Typescript

Ts-node

Yükle. Ts node tsc yazarak terminalde typescripte erişip kod yazmamızı saglıyor.

Index.ts dosyasini hazirladiktan sonra tsc index.ts yaparak ts dosyamizin index.js adiylka ciktisini aliyoruz ve node index.js yaparak calistiriyoruz.

Yada kisaca ts-node index.ts yapabiliriz.

**TYPE DEFINITION FILE**

Typescriptte 3. Parti npm modulleri kutuphanelerini kullanmak istiyorsak type definition file larini yuklememiz gerekiyor bu modullerin. Fs cyrpto nodemon gibi. Eger hali hazirda iclerinde type definition file leri varsa yuklememize gerek yok. Eger hata aliyorsaniz import yapinca muhtemelen icinde yoktur.

Npm install @types/{name of library}

Npm install @types/cyrpto

Node icinde hazi hazirla olan moduller icinse type definitionlar

Npm install @types/node

seklinde yuklenir.

ENUMS – JAVASCRIPT

*// enum - enumeration == Obje gibi icinde string ve sayi tutabilirler*

*// enumlar diger gelistiricilere bu birbirine cok yakin degerler iceren bir colelction mesaji verir*

*//TITLE SUBJECT gibi dynamic degismesi gereken yerlerde kullanmayiz. Fakat belirli olan bir daha degismeyecek olan ve cok sayida olmayan tanimlamalar icin kullanabiliriz.*

