[스파르타코딩클럽] 프론트엔드의 꽃, 리액트 - 5주차

매 주차 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/5d3a396f-7013-4433-8b36-c13787b5de88/____5_.pdf

[수업 목표]

1. 비동기 통신을 해본다.
2. AWS의 S3 버킷으로 정적 웹사이트 호스팅을 할 수 있다.
3. Firebase로 배포할 수 있다.
4. 도메인을 연결한다.

[목차]

💡 모든 토글을 열고 닫는 단축키
Windows : `Ctrl` + `alt` + `t`
Mac : `⌘` + `⌥` + `t`

---

01. 리덕스에서 FireStore 데이터 가지고 놀기 (1)

🔥 버킷리스트 프로젝트에 firestore를 적용해볼거예요!

▼ 1) firestore 데이터를 리덕스 스토어에 넣으려면? (미들웨어 설치!)

🔥 우리가 firestore에서 데이터를 가져올 때 비동기 통신을 한다고 했죠!
리덕스에서 **비동기 통신을 할 때 필요한 미들웨어**라는 친구 먼저 설치할 거예요.

👉 미들웨어가 뭐냐구요?
리덕스 데이터를 수정할 때 **[액션이 디스패치 되고 → 리듀서에서 처리]** 하던 과정 기억하시죠?
미들웨어는 이 과정 사이에 미리 사전 작업을 할 수 있도록 하는 중간 다리 같은 거예요!
즉! **[액션이 일어나고 → 미들웨어가 할 일 하기 → 리듀서에서 처리]** 이 순서로 처리하게 됩니다!

```
yarn add redux-thunk
```

👉 **redux-thunk**는 뭐하는 미들웨어일까?
우리 액션 생성 함수가 뭘 반환한다고 했었죠? 맞아요! 객체 반환하죠. 🙂
redux-thunk는 **객체 대신 함수를 생성하는 액션 생성함수**를 작성할 수 있게 해줍니다!

그게 왜 필요하냐구요?
리덕스는 기본적으로는 **액션 객체를 디스패치**합니다! → 즉, 함수를 생성하면 특정 액션이 발생하기 전에 조건을 주거나, 어떤 행동을 사전에 처리할 수 있겠죠!!

설치가 끝났다면! **configStore.js에 미들웨어를 추가**해봅시다! 그럼 준비 끝!

```
import { createStore, combineReducers, applyMiddleware, compose } from "redux";
import thunk from "redux-thunk";
import bucket from "./modules/bucket";
import { createBrowserHistory } from "history";
```

```
export const history = createBrowserHistory();

const middlewares = [thunk];

const enhancer = applyMiddleware(...middlewares);
const rootReducer = combineReducers({ bucket });
const store = createStore(rootReducer, enhancer);

export default store;
```

02. 리덕스에서 FireStore 데이터 가지고 놀기 (2)

▼ 2) firestore 적용하기

▼ (1) load할 때 데이터를 가지고 와보자!

> 👉 만들고 안썼던 기능이 하나 있죠? loadBucket 액션 생성 함수! 드디어 사용할 차례입니다!!

▼ 파이어베이스랑 통신하는 함수 만들고,

```
const bucket_db = firestore.collection("bucket");

// 파이어베이스랑 통신하는 부분
export const loadBucketFB = () => {
  return function (dispatch) {

    bucket_db.get().then((docs) => {
      let bucket_data = [];
      docs.forEach((doc) => {
        // 도큐먼트 객체를 확인해보자!
        console.log(doc);
        // 도큐먼트 데이터 가져오기
        console.log(doc.data());
        // 도큐먼트 id 가져오기
        console.log(doc.id);

        if(doc.exists){
          bucket_data = [...bucket_data, {id: doc.id, ...doc.data()}];
        }
      });

      console.log(bucket_data);
      // 이제 액션 생성 함수한테 우리가 가져온 데이터를 넘겨줘요! 그러면 끝!
      dispatch(loadBucket(bucket_data));
    });
  };
};
```

▼ 리듀서를 고쳐요!

```
case "bucket/LOAD": {
      if(action.bucket.length >0){
        return { list: action.bucket };
      }

      return state;
    }
```

▼ 그 후에는요? 불러다 쓰면 되겠죠!

```
// App.js
...
// 잠깐!! loadBucketFB를 import해오는 거 잊지말기!
// load()를 componentDidMount에서 부르면 되겠죠? :)
const mapDispatchToProps = (dispatch) => ({
  load: () => {
    dispatch(loadBucketFB());
  },
  create: (new_item) => {
    console.log(new_item);
    dispatch(createBucket(new_item));
  }
});
...
```

▼ (2) create에 firestore 적용

👉 순서는 항상 똑같을 거예요!
파이어베이스랑 통신 → 필요하다면 리듀서 고치고 → 불러다 쓰기

```
bucket.add({ text: "수영 배우기", completed: false });
```

▼ 파이어베이스랑 통신하는 함수 만들고,

```
// 파이어베이스랑 통신하는 부분
export const addBucketFB = (bucket) => {
  return function (dispatch) {
    console.log(bucket);
    // 생성할 데이터를 미리 만들게요!
    let bucket_data = { text: bucket, completed: false };

    // add()에 데이터를 넘겨줍시다!
    bucket_db
      .add(bucket_data)
      .then((docRef) => {
        // id를 추가한다!
        bucket_data = { ...bucket_data, id: docRef.id };

        console.log(bucket_data);

        // 성공했을 때는? 액션 디스패치!
        dispatch(createBucket(bucket_data));
      })
      .catch((err) => {
        // 여긴 에러가 났을 때 들어오는 구간입니다!
```

```
        console.log(err);
        window.alert('오류가 났네요! 나중에 다시 시도해주세요!');
      });
  };
};
```

▼ 리듀서를 고쳐요!

```
case "bucket/CREATE": {
      const new_bucket_list = [
        ...state.list,
        action.bucket,
      ];
      return { list: new_bucket_list };
    }
```

▼ 그 후에는요? 불러다 쓰면 되겠죠!

```
// App.js
...
// 잠깐!! addBucketFB를 import해오는 거 잊지말기!
const mapDispatchToProps = (dispatch) => ({
  load: () => {
    dispatch(loadBucketFB());
  },
  create: (new_item) => {
    console.log(new_item);
    dispatch(addBucketFB(new_item));
  }
});
...
```

▼ (3) update에 firestore 적용

👉 순서는 항상 똑같을 거예요!
파이어베이스랑 통신 → 필요하다면 리듀서 고치고 → 불러다 쓰기

```
bucket.doc([id]).update({ text: "수영 배우기", completed: false });
```

▼ 파이어베이스랑 통신하는 함수 만들고,

```
// 파이어베이스랑 통신하는 부분
export const updateBucketFB = (bucket) => {
  return function (dispatch, getState) {
    // state에 있는 값을 가져옵니다!
    const _bucket_data = getState().bucket.list[bucket];

    // id가 없으면? 바로 끝내버립시다.
    if (!_bucket_data.id) {
      return;
    }

    // 새로운 값을 만들어요!
```

```
    let bucket_data = { ..._bucket_data, completed: true };

    bucket_db
      .doc(bucket_data.id)
      .update(bucket_data)
      .then((res) => {
        dispatch(updateBucket(bucket));
      })
      .catch((err) => {
        console.log("err");
      });
  };
};
```

▼ 그 후에는요? 불러다 쓰면 되겠죠!

```
// Detail.js
...
// 잠깐!! updateBucketFB를 import해오는 거 잊지말기!
<button onClick={() => {
          dispatch(updateBucketFB(bucket_index));
          props.history.goBack();
      }}>완료하기</button>
...
```

