УДК 004.65:004.43
DOI: 10.31866/2617-796X.4.2.2021.247484

Ткаченко Костянтин,
кандидат економічних наук,
доцент кафедри інформаційних технологій та дизайну,
Державний університет інфраструктури та технологій,
Київ, Україна
tkachenko.kostyantyn@gmail.com
https://orcid.org/0000-0003-0549-3396

Мороз Олександр,
магістрант, кафедра інформаційних технологій та дизайну,
Державний університет інфраструктури та технологій,
Київ, Україна
kore984134@gmail.com
https://orcid.org/0000-0001-5265-9761

ДЕЯКІ АСПЕКТИ СУВОРОЇ ТИПІЗАЦІЇ ЗА ДОПОМОГОЮ TYPESCRIPT ПІД ЧАС РОЗРОБКИ ВЕБЗАСТОСУНКІВ

Метою статті є дослідження, аналіз і розгляд загальних проблем, які розв'язує технологія TypeScript для суворої типізації під час розробки вебзастосунків.

Методами дослідження є методи семантичного аналізу основних понять цієї предметної сфери (розробка вебзастосунків засобами JavaScript та TypeScript). У статті розглянуто результати аналізу використання технології TypeScript у порівнянні з JavaScript.

Новизною проведеного дослідження є аналіз сучасної технології TypeScript і розв'язання за її допомогою проблем суворої типізації під час розробки вебзастосунків.

Висновки. У процесі дослідження технології TypeScript схарактеризовано її сутність, мету створення, описано переваги та недоліки. Визначено ситуації доцільності її використання.

Ключові слова: JavaScript; TypeScript; ECMAScript; статична типізація; динамічна типізація; транспілятор; компілятор; вебзастосунок.

Вступ. Мова JavaScript (Введение в JavaScript) створювалася для простих скриптів в інтернеті, але під час використання цієї мови на великих проєктах вебзастосунків почали з'являтися проблеми.

На допомогу розробникам прийшов TypeScript (TypeScript is JavaScript with syntax for types). Він усуває деякі недоліки JavaScript, не вимагаючи особливих зусиль від розробників для свого вивчення. Отже, актуальним є дослідження таких проблем, як:

– сутність TypeScript;
– організація та функції TypeScript;
– переваги та недоліки TypeScript;
– проблеми та перспективи використання TypeScript.

Тому актуальність проблем, пов'язаних із розробкою сучасних вебзастосунків, які підтримуються суворою типізацією, не викликає сумнівів.

Під час розробки таких вебзастосунків важливим є питання максимально зручної та комфортної роботи розробника через використання інструментарію TypeScript і користувача у віртуальному середовищі без появи для останнього можливих негативних наслідків.

Результати дослідження. У 2012 р. компанія «Microsoft» випустила першу версію TypeScript. А. Хейлсберг – один з авторів TypeScript – працював над створенням Pascal, Delphi і C#. Поява C# викликала в багатьох професійних програмістів інтерес до цієї мови.

Але в Microsoft в той час не було цілей, спрямованих на кросплатформеність своїх рішень і відкриття вихідних кодів. C# є умовно кросплатформеною і закритою мовою. На сьогодні Microsoft активно відкриває вихідні коди своїх проєктів і працює над багатоплатформеністю.

Поява Node.js (Anthony) і Electron (Прияцелюк, 2018) змусили Microsoft, залишивши собі проблеми з багатоплатформеністю, випустити компілятор у відкритий доступ. Отож створення TypeScript обумовлено тим, що:

– JavaScript не є модульною мовою (у той час ES2015 Modules лише були в планах, а «require» – задум Node.js, що не є стандартом);

– JavaScript поводиться іноді непередбачувано, що визначається динамічною типізацією;

– є потреба в методах і технологіях, які сприяють оптимізації процесів суворої типізації під час розробки вебзастосунків та ефективному розв'язанню пов'язаних з цим проблем.

