2023-하계방학 자기주도학습동아리 활동보고서

팀명	JUMP		
학습일시	2023년 07월 24일 (월요일)	진행차수	5주차
학습시간	15시 00분 ~ 17시 00분 (2시간)		
참석자	이 영 교 (서명)	김 태 현 (서명)	손 민 재 (서명)
	이 호 영 (서명)	최 재 신 (서명)	진 정 원 (서명)
불참자	X		
구 분	내 용		
학습목표	- 의존성 주입 패턴과 서비스 로케이터 패턴에 대해 알아보자 - 웹 페이지의 기능을 정의하고, 필요한 기술 문서를 분석해보		
학습 방법 및 학습 내용	0. 이번 주차는 노션에 내용을 정리해서 세션처럼 설명하는 방식으로 진행했습니다. 코드를 작성하는데 있어, 어떻게 하면 더 깔끔하고 간편하게 코드를 관리할 수 있을까에 대한 고민을 하던 중 의존성 주입(DI)와 제어의 역전(IOC), 즉 결합성을 낮추고 응집성을 높게 가져가는 코드 작성 방법에 대해 알아보는 시간을 갖게 되었습니다. 처음 코드를 발견했을 때는 자바로 작성된 코드가 많았지만, 이를 프론트엔드에서도 사용하기 위해 자바스크립트 버전으로도 따로 작성해보는 시간을 갖게 되었습니다. - 노션 링크 : 1) DI&IOC https://citrine-situation-94c.notion.site/DI-IOC-NodeJS-Version-255fe5e8c2d34f12999b008a8bc87d83?pvs=4 1. 의존성 주입(DI)와 제어의 역전(IOC) 객체지향 프로그래밍 패러다임에서 소프트웨어는 높은 응집성(Cohesion)과 낮은 결합성(Coupling)을 갖도록 설계되어야 하며, 확장성이 높은 모듈을 만들 수 있어야 한다. 평소대로 코드를 작성하면, 비슷한 기능을 설계하는 과정에서 기능별로 따로 코드를 분리해서 작성해야 하지만, 이러한 패턴을 사용하게 되면, 비슷한 기능은 기존에 작성한 코드를 재사용할 수 있어 개발하는 시간과 복잡성을 낮출 수 있다는 장점이 있다. - 응집성 : 하나의 모듈이 단일의 역할만을 할 경우 높은 응집성 - 결합성 : 모듈이 다른 모듈에 얼마나 의존하는지 결합성을 낮추기 위해, 상위 수준 모듈이 하위 수준 모듈에 의존하지 않도록 설계하는 것을 의존성 역전 원칙이라고 부른다. 이를 구현하기 위한 패턴으로는 의존성 주입 패턴을 사용할 수 있으며, 자바로 작성하는 스프링부트로는 자동으로 빈에서 관리하여 따로 개발하지 않아도 모듈 자체를 가져다가 사용하면 되는 구조이고, 자바스크립트의 NodeJS는 TypeDI를 사용하여 의존성을 주입할 수 있다.		

```
// 좋지 않은 종속성 패턴
// 구현체를 직접 import 의 예
const user = require('./user') // 하드 코딩됨

exports.login = (username, password, callback) => {
  user.get(username, (err, user) => {
    if(err) return callback(err)
  } ...
}
```

이는 자바스크립트로 작성한 코드의 일부로, 자바스크립트를 사용하여 개발된 코드들을 보면 하드코딩된 의존 관계를 구축하는 코드를 많이 볼 수 있다. 인터페이스와 상관 없이 임의의 객체를 집어 넣어 테스트할 수 있기 때문이라고 생각이 되는데, 이는 다른 언어와 다르게 동적 타입 언어의 특징이라고 볼 수 있다.

이런 의존성 구현체를 의존하지 않게 개발하기 위해서는 2가지 패턴을 생각해볼 수 있다.

2. 서비스 로케이션 패턴

의존성이 있는 각 객체가 서비스 로케이터 객체만을 직접 의존하고 각 객체는 서비스 로케이터에 의존성을 명시해 구현체를 받아오게 설계하는 방법이다.

```
// AuthController.js -> AuthService에 의존
// AuthController는 ServiceLocator에만 직접 의존
module.exports = (serviceLocator) => {
  const authService = serviceLocator.get('authervice'); /
  // require()와 비슷하게 사용하지만, 전체 경로를 받아오지 않는다는 점
}
```

서비스 로케이션 패턴의 장점으로는 의존성이 구현체에 의존하지 않게 설계되기 때문에 의존성 주입과 동일하게 비슷한 기능을 구현하는 과정에서 코드를 재사용할 수 있다는 장점이 있다. 하지만, 객체의 구현 코드를 보지 않으면 곧바로 의존 관계를 파악하기 어렵다는 점과 생성자 등으로 명시되지 않기 때문에 문서화를 해야 한다는 불편함을 가지고 있었다.

그래서 우리는 의존성 주입 패턴을 사용하여 개발하기로 했다.

