Typescript Migration

Javascript to Typescript

2025.08.19

두선아

목차

타입스크립트

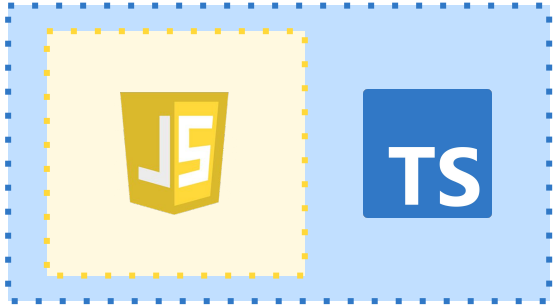
타입스크립트는 자바스크립트의 Superset이다. (O / X)

모든 자바스크립트는 타입스크립트이다. (O / X)

타입스크립트는 자바스크립트의 런타임 동작을 모델링하는 타입 시스템을 가지고 있다. (O / X)

Superset language

TypeScript is **a strongly typed programming language** that **builds on JavaScript**, giving you better tooling at any scale.



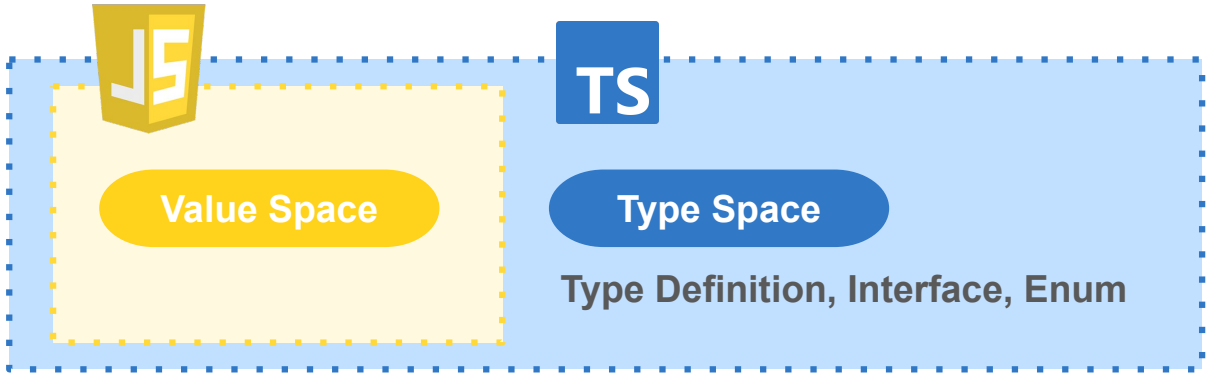
TypeScript is JavaScript with syntax for types.

타입스크립트는 **타입이 정의**된 **자바스크립트**의 Superset (상위 집합) 입니다.
=> 자바스크립트의 문법을 모두 포함하면서, 추가적으로 타입 시스템을 얹은 언어
=> 자바스크립트에 없는 문법이 추가되어 있다. ex) 타입 정의, interface, enum 등

Javascript 와 Typescript

$JS \subseteq TS$ 모든 유효한 JS 는 유효한 TS 이다.

$TS \not\subseteq JS$ 일부 TS 는 JS가 아니다



- **Typescript**: https://www.typescriptlang.org/

Javascript의 타입

자바스크립트는 **원시 타입 (Primitive Type)**과 **객체 타입 (Object Type)**을 기반으로 유연한 타입을 가지고 있습니다.

7가지 원시값
string, number, bigint, boolean,
undefined, symbol, null

```
let x = 10;      // number
x = "hi";        // string
x = { a: 1 };    // object
```

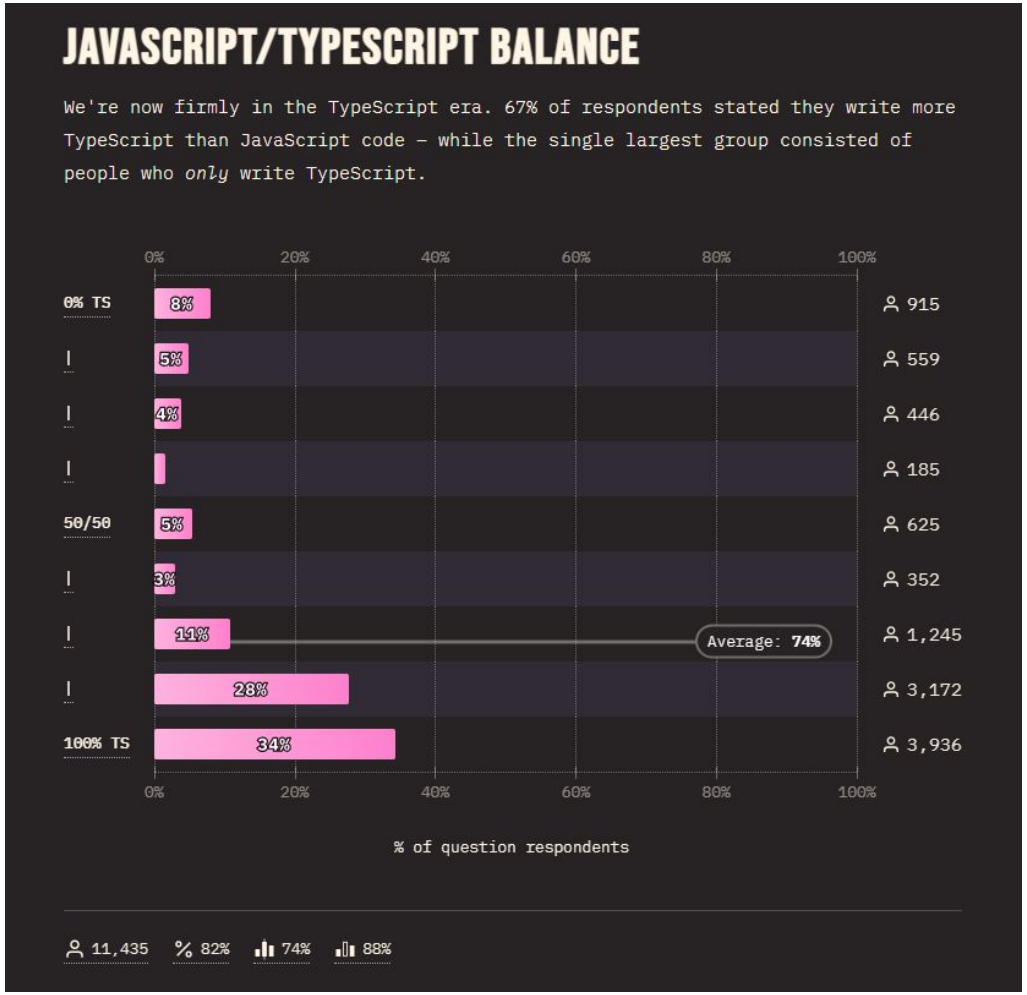
```
String.prototype.hello = function() {
  return "Hello, " + this;
};

console.log("World".hello()); // "Hello, World"
```

