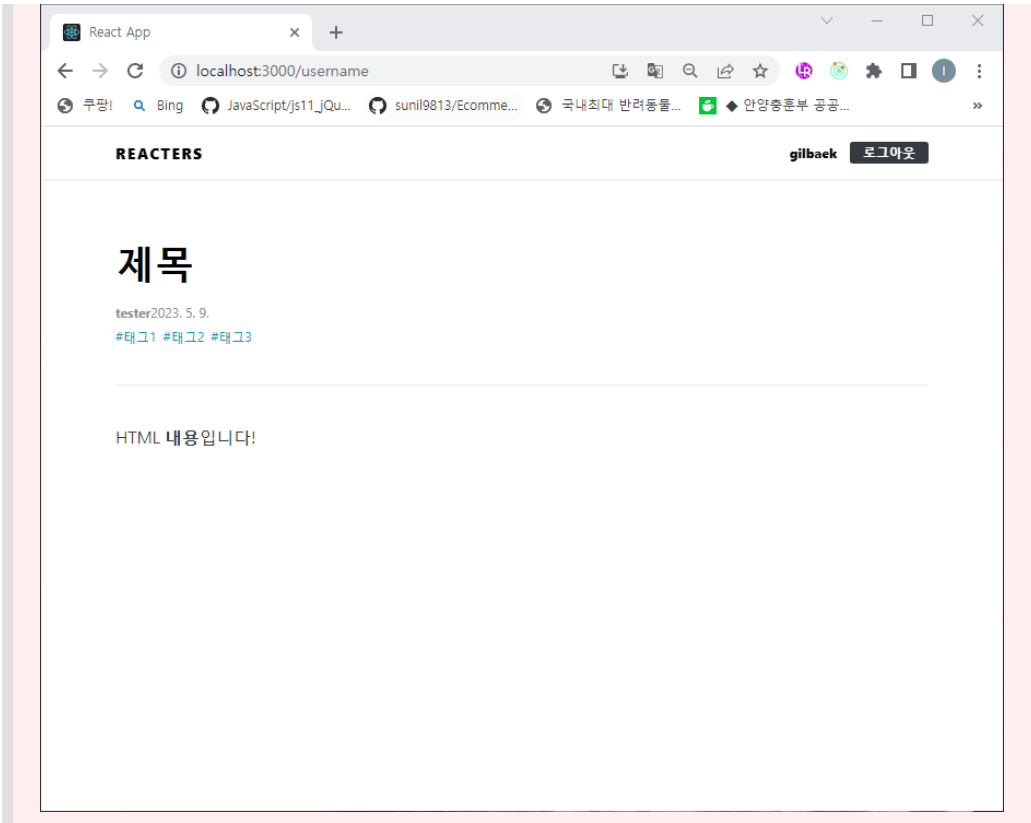
```
import styled from 'styled-components';
import palette from '../../lib/styles/palette';
import Responsive from '../common/Responsive';


const PostViewerBlock = styled(Responsive)`
  margin-top: 4rem;
`;

const PostHead = styled.div`
  border-bottom: 1px solid ${palette.gray[2]};
  padding-bottom: 3rem;
  margin-bottom: 3rem;
  h1 {
    font-size: 3rem;
    line-height: 1.5;
    margin: 0;
  }
`;

const SubInfo = styled.div`
  margin-top: 1rem;
  color: ${palette.gray[6]};

  // span 사이에 가운데점 문자 보여주기
  span + span:before {
    color: ${palette.gray[5]};
    padding-left: 0.25rem;
    padding-right: 0.25rem;
    content: '\\B7'; // 가운뎃점 점 문자
  }
`;

const Tags = styled.div`
  margin-top: 0.5rem;
  .tag {
    display: inline-block;
    color: ${palette.cyan[7]};
    text-decoration: none;
    margin-right: 0.5rem;
    &:hover {
      color: ${palette.cyan[6]};
    }
  }
`;
```

```
const PostContent = styled.div`
  font-size: 1.3125rem;
  color: ${palette.gray[8]};
`;

const PostViewer = () => {

  return (
    <PostViewerBlock>
      <PostHead>
        <h1>제목</h1>
        <SubInfo>
          <span><b>tester</b></span>
          <spa>{new Date().toLocaleDateString()}</spa>
        </SubInfo>
        <Tags>
          <div className='tag'>#태그1</div>
          <div className='tag'>#태그2</div>
          <div className='tag'>#태그3</div>
        </Tags>
      </PostHead>
      <PostContent dangerouslySetInnerHTML={{ __html: '<p>HTML <b>내용</b>입니
다!</p>' }} />
    </PostViewerBlock>
  );
};

export default PostViewer;
```

pages\PostPage.js

```
import HeaderContainer from '../containers/common/HeaderContainer';
import PostViewer from '../components/post/PostViewer';

const PostPage = () => {
  return (
    <>
      <HeaderContainer />
      <PostViewer />
    </>
  );
};

export default PostPage;
```

- http://localhost:3000/@testet/samplepostid
  - 현재까지 `no routes matched location "/@tester/sampleid"` 로 작동 않되어서
  - http://localhost:3000/username으로 테스트



26.1.2 API연동하기

lib/api/posts.js

```
import client from './client';

export const writePost = ({ title, body, tags }) =>
  client.post('/api/posts', { title, body, tags });

export const readPost = id => client.get(`/api/posts/${id}`);
```

modules/post.js - redux 모듈

- READ_POST액션외에 UNLOAD_POST 액션을 추가, UNLOAD_POST의 용도는 포스트페이지를 벗어날 때 리덕스상태 데이터를 비우는 역할
- 페이지를 벗어날 때 데이터를 비우지 않을 경우 다른 포스트를 읽을 때 이전 포스트가 나타나는 깜박임현상이 발생한다.

```
import { createAction, handleActions } from 'redux-actions';
import createRequestSaga, {
  createRequestActionTypes,
} from '../lib/createRequestSaga';
import * as postsAPI from '../lib/api/posts';
import { takeLatest } from 'redux-saga/effects';

const [
  READ_POST,
  READ_POST_SUCCESS,
  READ_POST_FAILURE,
] = createRequestActionTypes('post/READ_POST');
const UNLOAD_POST = 'post/UNLOAD_POST'; // 포스트 페이지에서 벗어날 때 데이터 비우기
```

```
export const readPost = createAction(READ_POST, id => id);
export const unloadPost = createAction(UNLOAD_POST);

const readPostSaga = createRequestSaga(READ_POST, postsAPI.readPost);
export function* postSaga() {
  yield takeLatest(READ_POST, readPostSaga);
}

const initialState = {
  post: null,
  error: null,
};

const post = handleActions(
  {
    [READ_POST_SUCCESS]: (state, { payload: post }) => ({
      ...state,
      post,
    }),
    [READ_POST_FAILURE]: (state, { payload: error }) => ({
      ...state,
      error,
    }),
    [UNLOAD_POST]: () => initialState,
  },
  initialState,
);

export default post;
```

modules/index.js - post리덕스를 루트리듀서와 사가리듀서에 등록

```
import { combineReducers } from 'redux';
import { all } from 'redux-saga/effects';
import auth, { authSaga } from './auth';
import loading from './loading';
import user, { userSaga } from './user';
import write, { writeSaga } from './write';
import post, { postSaga } from './post';

const rootReducer = combineReducers({
  auth,
  loading,
  user,
  write,
  post
});

export function* rootSaga() {
  yield all([authSaga(), userSaga(), writeSaga(), postSaga() ]);
}

export default rootReducer;
```

