React

React의 철학

MSA: Micro Service Architecture

- Monolithic Architecture
  - 소프트웨어의 모든 구성요소가 한 프로젝트에 통합되어 있는 형태
  - 웹 개발의 경우, 웹 프로그램을 개발하기 위해 모듈별로 개발을 하고, 개발이 완료된 웹 어플리케이션을 하나의 결과물(WAR 파일)로 패키징하여 WAS에 배포되는 형태
  - 소규모 프로젝트에는 Monolithic Architecture가 훨씬 합리적
  - 간단한 Architecture이고 유지보수가 용이
  - 단점
    - 서비스/프로젝트가 커지면 커질수록, 전체 시스템 구조의 파악에 어려움
    - 빌드 시간 및 테스트시간, 그리고 배포시간이 기하급수적으로 늘어남
    - 서비스를 부분적으로 scale-out(여러 server로 나누어 일을 처리하는 방식) 하기가 힘듬
    - 부분의 장애가 전체 서비스의 장애로 이어지는 경우가 발생
    - 서비스의 변경이 어렵고, 수정 시 장애의 영향도 파악이 힘들다.
    - 여러 컴포넌트가 하나의 서비스에 강하게 결합되어 있기 때문에 수정에 대한 영향도 파악이 힘들다.
    - 한 Framework와 언어에 종속적이다.

MSA: Micro Service Architecture

- Monolithic Architecture
  - 이러한 Monolithic Architecture의 문제점들을 보완하기 위해 MSA가 등장하게 되었다. 기존의 특정한 물리적인 서버에 서비스를 올리던 on-promise 서버 기반의 Monolithic Architecture에서 이제는 클라우드 환경을 이용하여 서버를 구성하는 Micro Service Architecture로 많은 서비스들이 전환되고 있음.

MSA: Micro Service Architecture

- MSA: Micro Service Architecture
  - 작고, 독립적으로 배포 가능한 각각의 기능을 수행하는 서비스로 구성하는 프로그램 아키텍처
  - 하나의 큰 애플리케이션을 여러 개의 작은 애플리케이션으로 쪼개어 변경과 조합이 가능하도록 만든 아키텍처
  - ***small services, each running in its own process(스스로 돌아 갈 수 있는 작은 서비스)*** 와, ***independently deployable(독립적 배포 가능)***
  - 장점
    - 배포(Deployment) : 서비스 별 개별 배포 가능
    - 확장(scaling) : 특정 서비스에 대한 확장 용이(클라우드 사용에 적합)
    - 장애(failure): 장애가 전체 서비스로 확대될 가능성이 적음
  - 단점
    - 성능 : 서비스 간 호출 시 API를 사용하기 때문에 통신 비용이나 Latency 증가
    - 테스트/트랜잭션 : 서비스가 분리되어 있어 테스트와 트랜잭션의 복잡도가 증가하고 많은 자원을 필요
    - 데이터 관리 : 데이터가 여러 서비스에 걸쳐 분산되기 때문에 한번에 조회하기 어렵고 데이터의 정합성 또한 관리하기 어려움

브라우저 동작 원리

- Parsing
  - HTML을 DOM 트리로 변환
  - Image, CSS, Script 등 다운로드해야 할 리소스 다운로드
    - CSS의 경우 다운로드 한 후 CSSOM(CSS Object Model) 트리로 변환
    - 다운받은 자바스크립트를 인터프리트, 컴파일, 파싱 및 실행
- Rendering
  - DOM 트리와 CSSOM 트리를 합쳐 render 트리를 만든다
    - 문서 내용인 HTML과 디자인인 CSSOM을 합치는 것
  - 레이아웃 정하기
    - 트리의 각 노드가 브라우저의 어디에, 어떤 크기로 배치될지 결정
  - 브라우저 스크린에 렌더 트리의 각 노드를 그려 준다
  - 사용자는 브라우저 상에서 시각화된 HTML을 보게 된다

SPA

- Single Page Application
- 한번 페이지를 로딩하면 사용자가 임의로 새로고침 하지 않는 이상 새로 페이지를 로딩하지 않는 application
- React의 화면 표시
  - 첫 페이지 로딩: index.html 로딩
  - <body></body> 를 렌더링 하는 중 bundle.js 로딩
  - bundle.js는 index.js 를 포함하고 있으며
  - Index.js에 있는 ReactDom.render() 함수가 실행되고 동적으로 HTML 엘리먼트를 리액트 첫 화면으로 바꿔준다
  - 렌더링된 하위 엘리먼트는 <App />
  - 페이지를 바꾸고 싶다면 root의 하위 엘리먼트를 다른 HTML로 수정함으로써 가능
  - 브라우저의 자바스크립트는 fetch, ajax 등의 함수로 서버에 데이터를 요청하고 받은 데이터를 이용하여 자바스크립트 내에서 HTML을 재구성한다. 이렇게 서버에게 새 HTML 페이지를 요청하지 않고 자바스크립트가 동적으로 HTML을 재구성하여 만드는 클라이언트 애플리케션이라하고 이러한 렌더링 과정을 client-side rendering 이라고 함.

