Full-stack Application Development

TypeScript

Where to Find The Code and Materials?

https://github.com/iproduct/react-typescript-academy-2022
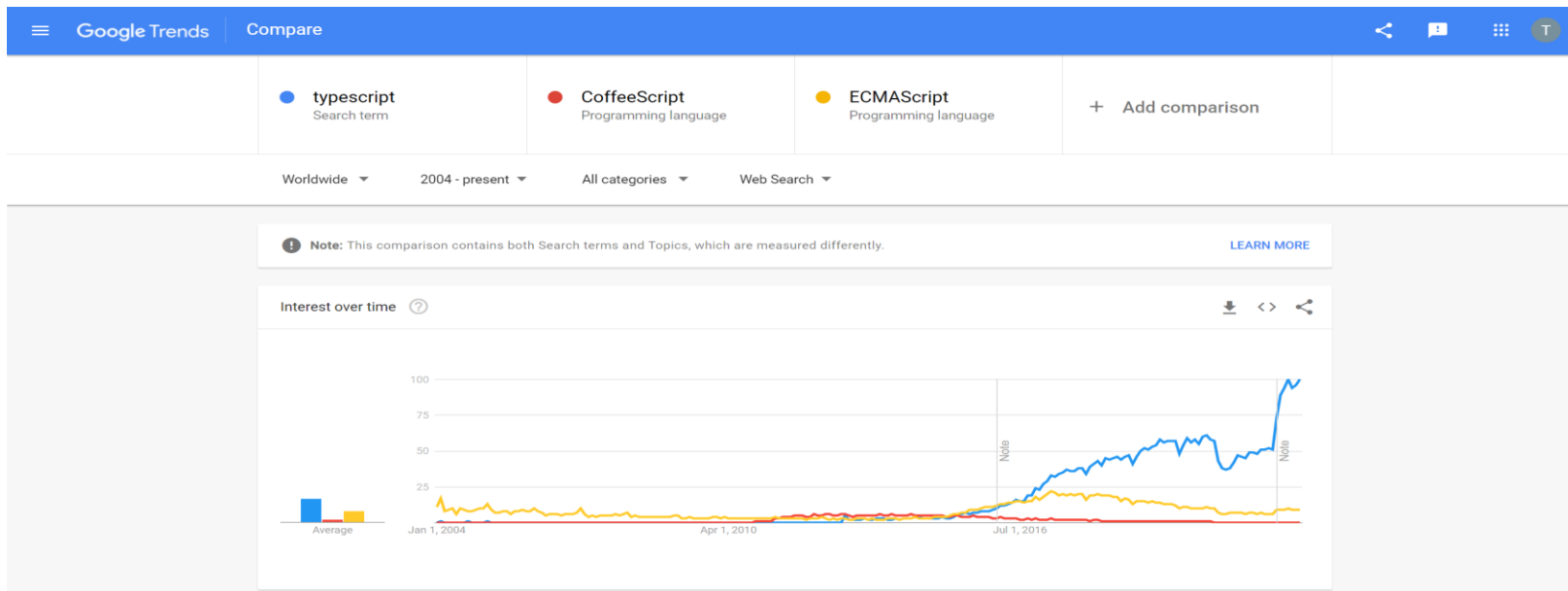
TypeScript
[http://www.typescriptlang.org/]
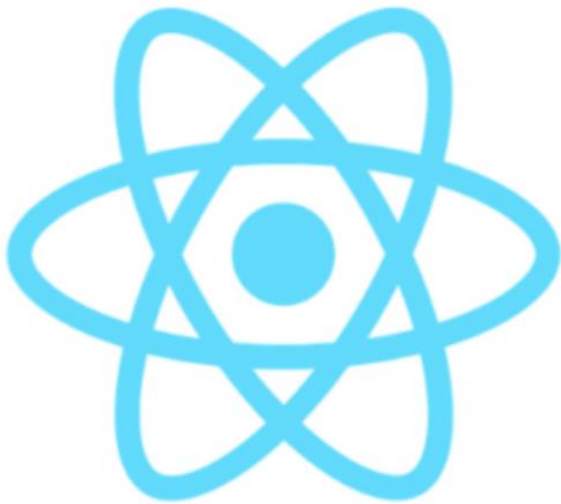
- Typescript → since October 2012, Anders Hejlsberg (lead architect of C# and creator of Delphi and Turbo Pascal)