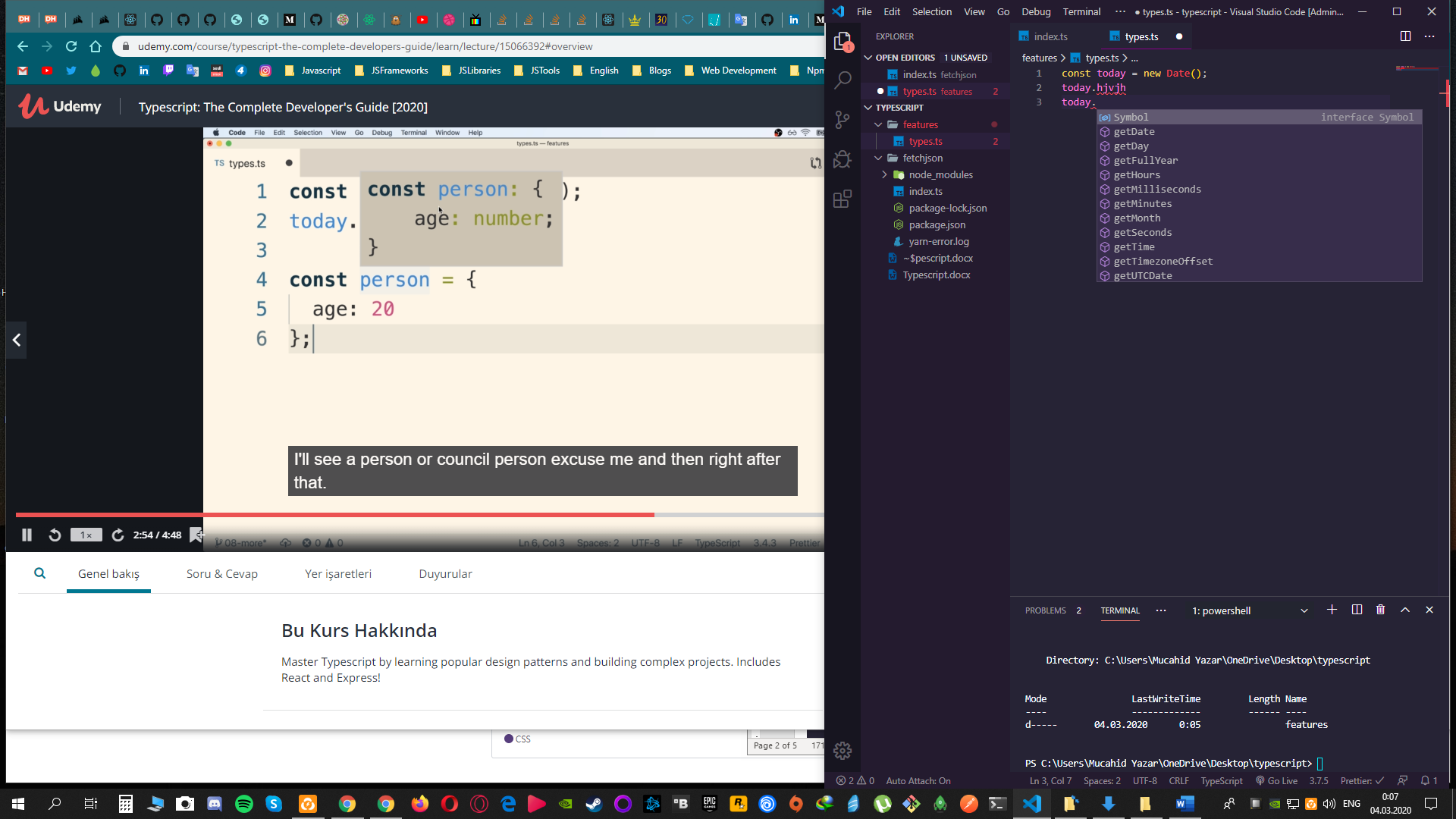
enum matchResult {

  HomeWin = "H",

  AwayWin = "A",

  Draw = "D"

}



Tsc –init yaparak tsconfig.json dosyasini olusturabiliriz.

outDir dosyanin cikarilacagi yer. rootDir dosyanin alinacagi yer. Index dosyalari alinir herhangi ozel bir dosya isim verilemzse.

Ve artik tsc yazarak cikartma islemleriniz yapabiliriz. Tsc -w yaparakda izlemesini saglariz ve her save sonrasi otomatik olarak cikartmayi kendisi yapar.

Nodemon ve concurrently yukle

    "start:build": "tsc -w",

    "start:run": "nodemon build/index.js",

    "start": "concurrently npm:start:\*" *//Npm Startlarin hepsini calistiracak*

{

  "compilerOptions": {

*/\* Basic Options \*/*

*// "incremental": true,                   /\* Enable incremental compilation \*/*

    "target": "es5" */\* Specify ECMAScript target version: 'ES3' (default), 'ES5', 'ES2015', 'ES2016', 'ES2017', 'ES2018', 'ES2019', 'ES2020', or 'ESNEXT'. \*/*,

    "module": "commonjs" */\* Specify module code generation: 'none', 'commonjs', 'amd', 'system', 'umd', 'es2015', 'es2020', or 'ESNext'. \*/*,

*// "lib": [],                             /\* Specify library files to be included in the compilation. \*/*

*// "allowJs": true,                       /\* Allow javascript files to be compiled. \*/*

*// "checkJs": true,                       /\* Report errors in .js files. \*/*

*// "jsx": "preserve",                     /\* Specify JSX code generation: 'preserve', 'react-native', or 'react'. \*/*

*// "declaration": true,                   /\* Generates corresponding '.d.ts' file. \*/*

*// "declarationMap": true,                /\* Generates a sourcemap for each corresponding '.d.ts' file. \*/*

*// "sourceMap": true,                     /\* Generates corresponding '.map' file. \*/*

*// "outFile": "./",                       /\* Concatenate and emit output to single file. \*/*

    "outDir": "./build" */\* Redirect output structure to the directory. \*/*,

    "rootDir": "./src" */\* Specify the root directory of input files. Use to control the output directory structure with --outDir. \*/*,

*// "composite": true,                     /\* Enable project compilation \*/*

*// "tsBuildInfoFile": "./",               /\* Specify file to store incremental compilation information \*/*

*// "removeComments": true,                /\* Do not emit comments to output. \*/*

*// "noEmit": true,                        /\* Do not emit outputs. \*/*

*// "importHelpers": true,                 /\* Import emit helpers from 'tslib'. \*/*

