MERN 스택기반 CRUD 로그인 개발 보고서

비트캠프 딥러닝 개발자과정(NCS) 조현국

코드상세 Github

주소: https://github.com/ChoHyunGook/CRUD_PROJECT

1.MERN 스택이란?

M= MongoDB, E=Express.js, R= React.js, N= Node.js 를 사용하여 웹사이트를 개발하는 것을 말한다.

Server 에는 Node.js, Express.js, MongoDB 가 있으며 Client(Browser)인 React.js 가 있고 서로 Requests 와 Responses 를 주고받는다. 이 안에서의 Data 는 JSON Format 이라고 한다.

Container 기반 가상화 플랫폼인 Docker 위에 쌓아가는 구조로 Node.js, React.js, Express.js, MongoDB 의 순으로 쌓아간다.

1.1 Stack(스택)이란? :스택(stack)은 제한적으로 접근할 수 있는 나열 구조이다. 그 접근 방법은 언제나 목록의 끝에서만 일어난다. 끝먼저내기 목록(Pushdown list)이라고도 한다. 스택은 한 쪽 끝에서만 자료를 넣거나 뺄 수 있는 선형 구조(LIFO - Last In First Out)으로 되어 있다. 자료를 넣는 것을 '밀어넣는다' 하여 푸쉬(push)라고 하고 반대로 넣어둔 자료를 꺼내는 것을 팝(pop)이라고 하는데, 이때 꺼내지는 자료는 가장 최근에 푸쉬한 자료부터 나오게 된다. 이처럼 나중에 넣은 값이 먼저 나오는 것을 LIFO 구조라고 한다. -위키백과-

1.2 Docker : Docker 는 컨테이너 기반의 오픈소스 가상화 플랫폼이다. 가상 머신처럼 독립된 실행환경을 만들어주는 것으로, 운영체제를 설치한 것과 유사한 효과를 낼 수 있지만, 실제 운영체제를 설치하지 않기 때문에 설치 용량이 적고 실행 속도 또한 빠르다.

1.3 MongoDB : MongoDB 는 C++로 작성된 문서지향적 데이터베이스이며 NoSQL DB 이다 Not Only SQL 의 약자로 기존 DBMS 가 가지고 있는 특성도 가지고 있으며, JavaScript 의 JSON 과 같은 동적 스키마형 Documents 들을 선호하는 특징이 있다. 이러한 것들을 MongoDB 에선 BSON 이라고 한다.

1.4 Express.js : Express.js 는 Node.js 를 위한 Web FrameWork 이며, 빠르고 개방적이며 간결한 표준 서버 FrameWork 이며, JavaScript 언어로 작성된다.

1.5 React.js : React.js 는 사용자의 Interface 를 만들기 위한 자바스크립트의 Library 이다. 스스로의 상태를 관리하는 Encapsulated 된 Component 를 기반으로 되어있으며, JavaScript 로 작성된다.

1.6 Node.js : Node.js 는 JavaScript Engine 에 기반한 Server Side Platform 이며, 내장 HTTP Server Library 를 포함하고 있어서 별도의 소프트웨어가 없이 동작할 수 있다. MERN 스택의 제일 밑 부분에 Server 에서 JavaScript 가 작동되게 하는 Runtime Environment Platform 이다. Node.js 의 특징 중 하나는 카카오톡에서 쓰는 것을 예로 들어 Non-Blocking 으로 Requests 한 후 Responses 를 기다리지 않고 다른 API 를 Requests 할 수 있으며, 이전에 Requests 했던 API 의 Responses 가 오면 Event Loop 가 확인하여 처리해준다.

2. Docker 설정

Window 기반으로 Docker 를 Install 하는 방법이다.

해당 사이트 https://docs.docker.com/desktop/windows/install/

에서 Docker Desktop for Windows 를 클릭하여 exe 파일을 다운 받아준다. 해당 파일을 실행하여 설치를 진행해주는데 Configuration 에서 **Install required Windows components for WSL2** 와 **Add shortcut to desktop** 둘다 체크를 하고 설치를 진행한다. 설치가 완료된 후 정상적으로 Install 이 되었는지 확인하기 위해 Windows PowerShell 또는 cmd 에서 Docker 버전을 확인하는 명령어 **docker –version** 을 적은후 버전을 확인한다.

