JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 22 (2022) 26–34
Received: 18 October 2021
Accepted: 7 November 2021

Comparison of frameworks for developing web applications in PHP

Porównanie szkieletów do wytwarzania aplikacji internetowych dla języka PHP

Kamil Pawelec*, Piotr Kopniak

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The article presents a comparative analysis of two frameworks of PHP application development such as Laravel and Symfony. Test results obtained using the implemented implementation problem compare the application code size, code execution time, hardware resources used, as well as the complexity of the code. The purpose of the research was to present the possibilities of given application development skeletons, with the help of which the web application programmer will be able to decide on its use depending on the implemented solution.

Keywords: PHP; Laravel; Symfony

Streszczenie

Artykuł przedstawia analizę porównawczą dwóch szkieletów budowy aplikacji w języku PHP jakimi są Laravel oraz Symfony. Wyniki badań uzyskane za pomocą analizy dokumentacji oraz zaimplementowanych projektów analogicznie dla dwóch szkieletów porównują funkcjonalność, złożoność kodu aplikacji, czas wykonania kodu, oferowane mechanizmy bezpieczeństwa, obsługi baz danych oraz gniazd sieciowych typu WebSocket. Celem badań była prezentacja możliwości danych szkieletów budowy aplikacji, z pomocą której programista aplikacji internetowych w zależności od implementowanego rozwiązania będzie mógł zdecydować o ich użyciu.

Słowa kluczowe: PHP; Laravel; Symfony

*Corresponding author

Email address: **kamil.m.pawelec@gmail.com** (K. M. Pawelec)

1. Wstęp

Język PHP ciągle jest jednym z najpopularniejszych języków wykorzystywanych do implementacji aplikacji internetowych. Wiele istniejących już systemów i aplikacji internetowych wykorzystuje właśnie ten język, co wiąże się z potrzebą jego ciągłego rozwoju. Według badań przeprowadzonych przez Stackoverflow na 90 tys. programistów 26.4% z nich używa języka PHP, co plasuje go na ósmym miejscu w rankingu najczęściej wykorzystywanych języków programowania [1]. W niniejszej pracy porównano możliwości dwóch z najbardziej popularnych szkieletów do budowy aplikacji dla języka PHP tj. Laravel i Symfony. Przeprowadzenie badań mających na celu wskazanie istotnych różnic pomiędzy nimi może ułatwić podjęcie decyzji o wyborze odpowiedniego narzędzia przed rozpoczęciem projektu. W artykule opisano przeprowadzone badania oraz uzyskane w ich rezultacie wyniki. Pierwszą część artykułu stanowi przegląd publikacji dotyczących tego tematu. Następnie przedstawiono proponowaną metodologię badań wraz z opisem przeprowadzonych badań i uzyskanymi wynikami. W ostatniej części artykułu przedstawiono wnioski z przeprowadzonych badań.

2. Przegląd publikacji dotyczących porównania szkieletów budowy aplikacji

Praca zbiorowa [2] opisuje możliwe sposoby podejścia do tematu porównywania szkieletów budowy aplikacji dla języka PHP. Autorzy przeprowadzili badania przy wykorzystaniu Laravel oraz Symfony i zestawili je za pomocą następujących kryteriów: organizacja kodu, architektura techniczna, wymagania systemowe, wielojęzykowość. Kryteria zastosowane w niniejszej pracy mogą być zatem niejako rozszerzeniem badań przeprowadzonych przez autorów.

Sharma P. K [3] oraz Sol T. [4] przedstawili wyniki badań wskazujące na różnice pomiędzy najczęściej wykorzystywanymi wersjami języka PHP. W przypadku Sol badania przeprowadzono uruchamiając dwie witryny o takiej samej konfiguracji i działające przy wykorzystaniu takiej samej pojemności pamięci w środowisku, gdzie zainstalowany został PHP w wersji 5.6 i 7. Witryny uruchomiono przy pomocy systemu zarządzania treścią WordPress. Przeprowadzone badania wskazują na znacznie zwiększoną wydajność wersji 7 w porównaniu do 5, w związku z czym projekt badawczy wykorzystuje wersję 7.

Autor Skaraczyński w swojej publikacji o szkieletach budowy aplikacji w PHP [5] opisuje wykorzystanie Symfony w praktyce krok po kroku. Publikacja jest instrukcją utworzenia projektu aplikacji. Instrukcje zawarte w publikacji sugerują, że Symfony jest szkieletem przeznaczonym do większych projektów.

Autor Gołda w książce [6] szczegółowo opisuje wykorzystanie Laravel krok po kroku. Prezentuje schemat instalacji szkieletu, zasadę działania kontrolerów oraz routingu, a także widoków. Opisuje również efektywne wykorzystywanie mechanizmów bezpieczeństwa

i sesji, a także m.in. Tworzenie i wykorzystywanie pakietów Composer.

Informacje pochodzące z dokumentacji Laravel [7] oraz Symfony [8] były pomocne w organizacji środowiska badawczego, ze względu m.in. na opis wymagań oraz instrukcje instalacji. Zawierały one również niezbędny opis sposobu wykorzystania poszczególnych funkcjonalności szkieletów budowy aplikacji.

Zaprezentowany przegląd literatury wykazał, że prezentowane dotychczas porównania nie opierają się na implementacji tego samego projektu w różnych szkieletach, co stało się podstawą określenia kierunku badawczego i celu niniejszej pracy. Przegląd literatury pozwolił również na dobór metodyki badawczej, w tym, w określeniu kryteriów porównawczych wykorzystanych w pracy, a także wskazał na jakich aspektach w przeprowadzanych badaniach należy się skupić.