- Targets large scale client-side and mobile applications by compile time type checking + @Decorators -> Microsoft, Google

- TypeScript is strictly superset of JavaScript, so any JS is valid TS

Source: Google Trends comparison

TypeScript Hello World I

- Installing Typescript: **npm install -g typescript**

- Create new directory: **md 02-ts-demo-lab**

- Create a new TS project using npm: **npm init**

- Write a simple TS function – file **greeter.ts** :

```typescript
function greeter(person: string) {

    return 'Hello, ' + person + ' from Typescript!';

}
const user = 'TypeScript User';

document.body.innerHTML = greeter(user);
```

TypeScript Hello World II

- Compile TypeScript to JavaScript (ES5): **tsc greeter.ts**

- Include script in **index.html** :

```html
<html>
<head>
    <title>ES6 Simple Demo 01</title>
</head>
<body>
    <script src="greeter.js"></script>
</body>
</html>
```

- And open it in web browser – thats all :)

- If you make changes - use watch flag: **tsc -w greeter.ts**

Configuring, Building and Deploying TypeScript Project

- Node package manager (**npm**) configuraton – **package.json**

- TypeScript configuration – **tsconfig.json**, to generate it run: **tsc --init**

- Configuring System.js module loader – **systemjs.config.js**

- Using external JS librarries – **@types** and npm packages

- TypeScript compiler options:
  http://www.typescriptlang.org/docs/handbook/compiler-options.html

- Linting TypeScript code – **tslint.json -> .eslintrc.js**:
  https://palantir.github.io/tslint/rules/

- Developing simple TS project – **Login Demo**

# *typescript-eslint configuration:* **.eslintrc.cjs**

- *npm install --save-dev @typescript-eslint/parser @typescript-eslint/eslint-plugin eslint typescript*

- *yarn add --dev @typescript-eslint/parser @typescript-eslint/eslint-plugin eslint typescript*

- <u>.eslintrc.cjs</u>:

```
/* eslint-disable no-undef */
module.exports = {
    extends: ['eslint:recommended', 'plugin:@typescript-eslint/recommended'],
    parser: '@typescript-eslint/parser',
    plugins: ['@typescript-eslint'],
    root: true,
    rules: {
        "@typescript-eslint/no-non-null-assertion": "off",
    }
};
```

