**TypeScript**

**let age: number;** число // age = 50;  
**let name: string;** строка // name = 'Max';  
**let toggle: boolean;** булиан // toggle = true;  
**let empty: null;** налл // empty = null;  
**let notInitialize: undefined;** андефайнед // notInitialize = undefined;  
**let callback = (a: number) => number;** передача в обратную функцию // callback = (a) => { return 100 + a };

**let anything: any;** любой тип // anything = -20;  
// anything = 'Text';  
// anything = {};

**let person = [string, number];**  кортеж(**Tuple**) неизм. массив // person = ['Max', 21];

|  |  |
| --- | --- |
| **enum Status { LOADING, READY};** | инам (Enum) // enum Status { LOADING, READY};  const loading = {status: Status.LOADING; } if (loading.status === Status.LOADING) {  console.log('Страница загружается'); } else {console.log('Страница загружена'); } |

**enum Status { LOADING, READY};**

**let union: string | number;** несколько типов (**Union**)// union = 'Text';

union = 10;

**let some:unknown**; (**Unknown)** тоже самое, что и any, только когда мы постараемся его сохранить в другую переменную мы получим ошибку.  
some = 10;  
let num: number;  
if (typeof some === 'number') {num = some; }

**let literal: 'first' | 'second';** литерал (**Literal**) // literal = 'first'; или literal = 'second';

|  |  |
| --- | --- |
| **function returnResult (num: number): number {**  **return num; }** | // указ. тип, который вернётся из функции (**Return**) |

|  |  |
| --- | --- |
| **function print (): void {**  **console.log('Print some text'); }** | (**Void**) // Это когда функция ничего не возвращает |

|  |  |
| --- | --- |
| function generateError (message: string, status: number): never {throw {message, status}; } generateError('An error', 500); | (Never) функция никогда не заканчивается и ничего не возвращает |

|  |  |
| --- | --- |
| **let callback: (num:number ) => void  callback = (num) => {  console.log(num); }** | Описывает функцию как тип. Это особенно актуально для callback или когда мы просто пробрасываем функцию. |

# Установка TypeScript

Для начала нам нужно установить NodeJs если у вас его нет. <https://nodejs.org/en/>

Выбираем версию LTS - это означает, что она стабильная.

Теперь нам нужно установить глобально сам TypeScript. <https://www.npmjs.com/package/typescript>

**npm install -g typescript**

Документация по ts <https://www.typescriptlang.org/docs/>

Если вы используете VSCode желательно установить дополнения

* ESLint
* Prettier - Code formatter
* Path Intellisense

Это упростит разработку.

# Настройки компилятора

## Команда watch[​](https://textbook.edu.goit.global/typescript-v7ex6g/v1/docs/module3/docs/doc1#%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%B0-watch)

Данная команда следит за изменением файла и компилирует его

**tsc app.ts –w** или так **tsc app.ts –watch**

Каждый раз при сохранении файла файл будет компилироваться снова, если будет ошибка вы сразу увидите ее в консоли.

# Создание проекта

Чтобы компилировались вместе все файлы, то для этого запустим команду

**tsc –init**

В проекте появится файл **tsconfig.json** с кучей закомментированных настроек.

**tsc** скомпилируется все файлы с расширением ts.

**tsc -w** будут компилироваться все файлы, которые будут изменены.

## Настройки проекта

Некоторые поля в файле tsconfig.json.   
 "compilerOptions": {  
 */\* Basic Options \*/*  
 "target": "es6", */\* Specify ECMAScript target version: 'ES3' (default), 'ES5', 'ES2015', 'ES2016', 'ES2017','ES2018' or 'ESNEXT'. \*/*  
 "module": "commonjs", */\* Specify module code generation: 'none', 'commonjs', 'amd', 'system', 'umd', 'es2015', or 'ESNext'. \*/*  
 "lib": [  
 "dom",  
 "es6",  
 "dom.iterable",  
 "scripthost"  
 ], */\* Specify library files to be included in the compilation. \*/*  
 "outDir": "./dist", */\* Redirect output structure to the directory. \*/*  
 "rootDir": "./src", */\* Specify the root directory of input files. Use to control the output directory structure with --outDir. \*/*  
 "removeComments": true, */\* Do not emit comments to output. \*/*  
 "noEmitOnError": true,  
 "strict": true, */\* Enable all strict type-checking options. \*/*  
 "sourceMap": true, */\* Generates corresponding '.map' 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. \*/*  
 "esModuleInterop": true */\* Enables emit interoperability between CommonJS and ES Modules via creation of namespace objects for all imports. Implies 'allowSyntheticDefaultImports'. \*/*  
 },  
 "exclude": [  
 "node\_modules",  
 "dist"  
 ]  
}

* **target** - говорит компилятору в какую версию js компилировать, если мы укажем es5 то в скомпилированном файле не будет **сonst** и **let** и других команд и новых функций которые были добавлены в es6, но зато у вас будет поддержка старых браузеров.
* **module** - как транслировать модули js при include файла.
* **lib** - какие библиотеки использовать, если убрать это поле вообще, то будет то, что я сейчас выписал в массив, но если ничего не задать, не будет даже console.log, по этому если вы пишете бек, вам как минимум нужно будет указать es6, чтобы были его базовые функции.
* **outDir** - куда сохранять скомпилированные файлы.
* **rootDir** - корневая папка проекта, у всех проектов есть папка src в которой и хранятся все файлы.
* **removeComments** - удалять ли комментарии в скомпилированном файле.
* **noEmitOnError** - компилировать ли файл если в нем ошибки.
* **strict** - добавлять ли файл js "use strict";.
* **sourceMap** - Создавать ли файлы source map.
* **noUnusedLocals** - показывать ошибку если есть неиспользуемые переменные.
* **noUnusedParameters** - показывать ошибку если есть неиспользуемые аргументы.
* **noImplicitReturns** - показывать ошибку если у функции не указано возвращаемое значение.
* **esModuleInterop** - это для Babel.
* **exclude** - по умолчанию если это значение не задавать то там уже есть "node\_modules", если хоти не включать какие-то значения то уже нужно задать "node\_modules", если хотим убрать какие-то типы файлов из всех папок например тесты, то можем указать \*\*/\*.spec.ts.