---

3. MERN 스택의 Node.js 설정

Window 기반으로 Node.js 를 Install 하는 방법이다.

해당사이트 https://nodejs.org/ko/ 에서 각 버전에 따른 exe 파일을 다운로드하여 설치를 진행한다. 그리고 Node.js 의 경우 환경변수를 설정해주어야하는데 **윈도우+R** 키 또는 시작프로그램에서 **실행**창을 열어 **sysdm.cpl ,3** 을 입력해 준다. 그럼 시스템 속성에서 환경변수를 클릭한 후 시스템변수에 새로만들기를 클릭하여 변수이름으로는 **NODE_HOME** 을 입력하고 값에는 설치되어있는 경로 **C:\Program Files\nodejs** 를 입력해 준다. 그리고 환경변수 및 시스템속성에 확인,적용 후 정상적으로 node.js 가 설치되어 있는지 확인하기 위해 Windows PowerShell 또는 cmd 에서 버전을 확인하는 명령어 **node –version** 을 적은후 다운로드했던 버전이 맞는지 확인한다.

---

4. React.js 설정

React 를 사용하기 위해 VsCode 를 해당사이트 https://code.visualstudio.com/ 에서 exe 파일을 받아 설치하여준다. 설치 진행 중 추가작업 선택에서 체크는 모두 하고 설치를 진행한다.그리고 진행에 필요한 확장프로그램들을 설치해 준다.

---

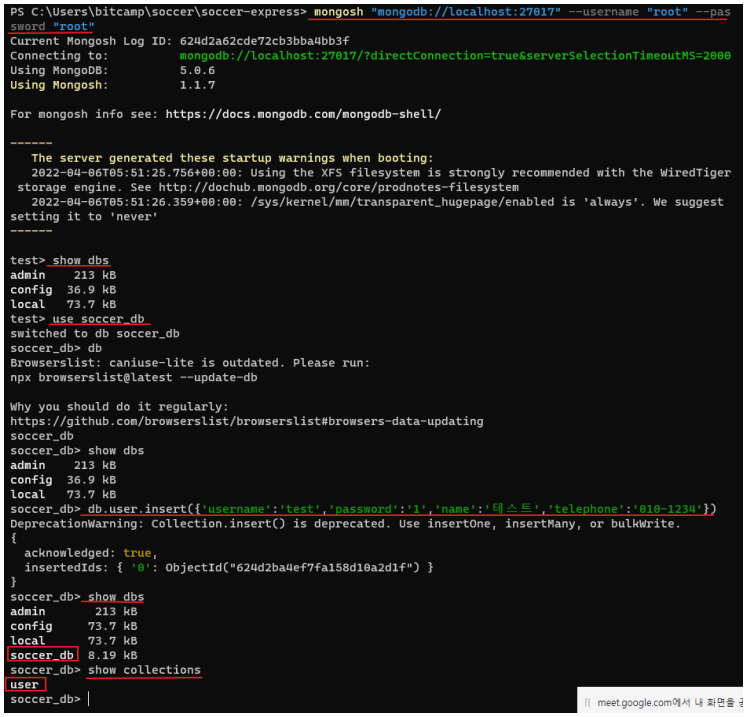
5. Express.js 설정

Express 를 설치하기 위하여 해당 Express 를 설치할 폴더에서 Windows PowerShell 또는 cmd 를 실행하여준다. Express 를 설치할 폴더에서 이름을 지정해주는 Linux 명령어 mkdir '폴더명'을 실행한후 만들어진 폴더 안에서 명령어 **npx express-generator** 를 작성한 후 실행하면 express.js 기본설정이 완료된다.

