24. FrontEnd Project : 시작 및 히원인증 구현

24.1 작업환경준비

1. mkdir 24.blog\blog-frontend
2. cd 24.blog\blog-frontend
3. yarn create react-app .

24.1.1 설정파일 만들기

.prettierrc 생성

```
{
  "singleQuote": true,
  "semi": true,
  "useTabs": false,
  "tabWidth": 2,
  "trailingComma": "all",
  "printWidth": 80
}
```

jsconfig.json

- 자동 import 기능 활성화

```
{
  "compilerOptions": {
    "target": "es6"
  }
}
```

24.1.2 라우터적용

- 설치 : `yarn add react-router-dom`
- src/pages : 작성할 컴퍼넌트

1. LoginPage.js - 로그인

```
const LoginPage = () => {
return <div>로그인</div>;
};
export default LoginPage;
```

2. RegisterPage.js - 회원가입

```
const RegisterPage = () => {
return <div>회원가입</div>;
}
export default RegisterPage;
```

3. WritePage.js - 글쓰기

```
const WritePage = () => {
return <div>글쓰기</div>;
}
export default WritePage;
```

4. PostPage.js - 포스트읽기

```
const PostPage = () => {
return <div>포스트읽기</div>;
};
export default PostPage;
```

5. PostListPage.js - 포스트 목록

```
const PostListPage = () => {
return <div>포스트 리스트</div>;
};
export default PostListPage;
```

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import { BrowserRouter } from 'react-router-dom';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

src\App.js

`@를 삭제해야 정상처리가 된다. 26장에서 확인해 볼 것`

- PostListPage의 경우 @:username 경로에서도 보여진다.
  - @:username은 http://localhost:3000/@gilabaek 같은 경로에서 username파라미터로 읽을 수 있게 한다.
  - `<Route path="/@:username" element={<PostListPage/>} />`
  - `<Route path="/@:username:postId" element={<PostListPage/>} />`

```
import { Route, Routes } from 'react-router-dom';
import PostListPage from './pages/PostListPage';
import LoginPage from './pages/LoginPage';
import RegisterPage from './pages/RegisterPage';
import WritePage from './pages/WritePage';
import PostPage from './pages/PostPage';

const App = () => {
  return (
    <Routes>
      <Route path="/" element={<PostListPage />} />
      <Route path="/login" element={<LoginPage />} />
      <Route path="/register" element={<RegisterPage />} />
      <Route path="/write" element={<WritePage />} />
      <Route path="/@:username">
        <Route index element={<PostListPage />} />
```

```
          <Route path=":postId" element={<PostPage />} />
        </Route>
      </Routes>
    );
};
export default App;
```

- 프로젝트 시작 : yarn start
  - http://localhost:3000/
  - http://localhost:3000/@gilbaek
  - http://localhost:3000/write
  - http://localhost:3000/@gilbaek/12345
  - http://localhost:3000/login
  - http://localhost:3000/register

24.1.3 스타일 설정

- styled-components를 사용하여 스타일링하기
  - `yarn add styled-components`

lib\styles\palette.js

- 색상소스 : https://yeun.github.io/open-color/
- open-color 라이브러리를 설치해서 사용해도 되지만 open-color의 색상들이 자동 import가 되지 않는다.
- 또한, 색상팔레트를 만들면 필요한 색상만 사용할 수 있다.

```
// source: https://yeun.github.io/open-color/
const palette = {
  gray: [
    '#f8f9fa',
    '#f1f3f5',
    '#e9ecef',
    '#dee2e6',
    '#ced4da',
    '#adb5bd',
    '#868e96',
    '#495057',
    '#343a40',
    '#212529'
  ],
  cyan: [
    '#e3fafc',
    '#c5f6fa',
    '#99e9f2',
    '#66d9e8',
    '#3bc9db',
    '#22b8cf',
    '#15aabf',
    '#1098ad',
    '#0c8599',
    '#0b7285'
  ]
};
```

```
export default palette;
```

24.1.4 Button 컴퍼넌트 만들기

src\components\common\Button.js

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

const StyledButton = styled.button`
  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]};
  }
`;

const Button = props => <StyledButton {...props} />;

export default Button;
```

pags\PostListPage.js

```
import Button from "../components/common/Button";

const PostListPage = () => {
  return (
    <div>
      <Button>버튼</Button>
    </div>
  );
};
export default PostListPage;
```

indes.css

```
body {
  margin: 0;
  padding: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto',
'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  box-sizing: border-box; /* 엘리먼트의 box-sizing 값을 border-box로 설정 */
  min-height: 100%;
}
```

```
#root {
  min-height: 100%;
}

/* 추후 회원인증 페이지에서
 배경화면을 페이지의 전체 영역에 채우기 위한 용도 */
html {
  height: 100%;
}

/* 링크에 색상 및 밑줄 없애기 */
a {
  color: inherit;
  text-decoration: none;
}

* {
  box-sizing: inherit; /* 모든 엘리먼트의 box-sizing 값을 border-box로 설정 */
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

24.1.5 Redux 적용

- 향후 redux-sega를 사용할 예정이나 지금은 `리덕스스토어생성 및 Provider 컴퍼넌트를 통해 리덕스 적용`
- 불변성을 관리하기 위해 spread연산자를 사용하지 않고 `immer로 불변성 관린`
- Ducks패턴을 사용하여 `액션타입, 액션생성함수, 리듀서를 하나의 모듈에 생성`
- 설치 : `yarn add redux react-redux redux-actions immer redux-devtools-extension`

src\modules\auth.js

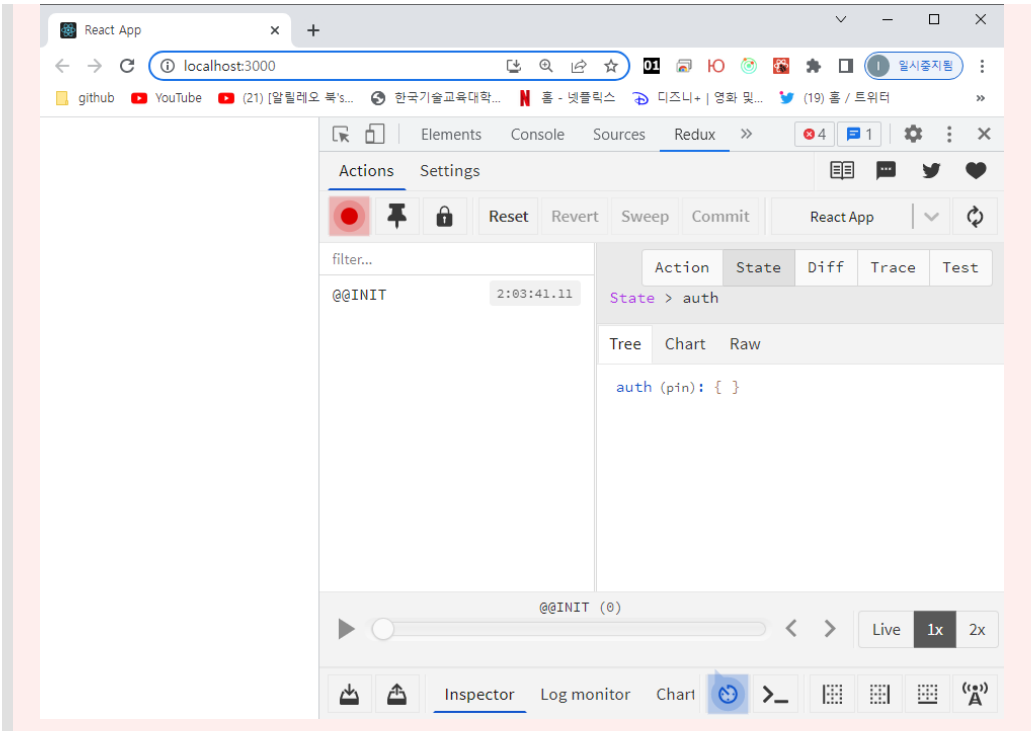
```
import {createAction, handleActions} from 'redux-actions'

const SAMPLE_ACTION = 'auth/SAMPLE_ACTION';

export const sampleAction = createAction(SAMPLE_ACTION);

const initialState = {};

const auth = handleActions(
  {
    [SAMPLE_ACTION]: (state, action) => state,
  },
  initialState
)

export default auth;
```

src\modules\index.js

```
import { combineReducers } from 'redux';
import auth from './auth';

const rootReducer = combineReducers({
  auth,
})

export default rootReducer;
```

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import { BrowserRouter } from 'react-router-dom';
import { Provider } from 'react-redux';
import { legacy_createStore as  createStore } from 'redux';
import { composeWithDevTools } from 'redux-devtools-extension';
import rootReducer from './modules';

const store = createStore(rootReducer, composeWithDevTools());

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Provider store={store}>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </Provider>
);
```



24.2 회원가입과 로그인 구현

24.2.1 UI 준비하기

components\auth\AuthForm.js

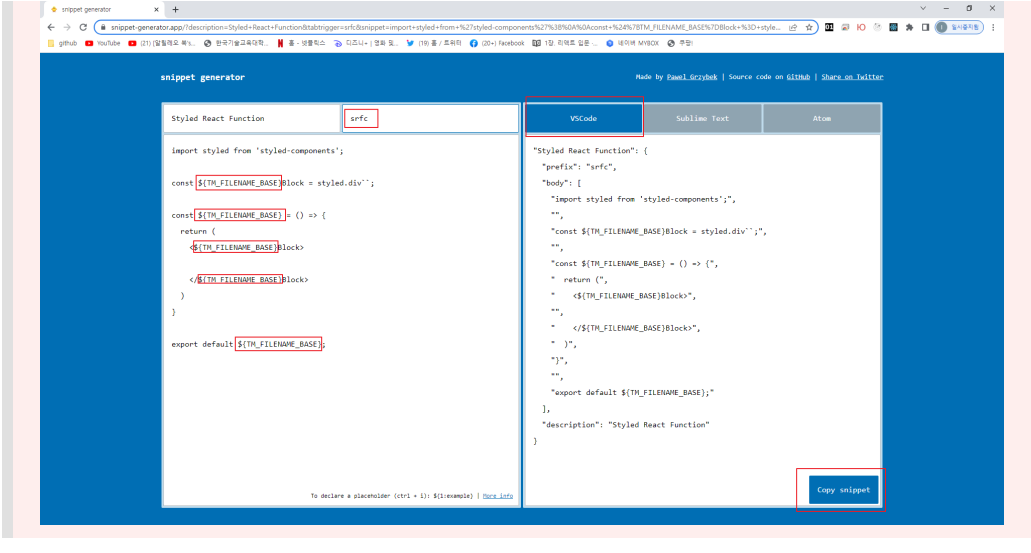
```
import styled from 'styled-components';

/**
 * 회원가입 또는 로그인 폼을 보여줍니다.
 */

const AuthFormBlock = styled.div``;

const AuthForm = () => {
  return (
    <AuthFormBlock>
      AuthForm
    </AuthFormBlock>
  )
}

export default AuthForm;
```

components\auth\AuthTemplate.js

```
import styled from 'styled-components';

