MERN 스택 개발 보고서

비트캠프 KDT 5기 김지혜

2022년 4월17일

- MERN 스택이란?
- Node.js
- React.js
- Next.js
- Express
- MongoDB
- Redux
- 전체적인 이론

- **MERN** 스택이란**?**

MERN은 동적 웹 사이트와 웹 애플리케이션을 개발하기 위한 자유-오픈 소스 자바스크립트 소프트웨어 스택이다. MERN 스택은 몽고DB, Express.js, React.js, Node.js이다. MERN 스택의 모든 구성 요소들이 자바스크립트로 작성된 프로그램들을 지원하기 때문에 MERN 애플리케이션들은 서버 사이드와 클라이언트 사이드 실행 환경을 위해 오직 하나의 언어로 작성이 가능하다.

- **Node.js**

Node.js

Node.js는 확장성 있는 네트워크 애플리케이션 개발에 사용되는 소프트웨어 플랫폼이다. JavaScript로 브라우저가 아니라 서버를 구축하고, 서버에서 JavaScript가 작동되도록 해주는 런타임 환경이다.
내장 HTTP 서버 라이브러리를 포함하고 있어 웹 서버에서 아파치 등의 별도의 소프트웨어 없이 동작하는 것이 가능하며 이를 통해 웹 서버의 동작에 있어 더 많은 통제를 가능케한다.

Node.js 개발 환경 설정

Node.js 사이트에서 해당 사양에 맞는 프로그램을 다운로드 받는다. 다운로드 받은 프로그램을 실행시키고, 시스템 속성창을 열어서 환경변수의 시스템 변수에 변수이름을 NODE_HOME으로, 변수 값을 Node.js 설치경로로 잡아준다. PATH설정값은 %NODE_HOME%으로 한다.
cmd창에서 node–version를 입력하여 해당 버전이 출력이 되면 설치 성공이다.

VSCode 설치

VSCode는 마이크로소프트가 마이크로소프트 윈도우, macOS, 리눅스용으로 개발한 소스 코드 편집기이다. 디버깅 지원과 Git 제어, 구문 강조 기능등이 포함되어 있으며, 사용자가 편집기의 테마와 단축키, 설정 등을 수정할 수 있다.
Visual Strudio Code사이트에서 PC 운영체제에 맞게 프로그램을 다운로드받는다. 다운로드 받은 프로그램을 실행한다.

- **React.js**

React.js

React.js는 사용자 인터페이스를 만들기 위한 JavaScript 라이브러리이다. React.js는 컴포넌트기반으로 다양한 형식의 데이터를 앱 안에서 손쉽게 전달할 수 있고, DOM과는 별개로 상태를 관리할 수 있다.
컴포넌트는 속성으로 props, state와 메소드로 render()를 가지고 있는 함수이다. Props는 외부에서 입력 데이터를 다루는 것이며, state는 내부적인 상태 데이터를 가지는 것으로 속성이 2가지로 나뉜다.

React Hook

Hook은 React v16.8에 새로 도입된 기능이다.
함수형태의 컴포넌트에서 사용되는 몇가지 기술을 Hook이라고 부른다. 함수형 컴포넌트에서도 상태 관리를 할 수 있는 useState, 그리고 렌더링 직후 작업을 설정하는 useEffect 등의 기능 등을 제공한다.
Hook은 React 개념에 보다 직관적인 API를 제공한다. 또한 상태에 관련된 로직을 재사용하기 쉽게 만들고, 함수형 컴포넌트가 클래스형 컴포넌트의 기능을 사용할 수 있도록 해주는 기능이다.

- Next.js

Next.js

현재 React.js를 대신해서 Next.js를 사용하는데 React.js는 라이브러리이지만 Next.js는 운영을 위한 리액트 프레임워크이다. Next.js는 하이브리드 정적 및 서버 렌더링, TypeScript 지원, 스마트 번들링, 경로 미리 가져오기 등 프로덕션에 필요한 모든 기능을 한다.
React.js는 UI 구축에 탁월하지만 해당 UI를 완전히 작동하는 확장 가능한 애플리케이션으로 독립적인 구축하려면 약간의 작업이 필요하다.
때문에 Next.js는 애플리케이션을 구성하는 프레임워크와 개발 프로세스와 최종 애플리케이션을 더 빠르게 만드는데 도움이 되는 최적화를 제공한다.