▼ (4) delete에 firestore 적용

👉 순서는 항상 똑같을 거예요!
파이어베이스랑 통신 → 필요하다면 리듀서 고치고 → 불러다 쓰기

```
bucket.doc([id]).delete();
```

▼ 파이어베이스랑 통신하는 함수 만들고,

```
// 파이어베이스랑 통신하는 부분
export const deleteBucketFB = (bucket) => {
  return function (dispatch, getState) {
    const _bucket_data = getState().bucket.list[bucket];
    // id가 없으면? 바로 끝내버립시다.
    if (!_bucket_data.id) {
      return;
    }
    // 삭제하기
    bucket_db
      .doc(_bucket_data.id)
      .delete()
      .then((res) => {
        dispatch(deleteBucket(bucket));
      })
      .catch((err) => {
        console.log("err");
      });
  };
};
```

▼ 그 후에는요? 불러다 쓰면 되겠죠!

```
// Detail.js
...
<button onClick={() => {
        //   dispatch(); <- 괄호안에는 액션 생성 함수가 들어가야겠죠?
        // 예를 들면 이렇게요.
        dispatch(deleteBucketFB(bucket_index));
        props.history.goBack();
      }}>삭제하기</button>
...
```

03. 머테리얼 UI 사용하기

▼ 3) 부트스트랩처럼 이미 다 만들어진 ui를 가져다 써보자!

👉 머테리얼 ui는 우리가 styled-components를 쓰던 것처럼 사용할 수 있어요.
공식 문서(https://material-ui.com/)에서 어떻게 생겼는 지 보고 사용 해봅시다!

▼ (1) 머테리얼 UI 설치하기

```
yarn add @material-ui/core @material-ui/icons
```

▼ (2) 버킷리스트 프로젝트 중 <Detail/>! 머테리얼 UI를 사용해서 고쳐봅시다!

```
// 리액트 패키지를 불러옵니다.
import React from "react";
import styled from "styled-components";
import Button from "@material-ui/core/Button";
import ButtonGroup from "@material-ui/core/ButtonGroup";

// redux hook을 불러옵니다.
import { useDispatch, useSelector } from "react-redux";
// 내가 만든 액션 생성 함수를 불러옵니다.
import {
  deleteBucket,
  updateBucket,
  deleteBucketFB,
  updateBucketFB,
} from "./redux/modules/bucket";

const Detail = (props) => {
    const dispatch = useDispatch();


  // 스토어에서 상태값 가져오기
  const bucket_list = useSelector((state) => state.bucket.list);
  // url 파라미터에서 인덱스 가져오기
  let bucket_index = parseInt(props.match.params.index);

  console.log(props);
  return (
    <div>
```

```
      <h1>{bucket_list[bucket_index].text}</h1>
      <ButtonGroup>
        <Button
          variant="outlined"
          onClick={() => {
            //   dispatch(); <- 괄호안에는 액션 생성 함수가 들어가야겠죠?
            // 예를 들면 이렇게요.
            dispatch(deleteBucketFB(bucket_index));
            props.history.goBack();
          }}
        >
          삭제하기
        </Button>
        <Button
          variant="outlined"
          color="primary"
          onClick={() => {
            dispatch(updateBucketFB(bucket_index));
            props.history.goBack();
          }}
        >
          완료하기
        </Button>
      </ButtonGroup>
    </div>
  );
};

export default Detail;
```

👉 쓰기 편해보이는 게 많죠? 이것, 저것 가져다가 버킷리스트를 마음껏 바꿔보세요. 😎

04. 페이지 의도적으로 가리기

▼ 4) 페이지를 왜 가려야 하나?

- 버킷리스트 앱을 새로고침 해볼까요?

👉 redux에 넣어둔 데이터 때문에 자꾸만 가짜 데이터 3개가 먼저 보이죠!
파이어스토어의 데이터만 제대로 보여주고 싶을 때, 어떻게 하면 좋을까요? 😎
→ 그렇죠! 페이지를 가려버리는 거예요. 언제까지? 파이어스토어에서 데이터를 가져올 때까지!

- 이 외에도 수정이나 추가하기 버튼을 눌렀을 때, 여러번 API를 호출하는 현상을 방지하기 위해 페이지를 가리기도 해요.

▼ 5) 로딩 스피너 만들기!

▼ (1) 로딩 스피너 컴포넌트 만들기

- 저는 머테리얼 UI의 아이콘을 사용해서 만들어 볼게요!

```
import React from "react";
import styled from "styled-components";
import {Eco} from "@material-ui/icons";

const Spinner = (props) => {

    return (
      <Outter>
        <Eco style={{ color: "#673ab7", fontSize: "150px" }} />
      </Outter>
    );
}

const Outter = styled.div`
  position: fixed;
  top: 0;
  left: 0;
  width: 100vw;
  height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
  background-color: #ede2ff;
`;

export default Spinner;
```

▼ (2) firestore 데이터 가져오기 전엔 페이지 진입을 막자!

- initialState에 is_loaded라는 변수를 추가하고 firestore에서 데이터를 받아오면 갱신합니다.

```
//bucket.js
...
const initialState = {
  is_loaded: false,
  list: [
    { text: "영화관 가기", completed: false },
    { text: "매일 책읽기", completed: false },
    { text: "수영 배우기", completed: false },
  ],
};
...
  case "bucket/LOAD": {
      if (action.bucket.length > 0) {
        return { list: action.bucket, is_loaded: true };
      }

      return state;
    }
```

- 변수를 App.js에서 보고, 조건부 랜더링을 합니다.

```
//App.js
...
// 이 함수는 스토어가 가진 상태값을 props로 받아오기 위한 함수예요.
const mapStateTopProps = (state) => ({
  bucket_list: state.bucket.list,
  is_loaded: state.bucket.is_loaded,
});
...
render() {
```

```
    // 콘솔로 확인해요!
    console.log(this.props.is_loaded);
    return (
      <div className="App">
        <Container>
          <Title>내 버킷리스트</Title>
          {/* firestore에서 데이터를 가져온 후에만 페이지를 보여줄거예요!  */}
          {!this.props.is_loaded ? (
            <Spinner />
          ) : (
            <React.Fragment>
              <Progress />
              <Line />
              <Switch>
                <Route path="/" exact component={BucketList} />
                <Route path="/detail/:index" component={Detail} />
                <Route component={NotFound} />
              </Switch>
            </React.Fragment>
          )}
        </Container>
       ...
```

05. Quiz_버킷리스트 생성 시 스피너 띄우기

▼ 6) ✍️Firestore에 데이터 추가하면 스피너를 띄워보자

🔥 추가하기 버튼을 누르면 → 로딩 스피너를 띄우고 → 추가가 끝나면 → 페이지를 보여줍시다!