3. Metodyka badawcza

Niniejsza praca prezentuje wyniki badań porównawczych szkieletów budowy aplikacji Symfony oraz Laravel, a dokładnie porównania: mechanizmów bezpieczeństwa, obsługi baz danych i możliwości wykorzystania poszczególnych szkieletów na podstawie opisu funkcjonalności oraz implementacji gniazd sieciowych typu WebSocket. Ponadto przeprowadzone zostały badania wydajności aplikacji zaimplementowanej z wykorzystaniem poszczególnych szkieletów oraz złożoność kodu powstałego w wyniku tych implementacji. Podstawą przeprowadzonych badań była aplikacja internetowa pozwalająca na rejestrację użytkowników i zapisywanie ich na utworzone wydarzenia. Aplikacja umożliwiała komunikację z innymi użytkownikami za pośrednictwem czatu. W przyszłości aplikacja zostanie rozwinięta o możliwość tworzenia/edycji wydarzeń, administrację zgłoszeniami, raporty statystyczne.

Do przeprowadzenia badań wykorzystano maszynę z dwurdzeniowym procesorem Intel 2.20GHz x2, 6GB RAM i systemem operacyjnym Windows 10. Wykorzystane oprogramowanie to: Google Chrome 85, PhpStorm 2019.3, pakiet XAMPP 3.2.4 (PHP 7.2.33, MySQL 127.0.0.1). Na strukturę bazy danych MySQL złożyły się tabele: users, events, apps, pozostające w relacjach: users : apps - jeden do wielu, events : apps – jeden do wielu.

4. Porównanie szkieletów budowy aplikacji

4.1. Funkcjonalność

Laravel jest najczęściej wybieranym szkieletem do realizacji niewielkich projektów, które nie wymagają zbyt dużych nakładów finansowych. Wykorzystuje on komponenty z różnych środowisk, które możliwe są do użycia za pośrednictwem narzędzia Composer. Composer dostarcza i standaryzuje format zarządzania zależnościami i bibliotekami i jest wymagany do instalacji samego szkieletu. Funkcje, które oferuje Laravel to przede wszystkim [9]: modułowość – wspomniany Composer pozwala na integrację również z zewnętrznymi bibliotekami, routing - definiowanie ścieżek użytkownika w aplikacji internetowej, łatwe zarządzanie konfiguracją - aplikacja zaprojektowana w Laravel będzie działać w różnych środowiskach - np. zmiana bazy danych (konfiguracja pliku .env), Query Builder i Eloquent ORM - zarządzanie bazą danych za pomocą modeli i łatwe wykonywanie zapytań, silnik szablonów Blade - widoki, gotowe szablony stron, gotowe mechanizmy uwierzytelniania, metody magiczne, relacyjno-obiektowy model bazy danych.

Symfony przeznaczony jest głównie do projektów złożonych i długoterminowych - jest dość skomplikowany i wymaga większych nakładów finansowych. Umożliwia łatwą rozbudowę projektu oraz integrację z innymi bibliotekami. Główne funkcje Symfony to: silnik szablonów Twig, doctrine ORM - zawiera kilka bibliotek PHP ułatwiających przechowywanie baz danych i mapowanie obiektów, wysoka wydajność, kompatybilność i rozszerzalność, elastyczny routing, zarządzanie sesjami, standaryzacja i interoperacyjność aplikacji.

4.2. Bezpieczeństwo

Laravel posiada mechanizmy zwiększające bezpieczeństwo aplikacji internetowej. Posiada ochronę przeciwko atakom SQL injection, o ile w projekcie wykorzystywany jest Query Builder lub Eloquent. Możliwa jest ochrona plików cookie poprzez wykorzystanie wygenerowanego klucza aplikacji. Jeśli klucz aplikacji nie zostanie ustawiony, sesje użytkownika i inne zaszyfrowane dane nie będą bezpieczne. Klucz aplikacji zwiększa jej bezpieczeństwo o ile pozostaje tajny. Możliwa jest tutaj również ochrona CSRF (ang. Cross Site Request Forgery). Mechanizm ochrony przed opisanym atakiem w Laravel polega na generowaniu tokena CSRF dla każdej sesji zalogowanego użytkownika aplikacji. Token służy sprawdzeniu czy uwierzytelniony użytkownik faktycznie wysyła żądania do aplikacji. W celu wykorzystania zabezpieczenia w danym formularzu HTML implementowanej aplikacji powinna znaleźć się dyrektywa @csrf, a cały proces weryfikacji czy dane wejściowe są zgodne z tokenem przechowywanym w sesji odbędzie się automatycznie. W Laravel możliwa jest również ochrona przed atakami typu Cross-site scripting (XSS). Zastosowanie konstrukcji *{{ }}* w kodzie uniemożliwia wykonanie skryptu osadzonego w treści atakowanej strony.

Autoryzacja w Laravel możliwa jest poprzez wykorzystanie middleware'ów, które odpowiadają za filtrowanie żądań HTTP wprowadzanych do aplikacji. Przykładem może być wpisanie w pasku adresu przeglądarki ścieżki do podstrony, która powinna być dostępna tylko dla uwierzytelnionych (zalogowanych) użytkowników, a w przypadku próby otwarcia takiej podstrony nastąpi np. przekierowanie do strony logowania. Wykorzystanie zdefiniowanego za pomocą klasy Authenticate middleware'a odbywa się w sposób widoczny na Listingu 1.

Listing 1: Klasa Authenticate definiująca middleware'a (plik Authenticate.php)

```
class Authenticate extends Middleware {
  protected function redirectTo($request) {
    if (! $request->expectsJson()) { return route('login'); }}}
```

Funkcja redirectTo przyjmuje argument, który jest ścieżką docelową wybraną przez użytkownika. Dla tej ścieżki wymagane jest logowanie użytkownika. Przekierowanie do strony logowania następuje poprzez metodę route('login'). Po zalogowaniu następuje automatyczne przekierowanie do ścieżki docelowej ($request).

