SCI-CONF.COM.UA

INNOVATIVE DEVELOPMENT OF SCIENCE, TECHNOLOGY AND EDUCATION

PROCEEDINGS OF IV INTERNATIONAL SCIENTIFIC AND PRACTICAL CONFERENCE JANUARY 18-20, 2024

VANCOUVER 2024

INNOVATIVE DEVELOPMENT OF SCIENCE, TECHNOLOGY AND EDUCATION

Proceedings of IV International Scientific and Practical Conference

Vancouver, Canada

18-20 January 2024

Vancouver, Canada

2024

UDC 001.1

The 4th International scientific and practical conference "Innovative development of science, technology and education" (January 18-20, 2024) Perfect Publishing, Vancouver, Canada. 2024. 700 p.

ISBN 978-1-4879-3792-8

The recommended citation for this publication is:
Ivanov I. Analysis of the phaunistic composition of Ukraine // Innovative development of science, technology and education. Proceedings of the 4th International scientific and practical conference. Perfect Publishing. Vancouver, Canada. 2024. Pp. 21-27. URL: https://sci-conf.com.ua/iv-mizhnarodna-naukovo-praktichna-konferentsiya-innovative-development-of-science-technology-and-education-18-20-01-2024-vankuver-kanada-arhiv/.

Editor
Komarytskyy M.L.
Ph.D. in Economics, Associate Professor

Collection of scientific articles published is the scientific and practical publication, which contains scientific articles of students, graduate students, Candidates and Doctors of Sciences, research workers and practitioners from Europe, Ukraine and from neighbouring countries and beyond. The articles contain the study, reflecting the processes and changes in the structure of modern science. The collection of scientific articles is for students, postgraduate students, doctoral candidates, teachers, researchers, practitioners and people interested in the trends of modern science development.

e-mail: vancouver@sci-conf.com.ua

homepage: *https://sci-conf.com.ua/*

TABLE OF CONTENTS

AGRICULTURAL SCIENCES

MEDICAL SCIENCES

УДК 004.4: 004.43:004.415.2

СУЧАСНА КОНЦЕПЦІЯ РОЗРОБКИ ТА ПІДТРИМКИ ВЕБ ДОДАТКІВ

Гуда Антон Ігорович,
д.т.н., професор
Кришталь Сергій Валерійович
студент
Український державний університет науки і технологій
м. Дніпро, Україна

Анотація: Аналіз та пошук рішень для побудови сучасного гнучкого, продуктивного та легкого веб додатку з динамічним інтерфейсом.

Ключові слова: концепція розробки, elasticsearch, php, slim, framework, symfony twig, javascript, jquery, laravel eloquent, modern web application.

Важко уявити сьогодення без використання можливостей мережі інтернету. Переплетіння інтернету з життям майже кожної люди дуже тісне. Щорічно відкриваються безліч веб ресурсів на різну тематику як для навчання так і для роботи або розваг. Також с кожним роком виникає все більше та більше викликів як оптимізувати та зробити кращими уже розроблені веб додатки так і як створити продукт, що буде кращим, швидшим, зручнішим за продукт конкурентів. Всі ці бажання та цілі розробників товкають їх до розробки все новіших та все більш технологічних рішень у сфері розробки веб проектів, і це потрібно як для кінцевого користувача так і для самих розробників. Кожна модернізація, кожне нове використання технологій має свої наслідки як для розробників та власників проектів, так і для кінцевого користувача. Тому розглянемо використання сучасних технологічних рішень, що дозволяють підвищити продуктивність під час розробки та роботи веб додатка в цілому, знижуючи час оброки запитів навіть для складних завдань, пов'язаних з базами даних.

Додатковим не менш важливим елементом для будь-якого веб додатку є

його інтерфейс та адаптивність цього інтерфейсу під потрібні задачі чи екрани. Тому розглянемо сучасні можливості для створення адаптивного інтерфейсу веб додатку та розроблений підхід для взаємодії з елементами інтерфейсу, яка підвищує додатково гнучкість та адаптивність вашого додатку під різноманітні задачі.