▼ Q. 퀴즈설명

힌트:
is_loaded를 false로 바꿔주면 스피너가 뜨겠죠?
is_loaded를 바꿔주는 액션을 만들고, [추가하기]를 누르면 액션을 디스패치 해봅시다.
그리고 addBucketFB()에서 추가가 끝나면 다시 is_loaded를 false로 바꿔줍시다. 🙂

▼ A. 함께하기(완성본)

▼ **[코드스니펫] - App.js**

```
import React from "react";

import { withRouter } from "react-router";
import { Route, Switch } from "react-router-dom";

// import [컴포넌트 명] from [컴포넌트가 있는 파일경로];
import BucketList from "./BucketList";
import styled from "styled-components";
import Detail from "./Detail";
import NotFound from "./NotFound";
```

```
// 리덕스 스토어와 연결하기 위해 connect라는 친구를 호출할게요!
import { connect } from "react-redux";
// 리덕스 모듈에서 (bucket 모듈에서) 액션 생성 함수 두개를 가져올게요!
import {
  loadBucket,
  createBucket,
  loadBucketFB,
  addBucketFB,
} from "./redux/modules/bucket";
import Progress from "./Progress";

import Spinner from "./Spinner";
// firestore 가져오기
import { firestore } from "./firebase";

// 이 함수는 스토어가 가진 상태값을 props로 받아오기 위한 함수예요.
const mapStateTopProps = (state) => ({
  bucket_list: state.bucket.list,
  is_loaded: state.bucket.is_loaded,
});

// 이 함수는 값을 변화시키기 위한 액션 생성 함수를 props로 받아오기 위한 함수예요.
const mapDispatchToProps = (dispatch) => ({
  load: () => {
    dispatch(loadBucketFB());
  },
  create: (new_item) => {
    console.log(new_item);
    dispatch(addBucketFB(new_item));
  },
});

// 클래스형 컴포넌트는 이렇게 생겼습니다!
class App extends React.Component {
  constructor(props) {
    super(props);
    // App 컴포넌트의 state를 정의해줍니다.
    this.state = {};
    // ref는 이렇게 선언합니다!
    this.text = React.createRef();
  }

  componentDidMount() {
    this.props.load();
  }

  addBucketList = () => {
    const new_item = this.text.current.value;
    this.props.create(new_item);
  };

  // 랜더 함수 안에 리액트 엘리먼트를 넣어줍니다!
  render() {
    // 콘솔로 확인해요!
    console.log(this.props.is_loaded);
    return (
      <div className="App">
        <Container>
          <Title>내 버킷리스트</Title>
          {/* firestore에서 데이터를 가져온 후에만 페이지를 보여줄거예요!  */}
          {!this.props.is_loaded ? (
            <Spinner />
          ) : (
            <React.Fragment>
              <Progress />
              <Line />
              <Switch>
                <Route path="/" exact component={BucketList} />
```

```
                <Route path="/detail/:index" component={Detail} />
                <Route component={NotFound} />
              </Switch>
            </React.Fragment>
          )}
        </Container>
        <Input>
          <input type="text" ref={this.text} />
          <button onClick={this.addBucketList}>추가하기</button>
        </Input>

        <button
          onClick={() => {
            window.scrollTo({ top: 0, left: 0, behavior: "smooth" });
          }}
        >
          위로가기
        </button>
      </div>
    );
  }
}

const Input = styled.div`
  max-width: 350px;
  min-height: 10vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
  border-radius: 5px;
  border: 1px solid #ddd;
  display: flex;
  align-items: center;
  justify-content: space-between;
  & > * {
    padding: 5px;
  }

  & input {
    border-radius: 5px;
    margin-right: 10px;
    border: 1px solid #888;
    width: 70%;
    &:focus {
      border: 1px solid #a673ff;
    }
  }

  & button {
    width: 25%;
    color: #fff;
    border: 1px solid #a673ff;
    background-color: #a673ff;
  }
`;

const Container = styled.div`
  max-width: 350px;
  min-height: 60vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
  border-radius: 5px;
  border: 1px solid #ddd;
`;

const Title = styled.h1`
  color: #673ab7;
```

```
  text-align: center;
`;

const Line = styled.hr`
  margin: 16px 0px;
  border: 1px dotted #ddd;
`;
// withRouter 적용
// connect로 묶어줬습니다!
export default connect(mapStateTopProps, mapDispatchToProps)(withRouter(App));
```

▼ [코드스니펫] - bucket.js

```
import { firestore } from "../../firebase";

// Actions
const LOAD = "bucket/LOAD";
const CREATE = "bucket/CREATE";
const DELETE = "bucket/DELETE";
const UPDATE = "bucket/UPDATE";
// is loaded
const LOADED = "bucket/LOADED";

const initialState = {
  is_loaded: false,
  list: [
    { text: "영화관 가기", completed: false },
    { text: "매일 책읽기", completed: false },
    { text: "수영 배우기", completed: false },
  ],
};

const bucket_db = firestore.collection("bucket");

// Action Creators
export const loadBucket = (bucket) => {
  return { type: LOAD, bucket };
};

export const createBucket = (bucket) => {
  return { type: CREATE, bucket };
};

export const deleteBucket = (bucket) => {
  return { type: DELETE, bucket };
};

export const updateBucket = (bucket) => {
  return { type: UPDATE, bucket };
};

// loaded를 받아서 is_loaded 값을 true/false로 바꿔줄 액션 생성 함수입니다.
export const isLoaded = (loaded) => {
  return {type: LOADED, loaded}
}

// 파이어베이스랑 통신하는 부분
export const loadBucketFB = () => {
  return function (dispatch) {
    bucket_db.get().then((docs) => {
      let bucket_data = [];
      docs.forEach((doc) => {
        // 도큐먼트 객체를 확인해보자!
        console.log(doc);
        // 도큐먼트 데이터 가져오기
```

```
        console.log(doc.data());
        // 도큐먼트 id 가져오기
        console.log(doc.id);

        if (doc.exists) {
          bucket_data = [...bucket_data, { id: doc.id, ...doc.data() }];
        }
      });

      console.log(bucket_data);
      // 이제 액션이 디스패치 되도록 해줍시다! 그러면 끝!
      dispatch(loadBucket(bucket_data));
    });
  };
};

export const addBucketFB = (bucket) => {
  return function (dispatch) {

    // 요청 보내기 전에 스피너를 보여줍시다
    dispatch(isLoaded(false));

    // 생성할 데이터를 미리 만들게요!
    let bucket_data = { text: bucket, completed: false };

    // add()에 데이터를 넘겨줍시다!
    bucket_db
      .add(bucket_data)
      .then((docRef) => {
        // id를 추가한다!
        bucket_data = { ...bucket_data, id: docRef.id };

        console.log(bucket_data);

        // 성공했을 때는? 액션 디스패치!
        dispatch(createBucket(bucket_data));
        // 스피너도 다시 없애줘야죠!
        dispatch(isLoaded(true));
      })
      .catch((err) => {
        // 여긴 에러가 났을 때 들어오는 구간입니다!
        console.log(err);
        window.alert("오류가 났네요! 나중에 다시 시도해주세요!");
        // 스피너도 다시 없애줘야죠!
        dispatch(isLoaded(true));
      });
  };
};

export const updateBucketFB = (bucket) => {
  return function (dispatch, getState) {
    // state에 있는 값을 가져옵니다!
    const _bucket_data = getState().bucket.list[bucket];

    // id가 없으면? 바로 끝내버립시다.
    if (!_bucket_data.id) {
      return;
    }

    // 새로운 값을 만들어요!
    let bucket_data = { ..._bucket_data, completed: true };

    bucket_db
      .doc(bucket_data.id)
      .update(bucket_data)
      .then((res) => {
        dispatch(updateBucket(bucket));
      })
```

```
      .catch((err) => {
        console.log("err");
      });
  };
};

export const deleteBucketFB = (bucket) => {
  return function (dispatch, getState) {
    const _bucket_data = getState().bucket.list[bucket];
    // id가 없으면? 바로 끝내버립시다.
    if (!_bucket_data.id) {
      return;
    }
    // 삭제하기
    bucket_db
      .doc(_bucket_data.id)
      .delete()
      .then((res) => {
        dispatch(deleteBucket(bucket));
      })
      .catch((err) => {
        console.log("err");
      });
  };
};

// Reducer
export default function reducer(state = initialState, action) {
  switch (action.type) {
    // do reducer stuff
    case "bucket/LOAD": {
      if (action.bucket.length > 0) {
        return { list: action.bucket, is_loaded: true };
      }

      return state;
    }

    case "bucket/CREATE": {
      const new_bucket_list = [...state.list, action.bucket];
      return { ...state, list: new_bucket_list };
    }

    case "bucket/DELETE": {
      const bucket_list = state.list.filter((l, idx) => {
        if (idx !== action.bucket) {
          return l;
        }
      });
      return { ...state, list: bucket_list };
    }

    case "bucket/UPDATE": {
      const bucket_list = state.list.map((l, idx) => {
        if (idx === action.bucket) {
          return { ...l, completed: true };
        }

        return l;
      });

      return { ...state, list: bucket_list };
    }

    case "bucket/LOADED": {

      return {...state, is_loaded: action.loaded};
    }
```

```
    default:
      return state;
  }
}
```

💡 조금 아리송하시다면, 삭제나 완료 시에도 스피너를 띄우며 연습해보세요!