Visual Studio Code
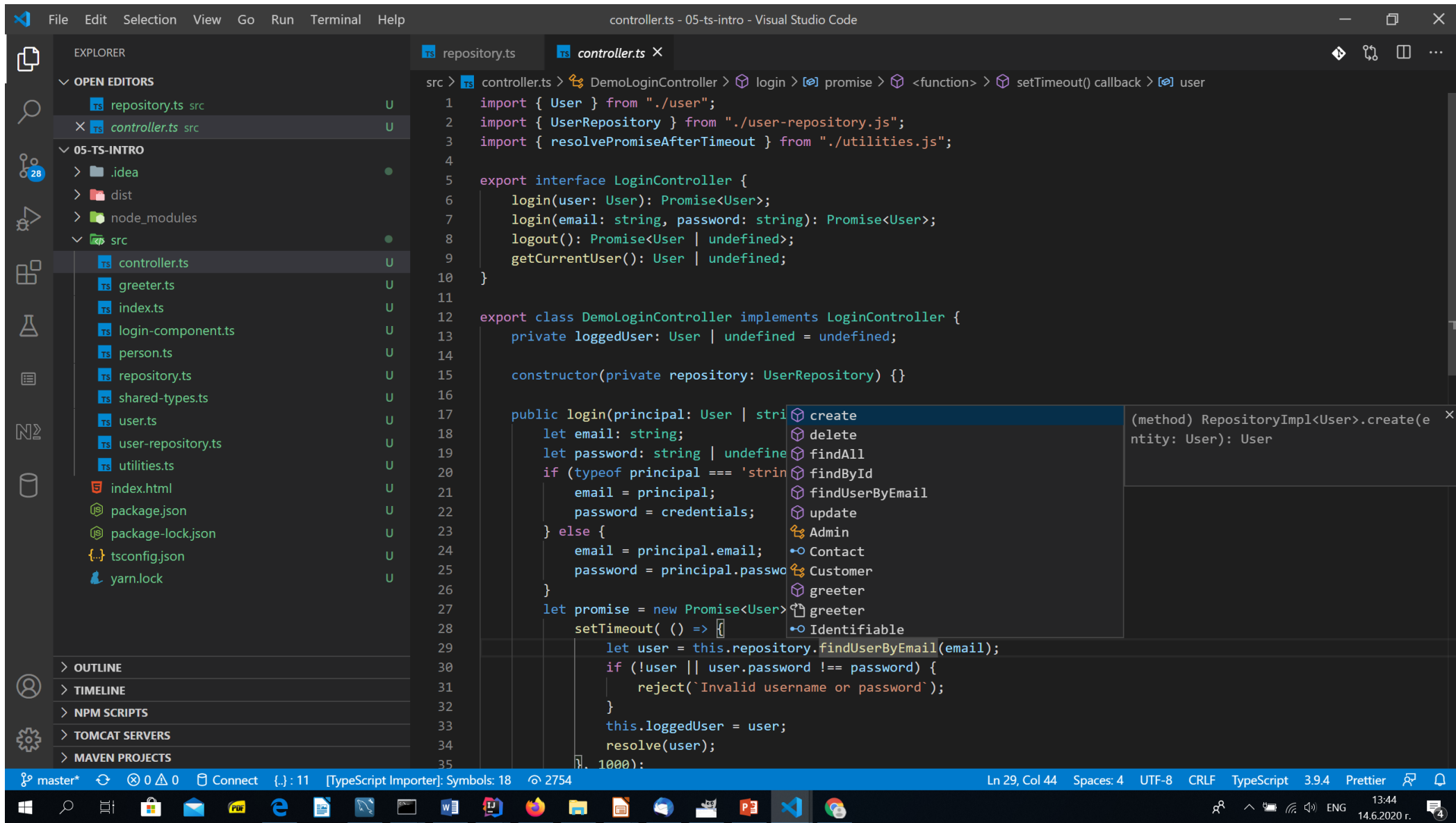
Visual Studio Code

```
10    export default class UserView extends Component<Props> {
        2 references
11        render() {
12            return (
13
14            );
15        }

        0 references
17        private getYearJoined(user: User) {
18            return _.padStart(user.dateJoined.format('yyyy'), 6);
19        }
20

        0 references
21        private getDisplayName(user: User) {
```

VS Code: Popular Angular/TS Plugins I

- **TSLint** - linter for the TypeScript language, help fixing error in TS code. Must have when working with TS -> merged with **ESLint**.

- **ESLint** - linter for the ECMAScript and TypeScript languages, help fixing error in TS code. TSLint announced that it will merge with ESLint:
  https://github.com/typescript-eslint/tslint-to-eslint-config

- **ES7 React/Redux/GraphQL/React-Native snippets** - This extension provides you JavaScript and React/Redux snippets in ES7 with Babel plugin features for VS Code. **Supported languages:** JavaScript (.js), JavaScript React (.jsx), TypeScript (.ts), TypeScript React (.tsx)

- **JavaScript (ES6) code snippets** – This extension contains code snippets for JavaScript in ES6 syntax for Vs Code editor – supports both JavaScript and TypeScript.

- **Typescript React code snippets** – code snippets for React with Typescript.

- **React + Typescript Code Snippets** – opinionated code snippets for React with Typescript

VS Code: Popular Angular/TS Plugins II

- **TypeScript Hero** - Sorts and organizes your imports according to convention and removes imports that are unused (Ctrl+Alt+o on Win/Linux or Ctrl+Opt+o on MacOS).

- **Path Intellisense** - VSCode has a very good auto import capability, but sometime you still need to import some files manually, and this extension helps a lot in these cases.

- **TypeScript Importer** - Automatically searches for TypeScript definitions in workspace files and provides all known symbols as completion item to allow code completion.

- **Debugger for Chrome** - allows to debug using chrome and add your breakpoints in VSCode.