/**
 * 회원가입 / 로그인 페이지의 레이아웃을 담당하는 컴포넌트입니다.
 */

const AuthTemplateBlock = styled.div``;

const AuthTemplate  = () => {
  return (
    <AuthTemplateBlock>

    </AuthTemplateBlock>
  )
}

export default AuthTemplate;
```

snippet 설정하기

- snippet generator : https://snippet-generator.app/



- VSCode : 설정ICON > user snippits > javascriptreact

```
{
    "Styled React Function": {
        "prefix": "srfc",
        "body": [
            "import styled from 'styled-components';",
            "",
            "const ${TM_FILENAME_BASE}Block = styled.div``;",
            "",
            "const ${TM_FILENAME_BASE} = () => {",
            "  return (",
            "    <${TM_FILENAME_BASE}Block>",
            "",
            "    </${TM_FILENAME_BASE}Block>",
            "  )",
            "}",
            "",
            "export default ${TM_FILENAME_BASE};"
        ],
        "description": "Styled React Function"
    }
}
```

- 저장후 VSCode 우측 Javascripreact로 변경 후
- 참고 : https://code.visualstudio.com/docs/editor/userdefinedsnippets

pages\LoginPage.js

```
import AuthForm from '../components/auth/AuthForm';
import AuthTemplate from '../components/auth/AuthTemplate';

const LoginPage = () => {
  return (
    <AuthTemplate>
      <AuthForm />
    </AuthTemplate>
  )
};

export default LoginPage;
```

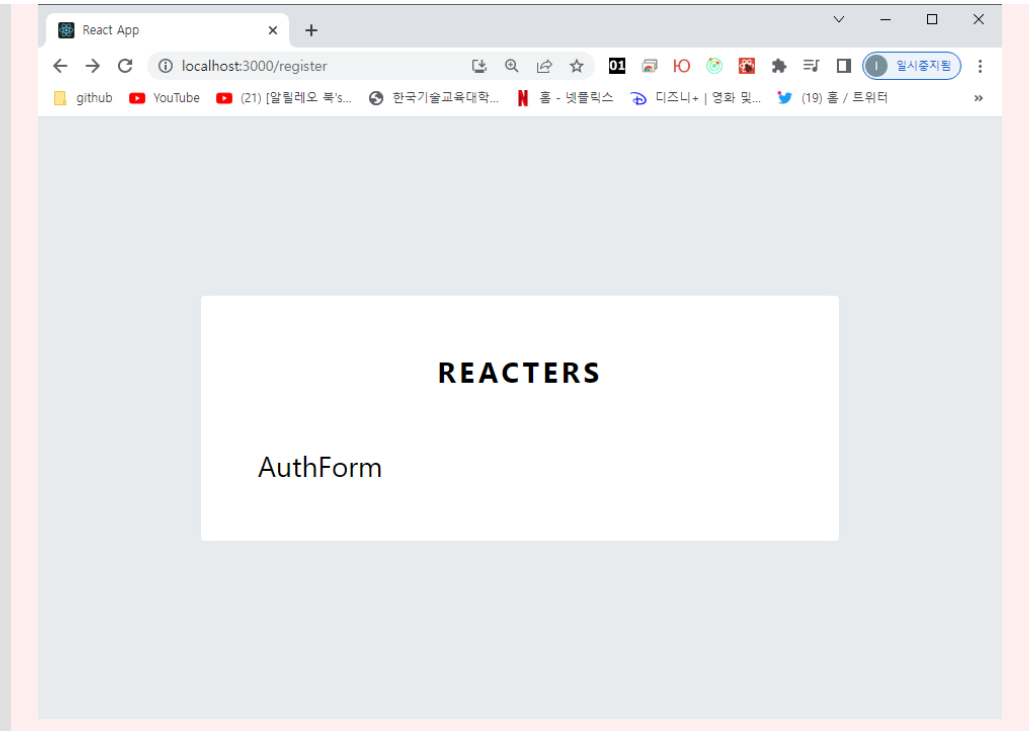
pages\RegisterPage.js

```
import AuthForm from '../components/auth/AuthForm';
import AuthTemplate from '../components/auth/AuthTemplate';

const RegisterPage = () => {
  return (
    <AuthTemplate>
      <AuthForm />
    </AuthTemplate>
  )
}

export default RegisterPage;
```

24.2.1.1 AutuTemplate.js 완성하기

coponents\autu\AutuTemplate.js

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

/**
 * 회원가입 / 로그인 페이지의 레이아웃을 담당하는 컴포넌트입니다.
 */

/* 화면 전체를 채움 */
const AuthTemplateBlock = styled.div`
  position: absolute;
  left: 0;
  top: 0;
  bottom: 0;
  right: 0;
  background: ${palette.gray[2]};
  /* flex로 내부 내용 중앙 정렬 */
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
`;

/* 흰색 박스 */
const WhiteBox = styled.div`
  .logo-area {
    display: block;
    padding-bottom: 2rem;
    text-align: center;
    font-weight: bold;
    letter-spacing: 2px;
  }
  box-shadow: 0 0 8px rgba(0, 0, 0, 0.025);
  padding: 2rem;
  width: 360px;
  background: white;
  border-radius: 2px;
`;

const AuthTemplate = ({ children }) => {
  return (
    <AuthTemplateBlock>
      <WhiteBox>
        <div className="logo-area">
          <Link to="/">REACTERS</Link>
        </div>
        {children}
      </WhiteBox>
    </AuthTemplateBlock>
  );
};

export default AuthTemplate;
```



24.2.1.2 AutuForm.js 완성하기

coponents\autu\AutuForm.js

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

/**
 * 회원가입 또는 로그인 폼을 보여줍니다.
 */

const AuthFormBlock = styled.div`
  h3 {
    margin: 0;
    color: ${palette.gray[8]};
    margin-bottom: 1rem;
  }
`;

/**
 * 스타일링된 input
 */
const StyledInput = styled.input`
  font-size: 1rem;
  border: none;
  border-bottom: 1px solid ${palette.gray[5]};
  padding-bottom: 0.5rem;
  outline: none;
  width: 100%;
  &:focus {
    color: $oc-teal-7;
    border-bottom: 1px solid ${palette.gray[7]};
  }
```

```
  & + & {
    margin-top: 1rem;
  }
`;

/**
 * 폼 하단에 로그인 혹은 회원가입 링크를 보여줌
 */
const Footer = styled.div`
  margin-top: 2rem;
  text-align: right;
  a {
    color: ${palette.gray[6]};
    text-decoration: underline;
    &:hover {
      color: ${palette.gray[9]};
    }
  }
`;

const ButtonWithMarginTop = styled(Button)`
  margin-top: 1rem;
`;

const textMap = {
  login: '로그인',
  register: '회원가입'
};

const AuthForm = ({ type }) => {
  const text = textMap[type];
  return (
    <AuthFormBlock>
      <h3>{text}</h3>
      <form action="" method="get">
        <StyledInput autoComplete='usename' name='username' placeholder='아이
디'/>
        <StyledInput
          autoComplete='new-password'
          name='password'
          placeholder='비밀번호'
          type='password'
        />
        {type === 'register' && (
          <StyledInput
            autoComplete='new-password'
            name='passwordConfirm'
            placeholder='비밀번호 확인'
            type='password'
          />
        )}
        <ButtonWithMarginTop cyan fullWidth style={{ maringTop: '1rem'}}>
          {text}
        </ButtonWithMarginTop>
      </form>
      <Footer>
        {type === 'login'
```

```
            ? (<Link to="/register">회원가입</Link>)
            : (<Link to="/login">로그인</Link>)
          }
        </Footer>
      </AuthFormBlock>
    );
  }

export default AuthForm;
```

components\common\Button.js

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

const StyledButton = styled.button`
  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]};
      }
    `}
`;

const Button = props => <StyledButton {...props} />;

export default Button;
```

pages\LoginPage.js

```
import AuthTemplate from '../components/auth/AuthTemplate';
import AuthForm from '../components/auth/AuthForm';
```

```
const LoginPage = () => {
  return (
    <AuthTemplate>
      <AuthForm type='login' />
    </AuthTemplate>
  );
};

export default LoginPage;
```

pages\RegisterPage.js

```
import AuthTemplate from '../components/auth/AuthTemplate';
import AuthForm from '../components/auth/AuthForm';

const RegisterPage = () => {
  return (
    <AuthTemplate>
      <AuthForm  type='register'/>
    </AuthTemplate>
  )
}

export default RegisterPage;
```

24.2.2 리덕스로 폼 상태 관리하기

modules\auth.js

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

const CHANGE_FIELD = 'auth/CHANGE_FIELD';
const INITIALIZE_FORM = 'auth/INITIALIZE_FORM';

export const changeField = createAction(
  CHANGE_FIELD,
  ({ form, key, value }) => ({
    form, // register , login
    key, // username, password, passwordConfirm
    value // 실제 바꾸려는 값
  })
);

export const initializeForm = createAction(INITIALIZE_FORM, form => form); //
register / login

const initialState = {
  register: {
    username: '',
    password: '',
    passwordConfirm: ''
  },
  login: {
    username: '',
    password: ''
  },
};
```

```
const auth = handleActions(
  {
    [CHANGE_FIELD]: (state, { payload: { form, key, value } }) =>
      produce(state, draft => {
        draft[form][key] = value; // 예: state.register.username을 바꾼다
      }),
    [INITIALIZE_FORM]: (state, { payload: form }) => ({
      ...state,
      [form]: initialState[form],
    }),
  },
  initialState
)

export default auth;
```

containers\auth\LoginForm.js

```
import { useEffect} from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { changeField, initializeForm } from '../../modules/auth';
import AuthForm from '../../components/auth/AuthForm';


const LoginForm = () => {
  const dispatch = useDispatch();
  const { form } = useSelector(({ auth }) => ({
    form: auth.login,
  }));
  // 인풋 변경 이벤트 핸들러
  const onChange = (e) => {
    const { value, name } = e.target;
    dispatch(
      changeField({
        form: 'login',
        key: name,
        value,
      }),
    );
  };

  // 폼 등록 이벤트 핸들러
  const onSubmit = (e) => {
    e.preventDefault();
  };

  // 컴포넌트가 처음 렌더링 될 때 form 을 초기화함
  useEffect(() => {
    dispatch(initializeForm('login'));
  }, [dispatch]);

  return (
    <AuthForm
      type="login"
      form={form}
      onChange={onChange}
```

