27.수정/삭제기능 구현 및 마무리

27.1 포스트수정

27.1.1 PostActionButtons 컴퍼넌트 만들기

components/post/PostActionButtons.js

```
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;
```

components/post/PostViewer.js

```
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, actionButtons }) => {
  // 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>
      { actionButtons }
      <PostContent dangerouslySetInnerHTML={{ __html: body }} />
    </PostViewerBlock>
  );
};

export default PostViewer;
```

container/post/PostViewerContainer .js

```
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';
import PostActionButtons from '../../components/post/PostActionButtons';

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}
      actionButtons={ <PostActionButtons />
      }
    />
  );
};
```

```
export default PostViewerContainer;
```

27.1.2 수정버튼클릭 - 글쓰기 페이지 이동

modules/write.js

```
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
const SET_ORIGINAL_POST = 'write/SET_ORIGINAL_POST';

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,
}));
export const setOriginalPost = createAction(SET_ORIGINAL_POST, post => post);

// 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,
  originalPostId: null,
};

const write = handleActions(
  {
    [INITIALIZE]: state => initialState, // initialState를 넣으면 초기상태로 바
뀜
```

```
    [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,
    }),
    [SET_ORIGINAL_POST]: (state, { payload: post }) => ({
      ...state,
      title: post.title,
      body: post.body,
      tags: post.tags,
      originalPostId: post._id,
    }),
  },
  initialState,
);

export default write;
```

container/post/PostViewerContainer.js - 글쓰기, 글삭제

```
import React, { useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { readPost, unloadPost } from '../../modules/post';
import PostViewer from '../../components/post/PostViewer';
import PostActionButtons from '../../components/post/PostActionButtons';
import { setOriginalPost } from '../../modules/write';
import { removePost } from '../../lib/api/posts';
import { useParams, useNavigate } from 'react-router-dom';

const PostViewerContainer = () => {
  // 처음 마운트될 때 포스트 읽기 API 요청
  const { postId } = useParams();
  const navigate = useNavigate();
  const dispatch = useDispatch();
  const { post, error, loading, user } = useSelector(
    ({ post, loading, user }) => ({
      post: post.post,
      error: post.error,
      loading: loading['post/READ_POST'],
      user: user.user,
    }),
  );
  useEffect(() => {
    dispatch(readPost(postId));
    // 언마운트될 때 리덕스에서 포스트 데이터 없애기
    return () => {
      dispatch(unloadPost());
    };
  }, [dispatch, postId]);

  const onEdit = () => {
    dispatch(setOriginalPost(post));
    navigate('/write');
  };

  const onRemove = async () => {
    try {
      await removePost(postId);
      navigate('/'); // 홈으로 이동
    } catch (e) {
      console.log(e);
    }
  };

  const ownPost = (user && user._id) === (post && post.user._id);

  return (
    <PostViewer
      post={post}
      loading={loading}
      error={error}
      actionButtons={
        ownPost && <PostActionButtons onEdit={onEdit} onRemove={onRemove} />
      }
    />
  );
};

export default PostViewerContainer;
```

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 mounted = useRef(false);
  useEffect(() => {
    if (mounted.current) return;
    mounted.current = true;
    quillInstance.current.root.innerHTML = body;
  }, [body]);

  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;
```

- Editor에서 Quill에디터에서 body를 변경할 때마다 useEffect에 등록된 함수가 호출된ㄷ.
- 하지만 컴퍼넌트가 마운트되고 나서 단 한번만 useEffect의 작업이 실행되도록 해야 한다 방법은 2가지방법이 있다.
  1. useEffect의 두번째 파라미터에 빈배혈([])을 넣는 방법
     - 하지만 ESLint규칙은 useEffect에서 사용하는 모든 외부값을 두 번째 파라미터에 넣어서 처리를 권장한다.
  2. 따라서 2 번째 방법은 해당줄만 ESLint규칙을 비활성화 하는 방법도 있다.(여기서 채택한 방법)

     ```
     useEffect(() => {
        quillInstance.current.root.innerHTML = body;
      }, []); /* ellint-disabled-line */
     ```
- 취향에 따라 바업을 선태하면된다.
- 수정버튼을 클릭

lib/api/posts.js - updatePost

```
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 },
  });
};

export const updatePost = ({ id, title, body, tags }) =>
  client.patch(`/api/posts/${id}`, {
    title,
    body,
    tags,
```

```
  });