자바스크립트 형변환 예시

형변환 예시	결과	설명
"b" + "a" + + "a" + "a"	baNaNa	단항 + 연산자는 문자열을 숫자로 변환 시도 → NaN "ba" + NaN + "a"
true + true	2	true → 1, 1 + 1 = 2
!ItemList?.length	ItemList 배열이 비었을 때 true ItemList 배열에 요소가 존재하면 false ItemList가 length를 가진 자료형이 아니라면 true	?.length가 0 → falsy → !0 → true ?.length가 0이 아닌 값 → truthy → true ?.length는 undefined → falsy → !0 → true

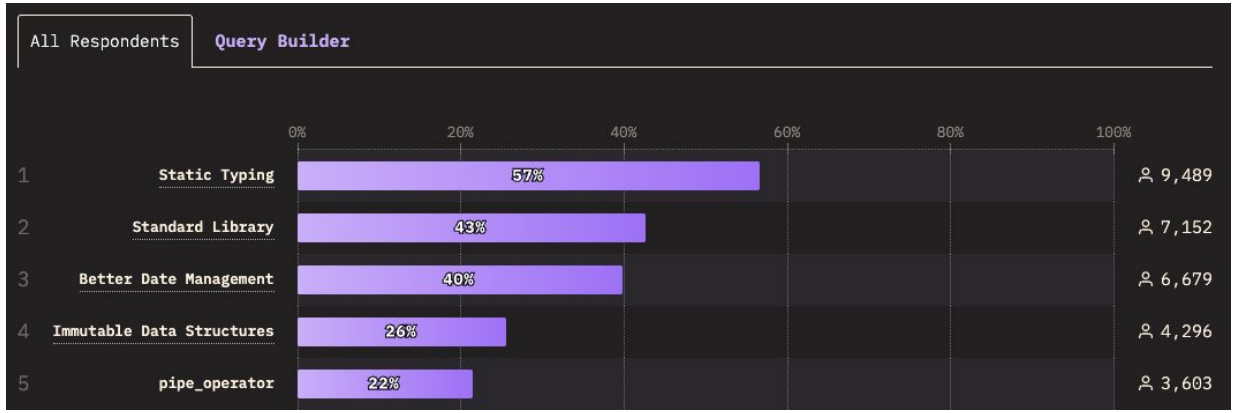
Typescript?



Typescript era

- (2024년 응답 기준) 67%가 **자바스크립트 코드보다 타입스크립트 코드를 더 많이 작성**한다고 응답
- 가장 많은 그룹인 34%가 **타입스크립트로만** 코드를 작성

Missing features



- (Missing features 설문 문항이 존재하는 2020~2024년의 응답 기준) 자바스크립트의 부족한 기능으로 **정적 타입**을 항상 1위로 응답
  - 정적 타입 언어 예시: Java, C, C++, Rust, Go, Kotlin

- **State of JavaScript (2020~2024)**: https://stateofjs.com/en-US

Typescript!

정적 분석 | Static Type

Javascript의 missing feature

타입 수집 | 타입 추론

- 변수, 함수, 클래스 등에 선언된 타입을 확인하거나 추론

정적 타입 검사

- 실행 전에 잘못된 타입 사용을 막아주는 Safety Net 역할

함수 시그니처 | 인터페이스

- 매개변수 타입과 반환 타입이 올바른지 확인

타입 체커 | Type Checker

코드에서 **값의 타입이 올바른지 자동으로 검사**하는 도구

- **타입스크립트**의 타입 검사는
  정적 분석(컴파일 시점 & IDE) 에서만 발생
- 실제 런타임에는 타입 정보 제거 (타입 오류 없음)

타입 검사 시점

1. 컴파일 (빌드) 시 정적 타입 검사 - tsc
2. IDE (에디터)에서 실시간 타입 체킹 - VSC, Cursor 등
3. 자바스크립트의 런타임 값 검사 - typeof, instanceof

TypeScript는 **정적 타입 시스템**을 제공하고,
타입 체커는 **컴파일 시점 & IDE에서만 동작**하며
런타임에는 사라집니다.