Next.js 개발환경 설정
Node.js 12.22.0 이상의 시스템 요구 사항을 확인을 하고, 만들고 싶은 경로의 터미널을 열어 yarn create next-app 명령어를 코딩하면 자동으로 Next.js가 생성된다.
Next.js의 디렉토리 구조는 public, styles, pages가 있는데 그중 pages는 라우팅 기능을 담당한다.
해당 Next.js를 VSCode로 열어 터미널에 yarn dev를 코딩하여 Next.js를 실행시킨다.

*Next.js*를 사용하는 이유
Next.js는 React를 기반으로 한 프레임워크이며, SSR를 구현하고 SEO에 유리하기 때문에 사용한다.
Next.js는 Server에서 받은 사용자의 접속 요청 초기에 SSR방식으로 렌더링 될 HTML을 보내고, 브라우저에서 JavaScript를 다운로드하고 React를 실행하기 때문에 SEO가 가능하다. 또한 다른 페이지로 이동할 경우 CSR방식으로 Server가 아닌 브라우저에서 처리함으로써 SPA장점도 유지할 수 있다.
참조 - https://ivorycode.tistory.com/entry/Nextjs%EB%A5%BC-%EC%82%AC%EC%9A%A9%ED%95%98%EB%8A%94-%EC%9D%B4%EC%9C%A0

라이브러리 *vs* 프레임워크
라이브러리는 기능위주인 함수집합으로 구성이 되어 있으며 환경을 설정하면 자동으로 라이브러리는 생성된다. 즉 라이브러리는 함수가 담겨있는 컨테이너로 함수를 호출하여 사용하는데 신경망이라고 말할 수 있다.
프레임워크는 틀이 굳어진 것으로 객체의 집합인데 프로토콜을 만들기 위해서 개발자가 따로 설치를 해야한다. 비유로 들면 뼈, 장기 등 마네킹으로 생각할 수 있다.
프레임워크는 웹 애플리케이션을 만들기 위한 각종 라이브러리와 미들웨어 등이 내장돼 있어 개발하기 편하고, 수많은 개발자들에게 개발 규칙을 강제하여 코드 및 구조의 통일성을 향상시킬 수 있다는 장점이 있다.
-개발을 할 때 프레임워크를 먼저 구축을 하여 몸을 만들고 라이브러리를 주입하여 움직이게 한다.

- **Express**

Express.js
Express는 Node.js를 위한 빠르고 개방적인 간결한 웹 프레임워크이다. Node.js는 오픈소스, 크로스 플랫폼이며 jsvascript로 브라우저가 아니라 서버를 구축하고 서버에서 javascript가 작동되도록 해주는 런타임 환경이다.
Express는 이런 Node.js의 원칙과 방법을 이용하여 웹 어플리케이션을 만들기 위한 프레임워크이다.

Express.js 개발환경 설정
Express를 생성하고 싶은 경로에서 터미널을 열고 mkdir 프로젝트명을 코딩하고, 프레임워크에 필요한 package.json와 기본 구조까지 잡을 수 있게 npx express-generator를 코딩한다. 다음 종속성을 설치하기 위해 VSCode로 열어 터미널에 npm instal 코딩하고 npm start하여 서버를 실행시킨다.

참조 - https://ninjaggobugi.tistory.com/9

- **MongoDB**

MoongoDB
MongoDB는 애플리케이션 개발 및 확장이 용이한 크로스 플랫폼 도큐먼트 지향 데이터베이스 시스템이다. NoSQL 데이터베이스로 분류되는 MongoDB는 JSON과 같은 동적 스키마형 도큐먼트들(MongoDB는 이러한 포맷을 BSON이라 한다.)을 선호함에 따라 전통적인 테이블 기반 관계형 데이터베이스 구조의 사용을 삼간다.