Будь-який веб проект має декілька складових частин: front-end, back-end та сервісна частини (рис. 1) [1]. Для створення функціональної частини front-end частини додатку беззаперечним лідером є мова Java Script [2] тому її використання майже без альтернативне (рис. 3). Гнучку, ефективну та просту back-end частину можна створити використовуючи мову PHP [3], особливо з погляду на те, що вона є однією з найбільш популярних мов для back-end (рис. 3) та має багато різноманітних рішень та модулів для побудови сучасного адаптивного веб додатку [4].

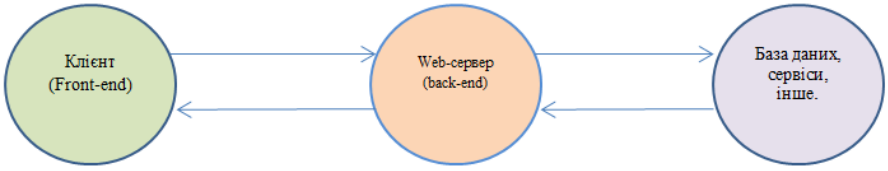


Рис. 1. Базова архітектура клієнт-серверного додатку.

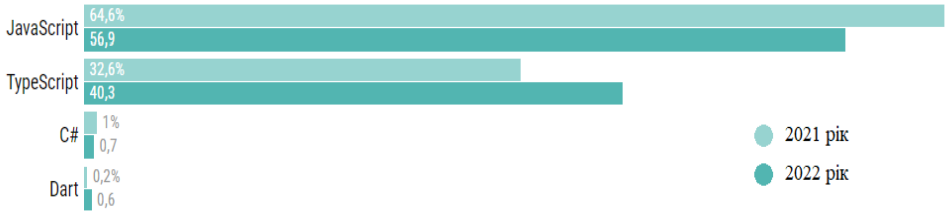


Рис. 2. Рейтинг мов програмування для розробки front-end частини.

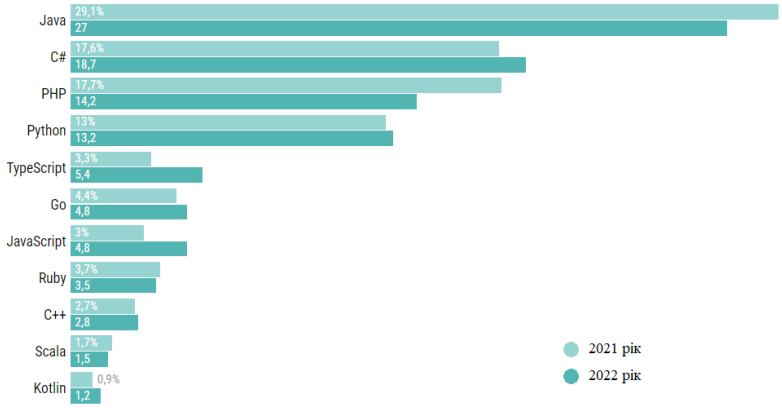


Рис. 3. Рейтинг мов програмування для розробки back-end частини.

Обраний підхід побудови веб додатку вказує, що наш тип додатоку буде багатосторінковим (MPA – multi page application) і він має достатньо багато переваг над одно сторінковим (SPA – single page application):

- швидке завантаження сторінок (на відміну від SPA, MPA додаток не потрібно завантажувати одразу весь);

- можливості більш гнучкого використання HTML шаблонів для побудови сторінок (можливість використання продуктивних шаблонізаторів на стороні back-end частини);

- більш коректна обробка сторінок проекту пошуковими ботами (завдяки меншому використанню Java Script коду для генерації сторінок проекту);

- більші можливості для оптимізації SEO проекту.

Також сьогодні не можливо уявити створення сучасного веб додатку без використання додаткових інструментів, які слугують для побудови готової публічної версії створеного додатку, для завантаження та контролю оновлень додаткових модулів, для контролю змін проекту і т.п. Тому важливо знайти баланс між завантаженістю програмним забезпечення та функціональністю, тому що, чим більше інструментальних залежностей проекту тим складніша його підтримка в майбутньому.

