25. FE 글쓰기기능 구현하기

1. mkdir 25.write\frontend
2. cd 25.write\frontend
3. 23.jwt 전체복사(node_modueles까지 포함) -> 25.write\backend
    - 23.jwt를 그대로 사용할 것
4. 24.blog\blog-frontend 소스 전체 복사하기 -> 25.write\frontend
    - node_modules는 24.blog\blog-frontend의 node_modules를 잘라내기해 올 것
    - `npm i`

25.1 에디터UI구현하기

- 글을 작성하는 에디터는 quill라이브러리 사용
    - 설치 `yarn add quill`

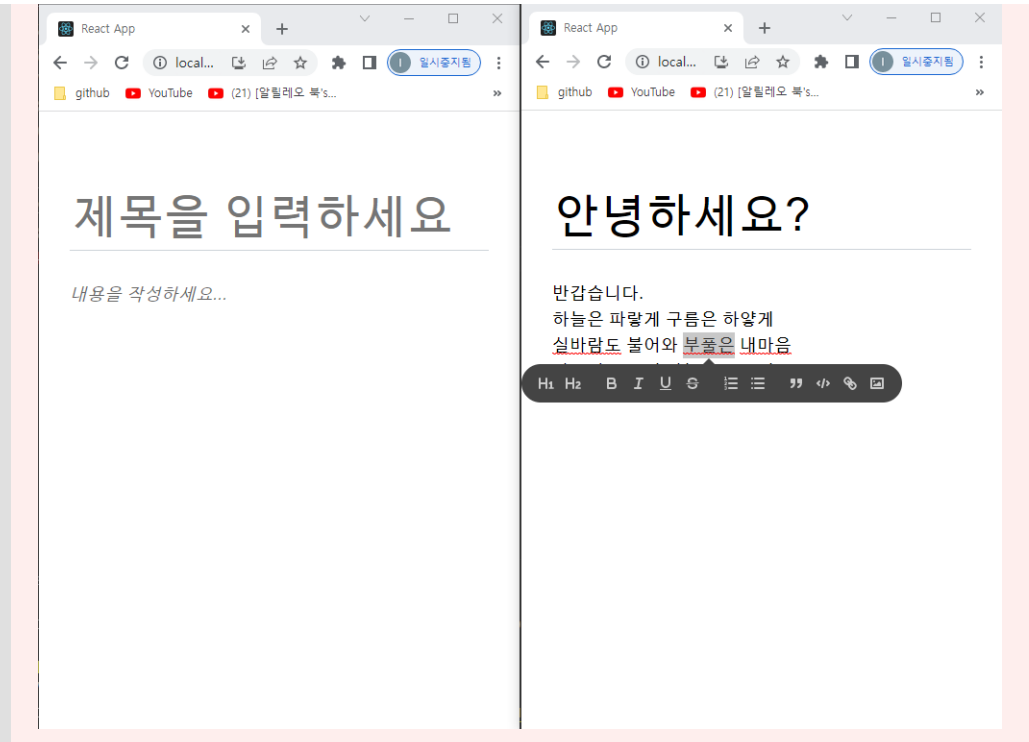
components\write\Editor.js

```
import React, { useRef, useEffect } from 'react';
import Quill from 'quill';
import 'quill/dist/quill.bubble.css';
import styled from 'styled-components';
import palette from '../../lib/styles/palette';
import Responsive from '../common/Responsive';

const EditorBlock = styled(Responsive)`
  /* 페이지 위 아래 여백 지정 */
  padding-top: 5rem;
  padding-bottom: 5rem;
`;
const TitleInput = styled.input`
  font-size: 3rem;
  outline: none;
  padding-bottom: 0.5rem;
  border: none;
  border-bottom: 1px solid ${palette.gray[4]};
  margin-bottom: 2rem;
  width: 100%;
`;
const QuillWrapper = styled.div`
  /* 최소 크기 지정 및 padding 제거 */
  .ql-editor {
    padding: 0;
    min-height: 320px;
    font-size: 1.125rem;
    line-height: 1.5;
  }
  .ql-editor.ql-blank::before {
    left: 0px;
  }
`;
```

```
const Editor = ({ title, body, onChangeField }) => {
  const quillElement = useRef(null); // Quill을 적용할 DivElement를 설정
  const quillInstance = useRef(null); // Quill 인스턴스를 설정

  useEffect(() => {
    quillInstance.current = new Quill(quillElement.current, {
      theme: 'bubble',
      placeholder: '내용을 작성하세요...',
      modules: {
        // 더 많은 옵션
        // https://quilljs.com/docs/modules/toolbar/ 참고
        toolbar: [
          [{ header: '1' }, { header: '2' }],
          ['bold', 'italic', 'underline', 'strike'],
          [{ list: 'ordered' }, { list: 'bullet' }],
          ['blockquote', 'code-block', 'link', 'image'],
        ],
      },
    });
  }, []);

  return (
    <EditorBlock>
      <TitleInput
        placeholder="제목을 입력하세요" />
      <QuillWrapper>
        <div ref={quillElement} />
      </QuillWrapper>
    </EditorBlock>
  );
};

export default Editor;
```

pages\WritePage.js

```
import Editor from '../components/write/Editor';
import Responsive from '../components/common/Responsive';

const WritePage = () => {
  return (
    <Responsive>
      <Editor />

    </Responsive>
  );
};

export default WritePage;
```



25.2 에디터하단 컴퍼넌트 UI 구현하기

25..1 TagBox 만들기

components\write\TagBox.js

- TagItem, TagList 컴퍼넌트추가, 이렇게 분리시킨이유는 랜더링을 최소화 하기 위해서 이다.
- TagBox는 input변경시, 태그목록변경시에 랜더링을 한다. 분리하지 않으면 각각의 값변경시마다 랜더링이 된다.
- React.memo를 사용하여 컴퍼넌트를 감싸주면 해당 컴퍼넌트가 받아오는 props가 변경시에만 리랜더링이 된다.
- 태그등록 및 삭제확인, 삭제는 추가된 태그를 클릭하면 삭제가 된다.