---

6. MongoDB 설정

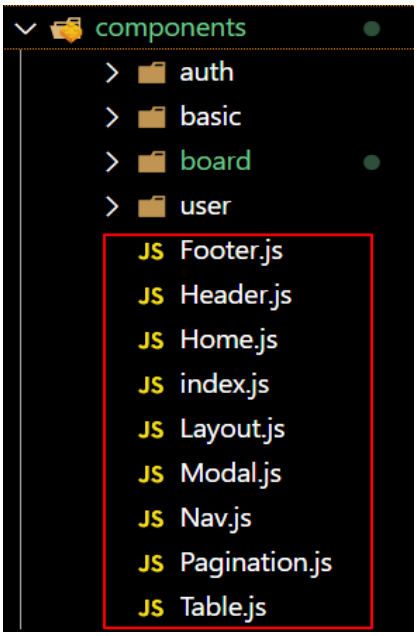
MongoDB 를 설치하기위해 해당사이트 https://www.mongodb.com/try/download/compass 에서 원하는 버전에 맞는 파일을 받아준다. Zip 파일로 받았을때 압축을 푼 후 압축을 푼 폴더 내 **MongoDBCompass.exe** 파일을 실행하여 준다. 해당 프로그램에 설치 후 . Docker 에 MongoDB 를 올려주기 위하여 터미널에 다음 명령어들을 입력한다. Docker pull mongo, docker run --name mongodb_docker -e MONGO_INITDB_ROOT_USERNAME=root -e MONGO_INITDB_ROOT_PASSWORD=root -d -p 27017:27017 mongo 를 차례대로 입력해주면 Docker 에 mongodb_docker 라는 container 가 추가되어 있는 것을 확인할 수 있다.



---

7. 개발 시작

MERN 스택의 Node.js 는 앞서 설치를 하였고 그다음 순서인 React 를 시작한다. 프로젝트를 생성 한 후 화면이 직접적으로 보이는 components 폴더와 Router, Middleware,Saga 가 있는 리덕스인 modules 폴더를 생성해주고 기존에 있던 pages 는 Next 로 프레임워크로 사용한다. 첫 화면을 구성하기 위해 기본 아래 구성들을 만들어준다.

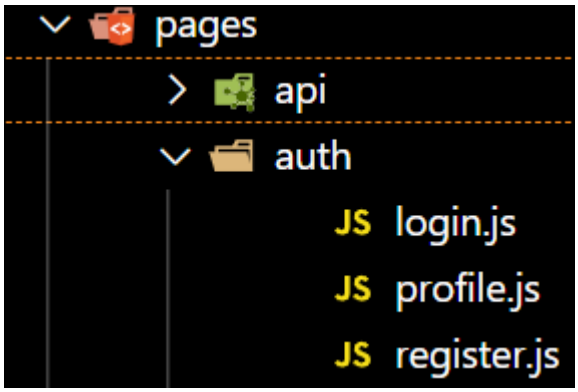


위 사진과 같이 auth 폴더를 생성한후 로그인에 필요한 login.js 와 회원가입에 필요한 Register.js 를 생성해준다.

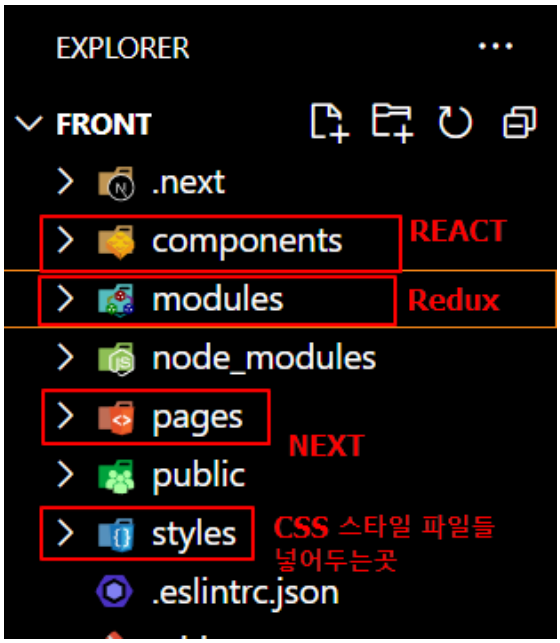
그리고 user 폴더를 생성해 프로필을 볼수있는 Profile.js 를 생성해준다. 화면을 구성하는 코딩을 한 후 아래와 같이 index.js 에 경로를 지정한다

```
export * from './Home'
export * from './Header'
export * from './Nav'
export * from './Table'
export * from './Pagination'
export * from './Footer'
export * from './Modal'
export * from './Layout'
export * from './auth/Login'
export * from './auth/Register'
export * from './user/Profile'
export * from './board/Write'
export * from './board/Update'
export * from './board/Remove'
export * from './board/List'
```