- **Redux**

Redux
Redux는 JS앱을 위한 예측 가능한 상태 컨테이너이다.
컴포넌트들의 상태 관련 로직들을 다른 파일들로 분리시켜서 효율적으로 상태관리할 수 있다.
Redux는 일관되게 동작하고 다양한 환경(클라이언트, 서버 및 기본)에서 실행되며 테스트하기 쉬운 애플리케이션을 작성하는데 도움이 된다.
또한 애플리케이션의 상태와 논리를 중앙 집중화하면 실행 취소/ 다시실행, 상태 지속성 등과 같은 강력한 기능을 사용할 수 있다. 애플리케이션 상태가 언제, 어디서, 왜, 어떻게 변경되었는지 쉽게 추적이 가능하다.

Redux Toolkit
Redux Toolkit은 효율적인 Redux 개발을 위한 공식적이고 독단적인 배터리 포함 도구이다.
기존 redux는 복잡한 스토어 설정, radux를 유용하게 사용하기 위해서 추가되어야 하는 많은 패키지들, redux 사용을 위해 요구되는 다량의 상용구 코드들 등 다양한 문제점이 있다. 이러한 문제점을 개선하기 위해서 Redux Toolkit이 나오게 되었다.

- 전체적인 이론

리액트 컴포넌트
리액트에서 객체는 컴포넌트로 속성인 props와 state로, 기능인 render()로 구성되어 있다.
속성으로 props와 state가 있는데 둘의 차이점으로 props는 컴포넌트(함수 매개변수처럼)에 전달되는 반면 state는 컴포넌트(함수 내에 선언된 변수처럼) 안에서 관리된다.
기능인 render()은 입력받은 데이터를 화면에 표시할 내용으로 반환하는 역할을 한다.

React와 Next 연결

React와 Next는 속성인 props와 함수로 연결이 되어있다. Next에서 props로 데이터를 전달하고 React는 함수로 데이터를 전달한다.

SSR과 CSR

SPA

SPA는 웹 애플리케이션에 필요한 모든 정적 리소스를 최초 접근시 단 한번만 다운로드한다. 이후 새로운 페이지 요청 시, 전체 페이지를 다시 렌더링하지 않고 변경되는 부분만 갱신하므로 새로고침이 발생하지 않아 네이티브 앱과 유사한 사용자 경험을 제공할 수 있다.

CSR(Client Side Rendering)

CSR은 클라이언트에서 모두 처리하는 것으로 서버에서 전체 페이지를 한번 렌더링 하여 보여주고 사용자가 요청할 때마다 리소스를 서버에서 제공받아 클라이언트가 해석하고 렌더링하는 방식이다. HTML이 텅텅 비어있기 때문에 처음 접속하게 되면 빈 화면만 보이게 되고, 링크된 자바스크립트를 다운로드 받게 된다.

SSR(Server Side Rendering)

SSR은 클라이언트에서 모든 것을 처리하지 않고, 웹 사이트에 접속하면 서버에서 필요한 데이터를 모두 가져와서 HTML파일을 만들게 되고, 만들어진 HTML과 HTML파일을 동적으로 제어할 수 있는 소스코드와 함께 클라이언트에게 보낸다. 클라이언트는 잘 만들어진 HTML 문서를 사용자에게 바로 보여주게 된다. 그렇기 때문에 페이지 로딩이 빨라지게 되고, CSR과 달리 모든 컨텐츠가 HTML에 담겨있기 때문에 효율적인 SEO가 가능하다.

함수형 프로그래밍

자료처리를 수학적 함수의 계산으로 취급하고 상태와 가변데이터를 멀리하는 프로그래밍 패러다임을 의미한다. 람다대수를 근간으로 하고 있다.

순수함수(무상태)

함수형 프로그래밍의 이상은 이른바 순수함수이다. 순수함수는 결과가 오로지 입력 매개변수에 의해서만 좌우되며 연산이 아무런 부작용을 일으키지 않는, 즉 반환값 이외의 외부 영향이 없는 함수이다.
순수함수는 인수와 반환값으로 축약된다. 순수함수와 외부 시스템과의 유일한 상호작용은 정의된 API를 통해 이뤄진다. 함수 내부에서 인자의 값을 변경하거나 프로그램 상태를 변경하는 Side Effect가 없어야 한다.

*이는 객체 메소드가 객체의 상태와 상호작용하도록 설계되는 OOP와 대비되며, 외부 상태가 함수 내에서 조작되는 경우가 많은 절차적 스타일 코드와도 대비된다.