Let's Install Some VSCode Plugins III

Auto Import 1.5.3   959K ★ 4.5
Automatically finds, parses and provides code actions and code completion for all available imports. Works with Typescript and TSX
steoates

Beautify 1.5.0   4.9M ★ 4.5
Beautify code in place for VS Code
HookyQR

Docker 0.8.2   4.7M ★ 4.5
Makes it easy to create, manage, and debug containerized applications.
Microsoft

Document This 0.7.1   685K ★ 3.5
Automatically generates detailed JSDoc comments in TypeScript and JavaScript files.
Joel Day

ES7 React/Redux/GraphQL/React-Native snippets 2.8.0   1.5M ★ 4.5
Simple extensions for React, Redux and Graphql in JS/TS with ES7 syntax
dsznajder

ESLint 1.9.1   9.5M ★ 4.5
Integrates ESLint JavaScript into VS Code.
Dirk Baeumer

Git History 0.6.5   2.5M ★ 4.5
View git log, file history, compare branches or commits
Don Jayamanne

GitLens — Git supercharged 10.2.2   5.4M ★ 5
Supercharge the Git capabilities built into Visual Studio Code — Visualize code authorship at a glance via Git blame annotations and code lens, seamlessly navigate and ex...
Eric Amodio

JavaScript (ES6) code snippets 1.8.0   3.5M ★ 5
Code snippets for JavaScript in ES6 syntax
charalampos karypidis

Let's Install Some VSCode Plugins IV

JSON to TS 1.7.5   ⬇ 190K ★ 4.5
Convert JSON object to typescript interfaces
MariusAlchimavicius

Latest TypeScript and Javascript Grammar 0.0.53   ⬇ 292K ★ 3.5
This is development branch of VSCode JS/TS colorization. Please file any issues you find against https://github.com/Microsoft/TypeScript-TmLanguage/issues
Microsoft

Material Icon Theme 4.1.0   ⬇ 4.4M ★ 5
Material Design Icons for Visual Studio Code
Philipp Kief

Move TS - Move TypeScript files and update relative imports 1.12.0   ⬇ 291K ★ 4.5
extension for moving typescript files and folders and updating relative imports in your workspace
stringham

Node Exec 0.5.1   ⬇ 194K ★ 5
Execute the current file or your selected code with node.js.
Miramac

npm Intellisense 1.3.0   ⬇ 2.1M ★ 4.5
Visual Studio Code plugin that autocompletes npm modules in import statements
Christian Kohler

Nx Console 12.0.0   ⬇ 437K ★ 3.5
Nx Console for Visual Studio Code
nrwl

Paste JSON as Code 12.0.46   ⬇ 500K ★ 4.5
Copy JSON, paste as Go, TypeScript, C#, C++ and more.
quicktype

Peacock 3.7.2   ⬇ 754K ★ 4.5
Subtly change the workspace color of your workspace. Ideal when you have multiple VS Code instances and you want to quickly identify which is which.
John Papa

Let's Install Some VSCode Plugins V

React + Typescript Code Snippets 2.1.0   ⊕1K
Code snippets for react in typescript
weffe

refactorix 0.3.6   ⊕110K ★5
TypeScript refactoring tools for Visual Studio Code
Christian Oetterli

Simple React Snippets 1.2.2   ⊕511K ★5
Dead simple React snippets you will actually use
Burke Holland

TSLint 1.2.3   ⊕1.9M ★3
TSLint support for Visual Studio Code
Microsoft

TypeScript Hero 3.0.0   ⊕593K ★3.5
Additional toolings for typescript
Christoph Bühler

TypeScript Importer 2.0.1   ⊕276K ★4.5
Automatically searches for TypeScript definitions in workspace files and provides all known symbols as completion item to allow code completion.
pmneo

Typescript React code snippets 1.3.1   ⊕98K ★4.5
Code snippets for react in typescript
infeng

Typescript React/Redux Snippets 0.2.0   ⊕15K
Typescript React Snippets
Alexander Botteram

TypeScript Toolbox 0.5.0   ⊕143K ★3.5
Add and Optimize Imports, Generate Getters / Setters and Constructors
DSKWRK

Introduction to TypeScript I

- Functions, interfaces, classes and constructors.

- Common types – **Boolean, Number, String, Array, Tuple, Enum, Any, Void, Null, Undefined**.

- **--strictNullChecks** flag

- Type assertions: let length: number = (**<string>** data).length;

  let length: number = (data **as string**).length;

- **Duck typing** = structural similarity based typing

- Declaring variables – **let**, **const** and **var**. Scopes, variable capturing, Immediately Invoked Function Expressions (IIFE), closures and **let**.