Listing 2: Wykorzystanie middleware'a (plik ChatsController.php)

```
  public function __construct() { $this->middleware('auth'); }
```

Funkcja widoczna na Listingu 2 jest konstruktorem kontrolera. Alias auth definiowany jest w tablicy $routeMiddleware zawartej w klasie Kernel. Automatyczne wygenerowanie systemu uwierzytelniania przede wszystkim zwiększa bezpieczeństwo aplikacji, ze względu na wdrożone już mechanizmy bezpieczeństwa, a także poprzez uniknięcie przypadkowych prostych błędów zabezpieczeń.

Symfony posiada bardzo rozbudowany system zabezpieczeń [10], a jego wykorzystanie odbywa się poprzez specjalnie przygotowany pakiet zabezpieczeń pozwalający na kontrolę dostępu, zarządzanie sesjami i zaporami, czy uwierzytelnianie. Wykorzystanie pakietu możliwe jest po jego uruchomieniu poleceniem:

```
composer require symfony/security-bundle
```

W Symfony plikiem konfiguracyjnym dla uwierzytelniania jest plik security.yaml, który zawiera jednocześnie większość ustawień dotyczących bezpieczeństwa. Definicja uwierzytelniania tworzona jest przy pomocy klucza access_control, w którym podawana jest ścieżka do podstrony, dla której ustalane są reguły dostępu. Możliwe jest blokowanie dostępu po adresie IP, porcie, nazwie hosta, metody łączenia lub uprawnień użytkownika. Przykładowe wykorzystanie reguł dostępu do podstron w projekcie wygląda jak na Listingu 3.

Listing 3: Przykładowe wykorzystanie access_control (plik security.yaml)

```
  access_control:
#     - { path: ^/admin, roles: ROLE_ADMIN }
#     - { path: ^/profile, roles: ROLE_USER }
      - { path: ^/login, roles:
IS_AUTHENTICATED_ANONYMOUSLY }
      - { path: ^/random, roles:
IS_AUTHENTICATED_ANONYMOUSLY }
      - { path: ^/*, roles: ROLE_USER }
      - { path: ^/admin, roles: ROLE_ADMIN }
```

Zabezpieczenie przed atakami typu SQL injections wprzypadku Symfony odbywa się podobnie jak w Laravel po stronie mechanizmu obsługującego bazę danych - w tym przypadku biblioteka Doctrine zawiera wszystkie niezbędne zabezpieczenia przed tego typu atakami. W przypadku ataków typu CSRF sytuacja wygląda identycznie jak w przypadku Laravel. Podczas wysyłania danych z formularzy generowany jest i sprawdzany odpowiedni token CSRF (listing 4).

Listing 4: Zastosowanie tokenu CSRF (plik login.html.twig)

```
<input type="hidden" name="_csrf_token"
  value="{{ csrf_token('authenticate') }}" >
```

Ochrona przed atakami typu XSS również odbywa się analogicznie. Wsparciem w sanityzacji danych wejściowych dostarczanych przez użytkowników jest zastosowanie znaków *{{ }}*, które uniemożliwiają wykonanie skryptu osadzonego pomiędzy znakami. Wsparciem w implementacji wszelkich zabezpieczeń związanych z formularzami może być odpowiedni mechanizm generujący formularze, który pozwala na prostą implementację formularzy wykorzystywanych przez użytkowników i udostępnia szereg metod wspierających.

Symfony dostarcza obiekt konstruktora formularzy, który umożliwia opisywanie pól formularza przy użyciu płynnego interfejsu. Kreator ten w następnym kroku tworzy rzeczywisty obiekt formularza używany do renderowania i przetwarzania treści. Utworzenie formularza możliwe jest np. w kontrolerze, który rozszerza klasę AbstractController przy pomocy metody createFormBuiler(). Symfony zaleca umieszczanie jak najmniejszej logiki w kontrolerach, dlatego lepiej jest przenosić złożone formularze do dedykowanych klas, zamiast definiować je w akcjach kontrolera. Ponadto formularze zdefiniowane w klasach mogą być ponownie wykorzystywane w wielu akcjach i usługach. W przypadku projektu badawczego formularz wykorzystany został w funkcjonalności zapisywania na wydarzenie, w której użytkownik do uzupełnienia ma tylko dwa pola, w związku z czym definicja formularza umieszczona została w kontrolerze. W kontrolerze przy pomocy wspomnianej metody createFormBuiler() tworzona jest definicja formularza. Kolejne pola formularza definiowane są za pomocą metody add(), która w argumentach przyjmuje nazwę pola oraz typ. Metoda getForm() zwraca gotowy obiekt formularza.

Przekazanie formularza do widoku odbywa się przy użyciu metody createView(). Formularz przekazany może być do widoku signup/index.html.twig poprzez parametr applicationForm, w którym jest renderowany za pomocą funkcji pomocniczych form_start, form_widget, form_end. Przycisk zatwierdzający znajduje się bezpośrednio w widoku. Przekazanie danych z formularza odbywa się poprzez metodę handleRequest() oraz obiekt uzupełnionego formularza ($form), dla którego sprawdzane są warunki isSubmitted() i isValid(). Po spełnieniu warunków na obiekcie encji Application przypisywane są wartości jej pól. Obiekt encji poprzez menadżer encji jest następnie kolejkowany (metoda persist()) i zapisywany w tabeli bazy danych (metoda flush()).

4.3. Obsługa baz danych

Laravel w wersji 7 obecnie wspiera następujące systemy zarządzania bazami danych: MySQL, PostgreSQL, SQLite, SQL Server.

Konfiguracja bazy danych odbywa się w pliku config/database.php, jednak dla ułatwienia parametry dostępu do bazy danych definiowane mogą być również w pliku .env, który zawiera zmienne globalne projektu. Na Listingu 5 zaprezentowane są parametry połączenia z bazą danych, która została wykorzystana w projekcie badawczym.