불변성(XOR 상호배타적인 상태)

함수형 프로그래밍에서의 데이터는 변하지 않는 불변성을 유지해야 한다.

일급객체와 고차함수

순수함수 이상 외에 실제함수형 프로그래밍 코딩을 좌우하는 요소는 일급함수이다. 일급함수는 변수나 데이터 구조안에 담을 수 있고, 파라미터로 전달 할 수 있다. 동적으로 프로퍼티 할당이 가능하다.
고차함수는 함수를 인자로써 전달 할 수 있어야 하고, 함수의 반환 값으로 또 다른 함수를 사용할 수 있다.

MVC 패턴

소프트웨어 공학에서 사용되는 소프트웨어 디자인 패턴이다. 사용자 인터페이스로부터 비즈니스 로직을 분리하여 애플리케이션의 시각적 요소나 그 이면에서 실행되는 비즈니스 로직을 서로 영향 없이 쉽게 고칠 수 있는 애플리케이션을 만들 수 있다.
MVC에서 모델은 애플리케이션의 정보(데이터)를 나타내며, 뷰는 텍스트, 체크박스 항목 등과 같은 사용자 인터페이스 요소를 나타내고, 컨트롤러는 데이터와 비즈니스 로직 사이의 상호동작을 관리한다.
Express에서는 MVC패턴의 구조를 사용하고 있다.

Flux 패턴

React, Next, Redux는 Flux패턴의 구조를 사용하고 있다.
Flux는 MVC과 다르게 단방향으로 데이터가 흐른다.
React view에서 사용자가 상호작용을 할 때, 그 view는 중앙의 dispatcher를 통해 action을 전파하게 된다.
어플리케이션의 데이터와 비즈니스 로직을 가지고 있는 store는 action이 전파되면 이 action에 영향이 있는 모든 view를 갱신한다. 이 방식은 특히 react의 선언형 프로그래밍 스타일 즉, view가 어떤 방식으로 갱신해야 되는지 일일이 작성하지 않고서도 데이터를 변경할 수 있는 형태에서 편리하다.

<Flux-redux>

Dispatch

Flux 애플리케이션의 중앙 허브로 모든 데이터의 흐름을 관리한다. 본질적으로 store의 콜백을 등록하는데 쓰이고 action을 store에 배분해주는 간단한 방식이다.
Action creator가 새로운 action이 있다고 dispatch에게 알려주면 어플리케이션에 있는 모든 store는 해당 action을 앞서 등록한 callback으로 전달받는다.

Store

어플리케이션의 상태와 로직을 포함하고 있다.
단순히 ORM 스타일의 객체 컬렉션을 관리하는 것을 넘어 어플리케이션 내의 개별적인 도메인에서 어플리케이션의 상태를 관리한다. Store는 자신을 dispatch에 등록하고 callback을 제공한다. 이 callback은 action을 파라미터로 받는다. Store 내부 메소드에 적절하게 연결될 수 있는 훅을 제공한다. 여기서 결과적으로 action은 dispatch를 통해 store의 상태를 갱신한다.

Action

클릭같은 이벤트가 발생했을 때 그 이벤트가 발생했음을 Action 정보를 담고 있는 객체를 만들어내 dispatch에

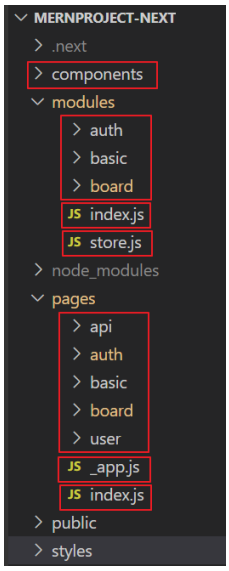
전달하는 역할을 한다. 타입(type)과 데이터(payload)를 가지고 있다.

```
const sagaMiddleware = createMiddleware()

const store = createStore(
    rootReducer,
    composeWithDevTools(applyMiddleware(sagaMiddleware))
)
sagaMiddleware.run(rootSaga)
const makeStore = () => store

export const wrapper = createWrapper(makeStore)
```

Store는 리듀서와 미들웨어인 사가를 가지고 있어 리듀서가 액션을 통해 생성되면 이를 사가가 인지하고 실행시킨다.