TypeScript – компільована надмножина JavaScript, що приносить опціональну статичну типізацію, у тому числі й сувору, і деякі можливості сучасних стандартів ECMAScript (Myzgin, 2016).

Статична типізація відрізняється від динамічної тим, що в першому випадку перевірка типів здійснюється під час компіляції, а в другому – під час виконання інструкцій щодо проєктування вебзастосунку.

Опціональна типізація. Усе дуже просто: користувач (у цьому разі розробник вебзастосунку) використовує типи там, де він вважає за потрібне.

Розробник може писати весь код типізованим, взагалі використовуючи TypeScript лише як транспілятор і помічник-підказувач у своєму редакторі.

У цьому разі TypeScript з каркаса («коробки») скрипта вебзастосунку намагається визначити змінних, тип значень, тип результату виконання функцій та інших конструкцій і на основі цих визначень підказувати розробнику оптимальні та ефективні типи.

Слід пам'ятати про одну важливу особливість, на яку не завжди звертають увагу розробники: будь-який валідний JavaScript-код – це валідний TypeScript-код. Про це варто не забувати, коли перед розробником постає завдання написати щось на TypeScript, бо якщо він знає JavaScript, то він уже знає і TypeScript. Просто слід почитати про його особливості та запам'ятати додатковий синтаксис опису типів.

Розглянемо використання сучасних стандартів ECMAScript під час розробки вебзастосунків.

TypeScript потребує компіляції, тому було б доцільно додати можливість використання сучасних стандартів, і в процесі компіляції трансполювати їх у нижчу версію стандартів. Можемо навести як приклад всім відомий Babel (Babel is a JavaScript compiler), який робить усе те ж саме. Але порівнювати транспіляцію TypeScript слід з Bubble (Лаврова, 2021), а не з Babel, тому що він, на відміну від Babel, на виході надає розробнику читкий, налагоджений і підтримуваний програмний код.

Отже, навіть якщо буде втрачено вихідні файли, то завжди можна працювати зі згенерованим кодом.

TypeScript вміє трансполювати код в ES2015, ES5 і ES8 (Learn ES2015, JavaScript ES5, Kulkarni). Однак TypeScript не використовує властивості окремих компонентів коду, так само як і Babel. Це означає, якщо розробник напише

```
Object.assign({}, {})
```

і спробує скомпілювати цей код в ES5, то TypeScript відмовить йому в цьому, вказавши на помилку (рис. 1).

```
error TS2339: Property 'assign' does not exist on type 'ObjectConstructor'.
```

Рис. 1. Повідомлення від TypeScript про помилку

Однак TypeScript уміє знижувати рівень генераторів та ітераторів аж до ES3. Отож розробник може написати код, який використовує Async/Await і скомпілювати його в ES3, і це йому буде зручно зробити.

Структуру TypeScript представлено на рис. 2. TypeScript «зсередини» можна уявити у вигляді чотирьох шарів, кожен з яких виконує свою певну роль.

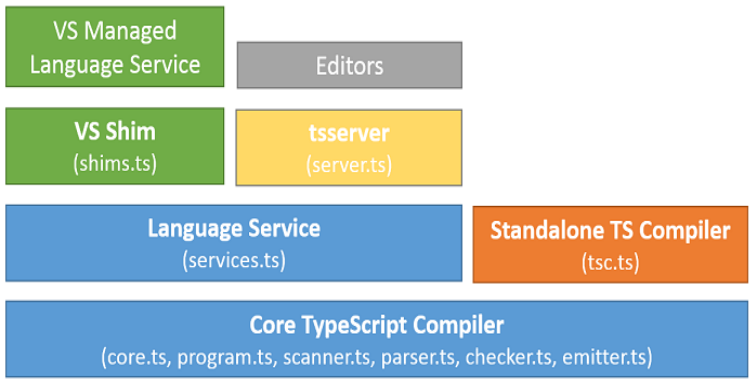

Рис. 2. Структура TypeScript «зсередини»

В основі TypeScript є ядро компілятора, яке містить:

– парсер;
– препроцесор (збирає контексти з «.ts»- і «.d.ts»-файлів);
– з'єднувач (пов'язує всі декларації);
– контролер типів (визначає і перевіряє тип кожної конструкції);
– емітер (генерує код «**.**js» і «**.d.**ts»).