Listing 5: Plik .env z konfiguracją bazy danych

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel1
DB_USERNAME=root
DB_PASSWORD=
```

Korzystanie z bazy danych w projekcie Laravela wspierane jest przez konstruktor zapytań Database Query Builder, a także Eloquent ORM. Query Builder pozwala na generowanie zapytań do bazy danych w wygodny i płynny sposób nawet bez dogłębnej znajomości języka SQL. Wykorzystywane są tutaj metody wywoływane na wbudowanej fasadzie DB. Na ich podstawie możliwe jest skonstruowanie dowolnych zapytań SQL. Przykładowo metoda index() kontrolera ApplicationController zawiera zapytanie zwracające informacje o wydarzeniu, którego identyfikator zawarty jest w zmiennej $event_id. Metoda zwraca wynik zapytania poprzez parametr $event do widoku signup.create (listing 6).

Listing 6: Zapytanie zwracające informacje o wydarzeniach (plik AppController.php)

```
public function index($event_id){
  $event = DB::table('events')
    ->select('events.id', 'events.name', 'events.price',
      'events.start_date', 'events.end_date',
      'events.city', 'events.city')
    ->where('events.id', '=', $event_id)
    →first();
    return view('signup.create')->with('event', $event); }
```

Stosowane w Laravel narzędzie Eloquent ORM pozwala natomiast na prostą interakcję z bazą danych, która polega na przechowywaniu w pamięci obiektu, którego odpowiednikiem jest tabela w bazie danych. Obiekt taki, może posiadać zestaw metod, które pozwalają na interakcję z tą tabelą. Z pomocą modeli możliwe jest wyszukiwanie danych w tabelach, a także dodawanie nowych rekordów. Zastosowana konwencja nazw przyjmuje, że model Event przechowuje rekordy w tabeli events, a więc nazwa tabeli jest nazwą modelu z dopisaną literą ‘s’. Możliwe jest jednak wskazanie niestandardowej tabeli definiując w klasie modelu wartość chronioną $table, której wartością będzie nazwa tabeli, np.

protected $table = 'my_events';

Utworzony model znajduje się w katalogu app i jest on rozszerzeniem wbudowanej klasy Model (Listing 7). Każdy model pozwala na używanie metod Query Builder’a.

Listing 7: Definicja modelu Event (plik Event.php)

```
namespace App;
use Illuminate\Database\Eloquent\Model;
class Event extends Model{ }
```

Migracje w Laravel to mechanizm pozwalający na zautomatyzowane tworzenie tabel w bazie danych. Umożliwiają zachowywanie informacji o tabelach w repozytorium dzięki czemu ułatwia ich odtworzenie w innym środowisku. Migracje znajdują się w katalogu database/migrations. Klasa migracji, która rozszerza klasę Migration, zawiera publiczną funkcję tworzącą tabelę up() oraz funkcję usuwającą tabelę down().

Zastosowanie seed’ów (ang. seeds) pozwala z kolei na automatyczne generowanie danych i uzupełnianie nimi tabel w bazie danych. Wykorzystanie tej funkcjonalności odbywa się poprzez utworzenie klasy rozszerzającej klasę Seeder i implementację metody run(), w której znajdzie się przypisanie wartości do kolumn. Listing 8 prezentuje automatyczne generowanie wartości dla tabeli wydarzeń.

Listing 8: Klasa EventsTableSeeder (plik EventsTableSeeder.php)

```
class EventSeeder extends Seeder{
  public function run() {
    $event = new Event();
    $event->name = 'Oaza Nowego Życia I';
    $event->city = 'Lublin';
    $event->start_date = '2020-09-20';
    $event->end_date = '2020-09-30';
    $event->app_deadline = '2020-09-19';
    $event->description = 'Twoje pierwsze rekolekcje';
    $event->max_app_count = 50;
    $event->price = 550;
    $event→save(); }}
```

Symfony udostępnia wszystkie narzędzia potrzebne do korzystania z baz danych dzięki Doctrine, zestawowi bibliotek PHP do pracy z bazami danych. Narzędzia te obsługują silniki relacyjnych baz danych, takie jak MySQL i PostgreSQL, a także bazy danych NoSQL, takie jak MongoDB [16]. Konfiguracja bazy danych odbywa się tutaj w pliku .env projektu, w którym to w zmiennej środowiskowej o nazwie DATABASE_URL przechowywane są takie dane jak: typ silnika bazodanowego (mysql), nazwa użytkownika serwera MySQL (root), hasło użytkownika (w tym przypadku brak hasła), adres i numer portu, pod którym dostępny jest serwer (127.0.0.1:3306), nazwa bazy danych (symfony) oraz wersja serwera (mariadb-10.4.14).

Podobnie jak w przypadku Laravela, możliwe jest utworzenie klasy obiektu odpowiadającego rekordowi w tabeli. Klasa ta nazywana jest encją. W Doctrine możliwe jest użycie narzędzia make:entity, które w przypadku konieczności utworzenia nowej encji (tabeli) zdecydowanie ułatwia dodawanie, usuwanie czy aktualizację kolejnych jej pól przed wykonaniem migracji na bazie. Utworzenie encji odbywa się za pomocą polecenia:

php bin/console make:entity

Po wywołaniu powyższego polecenia w konsoli, pojawią się odpowiednie zapytania co do nazwy encji oraz

nazw pól, które będzie ona posiadać. W ten sposób utworzona została klasa Event ze zdefiniowanymi polami i zestawem metod dostępowych typu get i set. Klasa ta ma postać jak na listingu 9.

Listing 9: Fragment klasy encji Event (plik Event.php)

```
class Event{
  private $id;
  ...
  private $description;
  public function getId(): ?int{
            return $this->id; }
  public function getName(): ?string{
            return $this->name; }
  public function setName(string $name): self{
     $this->name = $name;
     return $this; }
  public function getCity(): ?string{
     return $this->city; }
  public function setCity(string $city): self{
     $this->city = $city;
     return $this; }  … }
```

Kolejno, podobnie jak w Laravel generowana jest klasa migracji. Wygenerowanie klas migracji dla wszystkich istniejących encji możliwe jest za pomocą polecenia:

php bin/console make:migration

W ten sposób dla encji Event tworzona utworzona zostanie klasa migracji jak na listingu 10.

Listing 10: Klasa migracji (plik Version20201105193925.php)