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

modules/write.js - updatePost

```
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
const SET_ORIGINAL_POST = 'write/SET_ORIGINAL_POST';

const [
  UPDATE_POST,
  UPDATE_POST_SUCCESS,
  UPDATE_POST_FAILURE,
] = createRequestActionTypes('write/UPDATE_POST');

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,
}));
export const setOriginalPost = createAction(SET_ORIGINAL_POST, post => post);

 // 포스트 수정
 export const updatePost = createAction(
  UPDATE_POST,
  ({ id, title, body, tags }) => ({
    id,
    title,
    body,
    tags,
  }),
);

// saga 생성
const writePostSaga = createRequestSaga(WRITE_POST, postsAPI.writePost);
const updatePostSaga = createRequestSaga(UPDATE_POST, postsAPI.updatePost);

export function* writeSaga() {
  yield takeLatest(WRITE_POST, writePostSaga);
  yield takeLatest(UPDATE_POST, updatePostSaga);
}

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

const write = handleActions(
  {
    [INITIALIZE]: state => initialState, // initialState를 넣으면 초기상태로 바
뀜
    [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,
    }),
    [SET_ORIGINAL_POST]: (state, { payload: post }) => ({
      ...state,
      title: post.title,
      body: post.body,
      tags: post.tags,
      originalPostId: post._id,
    }),
    [UPDATE_POST_SUCCESS]: (state, { payload: post }) => ({
      ...state,
      post,
    }),
    [UPDATE_POST_FAILURE]: (state, { payload: postError }) => ({
      ...state,
      postError,
    }),
  },
  initialState,
);

export default write;
```

containers/write/WriteActionButtonsContainer.js - updatePost

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

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

  // 포스트 등록
  const onPublish = () => {
    if (originalPostId) {
      dispatch(updatePost({ title, body, tags, id: originalPostId }));
      return;
    }
    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}
      isEdit={!!originalPostId}
    />
  );
};

export default WriteActionButtonsContainer;
```

containers/write/WriteActionButtonsContainer.js - updatePost

```
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, isEdit }) => {
  return (
    <WriteActionButtonsBlock>
      <StyledButton cyan onClick={onPublish}>
        포스트 {isEdit ? '수정' : '등록'}
      </StyledButton>
      <StyledButton onClick={onCancel}>취소</StyledButton>
    </WriteActionButtonsBlock>
  );
};

export default WriteActionButtons;
```

27.2 포스트 삭제

components/common/AscModal.js

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

const Fullscreen = styled.div`
  position: fixed;
  z-index: 30;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: rgba(0, 0, 0, 0.25);
  display: flex;
  justify-content: center;
  align-items: center;
`;
const AskModalBlock = styled.div`
  width: 320px;
  background: white;
  padding: 1.5rem;
  border-radius: 4px;
  box-shadow: 0px 0px 8px rgba(0, 0, 0, 0.125);
  h2 {
    margin-top: 0;
    margin-bottom: 1rem;
  }
  p {
    margin-bottom: 3rem;
  }
  .buttons {
    display: flex;
    justify-content: flex-end;
  }
`;

const StyledButton = styled(Button)`
  height: 2rem;
  & + & {
    margin-left: 0.75rem;
  }
`;

const AskModal = ({
  visible,
  title,
  description,
  confirmText = '확인',
  cancelText = '취소',
  onConfirm,
  onCancel,
}) => {
  if (!visible) return null;
  return (
    <Fullscreen>
      <AskModalBlock>
        <h2>{title}</h2>
        <p>{description}</p>
        <div className="buttons">
          <StyledButton onClick={onCancel}>{cancelText}</StyledButton>
          <StyledButton cyan onClick={onConfirm}>
            {confirmText}
          </StyledButton>
        </div>
      </AskModalBlock>
    </Fullscreen>
  );
};

export default AskModal;
```

components/post/AskRemoveModal.js

```
import React from 'react';
import AskModal from '../common/AskModal';

const AskRemoveModal = ({ visible, onConfirm, onCancel }) => {
  return (
    <AskModal
      visible={visible}
      title="포스트 삭제"
      description="포스트를 정말 삭제하시겠습니까?"
      confirmText="삭제"
      onConfirm={onConfirm}
      onCancel={onCancel}
    />
  );
};

export default AskRemoveModal;
```

component/post/PostActionButtons.js - AskRemoveModsal

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

const PostActionButtonsBlock = styled.div`
  display: flex;
  justify-content: flex-end;
  margin-bottom: 2rem;
  margin-top: -1.5rem;
`;

const ActionButton = styled.button`
  padding: 0.25rem 0.5rem;
  border-radius: 4px;
  color: ${palette.gray[6]};
  font-weight: bold;
  border: none;
  outline: none;
  font-size: 0.875rem;
  cursor: pointer;
  &:hover {
    background: ${palette.gray[1]};
    color: ${palette.cyan[7]};
  }
  & + & {
    margin-left: 0.25rem;
  }
`;

const PostActionButtons = ({ onEdit, onRemove }) => {
  const [modal, setModal] = useState(false);
  const onRemoveClick = () => {
    setModal(true);
  };
  const onCancel = () => {
    setModal(false);
  };
  const onConfirm = () => {
    setModal(false);
    onRemove();
  };

  return (
    <>
      <PostActionButtonsBlock>
        <ActionButton onClick={onEdit}>수정</ActionButton>
        <ActionButton onClick={onRemoveClick}>삭제</ActionButton>
      </PostActionButtonsBlock>
      <AskRemoveModal
        visible={modal}
        onConfirm={onConfirm}
        onCancel={onCancel}
      />
    </>
  );
};

export default PostActionButtons;
```

lib/api/posts.js

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

containes/post/PostViewerContainer.js - onRemove