타입스크립트의 특징

특성

Optional Typing | Gradual Typing 🥗

선택적, 점진적 타입 시스템

- 유효한 자바스크립트는 타입스크립트이기 때문에 프로그램의 일부에만 타입 시스템을 적용할 수 있습니다.
- **타입이 적용된 TS/TSX 파일과, 그 외의 JS/JSX 파일을 혼합**하여 사용할 수 있기 때문에 부분적인 타입 적용이 가능합니다.
- 부분적으로 타입 체크를 비활성화시켜주는 **ANY 타입** => 점진적 마이그레이션

Dynamic Typing

동적 특성 / 동적 타이핑

타입을 명시하지 않은 변수나 함수는 동적으로 타입이 결정됩니다.

즉, 런타임 시에 변수의 타입이 결정되며 다양한 타입의 값을 가질 수 있습니다.
타입을 명시하지 않은 부분은 런타임에 동적으로 타입을 결정합니다.

Static Typing

정적 특성 / 정적 타이핑

타입을 명시한 변수나 함수는 정적으로 타입이 결정됩니다.
즉, 컴파일 시에 타입이 고정되며, 해당 타입의 값만을 가질 수 있습니다.

컴파일 타임에 타입 오류를 발견하고 수정합니다. 타입을 명시함으로써 IDE에서의 자동 완성을 제공하고, 코드 리팩토링, 타입 안전성 등의 혜택을 받을 수 있습니다.

- 『이펙티브 타입스크립트』

퀴즈 풀어보기

여기서 잠깐!

TS

```ts
interface Fruits {
  name: string;
  details: {
    info: {
      icon: string;
    };
  };
}

const fruits: Fruits[] = [
  { name: "Apple", details: { info: { icon: "🍎" } } },
  { name: "Orange", details: { info: { iocn: "🍊" } } }, // 1
  { name: "Grape", details: { info: { icon: "🍇" } } },
  { name: "Cherry", details: null }, // 3
];

for (const fruit of fruits) {
  console.log(fruit.details.icon); // 2
}
```

Quiz ?

코드의 1, 2, 3번 위치에 경고가 발생합니다

3가지 경고 메시지와, 발생 위치는 어디일까요?

<보기>

- a) Type 'null' is not assignable to type '{ info: { icon: string; }; }'.
- b) Property 'icon' does not exist on type '{ info: { icon: string; }; }'
- c) Type '{ iocn: string; }' is not assignable to type '{ icon: string; }'.
  Object literal may only specify known properties,
  and 'iocn' does not exist in type '{ icon: string; }'.
- d) 'fruit.details' is possibly 'undefined'.

TS vs JS

Quiz ?

TS { noImplicitAny: false }

```ts
// 경우 A
function 둘_나누기_A(input) {
  return input / 2;
}
console.log(둘_나누기_A("10"));

// 경우 B
function 둘_나누기_B(input: number) {
  return input / 2;
}
console.log(둘_나누기_B("10"));

// 경우 C
function 둘_나누기_C(input: number) {
  return input / 2;
}
console.log(둘_나누기_C("10" as any));
```

경고가 발생할까요?
어떤 위치에서 발생할까요?
이유는 무엇인가요?

둘_나누기_C("10" as any)의 결과는 무엇인가요?

TS vs JS

- 『이펙티브 타입스크립트』 스터디 장표: https://github.com/monthly-cs/2024-05-effective-typescript/blob/main/docs/dusunax/presentation/week_1.pdf

퀴즈 풀어보기: 정답

심심할 때 읽어보세요😊

TS

```ts
interface Fruits {
  name: string;
  details: {
    info: {
      icon: string;
    };
  };
}

const fruits: Fruits[] = [
  { name: "Apple", details: { info: { icon: "🍎" } } },
  { name: "Orange", details: { info: { iocn: "🍊" } } }, // 1
  { name: "Grape", details: { info: { icon: "🍇" } } },
  { name: "Cherry", details: null }, // 3
];

for (const fruit of fruits) {
  console.log(fruit.details.icon); // 2
}
```

보기 a - 위치 3
보기 c - 위치 1
보기 b - 위치 2

Quiz !

a) Type 'null' is not assignable to type '{ info: { icon: string; }; }'.

- null로 초기화할 수 없습니다. null을 사용하고 싶다면 details: { info: { icon: string }} | null 과 같이 작성합니다.

c) Type '{ iocn: string; }' is not assignable to type '{ icon: string; }'.
Object literal may only specify known properties, and 'iocn' does not exist in type '{ icon: string; }'.

- 오타를 작성한 경우입니다!

b) Property 'icon' does not exist on type '{ info: { icon: string; }; }'

- fruit.details.info.icon과 같이 접근해야 합니다.
- 마찬가지로 휴먼에러입니다.

TS vs JS

Quiz !

TS { noImplicitAny: false }

```ts
// 경우 A
function 둘_나누기_A(input) {
  return input / 2;
}
console.log(둘_나누기_A("10"));

// 경우 B
function 둘_나누기_B(input: number) {
  return input / 2;
}
console.log(둘_나누기_B("10"));

// 경우 C
function 둘_나누기_C(input: number) {
  return input / 2;
}
console.log(둘_나누기_C("10" as any));
```

둘_나누기_C("10" as any)는 5를 반환합니다
자바스크립트는 암시적 타입 변환을 통해 "10"을 10으로 강제 타입변환(type coercion)합니다.

경우A

noImplicitAny가 false 이므로 경고가 발생하지 않습니다
매개변수 input의 타입은 any입니다

경우B

인수의 타입이 일치하지 않아 경고가 발생합니다

```
Argument of type 'string' is not assignable to parameter of type 'number'. ts(2345)
```

경우C

타입 단언(type assertion)된 인수를 전달했기 때문에
경고가 발생하지 않습니다
실제와 다른 타입을 단언했기 때문에
런타임에 에러가 발생할 수 있는 가능성이 있습니다.