Після розробки декількох проектів різної складності, прийшов до висновків, що найбільш проста та ефективна конструкція додаткових інструментів для визначених частин проект складається з Composer [5] (використовується для back-end), NodeJS [6] (використовується для front-end та сервісної частин) та Git [7] (сервісна частина). Визначені інструменти покривають максимально можливу більшість потреб під час розробки та підтримки веб проектів.

Composer - це інструмент керування залежностями для PHP проектів та він є найбільш важливим в середовищі розробки PHP проектів. Такі PHP фреймворки, як Laravel, Slim, Symfony та CodeIgniter використовують Composer для керування бібліотеками та пакетами додаткових модулів. Також додатковою важливою функцією Composer є створення автоматичного

завантаження всіх PHP файлів та залежних пакетів модулів, що достатньо сильно розширює можливості розвитку та підтримки проекту, так як більшість публічних додаткових пакетів постійно розвиваються та отримують оновлення безпеки та функціоналу. З огляду на вище зазначені особливості Composer можна вважати фундаментом вашого back-end проекту.

Найбільш вдалим варіантом архітектури для побудови універсального веб додатку є MVC архітектура (Model View Controller) (рис 3). Основна логіка цієї архітектури полягає в тому, що всі зміни відбуваються на основі запитів, бідь-то між модулями back-end частини додатку або між front-end та back-end частинами.

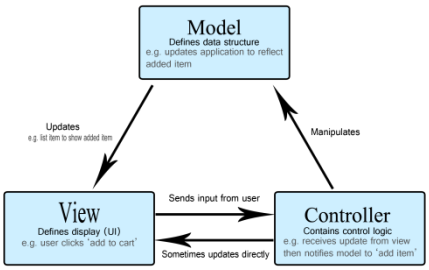


Рис. 3. Схема MVC архітектури.

Сучасний принцип розробки веб додатків потребує швидкої розробки та розвитку після запуску, а тому на сьогодні для досягнення цих цілей має сенс використання якогось готового рішення для побудови основного скелету вашого додатку. Для вирішення цієї проблеми використовують спеціальні додаткові програмні пакети Framework. Деякі з цих пакетів мають багато готових рішень для створення html сторінок, взаємодії с базами даних, файлами, кеш сервісами та інше. Найбільшою проблемою більшості PHP Framework пакетів є їх складність та унікальний для кожного принцип побудови додатку, що є проблемою, так як це підвищує поріг знань і тим самим підвищує собівартість розробки в цілому за рахунок найму співробітника більшої кваліфікації. Тому вивчаючи різноманітні Framework пакети було визначено найбільш простий, гнучкий, швидкий і достатньо функціональний

пакет – це Slim Framework [8].

Найбільшою перевагою Slim Framework є мінімалізм у його виконанні. Він має можливості побудови додатку на основі MVC архітектури (рис. 4) зі створення маршрутів для посилань на сторінки додатку, а також його основний код реалізований дотримуючись стандартів PHP-FIG PSR [9] та принципів SOLID [10].

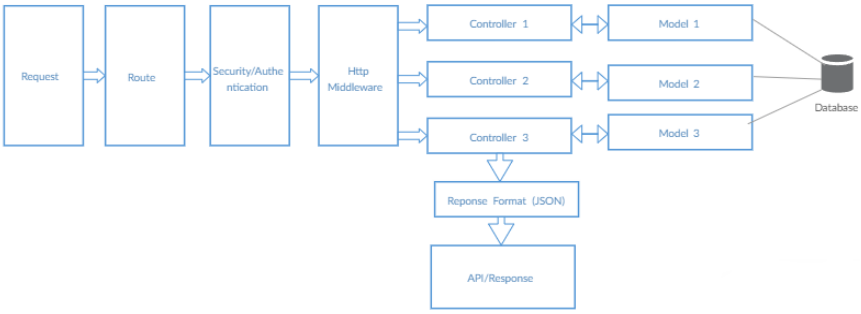


Рис. 4. Схема роботи програмного пакету Slim Framework.