Declaring Contracts using Interfaces I

```typescript
export interface User {
    id: number;
    firstName: string;
    lastName: string;
    email: string;
    password: string;
    contact?: Contact;        ←— Optional properties
    roles: Role[];
    getSalutation(): string;  ←— Required methods
}
```

Declaring Contracts using Interfaces II

```typescript
import { User } from './users';

export interface UserRepository {
    addUser(user: User): void;
    editUser(user: User): void;
    ...
    findAllUsers(): User[];
}

export class DemoUserRepository implements UserRepository {
    private users = new Map<number, User>();
    public addUser(user: User): void {
        if (this.findUserByEmail(user.email)) {
            throw `User with email ${user.email} exists.`;
        }
        user.id = this.getNextId();
        this.users.set(user.id, user);
    }
    ...
```

Declaring Contracts using Interfaces II

- Properties, optional properties and readonly properties:

```
interface UserRepository {
    readonly size: number;
    addUser: (user: User) => void;
}
```

- Function types:

```
interface RoleFinder {
    (user: User) : Role[];
}
```

- Array (indexable) types. Dictionary pattern:

```
interface EmailUserMap {
    [key: string]: User;
}
```

Class Types

```typescript
export class Customer implements User {
  public id: number; // set automatically by repository
  constructor(public firstName: string,
      public lastName: string,
      public email: string,
      public password: string,
      public contacts?: Contact,
      public roles: Array<Role> = [ Role.CUSTOMER ]) {
  }
  public get salutation() {
      return `${this.firstName} ${this.lastName}
             in role ${Role[this.roles[0]]}`;
  }
}
```

Default value

Extension of Interfaces. Hybrid Types.

```typescript
export interface Person {
    id: number;
    firstName: string;
    lastName: string;
    email: string;
    contact?: Contact;
}

export interface User extends Person{
    password: string;
    roles: Role[];
    getSalutation(): string;
}
```

Classes

- Constructors, constructor arguments as properties

- Public/private/protected properties

- Get and set accessors

```
export interface User extends Person{

    password: string;

    roles: Role[];

    readonly salutation: string;

} ...
public get salutation() {

    return `${this.firstName} ${this.lastName}`;

}
```

- Static and instance sides of a class. Abstract classes

Functions and Function Types

- Optional, default and **rest (…)** parameters

```typescript
export class Person {
    public restNames: string[];
    constructor(public firstName: string, ...restNames: string[]) {
        this.restNames = restNames;
    }
    public get salutation() {
        let salutation = this.firstName;
        for (let name of this.restNames) {
            salutation += ' ' + name;
        }
        return salutation;
    }
}
console.log(new Person('Ivan', 'Donchev', 'Petrov').salutation);
```

Function Lambdas (=>) and Use of this

```
export class LoginComponent {
    constructor(private jqElem: string, private loginC: LoginController){
        const keyboardEventHandler = (event: JQueryKeyEventObject) => {
            if (event.keyCode === 13) {
                loginEventHandler();
            }
        };
        const loginEventHandler = () => {
            this.loginC.login(usernameInput.val(), passwordInput.val())
                .then(() => {
                    this.showCurrentUser();
                }).catch(err => {
                    this.showError(err);
                });
            return false;
        };
    ...
```

Type Guards & Method Overloading

```typescript
export class DemoLoginController implements LoginController {
    public login(email: string, password: string): Promise<User>;
    public login(user: User): Promise<User>;
    public login(principal: User | string, credentials?: string)
            : Promise<User> {
        let email: string, password: string;
        if (typeof principal === 'string') {
            email = principal;
            password = credentials;
        } else {
            email = principal.email;
            password = principal.password;
        }
        let promise = new Promise<User>( (resolve, reject) => { …});
        return promise;
    }
}
```

Using Enums

- Defining enumeration:

```
enum Role {

    ADMIN = 1, CUSTOMER

}
```

- In generated code an enum is compiled into an object that stores both forward (name -> value) and reverse (value -> name) mappings.