- PostViwer를 위한 컨테이너컴퍼넌트 만들기

containers\post\PostViewerContainer.js

- url의 id값을 조회해야 하기 때문에 useParams를 함께 사용

```
import React, { useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { useParams } from 'react-router-dom';
import { readPost, unloadPost } from '../../modules/post';
import PostViewer from '../../components/post/PostViewer';

const PostViewerContainer = () => {
  // 처음 마운트될 때 포스트 읽기 API 요청
  const { postId } = useParams();
  const dispatch = useDispatch();
  const { post, error, loading } = useSelector(({ post, loading }) => ({
    post: post.post,
    error: post.error,
    loading: loading['post/READ_POST'],
  }));

  useEffect(() => {
    dispatch(readPost(postId));
    // 언마운트될 때 리덕스에서 포스트 데이터 없애기
    return () => {
      dispatch(unloadPost());
    };
  }, [dispatch, postId]);

  return <PostViewer post={post} loading={loading} error={error} />;
};

export default PostViewerContainer;
```

pages/PostPage.js - PostViewerContainer로 대체

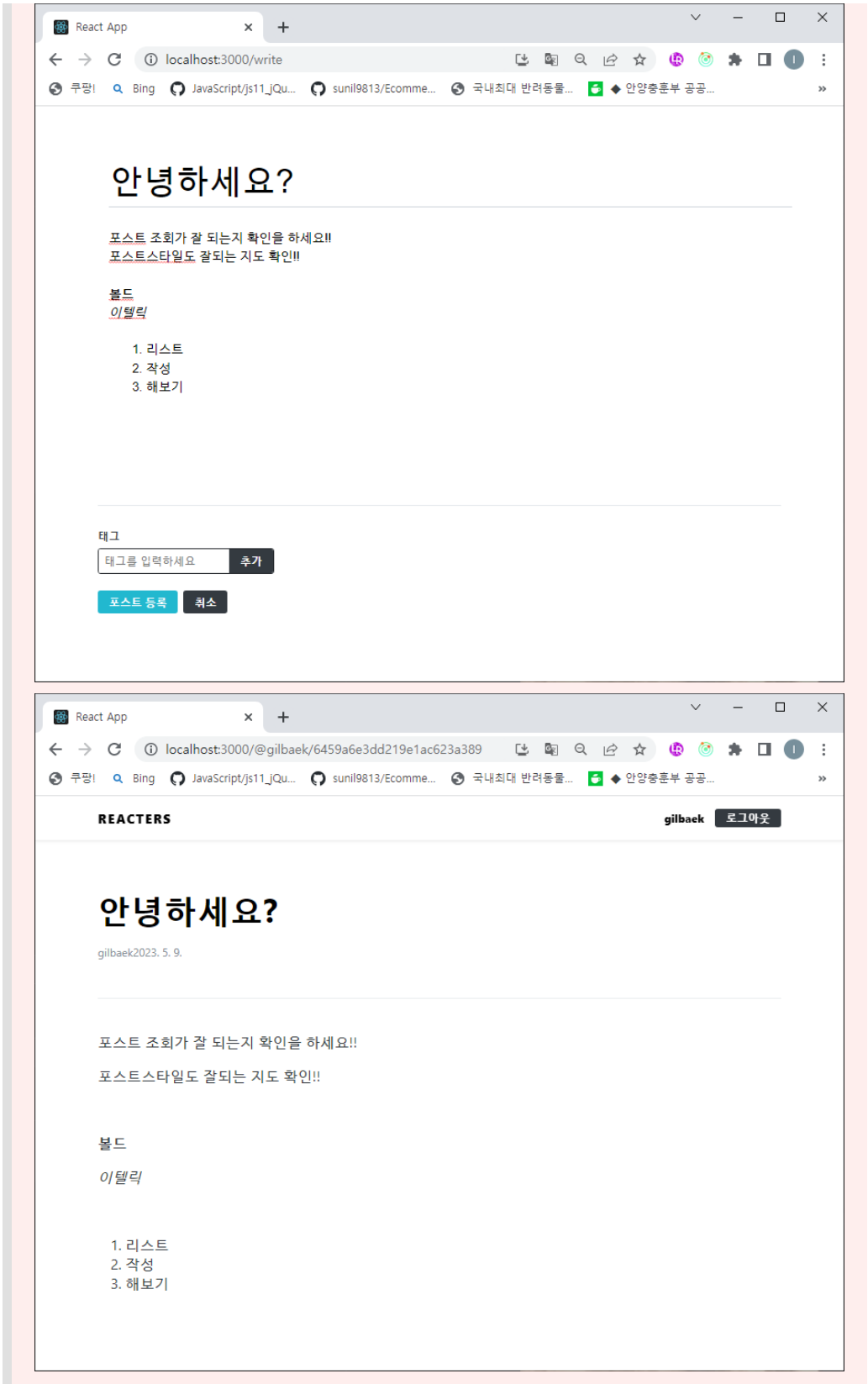
```
import HeaderContainer from '../containers/common/HeaderContainer';
import PostViewerContainer from '../containers/post/PostViewerContainer';
// import PostViewer from '../components/post/PostViewer';

const PostPage = () => {
  return (
    <>
      <HeaderContainer />
      {/* <PostViewer /> */}
      <PostViewerContainer />
    </>
  );
};

export default PostPage;
```