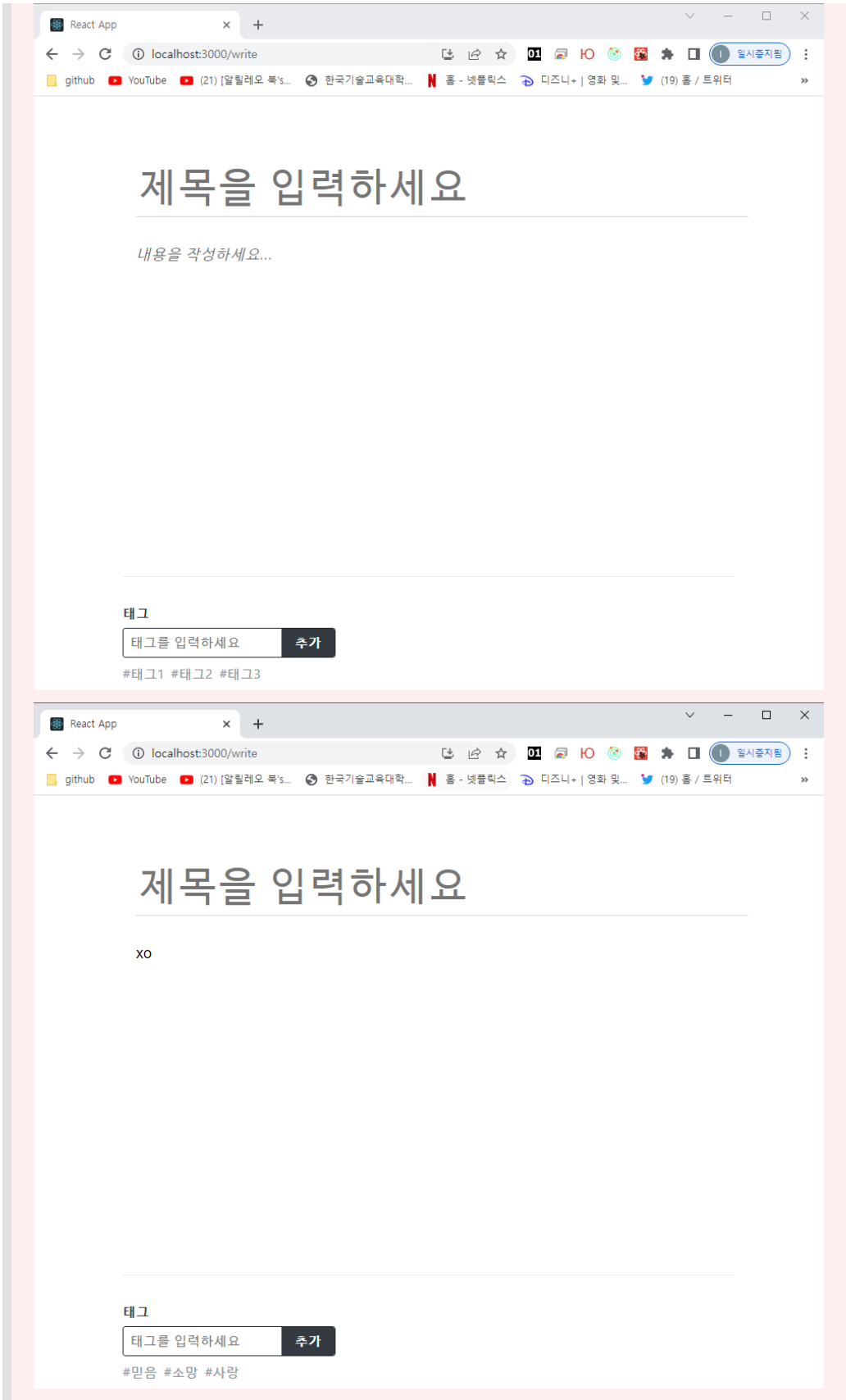
```
import React, { useState, useCallback } from 'react';
import styled from 'styled-components';
import palette from '../../lib/styles/palette';

const TagBoxBlock = styled.div`
  width: 100%;
  border-top: 1px solid ${palette.gray[2]};
  padding-top: 2rem;

  h4 {
    color: ${palette.gray[8]};
    margin-top: 0;
    margin-bottom: 0.5rem;
  }
`;

const TagForm = styled.form`
  border-radius: 4px;
  overflow: hidden;
```

```
  display: flex;
  width: 256px;
  border: 1px solid ${palette.gray[9]}; /* 스타일 초기화 */
  input,
  button {
    outline: none;
    border: none;
    font-size: 1rem;
  }

  input {
    padding: 0.5rem;
    flex: 1;
    min-width: 0;
  }

  button {
    cursor: pointer;
    padding-right: 1rem;
    padding-left: 1rem;
    border: none;
    background: ${palette.gray[8]};
    color: white;
    font-weight: bold;
    &:hover {
      background: ${palette.gray[6]};
    }
  }
`;

const Tag = styled.div`
  margin-right: 0.5rem;
  color: ${palette.gray[6]};
  cursor: pointer;
  &:hover {
    opacity: 0.5;
  }
`;

const TagListBlock = styled.div`
  display: flex;
  margin-top: 0.5rem;
`;

// React.memo를 사용하여 tag 값이 바뀔 때만 리렌더링되도록 처리
const TagItem = React.memo(({ tag, onRemove, onChangeTags }) => (
  <Tag onClick={() => onRemove(tag)}>#{tag}</Tag>
));

// React.memo를 사용하여 tags 값이 바뀔 때만 리렌더링되도록 처리
const TagList = React.memo(({ tags, onRemove }) => (
  <TagListBlock>
    {tags.map(tag => (
      <TagItem key={tag} tag={tag} onRemove={onRemove} />
    ))}
  </TagListBlock>
));
```

```
const TagBox = ({ tags, onChangeTags }) => {
  const [input, setInput] = useState('');
  const [localTags, setLocalTags] = useState([]);

  const insertTag = useCallback(
    tag => {
      if (!tag) return; // 공백이라면 추가하지 않음
      if (localTags.includes(tag)) return; // 이미 존재한다면 추가하지 않음
      const nextTags = [...localTags, tag];
      setLocalTags(nextTags);
    },
    [localTags],
  );

  const onRemove = useCallback(
    tag => {
      const nextTags = localTags.filter(t => t !== tag);
      setLocalTags(nextTags);
    },
    [localTags],
  );

  const onChange = useCallback(e => {
    setInput(e.target.value);
  }, []);

  const onSubmit = useCallback(
    e => {
      e.preventDefault();
      insertTag(input.trim()); // 앞뒤 공백 없앤 후 등록
      setInput(''); // input 초기화
    },
    [input, insertTag],
  );

  return (
    <TagBoxBlock>
      <h4>태그</h4>
      <TagForm onSubmit={onSubmit}>
        <input
          placeholder="태그를 입력하세요"
          value={input}
          onChange={onChange}
        />
        <button type="submit">추가</button>
      </TagForm>
      <TagList tags={localTags} onRemove={onRemove} />
    </TagBoxBlock>
  );
};

export default TagBox;
```