*// "downlevelIteration": true,            /\* Provide full support for iterables in 'for-of', spread, and destructuring when targeting 'ES5' or 'ES3'. \*/*

*// "isolatedModules": true,               /\* Transpile each file as a separate module (similar to 'ts.transpileModule'). \*/*

*/\* Strict Type-Checking Options \*/*

    "strict": true */\* Enable all strict type-checking options. \*/*,

*// "noImplicitAny": true,                 /\* Raise error on expressions and declarations with an implied 'any' type. \*/*

*// "strictNullChecks": true,              /\* Enable strict null checks. \*/*

*// "strictFunctionTypes": true,           /\* Enable strict checking of function types. \*/*

*// "strictBindCallApply": true,           /\* Enable strict 'bind', 'call', and 'apply' methods on functions. \*/*

*// "strictPropertyInitialization": true,  /\* Enable strict checking of property initialization in classes. \*/*

*// "noImplicitThis": true,                /\* Raise error on 'this' expressions with an implied 'any' type. \*/*

*// "alwaysStrict": true,                  /\* Parse in strict mode and emit "use strict" for each source file. \*/*

*/\* Additional Checks \*/*

*// "noUnusedLocals": true,                /\* Report errors on unused locals. \*/*

*// "noUnusedParameters": true,            /\* Report errors on unused parameters. \*/*

*// "noImplicitReturns": true,             /\* Report error when not all code paths in function return a value. \*/*

*// "noFallthroughCasesInSwitch": true,    /\* Report errors for fallthrough cases in switch statement. \*/*

*/\* Module Resolution Options \*/*

*// "moduleResolution": "node",            /\* Specify module resolution strategy: 'node' (Node.js) or 'classic' (TypeScript pre-1.6). \*/*

*// "baseUrl": "./",                       /\* Base directory to resolve non-absolute module names. \*/*

*// "paths": {},                           /\* A series of entries which re-map imports to lookup locations relative to the 'baseUrl'. \*/*

*// "rootDirs": [],                        /\* List of root folders whose combined content represents the structure of the project at runtime. \*/*

*// "typeRoots": [],                       /\* List of folders to include type definitions from. \*/*

*// "types": [],                           /\* Type declaration files to be included in compilation. \*/*

*// "allowSyntheticDefaultImports": true,  /\* Allow default imports from modules with no default export. This does not affect code emit, just typechecking. \*/*

    "esModuleInterop": true */\* Enables emit interoperability between CommonJS and ES Modules via creation of namespace objects for all imports. Implies 'allowSyntheticDefaultImports'. \*/*,

*// "preserveSymlinks": true,              /\* Do not resolve the real path of symlinks. \*/*

*// "allowUmdGlobalAccess": true,          /\* Allow accessing UMD globals from modules. \*/*

*/\* Source Map Options \*/*

*// "sourceRoot": "",                      /\* Specify the location where debugger should locate TypeScript files instead of source locations. \*/*

*// "mapRoot": "",                         /\* Specify the location where debugger should locate map files instead of generated locations. \*/*

*// "inlineSourceMap": true,               /\* Emit a single file with source maps instead of having a separate file. \*/*

*// "inlineSources": true,                 /\* Emit the source alongside the sourcemaps within a single file; requires '--inlineSourceMap' or '--sourceMap' to be set. \*/*

*/\* Experimental Options \*/*

*// "experimentalDecorators": true,        /\* Enables experimental support for ES7 decorators. \*/*

*// "emitDecoratorMetadata": true,         /\* Enables experimental support for emitting type metadata for decorators. \*/*

*/\* Advanced Options \*/*

    "forceConsistentCasingInFileNames": true */\* Disallow inconsistently-cased references to the same file. \*/*

  }

}

**ABSTRACT CLASSES**

Senaryo soyle bir classimizi baska bir classimiza extends ettigimizde asagidaki this’den sonraki kullandigimiz collection gibi degerlere gerek kalmiyor ve bunlari kaldiriyoruz. Classimizin onune exporttan once abstract ekliyoruz. Ama bu seferde this.swat gibi yerlerde hata alacagiz bunlarida abstract ile class icinde asagidaki gibi promise vererek bunlar suan yok ama gelecekler diyoruz.