Поверх ядра розташовуються «Автономний компілятор» і «Мовний сервіс». Перший потрібен для того, щоб працювати з необхідними API компілятора незалежно від поточного оточення. Наприклад, у Node.js – це читання і запис файлів.

Розглянемо тепер «Мовний сервіс». На відміну від інших мов, таких як CoffeeScript, Elm і Dart, TypeScript має інструментарій, що містить усе необхідне для того, щоб зробити TypeScript мовою, підтримку якої легко вбудувати в будь-який редактор. Підтримку витлумачуємо як:

– «підсвічування» мови програмування;
– базовий рефакторинг;
– автозавершення коду;
– перевірку типів «на льоту»;
– форматування коду;
– підказку сигнатур;
– деяку функціональність для відладчика;
– API для інкрементальної компіляції, щоб не навантажувати редактор розробника;
– інше, що робить життя розробника простіше.

І, нарешті, компілятор і мовний сервіс об'єднуються в «Автономний сервер», який спрощує спілкування з API тих шарів, що розташовані нижче, за допомогою JSON-протоколу, який також був розроблений Microsoft й успішно застосовується не тільки в TypeScript, а й в їхньому редакторі VS Code (Visual Studio Code).

Усе це дає можливість з мінімальними затратами вбудовувати TypeScript в будь-який редактор, адже все необхідне вже є в самому компіляторі. Це роблять простим так званим «смиканням» потрібного API. Тому TypeScript підтримується в багатьох редакторах.

TypeScript версії 2.3.0 дає змогу використовувати компілятор тільки для перевірки типів і/або збірки проєкту розробника. Жодних «* **.**ts» файлів, замість цього слід указати повний JSDoc для кожної функції та далі писати код на JavaScript.

Розробник також зможе використовувати «***.d.**ts»-файли для опису структур, використовуючи, наприклад, інтерфейси. Мовний сервіс буде аналізувати програмний код і згідно із зазначеними типами підказувати розробнику чи вказувати на їх недотримання.

TypeScript має свої переваги та недоліки.

До *переваг* можемо, зокрема, зарахувати такі:

– Підтримка багатьма популярними IDE:
– TypeScript for Visual Studio;
– Visual Studio Code;
– WebStorm;
– Sublime Text;

– Eclipse.

– TypeScript – суворо типізована (опціонально) і компільована в мову JavaScript. TypeScript є більш простим для освоєння Java- та C#-програмістами.

– TypeScript реалізує багато концепцій парадигми об'єктно-орієнтованого програмування, таких як успадкування, поліморфізм, інкапсуляція та модифікатори доступу. У TypeScript є класи, інтерфейси й абстрактні класи.

– Потенціал мови TypeScript сприяє більш швидкому, зручному та простому написанню програмного коду для складних комплексних рішень, які легше потім розвивати й тестувати, ніж на мові JavaScript.

– Програмний код на TypeScript буде читатися простіше через відсутність нагромаджень коду, характерних для Javascript.

– Компілятор суттєво зменшує ймовірність появи безглуздої помилки, наприклад такої, як пропущені коми, неправильно написані імена змінних.

– TypeScript є надмножиною JavaScript, тому будь-який програмний код, написаний мовою JavaScript, буде виконаний і в TypeScript. Це є однією з основних переваг перед конкурентами (такими як, наприклад, Dart від компанії Google (Differences Between TypeScript vs Dart), який є кардинально переробленим мовою з Javascript).

До *недоліків* TypeScript можемо, зокрема, зарахувати такі:

– Під час розробки програмного продукту розробник-програміст має справу з файлами * .ts, * .d.ts, * .map, * .js. У процесі розробки треба використовувати занадто багато додаткових файлів, що буває незручно, якщо розробляється невеликий проєкт.