- Getting enumerated name by value:

```
public get salutation() {

    return `${this.name} in role ${Role[this.roles[0]]}`;

}
```

JavaScript Module Systems – ES6

```
// lib/math.js
export function sum (x, y) { return x + y }
export var pi = 3.141593

// someApp.js
import * as math from "lib/math"
console.log("2π = " + math.sum(math.pi, math.pi))

// otherApp.js
import { sum, pi } from "lib/math"
console.log("2π = " + sum(pi, pi))
```

Modules in TypeScript

- **Namespaces** and **modules** – former internal and external modules – prefer **namespace X { … }** instead module X { … }

- ES6 modules (preferred) – using **export** and **import**:

```typescript
export interface Person {...}
export interface User extends Person{...}
export interface Contact {...}
export enum Role { ADMIN, CUSTOMER }
export class Customer implements User {...}
import {User} from './users';
import {resolvePromiseAfterTimeout} from './utilities';
import $ from 'jquery';
import * as repository from './user-repository';//default import
import './my-module.js'; //import a module for side-effects only
let module = await import('/modules/my-module.js'); // dynamic
```

Interoperability with External JS Libraries

- Ambient type declarations and ambient modules (***.d.ts**) – **typings**, **@types** – Example **node.d.ts** (simplified):

```typescript
declare module "url" {
    export interface Url {
        protocol?: string;
        hostname?: string;
        pathname?: string;
    }
    export function parse(urlStr: string, parseQueryString?,
 slashesDenoteHost?): Url;
}
/// <reference path="node.d.ts"/>
import * as URL from "url";
let myUrl = URL.parse("https://github.com/iproduct");
```

Generic Type Parameters I

- Writing generic functions, interfaces and classes – Examples:

```
interface Repository {
    findById<T>(id: number): T;
    findAll<T>(): Array<T>;
}

//OR

interface Repository<T> {
    findById: (id: number) => T;
    findAll(): Array<T>;
}
```

Generic Type Parameters II

```typescript
export class RepositoryImpl<T> implements Repository<T> {
    private data = new Map<number, T>();
    public findById(id: number): T {
        return this.data.get(id);
    }
    public findAll(): T[] {
        let results: T[] = [];
        this.data.forEach(item => results.push(item));
        return results;
    }
}
```

- Bounded generics
- Generic constructors

Advanced TypeScript Topics

- Advanced types

- Type Guards and Differentiating Types

- Type Aliases

- Symbols

- Declaration merging
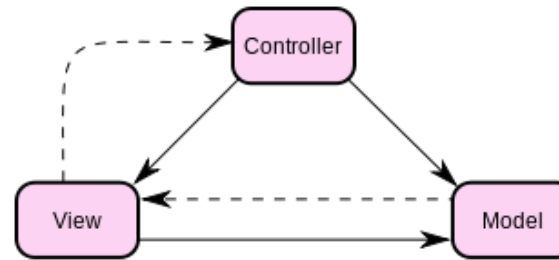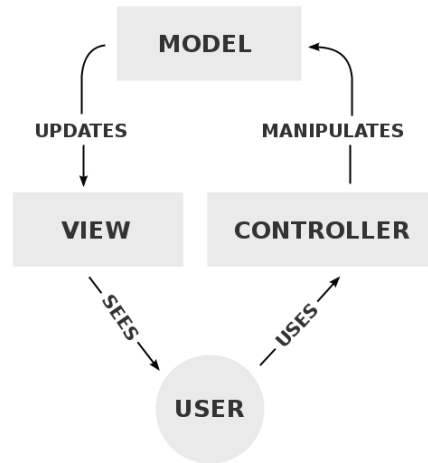
- Decorators