```
        onSubmit={onSubmit}
      />
    );
};

export default LoginForm;
```

pages\LoginPage.js

```
import AuthTemplate from '../components/auth/AuthTemplate';
import LoginForm from '../containers/auth/LoginForm';

const LoginPage = () => {
  return (
    <AuthTemplate>
      <LoginForm />
    </AuthTemplate>
  );
};

export default LoginPage;
```

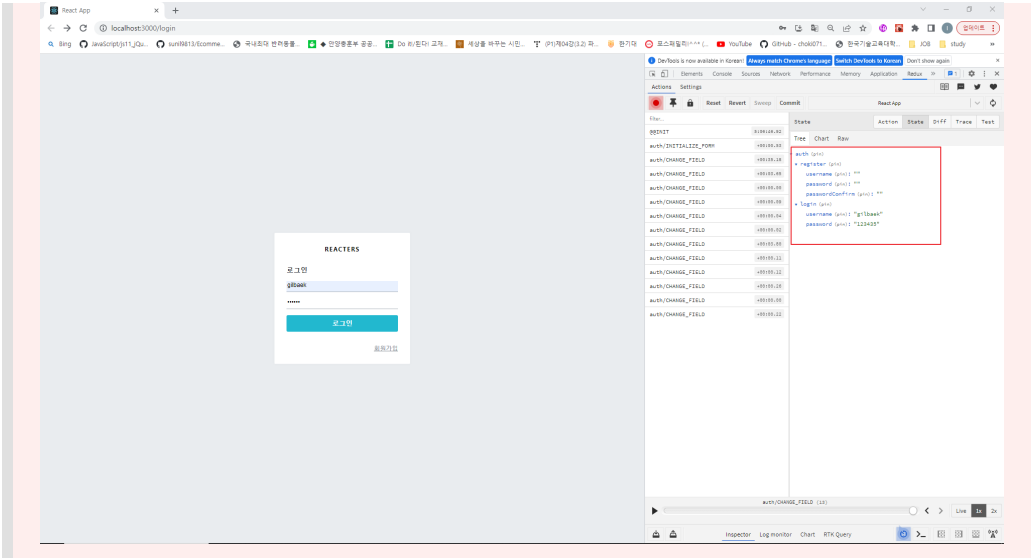
components\auth\AuthForm.js

- http://localhost:3000/login
  - input태그에 입력하고 개발자도구에서 redux 스토어에 들어가는지 확인

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

/**
 * 회원가입 또는 로그인 폼을 보여줍니다.
 */

const AuthFormBlock = styled.div`
  h3 {
    margin: 0;
    color: ${palette.gray[8]};
    margin-bottom: 1rem;
  }
`;

/**
 * 스타일링된 input
 */
const StyledInput = styled.input`
  font-size: 1rem;
  border: none;
  border-bottom: 1px solid ${palette.gray[5]};
  padding-bottom: 0.5rem;
  outline: none;
  width: 100%;
  &:focus {
    color: $oc-teal-7;
    border-bottom: 1px solid ${palette.gray[7]};
```

```
  }
  & + & {
    margin-top: 1rem;
  }
`;

/**
 * 폼 하단에 로그인 혹은 회원가입 링크를 보여줌
 */
const Footer = styled.div`
  margin-top: 2rem;
  text-align: right;
  a {
    color: ${palette.gray[6]};
    text-decoration: underline;
    &:hover {
      color: ${palette.gray[9]};
    }
  }
`;

const ButtonWithMarginTop = styled(Button)`
  margin-top: 1rem;
`;

const textMap = {
  login: '로그인',
  register: '회원가입'
};

const AuthForm = ({ type, form, onChange, onSubmit }) => {
  const text = textMap[type];
  return (
    <AuthFormBlock>
      <h3>{text}</h3>
      <form onSubmit={onSubmit}>
        <StyledInput
          autoComplete='usename'
          name='username'
          placeholder='아이디'
          onChange={onChange}
          value={form.username}
        />
        <StyledInput
          autoComplete='new-password'
          name='password'
          placeholder='비밀번호'
          type='password'
          onChange={onChange}
          value={form.password}
        />
        {type === 'register' && (
          <StyledInput
            autoComplete='new-password'
            name='passwordConfirm'
            placeholder='비밀번호 확인'
            type='password'
```

```
            onChange={onChange}
            value={form.passwordConfirm}
          />
        )}
        <ButtonWithMarginTop cyan fullWidth style={{ maringTop: '1rem'}}>
          {text}
        </ButtonWithMarginTop>
      </form>
      <Footer>
        {type === 'login'
          ? (<Link to="/register">회원가입</Link>)
          : (<Link to="/login">로그인</Link>)
        }
      </Footer>
    </AuthFormBlock>
  );
}

export default AuthForm;
```



containers\auth\RegisterForm.js

- LoginForm.js를 복사 RegisterForm.js
- Login -> Register, login -> register로 변경

```
import { useEffect} from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { changeField, initializeForm } from '../../modules/auth';
import AuthForm from '../../components/auth/AuthForm';


const RegisterForm = () => {
  const dispatch = useDispatch();
  const { form } = useSelector(({ auth }) => ({
    form: auth.register,
  }));
  // 인풋 변경 이벤트 핸들러
  const onChange = (e) => {
    const { value, name } = e.target;
    dispatch(
      changeField({
```

```
            form: 'register',
            key: name,
            value,
          }),
        );
      };

      // 폼 등록 이벤트 핸들러
      const onSubmit = (e) => {
        e.preventDefault();
      };

      // 컴포넌트가 처음 렌더링 될 때 form 을 초기화함
      useEffect(() => {
        dispatch(initializeForm('register'));
      }, [dispatch]);

      return (
        <AuthForm
          type="register"
          form={form}
          onChange={onChange}
          onSubmit={onSubmit}
        />
      );
    };

    export default RegisterForm;
```

pages\RegisterPage.js

```
    import AuthTemplate from '../components/auth/AuthTemplate';
    import RegisterForm from '../containers/auth/RegisterForm';

    const RegisterPage = () => {
      return (
        <AuthTemplate>
          <RegisterForm />
        </AuthTemplate>
      )
    }

    export default RegisterPage;
```

24.2.3 API 연동하기

- redux-sega설치 및 이전 작성한 createRequestSega 유틸함수 이용
- 설치 : `yarn add axios redux-saga`

24.2.3.1 axios 인스턴스 생성

src\lib\api\client.js

- API함수를 작성하기 전에 axios인스턴스를 생성, 이렇게 하면 나중에 API클라이언트에 공통설정을 쉽게 넣어줄 수 있다.

- 인스턴스를 만들지 않아도 이러한 작업을 할 수는 있지만 생성하지 않았을 경우 모든 요청에 대해 설정하게 된다.
- 처음부터 개발할 때 인스턴스를 만들어서 작업할 것을 권장한다.

```
import axios from 'axios';

const client = axios.create();

/*
  글로벌 설정 예시:

  // API 주소를 다른 곳으로 사용함
  client.defaults.baseURL = 'https://external-api-server.com/'

  // 헤더 설정
  client.defaults.headers.common['Authorization'] = 'Bearer a1b2c3d4';

  // 인터셉터 설정
  axios.intercepter.response.use(\
    response => {
      // 요청 성공 시 특정 작업 수행
      return response;
    },
    error => {
      // 요청 실패 시 특정 작업 수행
      return Promise.reject(error);
    }
  })
*/

export default client;
```

24.2.3.2 프록시설정

- 백앤드는 4000, 프론트는 3000을 사용하기 때문에 별도의 설정없이 API호출시 에러 발생
- 이런 에러를 CORS(Cross Orign Request)에러라고 한다.
- 이를 해결하려면 package.json에 proxy기능을 설정(웹팩개발서버에서 지원하는 기능)
  - 임의의 위치세 `"proxy": "http://localhost:4000/"`으로 설정
- 리액트에서 client.get('/api/posts')를 요청하면 웹팩개발서버가 프록시역할을 해서 http://localhost:4000/api/posts 요청
- `설정후 서버 재시작`

24.2.3.3 API함수 작성

lib/api/auth.js

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

// 로그인
export const login = ({ username, password }) =>
  client.post('/api/auth/login', { username, password });

// 회원가입
```

```
export const register = ({ username, password }) =>
  client.post('/api/auth/register', { username, password });

// 로그인 상태 확인
export const check = () => client.post('/api/auth/check');

// 로그아웃
export const logout = () => client.post('/api/auth/logout');
```

24.2.3.4 더 쉬운 API요청 상태관리

- redux-saga를 통해 더 쉽게 API를 요청할 수 있도록 loading리덕스모듈과 createRequestSaga 유틸함수를 설정

modules\loading.js

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

const START_LOADING = 'loading/START_LOADING';
const FINISH_LOADING = 'loading/FINISH_LOADING';

/*
 요청을 위한 액션 타입을 payload로 설정합니다 (예: "sample/GET_POST")
*/

export const startLoading = createAction(
  START_LOADING,
  requestType => requestType
);

export const finishLoading = createAction(
  FINISH_LOADING,
  requestType => requestType
);

const initialState = {};

const loading = handleActions(
  {
    [START_LOADING]: (state, action) => ({
      ...state,
      [action.payload]: true
    }),
    [FINISH_LOADING]: (state, action) => ({
      ...state,
      [action.payload]: false
    })
  },
  initialState
);

export default loading;
```

modules\index.js - 리덕스에 등록

```
import { combineReducers } from 'redux';
import auth from './auth';
import loading from './loading';
```

```
const rootReducer = combineReducers({
  auth,
  loading,
})

export default rootReducer;
```

lib\createRequestSaga.js

```
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
      });
    } catch (e) {
      yield put({
        type: FAILURE,
        payload: e,
        error: true
      });
    }
    yield put(finishLoading(type)); // 로딩 끝
  };
}
```

24.2.3.5 auth리덕스 모듈에서 API사용하기

modules\auth.js - 6가지 액션타입 추가

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

const CHANGE_FIELD = 'auth/CHANGE_FIELD';
const INITIALIZE_FORM = 'auth/INITIALIZE_FORM';

const [REGISTER, REGISTER_SUCCESS, REGISTER_FAILURE] =
createRequestActionTypes(
```

```
  'auth/REGISTER'
);

const [LOGIN, LOGIN_SUCCESS, LOGIN_FAILURE] = createRequestActionTypes(
  'auth/LOGIN'
);

export const changeField = createAction(
  CHANGE_FIELD,
  ({ form, key, value }) => ({
    form, // register , login
    key, // username, password, passwordConfirm
    value // 실제 바꾸려는 값
  })
);

export const initializeForm = createAction(INITIALIZE_FORM, form => form); //
register / login

export const register = createAction(REGISTER, ({ username, password }) => ({
  username,
  password
}));

export const login = createAction(LOGIN, ({ username, password }) => ({
  username,
  password
}));

// saga 생성
const registerSaga = createRequestSaga(REGISTER, authAPI.register);
const loginSaga = createRequestSaga(LOGIN, authAPI.login);
export function* authSaga() {
  yield takeLatest(REGISTER, registerSaga);
  yield takeLatest(LOGIN, loginSaga);
}

const initialState = {
  register: {
    username: '',
    password: '',
    passwordConfirm: ''
  },
  login: {
    username: '',
    password: ''
  },
  auth: null,
  authError: null
};

const auth = handleActions(
  {
    [CHANGE_FIELD]: (state, { payload: { form, key, value } }) =>
      produce(state, draft => {
        draft[form][key] = value; // 예: state.register.username을 바꾼다
      }),
```