useState

리액트 컴포넌트에서 동적인 값을 상태(state)라고 부른다. 사용자 인터랙션을 통해 컴포넌트의 상태값이 동적으로 바뀔 경우에는 상태를 관리하는 것이 필요하다.
구조 - const [value, setValue] = useState(초기값);
Value를 바꾸고 싶다면 setValue을 이용해서 변경을 해줄 수 있다.

```
const [board, setBoard] = useState({
    date:'', title:'', content: ''
})
```

useEffect

Side effect는 component가 렌더링 된 이후에 비동기로 처리되어야 하는 부수적인 효과를 뜻한다.
useEffect를 사용하면 함수 컴포넌트에서 side effect를 수행할 수 있다. useEffect는 콜백함수를 부르게 되며, 렌더링 혹은 값, 오브젝트의 변경에 따라 어떠한 함수 혹은 여러 개의 함수들을 동작시킬 수 있다.
Event가 없이 실행되는 것으로 파라미터의 요청이 없는 상태의 리턴값을 요구하는 supplier이다.

옵저버패턴

객체의 상태 변화를 관찰하는 관찰자들, 즉 옵저버들의 목록을 객체에 등록하여 상태 변화가 있을 때마다 메서드 등을 통해 객체가 직접 목록의 각 옵저버에게 통지하도록 하는 디자인 패턴이다. useEffect에서 옵저버 패턴이 사용되는데 event 발생으로 useEffect가 실행되는 것이 아니기 때문에 옵저버가 상태변화가 있을 때 useEffect를 실행시킨다.

```
useEffect(()=>{
        axios.get('http://localhost:5000/board/list').then(res=>{
            setData(res.data.boards)
    }).catch(err=>{})
}, [])
```

이벤트

브라우저에서 사용자의 조작이나 환경의 변화로 벌어진 사건이다. 핸들러는 하나의 요소에 하나의 이벤트를 처리가능하게 한다.React에서는 false를 반환해도 기본 동작을 방지할 수 없다. 반드시 preventDefault를 명시적으로 호출해야 한다.

이벤트 핸들러
웹 페이지에서는 수많은 이벤트가 계속해서 발생한다. 특정 요소에서 발생하는 이벤트를 처리하기 위해서는 이벤트 핸들러라는 함수를 작성하여 연결해야 한다.
이벤트 핸들러가 연결된 특정 요소에서 지정된 타입의 이벤트가 발생하면, 웹 브라우저는 연결된 이벤트 핸들러가 실행된다.
onClick은 사용자가 클릭을 할 때마다 변경되는 데이터를 저장하기 위한 함수이며, onSubmit은 입력된 데이터를 설정해놓은 경로로 전송하기 위한 함수이다.

```
const onSubmit = e => {
    e.preventDefault()
    alert('게시판 등록정보:'+JSON.stringify(board))
    dispatch(writeRequest(board))
}
```

REST API

REST는 HTTP URL를 통해 자원을 명시하고, HTTP Method(POST, GET, PUT, DELETE)를 통해 해당 자원에 대한 CRUD를 적용하는 것을 의미한다.

<특징>
클라이언트- 서버 구조
클라이언트는 유저와 관련된 처리를, 서버는 REST API를 제공함으로써 각각의 역할이 확실하게 구분되고 일괄적인 인터페이스로 분리되어 작동할 수 있게 한다.
서로간의 의존성이 줄어든다.

무상태성
REST는 서버에서 어떤 작업을 하기 위해 상태정보를 기억할 필요가 없고 들어온 요청에 대해 처리만 해주면 되기 때문에 구현이 쉽고 단순해진다.

계층화

클라이언트와 서버가 분리되어 있기 때문에 중간에 프록시 서버, 암호화 계층 등 중간매체를 사용할 수 있어 자유도가 높다

자체표현구조
JSON을 이용한 메시지 포맷을 이용하여 직관적으로 이해할 수 있고 REST API 메시지 만으로 그 요청이 어떤 행위를 하는 지 알 수 있다.