```
final class Version20201105193925 extends AbstractMigration{
   public function getDescription() : string{
      return ''; }
   public function up(Schema $schema) : void {
      $this->addSql('CREATE TABLE event (id INT AU-
TO_INCREMENT NOT NULL, name VARCHAR(255) NOT
NULL, city VARCHAR(255) NOT NULL, start_date DATE NOT
NULL, end_date DATE NOT NULL, app_deadline DATE NOT
NULL, max_app_count INT NOT NULL, price INT NOT NULL,
description VARCHAR(255) NOT NULL, PRIMARY KEY(id))
DEFAULT CHARACTER SET utf8mb4 COLLATE
`utf8mb4_unicode_ci` ENGINE = InnoDB'); }
   public function down(Schema $schema) : void {
      $this->addSql('DROP TABLE event'); }}
```

Klasa ta w funkcji up() zawiera wygenerowane zapytanie SQL tworzące odpowiednią tabelę w bazie danych. Wywołanie funkcji down() natomiast spowoduje usunięcie utworzonej w bazie danych tabeli. Wywołanie migracji na bazie danych odbywa się przy użyciu polecenia:

php bin/console doctrine:migrations:migrate

W Symfony analogicznie jak w przypadku Laravela możliwe jest wykorzystanie mechanizmu uzupełniania tabel danymi. Tutaj jednak do czynienia mamy z wbudowaną klasą o nazwie Fixture i metodą load(). Wykorzystanie jej możliwe jest po wywołaniu następującego polecenia:

composer require --dev orm-fixtures

Po implementacji klasa definicji Fixture'a dla encji Event przyjmuje postać jak na listingu 11.

Listing 11: Klasa EventFixtures (plik EventFixtures.php)

```
namespace App\DataFixtures;
use App\Entity\Event;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;

class EventFixtures extends Fixture{
   public function load(ObjectManager $manager){
      $event = new Event();
      $event->setName('Oaza Nowego Życia Ist.');
      $event->setAppDeadline(new \DateTime(2021-03-31));
      $event->setCity('Lublin');
      $event->setDescription('To jest pierwsza oaza');
      $event->setStartDate(new \DateTime('2021-04-01'));
      $event->setEndDate(new \DateTime('2021-04-10'));
      $event->setMaxAppCount(50);
      $event->setPrice(550);
      $manager->persist($event);
      $manager→flush(); }}
```

W powyższej definicji kluczowe jest wskazanie klasy encji, ponieważ obiekt tej klasy tworzony jest w funkcji load(). Jako, że nazwa encji odpowiada nazwie tabeli w bazie danych, w późniejszym kroku dany obiekt zapisany zostanie w odpowiadającej mu tabeli. W funkcji load() wywoływane są metody typu set dla poszczególnych pól obiektu klasy Event. Po ustawieniu wartości pól konieczne jest zapisanie obiektu poprzez wykorzystanie menadżera obiektów (ObjectManager) i metod persist() oraz flush(). Zapisanie utworzonego obiektu w tabeli bazy danych możliwe jest poprzez polecenie:

php bin/console doctrine:fixtures:load

Po jego wywołaniu dane wszystkich zdefiniowanych Fixtures zostaną zapisane w odpowiednich tabelach bazy danych. Doctrine podobnie jak Eloquent wykorzystuje mapowanie relacyjno-obiektowe (ORM) w związku z czym ułatwione zostaje wykonywanie zapytań na bazie danych. Przykładowo metoda new() widoczna na Listingu 12 zwróci tablicę z wartościami kolumn z tabeli event (klasa encji Event), gdzie wartość w kolumnie id tablicy będzie równa wartości zapisanej w zmiennej $event_id. Wynik zapytania w dalszych krokach przekazany zostanie za pomocą zmiennej $events do widoku home.html.twig.

Listing 12: Metoda zwracająca rekordy z tabeli event (plik HomeController.php)

```
public function new(Request $request, $event_id){
   $events = $this->getDoctrine()
      ->getRepository(Event::class)
      ->findBy(array('id'=> $event_id)); …
```

4.4. Obsługa gniazd sieciowych - WebSocket

Gniazda sieciowe typu Websocket pozwalają na połączenie dwukierunkowe serwera aplikacji z przeglądarką. Laravel umożliwia ich obsługę np. za pomocą usługi Pusher [12]. Kolejno, wykorzystanie pakietu laravel-websockets znacząco ułatwia implementację połączeń typu WebSocket, ponieważ zawiera on odpowiednią konfigurację, panel debugowania oraz migracje tabel zawierających statystyki przesyłanych danych. Użycie tego pakietu w pierwszej kolejności odbywa się

poprzez ustawienie jego wymagalności w narzędziu Composer za pomocą polecenia:

```
composer require beyondcode/laravel-websockets
```

Kolejnym krokiem jest wykonanie migracji i publikacja pliku konfiguracyjnego za pomocą polecenia:

```
php artisan vendor:publish –
provider="BeyondCode\LaravelWebSockets
WebSocketsServiceProvider" --tag="config".
```

Konfiguracja połączeń typu WebSocket znajduje się w pliku websockets.php. Standardowe ustawienia zawarte w pliku w przypadku realizacji projektu badawczego są wystarczające. Następnym krokiem jest wypełnienie zmiennych środowiskowych tj. parametrów:

```
APP_NAME, PUSHER_APP_ID, PUSHER_APP_KEY,
PUSHER_APP_SECRET
```

w pliku .env. Po uruchomieniu serwera istnieje możliwość monitorowania zdarzeń. Parametr path w pliku konfiguracyjnym określa podstronę, na której dostępny jest panel debugowania. Panel prezentuje statystyki przepływu danych przez zestawione połączenie, umożliwia podgląd przesyłanych danych, a także ich testowe przesyłanie. W pliku broadcasting.php zdefiniowana jest konfiguracja usługi Pusher poprzez podanie adresu hosta i port, na którym ma działać usługa. Przesyłanie wiadomości w czasie rzeczywistym za pomocą czatu, który jest przykładem wykorzystania połączeń typu WebSocket możliwe jest dzięki implementacji zdarzeń (ang. Event).

Listing 13: Przykład klasy Eventu (plik WebSocket Demo Event.php)

```
class WebsocketDemoEvent implements ShouldBroadcast{
use Dispatchable, InteractsWithSockets, SerializesModels;
public $somedata;
public function __construct($somedata){
 $this->somedata = $somedata; }
  public function broadcastOn(){
     return new Channel('DemoChannel'); }}
```

Klasa zdarzenia widoczna na listingu 13 jest implementacją wbudowanego interfejsu ShouldBroadcast. Interfejs ten wymaga zdefiniowania metody broadcastOn(), która odpowiada za zwrócenie kanałów, na których wydarzenie powinno być transmitowane. W parametrze konstruktora klasy przekazywana jest wartość, która ma zostać przesłana za pomocą połączenia Websocket. Metoda sendMessage wysyłająca wiadomości czatu wygląda jak na listingu 14. Zmienna message przechowuje treść przesyłanej wiadomości. Funkcja broadcast przyjmuje jako argument obiekt klasy MessageSent, a w nim zmienną message, którą następnie przesyła (rozgłasza) na wszystkich użytkowników.

Listing 14: Metoda sendMessage (plik ChatsController.php)

```
public function sendMessage(Request $request){
  $message = auth()->user()->messages()->create([
     'message' => $request→message ]);
  broadcast(new MessageSent($message->load('user')))->toOthers();
  return ['status' => 'success'];}
```

W celu implementacji połączeń typu WebSocket w Symfony konieczne jest wykorzystanie biblioteki Ratchet. Biblioteka ta udostępnia narzędzia potrzebne to tworzenia dwukierunkowych aplikacji działających w czasie rzeczywistym pomiędzy klientami, a serwerem poprzez połączenia typu WebSocket. W projekcie wykorzystanie Ratchet rozpoczyna się od modyfikacji pliku *composer.json*, w którym należy dopisać odpowiednią klauzulę:

```
"require":{
"cboden/ratchet": "^0.4.1", …}
```

Kolejnym krokiem jest stworzenie klasy WebSocket. W projekcie klasa ta ma postać jak na listingu 15.

Listing 15: Klasa WebSocket (plik Chat.php)

```
class Chat implements MessageComponentInterface{
  protected $connections = [];
  public function __construct(){ }
  function onOpen(ConnectionInterface $conn): void{
    $this->connections[] = $conn;
    $conn->send('Witaj!' . PHP_EOL);}
  function onClose(ConnectionInterface $conn): void{
     foreach ($this->connections as $key => $connection){
        if($connection === $conn){
           unset($this->connections[$key]);
           break; }}
     $conn->send('Do zobaczenia!' . PHP_EOL);
     $conn→close(); }
  function onError(ConnectionInterface $conn, \Exception $e): void{
     $conn->send('Error ' . $e->getMessage() . PHP_EOL);
     $conn→close(); }
  function onMessage(ConnectionInterface $from, $msg): void{
     $messageData = json_decode(trim($msg));
     foreach ($this->connections as $connection){
        $connection→send($messageData); }}}
```

Klasa MessageComponentInterface jest klasą abstrakcyjną, dlatego wymaga użycia wszystkich czterech metod: onOpen, onClose, onError i onMessage. Metoda onOpen odpowiada za działania podczas nawiązania nowego połączenia z serwerem, w tym przypadku wyświetlenie komunikatu: *Do zobaczenia!* Metoda onClose zostaje uruchomiona w momencie zakończenia połączenia przez klienta. Metoda onError jest uruchamiana w momencie zgłoszenia błędu przez połączenie. Metoda onMessage odpowiada za obsługę wiadomości przesyłanych na serwer. W projekcie badawczym każda wiadomość przesyłana jest do każdego klienta w czasie rzeczywistym.

Po implementacji logiki służącej do przetwarzania połączeń przychodzących na serwer należy zaimplementować nasłuchiwanie portu, w tym przypadku będzie to port 8080. W klasie ChatServer tworzony jest obiekt $chatServer (listing 16), który zawiera serwer HTTP wygenerowany za pomocą obiektu klasy WsServer (wbudowana klasa biblioteki Ratchet) oraz obiektu klasy Chat.

Listing 16: Funkcja execute klasy ChatServer (plik ChatServer.php)

```
protected function execute(InputInterface $input, OutputInterface
$output){
$chatServer = IoServer::factory(new HttpServer(new WsServer(new
```

```
Chat())), 8080);
$chatServer→run(); }
```

Wyświetlanie wiadomości jak i otwarcie połączenia typu WebSocket odbywa się z wykorzystaniem JavaScript jak na listingu poniżej. Zmienna socket przechowuje obiekt gniazda sieciowego, na którym wywoływane są metody onopen, onclose, onmessage, onerror. W dalszej części kodu, generowany jest widok, wyświetlający odpowiednie pola czatu (listing 17).

Listing 17: Obsługa połączenia (plik main.js)