```
    [INITIALIZE_FORM]: (state, { payload: form }) => ({
      ...state,
      [form]: initialState[form],
      authError: null // 폼 전환 시 회원 인증 에러 초기화
    }),
    // 회원가입 성공
    [REGISTER_SUCCESS]: (state, { payload: auth }) => ({
      ...state,
      authError: null,
      auth
    }),
    // 회원가입 실패
    [REGISTER_FAILURE]: (state, { payload: error }) => ({
      ...state,
      authError: error
    }),
    // 로그인 성공
    [LOGIN_SUCCESS]: (state, { payload: auth }) => ({
      ...state,
      authError: null,
      auth
    }),
    // 로그인 실패
    [LOGIN_FAILURE]: (state, { payload: error }) => ({
      ...state,
      authError: error
    })
  },
  initialState,
);

export default auth;
```

modules\index.js - rootSaga 작성

```
import { combineReducers } from 'redux';
import { all } from 'redux-saga/effects';
import auth, { authSaga } from './auth';
import loading from './loading';

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

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

export default rootReducer;
```

src\index.js - redux-saga 미들웨어 적용

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import { BrowserRouter } from 'react-router-dom';
```

```
import { Provider } from 'react-redux';
import { legacy_createStore as  createStore, applyMiddleware } from 'redux';
import { composeWithDevTools } from 'redux-devtools-extension';
import rootReducer, { rootSaga } from './modules';
import createSagaMiddleware from 'redux-saga';


const sagaMiddleware = createSagaMiddleware();

const store = createStore(
  rootReducer,
  composeWithDevTools(applyMiddleware(sagaMiddleware)),
  );

sagaMiddleware.run(rootSaga);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Provider store={store}>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </Provider>
);
```

23.2.4 회원가입구현

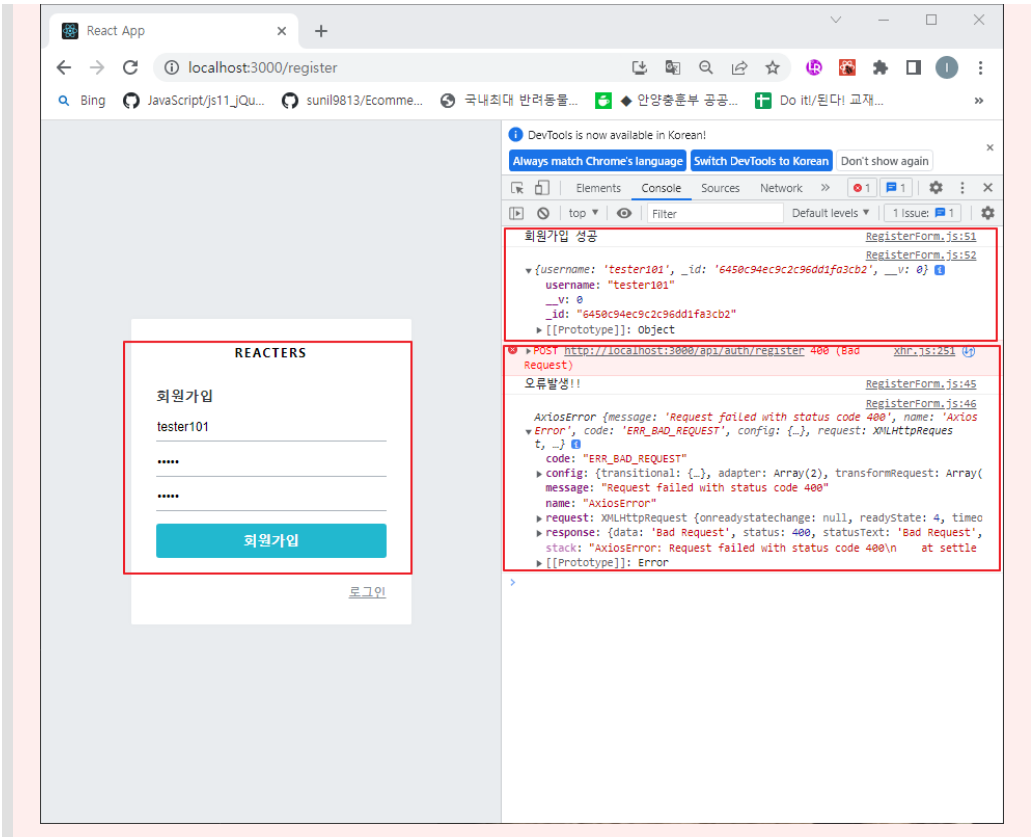
containers\auth\RegisterForm.js

- onSubmit이벤트발생시 register함수에 username, password를 넣어서 액션을 디스패치 한다.
- 그리고, 사가에서 API요청을 처리하고 결과를 auth/authError를 통해 조회가능
- 또한, 결과획득시 특정작업진행을 위해 useEffect를 사용
- useEffect에 전달한 함수는 auth, authError값의 유효여부에 따라 다른 작업을 진행

```
import { useEffect} from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { changeField, initializeForm, register } from '../../modules/auth';
import AuthForm from '../../components/auth/AuthForm';

const RegisterForm = () => {
  const dispatch = useDispatch();
  const { form, auth, authError } = useSelector(({ auth }) => ({
    form: auth.register,
    auth: auth.auth,
    authError: auth.authError,
  }));
  // 인풋 변경 이벤트 핸들러
  const onChange = (e) => {
    const { value, name } = e.target;
    dispatch(
      changeField({
        form: 'register',
        key: name,
        value,
```

```
      }),
    );
  };

  // 폼 등록 이벤트 핸들러
  const onSubmit = (e) => {
    e.preventDefault();
    const { username, password, passwordConfirm } = form;
    // 비밀번호가 일치하지 않는다면
    if (password !== passwordConfirm) {
      // Todo 오류처리
      return;
    }
    dispatch(register({ username, password }));
  };

  // 컴포넌트가 처음 렌더링 될 때 form 을 초기화함
  useEffect(() => {
    dispatch(initializeForm('register'));
  }, [dispatch]);

  // 회원가입 성공 / 실패 처리
  useEffect(() => {
    if (authError) {
      console.log('오류발생!!');
      console.log(authError);
      return;
    }

    if (auth) {
      console.log('회원가입 성공');
      console.log(auth);
    }
  }, [auth, authError]);

  return (
    <AuthForm
      type="register"
      form={form}
      onChange={onChange}
      onSubmit={onSubmit}
    />
  );
};

export default RegisterForm;
```

- blog-backend(`23.jwt>yarn start:dev`)실행 후 http://localhost:3000/register 에서 회원가입
  - test100 / 12345 /12345 가입성공후 동일사용자로 다시 가입할 경우 에러 발생 확인



modules\user.js - 사용자상태를 담은 리덕스모듈 작성

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

const TEMP_SET_USER = 'user/TEMP_SET_USER'; // 새로고침 이후 임시 로그인 처리
// 회원 정보 확인
const [CHECK, CHECK_SUCCESS, CHECK_FAILURE] = createRequestActionTypes(
  'user/CHECK',
);

export const tempSetUser = createAction(TEMP_SET_USER, user => user);
export const check = createAction(CHECK);

const checkSaga = createRequestSaga(CHECK, authAPI.check);

export function* userSaga() {
  yield takeLatest(CHECK, checkSaga);
}

const initialState = {
  user: null,
  checkError: null,
};

export default handleActions(
  {
    [TEMP_SET_USER]: (state, { payload: user }) => ({
      ...state,
      user,
```

```
    }),
    [CHECK_SUCCESS]: (state, { payload: user }) => ({
      ...state,
      user,
      checkError: null,
    }),
    [CHECK_FAILURE]: (state, { payload: error }) => ({
      ...state,
      user: null,
      checkError: error,
    }),
  },
  initialState,
);
```

modules\index.js - user.js모듈을 루트 리듀서에 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'

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

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

export default rootReducer;
```

containers\auth\RegisterForm.js - 회원가입성공후 check를 호출, 사용자로그인상태확인

- http://localhost:3000/ 이동확인

```
import { useEffect} from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { changeField, initializeForm, register } from '../../modules/auth';
import AuthForm from '../../components/auth/AuthForm';
import { check } from '../../modules/user';
import { useNavigate } from 'react-router-dom'

const RegisterForm = () => {
  const dispatch = useDispatch();
  const { form, auth, authError, user } = useSelector(({ auth, user }) => ({
    form: auth.register,
    auth: auth.auth,
    authError: auth.authError,
    user: user.user
  }));

  const navigate = useNavigate();
```

```
// 인풋 변경 이벤트 핸들러
const onChange = (e) => {
  const { value, name } = e.target;
  dispatch(
    changeField({
      form: 'register',
      key: name,
      value,
    }),
  );
};

// 폼 등록 이벤트 핸들러
const onSubmit = (e) => {
  e.preventDefault();
  const { username, password, passwordConfirm } = form;
  // 비밀번호가 일치하지 않는다면
  if (password !== passwordConfirm) {
    // Todo 오류처리
    return;
  }
  dispatch(register({ username, password }));
};

// 컴포넌트가 처음 렌더링 될 때 form 을 초기화함
useEffect(() => {
  dispatch(initializeForm('register'));
}, [dispatch]);

// 회원가입 성공 / 실패 처리
useEffect(() => {
  if (authError) {
    console.log('오류발생!!');
    console.log(authError);
    return;
  }

  if (auth) {
    console.log('회원가입 성공');
    console.log(auth);
    dispatch(check())
  }
}, [auth, authError, dispatch]);

// user 값이 잘 설정되었는지 확인
useEffect(() => {
  if (user) {
    // console.log('Checl API 성공');
    // console.log(user);
    navigate('/'); // 홈 화면으로 이동
  }
}, [navigate, user]);

return (
  <AuthForm
    type="register"
    form={form}
```

```
        onChange={onChange}
        onSubmit={onSubmit}
      />
    );
};

export default RegisterForm;
```

24.2.5 로그인 구현

containers\auth\LoginForm.js

```
import { useEffect} from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { changeField, initializeForm, login } from '../../modules/auth';
import AuthForm from '../../components/auth/AuthForm';
import { check } from '../../modules/user'
import { useNavigate } from 'react-router-dom'

const LoginForm = () => {
  const navigate = useNavigate();
  const dispatch = useDispatch();
  const { form, auth, authError, user } = useSelector(({ auth, user }) => ({
    form: auth.login,
    auth: auth.auth,
    authError: auth.authError,
    user: user.user,
  }));
  // 인풋 변경 이벤트 핸들러
  const onChange = (e) => {
    const { value, name } = e.target;
    dispatch(
      changeField({
        form: 'login',
        key: name,
        value,
      }),
    );
  };

  // 폼 등록 이벤트 핸들러
  const onSubmit = (e) => {
    e.preventDefault();
    const { username, password } = form;
    dispatch(login({ username, password }));
  };

  // 컴포넌트가 처음 렌더링 될 때 form 을 초기화함
  useEffect(() => {
    dispatch(initializeForm('login'));
  }, [dispatch]);

  useEffect(() => {
    if (authError) {
      console.log('오류 발생');
      console.log(authError);
      return;
```

```
    }
    if (auth) {
      console.log('로그인 성공');
      dispatch(check());
    }
  }, [auth, authError, dispatch]);

  useEffect(() => {
    if (user) {
      navigate('/');
    }
  }, [navigate, user]);

  return (
    <AuthForm
      type="login"
      form={form}
      onChange={onChange}
      onSubmit={onSubmit}
    />
  );
};

export default LoginForm;
```