components\post\PostViewer.js - 관련 props를 정의

```
import styled from 'styled-components';
import palette from '../../lib/styles/palette';
import Responsive from '../common/Responsive';


const PostViewerBlock = styled(Responsive)`
  margin-top: 4rem;
```

```
`;

const PostHead = styled.div`
  border-bottom: 1px solid ${palette.gray[2]};
  padding-bottom: 3rem;
  margin-bottom: 3rem;
  h1 {
    font-size: 3rem;
    line-height: 1.5;
    margin: 0;
  }
`;

const SubInfo = styled.div`
  margin-top: 1rem;
  color: ${palette.gray[6]};

  // span 사이에 가운데점 문자 보여주기
  span + span:before {
    color: ${palette.gray[5]};
    padding-left: 0.25rem;
    padding-right: 0.25rem;
    content: '\\B7'; // 가운뎃점 점 문자
  }
`;

const Tags = styled.div`
  margin-top: 0.5rem;
  .tag {
    display: inline-block;
    color: ${palette.cyan[7]};
    text-decoration: none;
    margin-right: 0.5rem;
    &:hover {
      color: ${palette.cyan[6]};
    }
  }
`;

const PostContent = styled.div`
  font-size: 1.3125rem;
  color: ${palette.gray[8]};
`;

const PostViewer = ({ post, error, loading }) => {
  // 에러 발생 시
  if (error) {
    if (error.response && error.response.status === 404) {
      return <PostViewerBlock>존재하지 않는 포스트입니다.</PostViewerBlock>;
    }
    return <PostViewerBlock>오류 발생!</PostViewerBlock>;
  }

  // 로딩중이거나, 아직 포스트 데이터가 없을 시
  if (loading || !post) {
    return null;
  }
```

```
  const { title, body, user, publishedDate, tags } = post;
  return (
    <PostViewerBlock>
      <PostHead>
        <h1>{title}</h1>
        <SubInfo>
          <span>{user.username}</span>
          <apan>{ new Date(publishedDate).toLocaleDateString()}</apan>
        </SubInfo>
        <Tags>
          {tags.map(tag => (
            <div className=" tag ">#{tag}</div>
          ))}
        </Tags>
      </PostHead>
      <PostContent dangerouslySetInnerHTML={{ __html: body }} />
    </PostViewerBlock>
  );
};

export default PostViewer;
```



26.2 포스트목록페이지 구현하기

26.2.1 PostList UI 준비

components/common/SubInfo.js

```
import styled, { css } from 'styled-components';
import { Link } from 'react-router-dom';
import palette from '../../lib/styles/palette';
```

```
const SubInfoBlock = styled.div`
  ${props =>
    props.hasMarginTop &&
    css`
      margin-top: 1rem;
    `}
  color: ${palette.gray[6]};

  /* span 사이에 가운뎃점 문자 보여주기*/
  span + span:before {
    color: ${palette.gray[4]};
    padding-left: 0.25rem;
    padding-right: 0.25rem;
    content: '\\B7'; /* 가운뎃점 문자 */
  }
`;

const SubInfo = ({ username, publishedDate, hasMarginTop }) => {
  return (
    <SubInfoBlock hasMarginTop={hasMarginTop}>
      <span>
        <b>
          <Link to={`/@${username}`}>{username}</Link>
        </b>
      </span>
      <span>{new Date(publishedDate).toLocaleDateString()}</span>
    </SubInfoBlock>
  );
};

export default SubInfo;
```

components/common/Tags.js

```
import styled from 'styled-components';
import palette from '../../lib/styles/palette';
import { Link } from 'react-router-dom';

const TagsBlock = styled.div`
  margin-top: 0.5rem;
  .tag {
    display: inline-block;
    color: ${palette.cyan[7]};
    text-decoration: none;
    margin-right: 0.5rem;
    &:hover {
      color: ${palette.cyan[6]};
    }
  }
`;

const Tags = ({ tags }) => {
  return (
    <TagsBlock>
      {tags.map(tag => (
        <Link className="tag" to={`/?tag=${tag}`} key={tag}>
          #{tag}
```

```
          </Link>
        ))}
      </TagsBlock>
    );
};

export default Tags;
```

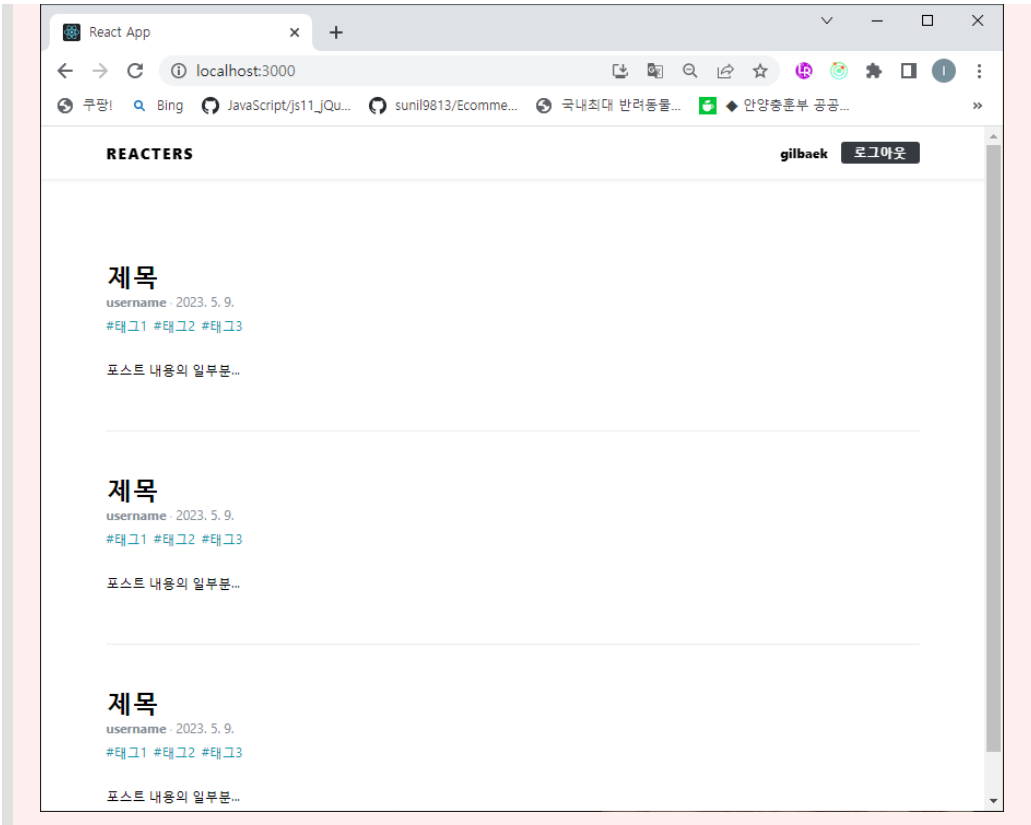
components/posts/PostList.js

```
import styled from 'styled-components';
import Responsive from '../common/Responsive';
import Button from '../common/Button';
import palette from '../../lib/styles/palette';
import SubInfo from '../common/SubInfo';
import Tags from '../common/Tags';

const PostListBlock = styled(Responsive)`
  margin-top: 3rem;
`;

const WritePostButtonWrapper = styled.div`
  display: flex;
  justify-content: flex-end;
  margin-bottom: 3rem;
`;

const PostItemBlock = styled.div`
  padding-top: 3rem;
  padding-bottom: 3rem;
  /* 맨 위 포스트는 padding-top 없음 */
  &:first-child {
    padding-top: 0;
  }
  & + & {
    border-top: 1px solid ${palette.gray[2]};
  }

  h2 {
    font-size: 2rem;
    margin-bottom: 0;
    margin-top: 0;
    &:hover {
      color: ${palette.gray[6]};
    }
  }
  p {
    margin-top: 2rem;
  }
`;

const PostItem = () => {
  return (
    <PostItemBlock>
      <h2>제목</h2>
      <SubInfo
        username="username"
```

```
            publishedDate={new Date()}
          />
          <Tags tags={["태그1", "태그2", "태그3"]} />
          <p>포스트 내용의 일부분...</p>
        </PostItemBlock>
      );
    };

    const PostList = ({ posts, loading, error, showWriteButton }) => {
      // 에러 발생 시
      if (error) {
        return <PostListBlock>에러가 발생했습니다.</PostListBlock>;
      }

      return (
        <PostListBlock>
          <WritePostButtonWrapper>
            {showWriteButton && (
              <Button cyan to="/write">
                새 글 작성하기
              </Button>
            )}
          </WritePostButtonWrapper>
          <div>
              <PostItem />
              <PostItem />
              <PostItem />
          </div>
        </PostListBlock>
      );
    };

    export default PostList;
```