CSR(Client Side Rendering)

- 한 개의 페이지로 이루어진 애플리케이션
- 하나의 페이지를 동적으로 변경하여 콘텐츠를 렌더링
- 클라이언트 영역에서 화면을 제어하는 렌더링 방식
- 페이지의 reloading이 없어 자연스러운 화면 전환과 좋은 UX 제공
- 브라우저의 자바스크립트는 fetch, ajax 등의 함수로 서버에 데이터를 요청하고 받은 데이터를 이용하여 자바스크립트 내에서 HTML을 재구성한다. 이렇게 서버에게 새 HTML 페이지를 요청하지 않고 자바스크립트가 동적으로 HTML을 재구성하여 만드는 클라이언트 애플리케이션 이라 하고 이러한 렌더링 과정을 client-side rendering 이라고 한다.
- 단점
  - 애플리케이션의 규모가 커지면 초기에 로딩하는 자바스크립트나 CSS 번들 파일 용량이 증가하여 초기 로딩 성능 저하 원인이 됨
  - 검색 엔진 최적화(Search Engine Optimization) 어려움
    - 검색 엔진의 봇(bot)은 자바스크립트 코드를 실행하지 않고 파싱된 HTML 만을 대상으로 크롤링하기 때문에 실제로 렌더링 되는 데이터를 크롤링하기 어려움

SSR(Server Side Rendering)

- 사용자에게 보여질 모든 페이지를 구성하여 사용자에게 보여주는 방식
- MPA(Monolithic Page Application)에서 사용하던 렌더링 방식
- 페이지의 전체 콘텐츠가 구성되는 시점이 빠르다는 장점
- 이미 구성이 완료된 페이지이므로 검색 엔진의 봇(bot)에서 별도의 자바스크립트 코드를 실행하지 않아도 올바른 페이지 데이터 크롤링 가능
- 필요한 콘텐츠를 모두 구성하기 위해 추가적인 자바스크립트 실행이 필요하기 때문에 CSR 보다 느림
  - CSR을 사용하되 적재적소에 SSR을 사용하여 단점 보완
  - 동적인 구성 요소가 많아 초기 페이지의 완벽한 구성 시간이 오래 걸릴 때(특히 네트워크가 느릴 경우) SSR을 도입하여 페이지의 콘텐츠 렌더링 완료 시점 개선
- CSR + SSR 방식을 고수준의 API로 지원하는 프레임워크
  - Next.js(React 기반)
  - Nuxt.js(Vue.js 기반)

React Programming

개발 환경 준비

- Node.js 다운로드
  - https://nodejs.org/en/download/releases
  - Node.js 14.4.0
- Node.js 설치
- Node.js 및 npm 설치 확인
  - node –v, npm –v
- React 프로젝트 workspace 폴더 생성
- yarn 설치
  - npm install
  - yarn –v
- create-react-app 설치
  - npm install –g create-react-app
- 프로젝트(client) 생성
  - create-react-app client
- React 서버 실행
  - yarn start
  - 브라우저에서 http://localhost:3000

Front-end app 생성

- Workspace 생성
  - mkdir react-workspace
  - cd react-workspace
- React app. 생성
  - npx create-react-app react-project
  - cd react-project
- React app. 실행
  - npm start
- Visual studio code에서 개발 환경 설정
  - File - add Folder to workspace
  - 위에서 생성한 react-project 선택 – add
- Material-ui 패키지 설치
  - npm install @material-ui/core
  - npm install @material-ui/icons

