

# Fullstack Development

# JS

# History

- <https://roadmap.sh/guides/history-of-javascript>

# Where is JS?

- Browser
  - Manipulate DOM.
- Host OS
  - JavaScript runtimes ([NodeJS](#), [Bun](#))

# Activity

- Run `console.log` in a browser and in an OS.

# Learning JS

- <https://roadmap.sh/javascript>
- <https://roadmap.sh/questions/javascript>



# TypeScript

# What is TypeScript?

- TypeScript is a super set of JavaScript.



# Why TypeScript?

- Improves your productivity while helping avoid bugs.
  - Catch bugs at the compile-time instead of at runtime.
- Brings the future JavaScript to today.
  - You can use the new JavaScript features before web browsers (or other environments) fully support them.
- More competitive in job market.

# How it works?

1. Write TypeScript codes. (`.ts` files)
2. Compile the TypeScript codes into plain JavaScript codes (`.js` files) using a TypeScript compiler.
3. Run JavaScript codes in an environment that JavaScript runs.

# Tools

- TypeScript Playground: <https://www.typescriptlang.org/play>
- VSCode Extension
  - Quokka.js
  - Pretty TypeScript Errors

# Demo

```
let a = 10;  
a.slice(0, 1); // See error message
```

```
let a: number;  
a = "Hello"; // See error message
```

```
let a: number;  
// Try typing "a" and trigger intellisense.
```

# Defining types

- Types by inference
  - TypeScript knows the JavaScript language and will generate types for you in many cases.
- Type by specification
  - We define it ourselves.
  - Keywords `type`, `interface`

# Type by inference

```
const user = {  
  name: "Hayes",  
  id: 0,  
};  
  
user.name = 20; // Error  
console.log(user.food); // Error
```

- TypeScript already knows the type of this variable.

# Type by specification

```
interface User {  
  name: string;  
  id: number;  
}
```

or

```
type User = {  
  name: string;  
  id: number;  
};
```



# Type Annotation

```
interface User {  
  name: string;  
  id: number;  
}  
  
// type User = {  
//   name: string;  
//   id: number;  
// };  
  
const user: User = {  
  name: "Hayes",  
  id: 0,  
  age: 30, // Error  
};
```

## type vs interface

- They are very similar, and for the most common cases act the same.
- However, TypeScript doc recommends `interface`.
  - `interface` provides better error message.
  - `interface` can be extended.

# More type demo

- [https://github.com/fullstack-67/typescript/blob/main/lecture/03\\_more\\_type\\_demo.ts](https://github.com/fullstack-67/typescript/blob/main/lecture/03_more_type_demo.ts)

# Type utilities

```
interface User {  
  id: number;  
  name: string;  
}  
  
// Extend  
interface UserExtended extends User {  
  isActive: boolean;  
}
```

[More information](#)

# Function (argument type)

```
// Give warning
function sumNumber(a, b) {
  return a + b;
}

// OK
function sumNumber(a: number, b: number) {
  return a + b;
}
```

# Function (type guards)

```
// Hover cursor on "text" to see the type
function greeter(text: string | null | undefined) {
  if (!text) {
    console.log("...");
    return;
  }
  console.log(text);
}
```

# Generics

Generics provide variables to types.

```
interface Backpack<Type> {  
  add: (obj: Type) => void;  
  get: () => Type;  
}
```

```
const backpack: Backpack<string> = {  
  add: (myStr) => {  
    myStr.slice(0, 1);  
  },  
  get: () => {  
    return "Hi";  
  },  
};
```

# Use Typescript in NodeJS project



- `npm init -y`
- `npm install -D typescript ts-node`
- Create `./src` and `./dist` directory

- Create `./src/index.ts`

```
function sayHello(name: string) {  
  console.log("Hello " + name);  
}  
  
sayHello("World");
```

## Compile

- `npx tsc src/index.ts --outDir dist`

## Run (node)

- `node ./dist/index.js`

## Run (ts-node)

- `npx ts-node ./src/index.ts`

# Use TS configuration file

- `npx tsc --init`
- Modify `tsconfig.json`

```
{  
  "compilerOptions": {  
    "outDir": "./dist"  
  },  
  "include": ["./src/**/*.ts"]  
}
```

## Use TS configuration file (cont)

- Now just type `npx tsc`
- Use `npx tsc --showConfig` to see config.

# Code

```
async function getData() {  
  const res = await fetch("https://jsonplaceholder.typicode.com/todos/1");  
  return await res.json();  
}  
  
getData().then((data) => console.log(data));
```

# Code

```
import fs from "fs";  
  
const dir = fs.readdirSync(__dirname);  
console.log(dir);
```

- If you don't have `ts-node`, you need to `npm install -D @types/node`

# TypeScript - NextJS



# Installation

```
npx create-next-app@latest
```

```
>_ pwsh >fullstack-67 38ms
>> npx create-next-app@latest
Need to install the following packages:
create-next-app@14.2.4
Ok to proceed? (y) y
✓ What is your project named? ... typescript-nextjs
✓ Would you like to use TypeScript? ... No / Yes
✓ Would you like to use ESLint? ... No / Yes
✓ Would you like to use Tailwind CSS? ... No / Yes
✓ Would you like to use `src/` directory? ... No / Yes
✓ Would you like to use App Router? (recommended) ... No / Yes
✓ Would you like to customize the default import alias (@/*)? ... No / Yes
Creating a new Next.js app in C:\Users\nnnpo\Coding\fullstack-67\typescript-nextjs.
```

tsconfig.json

```
{  
  // ...  
  "paths": {  
    "@app/*": ["../app/*"],  
    "@components/*": ["../components/*"]  
  }  
}
```

./globals.css

```
@tailwind base;  
@tailwind components;  
@tailwind utilities;  
  
// Remove everything else
```

./components/card.tsx

```
import { FC } from "react";

interface Props {
  title: string;
  text?: string;
}

const Card: FC<Props> = ({ title, text }) => {
  return (
    <div className="border border-gray-300 p-2 rounded shadow-sm flex flex-col items-center">
      <div className="font-bold text-lg text-gray-800">{title}</div>
      <div className="text-gray-600">{text ?? "...."}</div>
    </div>
  );
};

export default Card;
```

./app/page.tsx

```
import Card from "@components/Card";

export default function Home() {
  return (
    <div className="container flex m-2 gap-2">
      <Card title="Card 1" />
      <Card title="Card 2" text="Content Here" />
    </div>
  );
}
```