06. AWS S3 버킷

▼ 7) S3 버킷이란?

👉 S3(Simple Storage Service)는 단순 스토리지 서비스예요!
이미지나 파일을 저장할 수 있습니다.
html, css, js 같은 정적 자원을 올리고, 정적 웹 사이트를 호스팅할 수도 있어요.
우리가 컴퓨터에 폴더 만드는 것처럼 버킷을 만들고 사용할 수 있어요!

- 버킷을 하나 만들고 아무 텍스트 파일이나 올려볼까요?
- 파일을 올려봤으면, 브라우저 주소창을 통해 해당 파일을 열어도 봅시다!

▼ 8) 정적 웹 사이트란?

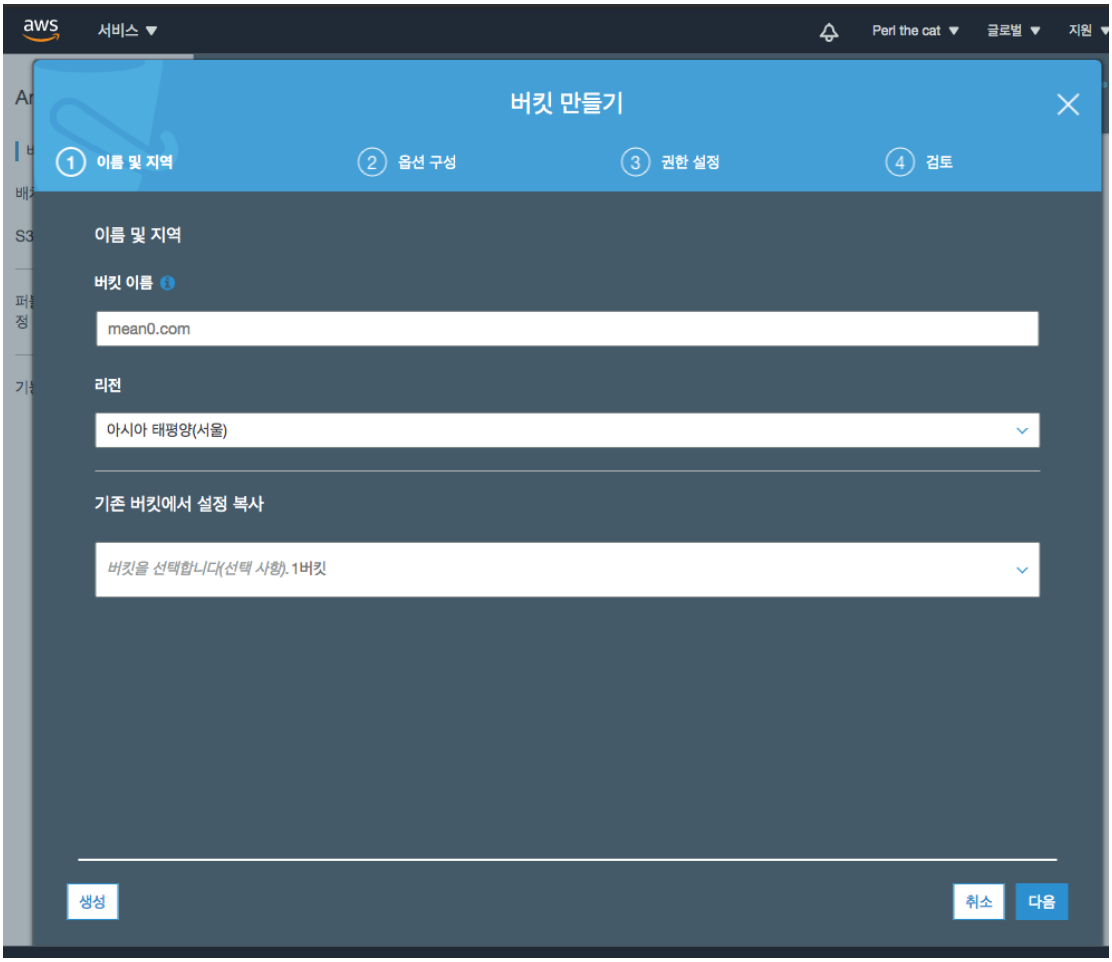
👉 웹 사이트는 **서버 측 스크립트 사용 유무를 기준**으로 동적 웹 사이트와 정적 웹 사이트로 나눠 볼 수 있어요. (서버 측 스크립트는 PHP, JSP, ASP같은 친구들을 말해요!)
정적 웹 사이트는 html, js, css같이 정적 자원으로만 이루어진 웹 사이트입니다. 🙂

07. S3 버킷 설정하기

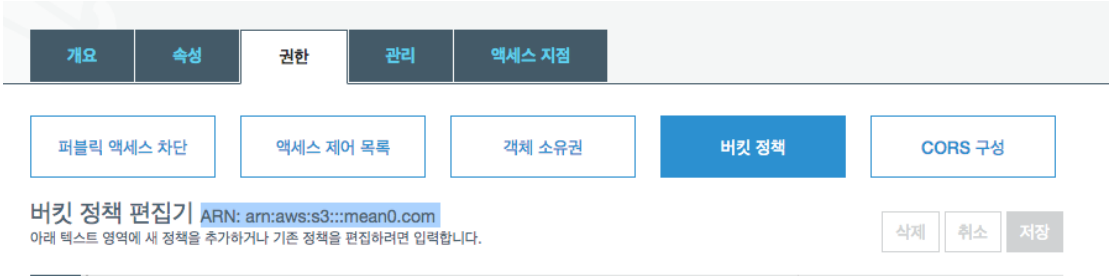
▼ 9) S3 버킷 설정하기

👉 [S3 정적 웹 사이트 호스팅 방법]
방법을 굳이 외울 필요 없어요. AWS가 제공하는 설명서가 있답니다!
(설명서 링크→)
준비물이 있어요! **도메인**! 다들 준비되셨나요! 😎

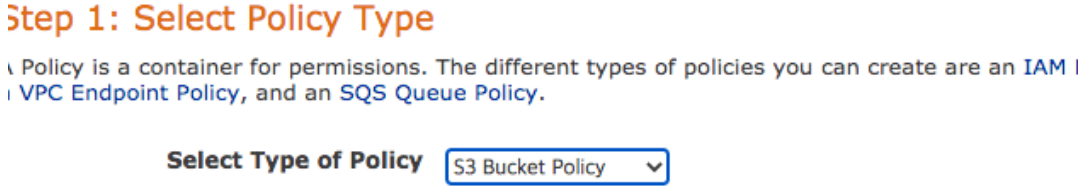
- (1) 버킷을 생성한다. (사이트 도메인과 버킷 이름이 같아야 합니다!)