24.2.6 회원인증 에러 처리하기

components\auth\AuthForm.js

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

/**
 * 회원가입 또는 로그인 폼을 보여줍니다.
 */

const AuthFormBlock = styled.div`
  h3 {
    margin: 0;
    color: ${palette.gray[8]};
    margin-bottom: 1rem;
  }
`;

/**
 * 스타일링된 input
 */
const StyledInput = styled.input`
  font-size: 1rem;
  border: none;
  border-bottom: 1px solid ${palette.gray[5]};
  padding-bottom: 0.5rem;
  outline: none;
  width: 100%;
```

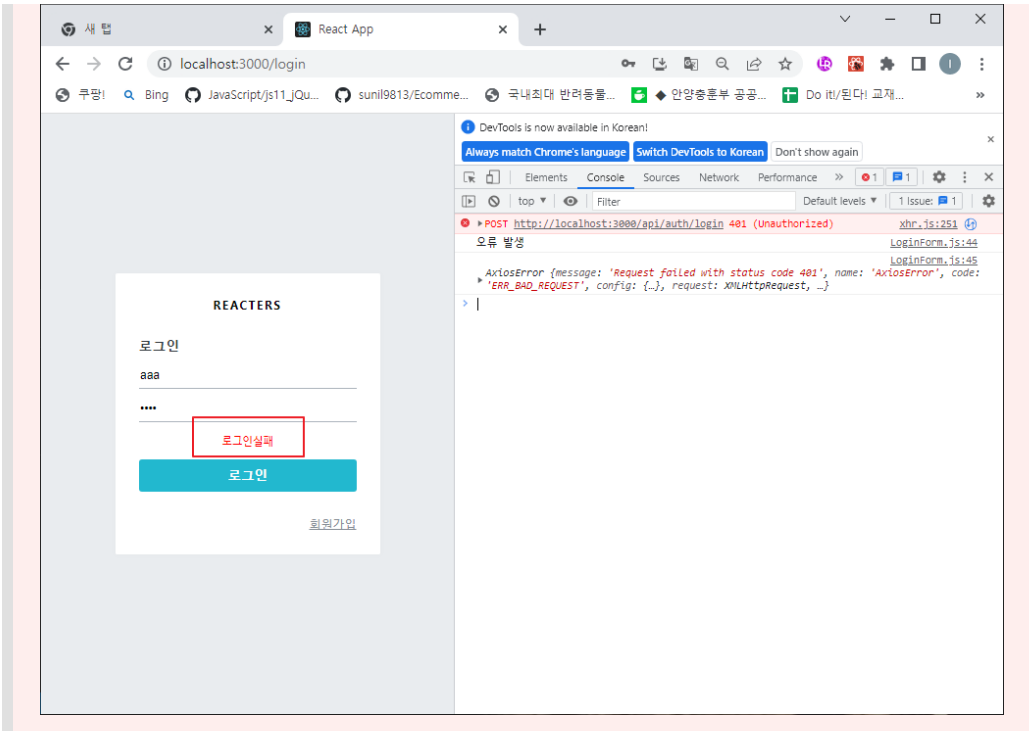
```
  &:focus {
    color: $oc-teal-7;
    border-bottom: 1px solid ${palette.gray[7]};
  }
  & + & {
    margin-top: 1rem;
  }
`;

/**
 * 폼 하단에 로그인 혹은 회원가입 링크를 보여줌
 */
const Footer = styled.div`
  margin-top: 2rem;
  text-align: right;
  a {
    color: ${palette.gray[6]};
    text-decoration: underline;
    &:hover {
      color: ${palette.gray[9]};
    }
  }
`;

const ButtonWithMarginTop = styled(Button)`
  margin-top: 1rem;
`;

const textMap = {
  login: '로그인',
  register: '회원가입'
};

/**
 * 에러를 보여줍니다
 */
const ErrorMessage = styled.div`
  color: red;
  text-align: center;
  font-size: 0.875rem;
  margin-top: 1rem;
`;

const AuthForm = ({ type, form, onChange, onSubmit, error }) => {
  const text = textMap[type];
  return (
    <AuthFormBlock>
      <h3>{text}</h3>
      <form onSubmit={onSubmit}>
        <StyledInput
          autoComplete='usename'
          name='username'
          placeholder='아이디'
          onChange={onChange}
          value={form.username}
        />
        <StyledInput
```

```
            autoComplete='new-password'
            name='password'
            placeholder='비밀번호'
            type='password'
            onChange={onChange}
            value={form.password}
          />
          {type === 'register' && (
            <StyledInput
              autoComplete='new-password'
              name='passwordConfirm'
              placeholder='비밀번호 확인'
              type='password'
              onChange={onChange}
              value={form.passwordConfirm}
            />
          )}
          {error && <ErrorMessage>{error}</ErrorMessage>}
          <ButtonWithMarginTop cyan fullWidth style={{ maringTop: '1rem'}}>
            {text}
          </ButtonWithMarginTop>
        </form>
        <Footer>
          {type === 'login'
            ? (<Link to="/register">회원가입</Link>)
            : (<Link to="/login">로그인</Link>)
          }
        </Footer>
      </AuthFormBlock>
    );
  }

  export default AuthForm;
```

containers\auth\LoginForm.js - 로그인인증처리

```
  import { useEffect, useState} from 'react';
  import { useSelector, useDispatch } from 'react-redux';
  import { changeField, initializeForm, login } from '../../modules/auth';
  import AuthForm from '../../components/auth/AuthForm';
  import { check } from '../../modules/user'
  import { useNavigate } from 'react-router-dom'

  const LoginForm = () => {
    const [error, setError] = useState(null);
    const navigate = useNavigate();
    const dispatch = useDispatch();
    const { form, auth, authError, user } = useSelector(({ auth, user }) => ({
      form: auth.login,
      auth: auth.auth,
      authError: auth.authError,
      user: user.user,
    }));
    // 인풋 변경 이벤트 핸들러
    const onChange = (e) => {
      const { value, name } = e.target;
      dispatch(
```

```
        changeField({
          form: 'login',
          key: name,
          value,
        }),
      );
    };

    // 폼 등록 이벤트 핸들러
    const onSubmit = (e) => {
      e.preventDefault();
      const { username, password } = form;
      dispatch(login({ username, password }));
    };

    // 컴포넌트가 처음 렌더링 될 때 form 을 초기화함
    useEffect(() => {
      dispatch(initializeForm('login'));
    }, [dispatch]);

    useEffect(() => {
      if (authError) {
        console.log('오류 발생');
        console.log(authError);
        setError('로그인실패');
        return;
      }
      if (auth) {
        console.log('로그인 성공');
        dispatch(check());
      }
    }, [auth, authError, dispatch]);

    useEffect(() => {
      if (user) {
        navigate('/');
      }
    }, [navigate, user]);

    return (
      <AuthForm
        type="login"
        form={form}
        onChange={onChange}
        onSubmit={onSubmit}
        error={error}
      />
    );
  };

  export default LoginForm;
```



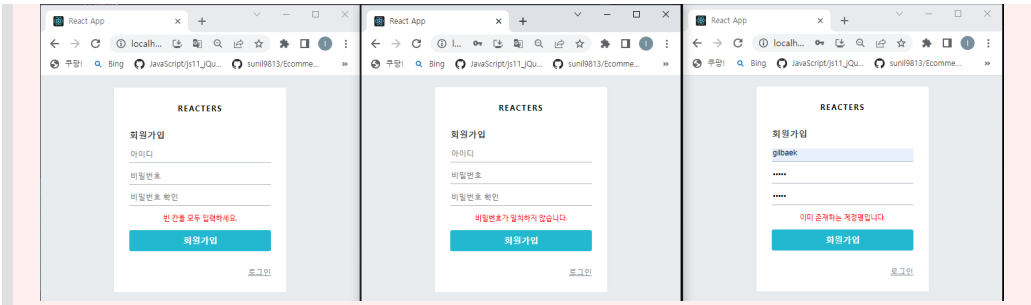
containers\auth\RegisterForm.js - 회원가입에러처리

- username, password, passwordConfirm 공란처리
- password, passwordConfirm 불일치
- username 중복처리

```
import { useEffect, useState} from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { changeField, initializeForm, register } from '../../modules/auth';
import AuthForm from '../../components/auth/AuthForm';
import { check } from '../../modules/user';
import { useNavigate } from 'react-router-dom'

const RegisterForm = () => {
  const [error, setError] = useState(null);
  const dispatch = useDispatch();
  const { form, auth, authError, user } = useSelector(({ auth, user }) => ({
    form: auth.register,
    auth: auth.auth,
    authError: auth.authError,
    user: user.user
  }));

  const navigate = useNavigate();

  // 인풋 변경 이벤트 핸들러
  const onChange = (e) => {
    const { value, name } = e.target;
    dispatch(
      changeField({
        form: 'register',
        key: name,
        value,
      }),
    );
  };
```

```
// 폼 등록 이벤트 핸들러
const onSubmit = (e) => {
  e.preventDefault();
  const { username, password, passwordConfirm } = form;
  // 하나라도 비어있다면
  if ([username, password, passwordConfirm].includes('')) {
    setError('빈 칸을 모두 입력하세요.');
    return;
  }
  // 비밀번호가 일치하지 않는다면
  if (password !== passwordConfirm) {
    setError('비밀번호가 일치하지 않습니다.');
    dispatch(changeField({ form: 'register', key: 'password', value: '' }));
    dispatch(
      changeField({ form: 'register', key: 'passwordConfirm', value: '' }),
    );
    return;
  }
  dispatch(register({ username, password }));
};

// 컴포넌트가 처음 렌더링 될 때 form 을 초기화함
useEffect(() => {
  dispatch(initializeForm('register'));
}, [dispatch]);

// 회원가입 성공 / 실패 처리
useEffect(() => {
  if (authError) {
    // 계정명이 이미 존재할 때
    if (authError.response.status === 400) {
      setError('이미 존재하는 계정명입니다.');
      return;
    }
    // 기타 이유
    setError('회원가입 실패');
    return;
  }

  if (auth) {
    console.log('회원가입 성공');
    console.log(auth);
    dispatch(check())
  }
}, [auth, authError, dispatch]);