Вот еще дополнительная статья по настройкам <https://habr.com/ru/post/542234/>

**Debugging**

Так как мы в настройках tsconfig.json указали **sourceMap** давайте посмотрим как это работает.

Я переписал html файл

**Установка TypeScript для React**

**Создание проекта в VS Code**

Документация <https://create-react-app.dev/docs/adding-typescript/>

1. Открыть терминал (Alt + “~”, вкладка “Terminal”)
2. Добавить терминал PowerShell
3. Если нужно создать директорию, то

*npx create-react-app my-app --template typescript*

или

*yarn create react-app my-app --template typescript*

Если уже в директории, то

*npx create-react-app . --template typescript*

или

*yarn create react-app . --template typescript*

В VSCode может быть ошибка такого плана

*Argument for '--jsx' option must be : 'preserve', 'react-native', 'react'.*

Для ее решения нужно нажать Ctrl + Shift + P

Вписать Select TypeScript version

И выбрать последнюю версию.

**Типы компонентов**

В реакте компонент может быть функциональным и классовым.

**Создание функциональных компонентов**

*import* React *from* 'react';  
  
*const* App: React.FC = () => {  
 *return* (  
 <div className="App"></div>  
 );  
}  
  
*export* *default* App;

Вместо *React.FC* можно использовать *React.FunctionComponent*.

**Создание классового компонента.**

*import* React *from* 'react';  
  
*class* App *extends* React.Component {  
 render () {  
 *return* (  
 <div className="App"></div>  
 );  
 }  
}  
  
*export* *default* App;

Тут ничего необычного.

**Добавление Props и State**

*import* React *from* 'react';  
  
*interface* IItem {  
 id: string;  
 text: string;  
}  
  
*interface* IProps {  
 items: IItem[];  
};  
  
*const* TodoList: React.FC<IProps> = (props) => {  
 *return* (  
 <ul>  
 {props.items.map(todo => (  
 <li key={todo.id}>{todo.text}</li>  
 ))}  
 </ul>  
 );  
};  
  
*export* *default* TodoList;

Создан интерфейс IProps и прокинут как дженерик тип для пропсов.

*import* React *from* 'react';  
  
*interface* IState {  
 count: number  
}  
  
*class* ClickStatistic *extends* React.Component<{}, IState> {  
 handleClick () {  
 *this*.setState(({ count }) => {  
 *return* {  
 count: ++count,  
 };  
 });  
 }  
  
 render () {  
 *return* (  
 <button onClick={*this*.handleClick}>Click my!</button>  
 );  
 }  
}  
  
*export* *default* ClickStatistic;

Компонент принимает два аргумента в дженерик, первое это props, второе state. Туда передан интерфейс IState и теперь им можно пользоваться по коду.

**Когда мы используем** **refs**

*import* React, { useRef } *from* 'react';

Нам нужно всегда уточнять, что за тип там будет лежать.

*const* textInputRef = useRef<HTMLInputElement>(*null*);

Чтобы при получении значений из refs не было ошибки, то нужно поставить **!** знак.

*const* enteredText = textInputRef.current!.value;

**Когда мы используем** **useState**

*import* React, { useState } *from* 'react';  
Когда мы создаем useState нам так же нужно передавать типы.

*const* [todos, setTodos] = useState<{id: number, title: string}[]>([]);  
Если нам нужно старые значения, мы можем передавать функцию

setTodos((prevState) => {  
 *return* [  
 ...prevState,  
 { id: 1, title: 'Some text' }  
 ];  
});

И уже для prevState нам не нужно указывать никаких типов.

**Когда мы используем** **useEffect,** функция всегда возвращает void

*const* [state, setState] = useState<{name: string}[]>([]);  
  
*const* getPerson = *async* () => {  
 *const* result = *await* fetch('api/getPerson');  
  
 *const* person = *await* result.json();  
 setState(person);  
}  
  
useEffect(() => {  
 getPerson();  
}, []);

Как мы видим никакой дополнительной типизации не нужно.

**Когда мы используем** **событие** (в примере onChange)

<input onChange={changeHandler} />

Описываем функцию changeHandler

*function* changeHandler (e: React.ChangeEvent<HTMLInputElement>) {  
 console.log(e.target.value);  
}

Описан тип React.ChangeEvent<HTMLInputElement>, чтобы указать, что это ChangeEvent и дженерик HTMLInputElement так как это input.

**Redux**

Документация <https://redux.js.org/recipes/usage-with-typescript>

В документации все очень подробно описано, как разработчики Redux видят его использования с TypeScript

**Если библиотека не работает в TS**

Очень часто бывает такое, что какая-то библиотека не работает в TS, в случае с React библиотека react-router-dom не работает.

Нужно установить на нее типы. На многие библиотеки приходится устанавливать типы отдельно.

yarn add -D @types/react-router-dom  
# or  
npm install --save-dev @types/react-router-dom

И теперь все будет работать. Это правило, все типы для библиотек начинаются с @types/ и имя библиотеки.