ABSTRACT CLASSES asagidaki sekilde olusturmak icin kullanilamazlar

const numbersCollection = new NumbersCollection([10, 3, -5, 0]);

ABSTRACT CLASSES larda var olmayan metodlar icin tanimlamalar isim kullanmalari yapabiliriz.

import { NumbersCollection } from "./NumbersCollection";

interface Sortable {

  length: number;

  compare(*leftIndex*: number, *rightIndex*: number): boolean;

  swap(*leftIndex*: number, *rightIndex*: number): void;

}

export abstract class Sorter {

constructor(public *collection*: Sortable) {} //SILINDI

  abstract compare(*leftIndex*: number, *rightIndex*: number): boolean;

  abstract swap(*leftIndex*: number, *rightIndex*: number): void;

  abstract length: number;

  sort(): void {

    const { length } = **this**;

    for (let i = 0; i < length; i++) {

      for (let j = 0; j < length - i - 1; j++) {

*//if collection is number[]*

*//If collection is an array of numbers*

        if (**this**.compare(j, j + 1)) {

**this**.swap(j, j + 1);

        }

      }

    }

  }

}

**GENERICS**

Oyle bir yapi kuralim istiyoruz ki data: mapData[] = [] olan yerlerde sadece MatchData daki degerler degil tum degerleri any gibi kontrol edelim diyoruz fakat any kullanmak istemiyoruz cunku herne olursa olsun bir deger belirtmek istiyoruzda ayni zamanda bunun icin GENERIC kullanmamiz gerekiyor.

import fs from "fs";

import { MatchResult } from "./MatchResult";

type MatchData = [Date, string, string, number, number, MatchResult, string];

export abstract class CsvFileReader {

  data: MatchData[] = [];

  constructor(public *filename*: string) {}

  abstract mapRow(*row*: string[]): MatchData;

  read(): void {

**this**.data = fs

      .readFileSync(**this**.filename, {

*//utf-8 ile export etmek istiyoruz anlaminda*

*//Eger encoding olmasaydi buffer seklinde donecekti bu reading islemi bize*

        encoding: "utf-8"

      })

      .split("\n")

      .map((*row*: string): string[] => row.split(","))

      .map(**this**.mapRow);

  }

}

Asagida ki gibiayri ayrida tanim yapmak istemiyoruz. Bu cok uzun bir yol.

class HoldNumber {

  data: number;

}

class HoldString {

  data: string;

}

const holdNubmer = new HoldNumber();

holdNubmer.data = 123;

const holdString = new HoldString();

holdNubmer.data = "Stringgggg";

Ve yukaridaki islemi tek bir class tanimlayarak generic tanimlayarak bu sekilde cok basit bir sekilde yapabiliyoruz.

class HoldAnything<TypeOfData> {

  data: TypeOfData;

}

const holdNumber = new HoldAnything<number>();

holdNubmer.data = 123;

const holdString = new HoldAnything<string>();

holdString.data = "sdadsafa";

Genel olarak T olarak kullanilir.

class HoldAnything<T> {

  data: T;

  add(*a*: T): T {

    return a;

  }

}

const holdNumber = new HoldAnything<number>();

holdNubmer.data = 123;

holdNumber.add(10);

const holdString = new HoldAnything<string>();

holdString.data = "sdadsafa";

GERCEK ORNEK

import fs from "fs";

import { MatchResult } from "./MatchResult";

export abstract class CsvFileReader<T> {

  data: T[] = [];

  constructor(public *filename*: string) {}

  abstract mapRow(*row*: string[]): T;

  read(): void {

**this**.data = fs

      .readFileSync(**this**.filename, {

        encoding: "utf-8"

      })

      .split("\n")

      .map((*row*: string): string[] => row.split(","))

      .map(**this**.mapRow);