TS vs JS

- 『이펙티브 타입스크립트』 스터디 장표: https://github.com/monthly-cs/2024-05-effective-typescript/blob/main/docs/dusunax/presentation/week_1.pdf

to be!

개발 환경 개선

타입 안정성

유지보수 용이

마이그레이션 시작하기

tsconfig.json

타입스크립트 컴파일러에게 규칙을 알려주기 위한 파일

- 설정 파일이 없어도 타입스크립트 프로젝트를 만들 수 있으나,
  다른 도구와의 호환성을 보장하기 위해 설정 파일을 사용합니다.

JS → TS 점진적 마이그레이션 프로젝트

엄격 모드(strict)를 꺼서 타입 안정성을 낮추고,
js와 any를 허용해서 마이그레이션 편의성을 높인 설정 파일입니다.

noEmit: 타입 검사만 수행하고 실제 JavaScript 파일은 생성하지 않음 (tsc --noEmit)

“react-jsx”: 리액트 프로젝트 (React 17+ JSX Transform)

allowJS: .js 파일을 프로젝트에 포함

noImplicitAny: 타입이 명시되지 않은 값이 any가 되도록 허용
(타입스크립트 컴파일러는 모든 함수의 매개변수에 대한
타입을 명시하지 않으면 경고를 발생시킵니다)

You, 5 days ago | 1 author (You)

```jsonc
{
  "$schema": "https://json.schemastore.org/tsconfig",
  "compilerOptions": {
    "declaration": true,
    "declarationMap": true,
    "moduleResolution": "node",
    "noUncheckedIndexedAccess": true,
    "resolveJsonModule": true,
    "target": "ESNext",
    "useDefineForClassFields": true,
    "lib": ["DOM", "DOM.Iterable", "ESNext"],
    "module": "ESNext",
    "noEmit": true,
    "skipLibCheck": true,
    "strict": false,
    "noUnusedLocals": false,
    "noUnusedParameters": false,
    "isolatedModules": true,
    "esModuleInterop": true,
    "incremental": true,
    "jsx": "react-jsx",
    "allowJs": true, // 마이그레이션을 위한 JavaScript 파일 허용
    "noImplicitAny": false, // 마이그레이션을 위한 any 허용
  },
  "include": [
    "src/**/*"
  ],
  "exclude": [
    "node_modules",
    "build"
  ]
}
```

You, 5 days ago

예제 프로젝트

TypeScript 마이그레이션 프로젝트 예제:
https://github.com/dusunax/ts-migration-example

```
ts-migration-example/
├── javascript/          # 1 순수 JavaScript 프로젝트
├── typescript/          # 2 TypeScript 마이그레이션
├── react-jsx/           # 3 React + JSX 프로젝트
├── react-tsx/           # 4 React + TypeScript (TSX)
└── README.md
```

1

```js
// 사용 예제
const userManager = new UserManager();
console.log('🚀 JavaScript User Manager 시작');

// 사용자 추가
userManager.addUser('김철수', 'kim@example.com', 25);
userManager.addUser('이영희', 'lee@example.com', '서른');
userManager.addUser('박민수');
```

2

```ts
// 사용 예제
const userManager = new UserManager();
console.log('🚀 TypeScript User Manager 시작');

// 사용자 추가 (타입 안전성 보장)
userManager.addUser('김철수', 'kim@example.com', 25);
userManager.addUser('이영희', 'lee@example.com', '서른');
// ❌ Argument of type 'string' is not assignable to parameter of type 'number'.ts(2345)
userManager.addUser('박민수');
// ❌ index.ts(19, 31): An argument for 'email' was not provided.
```

3

useUser.js

ts-migration-example > react-jsx > src > hooks > useUser.js > ...

You, 59 minutes ago | 1 author (You)

```js
import { useState } from 'react'

function useUser() {
  const [users, setUsers] = useState([])
  const [nextId, setNextId] = useState(1)

  const addUser = (userData) => {
    const newUser = {
      id: nextId,
      ...userData,
      createdAt: new Date().toLocaleDateString()
    }
    setUsers(prevUsers => [...prevUsers, newUser])
    setNextId(prevId => prevId + 1)
  }

  const deleteUser = (id) => {
    setUsers(prevUsers => prevUsers.filter(user => user.id !== id))
  }

  const updateUser = (id, updates) => {
    setUsers(prevUsers =>
      prevUsers.map(user =>
        user.id === id ? { ...user, ...updates } : user
      )
    )
  }

  return {
    users,
    addUser,
    deleteUser,
    updateUser
  }
}

export default useUser
```

4

useUser.ts

ts-migration-example > react-tsx > src > hooks > useUser.ts > useUser > updateUser

You, 13 minutes ago | 1 author (You)

```ts
import { useState } from 'react'
import { User, UserFormData } from '../types/User'

const useUser = () => {
  const [users, setUsers] = useState<User[]>([])
  const [nextId, setNextId] = useState<User['id']>(1)

  const addUser = (userData: UserFormData) => {
    const newUser: User = {
      id: nextId,
      name: userData.name,
      email: userData.email,
      age: userData.age,
      createdAt: new Date().toLocaleDateString()
    }
    setUsers(prev => [...prev, newUser])
    setNextId(prev => prev + 1)
  }

  const deleteUser = (id: User['id']) => {
    setUsers(prev => prev.filter(user => user.id !== id))
  }

  const updateUser = (id: User['id'], updates: Partial<User>) => {
    setUsers(prev => prev.map(user =>
      user.id === id ? { ...user, ...updates } : user
    ))
  }

  return {
    users,
    addUser,
    deleteUser,
    updateUser
  }

export default useUser;
```