```
import React, { useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { readPost, unloadPost } from '../../modules/post';
import PostViewer from '../../components/post/PostViewer';
import PostActionButtons from '../../components/post/PostActionButtons';
import { setOriginalPost } from '../../modules/write';
import { removePost } from '../../lib/api/posts';
import { useParams, useNavigate } from 'react-router-dom';

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

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

  const onEdit = () => {
    dispatch(setOriginalPost(post));
    navigate('/write');
  };

  const onRemove = async () => {
    try {
      await removePost(postId);
      navigate('/'); // 홈으로 이동
    } catch (e) {
      console.log(e);
    }
  };

  const ownPost = (user && user._id) === (post && post.user._id);

  return (
    <PostViewer
      post={post}
      loading={loading}
      error={error}
      actionButtons={
        ownPost && <PostActionButtons onEdit={onEdit} onRemove={onRemove} />
      }
    />
  );
};

export default PostViewerContainer;
```

27.3 react-helmet-async로 meta태그 설정하기

- title탭에 React App 제목의 meta태그 정보 수정하기
- 설치 : yarn add react-helmet-async
- HelmetProvider 적용
  - import { HelmetProvider } from 'react-helmet-async';
  - index.js, App.js, pages/WritePage.js, compoents/post/PostViewer.js

27.4 프로젝트 마무리

27.4.1 프로젝트 빌드

- yarn build
  - 작업이 끝나면 `forntend/build` 디렉토리생성

27.4.2 koa-statc으로 정적파일제공

- backend 에서 `yarn add koa-static` 설치
- src/main.js 수정

src/main.js

```
require('dotenv').config();
import Koa from 'koa';
import Router from 'koa-router';
import bodyParser from 'koa-bodyparser';
import mongoose from 'mongoose';
import serve from 'koa-static';
import path from 'path';
import send from 'koa-send';

import api from './api';
import jwtMiddleware from './lib/jwtMiddleware';

// import createFakeData from './createFakeData';

// 비구조화 할당을 통해 process.env내부값에 대한 레퍼런스 만들기
const { PORT, MONGO_URI } = process.env;

mongoose
  .connect(MONGO_URI)
  .then(() => {
    console.log("Conntected to MongoDB");
    // createFakeData();
  })
  .catch(e => {
    console.error(e);
})

const app = new Koa();
const router = new Router();

// 라우터설정
router.use('/api', api.routes()); // api 라우트 적용

// 라우터적용전에 bodyParser적용
app.use(bodyParser());
app.use(jwtMiddleware);

// app 인스턴스에 라우터 적용
app.use(router.routes()).use(router.allowedMethods());

const buildDirectory = path.resolve(__dirname, '../../frontend/build');
app.use(serve(buildDirectory));
app.use(async ctx => {
  // Not Found 이고, 주소가 /api 로 시작하지 않는 경우
  if (ctx.status === 404 && ctx.path.indexOf('/api') !== 0) {
    // index.html 내용을 반환
    await send(ctx, 'index.html', { root: buildDirectory });
  }
});

// PORT가 지정되어 있지 않다면 4000사용
const port = PORT || 4000;
```

```
app.listen(port, () => {
  console.log('Listening to port %d', port)
});
```

- 서버프로그램실행
  - backend\yarn start
  - 서버접속 : 웹브라우저 주소 : localhost:4000

27.5 더 할 수 있는 작업

1. 코드 스플리팅
2. 서버 호스팅
   - AWS EC2 : https://aws.amazon.com
   - Google Cloud : https://cloud.google.com/compute/pricing
   - NCloud Compute : https://www.ncloud.com/product/compute
   - Vultr : https://www.vultr.com/pricing
3. SSR

```
import client from './lib/api/client';
client.defults.baseURL = 'http://localhost:4000';
```

- nginx 사용예시 (성능상 더 빠름)

```
server {
listen  8080;
server_name localhost;

location /api/ {
 proxy_pass http://localhost:4000;
 proxy_set_header Upgrade $jttp_upgrade;
 proxy_set_header Connction 'upgrade';
 proxy_set_header Host $host;
 proxy_cache_nypass $http_upgrade;
}

location /static {
 alias /01.MYREACT/27.final/frontend/build/static;
}

location / {
 proxy_pass http://localhost:5000;
 proxy_set_header Upgrade $jttp_upgrade;
 proxy_set_header Connction 'upgrade';
 proxy_set_header Host $host;
 proxy_cache_nypass $http_upgrade;
}
}
```