pages\WritePage.js

```jsx
import Editor from '../components/write/Editor';
import TagBox from '../components/write/TagBox';
import Responsive from '../components/common/Responsive';

const WritePage = () => {
  return (
    <Responsive>
      <Editor />
      <TagBox />
    </Responsive>
  );
};

export default WritePage;
```



25.2.1.1 WriteActionButton 만들기

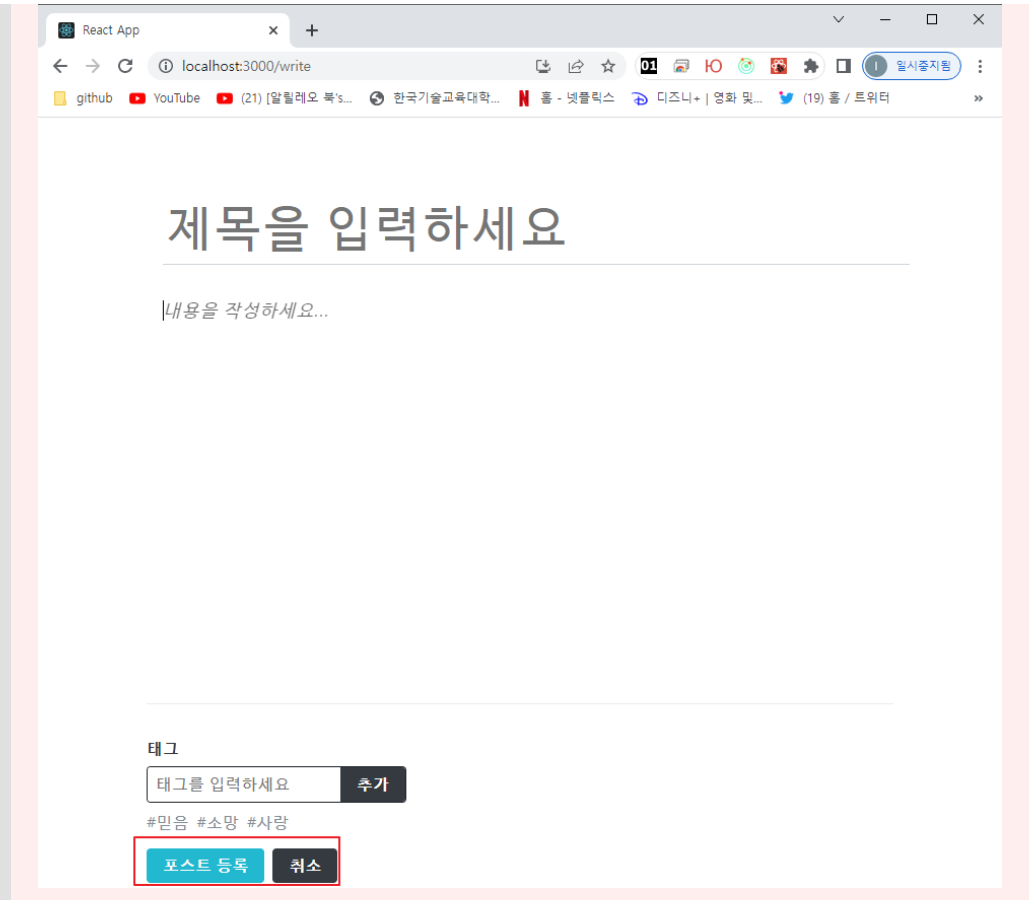
components/write/WriteActionButtons.js

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

const WriteActionButtonsBlock = styled.div`
  margin-top: 1rem;
```

```
  margin-bottom: 3rem;
  button + button {
    margin-left: 0.5rem;
  }
`;

/* TagBox에서 사용하는 버튼과 일치하는 높이로 설정 후 서로 간의 여백 지정 */
const StyledButton = styled(Button)`
  height: 2.125rem;
  & + & {
    margin-left: 0.5rem;
  }
`;

const WriteActionButtons = ({ onCancel, onPublish }) => {
  return (
    <WriteActionButtonsBlock>
      <StyledButton cyan onClick={onPublish}>
        포스트 등록
      </StyledButton>
      <StyledButton onClick={onCancel}>취소</StyledButton>
    </WriteActionButtonsBlock>
  );
};

export default WriteActionButtons;
```

pages\WritePage.js

```
import Editor from '../components/write/Editor';
import TagBox from '../components/write/TagBox';
import Responsive from '../components/common/Responsive';
import WriteActionButtons from '../components/write/WriteActionButtons';

const WritePage = () => {
  return (
    <Responsive>
      <Editor />
      <TagBox />
      <WriteActionButtons />
    </Responsive>
  );
};

export default WritePage;
```



25.3 리덕스로 글쓰기 상태 관리하기

- write 리덕스모듈 작성하기

modules\write.js

```
import { createAction, handleActions } from 'redux-actions';

const INITIALIZE = 'write/INITIALIZE'; // 모든 내용 초기화
const CHANGE_FIELD = 'write/CHANGE_FIELD'; // 특정 key 값 바꾸기

export const initialize = createAction(INITIALIZE);
export const changeField = createAction(CHANGE_FIELD, ({ key, value }) => ({
  key,
  value,
}));

const initialState = {
  title: '',
  body: '',
  tags: [],
};