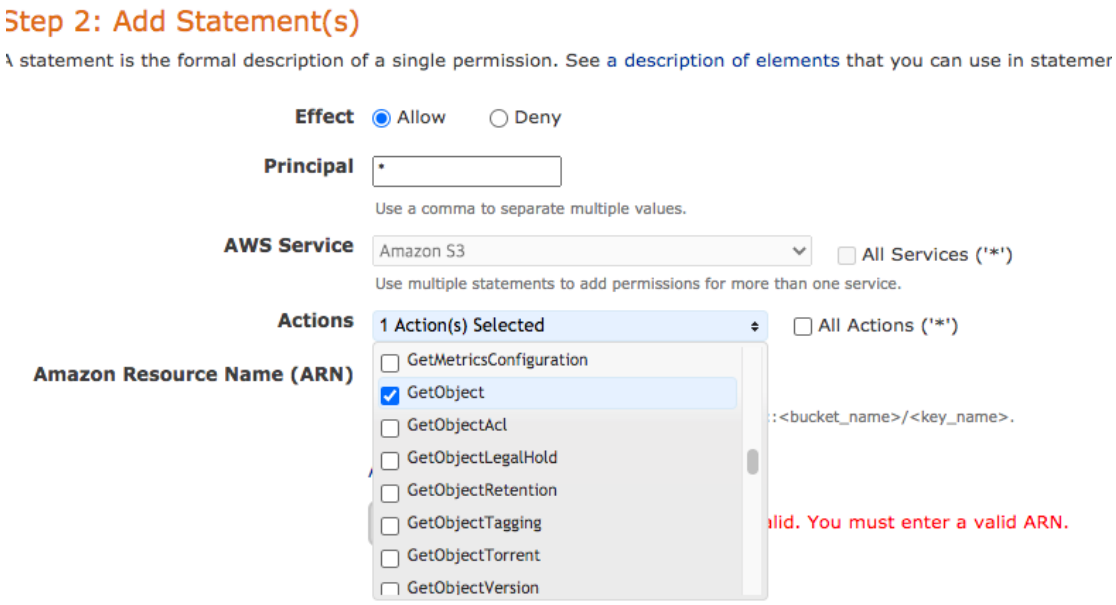

aws
서비스
Perl the cat
글로벌
지원
버킷 만들기
1 이름 및 지역
2 옵션 구성
3 권한 설정
4 검토
이름 및 지역
버킷 이름
mean0.com
리전
아시아 태평양(서울)
기존 버킷에서 설정 복사
버킷을 선택합니다(선택 사항). 1버킷
생성
취소
다음

- (2) 권한 탭으로 들어갑니다.

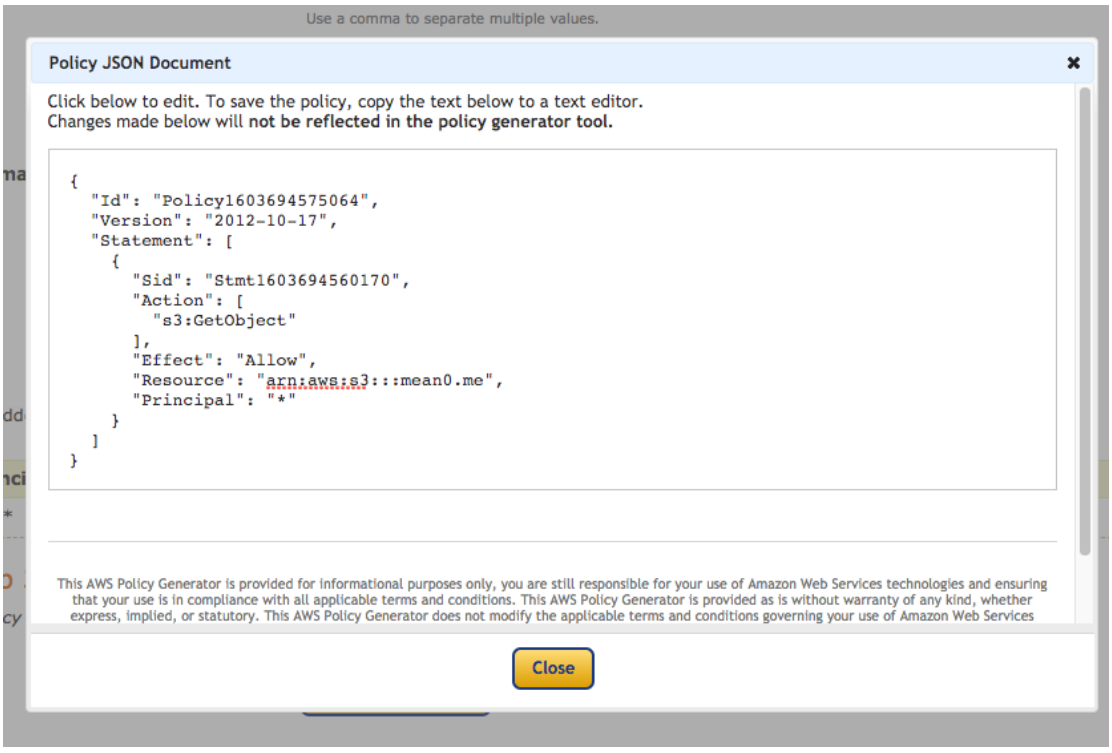


- (3) 상단 ARN([ARN: arn:aws:s3:::[도메인 주소]])를 복사해서 아래의 정책 생성기를 누른다.
- (4) 정책 생성하고,





- (5) 생선한 정책을(json) 복사해서 붙여 넣는다. (주의!! ARN 뒤에 꼭 /* 써주기)



▼ 10) S3 버킷에 내 결과물 올리기

👉 빌드 먼저 하고, 올려볼까요 😉

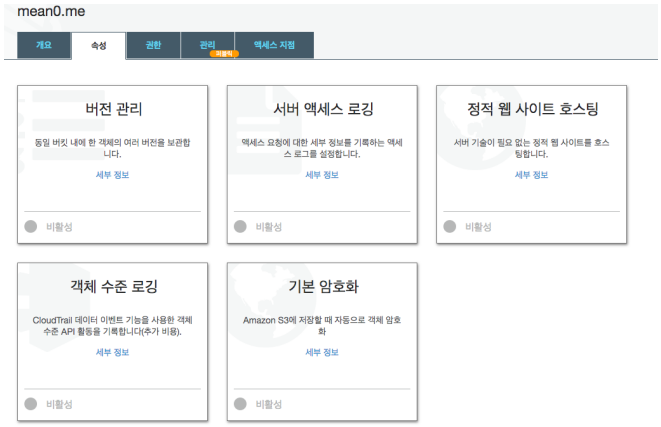
▼ (1) 빌드하기

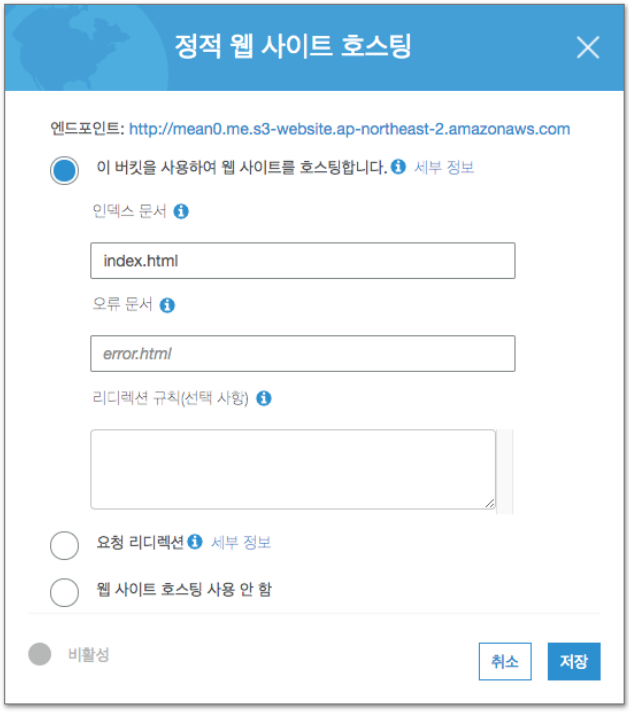
```
yarn build
```

▼ (2) 결과물 올리기

- 버킷에 build 폴더 내 파일을 올린다.