– Не всі браузери підтримують налагодження TypeScript в консольному режимі без додаткових налаштувань.

– У TypeScript використовується багато нетривіальних класів. Щоб написати програмний код, використовуючи класи, слід постійно контролювати те, де перебуває та чи та властивість. Наприклад, окрім одного класу Event, є ще такі, як MouseEvent, TouchEvent, KeyboardEvent та ін.

– У TypeScript присутня неявна статична типізація. Завжди можна описати для змінної тип як any, що фактично відключить приведення до конкретного типу цієї змінної.

– TypeScript – це транспайлер (Kulkarni), тому розробнику слід мати на увазі, що в нього завжди під рукою має бути tsc.

– d.ts-декларації підтримуються DefinitelyTyped і часто або не відповідають поточній версії бібліотеки, або не враховують складних варіантів (наприклад, generic-функції, які повертають значення декількох типів).

Транспайлер, у тому числі й TypeScript, здійснює переклад між мовами програмування, які працюють приблизно на одному й тому ж рівні абстракції, у той час як традиційний компілятор переводить з більш високого рівня мови програмування на мову більш низького рівня. Інша мета транспайлера – автоматичний рефакторинг коду. Транспайлери:

– зберігають перекладений код якомога ближче до вихідного коду, щоб максимально спростити процес розробки та налагодження вихідного коду;

– змінюють структуру коду так, що перекладений код не буде схожий на вихідний;

– мають утиліти налагодження, які дають змогу зіставити транскомпільований вихідний код з оригінальним вихідним кодом.

Підтримуваний код в TypeScript. Якщо розробник опише код типами, то підтримувати цей код стане простіше (особливо з урахуванням функціональності інструментарію TypeScript).

На рис. 3 наведено приклад, де є функція, що приймає якийсь об'єкт, який містить декілька властивостей, причому деякі з них можуть бути необов'язковими. Ось такий вигляд мав би код, написаний на JavaScript разом з JSDoc:

```
/**
 * Відправляє листа
 * @param {Object} message
 * @param {String} message.from
 * @param {String[]}message.to
 * @param {String} message.subject
 * @param {String} [message.html]
 * @param {String} [message.text]
 * @param {Boolean} [message,autoClose=true]Закрити пул SMTP з'єднань.
 * @returns {Promise<Object>}
 *
 */

function sendEmail(message) {
    return smtp.sendMail(message).then((res) => {
        if (message.autoClose) {
            smtp.close();
        }
        return res;
    });
}
```

Рис. 3. Приклад коду мовою JavaScript разом з JSDoc

Якщо під час опису функції не зазначено властивості об'єкта в документації чи у відповідних тестах, то розробникам доведеться витратити багато часу на те, щоб:

– прочитати код функції та зібрати всі властивості об'єкта;

– знайти приклад використання функції в документації, місцях виклику чи в тестах, якщо вони є і добре написані.

Проблеми, які присутні в наведеному на рис. 3 програмному коді:

– величезний JSDoc, який з часом застаріє, про що розробники (інші користувачі, які мають доступ до програмного коду) дізнаються лише тоді, коли буде прочитано модуль changelog (Keep a changelog) або щось буде працювати не так («зламається»);

– якщо розробник захоче написати функцію-обгортку, яка відправляє повідомлення на зазначені адреси, то треба буде копіювати JSDoc або створити «@ typedef»;

– необхідність вказати редактору на те, де має бути розміщений JSDoc чи «@typedef»;

– якщо не було передано інформацію про тих, хто є отримувачем повідомлень, або про властивості об'єктів повідомлення, які не задокументовані в JSDoc, то як бути в цьому випадку;

– якщо забули передати «message» під час виклику, то це призведе до відповідної помилки.