const write = handleActions(
  {
    [INITIALIZE]: state => initialState, // initialState를 넣으면 초기상태로 바
꿈
    [CHANGE_FIELD]: (state, { payload: { key, value } }) => ({
      ...state,
      [key]: value, // 특정 key 값을 업데이트
```

```
    }),
  },
  initialState,
);

export default write;
```

modules/index.js

- write 리듀서를 루트리슈서에 등록하기

```
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 from './write';

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

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

export default rootReducer;
```

25.3.1 EditorContainer 만들기

- title값과 body값을 리덕스스토어에서 블러와 Editor컴퍼넌트에 전달
- Quill에디터는 일반 input, textarea태그가 아니기 때문에 onChange와 value값으로 상태관리를 할 수 없다.
- 따라서, 지금은 에디터값변경시 리덕스스토어에 값을 전달하는 기능만 구현 추후 포스트수정기능 구현시에 처리
- onChangeField함수는 useCallback으로 감싼것은 Editor컴퍼넌트에서 사용할 useEffect에서 onChangeField를 사용예정
- onChangeField를 useCallback으로 감싸 주어야 나중에 Editor의 useEffect가 나타났을 때 한 번만 실행된다.
- 또한, 사용자가 WritePage를 벗어날 때는 데이터를 초기화 해야 한다.
- 컴퍼넌트가 언마운트될 때 useEffect로 INITIALIZE액션을 발생시켜 리덕스의 write 관련상태를 초기화
- 만약, 초기화하지 않는다면 포스트작성 후 다시 글쓰기 페이지에 들어 왔을 때 이전내용이 남아있게 된다.
- 컨테이너 컴퍼넌트(EditorContainer)작성완료 후 WritePage에서 Editor를 EditorContainer로 대체

containers/write/EditorContainer

```
import React, { useEffect, useCallback } from 'react';
import Editor from '../../components/write/Editor';
```

```
import { useSelector, useDispatch } from 'react-redux';
import { changeField, initialize } from '../../modules/write';

const EditorContainer = () => {
  const dispatch = useDispatch();
  const { title, body } = useSelector(({ write }) => ({
    title: write.title,
    body: write.body,
  }));
  const onChangeField = useCallback(payload => dispatch(changeField(payload)),
[
    dispatch,
  ]);
  // 언마운트될 때 초기화
  useEffect(() => {
    return () => {
      dispatch(initialize());
    };
  }, [dispatch]);
  return <Editor onChangeField={onChangeField} title={title} body={body} />;
};

export default EditorContainer;
```

pages/WritePage.js - Editor를 EditorContainer로 대체