- Using mixins in TypeScript

Exercise: Users Login Demo

https://github.com/iproduct/fullstack-typescript-react/tree/master/05-ts-intro

MVC Comes in Different Flavors

- MVC

- MVVM

- MVP

33
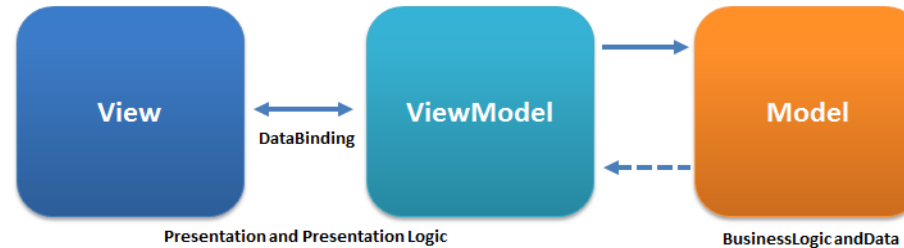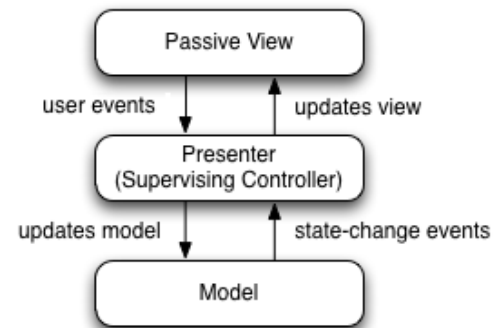
Using JavaScript Libraries in TypeScript: JQuery I

```typescript
import { LoginController } from './controller';
import '../node_modules/jquery/dist/jquery.js';

export class LoginComponent {
  private messagesElement: JQuery;
  constructor(private jqElementSelector: string, private loginController: LoginController) {

    const keyboardEventHandler = (event: JQuery.Event) => {
      if (event.keyCode === 13) {
        loginEventHandler();
      }
    };

    const loginEventHandler: any = () => {
      this.loginController.login(usernameInputElem.val() as string, passwordInputElem.val() as string)
      .then(() => {
        this.showCurrentUser();
      }).catch(err => {
        this.showError(err);
      });
      return false;
    };
```

(- continues -)

Using JavaScript Libraries in TypeScript: JQuery II

```typescript
const formElem = $("<form class='form-inline' role='form'>").addClass('form-inline');
const usernameInputElem: JQuery<HTMLElement> =
    $("<input id='username' type='email' placeholder='email'>")
    .addClass('form-control').bind('keypress', keyboardEventHandler);
const passwordInputElem: JQuery<HTMLElement> =
    $("<input id='password' type='password' ', keyboardEventHandler);
const loginButtonElem: JQuery<HTMLElement> =
    $('<button>Login</buttton>').addClass('btn btn-primary')
    .click(loginEventHandler);

// build the login form
formElem.append(usernameInputElem);
formElem.append(passwordInputElem);
formElem.append(loginButtonElem);
this.messagesElement = $('<div id="message" class="well well-lg">');
$(jqElementSelector).append(formElem).append(this.messagesElement);

this.showCurrentUser();
}

public showCurrentUser(): void {
    const user = this.loginController.getCurrentUser();
    this.messagesElement.html(user ? `Welcome ${user.salutation}.` : `No user is logged in.`);
}
```

Webpack Builder & Bundler Tool

Creating New Project: NPM + WebPack + React

mkdir my-project
cd my-project
npm init
npm install --save-dev webpack webpack-cli webpack-dev-server
touch index.html src/index.js webpack.config.js
npm install --save react react-dom
npm install --save-dev @types/react @types/react-dom
npm install --save-dev typescript ts-loader source-map-loader
// OR npm install typescript awesome-typescript-loader --save-dev
npm install css-loader style-loader css-to-string-loader --save-dev
npm install file-loader url-loader html-loader clean-webpack-plugin --save-dev
npm install extract-text-webpack-plugin html-webpack-plugin --save-dev
npm i --save-dev eslint @typescript-eslint/parser @typescript-eslint/eslint-plugin eslint-plugin-react

In package.json:
"scripts": {
    "start": "webpack-dev-server --inline –hot --open",
    "watch": "webpack --watch",
    "build": "webpack -p"
},

Minimal tsconfig.json

```json
{
  "compilerOptions": {
    "outDir": "./dist/",
    "sourceMap": true,
    "noImplicitAny": true,
    "strictNullChecks": true,
    "module": "ESNext",
    "moduleResolution": "node",
    "target": "ES6",
    "jsx": "react",
    "allowSyntheticDefaultImports": true,
  },
  "include": [ "src/**/*" // *** The files TypeScript should type check *** ],
  "exclude": ["node_modules", "build"] // *** The files to not type check ***
}
```

- For more complete TS config see:
  https://www.sitepoint.com/react-with-typescript-best-practices/