Завдяки вище описаним властивостям Slim Framework ми отримуємо можливість використання будь-яких сумісних додаткових програмних пакетів, які можна включити в завантаження самого Slim Framework і використовувати їх будь-де в нашому проекті (рис. 5).

```php
<?php
//використання потрібних об'єктів фреймворку
use DI\Container;
use Slim\Factory\AppFactory;
use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;
//важливо! автоматичне завантаження всіх пакетів та нашого проекту
require 'vendor/autoload.php';
//створення контейнеру для зберігання додаткових модулів і доступу до них
$container = new Container();
//додавання додаткових модулів до контейнеру
$container->set('MyService', function() { return new MyService(); });
//додавання глобального доступу до створеного контейнеру
AppFactory::setContainer($container);
//ініціалізація основного файлу фреймворку
$app = AppFactory::create();
//додавання стандартного обробника шляхів зі запитів
//він розбирає запит на параметри згідно заданого патерну
$app->addRoutingMiddleware();
//додавання шляху та контролеру його обробки
//сервер відповість сторінкою з вмістом Hello World
$app->get('/hello',
function (ServerRequestInterface $request, ResponseInterface $response) {
    //використання створеного сервісу
    $myService = $this->get('MyService');
    //запис тексту у відповідь на запит
    $response->getBody()->write('Hello World');
    return $response;
});
//якщо шлях не був знайдений то виникне помилка
$app->map(['GET', 'POST', 'PUT', 'DELETE', 'PATCH'], '{routes:.+}',
function (ServerRequestInterface $request, ResponseInterface $response, array $args) {
    die('Помилка');
});
// обробка присланого в запиті контенту, наприклад якщо там код JSON
$app->addBodyParsingMiddleware();
//додавання об'єкту middleware для відстеження помилок коду
// цей об`єкт буде виконуватися першим
$app->addErrorMiddleware(false, true, true);
//виконання програми
$app->run();
```

Рис. 5. Приклад використання Slim Framework.

Виходячи прикладу використання Slim Framework (рис. 6), роль Controller частини MVC архітектури виконують класи або функції, які відповідають за формування відповіді за допомогою об'єднаного використання потрібних

модулів.

Для побудови View частини MVC архітектури використаємо додатковий потужний пакет Symfony Twig [11]. Особливими перевагами Symfony Twig є:

- простий цільовий синтаксис (рис. 6);

- швидкість і гнучкість (завдяки компіляції шаблонів в PHP код та відкритій до доповнень архітектури);

- функціональність (базовими можливості шаблонізатору можна порівнювати з мовою програмування).

Рис. 6. Приклад синтаксису шаблону Symfony Twig.

Останньою частиною MVC архітектури є Model частина. Зазвичай вона виконує роль джерела інформації, яке отримує або форматує збережені дані та передає їх до Controller. Більшість розробників веб проектів, в залежності від потреб проекту, як джерело інформації використовують бази даних (файлові або реляційні). Достатньо універсальним і потужним засобом для вирішення цієї задачі є використання пакету Laravel Eloquent [12].

Найбільшою перевагою Laravel Eloquent є об'єктно-орієнтований підхід в формуванні запитів до бази даних і відповідей з неї (рис. 7). Також цей модуль може використовувати декілька видів баз даних не порушуючи загальної структури моделей (моделями називаються таблиці в базі даних) [13]. Великі можливості, гнучкість та об'єктність створюють головний недолік цього модуля – це низька швидкість виконання запитів до бази даних. Цей недолік створює також проблему підвищення часу очікування відповіді від серверу зі сторони клієнтського інтерфейсу, особливо під час виконання тяжких запитів з багатьма залежними таблицями.

```php
<?php
namespace App\Model;

use Illuminate\Database\Eloquent\Model;

class Comment extends Model {
   //задаємо назву таблиці в базі даних
   protected $table = 'user_comments';
}

//отримуємо записи таблиці user_comments з умовою за допомогою моделі Comment
//як результат буде отриманий масив з об'єктами, кожен об'єкт – це модель Comment
$comments = \App\Model\Comment::query()->where('name','=','John')->get();
```