▼ (3) 정적 웹 사이트 호스팅 설정하기





▼ (4) 확인해보자!

- 앤드포인트 주소를 클릭 해 내 사이트에 들어갑니다!
- 잘 나오죠! 😉

08. 도메인 연결하기

▼ 11) 내 도메인 연결하기

- route 53에서 호스팅 영역을 생성한다

- 네임서버를 가비아에 등록(혹은 도메인을 산 곳에서 등록)

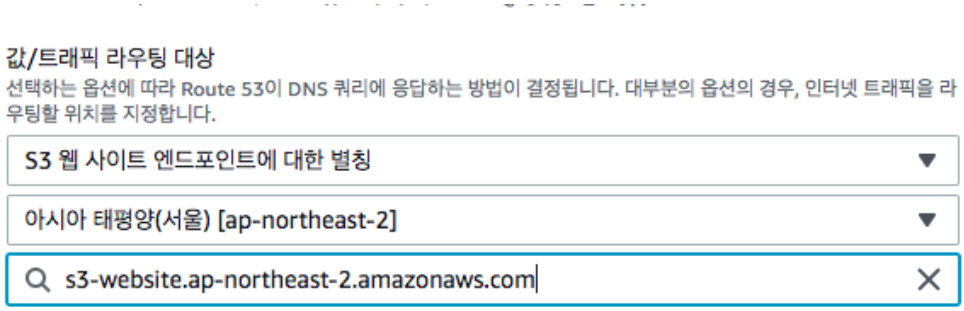
레코드 이름	유형	라우팅 정책	차별화 요소	별칭	값/트래픽 라우팅 대상
mean0.me	NS	단순	-	아니요	ns-1547.awsdns-01.co.uk. ns-623.awsdns-13.net. ns-1506.awsdns-60.org. ns-481.awsdns-60.com.

네임서버 설정

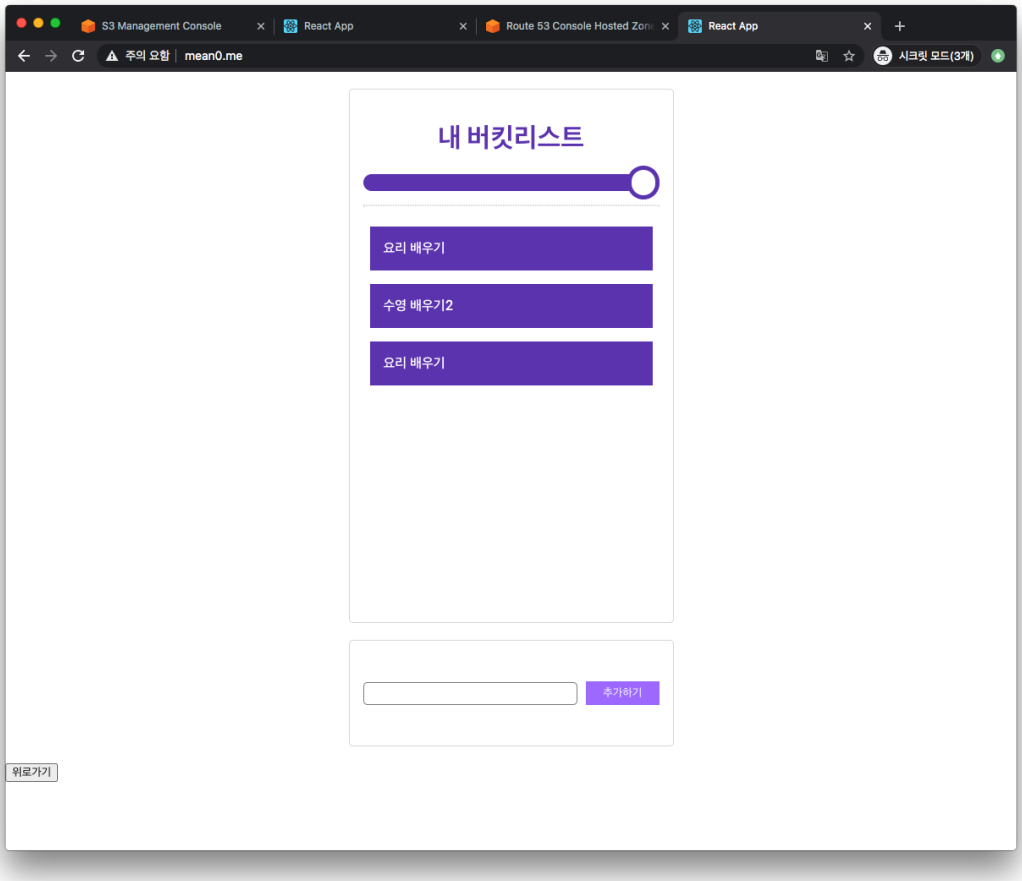
mean0.me

구분	호스트명	구분	
1차	ns-1547.awsdns-01.co.uk	2차	ns-623.awsdns-13.net
3차	ns-1506.awsdns-60.org	4차	ns-481.awsdns-60.com

- 레코드를 생성한다



- 확인하자!



09. Firebase로 배포하기

▼ 12) firebase 호스팅하기

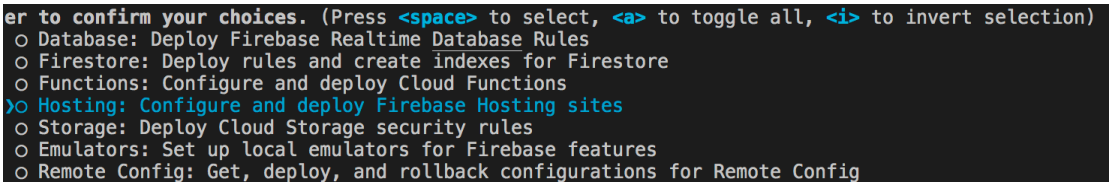
- 대시보드에서 호스팅 신청하기
- 우리 프로젝트에 cli 설치

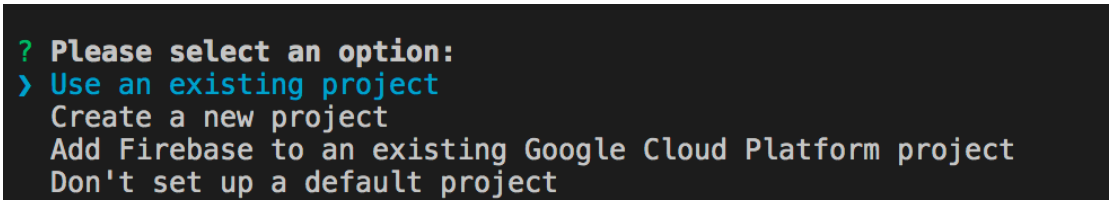
```
yarn global firebase-tools
```

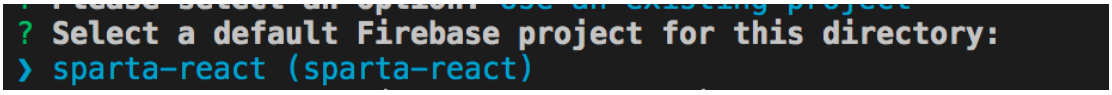
- firebase에 로그인하고 init 실행하기

```
#웹브라우저가 열리고 내 구글 계정을 물어볼거예요. 로그인해줍니다.
yarn firebase login
#로그인 후 init!
yarn firebase init
```

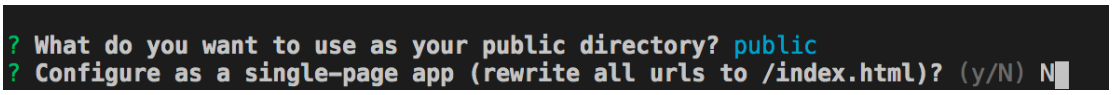
- 호스팅을 선택해줍니다(방향키로 이동해요)







- 설정을 해줍니다! 저는 덮어쓰지 않도록 했어요!