```
var socket = new WebSocket("ws://127.0.0.1:8080");
socket.onopen = function () {
  console.log('Connection successful'); };
socket.onclose = function (event) {
  if (event.wasClean) {
    console.log('Connection closed.'); }
else {
    console.log('Connection killed:('); }
  console.log(event.code + event.reason); };
socket.onmessage = function (event) {
  var list = document.getElementById('list');
  var message = document.createElement("div");
  var node = document.createTextNode(event.data);
  message.appendChild(node);
  message.classList.add('card');
  list.appendChild(message); };
socket.onerror = function (error) {
  console.log(error.message); };
var button = document.getElementById('send');
var textarea = document.getElementById('message-box');
function sendText() {
  var text = textarea.value;
  if (text.length > 0) {
    socket.send(JSON.stringify(text));
    textarea.value = '';
    return true; }
  return false; }
button.onclick = sendText;
textarea.onkeypress = function (ev) {
  if (ev.charCode === 13 && ev.shiftKey) { sendText(); } };
```

4.5. Złożoność kodu

W większości przypadków złożoność kodu przekłada się wprost proporcjonalnie do czasu poświęconego na implementację. W przypadku szkieletu Laravel średnia liczba linii kodu przypadająca na implementację funkcjonalności projektu badawczego wynosi: 96,5 (tabela 1).

Tabela 1: Złożoność kodu dla Laravel

Funkcjonalność	Liczba plików	Liczba linii kodu
Migracja tabeli wydarzeń	1	39
Wyświetlenie wydarzeń	3	89
Zapisanie na wydarzenie	4	114
Czat	7	144

W przypadku szkieletu Symfony średnia liczba linii kodu przypadająca na implementację funkcjonalności projektu badawczego wynosi: 150,75 (tabela 2).

Wynik badania może dowodzić większemu skomplikowaniu kodu implementowanego w Symfony.

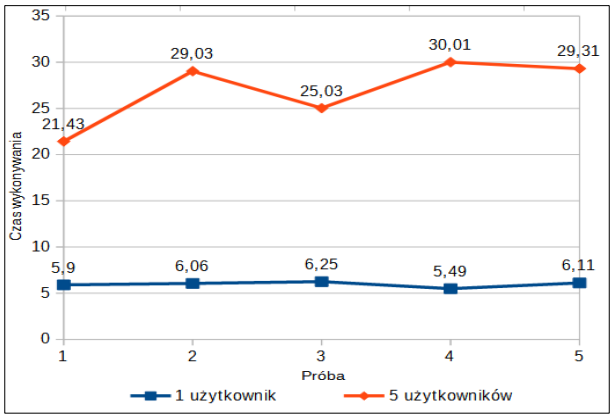
Funkcjonalność	Liczba plików	Liczba linii kodu
Migracja tabeli wydarzeń	1	31
Wyświetlenie wydarzeń	3	220
Zapisanie na wydarzenie	4	179
Czat	5	173

Tabela 2: Złożoność kodu dla Symfony

4.6. Wydajność

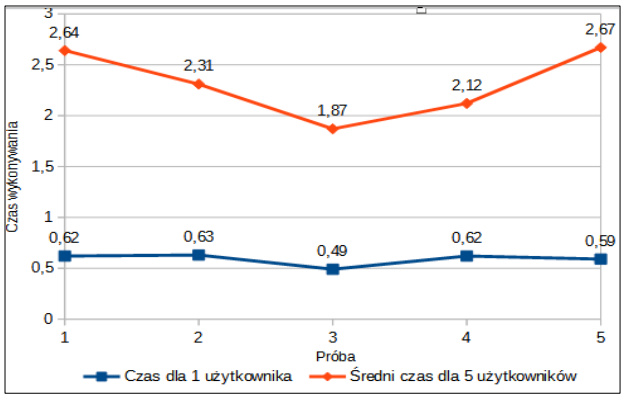
Wydajność projektu docelowego jest najczęstszym kryterium wyboru narzędzi programistycznych przez deweloperów. Przeprowadzono badanie z wykorzystaniem funkcjonalności wyświetlenia 100 rekordów znajdujących się w bazie danych, które dla poszczególnych szkieletów zawierały te same dane. Zaimplementowano metodę zwracającą informacje dotyczące wydarzeń i wyświetlająca te informacje w widoku oraz przeprowadzono zapisywanie na wydarzenia – zatwierdzono formularz zawierający uzupełnione przez użytkownika dane, który wywołał metodę zapisującą nowy rekord w bazie danych. Wykonano po 5 prób badań. Każdą próbę przeprowadzono w obrębie tego samego systemu, w którym uruchomiony został serwer PHP.

Średni czas oczekiwania na wyświetlenie wyników wyniósł odpowiednio 5,96 sekund w przypadku obsługi jednego użytkownika i 26,96 sekund w przypadku obsługi równoległej pięciu użytkowników w obrębie jednego systemu operacyjnego. Szczegółowe wyniki dla wykonanych prób przedstawia wykres widoczny na rysunku 1.



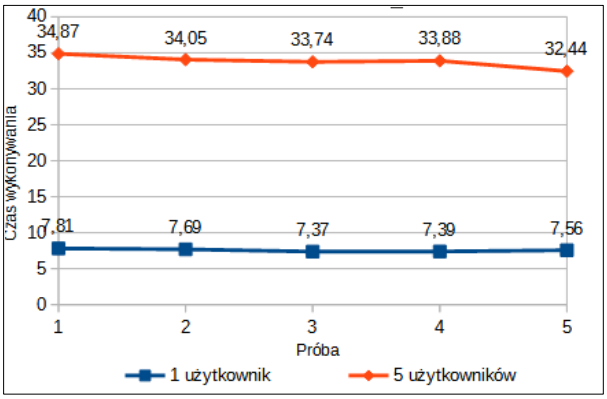
Rysunek 1: Czas wyświetlenia listy wydarzeń w Laravel.

Zapisanie danych w bazie i przeładowanie strony w Laravel odbyło się średnio w czasie odpowiednio 0,59 sekundy w przypadku obsługi jednego użytkownika i 2,32 sekundy w przypadku obsługi równoległej pięciu użytkowników w obrębie jednego systemu operacyjnego. Szczegółowe wyniki dla wykonanych prób przedstawia wykres widoczny na rysunku 2.



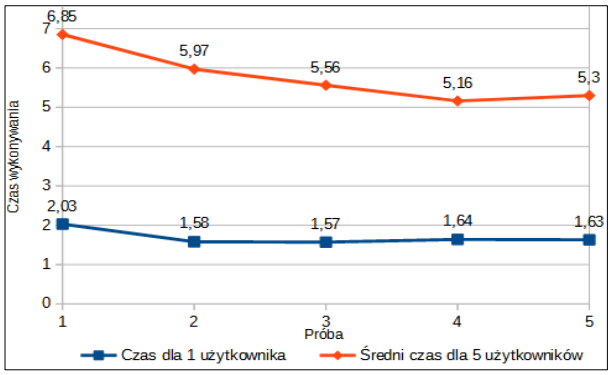
Rysunek 2: Czas zapisania zgłoszenia i przeładowania strony w Laravel.

Średni czas oczekiwania na wyświetlenie wyników wyniósł dla Symfony odpowiednio 7,56 sekund w przypadku obsługi jednego użytkownika i 33,79 sekund w przypadku obsługi równoległej pięciu użytkowników w obrębie jednego systemu operacyjnego. Szczegółowe wyniki dla wykonanych prób przedstawia wykres widoczny na rysunku 3.