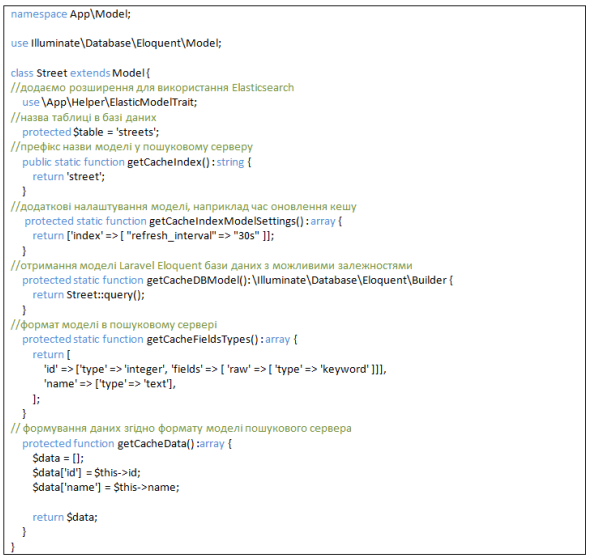
Рис. 7. Приклад моделі Laravel Eloquent та її використання.

Для вирішення проблеми з затримкою отримання інформації з бази даних було знайдене і розроблене рішення у вигляді комбінованого використання Laravel Eloquent та пошукового сервера Elasticsearch [14].

Elasticsearch – це розподілена система пошуку і аналітики, найпотужніший і найпопулярніший пошуковий рушій для сайтів і додатків будь-якого призначення. Він є одночасно сховищем даних і інструментом пошуку, включаючи все, що потрібно для точної настройки релевантності та всебічної аналітики. Основна ідея виправлення ситуації з затримками під час роботи з базою даних, полягає у створенні моделей зі своєю структурою для пошукового серверу Elasticsearch. Додаткові можливості додаються за допомогою окремого класу або розширенням до наявної моделі Laravel Eloquent. Розширення для класу має методи, які оперують з додатковим офіційним програмним пакетом для взаємодії з сервером Elasticsearch (рис. 8) і як результат кожна модель отримує додатковий функціонал у вигляді побудови Elasticsearch моделі та отримання потрібної інформації з бази у відповідному форматі, яку додаємо одразу до Elasticsearch (рис. 9).

```php
trait ElasticModelTrait {
   abstract public static function getCacheIndex(): string;
   abstract protected static function getCacheFieldsTypes() : array;
   abstract protected function getCacheData(): array;
   abstract protected function getCacheDBModel();
   //оновлення бази даних та elasticsearch моделі
   public function saveAndBuildCache(array $options = []) {
     //зберігаємо в базу даних
      $result = $this->saveOrFail($options);
     //отримуємо наявні дані моделі у потрібному форматі
      $data = $this->getCacheData();
     //зберігаємо їх до elasticsearch
      $elastic = App::getContainer()->get('elastic');
      $elastic->store(self::getCacheIndex(), $data);
      return $result;
   }
   //отримуємо модель для створення запитів до Elasticsearch
   public static function model() : ElasticIndex {
      return App::getContainer()->get('elastic')->getModel(self::getCacheIndex());
   }
  //зберігання всіх даних з таблиці моделі бази даних
   public static function buildCache(): bool {
     //отримуємо менеджер для взаємодії з сервером Elasticsearch
      $elastic = App::getContainer()->get('elastic');
     //отримуємо модель таблиці Laravel Eloquent
      $model = self::getCacheDBModel();
     //отримання всіх записів таблиці
      $items = $model->get();
     //отримання даних для кожного запису окремо у форматі для Elasticsearch
      foreach ($items as $item) {
         $itemData = $item->getCacheData();
         $elastic->store(self::getCacheIndex(), $data);
      }
      return true;
   }
}
```