// user 값이 잘 설정되었는지 확인
useEffect(() => {
  if (user) {
    // console.log('Checl API 성공');
    // console.log(user);
    navigate('/'); // 홈 화면으로 이동
  }
}, [navigate, user]);

return (
```

```
      <AuthForm
        type="register"
        form={form}
        onChange={onChange}
        onSubmit={onSubmit}
        error={error}
      />
    );
};

export default RegisterForm;
```



24.3 헤더 컴퍼넌트 생성 및 로그인 유지

24.3.1 헤더 컴퍼넌트 만들기

components\common/Responsive.js

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

const ResponsiveBlock = styled.div`
  padding-left: 1rem;
  padding-right: 1rem;
  width: 1024px;
  margin: 0 auto; /* 중앙 정렬 */

  /* 브라우저 크기에 따라 가로 사이즈 변경 */
  @media (max-width: 1024px) {
    width: 768px;
  }
  @media (max-width: 768px) {
    width: 100%;
  }
`;

const Responsive = ({ children, ...rest }) => {
  // style, className, onClick, onMouseMove 등의 props를 사용할 수 있도록
  // ...rest를 사용하여 ResponsiveBlock에게 전달
  return <ResponsiveBlock {...rest}>{children}</ResponsiveBlock>;
};

export default Responsive;
```

components\common/Header.js

```
import styled from 'styled-components';
import Responsive from './Responsive';
import Button from './Button';
import { Link } from 'react-router-dom';

const HeaderBlock = styled.div`
  position: fixed;
  width: 100%;
  background: white;
  box-shadow: 0px 2px 4px rgba(0, 0, 0, 0.08);
`;

/**
 * Responsive 컴포넌트의 속성에 스타일을 추가해서 새로운 컴포넌트 생성
 */
const Wrapper = styled(Responsive)`
  height: 4rem;
  display: flex;
  align-items: center;
  justify-content: space-between; /* 자식 엘리먼트 사이에 여백을 최대로 설정 */
  .logo {
    font-size: 1.125rem;
    font-weight: 800;
    letter-spacing: 2px;
  }
  .right {
    display: flex;
    align-items: center;
  }
`;

/**
 * 헤더가 fixed로 되어 있기 때문에 페이지의 컨텐츠가 4rem 아래 나타나도록 해주
는 컴포넌트
 */
const Spacer = styled.div`
  height: 4rem;
`;

const UserInfo = styled.div`
  font-weight: 800;
  margin-right: 1rem;
`;

const Header = ({ user, onLogout }) => {
  return (
    <>
      <HeaderBlock>
        <Wrapper>
          <Link to="/" className="logo">
            REACTERS
          </Link>
          {user ? (
            <div className="right">
              <UserInfo>{user.username}</UserInfo>
              <Button onClick={onLogout}>로그아웃</Button>
            </div>
```

```
            ) : (
              <div className="right">
                <Button to="/login">로그인</Button>
              </div>
            )}
          </Wrapper>
        </HeaderBlock>
        <Spacer />
      </>
    );
};

export default Header;
```

components\common/Responsive.js

```
import Header from "../components/common/Header";

const PostListPage = () => {
  return (
    <>
      <Header />
      <div>안녕하세요?</div>
    </>
  );
};
export default PostListPage;
```

components\common\Button.js - 1. useNavigate를 사용

- 로그인번튼 클릭하면 `/login` 페이지로 이동
  - button컴퍼넌트를 Link처럼 작동시키는 2가지 방법은
    1. useNavigate를 이용, to값이 이쓸 경우 페이지 이동하도록 구현헌 뒤 to값을 props를 전달하면 Link처럼 작동
    2. Link 컴퍼넌트를 직접 사용

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

const StyledButton = styled.button`
  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]};
      }
    `}
`;

const Button = ({ to, ...rest}) => {

  const navigate = useNavigate();
  const onClick = e => {
    // to가 있다면 to로 페이지 이동
    if (to) {
      navigate(to);
    }
    if (rest.onClick) {
      rest.onClick(e);
    }
  }
  return <StyledButton {...rest} onClick={onClick}/>;
}

export default Button;
```

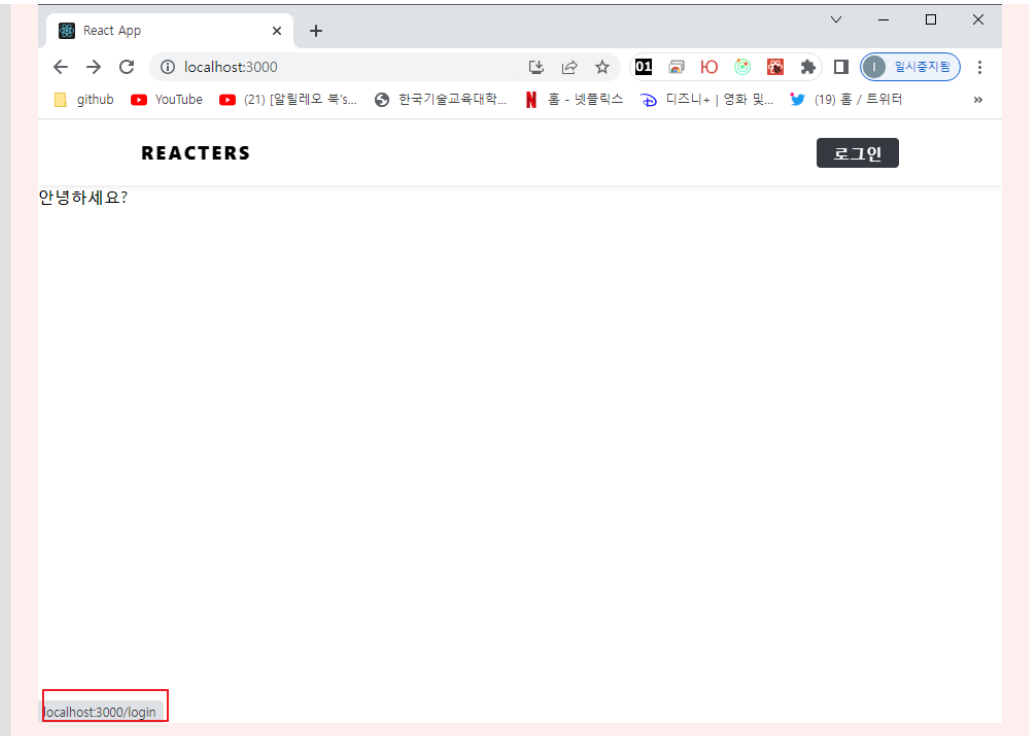
components\common\Button.js - 2. Link 컴퍼넌트를 직접 사용

- StyledLink 컴퍼넌트 추가, StyledButton과 똑같은 스타일을 사용하므로 기존 스타일을 ButtonStyle갑에 담아 재사용
- Button 내부에서 props.to값에 따라 StyledLink를 사용할지 StyledButton을 사용할지를 설정
- StyledLink를 사용하는과정에서 props.cyan값을 숫자 1과 0으로 변환
- 이렇게 한 이유는 styled()함수로 감싸서 만든 컴퍼넌트의 경우에는 임의 props가 필터링되지 않기 때문에
  - styled.button으로 만든 컴퍼넌트의 경우 cyan과 같은 임의 props가 자동으로 필터링되어 스타일을 만드는 용도로만 사용
  - 실제 button 엘리먼트에게 속성이 전달되지 않는다.
- 필터링이 되지 않으면 cyan={true}값이 Link에서 사용하는 a태그에 그대로 전달
- a태그는 boolean값이 임의 props로 설정되는 것을 허용하지 않고 수자/문자열만 허용하기 때문에 삼항연산자로 숫자로 변환

```
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]};
      }
    `}
`;

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;
```

- useNavigate보다 Link를 사용할 것을 권장, Link를 사용했을 경우 로그인버튼에 마우스를 올려놓으면 하단에 주소가 나타난다.



24.3.2 로그인상태 조회 및 유지하기

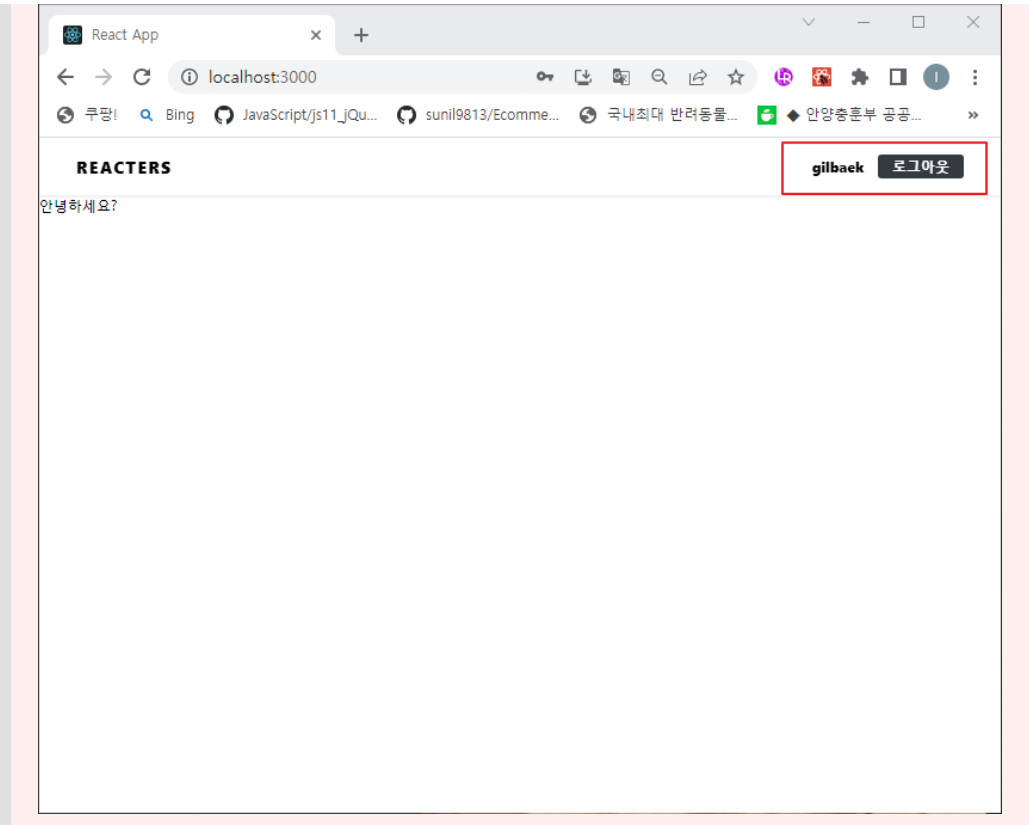
containers\commom\HeaderContainer.js

```
import { useSelector } from 'react-redux';
import Header from '../../components/common/Header';

const HeaderContainer = () => {
  const { user } = useSelector(({ user }) => ({ user: user.user }));
  return <Header user={user} />;
};

export default HeaderContainer;
```

pages\PostListPage.js

```
import HeaderContainer from '../containers/common/HeaderContainer'