▼ 13) firebase에 내 결과물 올리기

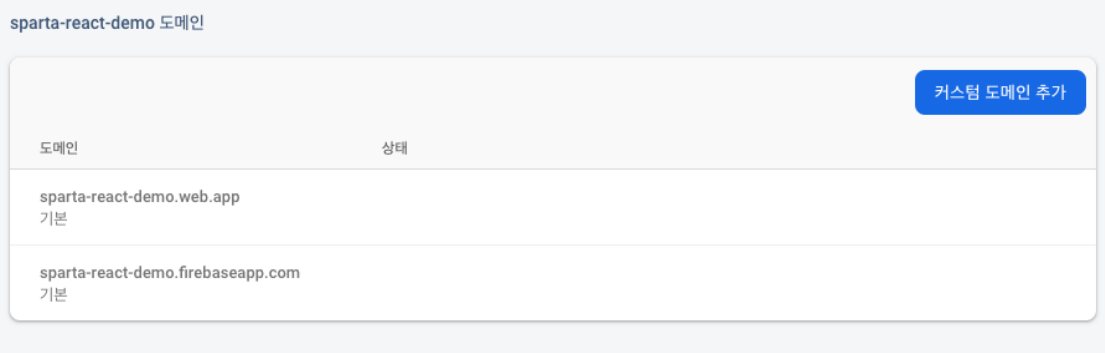
- firebase.json을 확인해봅시다

```
{
  "hosting": {
    "public": "build",
    "ignore": [
      "firebase.json",
      "**/.*",
      "**/node_modules/**"
    ]
  }
}
```

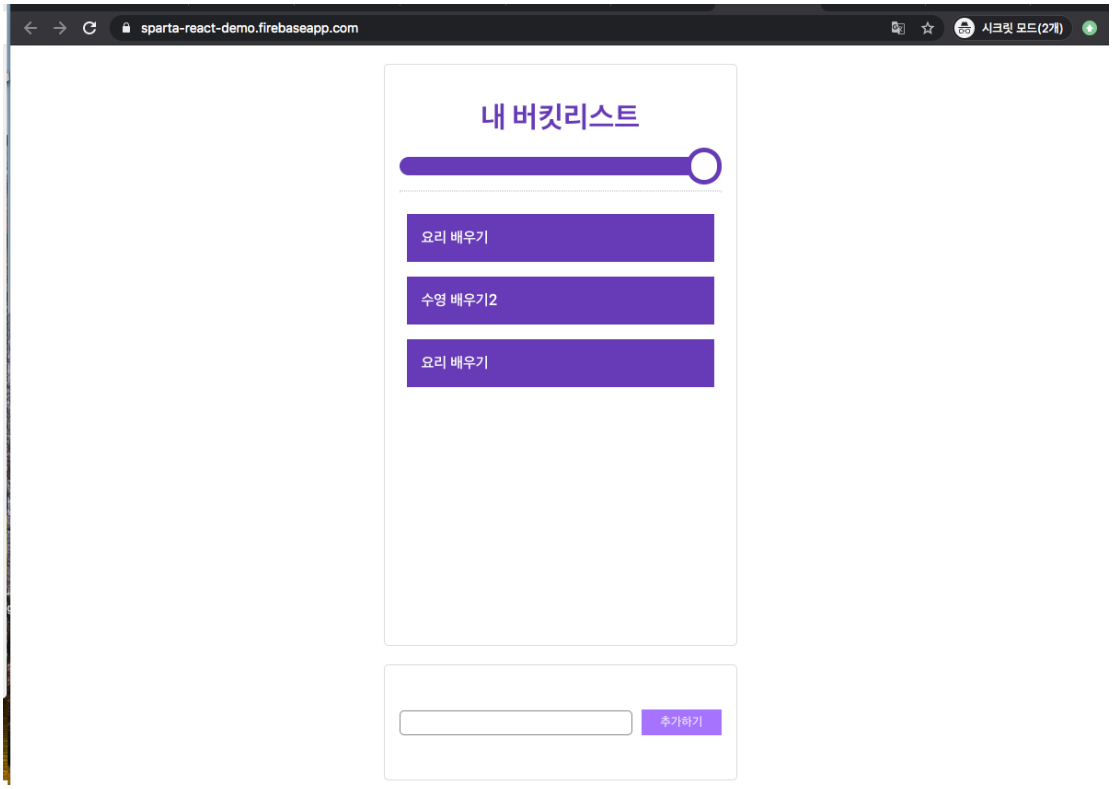
- 빌드한 결과물을 올려봅시다!

```
yarn firebase deploy
```

- 배포가 끝났다면 firebase 대시보드 → 호스팅으로 이동해주세요.



- 그리고 도메인으로 들어가 확인해봅니다 😎

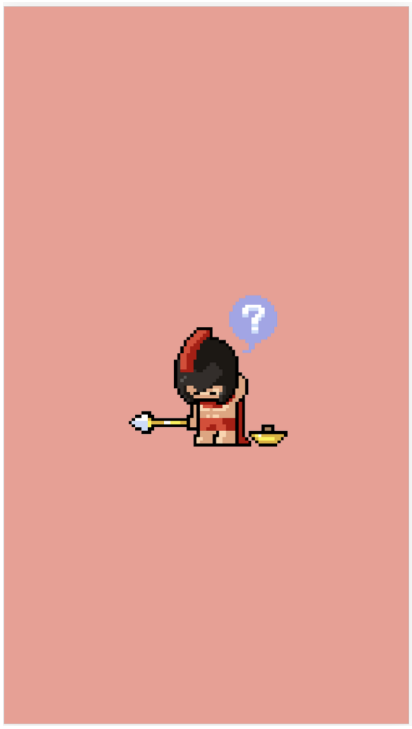


10. 끝 & 숙제 설명

너! 나를 얼마나 아니? 사이트에 firestore를 연결하고, 스피너를 붙여봅니다.
완성된 사이트를 s3로 배포해봐요!