pages/PostListPage.js

```
    import PostList from '../components/posts/PostList';
    import HeaderContainer from '../containers/common/HeaderContainer'

    const PostListPage = () => {
      return (
        <>
          <HeaderContainer />
          <PostList />
        </>
      );
    };
    export default PostListPage;
```



components/post/PostViewer.js - Post 목록

```
import styled from 'styled-components';
import palette from '../../lib/styles/palette';
import Responsive from '../common/Responsive';
import SubInfo from '../common/SubInfo';
import Tags from '../common/Tags';

const PostViewerBlock = styled(Responsive)`
  margin-top: 4rem;
`;

const PostHead = styled.div`
  border-bottom: 1px solid ${palette.gray[2]};
  padding-bottom: 3rem;
  margin-bottom: 3rem;
  h1 {
    font-size: 3rem;
    line-height: 1.5;
    margin: 0;
  }
`;

const PostContent = styled.div`
  font-size: 1.3125rem;
  color: ${palette.gray[8]};
`;

const PostViewer = ({ post, error, loading }) => {
  // 에러 발생 시
  if (error) {
    if (error.response && error.response.status === 404) {
      return <PostViewerBlock>존재하지 않는 포스트입니다.</PostViewerBlock>;
    }
```

```
    return <PostViewerBlock>오류 발생!</PostViewerBlock>;
  }

  // 로딩중이거나, 아직 포스트 데이터가 없을 시
  if (loading || !post) {
    return null;
  }

  const { title, body, user, publishedDate, tags } = post;
  return (
    <PostViewerBlock>
      <PostHead>
        <h1>{title}</h1>
        <SubInfo
          username={user.username}
          publishedDate={ publishedDate }
          hasMarginTop
        />
        <Tags tags={tags} />
      </PostHead>
      <PostContent dangerouslySetInnerHTML={{ __html: body }} />
    </PostViewerBlock>
  );
};

export default PostViewer;
```

26.2.2 포스트목록조회 API 연동하기

lib/api/posts.js

- list API는 username, page, tag값을 쿼리값으로 사용
- axios.get함수의 두 번째 파라미터에 params를 설정하멈 쿼리값 설정을 편하게 할 수 있다.
- /api/posts?username=gilbaek&page=2와 같이 호출

```
import client from './client';

export const writePost = ({ title, body, tags }) =>
  client.post('/api/posts', { title, body, tags });

export const readPost = id => client.get(`/api/posts/${id}`);

export const listPosts = ({ page, username, tag}) => {
  return client.get(`/api/posts`, {
    params: { page, username, tag },
  });
};
```

modules/posts.js - 리덕스상태관리 모듈 적용

```
import { createAction, handleActions } from 'redux-actions';
import createRequestSaga, {
  createRequestActionTypes,
} from '../lib/createRequestSaga';
import * as postsAPI from '../lib/api/posts';
import { takeLatest } from 'redux-saga/effects';
```

```
const [
  LIST_POSTS,
  LIST_POSTS_SUCCESS,
  LIST_POSTS_FAILURE,
] = createRequestActionTypes('posts/LIST_POSTS');

export const listPosts = createAction(
  LIST_POSTS,
  ({ tag, username, page }) => ({ tag, username, page }),
);

const listPostsSaga = createRequestSaga(LIST_POSTS, postsAPI.listPosts);
export function* postsSaga() {
  yield takeLatest(LIST_POSTS, listPostsSaga);
}

const initialState = {
  posts: null,
  error: null,
  // lastPage: 1,
};

const posts = handleActions(
  {
    [LIST_POSTS_SUCCESS]: (state, { payload: posts }) => ({
      ...state,
      posts,
    }),
    [LIST_POSTS_FAILURE]: (state, { payload: error }) => ({
      ...state,
      error,
    }),
  },
  initialState,
);

export default posts;
```

modules/index.js - 리덕스모듈 posts.js 추가

```
import { combineReducers } from 'redux';
import { all } from 'redux-saga/effects';
import auth, { authSaga } from './auth';
import loading from './loading';
import user, { userSaga } from './user';
import write, { writeSaga } from './write';
import post, { postSaga } from './post';
import posts, { postsSaga } from './posts';

const rootReducer = combineReducers({
  auth,
  loading,
  user,
  write,
  post,
  posts
```

```
});

export function* rootSaga() {
  yield all([authSaga(), userSaga(), writeSaga(), postSaga() , postsSaga() ]);
}

export default rootReducer;
```

containers/posts/PostListContainer.js - post목록 컨테이너 추가

- showWriteButton props를 현재 로그인중인 사용자의 정보를 지닌 user 객체로 설정

```
import React, { useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import PostList from '../../components/posts/PostList';
import { listPosts } from '../../modules/posts';
import { useParams, useSearchParams } from 'react-router-dom';

const PostListContainer = () => {
  const { username } = useParams();
  const [searchParams] = useSearchParams();
  const dispatch = useDispatch();
  const { posts, error, loading, user } = useSelector(
    ({ posts, loading, user }) => ({
      posts: posts.posts,
      error: posts.error,
      loading: loading['posts/LIST_POSTS'],
      user: user.user,
    }),
  );
  useEffect(() => {
    const tag = searchParams.get('tag');
    // page가 없으면 1을 기본값으로 사용
    const page = parseInt(searchParams.get('page'), 10) || 1;
    dispatch(listPosts({ tag, username, page }));
  }, [dispatch, searchParams, username]);

  return (
    <PostList
      loading={loading}
      error={error}
      posts={posts}
      showWriteButton={user}
    />
  );
};

export default PostListContainer;
```