```
// import Editor from '../components/write/Editor';
import TagBox from '../components/write/TagBox';
import Responsive from '../components/common/Responsive';
import WriteActionButtons from '../components/write/WriteActionButtons';
import EditorContainer from '../containers/write/EditorContainer';

const WritePage = () => {
  return (
    <Responsive>
      {/* <Editor /> */}
      <EditorContainer />
      <TagBox />
      <WriteActionButtons />
    </Responsive>
  );
};

export default WritePage;
```

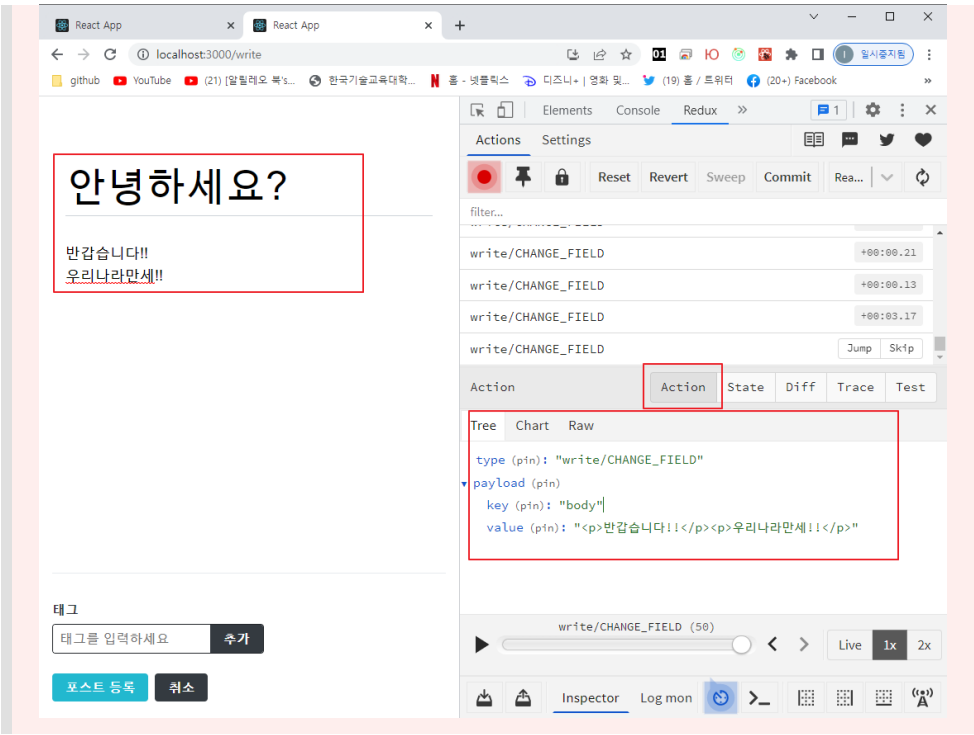
components/write/Editor.js

```
import React, { useRef, useEffect } from 'react';
import Quill from 'quill';
import 'quill/dist/quill.bubble.css';
import styled from 'styled-components';
import palette from '../../lib/styles/palette';
import Responsive from '../common/Responsive';

const EditorBlock = styled(Responsive)`
  /* 페이지 위 아래 여백 지정 */
  padding-top: 5rem;
  padding-bottom: 5rem;
```

```
`;
const TitleInput = styled.input`
  font-size: 3rem;
  outline: none;
  padding-bottom: 0.5rem;
  border: none;
  border-bottom: 1px solid ${palette.gray[4]};
  margin-bottom: 2rem;
  width: 100%;
`;
const QuillWrapper = styled.div`
  /* 최소 크기 지정 및 padding 제거 */
  .ql-editor {
    padding: 0;
    min-height: 320px;
    font-size: 1.125rem;
    line-height: 1.5;
  }
  .ql-editor.ql-blank::before {
    left: 0px;
  }
`;

const Editor = ({ title, body, onChangeField }) => {
  const quillElement = useRef(null); // Quill을 적용할 DivElement를 설정
  const quillInstance = useRef(null); // Quill 인스턴스를 설정

  useEffect(() => {
    quillInstance.current = new Quill(quillElement.current, {
      theme: 'bubble',
      placeholder: '내용을 작성하세요...',
      modules: {
        // 더 많은 옵션
        // https://quilljs.com/docs/modules/toolbar/ 참고
        toolbar: [
          [{ header: '1' }, { header: '2' }],
          ['bold', 'italic', 'underline', 'strike'],
          [{ list: 'ordered' }, { list: 'bullet' }],
          ['blockquote', 'code-block', 'link', 'image'],
        ],
      },
    });
    // quill에 text-change 이벤트 핸들러 등록
    // 참고: https://quilljs.com/docs/api/#events
    const quill = quillInstance.current;
    quill.on('text-change', (delta, oldDelta, source) => {
      if (source === 'user') {
        onChangeField({ key: 'body', value: quill.root.innerHTML });
      }
    });
  }, [onChangeField]);

  const onChangeTitle = e => {
    onChangeField({ key: 'title', value: e.target.value });
  };

  return (
```

```
      <EditorBlock>
        <TitleInput
          placeholder="제목을 입력하세요"
          onChange={onChangeTitle}
          value={title}
        />
        <QuillWrapper>
          <div ref={quillElement} />
        </QuillWrapper>
      </EditorBlock>
    );
};

export default Editor;
```

- 리덕스스토어확인하려면 프로젝트 리스타트!!



25.3.2 TagBoxContainer만들기

containers/write/TagBoxContainer.js