Рис. 8. Приклад побудови розширення для моделей Laravel Eloquent.



```
namespace App\Model;

use Illuminate\Database\Eloquent\Model;

class Street extends Model{
//додаємо розширення для використання Elasticsearch
   use \App\Helper\ElasticModelTrait;
//назва таблиці в базі даних
   protected $table = 'streets';
//префікс назви моделі у пошуковому серверу
   public static function getCacheIndex(): string {
      return 'street';
   }
//додаткові налаштування моделі, наприклад час оновлення кешу
    protected static function getCacheIndexModelSettings(): array {
      return ['index' => [ "refresh_interval" => "30s" ]];
   }
//отримання моделі Laravel Eloquent бази даних з можливими залежностями
   protected static function getCacheDBModel(): \Illuminate\Database\Eloquent\Builder {
      return Street::query();
   }
//формат моделі в пошуковому сервері
   protected static function getCacheFieldsTypes(): array {
      return [
         'id' => ['type' => 'integer', 'fields' => [ 'raw' => [ 'type' => 'keyword' ]]],
         'name' => ['type' => 'text'],
      ];
   }
// формування даних згідно формату моделі пошукового сервера
   protected function getCacheData():array {
      $data = [];
      $data['id'] = $this->id;
      $data['name'] = $this->name;

      return $data;
   }
}
```

Рис. 9. Приклад кооперації моделей Laravel Eloquent та Elasticsearch.

Перевага цього методу реалізації полягає в уніфікації методів взаємодії обох моделей і можливості використання переваг обох в будь-який момент (рис. 10). Принцип побудови моделі для Elasticsearch заснований на рекурсивному заповнені умов для побудови згідно API документації сервісу, а назви методів для створення потрібних умов запозичені у моделей Laravel Eloquent, що робить роботу с обома моделями більш зрозумілою.

```
//побудова моделі для Elasticsearch
Street::buildCache();
//отримання запису з бази даних
$row = Street::query()->where('id', '=', '30')->first();
//оновлення запису
$row->city = 'Дніпро';
//збереження змін в базі даних та оновлення кешу
 $row->saveAndBuildCache();
//зчитування кешу
$record = Street::model()->query()->where('id', '=', '30')->get()->first();
```

Рис. 10. Приклад використання розширення моделі Laravel Eloquent.

За результатами тестів, використання Elasticsearch додає приріст продуктивності майже у 200% (рис. 11), при цьому не майже не має різниці, яка складність запитів буде виконана до пошукового сервера Elasticsearch, а також все запити можна розділяти і виконувати окремо – цей момент також не знижує продуктивність виконання коду.

```
Зчитування з бази даних...
Завантажено строк даних: 1869
Час виконання запиту (сек): 2.0773339271545
Зчитування з elasticsearch серверу...
Завантажено строк даних: 1869
Час виконання запиту (сек): 0.091490030288696
Різниця часу у виконанні (сек): 1.9858438968658
```

Рис. 11. Результат тестування кооперацийної моделі.

Для побудови функціональної front-end частини використовуємо як основу відому бібліотеку JQuery. Вона дозволяє пошук скрізь весь DOM сторінки та маніпуляцію будь-якими елементами на HTML сторінці [16]. Так як будь-який сучасних веб додаток потребує динамічності, гнучкості та адаптивності під виконання будь-яких задач, проаналізувавши потреби у змінах було знайдене універсальне рішення.

Ідея динамічної взаємодії полягає в розділенні всіх елементів керування на сторінці по ролям, додаючи атрибути з назвами до тегів. В результаті отримаємо можливість ініціалізації одразу всіх елементів незважаючи на їх додаткові властивості. Додатковою зміною пропоную додати універсальну роль для відправки запитів до нашої back-end частини, а формат спілкування між частинами проекту виконати у вигляді запит-дія, тобто метод достатньо схожий на API стандарт, але використовується JSON масив для передачі дій від back-end (рис. 12). Тобто, ми уніфікуємо всі пов'язані з інтерфейсом події і отримуючи їх за допомогою JQuery AJAX [17] перебираємо та виконуємо кожну окремо від інших. Також додатково суть ідеї ще в розділені дій над будь-яким елементом керування, наприклад, зміна класу, додавання класу, зміна контенту, зміна значення атрибуту, видалення та додавання html коду, показування повідомлення та інше. Тобто інтерфейс повинен отримувати дії над інтерфейсом як результат на запит, але не конкретні назви команд. Дії не мають повторювати один одну, а виконуватись за ланцюгом, який в свою чергу отриманий і сформований на стороні back-end.