Час розробників можна заощадити, якщо використовувати TypeScript. На рис. 4 представлено програмний код мовою TypeScript, який раніше був написаний мовою JavaScript (рис. 3).

```
interface IMessageOptions {
    from: string;
    to: string[];
    subject: string;
    html?: string;
    text?: string;
    /**
     * Дозволяє закрити пул smtp-з'єднань. За замовчуванням: true.
     */
    autoClose?: boolean;
}

/**
 * Відправляємо листа
 */
function sendEmail(message: IMessageOptions): Promise<object> {
    return smtp.sendMail(message).then((res) => {
        if (message.autoClose) {
            smtp.close();
        }
        return res;
    });
}
```

Рис. 4. Приклад коду мовою TypeScript

Переглянувши код, представлений на рис. 4, окреслимо його переваги й недоліки. До переваг, зокрема, можемо зарахувати такі:

– можливість явно вказувати інтерфейс (опис) об'єкта, причому в будь-якому місці коду, бо його можна експортувати;

– відсутня можливість передати зайву властивість або забути передати одну з обов'язкових;

– явно вказується тип параметрів і значення, що повертається, не сподіваючись на підказку редактора;

– відсутня можливість не передати «message», тому що редактор відправить розробника виправляти визначені помилки.

Недоліком TypeScript є те, що розробник не може вказати значення за замовчуванням в інтерфейсі, як це зроблено в JSDoc.

Наприклад, використовується функція, що вміщується в п’ять рядків. Однак не слід забувати, що в реальних проєктах таких функцій небагато.

Усунення програмних помилок (багів, які відображені у відповідному баг-репорті) ще на етапі компіляції, зменшення часу розуміння того, що може приймати та чи та функція та інтерактивні підказки й зауваження в редакторі або IDE – це переваги використання TypeScript.

Під час виконання функції редактор сам підкаже розробнику прийняті параметри, їх тип і вкаже на те, що він робить не так. Усе це завдяки інструментам TypeScript.

Чистий код. Програмний код стає чистішим, якщо, наприклад, розробнику будуть надаватися (пропускатися) хоча б конструкції, які мають такий вигляд (рис. 5):

```
function getItemByIndex(arr, index) {
    if (!Array.isArray(arr)) {
        throw new TypeError('Параметр "arr" не є масивом');
    }

    // ...
}
```

Рис. 5. Приклад коду мовою TypeScript, який робить програмний код розробника чистішим

Ця перевірка розробнику просто не потрібна, тому що TypeScript не дасть скомпілювати код, де в цю функцію передається щось відмінне від масиву. Розробник може вказати тип елементів масиву і тип того, що має повернути функція.

Це допоможе розробнику розв’язати проблему неправильного використання функцій навіть до того, як йому захочеться написати відповідні тести з її використанням (рис. 6).

```
function getItemByIndex(arr: string[], index: number): string {
    // ...
    return "any string"
}
```

Рис. 6. Приклад коду тесту мовою TypeScript для перевірки правильності використання функції

Але це не аргумент для тих, хто пише коди для деякої бібліотеки, що використовуються як JavaScript-модулі поза TypeScript-оточенням. У такому разі вам усе одно доведеться писати такі перевірки, якщо ви хочете убезпечити користувача.

Рефакторинг без наслідків. Рефакторинг у великому проєкті – складний процес. Особливо тоді, коли в проєкті тести підготовлені або не для кожної функції, або так, що покривають не всю кодову базу чи не всі випадки використання.

Навіть трохи змінюючи поведінку функції, можна отримати ризик повернути з неї або передати / забути передати до неї не ті вхідні дані чи типи даних тощо. З TypeScript можна бути впевненими, що функція повертає той тип, що був вказаний, і приймає саме те, що було вказано.

Актуальна документація. Використовуючи TypeScript, можна не писати дуже повну документацію перед функцією, наприклад, використовуючи JSDoc.

Але коротко описати те, що робить відповідна функція, усе ж таки треба. Треба також описати властивості інтерфейсу чи параметри функції, якщо їхні імена не дають повного уявлення про своє призначення.