const updateUser: (id: number, updates: Partial<User>) => void

updateUser You, 13 minutes ago • Uncommitted changes

DefinitelyTyped

타입스크립트용 타입 정의 파일(.d.ts)들을 모아놓은 오픈소스 커뮤니티 프로젝트

- 대부분의 인기 JS 라이브러리에 대한 타입 정의가 존재
  - **npm install @types/라이브러리명**
- @types/*로 제공되는 타입 정의 파일(.d.ts)은 컴파일 시점에만 사용되고, 실제 빌드된 JavaScript 파일에는 포함되지 않습니다
  - devDependency에 설치하는 것이 관례

```
"devDependencies": {
  "@types/crypto-js": "^4.2.2",
  "@types/jest": "^30.0.0",
  "@types/lodash": "^4.17.20",
  "@types/node": "^24.2.1",
  "@types/react": "^19.1.10",
  "@types/react-dom": "^19.1.7",
  "@types/react-highlight": "^0.12.8",
  "@types/react-router-dom": "^5.3.3",
```

- DefinitelyTyped 한글 README: https://github.com/DefinitelyTyped/DefinitelyTyped/blob/master/README.ko.md
- DefinitelyTyped 타입 목록 (8월 18일 기준 7,657 폴더) : https://github.com/DefinitelyTyped/DefinitelyTyped/tree/master/types

Swagger TypeScript API

Swagger/OpenAPI 문서로부터 TypeScript API 타입 혹은 클라이언트를 자동 생성해주는 도구

설치

- npm install swagger-typescript-api

파일 생성

- npx swagger-typescript-api -p https://스웨거문서경로.json -o ./src/api -n api.ts
- node scripts/generate-api-types.js https://스웨거문서경로.json

생성되는 파일

- HTTP 요청 함수 (generateClient가 true일 때)
- **요청/응답 타입 정의**: Backend과 Frontend의 타입 일치로 **타입 안정성 확보**
- 유니온 타입 및 라우트 타입 (옵션 선택 가능)

```
const { generateApi } = require('swagger-typescript-api');

async function generateApiTypes() {
  try {
    console.log('🚀 API 타입 생성을 시작합니다...');

    if (!fs.existsSync(OUTPUT_DIR)) {
      fs.mkdirSync(OUTPUT_DIR, { recursive: true });
    }

    const prettierConfig = await loadPrettierConfig();

    const { files } = await generateApi({
      url: SWAGGER_URL,
      output: OUTPUT_DIR,
      prettier: prettierConfig,

      httpClientType: 'axios',      // axios 기반 클라이언트
      generateClient: false,        // API 클라이언트 코드 생성
      generateRouteTypes: true,     // 라우트 관련 타입 생성
      generateResponses: true,      // 응답 타입 생성
      generateUnionEnums: true,     // 유니온 타입 생성
      sortTypes: true,              // 타입 정의 정렬
      sortRoutes: true,             // 라우트 정렬
    });

    console.log('✅ API 타입 생성이 완료되었습니다!');
    console.log(`📄 생성된 파일 목록:`);
    files.forEach(file => console.log(`   - ${file.fileName}${file.fileExtension}`));
    console.log(`📁 출력 경로: ${OUTPUT_DIR}`);
  } catch (error) {
    console.error('❌ API 타입 생성 중 오류가 발생했습니다:', error);
    process.exit(1);
  }
}

generateApiTypes();
```

- Swagger TypeScript API: https://github.com/acacode/swagger-typescript-api

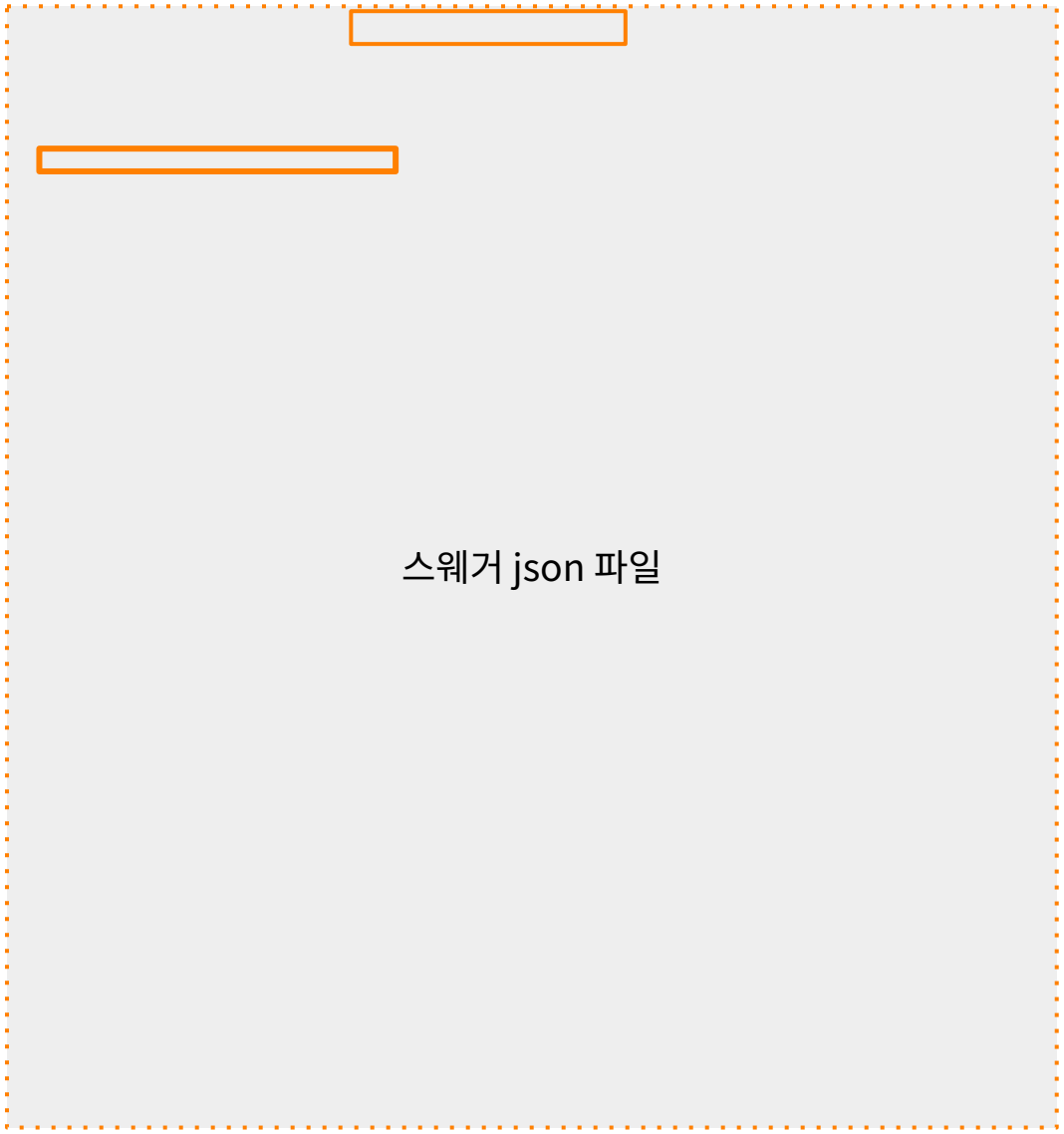
1. API 타입 정의
2. 주요 로직 타입 정의
3. 마이그레이션 100%
4. any, js 허용 속성 제거