```
import React from 'react';
import { useDispatch, useSelector } from 'react-redux';
import TagBox from '../../components/write/TagBox';
import { changeField } from '../../modules/write';

const TagBoxContainer = () => {
  const dispatch = useDispatch();
  const tags = useSelector(state => state.write.tags);

  const onChangeTags = nextTags => {
    dispatch(
      changeField({
        key: 'tags',
        value: nextTags,
      }),
```

```
    );
  };

  return <TagBox onChangeTags={onChangeTags} tags={tags} />;
};

export default TagBoxContainer;
```

pages/WritePage.js - TagBox를 TagBoxContainer 대체

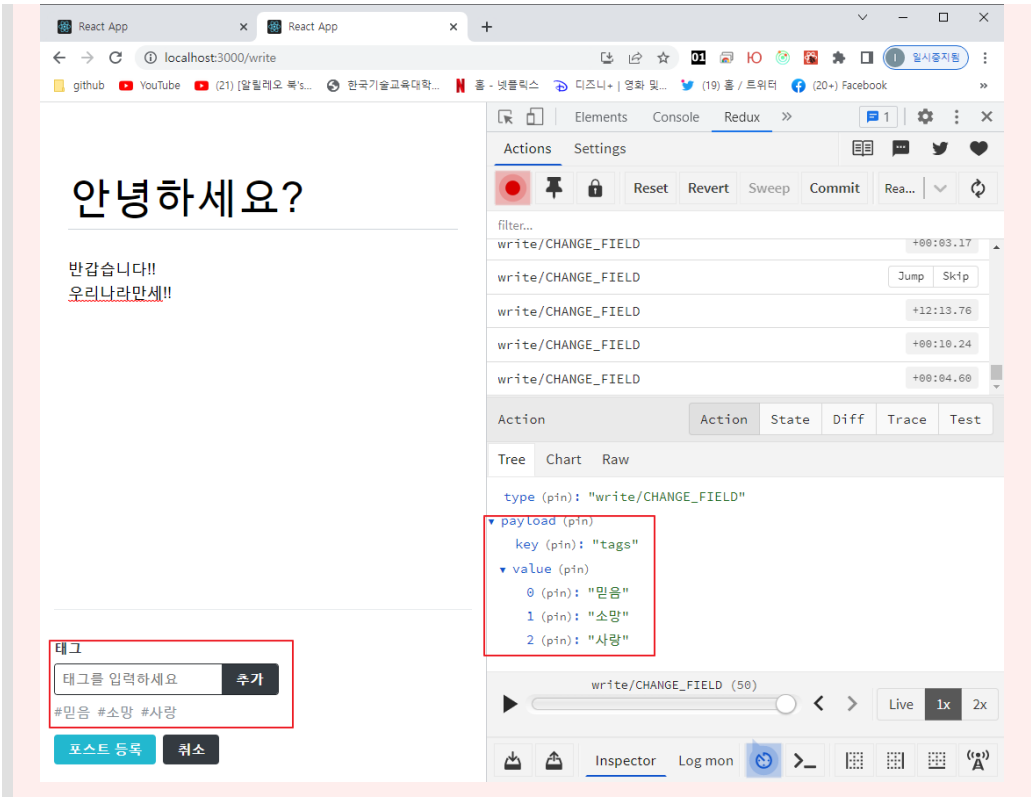
```
// import Editor from '../components/write/Editor';
// import TagBox from '../components/write/TagBox';
import Responsive from '../components/common/Responsive';
import WriteActionButtons from '../components/write/WriteActionButtons';
import EditorContainer from '../containers/write/EditorContainer';
import TagBoxContainer from '../containers/write/TagBoxContainer';

const WritePage = () => {
  return (
    <Responsive>
      {/* <Editor /> */}
      {/* <TagBox /> */}
      <EditorContainer />
      <TagBoxContainer />
      <WriteActionButtons />
    </Responsive>
  );
};

export default WritePage;
```

components/write/TagBox.js

```
import React, { useState, useCallback, useEffect } from 'react';
import styled from 'styled-components';
import palette from '../../lib/styles/palette';

const TagBoxBlock = styled.div`
  width: 100%;
  border-top: 1px solid ${palette.gray[2]};
  padding-top: 2rem;

  h4 {
    color: ${palette.gray[8]};
    margin-top: 0;
    margin-bottom: 0.5rem;
  }
`;

const TagForm = styled.form`
  border-radius: 4px;
  overflow: hidden;
  display: flex;
  width: 256px;
  border: 1px solid ${palette.gray[9]}; /* 스타일 초기화 */
  input,
  button {
    outline: none;
```

```
    border: none;
    font-size: 1rem;
  }

  input {
    padding: 0.5rem;
    flex: 1;
    min-width: 0;
  }

  button {
    cursor: pointer;
    padding-right: 1rem;
    padding-left: 1rem;
    border: none;
    background: ${palette.gray[8]};
    color: white;
    font-weight: bold;
    &:hover {
      background: ${palette.gray[6]};
    }
  }
`;

const Tag = styled.div`
  margin-right: 0.5rem;
  color: ${palette.gray[6]};
  cursor: pointer;
  &:hover {
    opacity: 0.5;
  }
`;

const TagListBlock = styled.div`
  display: flex;
  margin-top: 0.5rem;
`;