Сучасні можливості й стандарти. Багато можливостей TypeScript зараз є частиною специфікації ECMAScript. Уже зараз можна використовувати сучасні можливості JavaScript під час написання TypeScript-коду. При цьому вони будуть транспільовані в ES5 [9], якщо це необхідно.

Теоретично це може дати змогу позбавитися від Babel і тих проблем, що з'являться після трансполювання.

Допомога V8 (Ядро node.js, google chrome browser). У деяких ситуаціях TypeScript допомагає V8 оптимізувати програмний код розробника. JIT-компіляція (Nbondarchuk, 2021) у V8 (What is V8?) виробляє деоптимізацію конструкцій, якщо передані (у функцію) значення змінюються кілька разів поспіль.

Якщо розробник пише добре зібраний код, то передати щось відмінне від дозволеного значення просто неможливо, бо буде дотримуватися принцип єдності переданого типу, і деоптимізація не відбудеться. Особливо це стосується функцій, які приймають на вхід об'єкти.

TypeScript слід використовувати завжди тоді, коли проєкт буде розвиватися і його треба буде підтримувати та модифікувати.

Переклад уже написаних програмних кодів проєктів розробки вебзастосунків з JavaScript на TypeScript тісно пов'язаний з тим, що:

– у таких проєктах складно виділити час на рефакторинг і перехід на TypeScript;

– розробник отримує типізацію та можливість використання сучасних стандартів.

У тому разі, коли відомо, що проєкт має певні (часто дуже малі) межі кодової бази, то впровадження TypeScript надасть лише підтримку сучасних можливостей ECMAScript. Однак перейти з JavaScript на TypeScript можна завжди, а ось навпаки – доведеться вручну видаляти всі типи або використовувати згенеровані файли.

Висновки. У статті розглянуто технологію TypeScript, описано історію виникнення та структуру технології. Наведено переваги технології: підтримку багатьма IDE, сувору типізацію, парадигму об'єктно-орієнтованого програмування, прискорення розробки, читкість коду, зменшення шансу появи помилки. А також недоліки: багато додаткових файлів, підтримка консолі не у всіх браузерів, додаткова інформація, яку треба мати на увазі, неявна статична типізація, транспайлер. Розглянуто ситуації, в яких технологія буде корисна, а в яких її використання недоцільне.

СПИСОК ПОСИЛАНЬ

Введение в JavaScript. *MDN Web Docs Store*. [online] Доступно: <https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Introduction> [Дата обращения 30 сентября 2021].
Лаврова, И., 2021. *Разработка приложений на Bubble своими руками: инструкция по выживанию (часть 1)*. [online] Доступно: <https://vc.ru/dev/196734-razrabotka-prilozheniy-na-bubble-svoimi-rukami-instrukciya-po-vyzhivaniyu-chast-1> [Дата обращения 01 октября 2021].
Прияцелюк, Н., 2018. *Пишем настольное JS-приложение с Electron*. [online] 2 апреля 2018. Доступно: <https://tproger.ru/translations/desktop-js-app-with-electron/> [Дата обращения 02 октября 2021].
Транспайлер. *КартаСлов.Ру*. [online] Доступно: <https://kartaslov.ru/карта-знаний/Транспайлер> [Дата обращения 02 октября 2021].
Anthony Li. An Introduction to Node.js: Server Side JavaScript. *LaunchX LLC*, [online] 11 November 2020. Available at: <https://launchx.com/blog/an-introduction-to-node-js-server-side-javascript/> [Accessed 01 October 2021].
Babel is a JavaScript compiler. *Babel*. [online] Available at: <https://babeljs.io> [Accessed 01 October 2021].
Differences Between TypeScript vs Dart. *Javatpoint Services*. [online] Available at: <https://www.educbcom/typescript-vs-dart//> [Accessed 01 October 2021].
JavaScript ES5, 2009. *W3Schools*. [online] Available at: <https://www.w3schools.com/js/js_es5.asp> [Accessed 02 October 2021].
Keep a changelog. [online] Available at: <https://keepachangelog. com/en/1.0.0/> [Accessed 03 October 2021].
Kulkarni, K. ES3 ECMAScript (ES7 + ES8 + ES9 + ES10) New Features – Javascrip. *Blog Post*. [online] Available at: <https://www.cronj.com/blog/javascript-es7-es8-new-features/> [Accessed 02 October 2021].
Learn ES2015. *Babel.* [online] Available at: <https://babeljs.io/docs/en/learn/> [Accessed 02 October 2021].
Myzgin, A., 2016. *Полное руководство по ECMAScript*. [online] Доступно: <https://frontend-stuff.com/blog/ecmascript/> [Accessed 02 October 2021].
Nbondarchuk, 2021. Java HotSpot JIT компилятор – устройство, мониторинг и настройка (часть 1). *Habr*, [online] 7 января 2021. Доступно: <https://habr.com/ru/post/536288/> [Accessed 29 September 2021].
TypeScript is JavaScript with syntax for types. *TypeScript*. [online] Available at: <https://www.typescriptlang.org> [Accessed 01 October 2021].
Visual Studio Code. [online] Available at: <https://code. visualstudio.com> [Accessed 02 October 2021].
What is V8? *V8.* [online] Available at: <https://v8.dev> [Accessed 30 September 2021].