3. 의존성 주입 패턴

DI(의존성 주입 패턴)은 의존 관계를 가장 잘 다루는 방법으로 알려져 있으며, 모듈의 의존성을 외부 개체에 의해 입력으로 전달 받는 방법이다. 컨테이너와 지원 방식을 구현하는 것이 어렵다는 점이 있지만, 스프링부트는 이미 다 모듈이 구성되어 있기 때문에 간편하게 가져다가 사용하면 된다.

```
// Before DI
const authService = require('./authService');

exports.login = (req, res, next) => {
  authService.login(...);
}
```

```
// After DI
module.exports = (authService) => {
  const authController = {};

  authController.login = (req, res, next) => {
    authService.login(...);
  }
}
```

4. 총정리

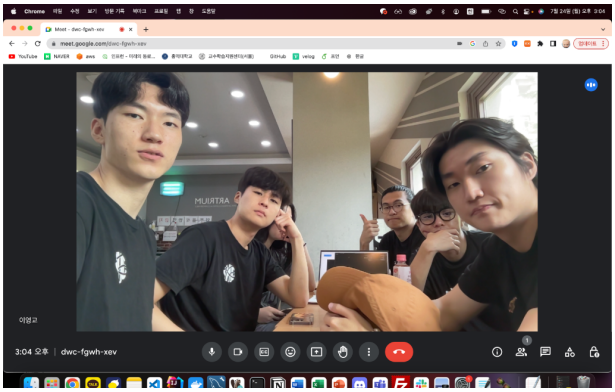
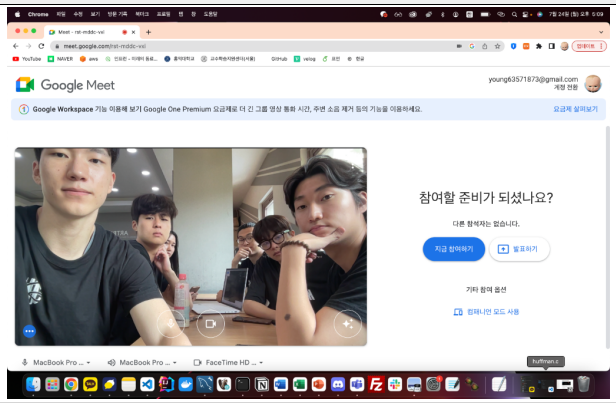
전체적으로 이번 세션에서 정리한 주제는 코드를 개발하는 과정에서 클린한 코드, 즉 관리하기 편하고 보기 편리한 코드를 작성하기 위한 방법 중 하나로 의존성 주입 패턴을 활용하는 방법에 대해 공부했습니다. 이는 프론트엔드 백엔드 둘 모두에 적용되는 방법으로, 기존의 하드 코딩된 코드보다 훨씬 여러 개의 비슷한 기능을 구현하는데 있어, 시간적으로나 관리 측면에서 이점을 얻을 수 있는 방법이었습니다.

추가로, 2주 후부터 구현할 웹 페이지의 기능들을 정의하고, 기능을 구현하기 위한 기술들을 찾아보는 시간을 갖게 되었습니다.

활동 후기

* 본인이 맡은 역할 활동과 학습한 내용에 대한 후기 작성

팀장 / 팀원 이름	작성란
이영교	**다른 팀원들의 회의 내용을 기록하며 해당 주차에 준비해온 프론트엔드 개발 지식들을 공유하는 역할을 했습니다. 이번 주차는 의존성 주입에 대해 전달하는 세션을 진행했는데, 기존의 코드보다 더 깔끔하게 정리할 수 있을 것 같아 한층 업그레이드 된 모습을 보고 뿌듯했습니다.**
김태현	**이번 주차는 백엔드에서도 중요하게 다뤄지는 의존성 주입 패턴을 공부했는데, 스프링부트에서 원래 사용하던 방법이라 어떤 이점이 있는지 몰랐는데 알게 되었습니다. 기능들도 정의하면서 추가로 살펴봐야 하는 문서들을 확인할 수 있는 시간이었습니다.**
손민재	**유튜브와 인터넷 강의를 통해 자바스크립트와 리액트를 학습해왔고, 다음**

		주부터 있을 웹 개발 과정에서 실제 실무 환경에서 많이 사용할 기능들을 직접 작성해보며 연습할 예정이다.
	이호영	신입 부원 모집을 위한 사이트를 직접 만든 동아리가 있다는 소식을 듣고 그 사이트를 비롯한 실제 신입 부원 모집 공고들을 살펴보며 삽입할 기능이나 작성받을 내용들을 살펴보았다.
	최재신	대면으로 진행하는 시간으로 확실히 비대면보다 집중을 더 잘 할 수 있었고, 모르는 것을 질문하기 수월했습니다.
	진정원	스프링 부트에서 사용하는 컨트롤러, 서비스 레이어 간의 데이터 전달 방법을 알게 되었고, 빈으로 관리되고 있다는 것이 아직 명확히 의미를 알 수 없어 추가로 공부할 예정입니다.
활동 증빙 사진	학습시작	
		
	학습종료	
		
	학습활동