// React.memo를 사용하여 tag 값이 바뀔 때만 리렌더링되도록 처리
const TagItem = React.memo(({ tag, onRemove, onChangeTags }) => (
  <Tag onClick={() => onRemove(tag)}>#{tag}</Tag>
));

// React.memo를 사용하여 tags 값이 바뀔 때만 리렌더링되도록 처리
const TagList = React.memo(({ tags, onRemove }) => (
  <TagListBlock>
    {tags.map(tag => (
      <TagItem key={tag} tag={tag} onRemove={onRemove} />
    ))}
  </TagListBlock>
));


const TagBox = ({ tags, onChangeTags }) => {
  const [input, setInput] = useState('');
  const [localTags, setLocalTags] = useState([]);
```

```
  const insertTag = useCallback(
    tag => {
      if (!tag) return; // 공백이라면 추가하지 않음
      if (localTags.includes(tag)) return; // 이미 존재한다면 추가하지 않음
      const nextTags = [...localTags, tag];
      setLocalTags(nextTags);
      onChangeTags(nextTags);
    },
    [localTags, onChangeTags],
  );

  const onRemove = useCallback(
    tag => {
      const nextTags = localTags.filter(t => t !== tag);
      setLocalTags(nextTags);
      onChangeTags(nextTags);
    },
    [localTags, onChangeTags],
  );

  const onChange = useCallback(e => {
    setInput(e.target.value);
  }, []);

  const onSubmit = useCallback(
    e => {
      e.preventDefault();
      insertTag(input.trim()); // 앞뒤 공백 없앤 후 등록
      setInput(''); // input 초기화
    },
    [input, insertTag],
  );

  // tags 값이 바뀔 때
  useEffect(() => {
    setLocalTags(tags);
  }, [tags]);

  return (
    <TagBoxBlock>
      <h4>태그</h4>
      <TagForm onSubmit={onSubmit}>
        <input
          placeholder="태그를 입력하세요"
          value={input}
          onChange={onChange}
        />
        <button type="submit">추가</button>
      </TagForm>
      <TagList tags={localTags} onRemove={onRemove} />
    </TagBoxBlock>
  );
};

export default TagBox;
```



25.3.3 글쓰기 API 연동하기

lib/api/posts.js

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

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

modules/write.js - writePost함수 호출 및 리덕스액션, saga 준비하기

```
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 INITIALIZE = 'write/INITIALIZE'; // 모든 내용 초기화
const CHANGE_FIELD = 'write/CHANGE_FIELD'; // 특정 key 값 바꾸기
const [
  WRITE_POST,
  WRITE_POST_SUCCESS,
  WRITE_POST_FAILURE,
] = createRequestActionTypes('write/WRITE_POST'); // 포스트 작성

export const initialize = createAction(INITIALIZE);
export const changeField = createAction(CHANGE_FIELD, ({ key, value }) => ({
  key,
  value,
}));
export const writePost = createAction(WRITE_POST, ({ title, body, tags }) =>
({
  title,
  body,
```

```
    tags,
  }));

  // saga 생성
  const writePostSaga = createRequestSaga(WRITE_POST, postsAPI.writePost);
  export function* writeSaga() {
    yield takeLatest(WRITE_POST, writePostSaga);
  }

  const initialState = {
    title: '',
    body: '',
    tags: [],
    post: null,
    postError: null,
  };

  const write = handleActions(
    {
      [INITIALIZE]: state => initialState, // initialState를 넣으면 초기상태로 바
  꿈
      [CHANGE_FIELD]: (state, { payload: { key, value } }) => ({
        ...state,
        [key]: value, // 특정 key 값을 업데이트
      }),
      [WRITE_POST]: state => ({
        ...state,
        // post와 postError를 초기화
        post: null,
        postError: null,
      }),
      // 포스트 작성 성공
      [WRITE_POST_SUCCESS]: (state, { payload: post }) => ({
        ...state,
        post,
      }),
      // 포스트 작성 실패
      [WRITE_POST_FAILURE]: (state, { payload: postError }) => ({
        ...state,
        postError,
      }),
    },
    initialState,
  );

  export default write;
```

modules/index.js - writeSaga를 rootSaga에 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';

  const rootReducer = combineReducers({
```

```
  auth,
  loading,
  user,
  write
});

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

export default rootReducer;
```