REFERENCES

Vvedenie v JavaScript. MDN Web Docs Store [Introduction to JavaScript. MDN Web Docs Store]. [online] Available at: <https://developer.mozilla. org/ru/docs/Web/JavaScript/Guide/Introduction> [Accessed 30 September 2021].

Lavrova, I., 2021. *Razrabotka prilozhenii na Bubble svoimi rukami: instruktciia po vyzhivaniiu* (chast 1) [DIY Bubble Application Development: Survival Guide (Part 1)]. [online] Available at: <https://vc.ru/dev/196734-razrabotka-prilozheniy-na-bubble-svoimi-rukami-instrukciya-po-vyzhivaniyu-chast-1> [Accessed 01 October 2021].
Priiatceliuk, N., 2018. *Pishem nastolnoe JS-prilozhenie s Electron* [Writing a desktop JS application with Electron], [online] 2 April 2018. Available at: <https://tproger.ru/translations/desktop-js-app-with-electron/> [Accessed 02 October 2021].
Transpailer [Transpiler]. *KartaSlov.Ru.* [online] Available at: <https://kartaslov.ru/карта-знаний/Транспайлер> [Accessed 02 October 2021].
Anthony Li. An Introduction to Node.js: Server Side JavaScript. *LaunchX LLC*, [online] 11 November 2020. Available at: <https://launchx.com/blog/an-introduction-to-node-js-server-side-javascript/> [Accessed 01 October 2021].
Babel is a JavaScript compiler. *Babel.* [online] Available at: <https://babeljs.io> [Accessed 01 October 2021].
Differences Between TypeScript vs Dart. *Javatpoint Services*. [online] Available at: <https://www.educbcom/typescript-vs-dart//> [Accessed 01 October 2021].
JavaScript ES5, 2009. *W3Schools.* [online] Available at: <https://www.w3schools.com/js/js_es5.asp> [Accessed 02 October 2021].
Keep a changelog. [online] Available at: <https://keepachangelog. com/en/1.0.0/> [Accessed 03 October 2021].
Kulkarni, K. ES3 ECMAScript (ES7 + ES8 + ES9 + ES10) New Features – Javascrip. *Blog Post.* [online] Available at: <https://www.cronj.com/blog/javascript-es7-es8-new-features/> [Accessed 02 October 2021].
Learn ES2015. *Babel.* [online] Available at: <https://babeljs.io/docs/en/learn/> [Accessed 02 October 2021].
Myzgin, A., 2016. *Polnoe rukovodstvo po ECMAScript* [The definitive guide to ECMAScript]. [online] Available at: <https://frontend-stuff.com/blog/ecmascript/> [Accessed 02 October 2021].
Nbondarchuk, 2021. Java HotSpot JIT kompiliator – ustroistvo, monitoring i nastroika (chast 1) [Java HotSpot JIT Compiler – Device, Monitoring, and Tuning (Part 1). *Habr,* [online] 7 January 2021. Available at: <https://habr.com/ru/post/536288/> [Accessed 29 September 2021].
TypeScript is JavaScript with syntax for types. *TypeScript*. [online] Available at: <https://www.typescriptlang.org> [Accessed 01 October 2021].
Visual Studio Code. [online] Available at: <https://code. visualstudio.com> [Accessed 02 October 2021].
What is V8? *V8.* [online] Available at: <https://v8.dev> [Accessed 30 September 2021].