  }

}

import { CsvFileReader } from "./CsvFileReader";

import { dateStringToDate } from "./utils";

import { MatchResult } from "./MatchResult";

type MatchData = [Date, string, string, number, number, MatchResult, string];

export class MatchReader extends CsvFileReader<MatchData> {

  mapRow(*row*: string[]): MatchData {

    return [

      dateStringToDate(row[0]),

      row[1],

      row[2],

      parseInt(row[3]),

      parseInt(row[4]),

      row[5] as MatchResult, *// Yani bunu diyerek row[5] 'in MatchResult'daki 'H' 'A' veya 'D' olabilecegini typescript anlamis oluyor.*

      row[6] ];}}

**GENERICS**

class ArrayOfNumbers {

  constructor(public *collection*: number[]) {}

  get(*index*: number): number {

    return **this**.collection[index];

  }

}

class ArrayOfStrings {

  constructor(public *collection*: string[]) {}

  get(*index*: number): string {

    return **this**.collection[index];

  }

}

class ArrayOfAnything<T> {

  constructor(public *collection*: T[]) {}

  get(*index*: number): T {

    return **this**.collection[index];

  }

}

new ArrayOfAnything<string>(["a", "b", "c"]);

*// Example of generics with functions*

function printStrings(*arr*: string[]): void {

  for (let i = 0; i < arr.length; i++) {

    console.log(arr[i]);

  }

}

function printNumbers(*arr*: number[]): void {

  for (let i = 0; i < arr.length; i++) {

    console.log(arr[i]);

  }

}

function printAnything<T>(*arr*: T[]): void {

  for (let i = 0; i < arr.length; i++) {

    console.log(arr[i]);

  }

}

printAnything<string>(["a", "b", "c"]);

*//Generic Constraints*

class Car {

  print() {

    console.log("I am a car");

  }

}

class House {

  print() {

    console.log("I am a house");

  }

}

interface Printable {

  print(): void;

}

function printHousesOrCars<T extends Printable>(*arr*: T[]): void {

  for (let i = 0; i < arr.length; i++) {

    arr[i].print();

  }

}

*//Hata vermez cunku House icinde print( instanceine sahip)*

printHousesOrCars<House>([new House(), new House()]);

printHousesOrCars<Car>([new Car(), new Car()]);

import axios from "axios";

### Asagida axiosdan gelen degerlerle tanimlama yaptik.

### Todo yu referans verdikmi sadece buradaki kadar ve isimlerle degiskenleri tanimlariz.

interface Todo {

  id: number;

  title: string;

  completed: boolean;

}

axios

  .get("https://jsonplaceholder.typicode.com/todos/1")

  .then(*res* => {

    const todo = res.data as Todo;

    const id = todo.id;

    const title = todo.title;

    const completed = todo.completed;

    console.log(`

      The Todo with ID: ${id}

      Has a title of: ${title}

      Is it fnished? ${completed}

    `);

    logTodo(id, title, completed);

  })

  .catch(*err* => {

    const error = err.message;

    console.log(error);

  });

### Burada argumanlara olmalari gereken typelari ayarlariz ve bundan sonra eger yukarida logTodo gibi yanlsi siralamada gonderirse typesizcript fonksiyonun hatali oldugunu soyleyecektir.

const logTodo = (*id*: number, *title*: string, *completed*: boolean) => {

  console.log(`

      The Todo with ID: ${id}

      Has a title of: ${title}

      Is it fnished? ${completed}

    `);

};

**TYPESCRIPT WITH REACT**

Create-react-app dosyaAdi –typescript ile boilerplate i olustur.

import React from "react";

import ReactDOM from "react-dom";

*//? isareti koyarak color propsunu optional olarak belirliyoruz. Eger koymasaydik bu componenta color props'u string olarak mecbur verilmek zorunda olacakti*

interface AppProps {

  color?: string;

}

class App extends React.Component<AppProps> {

  render() {

    return <div>React {**this**.props.color}</div>;

  }

}

ReactDOM.render(<App />, document.getElementById("root"));