▼ 기획서(레이아웃) 보기

▼ 예시 화면



11. 5주차 숙제 답안 코드

▼ **[코드스니펫] - 5주차 숙제 답안 코드**

전체 **코드**

▼ **rank.js**

▼ **configStore.js**

```
import { createStore, combineReducers, applyMiddleware } from "redux";
import thunk from "redux-thunk";
import quiz from "./modules/quiz";
import rank from "./modules/rank";
import { createBrowserHistory } from "history";

export const history = createBrowserHistory();

const middlewares = [thunk];

const enhancer = applyMiddleware(...middlewares);

const rootReducer = combineReducers({ quiz, rank });
const store = createStore(rootReducer, enhancer);

export default store;
```

▼ **firebase.js**

```
import firebase from "firebase/app";
import "firebase/firestore";
```

```
const firebaseConfig = {
  apiKey: "AIzaSyDVt53_A0kwA5NZ_KVLW3btY5yOYN40IeE",
  authDomain: "friend-test-3b9ff.firebaseapp.com",
  databaseURL: "https://friend-test-3b9ff.firebaseio.com",
  projectId: "friend-test-3b9ff",
  storageBucket: "friend-test-3b9ff.appspot.com",
  messagingSenderId: "535393314714",
  appId: "1:535393314714:web:4ef23f250ca87d7de17c23",
  measurementId: "G-M852M85RP0",
};

firebase.initializeApp(firebaseConfig);

const firestore = firebase.firestore();

export { firestore };
```

▼ Ranking.js

```
import React from "react";
import styled from "styled-components";

import { useSelector, useDispatch } from "react-redux";
import { resetAnswer } from "./redux/modules/quiz";
import {getRankFB} from "./redux/modules/rank";

import Spinner from "./Spinner";

const Ranking = (props) => {
  const dispatch = useDispatch();
  const _ranking = useSelector((state) => state.rank.ranking);
  const is_loaded = useSelector((state) => state.rank.is_loaded);
  // Array 내장 함수 sort로 정렬하자!
  // https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort


  const user_rank = React.useRef(null);

  React.useEffect(() => {
    dispatch(getRankFB());
    if(!user_rank.current){
      return;
    }

    window.scrollTo({top: user_rank.current.offsetTop, left: 0, behavior: "smooth"});
  }, []);
  const ranking = _ranking.sort((a, b) => {
    // 높은 수가 맨 앞으로 오도록!
    return b.score - a.score;
  });

  if(!is_loaded){
    return (<Spinner/>);
  }

  return (
    <RankContainer>
      <Topbar>
        <p>
          <span>{ranking.length}명</span>의 사람들 중 당신은?
        </p>
      </Topbar>
```

```
      <RankWrap>
        {ranking.map((r, idx) => {
          if (r.current) {
            return (
              <RankItem key={idx} highlight={true} ref={user_rank}>
                <RankNum>{idx + 1}등</RankNum>
                <RankUser>
                  <p>
                    <b>{r.name}</b>
                  </p>
                  <p>{r.message}</p>
                </RankUser>
              </RankItem>
            );
          }

          return (
            <RankItem key={idx}>
              <RankNum>{idx + 1}등</RankNum>
              <RankUser>
                <p>
                  <b>{r.name}</b>
                </p>
                <p>{r.message}</p>
              </RankUser>
            </RankItem>
          );
        })}
      </RankWrap>

      <Button
        onClick={() => {
          dispatch(resetAnswer());
          window.location.href = "/";
        }}
      >
        다시 하기
      </Button>
    </RankContainer>
  );
};

const RankContainer = styled.div`
  width: 100%;
  padding-bottom: 100px;
`;

const Topbar = styled.div`
  position: fixed;
  top: 0;
  left: 0;
  width: 100vw;
  min-height: 50px;
  border-bottom: 1px solid #ddd;
  background-color: #fff;
  & > p {
    text-align: center;
  }

  & > p > span {
    border-radius: 30px;
    background-color: #fef5d4;
    font-weight: 600;
    padding: 4px 8px;
  }
`;

const RankWrap = styled.div`
```

```
  display: flex;
  flex-direction: column;
  width: 100%;
  margin-top: 58px;
`;

const RankItem = styled.div`
  width: 80vw;
  margin: 8px auto;
  display: flex;
  border-radius: 5px;
  border: 1px solid #ddd;
  padding: 8px 16px;
  align-items: center;
  background-color: ${(props) => (props.highlight ? "#ffd6aa" : "#ffffff")};
`;

const RankNum = styled.div`
  text-align: center;
  font-size: 2em;
  font-weight: 600;
  padding: 0px 16px 0px 0px;
  border-right: 1px solid #ddd;
`;

const RankUser = styled.div`
  padding: 8px 16px;
  text-align: left;
  & > p {
    &:first-child > b {
      border-bottom: 2px solid #212121;
    }
    margin: 0px 0px 8px 0px;
  }
`;

const Button = styled.button`
  position: fixed;
  bottom: 5vh;
  left: 0;
  padding: 8px 24px;
  background-color: ${(props) => (props.outlined ? "#ffffff" : "#dadafc")};
  border-radius: 30px;
  margin: 0px 10vw;
  border: 1px solid #dadafc;
  width: 80vw;
`;

export default Ranking;
```

▼ Message.js

```
import React from "react";
import img from "./scc_img01.png";
import { useDispatch, useSelector } from "react-redux";
import { addRank, addRankFB } from "./redux/modules/rank";

const Message = (props) => {
  const dispatch = useDispatch();
  const name = useSelector((state) => state.quiz.name);
  const answers = useSelector((state) => state.quiz.answers);
  const user_name = useSelector((state) => state.rank.user_name);

  const input_text = React.useRef(null);
  // 정답만 걸러내기
```

```
let correct = answers.filter((answer) => {
  return answer;
});

// 점수 계산하기
let score = (correct.length / answers.length) * 100;

// 컬러셋 참고: https://www.shutterstock.com/ko/blog/pastel-color-palettes-rococo-trend/
return (
  <div
    style={{
      display: "flex",
      height: "100vh",
      width: "100vw",
      overflow: "hidden",
      padding: "16px",
      boxSizing: "border-box",
    }}
  >
    <div
      className="outter"
      style={{
        display: "flex",
        alignItems: "center",
        justifyContent: "center",
        flexDirection: "column",
        height: "100vh",
        width: "100vw",
        overflow: "hidden",
        padding: "0px 10vw",
        boxSizing: "border-box",
        maxWidth: "400px",
        margin: "0px auto",
      }}
    >
      <img
        src={img}
        style={{ width: "80%", margin: "-70px 16px 48px 16px" }}
      />
      <h1 style={{ fontSize: "1.5em", margin: "0px", lineHeight: "1.4" }}>
        <span
          style={{
            backgroundColor: "#fef5d4",
            padding: "5px 10px",
            borderRadius: "30px",
          }}
        >
          {name}
        </span>
        에게 한마디
      </h1>
      <input
        ref={input_text}
        type="text"
        style={{
          padding: "10px",
          margin: "24px 0px",
          border: "1px solid #dadafc",
          borderRadius: "30px",
          width: "100%",
        }}
        placeholder="한 마디 적기"
      />
      <button
        onClick={() => {
          let rank_info = {
            score: parseInt(score),
            name: user_name,
```

```
              message: input_text.current.value,
              current: true,
            };
            // 랭킹 정보 넣기
            // dispatch(addRank(rank_info));

            dispatch(addRankFB(rank_info));
            // 주소 이동
            // 시간 차를 두고 이동 시켜줘요.
            window.setTimeout(() => {
              props.history.push("/ranking");
            }, 1000);

          }}
          style={{
            padding: "8px 24px",
            backgroundColor: "#dadafc",
            borderRadius: "30px",
            border: "#dadafc",
          }}
        >
          한마디하고 랭킹 보러 가기
        </button>
      </div>
    </div>
  );
};

export default Message;
```

▼ **Spinner.js**

```
import React from "react";
import styled from "styled-components";

import img from "./scc_img01.png";
const Spinner = (props) => {
  return (
    <Outter>
      <img src={img} />
    </Outter>
  );
};

const Outter = styled.div`
  background-color: #df402c88;
  width: 100vw;
  height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
  & img {
    width: 150px;
  }
`;
export default Spinner;
```