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

- gilbaek/12345 로그인후 확인



24.3.2.2 로그인상태 유지하기

- 로그인상태를 유지하기 위해서 브라우저에 내장되어 있는 localstorage를 사용

containers\auth\LoginForm.js

```
import { useEffect, useState} from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { changeField, initializeForm, login } from '../../modules/auth';
import AuthForm from '../../components/auth/AuthForm';
import { check } from '../../modules/user'
import { useNavigate } from 'react-router-dom'

const LoginForm = () => {
  const [error, setError] = useState(null);
  const navigate = useNavigate();
  const dispatch = useDispatch();
  const { form, auth, authError, user } = useSelector(({ auth, user }) => ({
    form: auth.login,
    auth: auth.auth,
    authError: auth.authError,
    user: user.user,
  }));
  // 인풋 변경 이벤트 핸들러
  const onChange = (e) => {
    const { value, name } = e.target;
    dispatch(
      changeField({
        form: 'login',
        key: name,
        value,
      }),
    );
  };
```

```
  // 폼 등록 이벤트 핸들러
  const onSubmit = (e) => {
    e.preventDefault();
    const { username, password } = form;
    dispatch(login({ username, password }));
  };

  // 컴포넌트가 처음 렌더링 될 때 form 을 초기화함
  useEffect(() => {
    dispatch(initializeForm('login'));
  }, [dispatch]);

  useEffect(() => {
    if (authError) {
      console.log('오류 발생');
      console.log(authError);
      setError('로그인실패');
      return;
    }
    if (auth) {
      console.log('로그인 성공');
      dispatch(check());
    }
  }, [auth, authError, dispatch]);

  useEffect(() => {
    if (user) {
      navigate('/');
      try {
        localStorage.setItem('user', JSON.stringify(user));
      } catch (e) {
        console.log('localStorage is not working');
      }
    }
  }, [navigate, user]);

  return (
    <AuthForm
      type="login"
      form={form}
      onChange={onChange}
      onSubmit={onSubmit}
      error={error}
    />
  );
};

export default LoginForm;
```

containers\auth\RegisterForm.js

```
import { useEffect, useState} from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { changeField, initializeForm, register } from '../../modules/auth';
import AuthForm from '../../components/auth/AuthForm';
import { check } from '../../modules/user';
```

```
import { useNavigate } from 'react-router-dom'

const RegisterForm = () => {
  const [error, setError] = useState(null);
  const dispatch = useDispatch();
  const { form, auth, authError, user } = useSelector(({ auth, user }) => ({
    form: auth.register,
    auth: auth.auth,
    authError: auth.authError,
    user: user.user
  }));

  const navigate = useNavigate();

  // 인풋 변경 이벤트 핸들러
  const onChange = (e) => {
    const { value, name } = e.target;
    dispatch(
      changeField({
        form: 'register',
        key: name,
        value,
      }),
    );
  };

  // 폼 등록 이벤트 핸들러
  const onSubmit = (e) => {
    e.preventDefault();
    const { username, password, passwordConfirm } = form;
    // 하나라도 비어있다면
    if ([username, password, passwordConfirm].includes('')) {
      setError('빈 칸을 모두 입력하세요.');
      return;
    }
    // 비밀번호가 일치하지 않는다면
    if (password !== passwordConfirm) {
      setError('비밀번호가 일치하지 않습니다.');
      dispatch(changeField({ form: 'register', key: 'password', value: '' }));
      dispatch(
        changeField({ form: 'register', key: 'passwordConfirm', value: '' }),
      );
      return;
    }
    dispatch(register({ username, password }));
  };

  // 컴포넌트가 처음 렌더링 될 때 form 을 초기화함
  useEffect(() => {
    dispatch(initializeForm('register'));
  }, [dispatch]);

  // 회원가입 성공 / 실패 처리
  useEffect(() => {
    if (authError) {
      // 계정명이 이미 존재할 때
      console.log(authError.response.status);
```

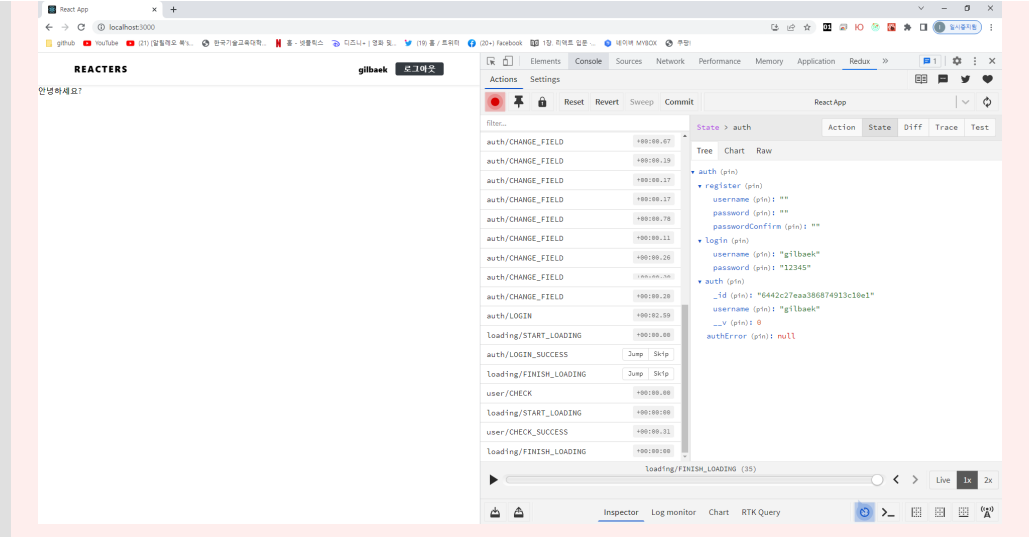
```
        if (authError.response.status === 400) {
          setError('이미 존재하는 계정명입니다.');
          return;
        }
        // 기타 이유
        setError('회원가입 실패');
        return;
      }

      if (auth) {
        console.log('회원가입 성공');
        console.log(auth);
        dispatch(check())
      }
    }, [auth, authError, dispatch]);

    // user 값이 잘 설정되었는지 확인
    useEffect(() => {
      if (user) {
        // console.log('Checl API 성공');
        // console.log(user);
        navigate('/'); // 홈 화면으로 이동
        try {
          localStorage.setItem('user', JSON.stringify(user));
        } catch (e) {
          console.log('localStorage is not working');
        }
      }
    }, [navigate, user]);

    return (
      <AuthForm
        type="register"
        form={form}
        onChange={onChange}
        onSubmit={onSubmit}
        error={error}
      />
    );
  };

  export default RegisterForm;
```

- 페이지를 새로고침했을 때도 로그인을 유지하기 위해서 랜더링시에 localstorage값을 리덕스스토어에 넣도록 구현해야 한다.
  - 이 작업ㄹ은 App컴퍼넌트에서 useEffect를 이용하거나 App를 클래스형컴퍼넌트로 변환후 componentDidMount로 처리가능
  - 하지만 이경우 랜더링한 이후에 실행되기 때문에 깜박임현상이 발생할 수 있다.
  - 따라서, src\index.js에서 사용자 정보를 로딩하도록 처리후 컴퍼넌트를 랜더링하면 깜박임현상이 발생하지 않는다.

src\index.js

- 코드를 작성할 때 sagaMiddleware.run()이 호출이후에 loadUser함수를 호출하는 것이 중요하다.
- loadUser()함수를 먼저 호출하면 CHECK액션을 디스패치 했을 때 사가에서 이를 제대로 처리하지 않는다.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import { BrowserRouter } from 'react-router-dom';
import { Provider } from 'react-redux';
import { legacy_createStore as  createStore, applyMiddleware } from 'redux';
import { composeWithDevTools } from 'redux-devtools-extension';
import rootReducer, { rootSaga } from './modules';
import createSagaMiddleware from 'redux-saga';
import { tempSetUser, check } from './modules/user';

const sagaMiddleware = createSagaMiddleware();

const store = createStore(
  rootReducer,
  composeWithDevTools(applyMiddleware(sagaMiddleware)),
);


function loadUser() {
  try {
    const user = localStorage.getItem('user');
    if (!user) return; // 로그인 상태가 아니라면 아무것도 안함
    store.dispatch(tempSetUser(JSON.parse(user)));
    store.dispatch(check());
  } catch (e) {
    console.log('localStorage is not working');
  }
}

sagaMiddleware.run(rootSaga);
loadUser()