pages/PostListPage.js - PostListContainer 적용

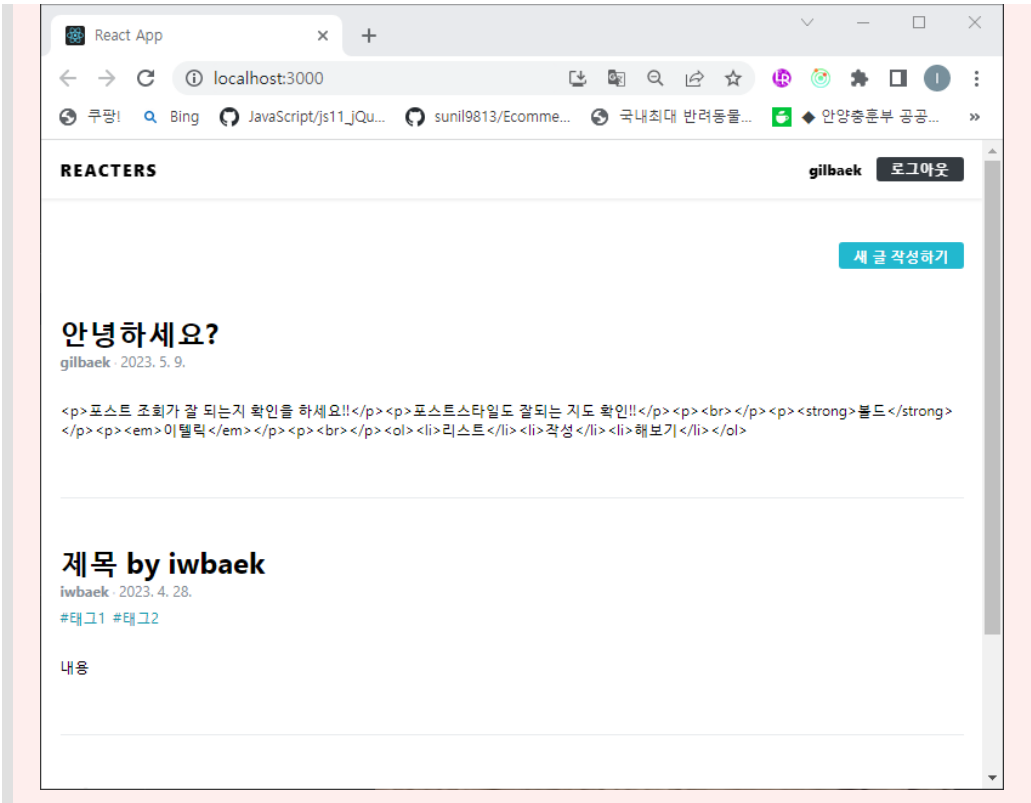
```
// import PostList from '../components/posts/PostList';
import HeaderContainer from '../containers/common/HeaderContainer'
import PostListContainer from '../containers/posts/PostListContainer';

const PostListPage = () => {
  return (
    <>
```

```
        <HeaderContainer />
        {/* <PostList /> */}
        <PostListContainer />
      </>
    );
};
export default PostListPage;
```

components/posts/PostList.js

```
import styled from 'styled-components';
import Responsive from '../common/Responsive';
import Button from '../common/Button';
import palette from '../../lib/styles/palette';
import SubInfo from '../common/SubInfo';
import Tags from '../common/Tags';
import { Link } from 'react-router-dom';

const PostListBlock = styled(Responsive)`
  margin-top: 3rem;
`;

const WritePostButtonWrapper = styled.div`
  display: flex;
  justify-content: flex-end;
  margin-bottom: 3rem;
`;

const PostItemBlock = styled.div`
  padding-top: 3rem;
  padding-bottom: 3rem;
  /* 맨 위 포스트는 padding-top 없음 */
  &:first-child {
    padding-top: 0;
  }
  & + & {
    border-top: 1px solid ${palette.gray[2]};
  }

  h2 {
    font-size: 2rem;
    margin-bottom: 0;
    margin-top: 0;
    &:hover {
      color: ${palette.gray[6]};
    }
  }
  p {
    margin-top: 2rem;
  }
`;

const PostItem = ({ post }) => {
  const { publishedDate, user, tags, title, body, _id } = post;
  return (
    <PostItemBlock>
      <h2>
```

```
            <Link to={`/@${user.username}/${_id}`}>{title}</Link>
          </h2>
          <SubInfo
            username={user.username}
            publishedDate={new Date(publishedDate)}
          />
          <Tags tags={tags} />
          <p>{body}</p>
        </PostItemBlock>
      );
    };

    const PostList = ({ posts, loading, error, showWriteButton }) => {
      // 에러 발생 시
      if (error) {
        return <PostListBlock>에러가 발생했습니다.</PostListBlock>;
      }

      return (
        <PostListBlock>
          <WritePostButtonWrapper>
            {showWriteButton && (
              <Button cyan to="/write">
                새 글 작성하기
              </Button>
            )}
          </WritePostButtonWrapper>
          {/*  로딩 중 아니고, 포스트 배열이 존재할 때만 보여줌 */}
          {!loading && posts && (
            <div>
              {posts.map(post => (
                <PostItem post={post} key={post._id} />
              ))}
            </div>
          )}
        </PostListBlock>
      );
    };

    export default PostList;
```



- 목록에 html 태그가 그대로 보이는 것을 없애는 작업은 서버쪽셍서 처리해야 한다.
- 클라이언트에서 처리할 수도 있지만 현재, body의 글자수(200자)제한으로 태그를 없애는 작업이 잘 안될 가능성이 있다.

26.2.3 HTML 필터링하기

- sanitize-html 라이브러리를 이용 html을 필터링. 이 라이브러리는 html작성 및 보여주기에 매우 유용하다.
- html제거기능 및 특정 html만 허용기능이 있어 악성 스크립트삽입을 막을 수 있다.
- backend에 설치
  - 프로젝트(서버) 중지후 설치해야 한다. 설치 후 재시작(yarn start)
  - `yarn add sanitize-html`