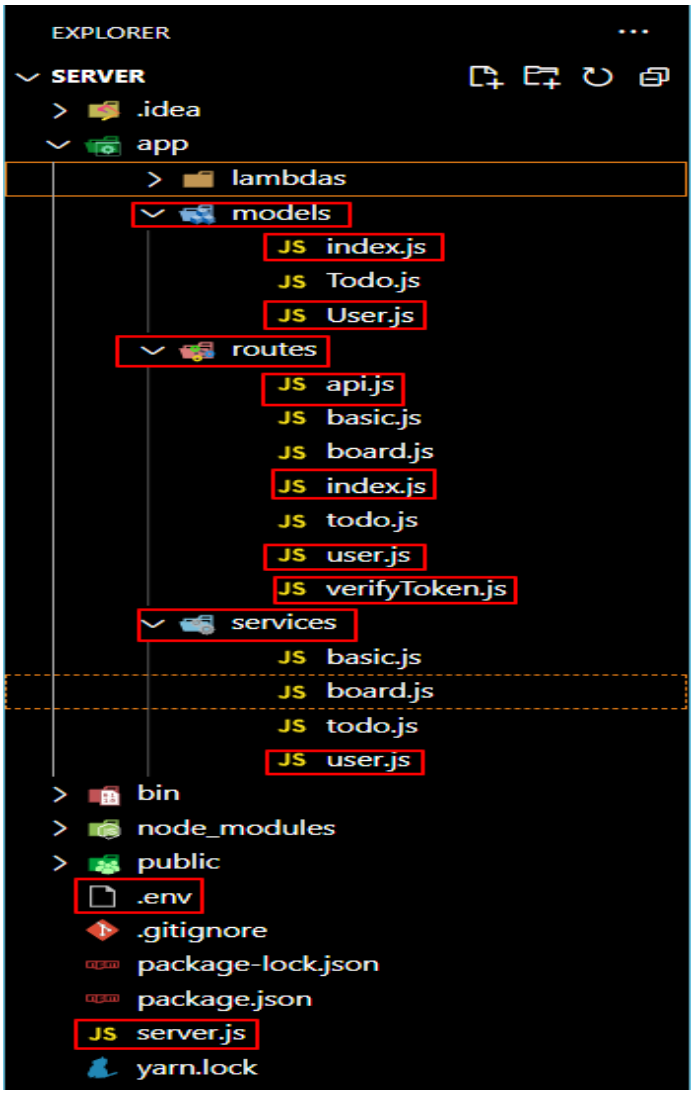
Layout.js 그리고 pages 폴더에 index.js 안에 화면 구성을 완료시킨다. 그리고 _app.js 에 코딩을 한 후 components 폴더처럼 auth 폴더에도 똑같이 login.js, profile.js, register.js 를 동일하게 만들어준다.



그리고 modules 폴더에도 auth,user 폴더를 생성하여 똑같이 3 개를 동일하게 생성해 준다.



앞서 말했듯 components 엔 화면을 구성하는 코딩이 필요하며 modules 에는 각 컴포넌트의 데이터 전달의 상태를 관리하기 위한 상태관리도구인 리덕스를 바닐라 스크립트로 자신이 직접 컨르롤하기 쉽게 구현하며 pages 는 React 의 프레임워크인 Next 를 코딩 해 준다.



그리고 MERN 의 다음인 Server 인 Express 로 넘어간다. 해당 Express 에 기본으로 되어있는 app.js 를 server.js 로 만들어주고 경로를 지정해준다. Express 에 필요한 models,routes,services 폴더들을 생성하고 위와 같이 만들어준다.

.env 로 들어가 해당 코드들을 입력하여 MongoDB 와 REACT 를 연결하기 위해 아래와 같이 입력해 준다.

여기서 로그인에서 필요한 Token 은 PostMan 을 사용하며 https://www.postman.com/에서 다운을 받아 사용한다.

---

8. 실행순서

MERN 의 실행순서는 Stack 구조로써 반대로 Docker 를 실행하며 몽고 DB 를 켜고 EXPRESS 를 실행시키고 REACT 를 실행하는 순서이다.