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Provider store={store}>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </Provider>
);
```



24.3.2.3 로그인 검증실패시 정보초기화

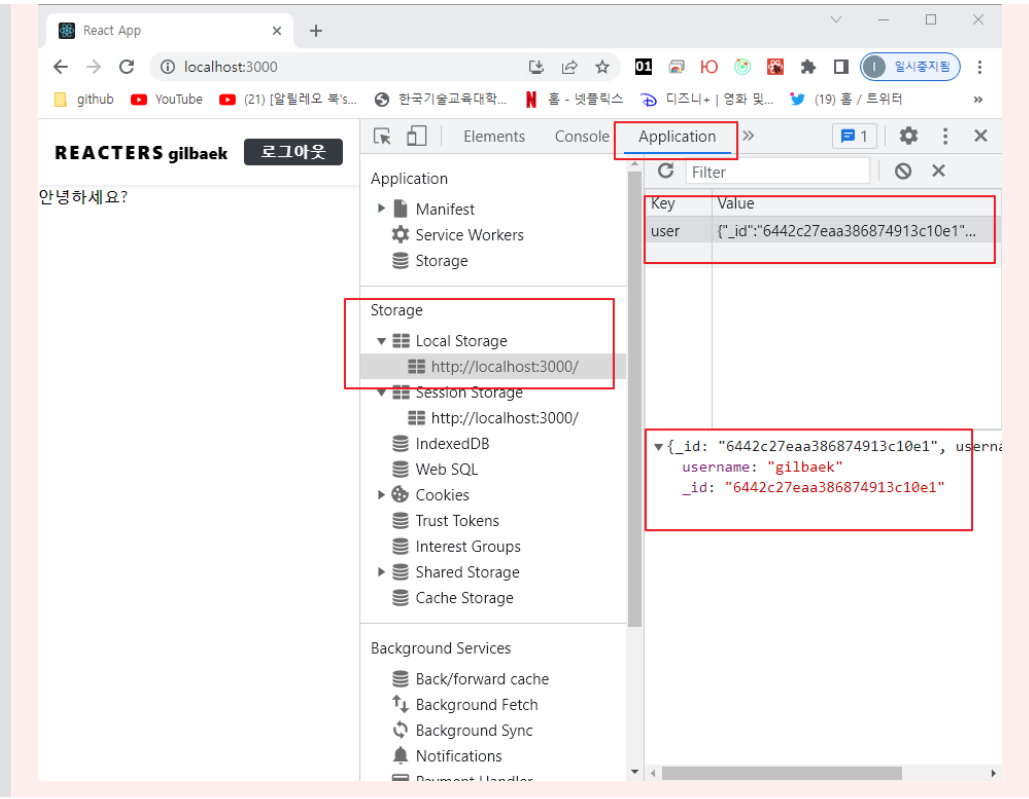
- 로그인정보가 만료되었을 때 사용자정보 초기화

modules\user.js

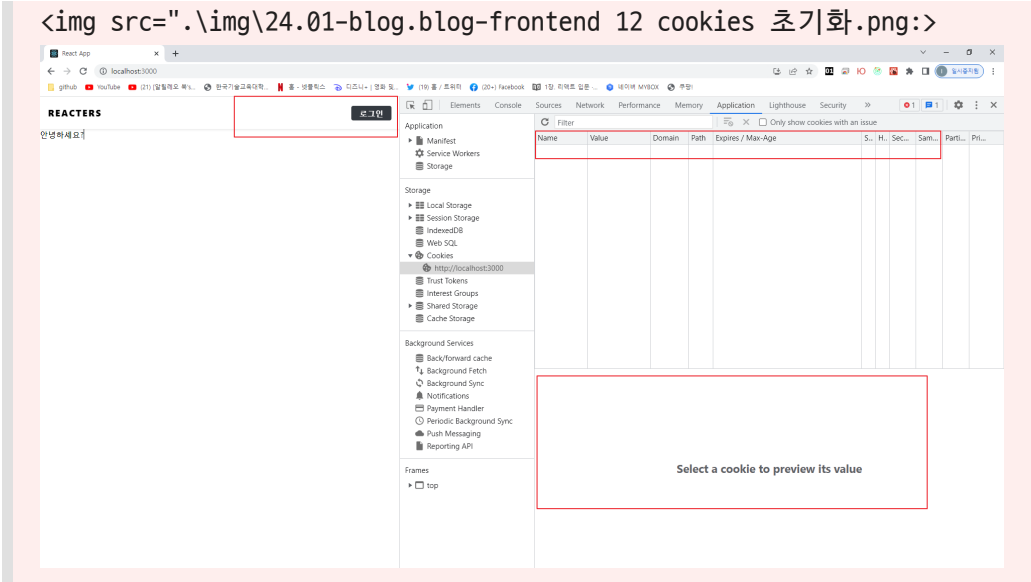
- checkFailureSaga함수를 만들고 CHECK_FAILURE액션발생시 해당 함수 호출
  - localStoage 안의 user값을 초기화
  - CHECK_FAILURE발생시 user값을 null로 설정
  - checkFailureSaga함수에서 yield를 사용하지 않기 때문에 function*사용, 제너레이터함수형태로 만들지 않아도 괜찮음

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

const TEMP_SET_USER = 'user/TEMP_SET_USER'; // 새로고침 이후 임시 로그인 처리
// 회원 정보 확인
const [CHECK, CHECK_SUCCESS, CHECK_FAILURE] = createRequestActionTypes(
  'user/CHECK',
);

export const tempSetUser = createAction(TEMP_SET_USER, user => user);
export const check = createAction(CHECK);

const checkSaga = createRequestSaga(CHECK, authAPI.check);

function checkFailureSaga() {
  try {
    localStorage.removeItem('user'); // localStorage 에서 user 제거하고
  } catch (e) {
    console.log('localStorage is not working');
  }
}

export function* userSaga() {
  yield takeLatest(CHECK, checkSaga);
  yield takeLatest(CHECK_FAILURE, checkFailureSaga);
```

```
}

const initialState = {
  user: null,
  checkError: null,
};

export default handleActions(
  {
    [TEMP_SET_USER]: (state, { payload: user }) => ({
      ...state,
      user,
    }),
    [CHECK_SUCCESS]: (state, { payload: user }) => ({
      ...state,
      user,
      checkError: null,
    }),
    [CHECK_FAILURE]: (state, { payload: error }) => ({
      ...state,
      user: null,
      checkError: error,
    }),
  },
  initialState,
);
```



- 쿠키를 초기화 할 때는 개발자도구 Applicaion탭에서 `Cookies > http://localhost:3000/` 선택 후 클리어버튼 클릭
- 쿠키초기화후 새로고침 : Applicaion탭과 console에 에러메시지 확인



24.3.3 로그아웃기능 구현

lib\api\auth.js

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

// 로그인
export const login = ({ username, password }) =>
  client.post('/api/auth/login', { username, password });

// 회원가입
export const register = ({ username, password }) =>
  client.post('/api/auth/register', { username, password });

// 로그인 상태 확인
export const check = () => client.post('/api/auth/check');

// 로그아웃
export const logout = () => client.post('/api/auth/logout');
```

modules\user.js

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

const TEMP_SET_USER = 'user/TEMP_SET_USER'; // 새로고침 이후 임시 로그인 처리
// 회원 정보 확인
const [CHECK, CHECK_SUCCESS, CHECK_FAILURE] = createRequestActionTypes(
  'user/CHECK',
);

const LOGOUT = 'user/LOGOUT'

export const tempSetUser = createAction(TEMP_SET_USER, user => user);
export const check = createAction(CHECK);
export const logout = createAction(LOGOUT);
```

```
const checkSaga = createRequestSaga(CHECK, authAPI.check);

function checkFailureSaga() {
  try {
    localStorage.removeItem('user'); // localStorage 에서 user 제거하고
  } catch (e) {
    console.log('localStorage is not working');
  }
}

function* logoutSaga() {
  try {
    yield call(authAPI.logout); // logout API 호출
    localStorage.removeItem('user'); // localStorage 에서 user 제거
  } catch (e) {
    console.log(e);
  }
}

export function* userSaga() {
  yield takeLatest(CHECK, checkSaga);
  yield takeLatest(CHECK_FAILURE, checkFailureSaga);
  yield takeLatest(LOGOUT, logoutSaga);
}

const initialState = {
  user: null,
  checkError: null,
};

export default handleActions(
  {
    [TEMP_SET_USER]: (state, { payload: user }) => ({
      ...state,
      user,
    }),
    [CHECK_SUCCESS]: (state, { payload: user }) => ({
      ...state,
      user,
      checkError: null,
    }),
    [CHECK_FAILURE]: (state, { payload: error }) => ({
      ...state,
      user: null,
      checkError: error,
    }),
    [LOGOUT]: state => ({
      ...state,
      user: null,
    }),
  },
  initialState,
);
```

containers\common\HeaderContainer.js

```
import { useSelector, useDispatch } from 'react-redux';
import Header from '../../components/common/Header';
import { logout } from '../../modules/user';

const HeaderContainer = () => {
  const { user } = useSelector(({ user }) => ({ user: user.user }));
  const dispatch = useDispatch();
  const onLogout = () => {
    dispatch(logout());
  };
  return <Header user={user} onLogout={onLogout} />;
};

export default HeaderContainer;
```

containers\common\Header.js

```
import styled from 'styled-components';
import Responsive from './Responsive';
import Button from './Button';
import { Link } from 'react-router-dom';

const HeaderBlock = styled.div`
  position: fixed;
  width: 100%;
  background: white;
  box-shadow: 0px 2px 4px rgba(0, 0, 0, 0.08);
`;

/**
 * Responsive 컴포넌트의 속성에 스타일을 추가해서 새로운 컴포넌트 생성
 */
const Wrapper = styled(Responsive)`
  height: 4rem;
  display: flex;
  align-items: center;
  justify-content: space-between; /* 자식 엘리먼트 사이에 여백을 최대로 설정 */
  .logo {
    font-size: 1.125rem;
    font-weight: 800;
    letter-spacing: 2px;
  }
  .right {
    display: flex;
    align-items: center;
  }
`;

/**
 * 헤더가 fixed로 되어 있기 때문에 페이지의 컨텐츠가 4rem 아래 나타나도록 해주
는 컴포넌트
 */
const Spacer = styled.div`
  height: 4rem;
`;

const UserInfo = styled.div`
```

```
  font-weight: 800;
  margin-right: 1rem;
`;

const Header = ({ user, onLogout }) => {
  return (
    <>
      <HeaderBlock>
        <Wrapper>
          <Link to="/" className="logo">
            REACTERS
          </Link>
          {user ? (
            <div className="right">
              <UserInfo>{user.username}</UserInfo>
              <Button onClick={onLogout}>로그아웃</Button>
            </div>
          ) : (
            <div className="right">
              <Button to="/login">로그인</Button>
            </div>
          )}
        </Wrapper>
      </HeaderBlock>
      <Spacer />
    </>
  );
};

export default Header;
```

24.4 정리

- 회원인증을 위한 기능이 모두 구현
- API연동시에 redux와 redux-saga를 사용

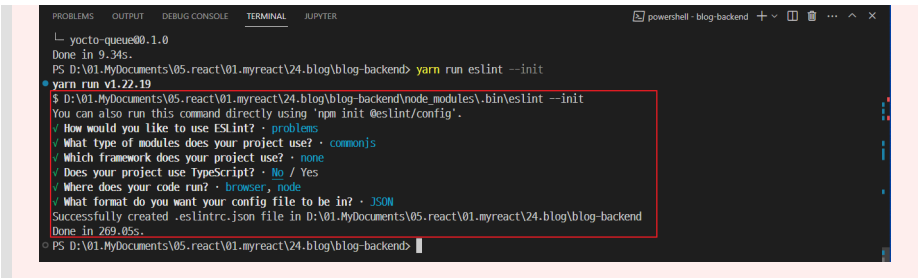
Backend 작업환경준비

23.jwt를 backend로 사용할 것

1. mkdir 24.blog\blog-backend
2. cd 24.blog\blog-backend
3. yarn init -y
4. yarn add koa
5. yarn add --dev eslint
6. yarn run eslint --init

    - To check syntax and find problems
    - CommonJS (require/exports)
    - None of these
    - oes your project use TypeScript? » No

- Where does your code run? · node
- What format do you want your config file to be in? · JSON



- eslintrc.json 확인

```
{
    "env": {
        "commonjs": true,
        "es2021": true,
        "node": true
    },
    "extends": "eslint:recommended",
    "parserOptions": {
        "ecmaVersion": "latest"
    },
    "rules": {
    }
}
```

7. .prettierrc 파일생성

```
{
  "singleQuote": true,
  "semi": true,
  "useTabs": false,
  "tabWidth": 2,
  "trailingComma": "all",
  "printWidth": 80
}
```

1. yarn add eslint-config-prettier

- eslintrc.json 수정

```
{
    "env": {
        "commonjs": true,
        "es2021": true,
        "node": true
    },
    "extends": ["eslint:recommended", "prettier"],
    "globals": {
      "Atomics": "readonly",
      "SharedArrayBuffer": "readonly"
    },
    "parserOptions": {
        "ecmaVersion": "latest"
    },
    "rules": {
      "no-unused-vars": "warn",
```

```
      "no-console": "off"
    }
  }
```

- 참고 : 프로젝트에 코드 스타일 셋팅하기
  - `https://velog.io/@parksil0/프로젝트에-코드-스타일-셋팅하기eslint-prettier-stylelint`

2. yarn add --dev nodemon

   - yarn start : 재시작이 필요 없을 때
   - yarn start:dev 쒸 재시작이 필요할 때

3. yarn add koa-router
4. yarn add koa-router-bodyparser
5. yarn add mongoose dotenv
6. yarn addm esm
7. yarn add joi
8. yarn add bcrypt
9. openssl 설치
   - openssl rand -hex 64