backend에서 src/api/posts/posts.ctrl.js - html 필터링하기

- 수정후 재시작한 후에 확인

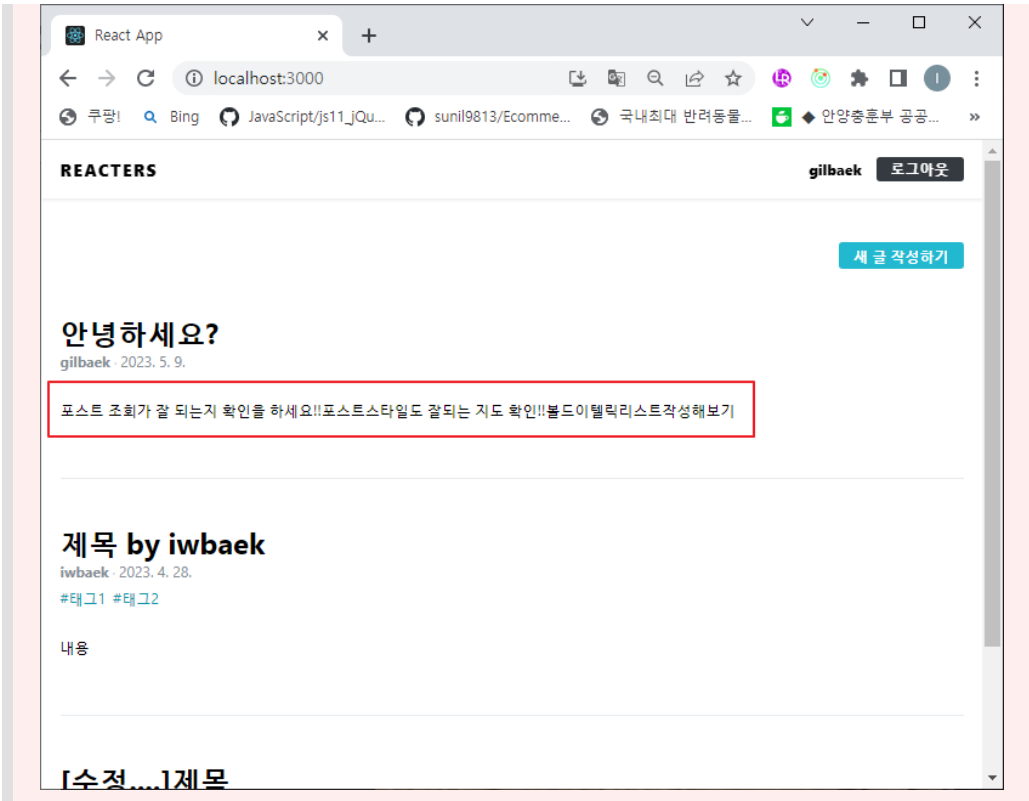
```
import Post from '../../models/post';
import mongoose from 'mongoose';
import Joi from 'joi';
import sanitizeHtml from 'sanitize-html';

const { ObjectId } = mongoose.Types;

// sanitize-html은 특정태그와 특정속성만 허용할 수 있다.
// sanitizeOption에 특정태그와 특정속성만 허용하도록 정의
const sanitizeOption = {
  allowedTags: [
    'h1',
    'h2',
    'b',
    'i',
```

```
    'u',
    's',
    'p',
    'ul',
    'ol',
    'li',
    'blockquote',
    'a',
    'img',
  ],
  allowedAttributes: {
    a: ['href', 'name', 'target'],
    img: ['src'],
    li: ['class'],
  },
  allowedSchemes: ['data', 'http'],
};

export const checkObjectId = (ctx, next) => {

  const { id } = ctx.params;
  if (!ObjectId.isValid(id)) {
    ctx.status = 400; // Bad Request
    return;
  }
  return next();
};

export const getPostById = async (ctx, next) => {

  const { id } = ctx.params;
  if (!ObjectId.isValid(id)) {
    ctx.status = 400; // Bad Request
    return;
  }

  try {
    const post = await Post.findById(id);
    // 포스트가 존재하지 않을 때
    if (!post) {
      ctx.status = 404; // Not Found
      return;
    }
    ctx.state.post = post;
    return next();
  } catch (e) {
    ctx.throw(500, e);
  }
};

export const checkOwnPost = (ctx, next) => {

  const { user, post } = ctx.state;

  if (post.user._id.toString() !== user._id) {
    ctx.status = 403;
    return;
```

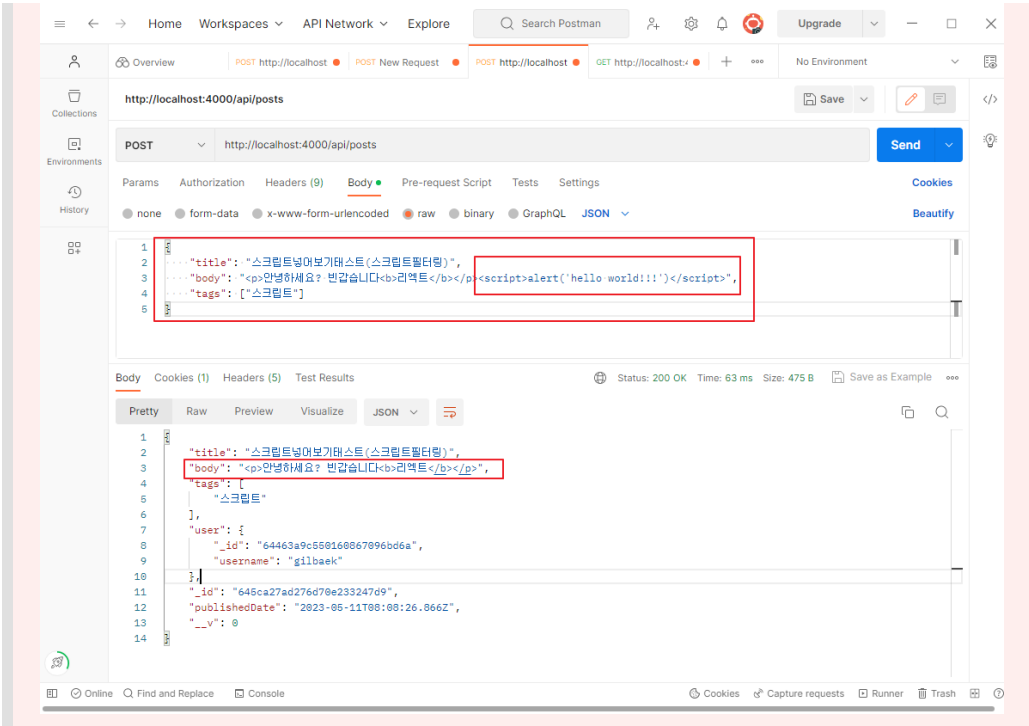
```
  }
  return next();
};

/*
  POST /api/posts
  {
    title: '제목',
    body: '내용',
    tags: ['태그1', '태그2']
  }
*/
export const write = async ctx => {

  const schema = Joi.object().keys({
    // 객체가 다음 필드를 가지고 있음을 검증
    title: Joi.string().required(), // required() 가 있으면 필수 항목
    body: Joi.string().required(),
    tags: Joi.array()
      .items(Joi.string())
      .required(), // 문자열로 이루어진 배열
  });

  // 검증 후, 검증 실패시 에러처리
  const result = schema.validate(ctx.request.body);
  if (result.error) {
    ctx.status = 400; // Bad Request
    ctx.body = result.error;
    return;
  }

  const { title, body, tags } = ctx.request.body;

  const post = new Post({
    title,
    body: sanitizeHtml(body, sanitizeOption),
    tags,
    user: ctx.state.user,
  });
  try {
    await post.save();
    ctx.body = post;
  } catch (e) {
    ctx.throw(500, e);
  }
};

// html태그를 없애고 내용이 너무길면 200자로 제한하는 함수
const removeHtmlAndShorten = (body) => {
  const filtered = sanitizeHtml(body, {
    allowedTags: [],
  });
  return filtered.length < 200 ? filtered : `${filtered.slice(0, 200)}...`;
};

/*
  GET /api/posts?username=&tag=&page=
```

```
*/

export const list = async (ctx) => {
  // query 는 문자열이기 때문에 숫자로 변환해주어야합니다.
  // 값이 주어지지 않았다면 1 을 기본으로 사용합니다.
  const page = parseInt(ctx.query.page || '1', 10);

  if (page < 1) {
    ctx.status = 400;
    return;
  }

  const { tag, username } = ctx.query;
  // tag, username 값이 유효하면 객체 안에 넣고, 그렇지 않으면 넣지 않음
  const query = {
    ...(username ? { 'user.username': username } : {}),
    ...(tag ? { tags: tag } : {}),
  };

  try {
    const posts = await Post.find(query)
      .sort({ _id: -1 })
      .limit(10)
      .skip((page - 1) * 10)
      .lean()
      .exec();
    const postCount = await Post.countDocuments(query).exec();
    ctx.set('Last-Page', Math.ceil(postCount / 10));
    ctx.body = posts.map((post) => ({
      ...post,
      body: removeHtmlAndShorten(post.body),
    }));
  } catch (e) {
    ctx.throw(500, e);
  }
};

/*
  GET /api/posts/:id
*/
export const read = async ctx => {
  ctx.body = ctx.state.post;
};

/*
  DELETE /api/posts/:id
*/

export const remove = async ctx => {
  const {id} = ctx.params;
  try {
    const post = await Post.findByIdAndRemove(id).exec();
    ctx.status = 204; // no content 성공했지만 응답데이터 없음
    ctx.body = post;
  } catch(e) {
    ctx.throw(500, e);
  }
```

```
};

/*
  PATCH /api/posts/:id
  {
    title: '수정',
    body: '수정 내용',
    tags: ['수정', '태그']
  }
*/
export const update = async ctx => {

  const { id } = ctx.params;
  // write 에서 사용한 schema 와 비슷한데, required() 가 없습니다.
  const schema = Joi.object().keys({
    title: Joi.string(),
    body: Joi.string(),
    tags: Joi.array().items(Joi.string()),
  });

  // 검증 후, 검증 실패시 에러처리
  const result = schema.validate(ctx.request.body);
  if (result.error) {
    ctx.status = 400; // Bad Request
    ctx.body = result.error;
    return;
  }

  const nextData = { ...ctx.request.body }; // 객체를 복사하고
  // body 값이 주어졌으면 HTML 필터링
  if (nextData.body) {
    nextData.body = sanitizeHtml(nextData.body, sanitizeOption);
  }

  try {
    const post = await Post.findByIdAndUpdate(id, nextData, {
      new: true, // 이 값을 설정하면 업데이트된 데이터를 반환한다, false일 경우
업데이트전 값을 리턴
    }).exec();
    if(!post) {
      ctx.status = 404;
      return;
    }
    ctx.status = 204; // no content 성공했지만 응답데이터 없음
    ctx.body = post;
  } catch(e) {
    ctx.throw(500, e);
  }
};
```




```
{
    "title": "스크립트넣어보기태스트(스크립트필터링)",
    "body": "<p>안녕하세요? 빈갑습니다<b>리엑트</b></p><script>alert('hello world!!!')</script>",
    "tags": ["스크립트"]
}
```