Rysunek 3: Czas wyświetlenia listy wydarzeń w Symfony.

Zapisanie danych w bazie i przeładowanie strony w Symfony odbyło się średnio w czasie odpowiednio 1,69 sekundy w przypadku obsługi jednego użytkownika i 5,76 sekundy w przypadku obsługi równoległej pięciu użytkowników w obrębie jednego systemu operacyjnego. Szczegółowe wyniki dla wykonanych prób przedstawia wykres widoczny na rysunku 4.



Rysunek 4: Czas zapisania zgłoszenia i przeładowania strony w Symfony.

5. Podsumowanie

Przeprowadzone badania informują o możliwościach, które daje każdy ze szkieletów. Laravel jest szkieletem programistycznym dość prostym do nauczenia, a jednocześnie posiadającym wszystkie niezbędne mechanizmy do implementacji projektu. Dostępność dokumentacji i aktywna społeczność sprawia, że jest to szkielet bardzo często wybierany przez deweloperów. Podobnie wygląda sytuacja dla Symfony. Tutaj jednak możliwe jest osiągnięcie dogodnych rozwiązań dla projektów rozbudowanych i wymagających ciągłego rozwoju. Tego typu projekty wymagane są w wielu przedsiębiorstwach, gdzie ważne jest ich nieustanne dostosowywanie do zmieniających się wymagań. Informacje zawarte w rozdziale dotyczącym funkcjonalności szkieletów budowy aplikacji znajdują potwierdzenie w kolejnych punktach niniejszej pracy.

W przypadku bezpieczeństwa obydwa szkielety oferują podobne rozwiązania, przy czym implementacja np. systemu uwierzytelniania w Laravel jest znacznie przyspieszona, ponieważ udostępnia on gotowy system uwierzytelniania. Szkielety jednak mają swoje społeczności, które często udostępniają rozwiązania gotowe do implementacji. Obsługa baz danych w przypadku omawianych szkieletów wygląda analogicznie. Implementują mechanizmy usprawniające działania na bazach typu Query Builder, wzorce Active Record lub Data Mapper. Symfony wymaga bardziej szczegółowego określenia reguł dostępu do bazy.

Oba szkielety umożliwiają implementację połączeń typu WebSocket. W przypadku Laravela wydaje się ona szybsza, np. ze względu na udostępniany panel debugowania ułatwiający analizę przepływu danych na serwerze, co pozwala na łatwiejsze wyszukanie błędów działania aplikacji.

Laravel i Symfony są ciągle rozwijane i dostosowywane do zmian wprowadzanych na bieżąco w nowych wersjach języka PHP. Wprowadzane usprawnienia zastosowane w czystym języku przekładają się na usprawnienia wprowadzane w szkieletach budowy aplikacji. W realizacji małych projektów jakim był zaprezentowany projekt badawczy, lepiej sprawdził się Laravel. Wydajność projektu badawczego w przypadku Laravel była większa niż Symfony, natomiast złożoność kodu w Laravel była mniejsza.

Otrzymane wyniki mogą posłużyć jako wskazówki, które mogą być wykorzystane podczas podejmowania decyzji o wyborze docelowego szkieletu programistycznego przez deweloperów.

Literatura

[1] Informacje z ApacheFriends o XAMPP, https://www.apachefriends.org/pl/index.html, [28.12.2020].

[2] M. Laaziri, K. Benmoussa, S. Khoulji, M. L. Kerkeb, A. E. Yamami, A comparative study of Laravel and Symfony PHP frameworks, IJECE, 2019.

[3] Artykuł zawierający porównanie frameworków, https://www.quora.com/What-are-the-major-difference between-PHP-5-and-PHP-7, [28.12.2020].

[4] Porównanie PHP 5.6 i PHP 7, https://gbksoft.com/blog/php-5-vs-php-7-performance-comparison/, [02.01.2020].

[5] T. Skaraczyński, A. Zioła, PHP5. Programowanie z wykorzystaniem Symfony, CakePHP, Zend Framework, Helion, Gliwice, 2009.

[6] M. Gołda, Laravel Tworzenie aplikacji Receptury, Helion, Gliwice, 2015.

[7] P. G. de Gennes, Scaling Concepts in Polymer Physics, Cornell University Press, London, 1979.

[8] Dokumentacja szkieletu Laravel, https://laravel.com/, [28.12.2021].

[9] Dokumentacja szkieletu Symfony, https://symfony.com/, [28.12.2021].

[10] Konfiguracja Doctrine, https://symfony.com/doc/current/reference/configuration/doctrine.html, [28.12.2020].