

# ІНТЕГРАЦІЯ МОВ ПРОГРАМУВАННЯ З ФУНКЦІОНАЛЬНИМ ПІДХОДОМ ДО ВЕБРОЗРОБКИ: ПЕРЕВАГИ, ВИКЛИКИ ТА ПРАКТИЧНІ РЕКОМЕНДАЦІЇ

**ОЛЕКСАНДРА ІГНАТОВИЧ,**

*студентка 2 курсу 3 групи,  
Державний торговельно-економічний університет,  
м. Київ, Україна*

**КАРИНА ХОРОЛЬСЬКА,**

*PhD, старший викладач кафедри інженерії програмного  
забезпечення та кібербезпеки,  
Державний торговельно-економічний університет,  
м. Київ, Україна*

У сучасному світі веб-розробка стає все складнішою та вимагає більшого уваги до якості коду та продуктивності розробки. Функціональне програмування відкриває нові можливості для веб-розробників, пропонуючи альтернативні підходи до створення програмного забезпечення. Проте інтеграція функціонального підходу у веб-розробку не є безпроблемною, і вона потребує уважного аналізу та розуміння переваг та викликів цього підходу.

Функціональне програмування (ФП) – це парадигма програмування, яка ґрунтується на концепції функцій. Функції у ФП розглядаються як першокласні об'єкти, їх можна використовувати як аргументи інших функцій, повертати з функцій, а також присвоювати їм змінні.

Переваги ФП у веб-розробці

1. Декларативність: ФП дозволяє описувати логіку програми декларативно, тобто зосередитися на тому, що потрібно зробити, а не на тому, як це зробити. Наприклад, наступний код JavaScript декларативно описує функцію, яка робить квадрат числа: *function square(x) {return x \* x;}*

Цей код не описує, як саме обчислюється квадрат числа, він просто декларує, що функція *square* приймає число *x* як аргумент і повертає його квадрат [1].

2. Чистота коду: Функціональні мови програмування зазвичай мають лаконічний і читабельний код, що полегшує його розуміння та модифікацію. Наприклад, наступний код Haskell декларативно описує функцію, яка робить квадрат числа: *square x = x \* x*

Цей код ще більш лаконічний, ніж код JavaScript, і його ще легше читати та розуміти [1].

3. Відсутність побічних ефектів: Функції у ФП не мають побічних ефектів, що робить код більш передбачуваним і менш

схильним до помилок. Побічний ефект – це зміна стану програми, яка не є явним результатом виконання функції. Наприклад, наступний код JavaScript має побічний ефект, оскільки він змінює значення змінної  $x$ :

```
function increment(x) { x++; return x; }
```

Цей код може призвести до помилок, якщо функція `increment` використовується в декількох місцях коду, оскільки зміна значення змінної  $x$  може вплинути на інші частини програми [1].

4. Композиційність: Функції у ФП можна легко компонувати одна з одною, що робить код більш модульним та повторно використовуваним. Наприклад, наступний код Haskell декларативно описує функцію, яка робить квадрат числа і потім додає до нього 1:

```
addOneAndSquare x = square x + 1.
```

Цей код можна легко скомпонувати з іншими функціями, наприклад, з функцією, яка фільтрує список чисел:

```
filteredSquares = filter isEven (map addOneAndSquare numbers)
```

Цей код спочатку застосовує функцію `addOneAndSquare` до кожного числа в списку `numbers`, а потім фільтрує список, видаляючи з нього непарні числа [1].

5. Незмінність: Дані у ФП зазвичай є незмінними, що робить код більш стійким до помилок. Незмінність означає, що значення даних не може бути змінено після того, як воно було створено. Це робить код більш передбачуваним, оскільки розробники можуть бути впевнені, що значення даних не зміниться несподівано [1].

*Виклики ФП у веб-розробці*

1. Навчальна крива. Імперативне програмування – це парадигма програмування, яка ґрунтується на концепції команд. Команди імперативного програмування описують, як крок за кроком змінюється стан програми. Розробникам, які звикли до імперативного стилю програмування, може бути складно перейти на декларативний стиль ФП [2].

2. Відсутність підтримки браузерів: Деякі функції ФП не підтримуються браузерами безпосередньо. Для використання таких функцій у веб-розробці може знадобитися трансляція коду з функціональної мови програмування в JavaScript або використання бібліотек, які надають реалізацію цих структур даних в JavaScript [2].

3. Продуктивність: Деякі функціональні мови програмування можуть бути менш продуктивними, ніж імперативні мови. Це пов'язано з тим, що деякі операції у ФП можуть потребувати додаткових обчислень для забезпечення незмінності даних. Однак, оптимізація компіляторів та розвиток віртуальних машин для функціональних мов постійно покращують їхню продуктивність [2].

4. Стан та побічні ефекти: Деякі веб-додатки потребують управління станом та виконання операцій з побічними ефектами,

наприклад, взаємодія з API або робота з DOM-деревом. ФП пропонує рішення для роботи з станом та побічними ефектами, такі як монади та ефекти, але вони можуть бути складними для розуміння та використання, особливо для розробників, які не звикли до ФП [2].

Практичні рекомендації щодо інтеграції ФП у веб-розробку

1. Почніть з малого: Не намагайтеся одразу переписати весь свій код на функціональній мові. Почніть з невеликих проектів або компонентів, щоб ознайомитися з ФП. Ви можете використовувати ФП для окремих задач, наприклад, обробки даних або управління бізнес-логікою.

2. Використовуйте бібліотеки: Існує багато бібліотек, які полегшують використання ФП у веб-розробці. Наприклад, бібліотека Redux допомагає керувати станом додатків, а бібліотека Ramda надає набір функцій для роботи з даними у стилі ФП.

3. Використовуйте фреймворки з підтримкою ФП: Деякі фреймворки для веб-розробки, такі як Elm та Elmish, побудовані на принципах ФП. Ці фреймворки пропонують структуру коду, інструменти та бібліотеки, які спрощують розробку веб-додатків з використанням функціонального підходу.

4. Не бійтеся експериментувати: ФП пропонує багато можливостей для творчого підходу до розробки веб-додатків. Експериментуйте з різними функціональними бібліотеками та техніками, щоб знайти підхід, який найкраще відповідає вашим потребам.

5. Зважуйте переваги та недоліки: Оцінюйте, чи переваги ФП переважають недоліки для конкретного проекту. Іноді використання ФП може бути недоцільним, наприклад, для невеликих проектів з коротким терміном виконання [3–4].

Отже, функціональне програмування може стати цінним інструментом для веб-розробників, які прагнуть до створення чистого, декларативного, стійкого та модульного коду. Зростання популярності ФП, розвиток бібліотек та фреймворків робить його все більш привабливим варіантом для веб-розробки.

### Список використаних джерел

1. Graham P. The Benefits of Functional Programming. 2011.
2. Lipovača M. Learn You a Haskell for Great Good!. 2012. 399 p. URL: <https://learnyouahaskell.com/>
3. Iqbal, Kashif. (2024). Full Stack Web Development: Vision, Challenges and Future Scope. INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT. 08. 1-5. 10.55041/IJSREM30338.
4. Gogoi, Bonashri & Kakoti, Mriganko. (2024). Unlocking the Potential of Web 2.0 Tools for Enhanced E-Learning Experiences: A Comprehensive Approach.