React 기본 개념

- 리액트의 핵심 개념에는 큰 변화가 없었으나 리액트를 사용하는 방식에는 꽤 많은 변화
- 2015년 등장한 Redux(state, 상태 관리 라이브러리)로 인해 단일 불변 스토어 사용, 컴포넌트와 컨테이너의 구분, Redux-Thunk / Redux-Saga 등을 이용한 비동기 처리 등의 개념은 이제 거의 관용적인 패턴으로 자리잡음.
- 중요한 개념인 Higher Order Component(HOC)를 이용한 코드 재사용 패턴도 사실상의 표준으로 정립됨.
- HOF(Higher order function)
- 함수형 프로그래밍
- 리액트에서 가장 중요한 개념 중의 하나임에도 불구하고 개발자들이 주로 간과하는 부분이 있는데, 그건 바로 리액트가 함수형 프로그래밍을 지향
- 컴포넌트는 (순수)함수
  - 리액트의 컴포넌트는 기본적으로 함수이다. 그것도 순수 함수. 좀더 자세히 이야기하자면 입력값으로 props(properties)를 받고 출력으로 ReactElement 트리를 반환하는 순수 함수
  - 리액트는 순수함수로 컴포넌트를 생성함

Front-end 개발 도구

- 의존성 관리 도구
  - npm(Node package manager)
    - Node.js의 패키지 관리 도구
    - 모든 패키지는 npm registry에 저장
    - 패키지는 npm cli를 사용하여 설치
    - package.json 파일로 프로젝트와 패키지 정보 관리
  - npx
    - npm 5.2 버전부터 기본 제공되는 패키지 실행 도구
    - 최신 버전의 패키지 가져와 실행
  - yarn
    - 페이스북에서 만든 Node.js 패키지 매니저
    - npm의 성능과 의존성 관리 문제 해결
    - package.json 파일을 기준으로 동작

Front-end 개발 도구

- Webpack
  - 모듈 지원이 불가한 브라우저에서의 의존성 관리 문제 해결
  - 가장 많이 사용되는 bundler
  - npm install –D webpack webpack-cli
- 트랜스파일러(transpiler)
  - 브라우저들은 브라우저 마다 ECMAScript 구현율이 조금씩 달라 어떤 브라우저에서든 애플리케이션의 일관성 있는 실행 환경 구축 필요가 있음
  - Babel
    - ES2015로 작성된 코드를 구형 브라우저에서도 동작 가능하도록 ES5로 변환
    - Webpack에서 babel-loader를 사용하여 번들링 작업 시 자바스크립트 코드를 트랜스파일함
    - JSX 코드를 자바스크립트 코드로 변환
- Sass(Syntactically Awesome Stylesheets)
  - Css 파일로 변환되는 스타일 시트 언어
  - Css를 프로그래밍 언어처럼 작성 가능

Front-end 개발 도구

- Linter
  - 소스 코드를 분석하여 오류, 오타, 잠재적 버그 찾아주는 도구
  - 자바스립크트 린터 ESLint
  - 코드 포맷터 Prettier
  - CSS 린터 Stylelint
- React 개발자 도구: 크롬 웹스토어에서 설치

React component(Ex: App)

- 자바스크립트 함수 또는 자바스크립트 클래스 형태로 생성
- JSX 문법 적용
  - 한 파일에서 HTML과 JavaScript를 함께 사용하려고 확장한 문법
- App 컴포넌트는 렌더링 부분인 HTML과 로직 부분인 자바스크립트를 포함하는 jsx 를 리턴
- JSX 구문은 빌드시에 Babel 이라는 transpiler가 자바스크립트로 번역
- App() 컴포넌트의 사용
  - ReactDOM.render() 함수의 매개변수로 <App />을 주면 ReactDOM는 <App /> 컴포넌트로 DOM 트리를 만드는데 이때 컴포넌트의 render 함수가 리턴한 jsx를 렌더링한다.
  - ReactDOM.render() 함수의 첫번째 매개변수는 react 컴포넌트, 두번째 매개변수는 root 엘리먼트이고, 첫번째 매개변수인 react 컴포넌트를 두번째 매개변수인 root 엘리먼트 아래에 추가하라는 의미
  - Root 엘리먼트는 index.html에 정의되어 있으며, 모든 react 컴포넌트는 이 root 엘리먼트 하위에 추가된다.

React 특징

- Virtual DOM
  - 실제 DOM에 접근하여 조작하는 대신 이를 추상화 자바스크립트 객체를 구성하여 사용
  - 실제 DOM의 사본
  - 리액트에서 데이터(state)가 변경되어 웹브라우저에 실제 DOM을 업데이트할 때의 과정
    - 데이터를 업데이트하면 전체 UI를 Virtual DOM에 리렌더링
    - 이전 virtual DOM 에 있던 내용과 현재 내용을 비교
    - 바뀐 부분만 실제 DOM에 적용

React 라이브러리

- React 기본
- React 와 데이터