예시 코드

프로젝트 적용 예시

1. Swagger Document (UI)

2. Swagger Document (json)

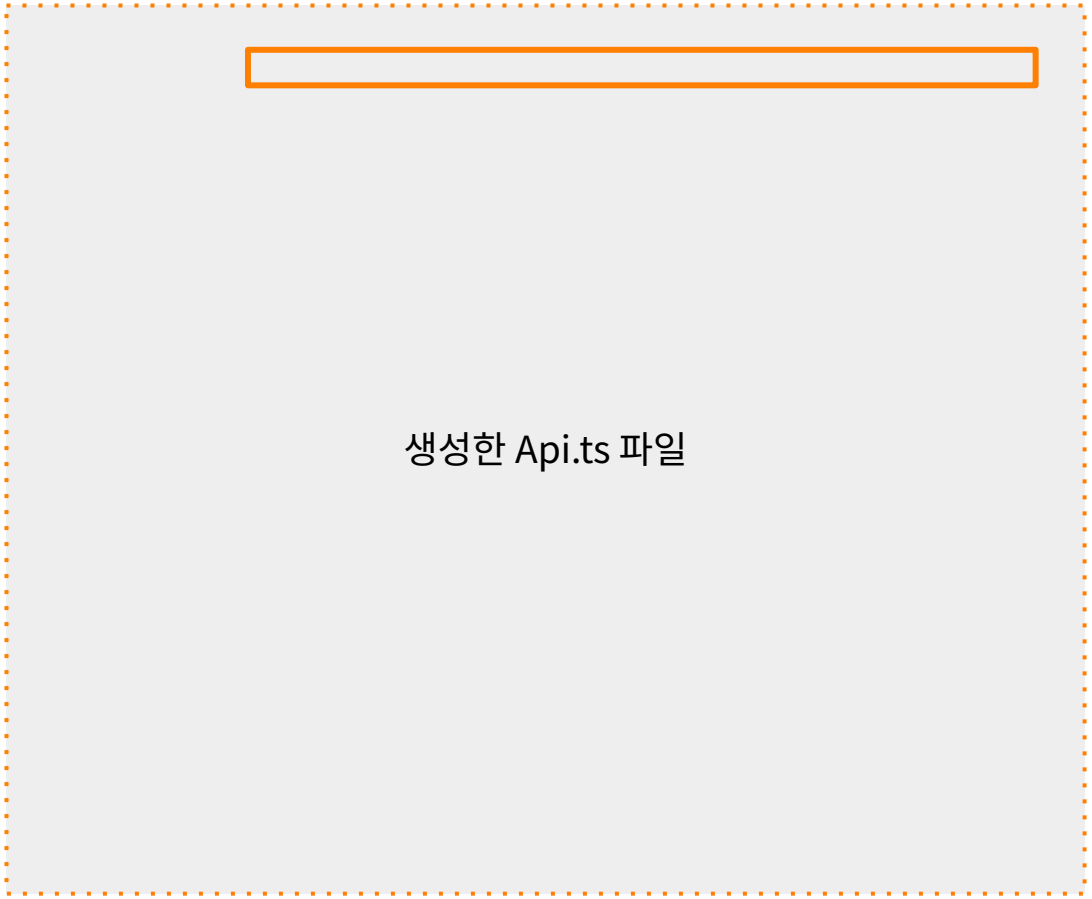


예시 코드

프로젝트 적용 예시

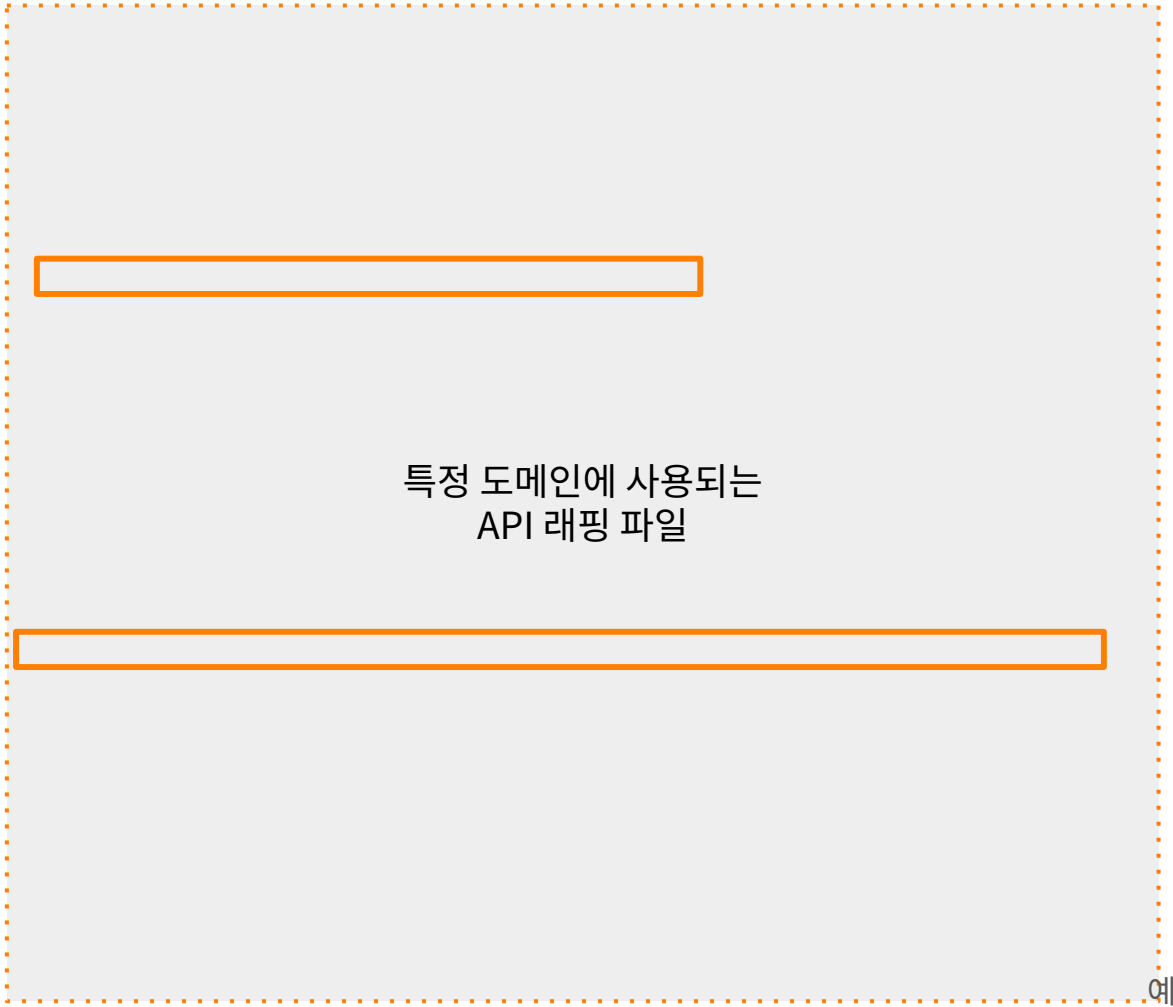
3. Swagger TypeScript API

- 타입 생성 스크립트 실행 => 지정한 경로에 Api.ts 파일 생성
- 클라이언트(API 요청 함수)를 생성할 수 있음 (타입 정의가 목적이므로 타입만 생성)



4. 가독성을 고려한 래핑 파일 구성

- 생성된 타입, 유틸리티 타입, API 관련 클라이언트 전용 타입
- Cursor, Copilot 과 같은 IDE 기반 AI 코딩 어시스턴트 활용



예시 코드

프로젝트 적용 예시

Cursor, Copilot 과 같은 IDE 기반 AI 코딩 어시스턴트 활용

Api.ts 타입을 사용해서 프론트엔드에서 사용하기 편하게 정리한 특정기능_타입.ts 타입 파일을 src/types/api에 만들어줘.

AI 코딩 어시스턴스가 작성한 타입 => 확인 작업

- 각 타입의 일치 여부
- 불필요한 중복
- 주석 정리

types/api
경로

코파일럿
대화

예시 코드

프로젝트 적용 예시

5. 기존 /apis 파일 교체

- API request/response 타입 정의(contract checking)
- 함수 시그니처 정의

As-is

자바스크립트 + any

- API request/response의 타입이 any
- 함수 입력값/출력값이 any

만약 JSDoc을 사용해 타입 정보를 부여한다면?
타입 오류 체크를 위해 IDE의 도움을 받을 수 있으나,
타입 추론이 되지 않기 때문에 주석을 업데이트하지 않으면
불일치가 발생할 수 있습니다.