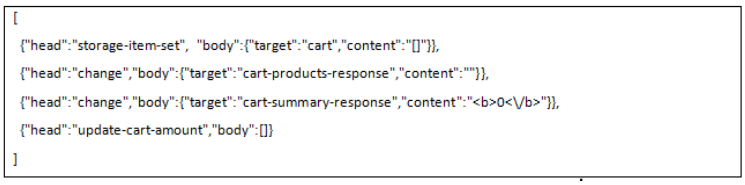


Рис. 12. Приклад формату подій від back-end до front-end.

В результаті використання такого підходу, ми отримаємо абсолютно адаптивний, гнучкий і динамічний інтерфейс з централізованим прийняттям рішення щодо змін в ньому на стороні back-end.

Можливості сучасних методів розробки дуже багаті на інформацію, тому

були проаналізовані наявні можливості і побудована легка, сучасна та універсальна концепція розробки веб додатків будь-якої складності і можливостей, з достатньо високою продуктивністю і можливість використання на високо навантажених проектах. На додаток завдяки цільовому використанню модулів, не виникає проблем з розвитком та підтримкою створених проектів.

СПИСОК ЛІТЕРАТУРИ

1. Мельник Р.А. Програмування веб-застосувань (фронт-енд та бек-енд). Львів: Львівська політехніка, 2018. 248 с.Melnik R.A. Programuvanya web-zastosuvany (front-end ta bek-end) [Programming web application (front-end and back-end)]. Lviv.

2. Adam Crute, Frederic Johnson, Coding HTML CSS JavaScript Made Easy: Web, Apps and Desktop. Flame Tree Illustrated, 2016. 256 p.

3. Васильєв О. Програмування мовою PHP. Київ: Ліра-К, 2022. 368 с. Vasiliev A., 2022, Programuvanya movou PHP [PHP Programming], Kyiv.

4. Веб ресурс duo.ua. Рейтинг мов програмування 2023. URL: https://dou.ua/lenta/articles/language-rating-2023/

5. Веб ресурс getcomposer.org. Composer Documentation. URL:https://getcomposer.org/doc/

6. Веб ресурс nodejs.org. Про проект. URL:https://nodejs.org/uk/about

7. Веб ресурс github.com. GitHub: Let’s build from here. URL:https://github.com/

8. Веб ресурс slimframework.com. Slim 4 Documentation. URL:https://www.slimframework.com/docs/v4/

9. Веб ресурс maxsite.org. Що таке PSR. URL:https://maxsite.org/page/php-psr

10. Веб ресурс duo.ua. Чому SOLID – важлива складова мислення програміста. URL: https://dou.ua/lenta/articles/solid-principles/

11. Веб ресурс symfony.com. Twig – the flexible, fast and secure php

template engine. URL:https://twig.symfony.com/

12. Веб ресурс laravel.com. Eloquent: Getting Started. URL:https://laravel.com/docs/10.x/eloquent

13. Matt Stauffer, Laravel: Up & Running: A Framework for Building Modern PHP Apps 2nd Edition. USA: O'reilly, Incorporated, 2019. 480 p.

14. Веб ресурс elastic.co. Elasticsearch Platform. URL: https://www.elastic.co

15. Clinton Gormley, Zachary Tong, Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine. USA: O'reilly, Incorporated, 2015. 721 p.

16. Веб ресурс jquery.com. JQuery - JavaScript library. URL: https://jquery.com/

17. Elisabeth Robson, Eric Freeman, Head FIRST Javascript Programming : a Brain-Friendly Guide. USA: O'reilly, Incorporated, 2014. 700 p.