containers/write/WriteActionButtonsContainer.js

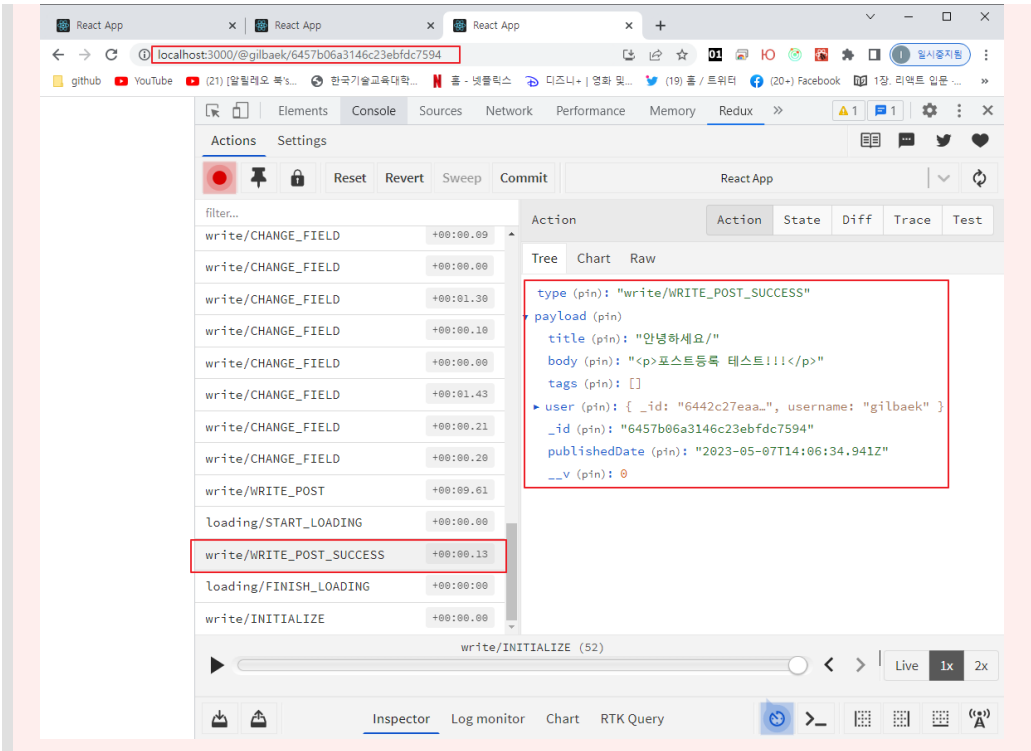
- 포스트등록버튼을 클릭하면 현재 리덕스스토어값이 새 포스트로 등록
- 취소버튼은 뒤로가기, 라우터가 아닌 컴퍼넌트에서 history객체를 사요하기 위해 useNavigatro를 사용
- 포스트작성 성공시 시버의 응답포스트의 _id와 username값을 참조하여 포스트경로 생서후 history.push사용 이동

```
import React, { useEffect } from 'react';
import WriteActionButtons from '../../components/write/WriteActionButtons';
import { useSelector, useDispatch } from 'react-redux';
import { useNavigate } from 'react-router-dom';
import { writePost } from '../../modules/write';

const WriteActionButtonsContainer = () => {
  const navigate = useNavigate();
  const dispatch = useDispatch();
  const { title, body, tags, post, postError } = useSelector(({ write }) => ({
    title: write.title,
    body: write.body,
    tags: write.tags,
    post: write.post,
    postError: write.postError,
  }));

  // 포스트 등록
  const onPublish = () => {
    dispatch(
      writePost({
        title,
        body,
        tags,
      }),
    );
  };

  // 취소
  const onCancel = () => {
    navigate(-1);
  };

  // 성공 혹은 실패시 할 작업
  useEffect(() => {
    if (post) {
```

```
      const { _id, user } = post;
      navigate(`/@${user.username}/${_id}`);
    }
    if (postError) {
      console.log(postError);
    }
  }, [navigate, post, postError]);
  return <WriteActionButtons onPublish={onPublish} onCancel={onCancel} />;
};

export default WriteActionButtonsContainer;
```

pages/WritePage.js - WriteActionButtonsContainer등록

```
// import Editor from '../components/write/Editor';
// import TagBox from '../components/write/TagBox';
// import WriteActionButtons from '../components/write/WriteActionButtons';
import Responsive from '../components/common/Responsive';
import EditorContainer from '../containers/write/EditorContainer';
import TagBoxContainer from '../containers/write/TagBoxContainer';
import WriteActionButtonsContainer from
'../containers/write/WriteActionButtonsContainer';

const WritePage = () => {
  return (
    <Responsive>
      {/* <Editor /> */}
      {/* <TagBox /> */}
      {/* <WriteActionButtons /> */}
      <EditorContainer />
      <TagBoxContainer />
      <WriteActionButtonsContainer />
    </Responsive>
  );
};

export default WritePage;
```


React App
localhost:3000/@gilbaek/6457b06a3146c23ebfdc7594
Elements
Console
Sources
Network
Performance
Memory
Redux
Actions
Settings
Reset
Revert
Sweep
Commit
React App
filter...
write/CHANGE_FIELD +00:00.09
write/CHANGE_FIELD +00:00.00
write/CHANGE_FIELD +00:01.30
write/CHANGE_FIELD +00:00.10
write/CHANGE_FIELD +00:00.00
write/CHANGE_FIELD +00:01.43
write/CHANGE_FIELD +00:00.21
write/CHANGE_FIELD +00:00.20
write/WRITE_POST +00:09.61
loading/START_LOADING +00:00.00
write/WRITE_POST_SUCCESS +00:00.13
loading/FINISH_LOADING +00:00:00
write/INITIALIZE +00:00.00
Action
Action
State
Diff
Trace
Test
Tree
Chart
Raw
type (pin): "write/WRITE_POST_SUCCESS"
payload (pin)
title (pin): "안녕하세요/"
body (pin): " 포스트등록 테스트!!! "
tags (pin): []
user (pin): { _id: "6442c27eaa…", username: "gilbaek" }
_id (pin): "6457b06a3146c23ebfdc7594"
publishedDate (pin): "2023-05-07T14:06:34.941Z"
__v (pin): 0
write/INITIALIZE (52)
Live
1x
2x
Inspector
Log monitor
Chart
RTK Query