UDC 004.65:004.43

Tkachenko Kostiantyn,
PhD in Economics, Associate Professor,
Department of Information Technologies and Design,
State University of Infrastructure and Technology,
Kyiv, Ukraine
tkachenko.kostyantyn@gmail.com
https://orcid.org/0000-0003-0549-3396

Moroz Oleksandr,
Master's student,
Department of Information Technologies and Design,
State University of Infrastructure and Technology,
Kyiv, Ukraine
kore984134@gmail.com
https://orcid.org/0000-0001-5265-9761

SOME ASPECTS OF STRICT TYPIG IN TYPESCRIPT WHILE WEB APPLICATION DEVELOPING

The purpose of the article is to research, analyze and consider the general problems that TypeScript technology for strict typing solves when developing web applications.

The research methodology consists of semantic analysis methods of the basic concepts of a given subject area (development of web applications using JavaScript and TypeScript). The article discusses the analysis results of the TypeScript technology's use in comparison with JavaScript.

The novelty of the research is the analysis of modern TypeScript technology and its solution to the problems of strict typing in the development of web applications.

The conclusion of the research of the TypeScript technology carried out in the article is to determine its essence, the purpose of creation, advantages and disadvantages, and determine the situations of the appropriateness of its use.

Keywords: JavaScript; TypeScript; ECMAScrip; static typing; dynamic typing; transpiler; compiler; web application.

УДК 004.65:004.43

Ткаченко Константин,
кандидат экономических наук,
доцент кафедры информационных технологий и дизайна,
Государственный университет инфраструктуры и технологий,
Киев, Украина
tkachenko.kostyantyn@gmail.com
https://orcid.org/0000-0003-0549-3396

Мороз Александр,
магистрант, кафедра информационных технологий и дизайна,
Государственный университет инфраструктуры и технологий,
Киев, Украина
kore984134@gmail.com
https://orcid.org/0000-0001-5265-9761

НЕКОТОРЫЕ АСПЕКТЫ СТРОГОЙ ТИПИЗАЦИИ С ПОМОЩЬЮ TYPESCRIPT ПРИ РАЗРАБОТКЕ ВЕБ-ПРИЛОЖЕНИЙ

Целью статьи является исследование, анализ и рассмотрение общих проблем, которые решает технология TypeScript для строгой типизации при разработке веб-приложений.

Методами исследования являются методы семантического анализа основных понятий данной предметной области (разработка веб-приложений средствами JavaScript и TypeScript). В статье рассмотрены результаты анализа использования технологии TypeScript по сравнению с JavaScript.

Новизной проведенного исследования является анализ современной технологии TypeScript и решение с ее помощью проблем строгой типизации при разработке веб-приложений.

Выводы. В процессе исследования технологии TypeScript охарактеризована ее сущность, цель создания, описаны преимущества и недостатки. Определены ситуации целесообразности ее использования.

Ключевые слова: JavaScript; TypeScript; ECMAScript; статическая типизация; динамическая типизация; транспилятор; компилятор; веб-приложение.

16.10.2021