26.2.4 페이지네이션 구현

- Failed to load plugin 'jsx-a11y' declared in 'package.json » eslint-config-react-app 에러발생
  - 어떻게 해결 되었는지 잘 모르겠다

- 하지만 front에서 package.json파일에 "react-app/jest"을 주석후 다시 해재한 후에 저장하니 해결되었다

```
"eslintConfig": {
"extends": [
 "react-app",
 "react-app/jest"
]
},
```

lib/createRequestSara.js - 페이지네이션

- 액션안에 meta값을 response로 넣어 주면 나중에 http헤더 및 상태코드를 쉽게 조회할 수 있다.

```
import { call, put } from 'redux-saga/effects';
import { startLoading, finishLoading } from '../modules/loading';

export const createRequestActionTypes = type => {
  const SUCCESS = `${type}_SUCCESS`;
  const FAILURE = `${type}_FAILURE`;
  return [type, SUCCESS, FAILURE];
};

export default function createRequestSaga(type, request) {
  const SUCCESS = `${type}_SUCCESS`;
  const FAILURE = `${type}_FAILURE`;

  return function*(action) {
    yield put(startLoading(type)); // 로딩 시작
    try {
      const response = yield call(request, action.payload);
      yield put({
        type: SUCCESS,
        payload: response.data,
        meta: response,
      });
    } catch (e) {
      yield put({
        type: FAILURE,
        payload: e,
        error: true
      });
    }
    yield put(finishLoading(type)); // 로딩 끝
  };
}
```

modules/posts.js - 페이지네이션

```
import { createAction, handleActions } from 'redux-actions';
import createRequestSaga, {
  createRequestActionTypes,
} from '../lib/createRequestSaga';
import * as postsAPI from '../lib/api/posts';
import { takeLatest } from 'redux-saga/effects';

const [
```

```
  LIST_POSTS,
  LIST_POSTS_SUCCESS,
  LIST_POSTS_FAILURE,
] = createRequestActionTypes('posts/LIST_POSTS');

export const listPosts = createAction(
  LIST_POSTS,
  ({ tag, username, page }) => ({ tag, username, page }),
);

const listPostsSaga = createRequestSaga(LIST_POSTS, postsAPI.listPosts);
export function* postsSaga() {
  yield takeLatest(LIST_POSTS, listPostsSaga);
}

const initialState = {
  posts: null,
  error: null,
  lastPage: 1,
};

const posts = handleActions(
  {
    [LIST_POSTS_SUCCESS]: (state, { payload: posts, meta: response }) => ({
      ...state,
      posts,
      lastPage: parseInt(response.headers['last-page'], 10), // 문자열을 숫자
로 변환
    }),
    [LIST_POSTS_FAILURE]: (state, { payload: error }) => ({
      ...state,
      error,
    }),
  },
  initialState,
);

export default posts;
```

components/posts/Pagination.js

- 리덕스 스토어 안에 lastPage값을 저장

```
import React from 'react';
import styled from 'styled-components';
import qs from 'qs';
import Button from '../common/Button';

const PaginationBlock = styled.div`
  width: 320px;
  margin: 0 auto;
  display: flex;
  justify-content: space-between;
  margin-bottom: 3rem;
`;
const PageNumber = styled.div``;
```

```
const buildLink = ({ username, tag, page }) => {
  const query = qs.stringify({ tag, page });
  return username ? `/@${username}?${query}` : `/?${query}`;
};

const Pagination = ({ page, lastPage, username, tag }) => {
  return (
    <PaginationBlock>
      <Button
        disabled={page === 1}
        to={
          page === 1 ? undefined : buildLink({ username, tag, page: page - 1
})
        }
      >
        이전
      </Button>
      <PageNumber>{page}</PageNumber>
      <Button
        disabled={page === lastPage}
        to={
          page === lastPage
            ? undefined
            : buildLink({ username, tag, page: page + 1 })
        }
      >
        다음
      </Button>
    </PaginationBlock>
  );
};

export default Pagination;
```