Axios
브라우저와 node를 위한 promise api를 활용하는 http 비동기 통신 라이브러리다. 비슷한 기능을 하는 브라우저 빌트인 도구 fetch api가 있지만 Restful API의 기본이 되는 CRUD 요청에 fetch를 사용하는 건 꽤나 불편하다.
기능적면으로 응답이 자동으로 JSON 데이터 형식으로 변환된다.

```
const boardWriteAPI = payload => axios.post(
    `${SERVER}/board/write`, payload,{headers})
```

App
App은 express에서 사용되는 브라우저와 통신방법이다.
App은 express의 객체이다.
라우팅은 URI 및 특정한 HTTP 요청 메소드(GET, POST 등)인 특정 엔드포인트에 대한 클라이언트 요청에 애플리케이션이 응답하는 방법을 결정하는 것을 말한다.
구조 - app.METHOD(PATH, HANDLER)

```
app.post('/write', cors(corsOptions), (req, res) => {
    BoardService().write(req, res)
})
```

Connect
Connect는 HOC(higher-order component)로, 리액트 컴포넌트를 개발하는 하나의 패턴으로, 컴포넌트의 로직을 재활용할 때 유용한 패턴이다.
컴포넌트를 store에 연결할 수 있도록 해준다.

mapStateToProps
Connect 함수의 첫번째 인수이다.
Store로부터 state를 가져와서, 컴포넌트의 props로 state를 보내주는 역할을 한다.

```
const mapStateToProps = state => ({isWrited: state.boardWrite.isWrited })
const boardWriteActions = {writeRequest}
export default connect(mapStateToProps, boardWriteActions)(BoardWritePage)
```

미들웨어
미들웨어는 양 쪽을 연결하여 데이터를 주고 받을 수 있도록 중간에서 매개 역할을 하는 소프트웨어, 네트워크를 통해서 연결된 여러 개의 컴퓨터에 있는 많은 프로세스들에게 어떤 서비스를 사용할 수 있도록 연결해 주는 소프트웨어이다.
중간에서 연결시켜 관리하는 것을 미들웨어라고 한다.
next-> 미들웨어(서버, API) -> express

Next에서는 saga가 미들웨어로, express에서는 service가 미들웨어 역할을 수행한다.

Saga
Saga는 직관적인 side effect manager이다.
Manager란 CRUD를 수행할 수 있다.
Saga는 액션을 모니터링하고 있다가, 특정 액션이 발생하면 이에 따라 특정 작업을 하는 방식으로 사용한다.
여기서 특정 작업이란, 특정 자바스크립트를 실행하는 것 일수도 있고 다른 액션을 디스패치 하는 것 일수도 있고, 현재 상태를 불러오는 것 일수도 있다.
Redux-saga는 제너레이터를 이용해 액션의 순수성이 보장되도록 해준다.

```
export function* rootSaga(){
    yield all([ counterSaga(), registerSaga(), loginSaga(), boardWriteSaga()])
}
```

Generator
Generator 함수는 액션이 생성되기 전까지 존재하지 않다가 액션이 발생하면 saga가 반응하여 generator 함수를 존재시켜 실행시킨다. Yield를 사용하여 계속 돌고있는 상태를 정지시켜 generator를 우선으로 실행시킨다.
*takeLatest는 마지막 액션에 대해 saga 함수가 동작한다. 같은 종류의 액션이 여러번 요청되면 이를 파기하고 마지막 액션에 대해서만 동작한다.

```
export function* boardWriteSaga(){
    yield takeLatest(WRITE_REQUEST, boardWrite);
}

function* boardWrite(action){
    try{
        const response = yield call(boardWriteAPI, action.payload)
        console.log("게시판등록 서버 다녀옴"+JSON.stringify(response.data))
        const result = response.data
        const listBoard = JSON.stringify(result)
        localStorage.setItem("listBoard", listBoard)
        yield put({type: WRITE_SUCCESS, payload: response.data})
        yield put(window.location.href= "/board/boardList")
    }catch(error){
        yield put({type: WRITE_FAILURE, payload: error.message})
    }
```

함수종류
Consumer
파라미터만 있고 리턴은 없는 상태
Supplier
파라미터는 없고 리턴만 있는 상태
Function(기본함수형식)
파라미터와 리턴 모두 있는 상태로 순수함수라고 칭한다.

Mongoose
사용자는 mongoose를 사용하여 특정 컬렉션의 문서에 대한 스키마를 정의할 수 있다.
MongoDB에서 데이터 생성 및 관리에 많은 편의성을 제공한다.