---

9. 코드설명

코드의 설명을 조금 더 하면 React 부분인 component 폴더의 경우 onSubmit 과 onChange 와 같은 이벤트가 발생하는 기능을 사용하며 값을 넘겨주는 무상태(props)로 만들어 주며, 아래의 그림과 같이 callback 함수를 사용하여준다.

```
useEffect(() => {
  console.log(` 모듈에 저장된 로그인값: '+JSON.stringify(loginUser))
  if (loginUser === null) {
    setUserUrls({
      subTitles: [
        '회원가입', '로그인'
      ],
      urls: ["/auth/register", "/auth/login"]
    })
```

다음으로 Next 부분의 pages 폴더의 경우 무상태를 State 로 만들어주는 코딩을 해야하는데 이곳에서도 동일하게 onSubmit 과 onChange 의 이벤트 발생을 사용하며 아래와 같이 lambda 를 사용하여 준다.

```
const onChange = e => {
    e.preventDefault()
    const {name, value} = e.target;
    setUser({
        ...user,
        [name]: value
    })
}
```

그다음 modules 에서는 서버인 express 로 넘기는 역할을 하게 되며 비동기 객체인 axios 를 아래와 같이 post 를 사용하여 값을 이동시킨다.

```
const SERVER = 'http://127.0.0.1:5000'
```

```
const loginAPI = payload => axios.post(
    `${SERVER}/user/login`,
    payload,
    {headers}
```

Modules 의 index.js 에는 rootSaga 와 rootReducer 부분을 잘 확인해야하며, 아래 사진과같이 해당부분들을 잘 확인해 주어야한다.

```
const rootReducer = combineReducers({
    index: (state = {}, action) => { // 데이터가 콘솔에만 찍
        switch (action.type) {
            case HYDRATE:
                console.log("HYDRATE", action);
                return { ...state, ...action.payload };
            default:
                return state;
        }
    },
    login,
    register,
});
export function* rootSaga() {
  yield all([counterSaga(), registerSaga(), loginSaga()]);
}

export default rootReducer;
```

그리고 express 로 넘어가 server.js 의 엔트리 포인트를 잡아주는 use 를 사용하여 주고 mongoose 라는 몽고 db 의 연결하는 라이브러리를 아래와 같이 코딩하여 준다. Mongoose 의 설치는 npm i mongoose 을 터미널에 실행하여준다.

```
async function startServer() {
    const app = express();
    const {mongoUri, port, jwtSecret } = applyDotenv
    app.use(express.static('public'));
    app.use(express.urlencoded({extended: true}));
    app.use(express.json());
    const _passport = applyPassport(passport, jwtSec
    app.use(_passport.initialize());
    app.use("/", index);
    app.use("/api", api);
    app.use("/basic", basic);
    app.use("/board", board);
    app.use("/todo", _passport.authenticate('jwt',
    app.use("/user", user);
    app.use(morgan('dev'))
    db
        .mongoose
        .connect(mongoUri, {
            useNewUrlParser: true,
            useUnifiedTopology: true
        })
```

Routes 폴더의 경우 req, res,next 를 받는 middleware 함수를 사용한다.

```
app.use(function (_req, res, next) {
    res.header(
        "Access-Control-Allow-Headers",
        "x-access-token, Origin, Content-Type, Accept"
    );
    next();
});
```

Service 는 확인시켜주는 코드를 console 로 확인하고 핸들링하는 코드를 넣어준다.

```
join(req, res) {
    new User(req.body).save(function (err) {
        if (err) {
            res
                .status(500)
                .send({message: err});
            console.log('회원가입 실패')
            return;
        } else {
            res
                .status(200)
                .json({ok: 'ok'})

        }
    })
```

Models 는 DB 에 저장될 스키마와 패스워드와 같은 암호화가 필요한 기능에 대한 코드인 bcrypt 를 사용하여준다.

```
const userSchema = mongoose.Schema({
        userid: {type: String, maxlength: 10, unique: 1},
        password: String,
        email: {type: String, trim: true, unique: 1},
        name: String,
        phone: {type: String, maxlength: 15},
        image: String,
        birth: String,
        address: String,
        token: String
    }, {timestamps: true})
```

```
userSchema.pre("save", function (next) {
    let user = this
    const saltRounds = 10
    //model 안의 paswsword가 변환될때만 암호화
    //if (user.isModified("password")) {
      bcrypt.genSalt(saltRounds, function (err, salt) {
        if (err) return next(err);
        bcrypt.hash(user.password, salt, function (err, hash) {
          if (err) return next(err);
          user.password = hash;
          next();
        });
      });
```