components/common/Button.js

```
import React from 'react';
import styled, { css } from 'styled-components';
import { Link } from 'react-router-dom';
import palette from '../../lib/styles/palette';

const buttonStyle = css`
  border: none;
  border-radius: 4px;
  font-size: 1rem;
  font-weight: bold;
  padding: 0.25rem 1rem;
  color: white;
  outline: none;
  cursor: pointer;

  background: ${palette.gray[8]};
  &:hover {
    background: ${palette.gray[6]};
  }

  ${props =>
```

```
      props.fullWidth &&
      css`
        padding-top: 0.75rem;
        padding-bottom: 0.75rem;
        width: 100%;
        font-size: 1.125rem;
      `}

    ${props =>
      props.cyan &&
      css`
        background: ${palette.cyan[5]};
        &:hover {
          background: ${palette.cyan[4]};
        }
      `}

    &:disabled {
      background: ${palette.gray[3]};
      color: ${palette.gray[5]};
      cursor: not-allowed;
    }
  `;

  const StyledButton = styled.button`
    ${buttonStyle}
  `;

  const StyledLink = styled(Link)`
    ${buttonStyle}
  `;

  const Button = props => {
    return props.to ? (
      <StyledLink {...props} cyan={props.cyan ? 1 : 0} />
    ) : (
      <StyledButton {...props} />
    );
  };

  export default Button;
```

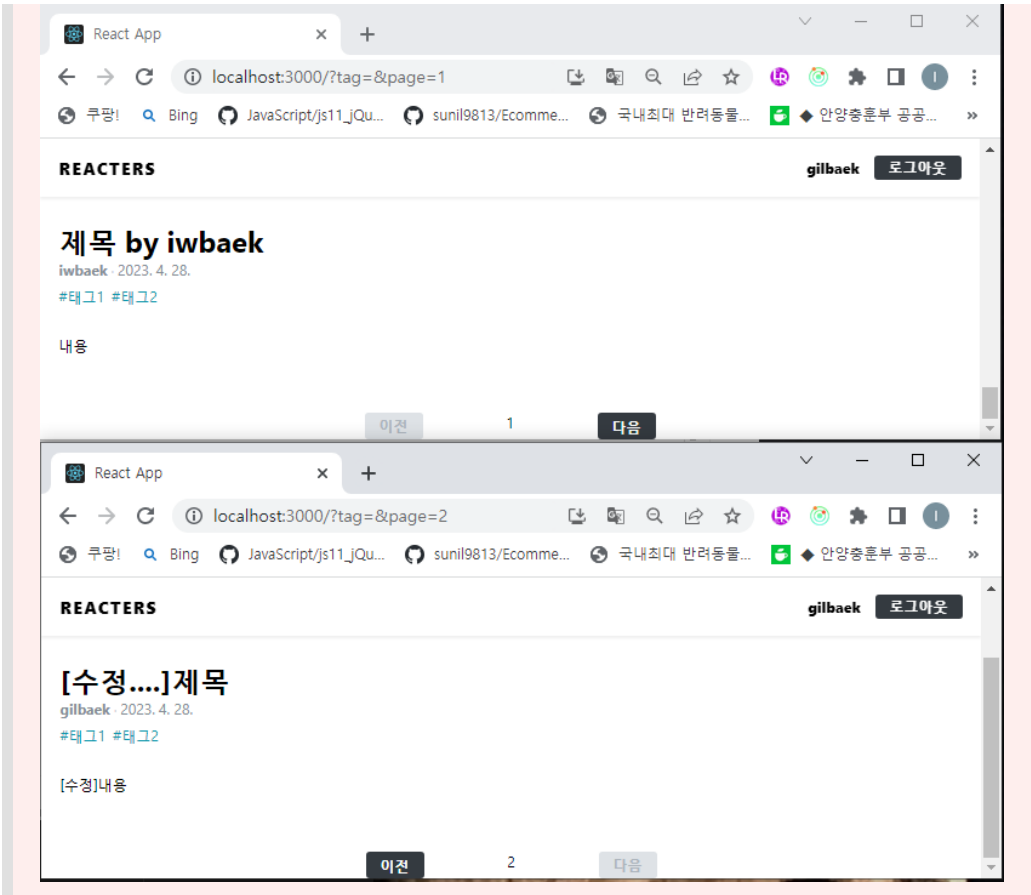
containers/posts/PaginationContainer.js

```
  import React from 'react';
  import Pagination from '../../components/posts/Pagination';
  import { useSelector } from 'react-redux';
  import { useParams, useSearchParams } from 'react-router-dom';

  const PaginationContainer = () => {
    const [searchParams] = useSearchParams();

    const { username } = useParams();
    const tag = searchParams.get('tag');
    // page가 없으면 1을 기본값으로 사용
    const page = parseInt(searchParams.get('page'), 10) || 1;
```

```
  const { lastPage, posts, loading } = useSelector(({ posts, loading }) => ({
    lastPage: posts.lastPage,
    posts: posts.posts,
    loading: loading['posts/LIST_POSTS'],
  }));

  // 포스트 데이터가 없거나 로딩 중이면 아무것도 보여주지 않음
  if (!posts || loading) return null;

  return (
    <Pagination
      tag={tag}
      username={username}
      page={parseInt(page, 10)}
      lastPage={lastPage}
    />
  );
};

export default PaginationContainer;
```

pages/PostListPage.js

```
// import PostList from '../components/posts/PostList';
import HeaderContainer from '../containers/common/HeaderContainer'
import PaginationContainer from '../containers/posts/PaginationContainer';
import PostListContainer from '../containers/posts/PostListContainer';

const PostListPage = () => {
  return (
    <>
      <HeaderContainer />
      {/* <PostList /> */}
      <PostListContainer />
      <PaginationContainer />
    </>
  );
};
export default PostListPage;
```


React App
localhost:3000/?tag=&page=1
쿠팡!
Bing
JavaScript/js11_jQu...
sunil9813/Ecomme...
국내최대 반려동물...
안양충훈부 공공...
REACTERS
gilbaek
로그아웃
제목 by iwbaek
iwbaek · 2023. 4. 28.
#태그1 #태그2
내용
이전
1
다음
React App
localhost:3000/?tag=&page=2
쿠팡!
Bing
JavaScript/js11_jQu...
sunil9813/Ecomme...
국내최대 반려동물...
안양충훈부 공공...
REACTERS
gilbaek
로그아웃
[수정....]제목
gilbaek · 2023. 4. 28.
#태그1 #태그2
[수정]내용
이전
2
다음