To-be

As-is
+ 타입

- API request/response의 타입 정의
  - 함수의 입력값/출력값 정의

호버했을 때 나타나는 함수 정보

예시 코드

프로젝트 적용 예시

6. IDE 지원, 언어 서비스 활용하기

- 자동완성 (IntelliSense)

IDE
자동 완성

- 타입을 사용하고 있는 곳 확인하기, 레퍼런스 검색(Find References)
- 호버 시 타입 정보 확인, 단축키로 타입 정의로 이동(Go to Definition)

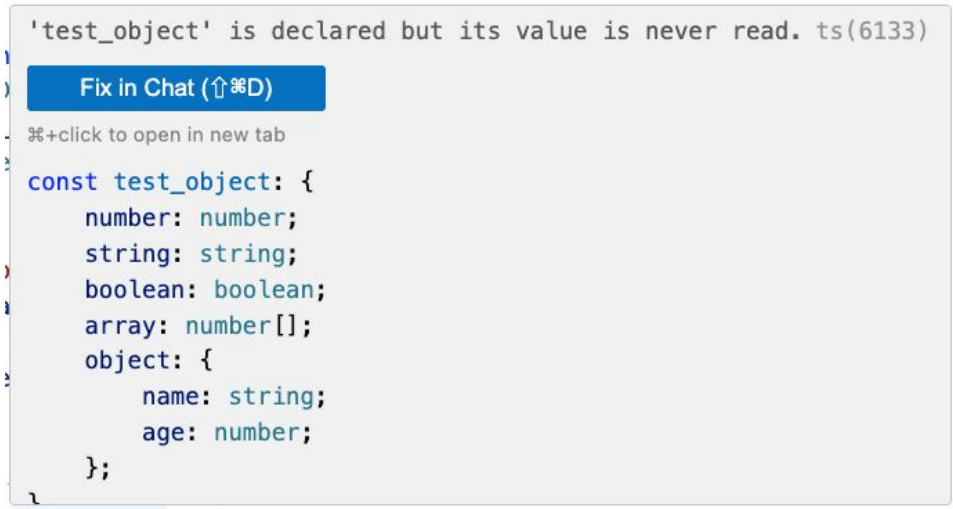
타입 정보

- 에러 감지: 코드를 실행하지 않고 IDE에서 타입 오류/잘못된 호출 즉시 표시

런타임에서 발생할 수 있는 오류를
타입 오류로 확인

- 코드 리팩터링 지원
https://code.visualstudio.com/Docs/languages/typescript#_refactoring

- 타입 추론: 모든 변수를 일일이 지정하지 않고 타입을 추론(권장)

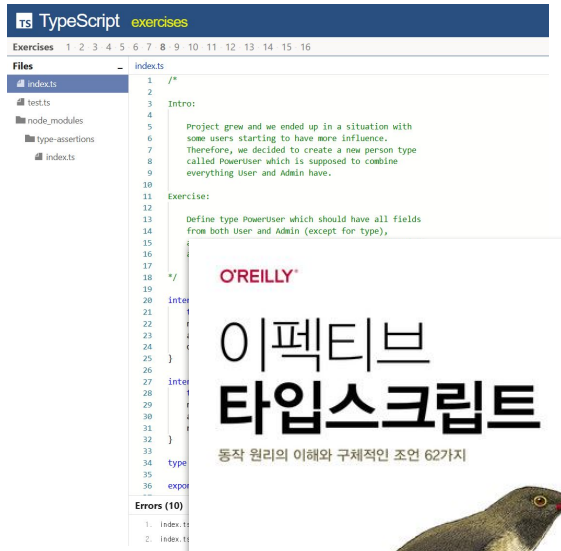


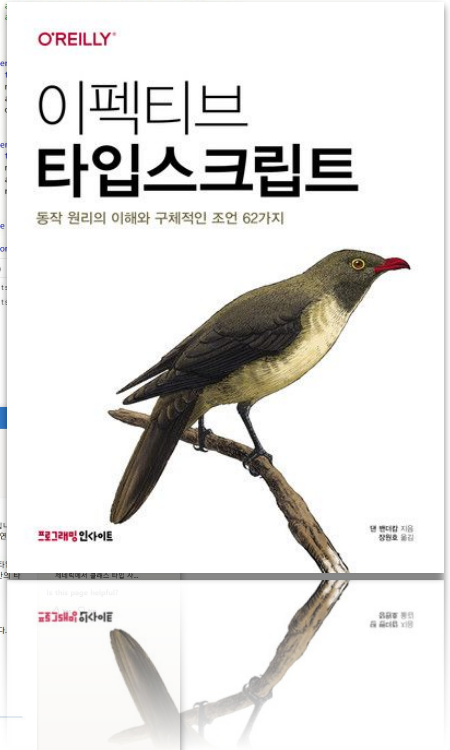
예시 코드

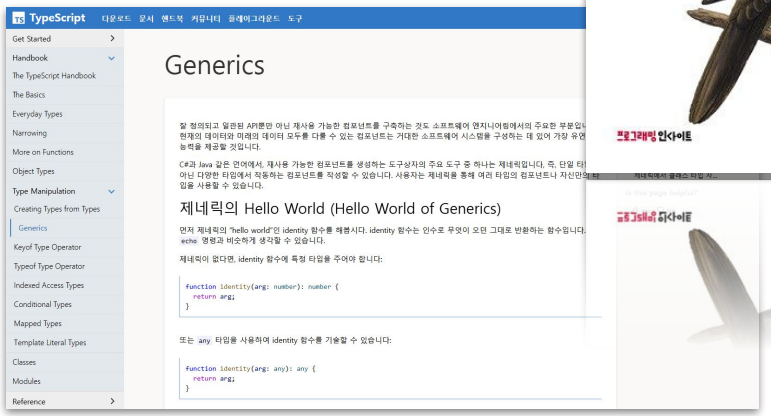
추가 학습 자료

추천

1. 타입스크립트 핸드북: https://www.typescriptlang.org/ko/docs/handbook/intro.html
2. 타입스크립트 연습: https://typescript-exercises.github.io/
3. 『이펙티브 타입스크립트』 : https://www.oreilly.com/library/view/ipegtibeu-taibseukeuribteu/9788966263134/
    - 참고용 장표: 타입스크립트 알아보기, 타입 시스템, 타입 추론, 타입 설계, Any 타입
4. 직접 코드 작성하기 💪







추가 학습 자료

감사합니다.

contact: https://github.com/dusunax