React Component File: /src/components/Hello.tsx

```tsx
import * as React from 'react';

export interface HelloProps {
  compiler: string;
  framework: string;
}

// 'HelloProps' describes the shape of props. State is never set so we use the '{}' type.
export class Hello extends React.Component<HelloProps, {}> {
  render() {
    return (
      <h1>
        Hello from {this.props.compiler} and {this.props.framework}!
      </h1>
    );
  }
}
```

React Main File: /src/index.tsx

```tsx
import * as React from "react";
import * as ReactDOM from "react-dom";

import { Hello } from "./components/Hello";

ReactDOM.render(
    <Hello compiler="TypeScript" framework="React" />,
    document.getElementById("demo")
);
```

Simple webpack.config.js I

```javascript
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const { CleanWebpackPlugin } = require('clean-webpack-plugin');
module.exports = {
  mode: 'development',
  entry: {
    app: './src/index.tsx',
  },
  devtool: 'inline-source-map',
  devServer: {
    contentBase: './dist',
    compress: true,
    hot: true,
    port: 9000
  },
  plugins: [
    new CleanWebpackPlugin({ cleanStaleWebpackAssets: false }),
    new HtmlWebpackPlugin({
      template: path.join(__dirname, 'index.html'),
      title: 'Hello React',
    }),
  ],
```

Simple webpack.config.js II

```
output: {
    filename: '[name].bundle.js',
    path: path.resolve(__dirname, 'dist'),
},
resolve: {
    extensions: [".wasm", ".ts", ".tsx", ".mjs", ".js", ".json"],
},

module: {
    rules: [
        {
            test: /\.ts(x?)$/,
            exclude: /node_modules/,
            use: [
                {
                    loader: "awesome-typescript-loader"
                }
            ]
        }]
},
```

Simple webpack.config.js III

```javascript
// optimization
optimization: {
    splitChunks: {
        cacheGroups: {
            default: false,
            vendors: false,
            // vendor chunk
            vendor: {
                // sync + async chunks
                chunks: 'all',
                // import file path containing node_modules
                test: /node_modules/
            }
        }
    }
};
```

Webpack Loaders and Plugins

- Loaders are transformations (functions running in node.js) that are applied on a resource file of your app

- You can use loaders to to load ES6/7 or TypeScript

- Loaders can be chained in a pipeline to the resource. The final loader is expected to return JavaScript

- Loaders can be synchronous or asynchronous

- Loaders accept query parameters – loader configuration

- Loaders can be bound to extensions / RegExps

- Loaders can be published / installed through npm

- Plugins – more universal than loaders, provide more features

WebPack Loaders

- ts-loader, awesome-typescript-loader -TypeScript => ES 5 or 6
- babel-loader - turns ES6 code into vanilla ES5 using Babel
- file-loader - emits the file into the output folder and returns the url
- url-loader - like file loader, but returns Data Url if file size <= limit
- extract-loader - prepares HTML and CSS modules to be extracted into separate files (alt. to ExtractTextWebpackPlugin)
- html-loader - exports HTML as string, requiring static resources
- style-loader - adds exports of a module as style to DOM
- css-loader - loads css file resolving imports and returns css code
- sass-loader - loads and compiles a SASS/SCSS file
- postcss-loader - loads and transforms a CSS file using PostCSS
- raw-loader - lets you import files as a string

WebPack Main Plugins

- ExtractTextWebpackPlugin - extracts CSS from your bundles into a separate file (e.g. app.bundle.css) ➔ mini-css-extract-plugin

- CompressionWebpackPlugin - prepare compressed versions of assets to serve them with Content-Encoding

- I18nWebpackPlugin - adds i18n support to your bundles

- HtmlWebpackPlugin - simplifies creation of HTML files (index.html) to serve your bundles

- ProvidePlugin - automatically loads modules, whenever used

- UglifyJsPlugin – tree transformer and compressor which reduces the code size by applying various optimizations

- ~~CommonsChunkPlugin~~ - generates chunks of common modules shared between entry points and splits them to separate bundles ➔ SplitChunksPlugin

Resources

- TypeScript Quick start – http://www.typescriptlang.org/docs/tutorial.html

- TypeScript Handbook – http://www.typescriptlang.org/docs/handbook/basic-types.html

- Official Webpack repo @ GiytHub – https://github.com/webpack/webpack

- Webpack: An Introduction (Angular website): https://v5.angular.io/guide/webpack

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

http://robolearn.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD