

DAMIAN JÓŹWIAK

LARAFONY: SOLIDny Framework PHP

TWORZENIE NOWOCZESNEGO SILNIKA W PHP 8.5



**SOLIDne fundamenty, wzorce projektowe
i najwyższa wydajność, zależny tylko od PSR**

Larafony: SOLIDny Framework PHP

Tworzenie nowoczesnego silnika w PHP8.5

Autor: **Damian Józwiak**

Projekt Okładki: **Maciej Peda - Peda Art Design**

Copyright © 2025 **Damian Józwiak**

Wydanie drugie: **2025**

Wszelkie prawa zastrzeżone. Żadna część tej publikacji nie może być reprodukowana, przechowywana w systemie odtwarzania ani przekazywana w jakiegokolwiek formie ani za pomocą jakichkolwiek środków, mechanicznych, fotograficznych, elektronicznych, magnetycznych ani innych bez uprzedniej pisemnej zgody autora.

Strona internetowa książki: **<https://masterphp.eu>**

Kod źródłowy dostępny pod adresem:
<https://github.com/DJWeb-Damian-Jozwiak/larafony>

Dokumentacja Larafony: **<https://larafony.com>**

Zastrzeżenie

Framework stworzony w niniejszej książce jest w pełni pokryty testami automatycznymi, co zapewnia, że każda z jego funkcji została dokładnie zweryfikowana pod kątem poprawności działania. Autor książki, bazując na swojej aktualnej wiedzy, uznaje framework za rozwiązanie stabilne i bezpieczne. Należy jednak pamiętać, że publikacja ta powstała w celach edukacyjnych i stanowi bardziej dowód koncepcji (*ang. proof of concept*) tworzenia własnego frameworka, przy czym szczegółowo opisuje działanie najważniejszych jego komponentów wewnętrznych.

Pomimo wysokiej jakości i bezpieczeństwa stworzonego kodu, autor pragnie wyraźnie zaznaczyć, że **nie rekomenduje** korzystania z tego frameworka w rzeczywistych środowiskach produkcyjnych (choć na tym frameworku została zbudowana działająca produkcyjnie strona larafony.com stanowiąca dokumentację frameworka - co pozwoliło mi wykryć od groma błędów które przeoczyłem w testach). Framework został opracowany głównie jako materiał dydaktyczny, a jego zastosowanie w warunkach produkcyjnych wiąże się z ryzykiem, które każdy użytkownik podejmuje na własną odpowiedzialność.

PS: Jeśli używane środowisko wymaga użycia MsSQL produkcyjne użycie Larafony jest wysoce niezalecane. Z dwóch powodów:

- w larafony stworzyłem jedynie driver do MySQL (co na potrzeby edukacyjne wystarcza). Być może kiedyś to zmienię (PS: jeśli czujesz się na siłach stwórz PR)
- Powód nr 2 jest dużo bardziej brutalną ścianą zderzenia z rzeczywistością. Na momemnt pisania tego ebooka (listopad/grudzień 2025) nie istnieje oficjalne rozszerzenie php mssql działające w php8.5 (nawet 8.4 ma problemy). Autorowi udało się zmusić php8.5 do współpracy z mssql ale zdecydowanie wykracza to poza zakres tego ebooka.

Dlatego też, autor **nie ponosi odpowiedzialności** za jakiegokolwiek ewentualne problemy wynikające z użycia frameworka w środowiskach produkcyjnych, takie jak błędy funkcjonalne, awarie systemowe czy potencjalna utrata danych.

Wstęp

Witaj w książce, która łączy technologię z pasją, a kod z opowieścią. Na kolejnych stronach znajdziesz zarówno praktyczne wskazówki, jak i osobiste refleksje, które towarzyszyły mi podczas tworzenia tego frameworka.

AI - przyjaciel Twój to czy wróg

Żyjemy w ciekawych czasach. Możesz zapytać sztuczną inteligencję o wszystko, a nawet... pogadać z tą książką! Dzięki MCP (ang. Model Context Protocol) od Anthropic opisanemu w jednym z końcowych rozdziałów, ten ebook potrafi stać się Twoim rozmówcą.

Ale AI nie zawsze ma rację.

Czasem odpowiada perfekcyjnie. Czasem halucynuje - czyli mówiąc wprost: zmyśla jak mistrz bajek.

Framework, który za chwilę poznasz, powstawał tak jak tworzy się dobre oprogramowanie: powoli, świadomie, z potknięciami (większość zostawiłem tu w formie notatek autora), refaktoryzacją i małymi zwycięstwami, które tylko programista potrafi docenić.

Pewnie zapytasz, czy korzystałem z AI przy pisaniu tej książki. Tak - ale tylko jako konduktora pociągu, nie maszynisty. To ja trzymam ster, ja piszę kod, ja podejmuję decyzje - AI czasem dopisze literówkę, czasem naprawi styl, czasem podsunie inspirację. Framework jest zero-dependency, i ta książka też. AI to tylko narzędzie - jak debugger, jak dokumentacja, jak kubek kawy obok monitora.

Jeśli coś będzie niejasne - pytaj AI śmiało. Tylko pamiętaj jedno: ta książka używa PHP 8.5, którego AI jeszcze nie zna natywnie. Więc jeśli chatbot twierdzi, że dany kod nie zadziała - najpierw uruchom go lokalnie. Bardzo możliwe, że to Ty masz rację.

Nawiązania literackie

By pokazać, że kodowanie to nie tylko sztuka algorytmów, ale także droga, którą każdy z nas przebywa na swój sposób, w książce znajdziesz dwa niecodzienne wprowadzenia.

Pierwsze, inspirowane hiszpańskim utworem „*La historia de un niño*” autorstwa Porty, oddaje hołd uniwersalnej naturze programowania i osobistej drodze rozwoju. Drugie zaś, w lekkim i humorystycznym tonie, odnosi się do ikonicznej czołówki serialu „*Walker, Texas Ranger*” i dedykowane jest wszystkim tym, którzy z determinacją i uśmiechem stawiają czoła codziennym wyzwaniom programistycznym.

Ta książka to nie tylko przewodnik po frameworku, ale także „**współczesna biblia PHP**” – kompendium nowoczesnych standardów, dobrych praktyk i inspiracji do tworzenia oprogramowania. Na jej stronach znajdziesz wszystko, co sprawia, że PHP pozostaje jednym z najpotężniejszych i najbardziej uniwersalnych języków programowania. Od zgodności z PSR po modularne podejście do kodu, każdy rozdział pokazuje, jak stworzyć i testować narzędzia na miarę XXI wieku.

Obie te inspiracje łączy przekonanie, że za każdą linią kodu kryje się historia, a programowanie to nie tylko narzędzie, ale także sposób wyrażania siebie. Rozpocznijmy tę podróż!

Cuando era solo un niño, el código no entendía,
Las letras y números se mezclaban en mi fantasía.
Cada error me frenaba, cada bug me asustaba,
Pero el sueño crecía en mí, mientras el miedo pasaba.

Frente a la pantalla largas noches, el teclado sonaba,
Paso a paso aprendí a crear, de lo pequeño algo se alzaba.
Ahora el código es mi arte, el mundo espera mi canción,
En cada línea veo el poder, es mi mayor ambición.

Este libro es mi guía, el código mi misión,
MVC, patrones, PSR, construyen mi visión.
Con cada capítulo, más claro es el sendero,
Escribiendo líneas, el futuro es lo que quiero.

Cuando era solo un niño, el código no entendía...

[FIRST VERSE]

In the eyes of new PHP
the unsuspecting Behav'
Had better know the truth from wrong or right;

Rules of Solid and PSR always followed
that "dead"language surely beloved
With the Lone Star of the Ranger shining bright.

[CHORUS]

Asymmetric visibility is upon you
everything is tested like TDD says
when it's that framework don't look behind you
'Cause you're the Ranger, who everyone cares

[SECOND VERSE]

In the heart of a Ranger,
he'll never know the danger;
From old where everything is untyped

our new code is comming
so there ain't no need to wonder any test is green and running
To new where each code line is hyped

Spis treści

1	Wstęp	57
1.1	Dobre praktyki	57
1.1.1	SOLID	58
1.1.2	YAGNI	61
1.2	Wzorce projektowe	62
1.2.1	Budowniczy	62
1.2.2	Dekorator	65
1.2.3	Fabryka	69
1.2.4	Fasada	71
1.3	PSR	75
1.3.1	PSR-3, tworzymy logi	75
1.3.2	PSR-4 i autoloading	76
1.3.3	PSR-6 cache	76
1.3.4	PSR-7, requesty HTTP	77
1.3.5	PSR-11, dodajemy kontener zależności	77

1.3.6	PSR-12 czyli jakość kodu	78
1.3.7	PSR-14, event dispatcher	78
1.3.8	PSR-15, dodajemy middleware	78
1.3.9	PSR-17, HTTP factory	79
1.3.10	PSR-20, Clock	79
1.4	Zarządzanie zależnościami	80
1.4.1	Zarządzanie zależnościami	80
1.4.2	Przykład użycia	81
1.5	Konfiguracja lokalnego repozytorium	81
1.5.1	Utworzenie struktury katalogów	82
1.5.2	Utworzenie głównego pliku projektu	82
1.5.3	Utworzenie pustego repozytorium w katalogu framework	83
1.5.4	Dodanie pliku <code>composer.json</code>	83
1.5.5	Dodanie pakietu <code>symfony/var-dumper</code>	84
1.5.6	Instalacja zależności	85
1.5.7	Konfiguracja vHosta	85
1.6	Punkt wejścia aplikacji	88
1.6.1	Zalety Jednego Entry Point	88
1.7	PHPUnit	89
1.8	phpstan	89
1.9	Instalacja i konfiguracja PHPUnit i phpstan	90
1.9.1	Pokrycie kodu	92

<i>SPIS TREŚCI</i>	9
2 Obsługa błędów	95
2.1 Architektura systemu obsługi błędów	96
2.2 Interfejs ErrorHandler	96
2.3 Implementacja DetailedErrorHandler	97
2.3.1 Analiza implementacji	99
2.4 Formatter HTML	100
2.4.1 Bezpieczeństwo	102
2.5 Aplikacja demonstracyjna	103
2.5.1 Analiza tras demonstracyjnych	105
2.6 Testy jednostkowe	105
2.6.1 Testy DetailedErrorHandler	105
2.6.2 Testy HtmlErrorFormatter	108
2.7 Podsumowanie	110
3 System czasu – PSR-20	112
3.1 Architektura systemu	113
3.2 Interfejs Clock	114
3.3 Enumy konfiguracyjne	116
3.3.1 TimeFormat	116
3.3.2 Timezone	117
3.4 SystemClock – implementacja produkcyjna	118
3.4.1 Analiza implementacji	123

3.5	FrozenClock – implementacja testowa	123
3.5.1	Podróżowanie w czasie	127
3.6	ClockFactory – wzorzec Strategia	128
3.6.1	Użycie w aplikacji	131
3.7	Integracja z aplikacją demonstracyjną	132
3.8	Testy jednostkowe	133
3.8.1	Testy SystemClock	133
3.8.2	Testy FrozenClock	136
3.8.3	Testy ClockFactory	140
3.9	Podsumowanie	144
4	Request i response	145
4.1	Request i Response – Podejście współczesne	146
4.2	Natywne wsparcie URI w PHP 8.5	146
4.2.1	Helperzy do tworzenia URI z globals	150
4.2.2	Fabryka URI	152
4.3	Implementacja interfejsu Stream	153
4.3.1	Stream Wrapper	153
4.3.2	Stream Meta Data	155
4.3.3	Stream Content Manager	157
4.3.4	Stream Size	158
4.3.5	Klasa Stream	159

4.3.6	Fabryka Stream	164
4.4	Implementujemy wgrywane pliki	165
4.4.1	Handler fizycznych plików	166
4.4.2	Handler strumieni	167
4.4.3	Dekoratory	168
4.4.4	Bazowa implementacja interfejsu UploadedFileInterface .	173
4.4.5	Fabryka UploadedFile z dekoratorami	175
4.5	Implementacja klasy Request	177
4.5.1	Klasa bazowa Message	177
4.5.2	Klasa HeaderManager	180
4.5.3	Klasa Request	182
4.5.4	Parsowanie nagłówków	184
4.5.5	Managery dla ServerRequest	186
4.5.6	Klasa ServerRequest	193
4.6	Response	198
4.6.1	Kody statusów HTTP	198
4.6.2	Klasa Response	202
4.7	PSR-17 Fabryki	203
4.7.1	Request Factory	203
4.7.2	Server Request Factory	204
4.7.3	Response Factory	207
4.8	Podsumowanie	209

5	Dependency Injection	210
5.1	Dlaczego Dependency Injection jest ważne?	211
5.2	PSR-11 – Container Interface	211
5.3	Architektura kontenera	212
5.4	Interfejs kontenera	212
5.5	Wyjątki zgodne z PSR-11	213
5.6	Reflection API – zagłębienie pod maskę	214
5.6.1	Interfejs ReflectionResolver	215
5.6.2	Implementacja ReflectionResolver	217
5.6.3	Ewolucja typów w PHP	221
5.6.4	Nullable types – historia i teraźniejszość	221
5.7	Autowire – automatyczne wstrzykiwanie zależności	222
5.7.1	Interfejs Autowire	222
5.7.2	Implementacja Autowire	222
5.7.3	Algorytm rozwiązywania zależności	226
5.8	DotContainer – storage z kropką	226
5.8.1	Interfejs ArrayContract	227
5.8.2	Pomocnik ArrayGet	227
5.8.3	Pomocnik ArraySet	228
5.8.4	DotContainer	229
5.9	Główna klasa Container	231
5.9.1	Atrybut #[NoDiscard] – nowość PHP 8.5	231

5.9.2	Dwa systemy przechowywania	234
5.9.3	Lazy autowiring	234
5.10	Service Provider – modularność usług	235
5.10.1	Interfejs ServiceProvider	235
5.10.2	Implementacja ServiceProvider	236
5.10.3	Cykl życia Service Providera	237
5.11	Testy jednostkowe	238
5.11.1	Klasy pomocnicze do testów	238
5.11.2	Testy ReflectionResolver	239
5.11.3	Testy DotContainer	242
5.11.4	Testy Container	243
5.12	Podsumowanie	245
6	Routing – trasowanie żądań	247
6.1	Dlaczego routing jest ważny?	248
6.2	Architektura routingu Basic	248
6.3	Enum HttpMethod	249
6.4	Klasa Route – pojedyncza trasa	250
6.4.1	PSR-15 RequestHandlerInterface	251
6.5	RouteMatcher – czy trasa pasuje?	251
6.5.1	Normalizacja trailing slash	253
6.6	RouteCollection – kolekcja tras	253

6.6.1	Funkcja <code>array_find</code> z PHP 8.4	255
6.6.2	Asymetryczna widoczność <code>private(set)</code>	256
6.7	Wyjątek <code>RouteNotFoundError</code>	256
6.8	Handlerzy – różne sposoby obsługi trasy	257
6.8.1	<code>ClosureRouteHandler</code>	257
6.8.2	<code>ClassMethodRouteHandler</code>	258
6.8.3	<code>InvokableClassRouteHandler</code>	260
6.8.4	<code>FunctionRouteHandler</code>	262
6.9	Fabryki handlerów	262
6.9.1	<code>RouteHandlerFactory</code>	262
6.9.2	<code>ArrayHandlerFactory</code>	264
6.9.3	<code>StringHandlerFactory</code>	265
6.10	Klasa <code>Router</code>	267
6.10.1	Fluent interface	269
6.11	Service Provider	269
6.12	Integracja z aplikacją	271
6.12.1	Czysty <code>index.php</code>	271
6.12.2	Bootstrap aplikacji	271
6.12.3	<code>DemoController</code> – pierwszy kontroler	273
6.13	Testy jednostkowe	273
6.13.1	Testy <code>RouteMatcher</code>	273
6.13.2	Testy <code>Route</code>	278

6.13.3 Testy Router	280
6.14 Podsumowanie	284
7 HTTP Client – PSR-18	286
7.1 PSR-18 – HTTP Client Interface	287
7.2 Architektura klienta HTTP	287
7.3 Konfiguracja – HttpClientConfig	288
7.3.1 Statyczne factory methods	291
7.4 Główna klasa CurlHttpClient	291
7.5 CurlOptionsBuilder – PSR-7 do CURL	294
7.5.1 Operator + zamiast array_merge	297
7.5.2 Pipe operator >	298
7.6 CurlHandleExecutor – wykonanie żądania	298
7.6.1 Mapowanie błędów CURL	302
7.6.2 Zamykanie CurlHandle	302
7.7 Abstrakcja CURL dla testowalności	303
7.7.1 Interfejs CurlWrapperContract	303
7.7.2 Produkcyjna implementacja CurlWrapper	305
7.7.3 Testowa implementacja FakeCurlWrapper	306
7.8 Parsowanie odpowiedzi	309
7.8.1 ResponseParser	309
7.8.2 ResponseHeadersParser	311

7.8.3	StatusLineParser	312
7.9	Hierarchia wyjątków	313
7.9.1	NetworkError	314
7.9.2	TimeoutError	317
7.10	Mockowanie w testach	318
7.10.1	Interfejs MockHandler	318
7.10.2	CallbackMockHandler	319
7.10.3	MockHttpClient	321
7.11	HttpClientFactory – Service Locator	324
7.11.1	Użycie w testach	327
7.12	Testy jednostkowe	327
7.12.1	Testy HttpClientFactory	327
7.13	Przykład użycia	332
7.14	Podsumowanie	333
8	Konfiguracja aplikacji	335
8.1	Notacja kropkowa	336
8.2	Własny parser .env	336
8.2.1	Architektura parsera	337
8.2.2	Typy linii – enum LineType	337
8.2.3	DTO dla sparsowanych danych	338
8.2.4	ValueParser – obsługa cudzysłowów i escape sequences	341

8.2.5	LineParser – parsowanie pojedynczej linii	344
8.2.6	DotenvParser – główny parser	346
8.2.7	EnvironmentLoader – fasada do ładowania	347
8.2.8	EnvReader – pomocnik do odczytu	350
8.3	Klasa ConfigBase	351
8.4	ConfigServiceProvider	353
8.5	Struktura plików w aplikacji	354
8.5.1	Plik .env	354
8.5.2	Plik config/app.php	355
8.6	Weryfikacja poprawności działania	355
8.6.1	Test DotenvParser	355
8.6.2	Test EnvironmentLoader	360
8.7	Podsumowanie	363
9	Aplikacja konsolowa	365
9.1	Ogólna struktura aplikacji konsolowej	366
9.2	Formatowanie wyjścia – kolory ANSI	367
9.2.1	Definicje kolorów i stylów	367
9.2.2	Bazowy formatter i konkretne style	369
9.2.3	OutputFormatter – mapowanie tagów na style	371
9.2.4	Output – fasada wyjścia	373
9.3	Atrybuty komend	380

9.4	Parsowanie inputu	383
9.5	Bazowa klasa Command	387
9.6	Rozwiązywanie argumentów i opcji	388
9.7	Rejestracja i wykrywanie komend	392
9.8	Console/Application i Kernel	397
9.9	ConsoleServiceProvider	401
9.10	Punkt wejścia – bin/larafony	403
9.11	Weryfikacja poprawności działania	405
9.11.1	Test klasy Output	405
9.11.2	Test aplikacji konsolowej	409
9.12	Przykładowa komenda w demo-app	410
9.13	Podsumowanie	411
10	DBAL – schemat bazy danych	413
10.1	Wprowadzenie do DBAL	420
10.2	Architektura DBAL	421
10.3	Interfejs ConnectionContract	422
10.4	Wzorzec Grammar	424
10.4.1	Interfejs GrammarContract	424
10.4.2	Implementacja MySQL Grammar	425
10.5	TableDefinition – fluent API	427
10.5.1	MySQL TableDefinition	427

10.6	Klasy kolumn	432
10.6.1	Bazowa klasa BaseColumn	432
10.6.2	IntColumn – kolumna liczbowa	434
10.6.3	StringColumn – kolumna tekstowa	438
10.6.4	DateTimeColumn – kolumna daty i czasu	439
10.6.5	EnumColumn – kolumna z predefiniowanymi wartościami	441
10.7	Indeksy	443
10.7.1	Bazowa klasa IndexDefinition	443
10.7.2	PrimaryIndex	445
10.8	SchemaBuilder – orkiestrator	445
10.8.1	Bazowa klasa SchemaBuilder	446
10.8.2	MySQL SchemaBuilder	447
10.9	Buildery SQL	450
10.9.1	CreateTableBuilder	450
10.9.2	AddColumns Builder	452
10.9.3	ChangeColumns Builder	454
10.9.4	DropColumns Builder	455
10.10	Testy jednostkowe	456
10.10.1	Testy klasy Grammar	456
10.10.2	Testy IntColumn	460
10.10.3	Testy TableDefinition	463
10.11	Podsumowanie	470

11 DBAL – QueryBuilder	472
11.1 Architektura QueryBuilder	473
11.2 QueryDefinition – stan zapytania	475
11.3 Enumy – typy zapytań i kierunki	477
11.4 Klauzule WHERE	480
11.4.1 Bazowa klasa WhereClause	480
11.4.2 WhereBasic – podstawowe porównanie	481
11.4.3 WhereIn – lista wartości	482
11.4.4 WhereNull – sprawdzanie NULL	483
11.4.5 WhereBetween – zakres wartości	484
11.4.6 WhereLike – wyszukiwanie wzorca	486
11.4.7 WhereNested – zagnieżdżone warunki	487
11.5 Pozostałe klauzule	489
11.5.1 JoinClause	489
11.5.2 OrderByClause	490
11.5.3 LimitClause	491
11.6 Grammar – kompilator zapytań	492
11.6.1 Interfejs GrammarContract	492
11.6.2 MySQL Grammar	493
11.7 Buildery zapytań	497
11.7.1 SelectBuilder	497
11.7.2 InsertBuilder	498

11.7.3	UpdateBuilder	499
11.7.4	DeleteBuilder	500
11.8	Komponenty builderów	502
11.8.1	WhereBuilder	502
11.8.2	JoinBuilder	503
11.8.3	OrderByBuilder	504
11.8.4	LimitBuilder	505
11.9	QueryBuilder – fluent API	506
11.9.1	Bazowa klasa QueryBuilder	506
11.9.2	MySQL QueryBuilder	510
11.10	Przykłady użycia	519
11.10.1	SELECT	519
11.10.2	INSERT	520
11.10.3	UPDATE	520
11.10.4	DELETE	520
11.10.5	Debugowanie	521
11.11	Testy jednostkowe	522
11.12	Podsumowanie	529
12	DBAL – migracje	530
12.1	Architektura systemu migracji	530
12.2	Klasa Migration	531

12.3 MigrationResolver – skanowanie migracji	533
12.4 MigrationRepository – śledzenie wykonanych migracji	535
12.5 MigrationExecutor – wykonywanie migracji	539
12.6 Komendy do migracji	541
12.6.1 Abstrakcyjna komenda Make	541
12.6.2 Stub migracji	543
12.6.3 Komenda make:migration	544
12.6.4 Komenda migrate	547
12.6.5 Komenda migrate:rollback	549
12.6.6 Komenda migrate:fresh	552
12.7 Przykładowa migracja	554
12.8 Konfiguracja bazy danych	556
12.9 Testy jednostkowe	558
12.9.1 Test komendy MakeMigration	558
12.10 Weryfikacja manualna	565
12.11 Podsumowanie	566
13 DBAL – ORM	568
13.1 Architektura ORM	569
13.2 Fasada DB	570
13.3 PropertyObserver – śledzenie zmian	572
13.4 Klasa Model	573

13.5 EntityManager – zapisywanie modeli	579
13.6 ModelQueryBuilder	582
13.7 Relacje	589
13.7.1 Bazowa klasa Relation	589
13.7.2 Atrybuty relacji	590
13.7.3 Implementacja HasMany	591
13.7.4 Implementacja BelongsTo	592
13.7.5 Implementacja BelongsToMany	594
13.7.6 RelationDecorator	597
13.8 Eager Loading – rozwiązanie problemu N+1	601
13.8.1 EagerRelationsLoader	602
13.8.2 BaseRelationLoader	605
13.8.3 HasManyLoader	608
13.8.4 BelongsToLoader	610
13.9 Przykładowe modele	613
13.9.1 Model User	613
13.10 Castowanie atrybutów	617
13.10.1 Atrybut CastUsing	617
13.10.2 AttributeCaster	618
13.11 Komendy generujące	620
13.11.1 Komenda make:model	620
13.11.2 Stub modelu	623

<i>SPIS TREŚCI</i>	24
13.11.3 Klasa Seeder	624
13.11.4 Komenda database:seed	625
13.12 Przykłady użycia	626
13.12.1 CRUD	626
13.12.2 Relacje	627
13.13 Integracja z aplikacją	628
13.13.1 DatabaseManager	628
13.13.2 Fasada Schema	632
13.13.3 DatabaseServiceProvider	635
13.14 Testy jednostkowe	637
13.15 Podsumowanie	643
14 PSR-3 – System logowania	645
14.1 Architektura systemu logowania	645
14.2 PSR-3 – 8 poziomów logowania	646
14.3 Obsługa kontekstu i placeholderów	647
14.4 Klasa Message	649
14.5 Klasa Logger	651
14.6 Formattery	653
14.6.1 JsonFormatter	653
14.6.2 TextFormatter	654
14.6.3 XmlFormatter	656

14.7	Handler bazy danych	658
14.7.1	Komenda tworzenia tabeli	658
14.7.2	Model DatabaseLog	660
14.7.3	Rozszerzenie CastUsing o castBack	662
14.7.4	Rozszerzenie AttributeCaster o castBack	663
14.7.5	ArrayCaster	665
14.7.6	DatabaseHandler	667
14.8	Handler plikowy z rotacją	668
14.8.1	DailyRotator	668
14.8.2	FileHandler	671
14.9	Integracja z frameworkiem	673
14.9.1	LoggerFactory	673
14.9.2	Konfiguracja	674
14.9.3	LogServiceProvider	675
14.9.4	Fasada Log	676
14.10	Przykłady użycia	679
14.11	Testy jednostkowe	680
14.11.1	Test TextFormatter	680
14.11.2	Test fasady Log	685
14.12	Podsumowanie	689
15	Middleware i Zaawansowane Trasowanie	691

15.1 Przypomnienie PSR-15	691
15.2 Implementacja stosu middleware	692
15.3 Router Middleware	694
15.4 Kernel aplikacji webowej	696
15.5 Trasowanie z parametrami	698
15.5.1 Klasa RouteParameter	698
15.5.2 Parsowanie parametrów	699
15.5.3 Zaawansowana trasa	700
15.6 Dopasowywanie tras z parametrami	704
15.7 Uzupełnienie handlerów o ParameterResolver	707
15.7.1 ParameterResolver	707
15.7.2 FunctionRouteHandler z ParameterResolver	710
15.7.3 ClassMethodRouteHandler z ParameterResolver	711
15.7.4 InvokableClassRouteHandler z ParameterResolver	713
15.8 Automatyczne bindowanie modeli	715
15.8.1 Klasa RouteBinding	716
15.8.2 Klasa ModelBinder	716
15.8.3 Implementacja findForRoute w modelu	719
15.9 Zaawansowany Router	720
15.10 Grupowanie tras	723
15.11 Middleware na poziomie trasy	726
15.12 Atrybuty kontrolera	728

15.12.1 Atrybut Route	728
15.12.2 Atrybut RouteGroup	729
15.12.3 Atrybut RouteParam	729
15.12.4 Atrybut Middleware	730
15.13 Skanowanie i ładowanie tras z atrybutów	731
15.13.1 Skanowanie katalogów	731
15.13.2 Ładowanie tras	733
15.13.3 Procesory atrybutów	735
15.13.4 Budowanie trasy	738
15.14 Przykład kontrolera z atrybutami	740
15.15 Struktura katalogów	741
15.16 Weryfikacja poprawności działania	742
15.16.1 Test RouteMatcher	742
15.16.2 Test atrybutu Middleware	748
15.16.3 Test Kernel	751
16 Weryfikacja formularzy	759
16.1 Dlaczego nie można ufać frontendowi	759
16.1.1 Łatwość manipulacji danymi wejściowymi	759
16.1.2 Weryfikacja po stronie klienta jako wsparcie, nie zabez- pieczenie	760
16.1.3 Znaczenie walidacji po stronie serwera	760
16.2 Wprowadzenie do narzędzi testujących	760

16.2.1	Burp Community	761
16.2.2	Postman	761
16.2.3	AI w testach penetracyjnych	762
16.2.4	OWASP ZAP	762
16.3	Czasowniki HTTP	763
16.3.1	GET	763
16.3.2	POST	763
16.3.3	PUT	764
16.3.4	DELETE	764
16.3.5	PATCH	764
16.4	Implementacja walidatora	764
16.5	Atrybuty walidacji	766
16.5.1	Atrybut IsValidated	768
16.5.2	Atrybut Required	768
16.5.3	Atrybut MaxLength	769
16.5.4	Atrybut MinLength	770
16.5.5	Atrybut Min i Max	771
16.5.6	Atrybut Email	772
16.5.7	Atrybut Confirmed	772
16.5.8	Atrybut RequiredWhen	774
16.6	Walidator atrybutów	777
16.6.1	Wyjątek ValidationFailed	778

16.6.2 Klasa AttributeValidator	780
16.7 Rozszerzamy ServerRequest	783
16.8 Integracja z routingiem – FormRequestAwareHandler	786
16.8.1 Kontrakt PreResolutionHookContract	786
16.8.2 Detektor typu FormRequest	787
16.8.3 Fabryka FormRequest	788
16.8.4 Handler z obsługą walidacji	793
16.9 Aktualizacja fabryk handlerów	796
16.10 Struktura katalogów	798
16.11 Weryfikacja poprawności działania	799
16.11.1 Test FormRequest	799
16.11.2 Test ValidationResult	804
16.11.3 Test FormRequestAwareHandler	806
17 Widoki – własny silnik szablonów	813
17.1 Dlaczego własny silnik szablonów	814
17.2 Architektura systemu widoków	815
17.3 Konfiguracja	815
17.4 Kontrakty	816
17.4.1 RendererContract	816
17.4.2 ViewContract	818
17.4.3 DirectiveContract	818

17.5 Klasa View	819
17.6 ViewManager	821
17.7 Ładowanie szablonów	822
17.8 Kompilator szablonów	823
17.8.1 TemplateCompiler	823
17.8.2 Bazowa klasa Directive	828
17.8.3 Kolejność dyrektyw ma ogromne znaczenie!	829
17.8.4 Ekstrakcja instrukcji use	830
17.9 Dyrektywy kontroli przepływu	831
17.9.1 IfDirective	831
17.9.2 ForeachDirective	833
17.9.3 ForDirective	834
17.9.4 WhileDirective	834
17.9.5 DoWhileDirective	835
17.9.6 UnlessDirective	836
17.9.7 SwitchDirective	837
17.9.8 IssetDirective	839
17.9.9 EmptyDirective	840
17.10 Dyrektywy layoutów	841
17.10.1 ExtendDirective	841
17.10.2 SectionDirective	842
17.10.3 YieldDirective	843

17.11	System komponentów	844
17.11.1	Kontrakt ComponentContract	844
17.11.2	Klasa bazowa Component	845
17.11.3	ComponentDirective	848
17.11.4	SlotDirective	853
17.12	Stosy zasobów (Asset Stacks)	854
17.12.1	AssetManager	854
17.12.2	StackDirective	855
17.13	Integracja z Vite	857
17.14	BladeAdapter	857
17.14.1	BaseAdapter	863
17.15	Cache widoków i wydajność	864
17.16	BladeSourceMapper – mapowanie błędów	865
17.17	ViewServiceProvider	868
17.18	Struktura katalogów	869
17.19	Testy jednostkowe	871
17.19.1	Test View	871
17.19.2	Test ViewManager	876
18	Publikacja frameworka – od monorepo do pakietów	881
18.1	Dlaczego rozdzielenie?	881
18.2	Nowa struktura pakietów	882

18.2.1	Przed rozdzieleniem (monorepo)	882
18.2.2	Po rozdzieleniu (dwa repozytoria)	882
18.3	Zmiany w composer.json	884
18.3.1	Zmiana nazwy pakietu	884
18.3.2	Deklaracja implementacji PSR	884
18.3.3	Wersjonowanie implementacji	885
18.4	Szkielet aplikacji	885
18.4.1	Struktura katalogów	885
18.4.2	Zależność od rdzenia	886
18.5	Proces migracji	887
18.5.1	Krok 1: Wydzielenie demo-app	887
18.5.2	Krok 2: Aktualizacja bootstrap	888
18.5.3	Krok 3: Czyszczenie repozytorium frameworka	889
18.5.4	Krok 4: Aktualizacja metadanych	889
18.6	Poprawki przed publikacją	889
18.6.1	Kolejność kompilacji dyrektyw Blade	889
18.6.2	Metoda redirect w kontrolerze bazowym	890
18.7	Publikacja na Packagist	890
18.7.1	Wymagania	890
18.7.2	Proces publikacji	891
18.7.3	Wersjonowanie semantyczne	891
18.8	Instalacja dla użytkowników	892

18.8.1	Nowy projekt	892
18.8.2	Tylko rdzeń (dla zaawansowanych)	892
18.9	Korzyści z separacji	893
18.9.1	Dla użytkowników frameworka	893
18.9.2	Dla twórców frameworka	893
18.9.3	Dla społeczności	893
18.10	Roadmapa dalszego rozwoju	893
18.11	Podsumowanie	894
19	Integracja z Vue.js przez Inertia.js	896
19.1	Nowoczesne aplikacje webowe	897
19.1.1	Czym jest SPA	897
19.2	Protokół Inertia.js	897
19.2.1	Podstawowe założenia protokołu	898
19.2.2	Mechanika działania	898
19.2.3	Struktura odpowiedzi	898
19.2.4	Mechanizm udostępniania danych globalnych	899
19.2.5	Obsługa błędów i stanu	899
19.2.6	Zalety wykorzystania Inertia.js	899
19.3	Vue 3 – krótkie wprowadzenie	900
19.3.1	Szablony	900
19.3.2	Dyrektywy i wirtualna struktura dokumentu	900

19.3.3 Options API	901
19.3.4 Options API – przykład komponentu	901
19.3.5 Composition API	902
19.3.6 Composition API – przykład komponentu	902
19.3.7 Composition API – script setup	903
19.4 Generowanie odpowiedzi Inertia	905
19.4.1 Klasa ResponseFactory	905
19.4.2 Klasa Inertia	908
19.5 Klasa Vite	913
19.6 Dyrektywa @vite	917
19.7 Middleware InertiaMiddleware	918
19.7.1 Konfiguracja middleware	921
19.7.2 Rozszerzanie middleware w aplikacji	922
19.8 Konfiguracja frontendu	923
19.8.1 Node.js i package.json	923
19.8.2 Czym jest Vite	924
19.8.3 Plik vite.config.js	924
19.8.4 Inertia Root View	926
19.8.5 Plik app.js	927
19.8.6 Przykładowy komponent strony	928
19.9 Uruchomienie aplikacji	930
19.9.1 Tryb deweloperski	930

19.9.2 Tryb produkcyjny	931
19.10Struktura katalogów	932
19.11Podsumowanie	932
20 Obsługa błędów – środowisko webowe	934
20.1 Przypomnienie – handlers PHP	935
20.2 Kontrakt ErrorHandler	935
20.3 Klasa CodeSnippet	936
20.4 Klasa TraceFrame	938
20.5 Klasa TraceCollection	939
20.6 Klasa Backtrace	941
20.7 Handler dla środowiska webowego	942
20.7.1 Klasa DetailedErrorHandler	942
20.7.2 Klasa FallbackErrorHandler	947
20.8 Widoki błędów w Blade	948
20.8.1 Widok debug	948
20.8.2 Widok 404	957
20.8.3 Widok 500	964
20.9 Formatter HTML	970
20.10Service Provider	973
20.11Struktura katalogów	975
20.12Testy jednostkowe	976

20.12.1 Test DetailedErrorHandler	976
20.12.2 Test FallbackErrorHandler	980
20.13 Podsumowanie	983
21 Obsługa błędów – środowisko konsolowe	985
21.1 Architektura konsolowego renderera	985
21.2 Klasy pomocnicze	986
21.2.1 PathNormalizer	986
21.2.2 TraceFrameFetcher	986
21.2.3 VariableFormatter	987
21.3 Renderery cząstkowe	989
21.3.1 BaseFrameRenderer	989
21.3.2 FrameDetailsRenderer	990
21.3.3 FrameSourceRenderer	992
21.3.4 FrameVariablesRenderer	992
21.3.5 CodeSnippetRenderer	993
21.3.6 ConsoleHeaderRenderer	995
21.3.7 ConsoleTraceRenderer	996
21.3.8 ConsoleEnvironmentRenderer	998
21.3.9 ConsoleHelpRenderer	999
21.4 Przetwarzanie komend	1000
21.4.1 ConsoleCommandProcessor	1000

21.4.2 ConsoleFrameRenderer	1002
21.5 Sesja debugowania	1004
21.5.1 DebugSession	1004
21.6 Fabryka i główny renderer	1005
21.6.1 ConsoleRendererFactory	1005
21.6.2 ConsoleRenderer	1009
21.7 Handler konsolowy	1011
21.7.1 ConsoleHandler	1011
21.8 Proces działania	1012
21.9 Przykładowy przebieg debugowania	1013
21.10 Struktura katalogów	1014
21.11 Testy jednostkowe	1015
21.11.1 Testowanie interaktywnej pętli	1015
21.12 Podsumowanie	1023
22 Sesja i ciasteczka	1024
22.1 Jak to działa?	1024
22.1.1 Sesje	1025
22.1.2 Ciasteczka	1025
22.1.3 Porównanie sesji i ciasteczek	1026
22.2 Szyfrowanie danych	1026
22.2.1 Podejście pierwotne – OpenSSL	1026

22.2.2	Czemu NONCE jest ważne	1027
22.2.3	Sodium – secretbox	1028
22.2.4	Sodium – AEAD	1029
22.3	Implementacja szyfrowania	1030
22.3.1	Generator kluczy	1030
22.3.2	Modyfikowanie pliku .env	1031
22.3.3	Komenda do generowania klucza	1032
22.3.4	Klasy walidacyjne	1033
22.3.5	Szyfrowanie i deszyfrowanie	1036
22.4	Handler do ciastek	1039
22.4.1	Metoda setcookie	1039
22.4.2	Klasa CookieOptions	1040
22.4.3	Klasa CookieManager	1041
22.5	Zarządzanie sesją	1043
22.5.1	Omówienie interfejsu SessionHandlerInterface	1044
22.5.2	Klasa SessionConfiguration	1044
22.5.3	Klasa SessionSecurity	1045
22.5.4	Klasa SessionManager	1046
22.6	Sesja oparta na plikach	1050
22.6.1	Zalety i wady	1050
22.6.2	Implementacja	1050
22.7	Sesja oparta na bazie danych	1053

22.7.1	Zalety i wady	1053
22.7.2	Komenda do tworzenia migracji	1053
22.7.3	Stub migracji	1055
22.7.4	Encja Session	1056
22.7.5	Implementacja handlera sesji	1058
22.8	Bezpieczeństwo aplikacji	1060
22.8.1	Session hijacking	1060
22.8.2	Zabezpieczenia	1061
22.9	Integracja z kontenerem DI	1062
22.10	Testy jednostkowe	1064
22.10.1	Testy szyfrowania	1064
22.10.2	Testy handlera plików	1067
22.10.3	Testy SessionManager	1070
22.11	Struktura katalogów	1074
22.12	Podsumowanie	1075
23	Wysyłka maili	1076
23.1	Protokoły pocztowe – wprowadzenie	1077
23.1.1	IMAP	1077
23.1.2	SMTP	1077
23.2	Architektura modułu	1078
23.3	Kontrakty	1078

23.3.1	TransportContract	1078
23.3.2	MailerContract	1079
23.3.3	MailHistoryLoggerContract	1080
23.3.4	Wyjątek TransportError	1081
23.4	Obsługa nagłówek wiadomości	1082
23.4.1	Klasa Address	1082
23.4.2	Klasa Content	1083
23.4.3	Klasa Envelope	1084
23.4.4	Klasa Email	1086
23.5	Warstwa transportu SMTP	1088
23.5.1	Value Objects	1088
23.5.2	SmtpConfig	1093
23.5.3	SmtpConnection	1094
23.5.4	SmtpCommand	1097
23.5.5	SmtpResponse	1099
23.5.6	SmtpResponseFactory	1101
23.5.7	Klasy walidacyjne	1104
23.5.8	SmtpTransport	1104
23.6	Fasada Mailable	1109
23.6.1	Przykładowe zastosowanie	1111
23.7	Klasa Mailer	1112
23.8	MailerFactory	1114

23.8.1 Czym jest MailHog?	1116
23.9 Logowanie wysłanych maili	1116
23.9.1 Komenda do tworzenia migracji	1116
23.9.2 Stub migracji	1117
23.9.3 Encja MailLog	1119
23.9.4 MailHistoryLogger	1121
23.10 Testy jednostkowe	1123
23.10.1 Test klasy Address	1123
23.10.2 Test SmtplibConfig	1125
23.10.3 Test SmtplibCommand	1127
23.10.4 Test MailerFactory	1130
23.11 Integracja z kontenerem DI	1133
23.11.1 Konfiguracja	1133
23.11.2 MailServiceProvider	1133
23.11.3 Rejestracja ServiceProvider	1135
23.11.4 Użycie w kontrolerze	1135
23.11.5 Użycie w komendzie konsolowej	1136
23.12 Struktura katalogów	1137
23.13 Podsumowanie	1138
24 Autoryzacja	1140
24.1 Architektura systemu	1140

24.2	Encje autoryzacji	1141
24.2.1	Encja User	1141
24.2.2	Encja Role	1147
24.2.3	Encja Permission	1150
24.3	Menadżery autoryzacji	1151
24.3.1	UserManager	1151
24.3.2	RoleManager	1155
24.3.3	PermissionManager	1156
24.4	Fasada Auth	1158
24.5	Middleware autoryzacji	1162
24.6	ServiceProvider	1163
24.7	Kontrolery autoryzacji	1164
24.7.1	DTO do walidacji	1164
24.7.2	RegisterController	1166
24.7.3	LoginController	1168
24.7.4	LogoutController	1170
24.8	Przykłady użycia	1171
24.8.1	Sprawdzanie ról w kontrolerze	1171
24.8.2	Sprawdzanie uprawnień	1172
24.8.3	Sprawdzanie wielu ról	1172
24.9	Struktura katalogów	1172
24.10	Podsumowanie	1173

25 Cache	1175
25.1 Wprowadzenie do Cache	1175
25.1.1 Rodzaje cache	1175
25.2 PSR-6 – Standard Cache Pool	1176
25.3 Architektura modułu	1176
25.4 Kontrakt StorageContract	1177
25.5 BaseStorage – warstwa abstrakcji	1178
25.6 FileStorage	1185
25.7 Redis	1191
25.7.1 Instalacja Redisa	1191
25.7.2 RedisEvictionPolicy	1191
25.7.3 RedisStorage	1192
25.8 CacheItemKeyValidator	1199
25.9 CacheItem	1200
25.10 CacheItemPool	1202
25.11 TaggedCache	1207
25.12 Fasada Cache	1211
25.13 Fabryki Storage	1217
25.13.1 StorageFactoryContract	1217
25.13.2 FileStorageFactory	1219
25.13.3 RedisStorageFactory	1220
25.13.4 MemcachedStorageFactory	1222

25.13.5 StorageFactory	1223
25.14 CacheServiceProvider	1225
25.15 Konfiguracja	1227
25.16 Przykłady użycia	1227
25.16.1 Podstawowe operacje	1227
25.16.2 Operacje atomowe (Redis)	1228
25.16.3 Tagged Cache	1228
25.16.4 Wiele store'ów	1229
25.17 Struktura katalogów	1229
25.18 Podsumowanie	1230
26 PSR-14 – System eventów	1231
26.1 Definicje pojęć	1231
26.1.1 Event	1231
26.1.2 Listener	1232
26.1.3 Dispatcher	1232
26.1.4 Listener Provider	1232
26.2 Implementacja PSR-14	1232
26.2.1 StoppableEvent	1232
26.2.2 EventDispatcher	1233
26.2.3 ListenerProvider	1234
26.3 Atrybut Listen	1237

26.4 ListenerDiscovery	1238
26.5 EventServiceProvider	1241
26.6 Wbudowane eventy	1242
26.6.1 Eventy Cache	1242
26.6.2 Eventy Database	1243
26.6.3 Eventy Framework	1244
26.6.4 Eventy Routing	1244
26.6.5 Eventy View	1244
26.7 Event UserRegistered	1245
26.8 Praktyczny przykład – Email powitalny	1245
26.8.1 Klasa WelcomeEmail Mailable	1245
26.8.2 Widok Blade dla emaila	1247
26.8.3 Listener wysyłający email	1250
26.8.4 Modyfikacja RegisterController	1252
26.8.5 Rejestracja listenera	1252
26.9 Priorytety listenerów	1252
26.10Zatrzymywanie propagacji	1253
26.11Struktura katalogów	1254
26.12Eventy poza PSR-14	1255
26.13Podsumowanie	1256

27.1	Architektura	1258
27.2	DataCollectorContract	1259
27.3	DebugBar	1259
27.4	Collectors	1261
27.4.1	QueryCollector – monitorowanie zapytań SQL	1261
27.4.2	CacheCollector	1264
27.4.3	PerformanceCollector	1267
27.4.4	RequestCollector	1268
27.4.5	RouteCollector	1270
27.4.6	ViewCollector	1271
27.4.7	TimelineCollector	1273
27.5	Emisja eventów w komponentach	1277
27.5.1	Application – eventy bootstrap	1277
27.5.2	View – eventy renderowania	1278
27.5.3	CacheItemPool – eventy cache	1279
27.5.4	QueryEventDispatcher – eventy SQL	1280
27.5.5	Router – event dopasowania trasy	1280
27.6	Middleware InjectDebugBar	1281
27.7	DebugBarServiceProvider	1283
27.8	Konfiguracja	1285
27.9	Widok Debugbar	1286
27.10	Przykład użycia	1287

27.11	Struktura katalogów	1288
27.12	Podsumowanie	1288
28	Kolejki i joby	1290
28.1	Kontrakty	1291
28.1.1	JobContract	1291
28.1.2	QueueContract	1291
28.2	Bazowa klasa joba	1292
28.2.1	Atrybut Serialize	1292
28.2.2	Klasa Job	1293
28.3	Przechowywanie jobów w bazie danych	1296
28.3.1	Zalety i wady	1296
28.3.2	Implementacja	1297
28.4	Przechowywanie jobów w Redisie	1299
28.4.1	Zalety i wady	1299
28.4.2	Implementacja	1300
28.5	Przetwarzanie jobów	1303
28.5.1	Klasa QueueWorker	1303
28.5.2	Klasa QueueFactory	1305
28.5.3	Komenda queue:work	1306
28.6	Obsługa nieudanych jobów	1308
28.7	Jak działa cron	1311

28.7.1	Crontab	1311
28.7.2	Przykład użycia	1312
28.8	Implementacja cron-jobs	1312
28.8.1	Enum CronSchedule	1312
28.8.2	Zalety podejścia z Enuma	1314
28.8.3	Parser wyrażeń crona	1314
28.8.4	Klasa ScheduledEvent	1317
28.8.5	Klasa Schedule	1318
28.8.6	Przetwarzanie crona	1321
28.8.7	Komenda schedule:run	1322
28.9	Dispatcher – wysyłanie jobów	1324
28.10	SchedulerServiceProvider	1326
28.11	Struktura katalogów	1328
28.12	Podsumowanie	1329
29	WebSockets	1331
29.1	Wprowadzenie do WebSockets	1331
29.1.1	Różnice między HTTP a WebSockets	1331
29.1.2	Zastosowania WebSockets	1332
29.2	Architektura modułowa	1332
29.3	Protokół WebSocket (RFC 6455)	1332
29.3.1	Handshake HTTP	1332

29.3.2 Opcodes	1332
29.4 Implementacja w Larafony	1333
29.4.1 Klasa Frame	1333
29.4.2 Encoder – kodowanie ramek	1335
29.4.3 Decoder – dekodowanie ramek	1341
29.4.4 Handshake	1347
29.4.5 FiberEngine – natywna implementacja PHP 8.5	1349
29.4.6 Server – wysoki poziom abstrakcji	1360
29.4.7 Connection	1366
29.5 Kontrakty (interfejsy)	1369
29.5.1 EngineContract	1369
29.5.2 ConnectionContract	1370
29.5.3 ServerContract	1370
29.6 ServiceProvider i DI	1371
29.7 Komenda konsolowa	1373
29.8 Paczka bridge: larafony/websocket-react	1375
29.9 Praktyczny przykład: Czat AI w demo-app	1384
29.9.1 Kontroler Inertia	1384
29.9.2 Listener obsługujący wiadomości	1385
29.9.3 ServiceProvider rejestrujący handlers	1388
29.9.4 Komponent Vue z WebSocket	1389
29.9.5 Przepływ danych	1396

29.10	Porównanie silników	1396
29.11	Porównanie z innymi frameworkami	1397
29.11.1	Laravel Reverb	1397
29.11.2	Symfony	1397
29.11.3	Dlaczego WebSockets w Larafony wyróżniają się	1398
29.12	Podsumowanie	1398
30	MCP – Model Context Protocol	1400
30.1	Wprowadzenie do MCP	1400
30.1.1	Dlaczego MCP?	1400
30.1.2	Architektura MCP	1401
30.2	Implementacja MCP w Larafony	1401
30.2.1	Oficjalne MCP SDK	1401
30.2.2	McpServerFactory	1402
30.2.3	Session Store z Redis	1404
30.2.4	McpServiceProvider	1407
30.3	Endpoint HTTP dla zdalnego dostępu	1410
30.3.1	McpController	1410
30.3.2	Konfiguracja	1412
30.3.3	Bootstrap aplikacji	1413
30.4	Komenda konsolowa	1415
30.5	Integracja z Claude Code	1417

30.5.1	Transport STDIO (lokalnie)	1417
30.5.2	Transport HTTP (zdalnie)	1417
30.6	Porównanie z innymi frameworkami	1418
30.7	Przepływ żądania MCP	1419
30.8	Bezpieczeństwo	1419
30.9	Paczka larafony/mcp-assistant	1419
30.9.1	MakeControllerTool – scaffolding przez AI	1420
30.9.2	FrameworkDocsResource – dokumentacja jako zasób	1422
30.9.3	LearnLarafonyPrompt – interaktywna nauka	1432
30.9.4	Dlaczego osobna paczka?	1436
30.10	Podsumowanie	1437
31	Porównanie z istniejącymi frameworkami	1438
31.1	Minimalna wersja PHP oraz wymagane zależności	1438
31.1.1	Stworzone rozwiązanie	1438
31.1.2	CodeIgniter 4	1439
31.1.3	Laravel 12	1439
31.1.4	Symfony 8	1439
31.2	Złożoność kodu	1440
31.2.1	CodeIgniter 4	1440
31.2.2	Laravel 12	1440
31.2.3	Symfony 8	1441

31.3	Automatyczne zależności	1441
31.3.1	CodeIgniter 4	1442
31.3.2	Laravel 12	1443
31.3.3	Symfony 8	1444
31.4	Routing i middleware	1445
31.4.1	CodeIgniter 4	1446
31.4.2	Laravel 12	1447
31.4.3	Symfony 8	1447
31.5	System zarządzania bazą danych	1448
31.5.1	CodeIgniter 4	1450
31.5.2	Laravel 12	1451
31.5.3	Symfony 8	1452
31.6	Aplikacje konsolowe	1453
31.6.1	CodeIgniter 4	1454
31.6.2	Laravel 12	1454
31.6.3	Symfony 8	1455
31.7	Widoki	1457
31.7.1	CodeIgniter 4	1458
31.7.2	Laravel 12	1458
31.7.3	Symfony 8	1459
31.8	Obsługa wyjątków	1460
31.8.1	CodeIgniter 4	1461

31.8.2	Laravel 12	1461
31.8.3	Symfony 8	1461
31.9	Sesja i ciastka	1461
31.9.1	CodeIgniter 4	1462
31.9.2	Laravel 12	1462
31.9.3	Symfony 8	1462
31.10	Wysyłka maili	1462
31.10.1	CodeIgniter 4	1463
31.10.2	Laravel 12	1464
31.10.3	Symfony 8	1465
31.11	Autoryzacja	1466
31.11.1	CodeIgniter 4	1466
31.11.2	Laravel 12	1467
31.11.3	Symfony 8	1467
31.12	Cache	1467
31.12.1	CodeIgniter 4	1467
31.12.2	Laravel 12	1468
31.12.3	Symfony 8	1468
31.13	System eventów	1468
31.13.1	CodeIgniter 4	1469
31.13.2	Laravel 12	1470
31.13.3	Symfony 8	1471

31.14	Kolejki	1472
31.14.1	CodeIgniter 4	1473
31.14.2	Laravel 12	1474
31.14.3	Symfony 8	1475
31.15	WebSocket	1476
31.15.1	CodeIgniter 4	1476
31.15.2	Laravel 12	1476
31.15.3	Symfony 8	1478
31.16	Model Context Protocol (MCP)	1479
31.16.1	CodeIgniter 4	1479
31.16.2	Laravel 12	1479
31.16.3	Symfony 8	1479
31.17	Podsumowanie	1480
32	Integracja z zewnętrznymi bibliotekami	1481
32.1	Monolog – zaawansowane logowanie	1482
32.1.1	Architektura integracji	1482
32.1.2	Fabryka loggerów	1483
32.1.3	Service Provider	1485
32.1.4	Konfiguracja	1486
32.1.5	Metody pomocnicze fabryki	1487
32.2	Symfony Mailer – profesjonalna wysyłka e-mail	1487

32.2.1	Adapter transportu	1488
32.2.2	Obsługiwane formaty DSN	1489
32.2.3	Service Provider	1490
32.2.4	Fabryka transportów	1491
32.3	Carbon – zaawansowana obsługa dat	1492
32.3.1	CarbonClock	1492
32.3.2	Rozszerzone możliwości	1494
32.3.3	Service Provider	1495
32.4	Guzzle HTTP – profesjonalny klient HTTP	1496
32.4.1	GuzzleHttpClient	1496
32.4.2	Fabryka klientów	1498
32.4.3	Middleware retry	1499
32.4.4	Service Provider	1500
32.5	Twig i Smarty – alternatywne silniki szablonów	1501
32.5.1	Twig	1501
32.5.2	Smarty	1504
32.5.3	Service Provider dla widoków	1505
32.6	PHP Debug Bar – wizualny debugger	1506
32.6.1	PhpDebugBar	1506
32.6.2	Adapter kolektorów Larafony	1508
32.6.3	Natywne kolektory	1510
32.6.4	Middleware do wstrzykiwania toolbara	1512

<i>SPIS TREŚCI</i>	56
--------------------	----

32.6.5 Service Provider	1513
-----------------------------------	------

32.7 Podsumowanie	1515
-----------------------------	------

Spis listingów	1545
-----------------------	-------------

Skorowidz	1546
---------------------	------

Rozdział 1

Wstęp

W tym rozdziale wprowadzimy kluczowe pojęcia, omówimy w skrócie dobre praktyki w inżynierii oprogramowania oraz zaprezentujemy wzorce projektowe, które zostały zastosowane. Przybliżymy również, czym jest PSR, dlaczego odgrywa istotną rolę w nowoczesnych aplikacjach, oraz omówimy wybrane standardy. Na zakończenie rozdziału skonfigurujemy środowisko pracy, przygotowując repozytorium oraz narzędzia do testowania, takie jak phpunit i phpstan.

1.1 Dobre praktyki

W inżynierii oprogramowania wymyślono wiele dobrych praktyk programistycznych. W przypadku tworzenia własnego frameworka szczególnie użyteczne będą zasady SOLID oraz DRY

Zasada DRY ("Don't Repeat Yourself") jest kluczowa w programowaniu obiektowym, ponieważ promuje tworzenie kodu, który jest bardziej czytelny, łatwiejszy w utrzymaniu i mniej podatny na błędy. Unikanie powtórzeń kodu oznacza, że każda informacja lub funkcjonalność jest zdefiniowana tylko raz, co ułatwia wprowadzanie zmian i aktualizacji bez ryzyka niespójności. W kontekście programowania obiektowego, DRY zachęca do wykorzystania mechanizmów takich jak dziedziczenie, kompozycja i polimorfizm, aby efektywnie zarządzać współdzielonymi zachowaniami i właściwościami między obiektami, co prowadzi do bardziej elastycznego i skalowalnego kodu.

1.1.1 SOLID

SOLID jest skrótem oznaczającym 5 zasad projektowania w programowaniu obiektowym wymyślonych przez Roberta C. Martina. Kolejne litery odnoszą się do następujących zasad

- **S** - Single-Responsibility Principle (SRP)
- **O** - Open-closed Principle (OCP)
- **L** - Liskov Substitution Principle (LSP)
- **I** - Interface Segregation Principle (ISP)
- **D** - Dependency Inversion Principle (DIP)

Rozważmy następujące klasy

```
1 <?php
2 declare(strict_types=1);
3
4 namespace Chapter1;
5
6 interface ShapeContract
7 {
8     public float $area { get; }
9 }
10
11 interface CalculatorContract
12 {
13     public function calculate(): float;
14 }
15
16 interface TextContract
17 {
18     public function text(): string;
19 }
20
21 interface JsonContract
22 {
```

```
23     public function json(): string;
24 }
25
26 class Square implements ShapeContract
27 {
28     public function __construct(private readonly int
29         $length){}
30
31     public float $area {
32         get => $this->length * $this->length;
33     }
34 }
35
36 class Circle implements ShapeContract
37 {
38     public function __construct(private readonly int
39         $radius){}
40
41     public float $area {
42         get => $this->radius * $this->radius * pi();
43     }
44 }
45
46 readonly class AreaCalculator implements
47     CalculatorContract
48 {
49     /**
50      * @param array<int, ShapeContract> $shapes
51      */
52     public function __construct(protected array $shapes)
53     {
54     }
55
56     public function calculate(): float
57     {
58         return array_map(fn(ShapeContract $shape) =>
59             $shape->area, $this->shapes)
60             |> array_sum(...);
61     }
62 }
```

```
60 readonly class BigAreaCalculator extends AreaCalculator
61 {
62     public function calculate(): float
63     {
64
65         $initial = parent::calculate();
66         //some additional logic and filters like not less
67         than
68         return $initial;
69     }
70 }
71 readonly class SumCalculatorOutputter implements
    TextContract, JsonContract
72 {
73     public function __construct(private CalculatorContract
        $calculator){}
74     public function text(): string
75     {
76         return "The sum of the areas of the shapes is: " .
            $this->calculator->calculate();
77     }
78
79     public function json(): string
80     {
81         return json_encode(['sum' => $this->calculator->
            calculate()]);
82     }
83 }
```

Listing 1.1: Zasady SOLID

Zasada jednej odpowiedzialności - SRP

Klasa powinna mieć tylko jedną odpowiedzialność. Jedyne co potrafią klasy `Square` i `Circle` to policzenie swojego pola. Zadaniem klasy `AreaCalculator` jest wyliczenie pola dla tablicy figur, i bez znaczenia jest tutaj, w jaki sposób każda z figur liczy swoje pole. Natomiast jedynym zadaniem klasy `SumCalculatorOutputter` jest wyświetlenie wyniku jako tekst lub json.

Zasada otwarte-zamknięte - OCP

Klasy powinny być otwarte na rozszerzenia i zamknięte na modyfikacje. Ponieważ klasy `Square` i `Circle` używają interfejsu do obliczenia swojego pola, dodanie nowej klasy (np trójkąta) nie wymaga od nas modyfikacji klasy kalkulatora. Dodatkowo użycie `array_map` gwarantuje, że wszystkie obiekty będą miały zaimplementowany getter dla pola `$area`.

Zasada podstawienia Liskova - LSP

Funkcje które używają wskaźników lub referencji do klas bazowych, muszą być w stanie używać również obiektów klas dziedziczących po klasach bazowych, bez dokładnej znajomości tych obiektów. Klasa `BigAreaCalculator` spełnia tę zasadę ponieważ nie ma większego znaczenia co dokładnie robi klasa bazowa

Zasada segregacji interfejsów - ISP

Wiele dedykowanych interfejsów jest lepsze niż jeden ogólny. Klasa `SumCalculatorOutputter` spełnia tę zasadę, ponieważ za każdy z rodzajów zwracanego wyjścia odpowiada dedykowany interfejs.

Zasada odwrócenia zależności - DIP

Wysokopoziomowe moduły nie powinny zależeć od modułów niskopoziomowych - zależności między nimi powinny wynikać z abstrakcji. Każdy z modułów spełnia tę zasadę. Kalkulatory w konstruktorze przyjmują interfejs dla kształtu, natomiast klasa do wyświetlania wyników przyjmuje w konstruktorze interfejs kalkulatora. Należy dodatkowo zwrócić uwagę, że wszystkie klasy otrzymują swoje zależności już w konstruktorze. Wykorzystamy to już w rozdziale trzecim gdzie zaimplementujemy kontener Dependency Injection stanowiący podstawę naszego frameworka.

1.1.2 YAGNI

YAGNI to akronim od angielskiego `You Aren't Gonna Need It`. W praktyce oznacza to, że do pliku `composer.json` dodajemy tylko te zależności, które są naprawdę niezbędne. W naszym przypadku będą to:

- Interfejsy PSR,

- biblioteka **Carbon** do zarządzania datami,
- biblioteka **Faker** do generowania przykładowych danych i fabryk modeli,
- biblioteka **Symfony/Mailer** do niskopoziomowej obsługi maili.

Takie podejście pozwala zminimalizować ryzyko potencjalnych luk w bezpieczeństwie oraz ogranicza nadmiarowy kod.

1.2 Wzorce projektowe

Wzorce projektowe są sprawdzonymi rozwiązaniami często występujących problemów w programowaniu obiektowym. Pomagają one w organizacji kodu w sposób, który jest czytelny, elastyczny i łatwy do rozwijania. W tej sekcji przyjrzemy się kilku popularnym wzorcom projektowym, które odgrywają kluczową rolę w tworzeniu skalowalnego i efektywnego oprogramowania.

1.2.1 Budowniczy

Wzorzec Budowniczy (Builder) służy do konstruowania złożonych obiektów krok po kroku. Jest szczególnie przydatny, gdy obiekt posiada wiele opcjonalnych parametrów lub wymaga skomplikowanej inicjalizacji. Dzięki zastosowaniu Budowniczego można oddzielić proces tworzenia obiektu od jego reprezentacji, co ułatwia modyfikacje i dodawanie nowych opcji.

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Database\Base\Query;
6
7 use Closure;
8 use Larafony\Framework\Database\Base\Contracts\
    ConnectionContract;
```

```
9 use Larafony\Framework\Database\Base\Query\Contracts\  
    GrammarContract;  
10 use Larafony\Framework\Database\Base\Query\Contracts\  
    QueryBuilderContract;  
11 use Larafony\Framework\Database\Base\Query\Enums\JoinType;  
12 use Larafony\Framework\Database\Base\Query\Enums\  
    OrderDirection;  
13  
14 /**  
15  * Base Query Builder - provides fluent API  
16  * Abstract class - no SQL building here, only state  
    management  
17  * Concrete implementations in drivers (MySQL, PostgreSQL,  
    etc.)  
18  *  
19  * @phpstan-consistent-constructor  
20  */  
21 abstract class QueryBuilder implements  
    QueryBuilderContract  
22 {  
23     protected GrammarContract $grammar;  
24     protected QueryDefinition $query;  
25  
26     public function __construct(protected readonly  
        ConnectionContract $connection)  
27     {  
28     }  
29  
30     abstract public function table(string $table): static;  
31  
32     /**  
33      * @param array<int, string> $columns  
34      */  
35     abstract public function select(array $columns):  
        static;  
36  
37     abstract public function where(string $column, string  
        $operator, mixed $value): static;  
38  
39     abstract public function orWhere(string $column,  
        string $operator, mixed $value): static;
```

```
40
41  /**
42   * @param array<int, mixed> $values
43   */
44  abstract public function whereIn(string $column, array
    $values): static;
45
46  /**
47   * @param array<int, mixed> $values
48   */
49  abstract public function whereNotIn(string $column,
    array $values): static;
50
51  abstract public function whereNull(string $column):
    static;
52
53  abstract public function whereNotNull(string $column):
    static;
54  abstract public function whereNested(Closure $callback
    , string $boolean): static;
55
56  /**
57   * @param array<int, mixed> $values
58   */
59  abstract public function whereBetween(string $column,
    array $values): static;
60
61  abstract public function whereLike(string $column,
    string $pattern): static;
62
63  abstract public function join(
64    string $table,
65    string|Closure $first,
66    ?string $operator = null,
67    ?string $second = null,
68    JoinType $type = JoinType::INNER,
69  ): static;
70
71  abstract public function leftJoin(
72    string $table,
73    string|Closure $first,
```

```
74         ?string $operator = null,  
75         ?string $second = null,  
76     ): static;  
77  
78     abstract public function rightJoin(  
79         string $table,  
80         string|Closure $first,  
81         ?string $operator = null,  
82         ?string $second = null,  
83     ): static;  
84  
85     //other abstract methods  
86 }
```

Listing 1.2: Przykład wzorca Budowniczy

Przy wszelkiego rodzaju frameworkach ORM (czyli mapowanie obiektowo-relacyjne *Object-relation mapping*) zapytania budowane są niczym z klocków Lego. Powyższa klasa *QueryBuilder* jest tego świetnym przykładem. Jej jedynym zadaniem jest stworzenie zapytania SQL z wykorzystaniem *Fluent Interface*. Klasa ta jest abstrakcyjna ponieważ definiuję ogólną strukturę SQL - konkretna implementacja znajduje się w klasach implementujących połączenie z bazą danych MySQL

1.2.2 Dekorator

Wzorzec Dekorator (Decorator) umożliwia dynamiczne rozszerzanie funkcjonalności obiektów bez potrzeby modyfikacji ich kodu. Dekorator jest często używany, aby dodać nowe zachowania do klasy w sposób elastyczny, bez konieczności korzystania z dziedziczenia. Pozwala to uniknąć nadmiernej ilości podklas i upraszcza utrzymanie kodu. Wzorzec ten jest niezwykle pomocny w osiągnięciu pierwszej zasady SOLID i tym samym złożoności klas wynoszącej 5 lub mniej.

Standardy PSR-7 oraz PSR-17 (dokładniej omówione w kolejnym rozdziale) dość szczegółowo opisują operacje jakie można wykonać na wgrywanym pliku. Dekoratory są tutaj świetnym przykładem. Spójrzmy na poniższe klasy

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Http\UploadedFile\Decorators;
6
7 use Psr\Http\Message\StreamInterface;
8 use Psr\Http\Message\UploadedFileInterface;
9
10 abstract class UploadedFileDecorator implements
    UploadedFileInterface
11 {
12     public function __construct(
13         protected readonly UploadedFileInterface $file,
14     ) {
15     }
16
17     public function getStream(): StreamInterface
18     {
19         return $this->file->getStream();
20     }
21
22     public function moveTo(string $targetPath): void
23     {
24         $this->file->moveTo($targetPath);
25     }
26
27     public function getSize(): ?int
28     {
29         return $this->file->getSize();
30     }
31
32     public function getError(): int
33     {
34         return $this->file->getError();
35     }
36
37     public function getClientFilename(): ?string
38     {
39         return $this->file->getClientFilename();
40     }
```

```
41  
42     public function getClientMediaType(): ?string  
43     {  
44         return $this->file->getClientMediaType();  
45     }  
46 }
```

Listing 1.3: Bazowy dekorator PSR-7

Klasa ta stanowi bazową logikę dla pozostałych jak ten poniżej sprawdzający że plik może być przeniesiony tylko raz

```
1  <?php  
2  
3  declare(strict_types=1);  
4  
5  namespace Larafony\Framework\Http\UploadedFile\Decorators;  
6  
7  class MoveStatusDecorator extends UploadedFileDecorator  
8  {  
9      private bool $moved = false;  
10  
11     public function moveTo(string $targetPath): void  
12     {  
13         if ($this->moved) {  
14             throw new \RuntimeException('Cannot move file;  
15                 already moved!');  
16         }  
17         parent::moveTo($targetPath);  
18         $this->moved = true;  
19     }  
20 }
```

Listing 1.4: Dekorator MoveStatusDecorator

Użycie tych dekoratów jest wyjątkowo proste i odbywa się przy pomocy wzorca

Fabryka

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Http\Factories;
6
7 use Larafony\Framework\Http\UploadedFile;
8 use Larafony\Framework\Http\UploadedFile\Decorators\
    ErrorValidatorDecorator;
9 use Larafony\Framework\Http\UploadedFile\Decorators\
    MoveStatusDecorator;
10 use Larafony\Framework\Http\UploadedFile\Decorators\
    PathValidatorDecorator;
11 use Psr\Http\Message\StreamInterface;
12 use Psr\Http\Message\UploadedFileFactoryInterface;
13 use Psr\Http\Message\UploadedFileInterface;
14
15 final readonly class UploadedFileFactory implements
    UploadedFileFactoryInterface
16 {
17     public function createUploadedFile(
18         StreamInterface $stream,
19         ?int $size = null,
20         int $error = \UPLOAD_ERR_OK,
21         ?string $clientFilename = null,
22         ?string $clientMediaType = null,
23     ): UploadedFileInterface {
24         return $this->create($stream, $size ?? 0, $error,
25             $clientFilename, $clientMediaType);
26     }
27
28     public static function create(
29         StreamInterface|string $streamOrFile,
30         int $size,
31         int $error,
32         ?string $clientFilename = null,
33         ?string $clientMediaType = null,
34     ): UploadedFileInterface {
```

```
34         $file = new UploadedFile($streamOrFile, $size,  
35             $error, $clientFilename, $clientMediaType);  
36         $file = new ErrorValidatorDecorator($file);  
37         $file = new MoveStatusDecorator($file);  
38         return new PathValidatorDecorator($file);  
39     }  
40 }
```

Listing 1.5: klasa UploadedFileFactory

1.2.3 Fabryka

Wzorzec Fabryka (Factory) pozwala na tworzenie obiektów bez bezpośredniego używania operatora `new`. Zamiast tego, cały proces jest zarządzany przez metodę fabrykującą, co umożliwia elastyczną kontrolę nad tworzonymi instancjami i unikanie ścisłych zależności między klasami.

Pierwszy przykład tego wzorca został pokazany powyżej. Ale fabryki to nie tylko PSR-17. Fabryki są jednym z najczęściej używanych w wzorców w Larafony, ponieważ przy pomocy fabryk i konfiguracji (rozdział 5) możemy tworzyć dowolny sterownik (ang *Driver*) dla dowolnego fragmentu logiki. Poniżej przykład dla Cache z rozdziału 25

```
1  <?php  
2  
3  declare(strict_types=1);  
4  
5  namespace Larafony\Framework\Cache\Factories;  
6  
7  use Larafony\Framework\Cache\CacheItemPool;  
8  use Larafony\Framework\Cache\Contracts\StorageContract;  
9  use Larafony\Framework\Config\Contracts\ConfigContract;  
10 use Psr\Cache\CacheItemPoolInterface;  
11  
12 class StorageFactory
```

```
13 {
14     public function __construct(
15         private ConfigContract $config,
16     ) {
17     }
18
19     public function create(string $storeName):
20         CacheItemPoolInterface
21     {
22         $storeConfig = $this->config->get('cache.stores.'
23             . $storeName);
24         $configArray = $storeConfig instanceof \
25             ArrayObject ? $storeConfig->getArrayCopy() : (
26                 array) $storeConfig;
27
28         $driver = $configArray['driver'] ?? throw new \
29             InvalidArgumentException(
30                 "Cache store '{$storeName}' must have 'driver'
31                 key in config (file, redis, or memcached)"
32             );
33
34         $storage = match ($driver) {
35             'file' => $this->createFileStorage(
36                 $configArray),
37             'memcached' => $this->createMemcachedStorage(
38                 $configArray),
39             'redis' => $this->createRedisStorage(
40                 $configArray),
41             default => throw new \InvalidArgumentException
42                 ("Unsupported cache driver: {$driver}"),
43         };
44         return new CacheItemPool($storage);
45     }
46     /**
47      * @param array<string, mixed> $config
48      */
49     private function createFileStorage(array $config):
50         StorageContract
51     {
52         return new FileStorageFactory()->create($config);
53     }
54 }
```

```
43
44     /**
45      * @param array<string, mixed> $config
46      */
47     private function createMemcachedStorage(array $config)
48         : StorageContract
49     {
50         if (! extension_loaded('memcached')) {
51             throw new \RuntimeException('Memcached
52                 extension is not loaded');
53         }
54         return new MemcachedStorageFactory()->create(
55             $config);
56     }
57
58     /**
59      * @param array<string, mixed> $config
60      */
61     private function createRedisStorage(array $config):
62         StorageContract
63     {
64         if (! extension_loaded('redis')) {
65             throw new \RuntimeException('Redis extension
66                 is not loaded');
67         }
68         return new RedisStorageFactory()->create($config);
69     }
70 }
```

Listing 1.6: klasa StorageFactory

1.2.4 Fasada

Wzorzec Fasada (Facade) dostarcza uproszczony interfejs do bardziej złożonego systemu klas lub skomplikowanego API. Fasada ukrywa złożoność systemu i umożliwia klientom łatwiejszy dostęp do jego funkcji bez konieczności głębszego zrozumienia wewnętrznych szczegółów. Wzorzec ten będzie najczęściej

wykorzystywany w tworzonym frameworku. Główna klasa danego modułu będzie zazwyczaj fasadą dla wielu innych klas tworzących dany moduł

Bardzo ciekawym przykładem wzorca Fasady jest standard PSR-3 (omówiony w rozdziale 14). Standard ten definiuje jeden interfejs `LoggerInterface` który jest właściwie instrukcją co powinna zawierać każda fasada logująca (czy to do bazy danych, czy plików, czy logów systemowych).

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Log;
6
7 use Larafony\Framework\Container\Contracts\
    ContainerContract;
8 use Larafony\Framework\Web\Application;
9 use Psr\Log\LoggerInterface;
10
11 final readonly class Log
12 {
13     private LoggerInterface $logger;
14     public function __construct(private ContainerContract
        $container)
15     {
16         $this->logger = $this->container->get(
            LoggerInterface::class);
17     }
18
19
20     /**
21      * @param string $message
22      * @param array<int|string, mixed> $context
23      *
24      * @return void
25      */
26     public function emergency(string $message, array
        $context = []): void
27     {
28         $this->logger->emergency($message, $context);
```

```
29     }
30
31     /**
32      * @param string $message
33      * @param array<int|string, mixed> $context
34      *
35      * @return void
36      */
37     public function alert(string $message, array $context
38         = []): void
39     {
40         $this->logger->alert($message, $context);
41     }
42
43     /**
44      * @param string $message
45      * @param array<int|string, mixed> $context
46      *
47      * @return void
48      */
49     public function critical(string $message, array
50         $context = []): void
51     {
52         $this->logger->critical($message, $context);
53     }
54
55     /**
56      * @param string $message
57      * @param array<int|string, mixed> $context
58      *
59      * @return void
60      */
61     public function error(string $message, array $context
62         = []): void
63     {
64         $this->logger->error($message, $context);
65     }
66
67     /**
68      * @param string $message
69      * @param array<int|string, mixed> $context
```

```
67      *
68      * @return void
69      */
70      public function warning(string $message, array
        $context = []): void
71      {
72          $this->logger->warning($message, $context);
73      }
74
75      /**
76      * @param string $message
77      * @param array<int|string, mixed> $context
78      *
79      * @return void
80      */
81      public function notice(string $message, array $context
        = []): void
82      {
83          $this->logger->notice($message, $context);
84      }
85
86      /**
87      * @param string $message
88      * @param array<int|string, mixed> $context
89      *
90      * @return void
91      */
92      public function info(string $message, array $context =
        []): void
93      {
94          $this->logger->info($message, $context);
95      }
96
97      /**
98      * @param string $message
99      * @param array<int|string, mixed> $context
100      *
101      * @return void
102      */
103      public function debug(string $message, array $context
        = []): void
```

```
104     {  
105         $this->logger->debug($message, $context);  
106     }  
107 }
```

Listing 1.7: Fasada Log

1.3 PSR

PSR (PHP Standard Recommendations) to zestaw zaleceń mających na celu ujednolicenie stylu i praktyk kodowania w ekosystemie PHP. Każdy standard określony w ramach PSR wpływa na różne aspekty tworzenia aplikacji, począwszy od organizacji kodu po bardziej złożone zagadnienia jak zarządzanie żądaniami HTTP czy wykorzystanie wzorców projektowych. PSR jest kluczowy, ponieważ zapewnia spójność i przewidywalność w różnych projektach, ułatwiając współpracę między deweloperami oraz integrację zewnętrznych bibliotek.

Każdy standard PSR jest rozwijany przez PHP-FIG (PHP Framework Interoperability Group) i jest szeroko stosowany w całym ekosystemie PHP. Zrozumienie tych standardów i ich implementacja jest kluczowa dla tworzenia nowoczesnych, skalowalnych aplikacji.

Wszystkie z wymienionych poniżej standardów zostaną wykorzystane w tworzonego frameworku, z wyjątkiem PSR-12, którego będziemy trzymać się na wszystkich listingach

1.3.1 PSR-3, tworzymy logi

PSR-3 definiuje interfejs do tworzenia logów w aplikacjach PHP. Standard ten wprowadza zestaw metod, które mogą być zaimplementowane przez różne biblioteki logowania, co pozwala na wymienne stosowanie różnych implementacji w zależności od potrzeb projektu. Główne metody PSR-3 to np. `emergency`, `alert`, `critical`, `error`, `warning`, `notice`, `info` oraz `debug`.

Implementacja PSR-3 pozwala na lepsze monitorowanie działania aplikacji, identyfikację potencjalnych problemów oraz analizę działania aplikacji w warunkach produkcyjnych. Dzięki spójności interfejsu możemy łatwo podmienić mechanizm logowania na inny, bez konieczności modyfikacji kodu, co zapewnia większą elastyczność i modularność.

1.3.2 PSR-4 i autoloading

PSR-4 wprowadza standard dla automatycznego ładowania klas (autoloading) w PHP, który jest kluczowy dla efektywnej organizacji kodu w większych projektach. Dzięki PSR-4 każda klasa w projekcie jest mapowana na strukturę katalogów, co pozwala na automatyczne wczytywanie klas bez potrzeby ręcznego `require` lub `include`.

Korzystanie z PSR-4 znacznie ułatwia zarządzanie kodem i wspiera lepszą organizację przestrzeni nazw (`namespace`). Umożliwia to także łatwiejszą współpracę z zewnętrznymi bibliotekami oraz frameworkami, ponieważ wszystkie przestrzenie nazw są jasno zdefiniowane i mogą być ładowane w sposób automatyczny.

W kontekście dużych projektów, takich jak frameworki MVC (Model-View-Controller), stosowanie standardów PSR, a szczególnie PSR-4, jest bardzo ważne. Frameworki MVC są złożone i składają się z wielu komponentów. Standardy takie jak PSR-4 pomagają utrzymać kod w porządku, ułatwiają nawigację i zrozumienie struktury projektu.

1.3.3 PSR-6 cache

PSR-6 to standard opracowany przez PHP-FIG, definiujący interfejsy dla systemów cache'owania w PHP. Jego celem jest ujednolicenie sposobu zarządzania pamięcią podręczną w aplikacjach PHP, niezależnie od konkretnej implementacji.

Główne pojęcia w PSR-6 obejmują:

- **Cache Pool:** Główna jednostka zarządzania pamięcią podręczną, która umożliwia przechowywanie i pobieranie obiektów cache'owanych.

- **Cache Item:** Pojedynczy element pamięci podręcznej, zawierający dane oraz metadane, takie jak czas wygaśnięcia.
- **Tagi i TTL:** PSR-6 pozwala na dodawanie tagów do elementów cache'owanych oraz określanie czasu ich życia (*Time-To-Live*).

1.3.4 PSR-7, requesty HTTP

PSR-7 definiuje standard dotyczący zarządzania żądaniami i odpowiedziami HTTP w aplikacjach PHP. Standard ten wprowadza interfejsy dla klas, które reprezentują zarówno żądania, jak i odpowiedzi HTTP, pozwalając na łatwe manipulowanie nimi podczas przetwarzania w aplikacji. Kluczowe pojęcia związane z PSR-7 to `RequestInterface`, `ResponseInterface`, `StreamInterface` oraz `UriInterface`.

PSR-7 jest często stosowany w ramach aplikacji webowych, szczególnie tych opartych na wzorcu middleware, gdzie każde żądanie jest przetwarzane na różnych poziomach. Dzięki standaryzacji, aplikacje mogą bez problemu integrować różne biblioteki, które również implementują ten standard.

1.3.5 PSR-11, dodajemy kontener zależności

PSR-11 definiuje interfejs dla kontenerów zależności (Dependency Injection Containers). Kontener zależności to kluczowy element architektury wielu aplikacji PHP, który umożliwia efektywne zarządzanie zależnościami między klasami. Dzięki PSR-11 różne kontenery zależności mogą być wymiennie stosowane w aplikacjach, pod warunkiem, że implementują wymagane interfejsy.

Użycie PSR-11 sprawia, że kod staje się bardziej modułarny i testowalny. Łatwość zarządzania zależnościami sprawia, że można dynamicznie dostarczać różne implementacje klas, co zwiększa elastyczność projektu i pozwala na lepszą separację odpowiedzialności.

1.3.6 PSR-12 czyli jakość kodu

PSR-12 rozszerza i doprecyzowuje wcześniejsze zalecenia dotyczące stylu kodowania w PHP, opierając się na standardach wprowadzonych przez PSR-1 i PSR-2. Określa on szczegółowe wytyczne dotyczące formatowania kodu, takie jak użycie odstępów, strukturę deklaracji klas i funkcji, a także formatowanie bloków kontrolnych.

Wprowadzenie PSR-12 ma na celu poprawę czytelności kodu oraz ułatwienie jego przeglądania przez różnych deweloperów. Zgodność z tym standardem sprawia, że kod jest bardziej spójny, co w długiej perspektywie poprawia jego utrzymywalność.

1.3.7 PSR-14, event dispatcher

PSR-14 definiuje interfejsy dla mechanizmu eventów, umożliwiając aplikacjom PHP korzystanie z wzorca projektowego Observer. Standard ten określa, w jaki sposób można rejestrować, wysyłać i przetwarzać zdarzenia w aplikacji. Mechanizm eventów pozwala na lepsze zarządzanie komunikacją wewnętrzną w aplikacji, szczególnie w przypadku skomplikowanych projektów, gdzie różne moduły muszą reagować na określone zdarzenia.

Stosowanie PSR-14 pozwala na luźne powiązanie między komponentami aplikacji, co ułatwia wprowadzanie nowych funkcji i modyfikacje bez naruszania struktury istniejącego kodu.

1.3.8 PSR-15, dodajemy middleware

PSR-15 wprowadza standard dotyczący middleware w aplikacjach PHP. Middleware to warstwy, które pośredniczą w przetwarzaniu żądań HTTP, umożliwiając np. autoryzację, walidację danych czy modyfikację żądania. PSR-15 definiuje interfejsy dla klas middleware, co pozwala na tworzenie spójnych i łatwo rozszerzalnych rozwiązań.

Dzięki implementacji PSR-15 możliwe jest tworzenie modułowych aplikacji, w których poszczególne warstwy logiki mogą być dodawane lub modyfikowane

bez potrzeby ingerencji w cały kod aplikacji. Middleware to kluczowy element w nowoczesnych frameworkach PHP, takich jak Symfony czy Laravel.

1.3.9 PSR-17, HTTP factory

PSR-17 definiuje interfejsy dla fabryk (factory) tworzących obiekty związane z HTTP, takie jak żądania (`RequestFactoryInterface`), odpowiedzi (`ResponseFactoryInterface`) i strumienie (`StreamFactoryInterface`). Dzięki temu standardowi możliwe jest tworzenie obiektów HTTP w sposób spójny i elastyczny, co jest szczególnie przydatne w dużych aplikacjach, które muszą dynamicznie reagować na różne rodzaje żądań.

PSR-17 pozwala na unifikację procesu tworzenia obiektów HTTP, co zwiększa interoperacyjność między różnymi bibliotekami i frameworkami. To podejście sprzyja budowaniu elastycznych i skalowalnych rozwiązań webowych.

1.3.10 PSR-20, Clock

PSR-20 wprowadza standaryzowany interfejs zegara (`ClockInterface`), którego zadaniem jest dostarczanie aktualnego czasu w sposób kontrolowany oraz testowalny. Zegar systemowy nie jest już traktowany jako globalny zasób dostępny poprzez funkcje takie jak `time()` czy `new DateTimeImmutable()`, lecz jako zależność przekazywana do klas, które tego czasu potrzebują.

To podejście znacząco poprawia testowalność kodu. Dzięki wykorzystaniu kontraktu zegara możemy w prosty sposób zamienić rzeczywisty czas na zegar testowy (np. zamrożony lub manipulowany), co pozwala na deterministyczne testowanie operacji zależnych od czasu. Zamiast polegać na nieprzewidywalnych wartościach systemowych, aplikacja korzysta z zegara, który można kontrolować.

PSR-20 eliminuje również powtarzający się problem „ukrytej zależności systemu” — czasu, który dotychczas był pobierany w różnych miejscach aplikacji bez możliwości jego ujednolicenia. Dzięki temu standardowi klasy stają się bardziej spójne, a logika czasu jest wyraźnie oddzielona od logiki biznesowej.

Obecnie najpopularniejsza biblioteką implementującą ten standard jest `Carbon` i integracja z tą biblioteką zostanie pokazana przy okazji innych mostów

w końcowej części książki. Natomiast już w rozdziale 3 zostania pokazana edukacyjna, lecz wystarczająca na potrzeby niniejszego ebooka, implementacja PSR-20 wykorzystywana konsekwentnie w całym frameworku.

1.4 Zarządzanie zależnościami

Composer jest standardowym narzędziem do zarządzania zależnościami w PHP. Umożliwia on deklarowanie bibliotek wymaganych przez projekt, instalowanie ich, aktualizowanie oraz kontrolowanie zgodności wersji. W ostatnich latach Composer rozwinął się również w kierunku narzędzia zapewniającego bezpieczeństwo środowiska pracy.

Od wersji 2.4 (2022), Composer włączył mechanizmy znane wcześniej z pakietu `roave/security-advisories` do polecenia `composer audit`, które pozwala wykrywać znane podatności w zainstalowanych bibliotekach.

Jednak dopiero wersja 2.9 - wymagana przez PHP 8.5 i wydana 13 listopada 2025 roku — wprowadziła najbardziej rygorystyczną zmianę: **domyślne blokowanie instalacji pakietów posiadających znane podatności bezpieczeństwa (CVE¹)**. Oznacza to, że uruchomienie polecenia `composer install` może zostać przerwane, jeśli którakolwiek z zależności w projekcie znajduje się na liście zagrożeń.

Jest to ogromny krok naprzód w kierunku bezpieczeństwa ekosystemu PHP i znacząco podnosi poziom kontroli nad wykorzystywanymi bibliotekami. Można śmiało powiedzieć, że od końca 2025 roku Composer pełni rolę nie tylko menedżera pakietów, lecz także warstwy bezpieczeństwa dla całych projektów.

1.4.1 Zarządzanie zależnościami

Composer nie jest menedżerem pakietów w tradycyjnym sensie, takim jak Yum czy Apt. Zamiast tego zarządza bibliotekami na poziomie konkretnego projektu, instalując je w katalogu (np. `vendor`) wewnątrz projektu. Domyślnie nic nie jest instalowane globalnie, co czyni Composer menedżerem zależności. Jed-

¹Common Vulnerabilities and Exposures

nak dla wygody dostępna jest możliwość instalacji globalnej za pomocą komendy `global`.

Idea ta nie jest nowa i Composer czerpie silne inspiracje z narzędzi takich jak `npm` dla Node.js czy `Bundler` dla Ruby.

1.4.2 Przykład użycia

Założmy, że masz projekt, który zależy od wielu bibliotek. Niektóre z tych bibliotek również mają swoje zależności.

Composer umożliwia:

- deklarowanie bibliotek, od których zależy projekt,
- określenie, które wersje pakietów mogą i powinny zostać zainstalowane, a następnie ich instalację (czyli pobieranie do projektu),
- aktualizację wszystkich zależności jednym poleceniem.

Więcej informacji można znaleźć na stronie: <https://getcomposer.org/doc/00-intro.md>

1.5 Konfiguracja lokalnego repozytorium

Nota autora

WAŻNE!

Poniższa sekcja bazuje na pierwszej wersji kursu. W kodzie, który znajdziesz na GitHubie, implementacja może minimalnie różnić się od prezentowanej tutaj. Wynika to z faktu, że aplikacja demonstracyjna została wydzielona do osobnego repozytorium (co nastąpi formalnie w rozdziale 18).

Na potrzeby tej książki nie ma to jednak większego znaczenia. Kluczowa jest jedynie struktura: **framework** znajduje się w **jednym folderze**, a **przykładowa aplikacja** — wraz z plikiem `public/index.php` -

w **drugim**. Jest to istotne, ponieważ w kolejnych rozdziałach do katalogu aplikacji będą stopniowo dodawane nowe elementy, takie jak konfiguracja czy migracje bazy danych.

Autor zakłada - z pełnym optymizmem — że w erze wszechobecnych chatbotów AI (bez wymieniania nazw, aby przypadkiem nikomu nie robić reklamy), Drogi Czytelniku, **poradzisz sobie bez problemu!**

1.5.1 Utworzenie struktury katalogów

Zacznijmy od utworzenia podstawowej struktury katalogów projektu. W konsoli (lub terminalu) wykonaj polecenie:

```
1 mkdir -p my_project/{public,app,framework}
```

1.5.2 Utworzenie głównego pliku projektu

Przejdź do katalogu `public` i utwórz plik `index.php`:

```
1 touch my_project/public/index.php
```

Możesz edytować ten plik i dodać podstawowy kod PHP, np.:

```
1 <?php
2 phpinfo();
```

1.5.3 Utworzenie pustego repozytorium w katalogu framework

W katalogu `framework` utwórz puste repozytorium:

```
1 cd my_project/framework
2 git init
```

1.5.4 Dodanie pliku `composer.json`

W katalogu `framework` utwórz plik `composer.json`:

```
1 touch composer.json
```

W pliku `composer.json` zdefiniuj minimalną strukturę:

```
1 {
2     "name": "my_project/framework",
3     "description": "Framework for my PHP project",
4     "autoload": {
5         "psr-4": {
6             "Framework\\": "src/"
7         }
8     }
9 }
```

1.5.5 Dodanie pakietu `symfony/var-dumper`

Paczka `symfony/var-dumper` oferuje znacznie bardziej czytelne i wygodne wyświetlanie informacji o zmiennych w porównaniu ze standardowymi funkcjami `var_dump` i `print_r`. Obie te funkcje wyświetlają dane bez formatowania, co w przypadku obiektów czy wielowymiarowych tablic szybko prowadzi do chaosu. Natomiast `VarDumper` renderuje struktury z kolorowaniem składni, czytelnymi wcięciami i interaktywnymi elementami, co znacząco ułatwia debugowanie.

Przejdź teraz do głównego katalogu projektu i utwórz plik `composer.json`:

Nota autora

WAŻNE - Uwaga nr 1

Choć paczki `VarDumper` sam często używałem (jest wielokrotnie wygodniejsza niż klasyczny `var_dump`), nie została ona umieszczona w kodzie samego frameworka. Natomiast w repozytorium aplikacji demonstracyjnej pakiet ten znajduje się już w sekcji `require-dev`.

WAŻNE - Uwaga nr 2

Tworząc pierwszą edycję kursu błędnie założyłem, że sam `dump()` z paczki `Symfony` wystarczy do debugowania błędów. Nic bardziej mylnego.

Dlatego tuż po niniejszym wstępie (w kolejnym rozdziale) przedstawiony zostanie prosty, ale wyjątkowo skuteczny handler błędów, który w dalszej części książki zostanie rozbudowany do interaktywnego debuggera - zarówno w środowisku webowym, jak i konsolowym.

Może Cię, Drogi Czytelniku, zdziwić pojawienie się w kodzie „tras” celowo rzucających błądem. Jest to mój własny mechanizm testowy, który zachowałem nawet po kolejnych refaktorach. Nie jest on wymagany (i w odpowiednim momencie zostanie usunięty), ale gorąco zachęcam, aby go na razie pozostawić — jest bardzo pomocny, szczególnie podczas eksperymentów, do których szczerze zachęcam.

```
1 cd ..  
2 touch composer.json
```

Dodaj do pliku następującą zawartość, aby dodać lokalne repozytorium frameworka oraz pakiet `symfony/var-dumper`:

```
1 {
2     "repositories": [
3         {
4             "type": "path",
5             "url": "./framework"
6         }
7     ],
8     "require-dev": {
9         "my_project/framework": "*",
10        "symfony/var-dumper": "^7.4"
11    }
12 }
```

1.5.6 Instalacja zależności

Aby zainstalować zależności, uruchom w terminalu:

```
1 composer install
```

To polecenie pobierze wszystkie wymagane pakiety i zainstaluje je w katalogu `vendor`.

1.5.7 Konfiguracja vHosta

Konfiguracja wirtualnych hostów (vHostów) jest kluczowa dla hostowania wielu stron na jednym serwerze. Poniżej przedstawiam przykładowe pliki konfiguracyjne dla Apache i Nginx. Upewnij się, że ścieżka podana w plikach odpowiada

miejscu, gdzie zostały utworzone foldery w krokach powyżej. Przed rozpoczęciem konfiguracji, upewnij się, że PHP 8.4 FPM jest zainstalowane:

```
1 sudo apt update
2 sudo apt install php8.5-fpm
```

dodaj do pliku `/etc/hosts`

```
1 127.0.0.1 framework.local
```

Apache

Stwórz plik konfiguracyjny dla vHosta:

```
1 sudo touch /etc/apache2/sites-available/framework.local.
   conf
```

Dodaj następującą konfigurację:

```
1 <VirtualHost *:80>
2     ServerName framework.local
3     DocumentRoot /var/www/framework/public
4     <Directory /var/www/framework/public>
5         Options Indexes FollowSymLinks
6         AllowOverride All
7         Require all granted
8     </Directory>
9     <FilesMatch \.php$>
10         SetHandler "proxy:unix:/run/php/php8.5-fpm.sock|
           fcgi://localhost/"
```

```
11     </FilesMatch>
12     ErrorLog ${APACHE_LOG_DIR}/framework.local_error.log
13     CustomLog ${APACHE_LOG_DIR}/framework.local_access.log
        combined
14 </VirtualHost>
```

Aktywuj vHosta i przeładuj Apache:

```
1 sudo a2ensite framework.local.conf
2 sudo systemctl reload apache2
```

Nginx Stwórz plik konfiguracyjny dla vHosta:

```
1 sudo touch /etc/nginx/sites-available/framework.local
```

Dodaj następującą konfigurację:

```
1 server {
2     listen 80;
3     server_name framework.local;
4     root /var/www/framework/public;
5     index index.php index.html index.htm;
6     location / {
7         try_files $uri $uri/ =404;
8     }
9     location ~ \.php$ {
10         include snippets/fastcgi-php.conf;
11         fastcgi_pass unix:/run/php/php8.5-fpm.sock;
12     }
13     error_log /var/log/nginx/framework.local_error.log;
14     access_log /var/log/nginx/framework.local_access.log;
```

15 }

Aktywuj vHosta i przeładuj Nginx:

```
1 sudo ln -s /etc/nginx/sites-available/framework.local /etc
   /nginx/sites-enabled/
2 sudo systemctl reload nginx
```

1.6 Punkt wejścia aplikacji

Konfiguracja pokazana w tym rozdziale zakłada jeden punkt wejścia aplikacji - plik `public/index.php`

Współczesne aplikacje internetowe często korzystają z jednego punktu wejściowego, zazwyczaj pliku `public/index.php`. To podejście, popularne w wielu frameworkach takich jak Laravel, Symfony czy Zend, ma wiele zalet, które przyczyniają się do jego powszechnego stosowania.

1.6.1 Zalety Jednego Entry Point

Pierwszą i najważniejszą zaletą jednego punktu wejściowego jest centralizacja zarządzania ruchem sieciowym. Cały ruch przychodzący jest kierowany do jednego pliku, co umożliwia łatwe zarządzanie routowaniem, autoryzacją i innymi aspektami aplikacji. Dzięki temu, można w sposób efektywny kontrolować dostęp do różnych zasobów i zapewnić spójność w obsłudze żądań.

Drugą istotną zaletą jest bezpieczeństwo. Posiadanie jednego punktu wejściowego pozwala na implementację globalnych mechanizmów bezpieczeństwa, takich jak filtrowanie wejść, ochrona przed atakami typu SQL Injection czy XSS. Dodatkowo, ułatwia to wdrażanie polityk bezpieczeństwa na poziomie serwera,

takich jak ograniczenie dostępu do określonych plików czy katalogów.

1.7 PHPUnit

PHPUnit to jeden z najpopularniejszych frameworków do testowania jednostkowego w PHP. Umożliwia on pisanie i uruchamianie testów jednostkowych, które pomagają w zapewnieniu, że poszczególne części aplikacji działają zgodnie z oczekiwaniami.

Podstawowe założenia testowania:

- **Arrange (Przygotowanie):** Przygotowanie środowiska testowego, w tym tworzenie obiektów i inicjalizacja danych.
- **Act (Działanie):** Wykonanie operacji, która ma być testowana.
- **Assert (Sprawdzenie):** Weryfikacja, czy wynik operacji jest zgodny z oczekiwaniami.

1.8 phpstan

phpstan to narzędzie do analizy statycznej kodu PHP, które pomaga wykrywać błędy bez potrzeby uruchamiania kodu. phpstan analizuje kod na różnych poziomach szczegółowości, od 0 do 9, gdzie każdy kolejny poziom jest bardziej restrykcyjny i dokładny.

Poziomy phpstan:

- **Poziom 0:** Podstawowa analiza, wykrywanie najprostszych błędów.
- **Poziom 1-4:** Stopniowe zwiększanie restrykcyjności, sprawdzanie typów zmiennych, zgodności metod, itp.
- **Poziom 5-8:** Bardziej szczegółowa analiza, weryfikacja bardziej zaawansowanych przypadków użycia.

- **Poziom 9:** Bardzo restrykcyjna analiza, pełna kontrola typów i zgodności kodu.
- **dostępne od PHPStan 2 Poziom 10:** jeszcze bardziej rygorystyczne podejście do typu mixed — zgłasza błędy nawet dla niejawnych przypadków typu mixed (gdy brak jest określonego typu), a nie tylko dla jawnie zadeklarowanego mixed.

1.9 Instalacja i konfiguracja PHPUnit i phpstan

Aby dodać PHPUnit i phpstan do projektu, należy wykonać następujące kroki:
Krok 1: Instalacja PHPUnit

```
1 composer require --dev phpunit/phpunit
```

Krok 2: Instalacja phpstan

```
1 composer require --dev phpstan/phpstan
```

Krok 3: Konfiguracja autoloadingu W pliku 'composer.json' należy dodać sekcję 'autoload' i 'autoload-dev':

```
1 {  
2     "autoload": {  
3         "psr-4": {  
4             "App\\": "src/"  
5         }  
6     },  
7     "autoload-dev": {
```

```
8         "psr-4": {
9             "Tests\\": "tests/"
10        }
11    }
12 }
```

Następnie należy zaktualizować autoloading:

```
1 composer dump-autoload
```

Krok 4: Konfiguracja phpstan Utwórz plik 'phpstan.neon' w głównym katalogu projektu:

```
1 parameters:
2     level: max
3     paths:
4         - src
5         - tests
```

Krok 5: Konfiguracja PHPUnit Utwórz plik 'phpunit.xml' w głównym katalogu projektu:

```
1 <phpunit bootstrap="vendor/autoload.php">
2     <testsuites>
3         <testsuite name="Unit Tests">
4             <directory>tests</directory>
5         </testsuite>
6     </testsuites>
7 </phpunit>
```

1.9.1 Pokrycie kodu

Xdebug to rozszerzenie PHP umożliwiające zaawansowane debugowanie, profilowanie oraz generowanie raportów pokrycia kodu. W kontekście testów jednostkowych Xdebug jest często wykorzystywany do analizy tego, które fragmenty kodu zostały faktycznie wykonane podczas testów — co jest kluczowe przy ocenie jakości testów i wykrywaniu nieprzetestowanej logiki.

Nota autora

UWAGA - ważne

W repozytorium towarzyszącym frameworkowi znajdziesz również drugie rozszerzenie służące do pokrycia kodu - `pcov`. W czasie tworzenia frameworka (oraz niniejszego ebooka) PHP 8.5 znajdowało się jeszcze w fazie RC, a `pcov` — ze względu na swoją prostotę - był znacznie łatwiejszy do samodzielnego skompilowania ze źródeł.

Obecnie, Drogi Czytelniku, masz pełną dowolność wyboru rozszerzenia. Oba działają poprawnie i są wspierane przez PHPUnit.

Instalacja Xdebug

Aby zainstalować Xdebug, wykonaj:

```
1 pecl install xdebug
```

Następnie dodaj wpis do pliku `php.ini`:

```
1 zend_extension="xdebug"
```

Generowanie raportu pokrycia kodu

Aby wygenerować raport pokrycia kodu w formacie HTML, uruchom PHPUnit z odpowiednią opcją:

```
1 phpunit --coverage-html coverage
```

Wygenerowany raport znajdziesz w katalogu `coverage`.

Ułatwienie pracy dzięki skryptom Composer

Aby znacznie uprościć codzienną pracę z testami, analizą statyczną oraz narzędziami jakości kodowania, gorąco zachęcam do umieszczenia w pliku `composer.json` następującej sekcji:

```
1 "scripts": {  
2     "test": "php8.5 vendor/bin/phpunit --no-coverage",  
3     "test-coverage": "php8.5 vendor/bin/phpunit --coverage  
4         -text",  
5     "test-coverage-html": "php8.5 vendor/bin/phpunit --  
6         coverage-html coverage",  
7  
8     "analyse": "php8.5 vendor/bin/phpstan analyse --memory  
9         -limit=512M",  
10  
11     "insights": "php8.5 vendor/bin/phpinsights",  
12     "insights:fix": "php8.5 vendor/bin/phpinsights --fix",  
13     "insights:quality":  
14         "php8.5 vendor/bin/phpinsights --min-quality=95 --  
15             min-complexity=95 "  
16         "--min-architecture=95 --min-style=95 --no-  
17             interaction",  
18  
19     "quality": [  
20         "@analyse",
```

```
16         "@test",  
17         "@insights:quality"  
18     ]  
19 },
```

Dzięki temu wszystkie kluczowe narzędzia jakości można uruchomić jednym poleceniem:

```
1 composer quality
```

Co — nie ukrywajmy - jest wyjątkowo wygodne

Nota autora

PS. Ustawienie jakości na poziomie **95%** jest przeznaczone raczej dla entuzjastów (takich jak autor), którzy chcą zagwarantować najwyższą jakość kodu w kursie, który właśnie czytasz. W rzeczywistych projektach — szczególnie tych większych — poziom **70–80%** jest najczęściej w zupełności wystarczający i pozwala zachować rozsądny balans pomiędzy jakością a produktywnością zespołu.

PS2. Jeśli pracujesz w środowisku Windows, istnieje duże prawdopodobieństwo, że będziesz mieć zakończenia linii w formacie **CRLF**. Narzędzie **phpinsights** jest na to dość wrażliwe (co wynika z PSR-12 "All PHP files MUST use the Unix LF (linefeed) line ending only."), szczególnie w sekcji **style**. Masz wówczas trzy możliwości:

1. zignorować wskaźnik stylu,
2. wymusić format **LF** w ustawieniach edytora,
3. uruchomić **insights:fix**, który automatycznie naprawi zakończenia linii.

W praktyce trzecia opcja działa najwygodniej.

Rozdział 5

Dependency Injection

W poprzednich rozdziałach utworzyliśmy system czasu i obsługę HTTP. Każdy z tych komponentów działa samodzielnie, ale w prawdziwej aplikacji muszą ze sobą współpracować. Tu wkracza Dependency Injection – wzorzec projektowy, który pozwala elegancko zarządzać zależnościami między obiektami.

W tym rozdziale zbudujemy kontener DI zgodny z PSR-11, wyposażony w:

- Automatyczne rozwiązywanie zależności (autowire)
- Wsparcie dla dot-notation przy konfiguracji
- Wzorzec Service Provider do modularnej rejestracji usług
- Wyjątki zgodne z PSR-11

Nota autora

W wielu kursach kontener DI pojawia się dopiero w połowie książki. W Larafony zaczynamy od niego wcześniej, bo bez kontenera każdy kolejny komponent wymagałby ręcznego “sklejania” zależności. A to szybko zamienia się w kod wyglądający jak talerz spaghetti – smaczny, ale trudny do utrzymania.

5.1 Dlaczego Dependency Injection jest ważne?

Dependency Injection to nie tylko buzzword do CV – przynosi realne korzyści:

- **Testowalność:** Zależności można łatwo podmienić na mocki lub stuby podczas testów. Zamiast walczyć z globalnym stanem, wstrzykujemy dokładnie to, czego potrzebujemy.
- **Modularność:** Klasy nie są sztywno powiązane z konkretnymi implementacjami. Dziś używasz MySQL, jutro PostgreSQL – zmiana to kwestia konfiguracji, nie refaktoryzacji.
- **Łatwość utrzymania:** Zmiana implementacji zależności nie wymaga modyfikacji klas, które z niej korzystają. Zasada Open/Closed w praktyce.
- **Zarządzanie cyklem życia:** Kontener wie, kiedy tworzyć nowe instancje, a kiedy zwracać istniejące (singleton pattern).

Nota autora

Lubię myśleć o kontenerze DI jak o kelnerze w restauracji. Zamawiasz “kawę” – nie interesuje cię, czy barista użył ekspresu ciśnieniowego czy french pressa. Dostajesz kawę. Kontener działa tak samo: prosisz o `DatabaseInterface`, dostajesz działającą bazę danych. Szczegóły implementacji zostają w kuchni.

5.2 PSR-11 – Container Interface

PSR-11 to standard definiujący interfejs kontenera DI. Jest minimalistyczny – wymaga tylko dwóch metod:

```
1 interface ContainerInterface {  
2     public function get(string $id): mixed;  
3     public function has(string $id): bool;  
4 }
```

Dzięki temu każda biblioteka zgodna z PSR-11 może używać dowolnego kontenera – czy to naszego Larafony, czy PHP-DI, Symfony Container lub innego.

5.3 Architektura kontenera

Nasz kontener składa się z następujących komponentów:

- `ContainerContract` – interfejs rozszerzający PSR-11 o dodatkowe metody
- `Container` – główna klasa kontenera
- `Autowire` – mechanizm automatycznego rozwiązywania zależności
- `ReflectionResolver` – wrapper na PHP Reflection API
- `DotContainer` – storage z obsługą dot-notation
- `ServiceProvider` – bazowa klasa dla modularnej rejestracji usług

5.4 Interfejs kontenera

Rozszerzamy PSR-11 `ContainerInterface` o metody przydatne w codziennej pracy:

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Container\Contracts;
6
7 use Psr\Container\ContainerInterface;
8
9 interface ContainerContract extends ContainerInterface
10 {
```

```
11  /**
12   * Bind a simple value to the container.
13   */
14  public function bind(string $key, string|int|float|
15      bool|null $value): void;
16
17  #[\NoDiscard]
18  public function getBinding(string $key): string|int|
19      float|bool|null;
20
21  public function set(string $key, mixed $value): self;
22 }
```

Listing 5.1: Interfejs ContainerContract

Metoda `bind()` służy do przechowywania prostych wartości skalarnych (konfiguracja), podczas gdy `set()` obsługuje pełne obiekty i klasy.

5.5 Wyjątki zgodne z PSR-11

PSR-11 definiuje dwa rodzaje wyjątków:

- `ContainerExceptionInterface` – ogólny błąd kontenera
- `NotFoundExceptionInterface` – żądany element nie istnieje

```
1  <?php
2
3  declare(strict_types=1);
4
5  namespace Larafony\Framework\Container\Exceptions;
6
7  use Psr\Container\ContainerExceptionInterface;
8
```

```
9 class ContainerError extends \RuntimeException implements
    ContainerExceptionInterface
10 {
11 }
```

Listing 5.2: Klasa ContainerError

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Container\Exceptions;
6
7 use Larafony\Framework\Core\Exceptions\NotFoundError as
    CoreNotFoundError;
8 use Psr\Container\NotFoundExceptionInterface;
9
10 class NotFoundError extends CoreNotFoundError implements
    NotFoundExceptionInterface
11 {
12 }
```

Listing 5.3: Klasa NotFoundError

5.6 Reflection API – zagłębienie pod maskę

Refleksja (Reflection) w PHP to mechanizm pozwalający na analizowanie klas, obiektów oraz ich właściwości w trakcie działania programu. Dzięki `ReflectionClass` możemy dynamicznie uzyskiwać informacje o konstruktorach, metodach i zależnościach obiektów.

Nota autora

Refleksja to jak rentgen dla kodu – pozwala zajrzeć do środka klasy bez jej otwierania. Przydatne, gdy chcesz wiedzieć, czego klasa potrzebuje do życia, zanim ją utworzysz.

5.6.1 Interfejs ReflectionResolver

Aby nie korzystać bezpośrednio z API refleksji w wielu miejscach, tworzymy wrapper:

```
1  <?php
2
3  declare(strict_types=1);
4
5  namespace Larafony\Framework\Container\Contracts;
6
7  use Larafony\Framework\Container\Exceptions\ContainerError
8      ;
9  use ReflectionParameter;
10
11 interface ReflectionResolverContract
12 {
13     /**
14      * Get the constructor parameters for a given class.
15      *
16      * @param class-string $className The fully qualified
17      *    class name
18      *
19      * @return array<int, ReflectionParameter> An array of
20      *    constructor parameters
21      */
22     public function getConstructorParameters(string
23         $className): array;
```

```
24      * @param ReflectionParameter $parameter The parameter
      *       to analyze
25      *
26      * @return string|null The type of the parameter or
      *       null if it can't be determined
27      */
28      public function getParameterType(ReflectionParameter
      $parameter): ?string;
29
30      /**
31      * Check if a parameter has a default value.
32      *
33      * @param ReflectionParameter $parameter The parameter
      *       to check
34      *
35      * @return bool True if the parameter has a default
      *       value, false otherwise
36      */
37      public function hasDefaultValue(ReflectionParameter
      $parameter): bool;
38
39      /**
40      * Get the default value of a parameter.
41      *
42      * @param ReflectionParameter $parameter The parameter
      *       to get the default value from
43      *
44      * @return mixed The default value of the parameter
45      *
46      * @throws ContainerError If the parameter has no
      *       default value
47      */
48      public function getDefaultValue(ReflectionParameter
      $parameter): mixed;
49
50      /**
51      * Check if a parameter allows null.
52      *
53      * @param ReflectionParameter $parameter The parameter
      *       to check
54      *
```

```
55     * @return bool True if the parameter allows null,  
    false otherwise  
56     */  
57     public function allowsNull(ReflectionParameter  
        $parameter): bool;  
58  
59     /**  
60     * Get the default value for a built-in type.  
61     *  
62     * @param string $type The built-in type  
63     *  
64     * @return mixed The default value for the type  
65     */  
66     public function getDefaultValueForBuiltInType(string  
        $type): mixed;  
67 }
```

Listing 5.4: Interfejs ReflectionResolverContract

5.6.2 Implementacja ReflectionResolver

```
1  <?php  
2  
3  declare(strict_types=1);  
4  
5  namespace Larafony\Framework\Container\Resolvers;  
6  
7  use Larafony\Framework\Container\Contracts\  
    ReflectionResolverContract;  
8  use Larafony\Framework\Container\Exceptions\ContainerError  
    ;  
9  use ReflectionClass;  
10 use ReflectionException;  
11 use ReflectionNamedType;  
12 use ReflectionParameter;  
13
```

```
14 class ReflectionResolver implements  
    ReflectionResolverContract  
15 {  
16     /**  
17     * Get the constructor parameters for a given class.  
18     *  
19     * @param class-string $className The fully qualified  
        class name  
20     *  
21     * @return array<int, ReflectionParameter> An array of  
        constructor parameters  
22     *  
23     * @throws ReflectionException If the class does not  
        exist or has no constructor  
24     */  
25     public function getConstructorParameters(string  
        $className): array  
26     {  
27         $reflectionClass = new ReflectionClass($className)  
            ;  
28         $constructor = $reflectionClass->getConstructor();  
29  
30         return $constructor?->getParameters() ?? [];  
31     }  
32  
33     /**  
34     * Get the type of a parameter.  
35     *  
36     * @param ReflectionParameter $parameter The parameter  
        to analyze  
37     *  
38     * @return string|null The type of the parameter or  
        null if it can't be determined  
39     */  
40     public function getParameterType(ReflectionParameter  
        $parameter): ?string  
41     {  
42         $type = $parameter->getType();  
43         return $type instanceof ReflectionNamedType ?  
            $type->getName() : null;  
44     }
```

```
45
46  /**
47   * Check if a parameter has a default value.
48   *
49   * @param ReflectionParameter $parameter The parameter
      to check
50   *
51   * @return bool True if the parameter has a default
      value, false otherwise
52   */
53  public function hasDefaultValue(ReflectionParameter
      $parameter): bool
54  {
55      return $parameter->isDefaultValueAvailable();
56  }
57
58  /**
59   * Get the default value of a parameter.
60   *
61   * @param ReflectionParameter $parameter The parameter
      to get the default value from
62   *
63   * @return mixed The default value of the parameter
64   *
65   * @throws ContainerError|ReflectionException If the
      parameter has no default value
66   */
67  public function getDefaultValue(ReflectionParameter
      $parameter): mixed
68  {
69      if (! $this->hasDefaultValue($parameter)) {
70          throw new ContainerError("Parameter {
              $parameter->getName()} has no default value
              .");
71      }
72
73      return $parameter->getDefaultValue();
74  }
75
76  /**
77   * Check if a parameter allows null.
```

```
78      *
79      * @param ReflectionParameter $parameter The parameter
      *       to check
80      *
81      * @return bool True if the parameter allows null,
      *       false otherwise
82      */
83      public function allowsNull(ReflectionParameter
      $parameter): bool
84      {
85          return $parameter->allowsNull();
86      }
87
88      /**
89      * Get the default value for a built-in type.
90      *
91      * @param string $type The built-in type
92      *
93      * @return mixed The default value for the type
94      */
95      public function getDefaultValueForBuiltInType(string
      $type): mixed
96      {
97          return match ($type) {
98              'int' => 0,
99              'float' => 0.0,
100             'string' => '',
101             'bool' => false,
102             'array' => [],
103             default => null,
104         };
105      }
106  }
```

Listing 5.5: Klasa ReflectionResolver

5.6.3 Ewolucja typów w PHP

W PHP 5 parametry funkcji nie miały typów – programista musiał zaufać, że dostanie odpowiednie dane. PHP 7.1 wprowadził typowanie skalarne (`int`, `string`, `float`, `bool`), PHP 8.0 dodał typy unii (`int|string`), a PHP 8.2 typy DNF (Disjunctive Normal Form).

Użyty w metodzie `getParameterType()` typ `ReflectionNamedType` pozwala na bezpieczne pobranie nazwy typu parametru.

5.6.4 Nullable types – historia i teraźniejszość

Od PHP 5.1 możliwe było deklarowanie typów z domyślną wartością `null`:

```
1 function test(string $test = null) {}  
2 test('PHP'); // Dozwolone  
3 test(null);  // Dozwolone
```

PHP 8.0 wprowadził Union Types, umożliwiając jawną deklarację `string|null`. W PHP 8.4 wycofano niejawne typy nullable – deklaracje bez `null` w typie, ale z domyślną wartością `null`, generują ostrzeżenie:

```
1 function test(array $value = null) {}  
2 test(null); // Deprecated: Implicit nullable parameter
```

Nota autora

PHP ewoluuje w kierunku jawności typów. To dobra wiadomość – mniej “magii” oznacza mniej niespodzianek o 3 w nocy na produkcji.

5.7 Autowire – automatyczne wstrzykiwanie zależności

Autowire to serce naszego kontenera. Analizuje konstruktor klasy i automatycznie dostarcza wszystkie wymagane zależności.

5.7.1 Interfejs Autowire

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Container\Contracts;
6
7 interface AutowireContract
8 {
9     public function instantiate(string $className): object
10    ;
11 }
```

Listing 5.6: Interfejs AutowireContract

5.7.2 Implementacja Autowire

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Container\Resolvers;
6
7 use Larafony\Framework\Container\Contracts\
    AutowireContract;
```

```
8 use Larafony\Framework\Container\Contracts\  
    ReflectionResolverContract;  
9 use Larafony\Framework\Container\Exceptions\NotFoundError;  
10 use Psr\Container\ContainerInterface;  
11 use ReflectionClass;  
12 use ReflectionException;  
13 use ReflectionParameter;  
14  
15 /**  
16  * Class Autowire  
17  *  
18  * Responsible for automatically resolving and injecting  
    dependencies.  
19  */  
20 class Autowire implements AutowireContract  
21 {  
22     public function __construct(  
23         private ContainerInterface $container,  
24         private ?ReflectionResolverContract $resolver =  
            null,  
25     ) {  
26         $this->resolver ??= new ReflectionResolver();  
27     }  
28  
29     /**  
30      * Autowire and instantiate a class.  
31      *  
32      * @template T of object  
33      *  
34      * @param class-string<T> $className The name of the  
        class to instantiate  
35      *  
36      * @return T The instantiated object  
37      *  
38      * @throws ReflectionException If the class cannot be  
        reflected  
39      * @throws NotFoundError If a dependency cannot be  
        resolved  
40      */  
41     public function instantiate(string $className): object  
42     {
```

```

43     $reflectionClass = new ReflectionClass($className)
44     ;
45     $constructor = $reflectionClass->getConstructor();
46
47     if ($constructor === null) {
48         return $reflectionClass->newInstance();
49     }
50
51     $parameters = $this->resolver->
52         getConstructorParameters($className) ?? [];
53     $arguments = $this->resolveParameters($parameters)
54     ;
55
56     return $reflectionClass->newInstanceArgs(
57         $arguments);
58 }
59
60 /**
61  * Resolve parameters for dependency injection.
62  *
63  * @param array<int, ReflectionParameter> $parameters
64  *
65  * @return array<mixed>
66  */
67 private function resolveParameters(array $parameters):
68     array
69 {
70     return array_map(function (ReflectionParameter
71         $parameter) {
72         $parameterName = $parameter->getName();
73         /** @var string $type */
74         $type = $this->resolver->getParameterType(
75             $parameter);
76
77         return match (true) {
78             // 1. return given value if exists
79             $this->container->has($parameterName) =>
80                 $this->container->get($parameterName),
81             $this->container->has($type) => $this->
82                 container->get($type),
83             // 2. return default value if exist

```

```
75         $this->resolver?->hasDefaultValue(  
76             $parameter) => $this->resolver->  
77                 getDefaultValue($parameter),  
78  
79         // 3. return null if allowed  
80         $this->resolver?->allowsNull($parameter)  
81             => null,  
82  
83         // 4. for builtin types return default  
84         value  
85         $type && $this->isBuiltInType($type) =>  
86             $this->resolver?->  
87                 getDefaultValueForBuiltInType($type),  
88  
89         // 5. for object check recursively  
90         $type && class_exists($type) => $this->  
91             instantiate($type),  
92         // otherwise throw not found exception  
93         default => throw new NotFoundError(  
94             "Unable to resolve parameter {  
95                 $parameterName} of type {$type}"  
96             )  
97     );  
98     }, $parameters);  
99 }  
100  
101 private function isBuiltInType(string $type): bool  
102 {  
103     return in_array($type, ['int', 'float', 'string',  
104         'bool', 'array']);  
105 }  
106 }  
107 }
```

Listing 5.7: Klasa Autowire

5.7.3 Algorytm rozwiązywania zależności

Metoda `resolveParameters()` sprawdza parametry w następującej kolejności:

1. **Nazwa parametru** – czy kontener ma wartość pod kluczem równym nazwie parametru?
2. **Typ parametru** – czy kontener ma wartość pod kluczem równym typowi?
3. **Wartość domyślna** – czy parametr ma zdefiniowaną wartość domyślną?
4. **Nullable** – czy parametr może być `null`?
5. **Typ wbudowany** – dla `int`, `string`, itp. zwracamy sensowne domyślne wartości
6. **Klasa** – rekursywnie tworzymy instancję wymaganej klasy
7. **Błąd** – jeśli nic nie pasuje, rzucamy `NotFoundError`

Nota autora

Użycie wyrażenia `match` z PHP 8 zamiast łańcucha `if-else` sprawia, że algorytm jest czytelny jak przepis kulinarny. No, może jak przepis dla programistów – z mniejszą ilością masła. Dodatkowo `match` ma 2 przewagi nad `switch`:

- Używa `===` zamiast `==`
- Rzuci wyjątkiem jeśli nic nie zostanie dopasowane, co pozwala łatwiej wykryć potencjalne błędy przy statycznej analizie kodu

5.8 DotContainer – storage z kropką

Konfiguracja w prawdziwych aplikacjach jest zagnieżdżona: `database.mysql.host`, `cache.redis.port`. Klasa `DotContainer` obsługuje taką notację naturalnie.

5.8.1 Interfejs ArrayContract

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Container\Contracts;
6
7 interface ArrayContract
8 {
9     public function get(string $key, mixed $default = null
10         ): mixed;
11     public function set(string $key, mixed $default = null
12         ): void;
13     public function has(string $key): bool;
14 }
```

Listing 5.8: Interfejs ArrayContract

5.8.2 Pomocnik ArrayGet

Klasa do odczytu zagnieżdżonych wartości:

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Container\Helpers;
6
7 use ArrayObject;
8
9 final readonly class ArrayGet
10 {
11     public function __construct(
12         private DotContainer $container
```

```
13     ) {
14     }
15
16     public function get(string $key, mixed $default = null
17         ): mixed
18     {
19         $keys = explode('.', $key);
20         $data = $this->container->getArrayCopy();
21         $data = new ArrayObject($data);
22
23         foreach ($keys as $segment) {
24             if ($data->offsetExists($segment)) {
25                 $value = $data[$segment];
26                 $value = is_array($value) ? new
27                     ArrayObject($value) : $value;
28                 $data = $value;
29             } else {
30                 return $default;
31             }
32         }
33
34         return $data;
35     }
36 }
```

Listing 5.9: Klasa ArrayGet

5.8.3 Pomocnik ArraySet

Klasa do zapisu zagnieżdżonych wartości z użyciem referencji:

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Container\Helpers;
6
```

```
7 use ArrayObject;
8
9 final class ArraySet
10 {
11     public function __construct(
12         private DotContainer $container
13     ) {
14     }
15
16     public function set(string $key, mixed $value): void
17     {
18         $keys = explode('.', $key);
19         $data = &$this->container;
20
21         foreach ($keys as $segment) {
22             if (! isset($data[$segment])) {
23                 $data[$segment] = new ArrayObject();
24             }
25
26             $data = &$data[$segment];
27         }
28
29         $data = $value;
30     }
31 }
```

Listing 5.10: Klasa ArraySet

5.8.4 DotContainer

Połączenie powyższych helperów w jedną klasę:

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Container\Helpers;
```

```
6
7 use ArrayObject;
8 use Larafony\Framework\Container\Contracts\ArrayContract;
9
10 /**
11  * @extends ArrayObject<string|int, mixed>
12  */
13 class DotContainer extends ArrayObject implements
    ArrayContract
14 {
15     private readonly ArrayGet $arrayGet;
16     private readonly ArraySet $arraySet;
17
18     /**
19      * @param array<int|string, mixed> $array
20      */
21     public function __construct(array $array = [])
22     {
23         parent::__construct($array);
24         $this->arrayGet = new ArrayGet($this);
25         $this->arraySet = new ArraySet($this);
26     }
27
28     public function get(string $key, mixed $default = null
29         ): mixed
30     {
31         return $this->arrayGet->get($key, $default);
32     }
33
34     public function set(string $key, mixed $default = null
35         ): void
36     {
37         $this->arraySet->set($key, $default);
38     }
39
40     public function has(string $key): bool
41     {
42         return $this->arrayGet->get($key) !== null;
43     }
44 }
```

Listing 5.11: Klasa DotContainer

Nota autora

Dot notation to jeden z tych wzorców, który gdy już poznasz, nie wyobrażasz sobie życia bez niego. `$config->get('database.mysql.host')` jest po prostu czytelniejsze niż `$config['database']['mysql']['host']` – i nie wymaga sprawdzania, czy każdy poziom istnieje.

5.9 Główna klasa Container

Czas połączyć wszystkie elementy w jedną całość:

5.9.1 Atrybut `#[NoDiscard]` – nowość PHP 8.5

W klasie `Container` przy metodach `get()` i `getBinding()` znajdziesz atrybut `#[NoDiscard]`. To nowość wprowadzona w PHP 8.5, która informuje analizatory statyczne (PHPStan, Psalm) oraz IDE, że wartość zwracana przez metodę **nie powinna być ignorowana**.

```
1 #[NoDiscard]
2 public function get(string $id): mixed { ... }
3
4 // Poprawne użycie:
5 $service = $container->get(UserService::class);
6
7 // Błąd - wynik jest ignorowany:
8 $container->get(UserService::class); // PHPStan zgłosi ostrzeżenie
```

Dlaczego to ważne? Wywołanie `get()` bez przypisania wyniku to prawie na pewno błąd programisty – albo zapomniał przypisać wartość, albo pomylił metodę. Dzięki `#[NoDiscard]` taki błąd zostanie wykryty podczas analizy

statycznej, a nie dopiero na produkcji.

Nota autora

Atrybut `#[NoDiscard]` to jeden z tych dodatków, które sprawiają, że PHP staje się coraz bardziej “dorosłym” językiem. Rust ma podobny mechanizm od lat i programiści go uwielbiają. Teraz PHP dołącza do klubu – lepiej późno niż wcale.

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Container;
6
7 use Larafony\Framework\Container\Contracts\ArrayContract;
8 use Larafony\Framework\Container\Contracts\
    AutowireContract;
9 use Larafony\Framework\Container\Contracts\
    ContainerContract;
10 use Larafony\Framework\Container\Exceptions\NotFoundError;
11 use Larafony\Framework\Container\Helpers\DotContainer;
12 use Larafony\Framework\Container\Resolvers\Autowire;
13 use Larafony\Framework\Core\Support\Str;
14 use NoDiscard;
15
16 class Container implements ContainerContract
17 {
18     /**
19      * @var array<string, string|int|float|bool|null>
20      */
21     private array $bindings = [];
22
23     public function __construct(
24         private ?AutowireContract $autowire = null,
25         private readonly ArrayContract $entries = new
            DotContainer(),
26     ) {
27         $this->autowire ??= new Autowire($this);
28     }
```

```

29     public function bind(string $key, float|bool|int|
30         string|null $value): void
31     {
32         $this->bindings[$key] = $value;
33     }
34
35     #[NoDiscard]
36     public function getBinding(string $key): string|int|
37         float|bool|null
38     {
39         return $this->bindings[$key] ?? throw new
40             NotFoundError('Binding not found: '.$key);
41     }
42
43     /**
44     * Sets item in container
45     *
46     * @param string $key
47     * @param mixed $value
48     *
49     * @return self
50     */
51     public function set(string $key, mixed $value):
52         Contracts\ContainerContract
53     {
54         if (Str::isClassString($value) && ! $this->has(
55             $value)) {
56             $value = $this->autowire?->instantiate($value)
57                 ;
58         }
59         $this->entries->set($key, $value);
60         return $this;
61     }
62
63     #[NoDiscard]
64     public function get(string $id): mixed
65     {
66         if (! $this->entries->has($id)) {
67             return $this->autowire?->instantiate($id);
68         }
69     }

```

```
64     $value = $this->entries->get($id);
65
66     return Str::isClassString($value) ? $this->
        autowire?->instantiate($value) : $value;
67 }
68
69 public function has(string $id): bool
70 {
71     return $this->entries->has($id);
72 }
73 }
```

Listing 5.12: Klasa Container

5.9.2 Dwa systemy przechowywania

Kontener używa dwóch oddzielnych systemów:

- `$bindings` – dla prostych wartości skalarnych (konfiguracja)
- `$entries` (DotContainer) – dla obiektów i złożonych struktur

Rozdzielenie to pozwala na szybki dostęp do konfiguracji bez przeszukiwania całego kontenera.

5.9.3 Lazy autowiring

W metodzie `get()` kontener sprawdza, czy żądany klucz istnieje. Jeśli nie – próbuje utworzyć instancję klasy o tej nazwie. To oznacza, że nie musisz rejestrować każdej klasy – kontener sam ją znajdzie i utworzy.

Nota autora

To podejście ma nazwę “convention over configuration” – jeśli klasa nazywa się `UserRepository`, kontener założy, że chcesz właśnie ją. Mniej konfiguracji, więcej pisania kodu, który robi rzeczy.

5.10 Service Provider – modularność usług

W dużych aplikacjach rejestrowanie wszystkich usług w jednym miejscu szybko staje się nieczytelne. Service Provider pozwala grupować powiązane usługi w oddzielne moduły.

5.10.1 Interfejs ServiceProvider

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Container\Contracts;
6
7 interface ServiceProviderContract
8 {
9     /**
10      * @return array<int|string, class-string>
11      */
12     public function providers(): array;
13     /**
14      * Register services in the given container.
15      *
16      * @param ContainerContract $container The container
17      *   to register services in
18      */
19     public function register(ContainerContract $container)
20         : self;
```

```
20     /**
21      * bootstrapping services
22      */
23     public function boot(ContainerContract $container):
24         void;
25 }
```

Listing 5.13: Interfejs ServiceProviderContract

5.10.2 Implementacja ServiceProvider

```
1  <?php
2
3  declare(strict_types=1);
4
5  namespace Larafony\Framework\Container;
6
7  use Larafony\Framework\Container\Contracts\
8      ContainerContract;
9  use Larafony\Framework\Container\Contracts\
10     ServiceProviderContract;
11
12 abstract class ServiceProvider implements
13     ServiceProviderContract
14 {
15     /**
16      * @return array<int|string, class-string>
17      */
18     public function providers(): array
19     {
20         return [];
21     }
22
23     public function register(ContainerContract $container)
24         : self
25     {
26         foreach ($this->providers() as $key => $class) {
```

```
23         $key = is_int($key) ? $class : $key;
24         $container->set($key, $class);
25     }
26     return $this;
27 }
28
29 public function boot(ContainerContract $container):
30     void
31     {
32         if (! $container->has($this::class)) {
33             $container->set($this::class, $this);
34         }
35     }
```

Listing 5.14: Klasa ServiceProvider

5.10.3 Cykl życia Service Providera

1. **providers()** – zwraca tablicę klas do zarejestrowania
2. **register()** – rejestruje wszystkie klasy w kontenerze
3. **boot()** – wykonuje dodatkową inicjalizację po rejestracji

Metoda `register()` inteligentnie obsługuje klucze tablicy:

- Klucz liczbowy: klasa rejestrowana pod własną nazwą
- Klucz tekstowy: klasa rejestrowana pod podanym aliasem (np `[LoggerContract => MyLogger]`)

Nota autora

Ten wzorzec jest inspirowany Laravelem, ale uproszczony. W Larafony `boot()` służy głównie do samorejestracji providera w kontenerze – przydatne, gdy chcesz później sprawdzić, które providery zostały załadowane.

Nota autora

Jedynymi zależnościami produkcyjnymi Larafony są pakiety PSR. Oznacza to, że jeśli kiedykolwiek zechcesz podmienić któryś z komponentów frameworka – wystarczy utworzyć ServiceProvider, w którym do odpowiedniego klucza (interfejsu) przypiszesz własną implementację. Tylko i aż tyle. Żadnego vendor lock-in, żadnych haków w kodzie frameworka. Twoja aplikacja, Twoje zasady.

5.11 Testy jednostkowe

5.11.1 Klasy pomocnicze do testów

Do testowania kontenera potrzebujemy kilku prostych klas:

```
1 class Database {}
2
3 class UserRepository {
4     public function __construct(
5         public readonly Database $database,
6     ) {}
7 }
8
9 class TestClassWithParams {
10     public function __construct(
11         public string $name,
12         public int $age,
13     ) {}
14 }
15
16 class TestClassWithDefaults {
17     public function __construct(
18         public string $required,
19         public string $optional = 'default',
20     ) {}
21 }
```

```
22 |
23 | class TestClassWithNullable {
24 |     public function __construct(
25 |         public ?string $nullable,
26 |         public string $required,
27 |     ) {}
28 | }
```

5.11.2 Testy ReflectionResolver

```
1 | <?php
2 |
3 | declare(strict_types=1);
4 |
5 | namespace Larafony\Container;
6 |
7 | use Larafony\Framework\Container\Exceptions\ContainerError
8 |     ;
9 | use Larafony\Framework\Container\Resolvers\
10 |    ReflectionResolver;
11 | use PHPUnit\Framework\TestCase;
12 | use ReflectionParameter;
13 |
14 | final class ReflectionResolverTest extends TestCase
15 | {
16 |     private ReflectionResolver $resolver;
17 |
18 |     protected function setUp(): void
19 |     {
20 |         $this->resolver = new ReflectionResolver();
21 |     }
22 |
23 |     public function testGetConstructorParameters(): void
24 |     {
25 |         $parameters = $this->resolver->
26 |             getConstructorParameters(TestClassWithParams::
```

```
class);  
  
$this->assertCount(2, $parameters);  
$this->assertInstanceOf(ReflectionParameter::class  
    , $parameters[0]);  
  
}  
  
public function testGetParameterType(): void  
{  
    $parameters = $this->resolver->  
        getConstructorParameters(TestClassWithParams::  
            class);  
  
    $type = $this->resolver->getParameterType(  
        $parameters[0]);  
    $this->assertSame('string', $type);  
}  
  
public function testHasDefaultValue(): void  
{  
    $parameters = $this->resolver->  
        getConstructorParameters(TestClassWithDefaults  
            ::class);  
  
    $this->assertFalse($this->resolver->  
        hasDefaultValue($parameters[0]));  
    $this->assertTrue($this->resolver->hasDefaultValue(  
        $parameters[1]));  
}  
  
public function testGetDefaultValue(): void  
{  
    $parameters = $this->resolver->  
        getConstructorParameters(TestClassWithDefaults  
            ::class);  
  
    $this->assertSame('default', $this->resolver->  
        getDefaultValue($parameters[1]));  
}
```

```
52 public function testGetDefaultValueThrowsException():  
    void  
53 {  
54     $parameters = $this->resolver->  
        getConstructorParameters(TestClassWithDefaults  
            ::class);  
55  
56     $this->expectException(ContainerError::class);  
57     $this->expectExceptionMessage('has no default  
        value');  
58  
59     $this->resolver->getDefaultValue($parameters[0]);  
60 }  
61  
62 public function testAllowsNull(): void  
63 {  
64     $parameters = $this->resolver->  
        getConstructorParameters(TestClassWithNullable  
            ::class);  
65  
66     $this->assertTrue($this->resolver->allowsNull(  
        $parameters[0]));  
67     $this->assertFalse($this->resolver->allowsNull(  
        $parameters[1]));  
68 }  
69  
70 public function testGetDefaultValueForBuiltInType():  
    void  
71 {  
72     $this->assertSame(0, $this->resolver->  
        getDefaultValueForBuiltInType('int'));  
73     $this->assertSame(0.0, $this->resolver->  
        getDefaultValueForBuiltInType('float'));  
74     $this->assertSame(' ', $this->resolver->  
        getDefaultValueForBuiltInType('string'));  
75     $this->assertSame(false, $this->resolver->  
        getDefaultValueForBuiltInType('bool'));  
76     $this->assertSame([], $this->resolver->  
        getDefaultValueForBuiltInType('array'));  
77     $this->assertNull($this->resolver->  
        getDefaultValueForBuiltInType('unknown'));
```

```
78     }  
79 }
```

Listing 5.15: Testy klasy ReflectionResolver

5.11.3 Testy DotContainer

```
1  <?php  
2  
3  declare(strict_types=1);  
4  
5  namespace Larafony\Framework\Tests\Container;  
6  
7  use ArrayObject;  
8  use Larafony\Framework\Container\Helpers\DotContainer;  
9  use PHPUnit\Framework\TestCase;  
10  
11 final class DotContainerTest extends TestCase  
12 {  
13     public function testSetAndGetWithDotNotation(): void  
14     {  
15         $container = new DotContainer();  
16  
17         $container->set('foo.bar.baz', 'value');  
18         $this->assertEquals('value', $container->get('foo.  
19             bar.baz'));  
20  
21         $container->set('foo.bar.qux', 123);  
22         $this->assertEquals(123, $container->get('foo.bar.  
23             qux'));  
24     }  
25  
26     public function testSetArray(): void  
27     {  
28         $container = new DotContainer();  
29         $container->set('test', ['foo' => 'bar', 'baz' =>  
30             'qux']);
```

```
28         $this->assertEquals('bar', $container->get('test.
29             foo'));
30     }
31
32     public function testGetWithDefaultValue(): void
33     {
34         $container = new DotContainer();
35
36         $this->assertEquals('default', $container->get('
37             non.existent.key', 'default'));
38     }
39
40     public function testNestedArrayObjectIsCreated(): void
41     {
42         $container = new DotContainer();
43
44         $container->set('foo.bar.baz', 'value');
45         $this->assertInstanceOf(ArrayObject::class,
46             $container['foo']);
47         $this->assertInstanceOf(ArrayObject::class,
48             $container['foo']['bar']);
49     }
50 }
```

Listing 5.16: Testy klasy DotContainer

5.11.4 Testy Container

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Tests\Container;
6
7 use Larafony\Framework\Container\Container;
```

```
8 use Larafony\Framework\Web\Application;
9 use PHPUnit\Framework\TestCase;
10
11 final class ContainerTest extends TestCase
12 {
13     private Container $container;
14
15     protected function setUp(): void
16     {
17         $this->container = Application::instance();
18     }
19
20     public function testBindings(): void
21     {
22         $this->container->bind('const', 1);
23         $this->assertEquals(1, $this->container->
            getBinding('const'));
24     }
25
26     public function testAutowireSimpleClass(): void
27     {
28         $instance = $this->container->get(Database::class)
29             ;
30         $this->assertInstanceOf(Database::class, $instance
31             );
32
33     }
34
35     public function testAutowireWithDependencies(): void
36     {
37         $instance = $this->container->get(UserRepository::
38             class);
39         $this->assertInstanceOf(UserRepository::class,
40             $instance);
41         $this->assertInstanceOf(Database::class, $instance
42             ->database);
43     }
44
45     public function testSetAndGet(): void
46     {
47         $this->container->set('key', 'value');
48         $this->assertTrue($this->container->has('key'));
```

```
43         $this->assertEquals('value', $this->container->get
44             ('key'));
45     }
46     public function testHasReturnsFalseForNonExistent():
47         void
48     {
49         $this->assertFalse($this->container->has('
50             nonexistent'));
51     }
52     public function testRegisterByClassName()
53     {
54         $this->container->set(Database::class, Database::
55             class);
56         $db = $this->container->get(Database::class);
57         $this->assertInstanceOf(Database::class, $db);
58     }
59 }
```

Listing 5.17: Testy klasy Container

5.12 Podsumowanie

W tym rozdziale zbudowaliśmy kompletny kontener Dependency Injection zgodny z PSR-11:

- Interfejs `ContainerContract` rozszerzający PSR-11 o praktyczne metody
- Wyjątki `ContainerError` i `NotFoundError` zgodne ze standardem
- `ReflectionResolver` jako wrapper na PHP Reflection API
- `Autowire` automatycznie rozwiązujący zależności konstruktora
- `DotContainer` z obsługą zagnieżdżonej konfiguracji

- `ServiceProvider` do modularnej organizacji usług
- Lazy autowiring – klasy tworzone na żądanie bez jawnej rejestracji

Nota autora

Nasz kontener jest celowo uproszczony względem kombajnów jak Symfony Container czy PHP-DI. Nie ma dekoratorów, factory definitions ani tagged services. Ale ma wszystko, czego potrzebujemy do zbudowania działającego frameworka – a to jest celem tej książki.

W kolejnych rozdziałach wykorzystamy kontener do rejestracji routera, middleware'ów i innych komponentów HTTP. Zobaczysz, jak DI zmienia chaotyczny kod w elegancką orkiestrację obiektów.

Czas na routing!

Rozdział 10

DBAL – schemat bazy danych

W nadchodzących czterech rozdziałach będziemy wdrażać pełny, choć uproszczony, mechanizm zarządzania bazą danych, znany jako Database Abstraction Layer (DBAL). Naszym celem będzie stworzenie elastycznego systemu, który pozwoli na łatwą interakcję z bazą danych, zarówno w kontekście zarządzania strukturą bazy, jak i operacji na danych.

W pierwszej części (Rozdział 10) skupimy się na podstawowym mechanizmie połączenia z bazą danych oraz na systemie **Grammar** – wzorcu inspirowanym Laravelem, który pozwala na eleganckie generowanie zapytań SQL z definicji obiektowych. Zbudujemy fluent API do definiowania tabel, które będzie czytelne i intuicyjne w użyciu.

W Rozdziale 11 przejdziemy do tworzenia klasy **QueryBuilder**. **QueryBuilder** zaimplementuje większość operacji związanych z danymi w bazie MySQL: wstawianie danych (**INSERT**), ich wybieranie (**SELECT**), aktualizacje (**UPDATE**) oraz usuwanie (**DELETE**).

W Rozdziale 12 dodamy pierwsze komendy do zarządzania schematami bazy danych, niezbędne do pracy z migracjami. Stworzymy komendę **make:migration** oraz system szablonów, a następnie zaimplementujemy komendę **migrate**.

Zwieńczeniem naszych prac będzie Rozdział 13, w którym wdrożymy pełny ORM z obsługą relacji oraz komendy do generowania modeli, fabryk i seederów.

Nota autora

Poniżej przedstawiono przykładowy proces automatycznego seedowania bazy danych przy wykorzystaniu pełnego potencjału PHP 8.5. Stworzenie migracji dla tabeli użytkowników:

```
php console/bin make:migration --table=users
```

Następnie musimy uzupełnić tę migrację zgodnie z zaplanowaną strukturą bazy danych.

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace DummyRootNamespace;
6
7 use Larafony\Framework\Database\Base\Migrations\Migration;
8 use Larafony\Framework\Database\Schema;
9
10 return new class extends Migration
11 {
12     /**
13      * run migration.
14      */
15     public function up(): void
16     {
17         Schema::create('users', function ($table) {
18             $table->id();
19             $table->string('email', 100);
20             $table->string('username', 50)->nullable(true)
21                 ;
22             $table->string('password');
23             $table->string('remember_token', 100)->
24                 nullable(true);
25             $table->string('password_reset_token', 100)->
26                 nullable(true);
```

```
24         $table->timestamp('password_reset_expires')->
           nullable(true);
25         $table->timestamp('email_verified_at')->
           nullable(true);
26         $table->integer('is_active', 'TINYINT')->
           length(1)->default(1);
27         $table->timestamp('last_login_at')->nullable(
           true);
28         $table->timestamps();
29
30         // Indexes
31         $table->unique('email');
32         $table->unique('username');
33         $table->index('is_active');
34     }) |> Schema::execute(...);
35 }
36
37 /**
38  * rollback migration.
39  */
40 public function down(): void
41 {
42     Schema::dropIfExists('users') |> Schema::execute
43         (...);
44 }
```

Listing 10.1: Migracja tabeli users

Nota autora

Powyższy kod to szablon, który wykorzystamy w rozdziale 18, gdy demo-app zostanie wydzielone jako osobny projekt i stworzymy pełne komendy do zarządzania migracjami. Na razie, jeśli chcesz użyć tej klasy, po prostu zamień wszystkie placeholdery na właściwe wartości.

Drugim krokiem jest w pełni **automatyczne** wygenerowanie modelu. Jedyne co musimy zrobić ręcznie to ustawić atrybuty zasilające naszą fabrykę danych.

```
1 php console/bin make:model User --table=users
```

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Database\ORM\Entities;
6
7 use Larafony\Framework\Cache\Cache;
8 use Larafony\Framework\Clock\ClockFactory;
9 use Larafony\Framework\Clock\Contracts\Clock;
10 use Larafony\Framework\Database\ORM\Attributes\
    BelongsToMany;
11 use Larafony\Framework\Database\ORM\Attributes\CastUsing;
12 use Larafony\Framework\Database\ORM\Model;
13
14 class User extends Model
15 {
16     public string $email {
17         get => $this->email;
18         set {
19             $this->email = $value;
20             $this->markPropertyAsChanged('email');
21         }
22     }
23     public string $username {
24         get => $this->username;
25         set {
26             $this->username = $value;
27             $this->markPropertyAsChanged('username');
28         }
29     }
30     public string $password {
31         get => $this->password;
32         set {
33             // Auto-hash password if not already hashed (
34                 Argon2id)
```

```
34         $this->password = str_starts_with($value, '
           $argon2') ? $value : password_hash($value,
           PASSWORD_ARGON2ID);
35         $this->markPropertyAsChanged('password');
36     }
37 }
38 public ?string $remember_token {
39     get => $this->remember_token;
40     set {
41         $this->remember_token = $value;
42         $this->markPropertyAsChanged('remember_token')
           ;
43     }
44 }
45 public ?string $password_reset_token {
46     get => $this->password_reset_token;
47     set {
48         $this->password_reset_token = $value;
49         $this->markPropertyAsChanged('
           password_reset_token');
50     }
51 }
52 //this is just for now, in final version it will
    attribute based casting to Clock instance
53 public ?string $password_reset_expires {
54     get => $this->password_reset_expires;
55     set {
56         $this->password_reset_expires = $value;
57         $this->markPropertyAsChanged('
           password_reset_expires');
58     }
59 }
60 public ?Clock $email_verified_at {
61     get => $this->email_verified_at;
62     set {
63         $this->email_verified_at = $value;
64         $this->markPropertyAsChanged('
           email_verified_at');
65     }
66 }
67 public int $is_active {
```

```
68         get => $this->is_active;
69         set {
70             $this->is_active = $value;
71             $this->markPropertyAsChanged('is_active');
72         }
73     }
74     //enough for now, other properties will be added after
75     //full support of ORM is done
76     //continue reading ...
77 }
```

Listing 10.2: Model User

Nota autora

Użyty w kodzie powyżej atrybut `FakeAs` przyjmuje jako parametr enum ze zdefiniowanymi metodami dla klasy `Faker/Generator`. Natomiast sam model w pełni czerpie z wprowadzonego w PHP 8.4 mechanizmu Property Hooks pozwalającego na śledzenie prawdziwych pól bez użycia metod magicznych `__get` i `__set`.

Nota autora

Pełne uruchomienie tej klasy będzie możliwe dopiero po skończeniu rozdziału 13 gdzie k

Ostatnim krokiem jest stworzenie seedera i wypełnienie bazy danych:

```
1 php console/bin make:seeder UserSeeder
2 php console/bin database:seed
```

```
1 <?php
2
3 declare(strict_types=1);
4
```

```
5 namespace App\Database\Seeders;
6
7 use App\Models\User;
8 use Larafony\Framework\Database\ORM\Entities\Permission;
9 use Larafony\Framework\Database\ORM\Entities\Role;
10
11 class UserSeeder
12 {
13     public function run(): void
14     {
15         // Create admin user
16         $admin = new User();
17         $admin->email = 'admin@example.com';
18         $admin->username = 'admin';
19         $admin->password = 'password'; // Auto-hashed with
20             Argon2id
21         $admin->is_active = 1;
22         $admin->save();
23
24         // Create regular user
25         $user = new User();
26         $user->email = 'user@example.com';
27         $user->username = 'user';
28         $user->password = 'password'; // Auto-hashed with
29             Argon2id
30         $user->is_active = 1;
31         $user->save();
32
33         // Create roles
34         $adminRole = new Role();
35         $adminRole->name = 'admin';
36         $adminRole->description = 'Administrator role';
37         $adminRole->save();
38
39         $userRole = new Role();
40         $userRole->name = 'user';
41         $userRole->description = 'Regular user role';
42         $userRole->save();
43
44         // Create permissions
45         $addNotePermission = new Permission();
```

```
44     $addNotePermission->name = 'notes.create';
45     $addNotePermission->description = 'Can create
46         notes';
47     $addNotePermission->save();
48
49     $editNotePermission = new Permission();
50     $editNotePermission->name = 'notes.edit';
51     $editNotePermission->description = 'Can edit notes
52         ';
53     $editNotePermission->save();
54
55     $deleteNotePermission = new Permission();
56     $deleteNotePermission->name = 'notes.delete';
57     $deleteNotePermission->description = 'Can delete
58         notes';
59     $deleteNotePermission->save();
60
61     // Assign roles to users
62     $admin->addRole($adminRole);
63     $user->addRole($userRole);
64 }
```

Listing 10.3: Seeder UserSeeder

10.1 Wprowadzenie do DBAL

Baza danych stanowi serce każdej aplikacji, w której przechowywanie i zarządzanie danymi ma kluczowe znaczenie. W naszym frameworku podejście do obsługi bazy danych oparte jest na DBAL – warstwie abstrakcji nad bazą danych, pozwalającej na eleganckie i przejrzyste zarządzanie schematem oraz operacjami na danych.

DBAL to koncepcja pozwalająca na abstrakcyjne podejście do operacji bazodanowych, odczytując je jako zestaw obiektów i operacji, a nie jako surowe zapytania SQL. Naszym celem jest stworzenie rozwiązania, które nie tylko ułatwi operacje na bazie danych, ale także pozwoli na łatwą rozbudowę i obsługę

różnych silników bazodanowych.

Nota autora

Architektura naszego DBAL jest inspirowana Laravelem i opiera się na wzorcu **Grammar**. Dzięki temu możemy łatwo dodawać wsparcie dla różnych baz danych – wystarczy stworzyć nową klasę **Grammar** dla danego silnika (np. PostgreSQL, SQLite), bez modyfikacji reszty kodu.

10.2 Architektura DBAL

Nasz system DBAL składa się z następujących komponentów:

- **ConnectionContract** – interfejs definiujący połączenie z bazą danych
- **GrammarContract** – interfejs kompilatora SQL dla danego silnika
- **SchemaBuilder** – główna klasa do zarządzania schematem
- **TableDefinition** – fluent API do definiowania struktury tabeli
- **BaseColumn** i klasy pochodne – definicje kolumn z metodami fluent
- **IndexDefinition** i klasy pochodne – definicje indeksów

Struktura katalogów odzwierciedla tę architekturę:

```
1 Database/
2 +-- Base/                                # Abstrakcyjne klasy bazowe
3 |   +-- Contracts/                      # Interfejsy
4 |   +-- Schema/                        # Definicje schematów
5 |   |   +-- Columns/                  # Bazowe klasy kolumn
6 |   |   +-- IndexDefinitions/
7 |   |   +-- Builders/
8 |   +-- Query/                        # QueryBuilder (rozdział 11)
9 +-- Drivers/
10     +-- MySQL/                        # Implementacja dla MySQL
```

```

11      +-- Schema/
12      |      +-- ColumnDefinitions/
13      |      +-- IndexDefinitions/
14      |      +-- Builders/
15      +-- Query/

```

10.3 Interfejs ConnectionContract

Podstawą każdej warstwy komunikacji z bazą danych jest połączenie. Interfejs `ConnectionContract` definiuje metody, które każda implementacja musi obsługiwać:

```

1  <?php
2
3  declare(strict_types=1);
4
5  namespace Larafony\Framework\Database\Base\Contracts;
6
7  interface ConnectionContract
8  {
9      public function connect(): void;
10
11      /**
12       * @param array<string|int, string|float|int|null>
13         $params
14       */
15      public function query(string $sql, array $params = [])
16         : \PDOStatement;
17
18      /**
19       * @return array<int, int|false>
20       */
21      public function getConnectionOptions(): array;
22
23      public function getLastInsertId(): ?string;

```

```
22
23     public function quote(int|float|string|bool|null
24         $value): string;
25
26     /**
27      * Execute a SELECT query and return all results
28      * Cursor is automatically closed after fetching
29      *
30      * @param string $sql
31      * @param array<int, mixed> $params
32      *
33      * @return array<int, array<string, mixed>>
34      */
35     public function select(string $sql, array $params =
36         []): array;
37
38     /**
39      * Execute an INSERT/UPDATE/DELETE query and return
40      * affected rows
41      * Cursor is automatically closed after getting row
42      * count
43      *
44      * @param string $sql
45      * @param array<int, mixed> $params
46      *
47      * @return int Number of affected rows
48      */
49     public function execute(string $sql, array $params =
50         []): int;
51 }
```

Listing 10.4: Interfejs ConnectionContract

Nota autora

Zwróć uwagę na metody `select()` i `execute()`. To uproszczenie typowych operacji: `select()` automatycznie zamyka kursor po pobraniu wyników, `execute()` zwraca liczbę zmienionych wierszy. Dzięki temu unikamy wycieków pamięci związanych z otwartymi kursorami.

10.4 Wzorzec Grammar

Serce naszego DBAL to wzorzec **Grammar** – klasa odpowiedzialna za kompilację obiektowych definicji tabel do surowego SQL. Dzięki temu rozdzieleniu logika definiowania schematu jest niezależna od składni konkretnego silnika bazy danych.

Nota autora

Wzorzec Grammar to jedna z najlepszych decyzji architektonicznych w Laravelu. Zamiast generować SQL bezpośrednio w **SchemaBuilder**, delegujemy to do osobnej klasy. Korzyści:

- **Separacja odpowiedzialności:** **SchemaBuilder** wie co robić (twórz tabelę, modyfikuj kolumnę), **Grammar** wie *jak* to zapisać w SQL
- **Łatwe testowanie:** **Grammar** to czysta funkcja – wchodzi **TableDefinition**, wychodzi string SQL. Bez mocków, bez bazy danych
- **Wsparcie wielu baz:** Nowa baza = nowa klasa **Grammar**. Zero zmian w istniejącym kodzie

W praktyce oznacza to, że Twoje migracje napisane dla MySQL zadziałają na PostgreSQL po zmianie jednej linijki konfiguracji – zakładając, że zaimplementujesz **PostgreSQL/Grammar**.

10.4.1 Interfejs GrammarContract

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Database\Base\Contracts;
6
7 use Larafony\Framework\Database\Base\Schema\
  TableDefinition;
```

```
8
9 interface GrammarContract
10 {
11     public function compileCreate(TableDefinition $table):
        string;
12     public function compileAddColumns(TableDefinition
        $table): string;
13     public function compileModifyColumns(TableDefinition
        $table): string;
14     public function compileDropColumns(TableDefinition
        $table): string;
15     public function compileDropTable(string $table, bool
        $ifExists = false): string;
16 }
```

Listing 10.5: Interfejs GrammarContract

10.4.2 Implementacja MySQL Grammar

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Database\Drivers\MySQL;
6
7 use Larafony\Framework\Database\Base\Contracts\
    GrammarContract;
8 use Larafony\Framework\Database\Base\Schema\
    TableDefinition;
9 use Larafony\Framework\Database\Drivers\MySQL\Schema\
    Builders\AddColumns;
10 use Larafony\Framework\Database\Drivers\MySQL\Schema\
    Builders\ChangeColumns;
11 use Larafony\Framework\Database\Drivers\MySQL\Schema\
    Builders\CreateTableBuilder;
12 use Larafony\Framework\Database\Drivers\MySQL\Schema\
    Builders\DropColumns;
```

```
13
14 class Grammar implements GrammarContract
15 {
16     public function compileCreate(TableDefinition $table):
17         string
18     {
19         return new CreateTableBuilder()->build($table);
20     }
21     public function compileAddColumns(TableDefinition
22         $table): string
23     {
24         return new AddColumns()->build($table);
25     }
26     public function compileModifyColumns(TableDefinition
27         $table): string
28     {
29         return new ChangeColumns()->build($table);
30     }
31     public function compileDropColumns(TableDefinition
32         $table): string
33     {
34         return new DropColumns()->build($table);
35     }
36     public function compileDropTable(string $table, bool
37         $ifExists = false): string
38     {
39         return sprintf('DROP TABLE %s%s', $ifExists ? 'IF
40             EXISTS ' : '', $table);
41     }
42 }
```

Listing 10.6: Klasa MySQL Grammar

Nota autora

Klasa `Grammar` deleguje kompilację do wyspecjalizowanych builderów. To kolejny poziom rozdzielania odpowiedzialności – `Grammar` wie co kompilować, buildery wiedzą *jak*. Dzięki temu dodanie nowego typu operacji (np. `RENAME TABLE`) wymaga tylko nowego buildera.

10.5 TableDefinition – fluent API

Klasa `TableDefinition` to serce naszego fluent API do definiowania tabel. Pozwala na czytelne i intuicyjne definiowanie struktury tabeli:

```
1 $schema->create('users', function (TableDefinition $table)
2     {
3         $table->id();
4         $table->string('name');
5         $table->string('email')->unique();
6         $table->timestamps();
7     });
```

10.5.1 MySQL TableDefinition

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Database\Drivers\MySQL\Schema
6     ;
7
8 use Larafony\Framework\Database\Drivers\MySQL\Schema\
9     ColumnDefinitions\DateTimeColumn;
```

```
8 use Larafony\Framework\Database\Drivers\MySQL\Schema\
   ColumnDefinitions\EnumColumn;
9 use Larafony\Framework\Database\Drivers\MySQL\Schema\
   ColumnDefinitions\IntColumn;
10 use Larafony\Framework\Database\Drivers\MySQL\Schema\
   ColumnDefinitions\StringColumn;
11 use Larafony\Framework\Database\Drivers\MySQL\Schema\
   ColumnDefinitions\TextColumn;
12 use Larafony\Framework\Database\Drivers\MySQL\Schema\
   IndexDefinitions\NormalIndex;
13 use Larafony\Framework\Database\Drivers\MySQL\Schema\
   IndexDefinitions\PrimaryIndex;
14 use Larafony\Framework\Database\Drivers\MySQL\Schema\
   IndexDefinitions\UniqueIndex;
15
16 class TableDefinition extends \Larafony\Framework\Database
   \Base\Schema\TableDefinition
17 {
18     public function integer(string $column, string $type =
       'INT'): IntColumn
19     {
20         $col = new IntColumn($column, type: $type);
21         $this->addColumn($col);
22         return $col;
23     }
24
25     public function bigInteger(string $column, string
       $type = 'MEDIUMINT'): IntColumn
26     {
27         return $this->integer($column, $type)->length(20);
28     }
29     public function smallInteger(string $column, string
       $type = 'MEDIUMINT'): IntColumn
30     {
31         return $this->integer($column, $type)->length(6);
32     }
33     public function string(string $column, int $length =
       255, string $type = 'VARCHAR'): StringColumn
34     {
35         $col = new StringColumn($column, length: $length,
           type: $type);
```

```
36     $this->addColumn($col);
37     return $col;
38 }
39 public function char(string $column, int $length =
255, string $type = 'CHAR'): StringColumn
40 {
41     return $this->string($column, $length, $type);
42 }
43 public function text(string $column, string $type = '
TEXT'): TextColumn
44 {
45     $col = new TextColumn($column, type: $type);
46     $this->addColumn($col);
47     return $col;
48 }
49 public function mediumText(string $column, string
$type = 'MEDIUMTEXT'): TextColumn
50 {
51     return $this->text($column, $type);
52 }
53 public function longText(string $column, string $type
= 'LONGTEXT'): TextColumn
54 {
55     return $this->text($column, $type);
56 }
57 public function json(string $column, string $type = '
JSON'): TextColumn
58 {
59     return $this->text($column, $type);
60 }
61 public function dateTime(string $column, string $type
= 'DATETIME'): DateTimeColumn
62 {
63     $col = new DateTimeColumn($column, type: $type);
64     $this->addColumn($col);
65     return $col;
66 }
67 public function timestamp(string $column, string $type
= 'TIMESTAMP'): DateTimeColumn
68 {
69     return $this->dateTime($column, $type);
```

```

70     }
71     public function time(string $column, string $type = '
72         TIME'): DateTimeColumn
73     {
74         return $this->dateTime($column, $type);
75     }
76     public function date(string $column, string $type = '
77         DATE'): DateTimeColumn
78     {
79         return $this->dateTime($column, $type);
80     }
81     /**
82      * @param string|array<int, string> $columns
83      */
84     public function index(string|array $columns, ?string
85         $indexName = null): NormalIndex
86     {
87         $index = new NormalIndex($this->tableName,
88             $columns, $indexName);
89         $this->addIndex($index);
90         return $index;
91     }
92     /**
93      * @param string|array<int, string> $columns
94      */
95     public function primary(string|array $columns, ?string
96         $indexName = null): PrimaryIndex
97     {
98         $index = new PrimaryIndex($this->tableName,
99             $columns, $indexName, type: 'primary');
100         $this->addIndex($index);
101         return $index;
102     }
103     /**
104      * @param string|array<int, string> $columns
105      */
106     public function unique(string|array $columns, ?string
107         $indexName = null): UniqueIndex
108     {

```

```
103         $index = new UniqueIndex($this->tableName,  
104             $columns, $indexName, type: 'unique');  
105         $this->addIndex($index);  
106         return $index;  
107     }  
108     /**  
109      * @param array<int, string> $values  
110      */  
111     public function enum(string $column, array $values):  
112         EnumColumn  
113     {  
114         $col = new EnumColumn($column, $values);  
115         $this->addColumn($col);  
116         return $col;  
117     }  
}
```

Listing 10.7: Klasa MySQL TableDefinition

Nota autora

Bazowa klasa `TableDefinition` nie została tu przytoczona, znajdziesz ją w repozytorium. Jeśli chcesz, spróbuj brakujące metody (PHPStorm powinien je wykryć), zaimplementować sam - powodzenia!

Nota autora

Metody takie jak `id()`, `timestamps()`, `softDeletes()` to skróty dla często używanych wzorców. `id()` tworzy kolumnę `INT AUTO_INCREMENT` z kluczem głównym. `timestamps()` dodaje `created_at` i `updated_at`. To podejście znane z Laravela, które znacząco przyspiesza definiowanie migracji.

10.6 Klasy kolumn

Każdy typ danych MySQL jest reprezentowany przez dedykowaną klasę dziedziczącą po `BaseColumn`. Klasy te oferują fluent API do konfiguracji kolumny.

Nota autora

PHP 8.5 wprowadza nową składnię do klonowania obiektów: `clone($obj, ['field' => value])`. Pozwala ona na tworzenie niemutowalnych (immutable) obiektów z fluent API – każda metoda zwraca nową instancję zamiast modyfikować istniejącą.

Jednak w przypadku klas definiujących typy kolumn świadomie wybieramy mutowalność. Dlaczego? Bo niemutowalne podejście wymagałoby, żeby `TableDefinition` śledziło każdą zwróconą instancję:

```
1 // Immutable – skomplikowane, łatwo o błąd
2 $col = $table->integer('age');
3 $col = $col->unsigned(true); // nowa instancja!
4 $col = $col->nullable(false); // kolejna nowa instancja
5 !
6 // $table nadal ma starą referencję... plus potencjalne
7 // wycieki pamięci
8
9 // Mutable – proste i intuicyjne
10 $table->integer('age')->unsigned(true)->nullable(false)
11 ;
12 // $table automatycznie ma zaktualizowaną kolumnę
```

W kontekście definiowania schematu bazy danych mutowalność jest pragmatycznym wyborem – obiekt `TableDefinition` żyje tylko przez czas wykonania callbacka, więc ryzyko niechcianych efektów ubocznych jest minimalne.

10.6.1 Bazowa klasa `BaseColumn`

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Database\Base\Schema\Columns;
6
7 abstract class BaseColumn
8 {
9     public protected(set) bool $modified = false;
10    public protected(set) bool $deleted = false;
11    public protected(set) bool $existsInDatabase = false;
12
13    public function __construct(
14        public readonly string $name,
15        public readonly string $type,
16        public bool $nullable = true,
17        protected mixed $default = null,
18    ) {
19    }
20
21    public function markAsExisting(): static
22    {
23        $this->existsInDatabase = true;
24        return $this;
25    }
26    public function change(): static
27    {
28        $this->modified = true;
29        return $this;
30    }
31    public function delete(): static
32    {
33        $this->deleted = true;
34        return $this;
35    }
36    public function nullable(bool $nullable): static
37    {
38        $this->nullable = $nullable;
39        return $this;
40    }
41 }
```

```

42     abstract public function getSqlDefinition(): string;
43     /**
44      * @param array<string, mixed> $description
45      */
46     abstract public static function fromArrayDescription(
47         array $description): static;
48     abstract protected function getNullableSqlDefinition()
49         : string;
50 }

```

Listing 10.8: Klasa BaseColumn

Nota autora

Zwróć uwagę na flagi `modified`, `deleted` i `existsInDatabase`. Pozwalają one na rozróżnienie między nowymi kolumnami (do dodania), istniejącymi (do modyfikacji) i usuniętymi (do DROP). `SchemaBuilder` używa tych flag do generowania odpowiednich zapytań ALTER TABLE.

10.6.2 IntColumn – kolumna liczbowa

```

1  <?php
2
3  declare(strict_types=1);
4
5  namespace Larafony\Framework\Database\Base\Schema\Columns;
6
7  abstract class IntColumn extends BaseColumn
8  {
9      public function __construct(
10         string $name,
11         bool $nullable = true,
12         mixed $default = null,
13         public int $length = 11,
14         public bool $unsigned = false,
15         public bool $autoIncrement = false,

```

```
16         string $type = 'INT',
17     ) {
18         parent::__construct($name, $type, $nullable,
19             $default);
20     }
21
22     public function unsigned(bool $unsigned): static
23     {
24         $this->unsigned = $unsigned;
25         return $this;
26     }
27
28     public function length(int $length): static
29     {
30         $this->length = $length;
31         return $this;
32     }
33
34     public function default(mixed $default): static
35     {
36         $this->default = $default;
37         return $this;
38     }
39
40     public function autoIncrement(bool $autoIncrement):
41         static
42     {
43         $this->autoIncrement = $autoIncrement;
44         return $this;
45     }
46
47     abstract protected function getUnsignedSqlDefinition()
48         : string;
49     abstract protected function
50         getAutoIncrementSqlDefinition(): string;
51     abstract protected function
52         getDefaultValueSqlDefinition(): string;
53 }
```

Listing 10.9: Klasa IntColumn

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Database\Drivers\MySQL\Schema
   \ColumnDefinitions;
6
7 class IntColumn extends \Larafony\Framework\Database\Base\
   Schema\Columns\IntColumn
8 {
9     public function getSqlDefinition(): string
10     {
11         $sql = "{$this->name} {$this->type}({$this->length
           });";
12         $sql .= $this->getUnsignedSqlDefinition();
13         $sql .= $this->getNullableSqlDefinition();
14         $sql .= $this->getDefaultValueSqlDefinition();
15         return $sql . $this->getAutoIncrementSqlDefinition
           ();
16     }
17
18     /**
19      * @param array<string, mixed> $description
20      */
21     public static function fromArrayDescription(array
       $description): static
22     {
23         preg_match('/int\(((\d+)\))/i', $description['Type']
           ], $matches);
24         $length = $matches[1] ?? 11;
25         $unsigned = str_contains($description['Type'], '
           unsigned');
26         $autoIncrement = $description['Extra'] === '
           auto_increment';
27
28         // Extract base type without length
29         preg_match('/^([a-z]+)/i', $description['Type'],
           $typeMatches);
30         $baseType = strtoupper($typeMatches[1] ?? 'INT');
31
32         return new IntColumn(
```

```
33         $description['Field'],
34         $description['Null'] === 'YES',
35         $description['Default'],
36         (int) $length,
37         $unsigned,
38         $autoIncrement,
39         $baseType
40     );
41 }
42
43 protected function getNullableSqlDefinition(): string
44 {
45     return $this->nullable ? ' NULL' : ' NOT NULL';
46 }
47
48 protected function getUnsignedSqlDefinition(): string
49 {
50     return $this->unsigned ? ' UNSIGNED' : '';
51 }
52
53 protected function getAutoIncrementSqlDefinition():
54     string
55 {
56     return $this->autoIncrement ? ' AUTO_INCREMENT' :
57         '';
58 }
59
60 protected function getDefaultValueSqlDefinition():
61     string
62 {
63     return $this->default !== null ? " DEFAULT {$this
64         ->default}" : '';
65 }
66 }
```

Listing 10.10: Klasa IntColumn

Nota autora

Zwróć uwagę że bazowa kolumna int definiuje to co nie zależy od składni SQL (np że unsigned =true ma ustawić unsigned i tyle) a kolumna docelowa z mysql buduje definicję kolumny w tym języku

10.6.3 StringColumn – kolumna tekstowa

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Database\Drivers\MySQL\Schema
   \ColumnDefinitions;
6
7 class StringColumn extends \Larafony\Framework\Database\
   Base\Schema\Columns\StringColumn
8 {
9     public function getSqlDefinition(): string
10     {
11         $sql = "{$this->name} {$this->type}({$this->length
           }) ";
12         $sql .= $this->getNullableSqlDefinition();
13         return trim($sql . $this->
           getDefaultValueSqlDefinition());
14     }
15
16     /**
17      * @param array<string, mixed> $description
18      */
19     public static function fromArrayDescription(array
       $description): static
20     {
21         preg_match('/varchar|char\((\d+)\)/i',
           $description['Type'], $matches);
22         $length = $matches[1] ?? 255;
23
24         // Extract base type without length
```

```
25     preg_match('/^([a-z]+)/i', $description['Type'],
26         $typeMatches);
27     $baseType = strtoupper($typeMatches[1] ?? 'VARCHAR
28         ');
29
30     return new StringColumn(
31         $description['Field'],
32         $description['Null'] === 'YES',
33         $description['Default'],
34         (int) $length,
35         $baseType
36     );
37
38     protected function getNullableSqlDefinition(): string
39     {
40         return $this->nullable ? 'NULL ' : 'NOT NULL ';
41     }
42
43     protected function getDefaultValueSqlDefinition():
44         string
45     {
46         return $this->default !== null ? "DEFAULT '{$this
47             ->default}' " : ' ';
48     }
49 }
```

Listing 10.11: Klasa StringColumn

Nota autora

Klasa bazowa wygląda niemal identycznie jak w przypadku `inta` - spróbuj sam lub pobierz z repo (zalecam pierwsze)

10.6.4 DateTimeColumn – kolumna daty i czasu

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Database\Drivers\MySQL\Schema
   \ColumnDefinitions;
6
7 class DateTimeColumn extends \Larafony\Framework\Database\
   Base\Schema\Columns\DateTimeColumn
8 {
9     public function getSqlDefinition(): string
10     {
11         $sql = "{$this->name} {$this->type} ";
12         $sql .= $this->getNullableSqlDefinition();
13         $sql .= $this->getDefaultValueDefinition();
14         return trim($sql . $this->getOnUpdateDefinition())
15             ;
16     }
17
18     public function getDefaultValueDefinition(): string
19     {
20         return $this->default !== null ? "DEFAULT {$this->
21             default} " : '';
22     }
23
24     public function getOnUpdateDefinition(): string
25     {
26         return $this->onUpdate !== null ? 'ON UPDATE
27             CURRENT_TIMESTAMP' : '';
28     }
29
30     /**
31      * @param array<string, mixed> $description
32      */
33     public static function fromArrayDescription(array
34         $description): static
35     {
36         $onUpdate = str_contains($description['Extra'] ??
37             '', 'on update') ? $description['Extra'] : null
38             ;
39     }
40 }
```

```
33     $precision = 0; // MySQL DESCRIBE doesn't return
34         precision, would need to parse from Type
35
36     return new DateTimeColumn(
37         $description['Field'],
38         $description['Null'] === 'YES',
39         $description['Default'],
40         $onUpdate,
41         $precision,
42         $description['Type'],
43     );
44
45     protected function getNullableSqlDefinition(): string
46     {
47         return $this->nullable ? 'NULL ' : 'NOT NULL ';
48     }
49 }
```

Listing 10.12: Klasa DateTimeColumn

10.6.5 EnumColumn – kolumna z predefiniowanymi wartościami

```
1  <?php
2
3  declare(strict_types=1);
4
5  namespace Larafony\Framework\Database\Drivers\MySQL\Schema
6      \ColumnDefinitions;
7
8  class EnumColumn extends \Larafony\Framework\Database\Base
9      \Schema\Columns\EnumColumn
10 {
11     public function getSqlDefinition(): string
12     {
```

```

11         $values = implode(',', array_map(static fn ($v) =>
12             "'{$v}'" , $this->values));
13         $sql = "{$this->name} {$this->type}({$values}) ";
14         $sql .= $this->getNullableSqlDefinition();
15         return trim($sql . $this->getDefaultDefinition());
16     }
17
18     /**
19      * @param array<string, mixed> $description
20      */
21     public static function fromArrayDescription(array
22         $description): static
23     {
24         preg_match('/enum\(((.*)\))/i', $description['Type']
25             ], $matches);
26         $values = str_getcsv($matches[1] ?? '', escape: ''
27             );
28         $values = array_filter($values, static fn (string|
29             null $value) => (bool) $value);
30         $values = array_map(
31             static fn (string $value) => str_replace('"',
32                 "'", $value),
33             $values
34         );
35         return new EnumColumn(
36             $description['Field'],
37             $values,
38             $description['Null'] === 'YES',
39             $description['Default']
40         );
41     }
42
43     protected function getNullableSqlDefinition(): string
44     {
45         return $this->nullable ? 'NULL ' : 'NOT NULL ';
46     }
47
48     protected function getDefaultDefinition(): string
49     {
50         return $this->default !== null ? "DEFAULT '{$this
51             ->default}' " : '';

```

```
45     }  
46 }
```

Listing 10.13: Klasa EnumColumn

Nota autora

W przypadku typu **ENUM** walidacja domyślnej wartości jest kluczowa. Jeśli ustawisz wartość domyślną spoza listy dozwolonych wartości, błąd pojawi się dopiero przy wykonaniu migracji – a wtedy debug jest znacznie trudniejszy. Walidacja w konstruktorze wykrywa błąd natychmiast.

10.7 Indeksy

MySQL obsługuje trzy podstawowe typy indeksów:

- **PRIMARY KEY** – unikalny identyfikator wiersza, jeden na tabelę
- **UNIQUE** – zapewnia unikalność wartości w kolumnie
- **INDEX** – przyspiesza wyszukiwanie, bez gwarancji unikalności

10.7.1 Bazowa klasa IndexDefinition

```
1 <?php  
2  
3 declare(strict_types=1);  
4  
5 namespace Larafony\Framework\Database\Base\Schema\  
6     IndexDefinitions;  
7  
8 abstract readonly class IndexDefinition  
9 {
```

```

9      public string $columns;
10
11     public string $indexName;
12
13     /**
14      * @param array<int, string>|string $columns
15      */
16     public function __construct(
17         public string $tableName,
18         array|string $columns,
19         ?string $indexName = null,
20         string $type = 'index'
21     ) {
22         $this->indexName = $indexName ?? $this->
            generateIndexName($type, $columns);
23         $this->columns = is_array($columns) ? implode(', '
            , $columns) : $columns;
24     }
25
26     abstract public function getSqlDefinition(): string;
27
28     /**
29      * Get inline SQL definition for use within CREATE
        TABLE statement
30      */
31     abstract public function getInlineSqlDefinition():
        string;
32
33     /**
34      * @param array<int, string>|string $columns
35      */
36     protected function generateIndexName(string $type,
        array|string $columns): string
37     {
38         $columns = is_array($columns) ? implode('_',
            $columns) : $columns;
39         $index = strtolower($this->tableName . '_' .
            $columns . '_' . $type);
40
41         return str_replace(['-', '.'], '_', $index);
42     }

```

43 }

Listing 10.14: Klasa IndexDefinition

10.7.2 PrimaryIndex

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Database\Base\Schema\
   IndexDefinitions;
6
7 abstract readonly class PrimaryIndex extends
   IndexDefinition
8 {
9 }
```

Listing 10.15: Klasa PrimaryIndex

Nota autora

Indeksy mają dwie formy SQL: `getSqlDefinition()` dla ALTER TABLE i `getInlineSqlDefinition()` dla CREATE TABLE. W CREATE TABLE indeksy są częścią definicji tabeli, w ALTER TABLE są osobnymi poleceniami.

10.8 SchemaBuilder – orkiestrator

Klasa `SchemaBuilder` łączy wszystkie elementy w jedną całość. Przyjmuje callback definiujący strukturę tabeli, kompiluje go przez `Grammar` i opcjonalnie wykonuje wygenerowany SQL.

10.8.1 Bazowa klasa SchemaBuilder

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Database\Base\Schema;
6
7 use Closure;
8 use Larafony\Framework\Database\Base\Contracts\
    ConnectionContract;
9 use Larafony\Framework\Database\Base\Contracts\
    SchemaBuilderContract;
10 use Larafony\Framework\Database\Drivers\MySQL\Grammar;
11
12 abstract class SchemaBuilder implements
    SchemaBuilderContract
13 {
14     protected Grammar $grammar;
15     public function __construct(protected readonly
        ConnectionContract $connection)
16     {
17     }
18
19     #[\NoDiscard]
20     abstract public function create(string $table, Closure
        $callback): string;
21     #[\NoDiscard]
22     abstract public function table(string $table, Closure
        $callback): string;
23     #[\NoDiscard]
24     abstract public function drop(string $table): string;
25     #[\NoDiscard]
26     abstract public function dropIfExists(string $table):
        string;
27
28     public function execute(string $sql): bool
29     {
30         $this->connection->query($sql);
31         return true;
```

```
32     }
33
34     /**
35      * @return array<int, string>
36      */
37     abstract public function getColumnListing(string
        $table): array;
38 }
```

Listing 10.16: Klasa SchemaBuilder (bazowa)

10.8.2 MySQL SchemaBuilder

```
1  <?php
2
3  declare(strict_types=1);
4
5  namespace Larafony\Framework\Database\Drivers\MySQL;
6
7  use Closure;
8  use Larafony\Framework\Database\Base\Contracts\
    ConnectionContract;
9  use Larafony\Framework\Database\Drivers\MySQL\Schema\
    DatabaseInfo;
10 use Larafony\Framework\Database\Drivers\MySQL\Schema\
    TableDefinition;
11
12 class SchemaBuilder extends \Larafony\Framework\Database\
    Base\Schema\SchemaBuilder
13 {
14     public function __construct(ConnectionContract
        $connection)
15     {
16         parent::__construct($connection);
17         $this->grammar = new Grammar();
18     }
19 }
```

```
20     #[\NoDiscard]
21     public function create(string $table, Closure
22         $callback): string
23     {
24         $tableDefinition = new TableDefinition($table);
25         $callback($tableDefinition);
26         return $this->grammar->compileCreate(
27             $tableDefinition);
28     }
29     #[\NoDiscard]
30     public function table(string $table, Closure $callback
31         ): string
32     {
33         $tableDefinition = new DatabaseInfo($this->
34             connection)->getTable($table);
35         $callback($tableDefinition);
36
37         $statements = array_filter([
38             $this->grammar->compileAddColumns(
39                 $tableDefinition),
40             $this->grammar->compileModifyColumns(
41                 $tableDefinition),
42             $this->grammar->compileDropColumns(
43                 $tableDefinition),
44         ]);
45
46         return implode('; ' . PHP_EOL, $statements);
47     }
48
49     #[\NoDiscard]
50     public function drop(string $table): string
51     {
52         return $this->grammar->compileDropTable($table);
53     }
54
55     #[\NoDiscard]
56     public function dropIfExists(string $table): string
57     {
58         return $this->grammar->compileDropTable($table,
59             ifExists: true);
60     }
```

```
53     }
54
55     /**
56      * @return array<int, string>
57      */
58     #[\NoDiscard]
59     public function getColumnListing(string $table): array
60     {
61         return new DatabaseInfo($this->connection)->
62             getTable($table)->columnNames;
63     }
64 }
```

Listing 10.17: Klasa MySQL SchemaBuilder

Nota autora

Zwróć uwagę na atrybut `#[NoDiscard]` przy metodach `create()`, `table()`, itp. To świadoma decyzja projektowa – metody te **jedynie zwracają string SQL**, nie wykonują żadnych zmian na bazie danych. Dlaczego tak? Bo rozdzielenie generowania SQL od jego wykonania daje elastyczność:

```
1 // Podgląd wygenerowanego SQL (np. do debugowania)
2 $sql = Schema::create('users', fn($t) => $t->id());
3 dump($sql);
4
5 // Wykonanie SQL
6 Schema::execute($sql);
7
8 // Lub elegancko z pipe operatorem (PHP 8.5)
9 Schema::create('users', fn($t) => $t->id()) |> Schema::
    execute(...);
```

`#[NoDiscard]` informuje analizatory statyczne (PHPStan, Psalm) i IDE, że ignorowanie zwracanej wartości to prawdopodobnie błąd – programista

zapomniał coś zrobić z wygenerowanym SQL. To kolejny przykład “fail fast” – błąd wykrywamy przy analizie statycznej, nie przy debugowaniu pustej bazy danych.

10.9 Buildery SQL

Kompilacja `TableDefinition` do SQL jest delegowana do wyspecjalizowanych builderów. To kluczowy element wzorca Grammar – buildery są “tłumaczami”, które przekształcają abstrakcyjną definicję tabeli w konkretny SQL dla danego silnika bazy danych.

Nota autora

Dlaczego wzorzec Grammar jest elegancki? Bo `TableDefinition` nie wie nic o SQL. To czysta struktura danych opisująca co chcemy osiągnąć. Buildery wiedzą *jak* to zrobić dla konkretnej bazy. Dzięki temu:

- Ten sam kod migracji działa na MySQL, PostgreSQL, SQLite
- Łatwo dodać wsparcie dla nowej bazy – piszesz nowe buildery, nie zmieniasz istniejącego kodu
- Testy są trywialne – przekazujesz `TableDefinition`, sprawdzasz wygenerowany string

To realizacja zasady Open/Closed z SOLID: system jest otwarty na rozszerzenia (nowe bazy), zamknięty na modyfikacje (istniejący kod nie wymaga zmian).

10.9.1 CreateTableBuilder

`CreateTableBuilder` generuje pełną instrukcję `CREATE TABLE`. Sprytnie wykorzystuje fakt, że każda kolumna w `TableDefinition` potrafi sama wygenerować swoją definicję SQL poprzez metodę `getSqlDefinition()`:

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Database\Drivers\MySQL\Schema
   \Builders;
6
7 use Larafony\Framework\Database\Base\Schema\Columns\
   BaseColumn;
8 use Larafony\Framework\Database\Base\Schema\
   IndexDefinitions\IndexDefinition;
9 use Larafony\Framework\Database\Base\Schema\
   TableDefinition;
10
11 class CreateTableBuilder extends \Larafony\Framework\
   Database\Base\Schema\Builders\Builders\
   CreateTableBuilder
12 {
13     public function build(TableDefinition $table): string
14     {
15         $tableName = $table->tableName;
16         $columns = $table->columns;
17         $columnDefinitions = array_map(static fn (
18             BaseColumn $column) => $column->
19             getSqlDefinition(), $columns);
20
21         // Add inline index definitions to CREATE TABLE
22         $inlineIndexes = $this->getInlineIndexes($table);
23
24         $allDefinitions = array_merge($columnDefinitions,
25             $inlineIndexes);
26
27         return sprintf(
28             'CREATE TABLE %s (%s);',
29             $tableName,
30             implode(', ', $allDefinitions),
31         );
32     }
33
34     /**
35      * Get inline index definitions for CREATE TABLE
```

```
statement
33  *
34  * @return array<int, string>
35  */
36  private function getInlineIndexes(TableDefinition
    $table): array
37  {
38      $indexes = $table->indexes;
39      return array_map(
40          static fn (IndexDefinition $index) => $index->
            getInlineSqlDefinition(),
41          $indexes,
42      );
43  }
44 }
```

Listing 10.18: Klasa CreateTableBuilder

Nota autora

Zauważ elegancję tego podejścia: builder nie musi wiedzieć, czy kolumna to `INT`, `VARCHAR` czy `ENUM`. Wywołuje `getSqlDefinition()` i dostaje gotowy fragment SQL. Polimorfizm w akcji – każda klasa kolumny sama wie, jak się “zserializować” do SQL.

10.9.2 AddColumns Builder

`AddColumns` generuje instrukcje `ALTER TABLE ... ADD COLUMN`. Wykorzystuje flagę `existsInDatabase` w kolumnach – przetwarza tylko te, które nie istnieją jeszcze w bazie:

```
1  <?php
2
3  declare(strict_types=1);
4
5  namespace Larafony\Framework\Database\Drivers\MySQL\Schema
```

```
\Builders;

6
7 use Larafony\Framework\Database\Base\Schema\Columns\
  BaseColumn;
8 use Larafony\Framework\Database\Base\Schema\
  TableDefinition;
9
10 class AddColumns extends \Larafony\Framework\Database\Base
  \Schema\Builders\Builders\AddColumns
11 {
12     #[\NoDiscard]
13     public function build(TableDefinition $table): string
14     {
15         $columns = $table->columns;
16         $columns = array_filter(
17             $columns,
18             static fn (BaseColumn $column) => ! $column->
                existsInDatabase && ! $column->modified &&
                ! $column->deleted
19         );
20         if ($columns === []) {
21             return '';
22         }
23         $tableName = $table->tableName;
24         $definitions = array_map(
25             static fn (BaseColumn $column) => 'ADD COLUMN
                ' . $column->getSqlDefinition(),
26             $columns
27         );
28         return sprintf('ALTER TABLE %s %s;', $tableName,
                implode(', ', $definitions));
29     }
30 }
```

Listing 10.19: Klasa AddColumns

10.9.3 ChangeColumns Builder

`ChangeColumns` generuje instrukcje `ALTER TABLE ... MODIFY COLUMN`. Przetwarza tylko kolumny oznaczone flagą `modified`:

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Database\Drivers\MySQL\Schema
   \Builders;
6
7 use Larafony\Framework\Database\Base\Schema\Columns\
   BaseColumn;
8 use Larafony\Framework\Database\Base\Schema\
   TableDefinition;
9
10 class ChangeColumns extends \Larafony\Framework\Database\
   Base\Schema\Builders\Builders\ChangeColumns
11 {
12     #[\NoDiscard]
13     public function build(TableDefinition $table): string
14     {
15         $columns = $table->columns;
16         $columns = array_filter(
17             $columns,
18             static fn (BaseColumn $column) => $column->
               modified
19         );
20         if ($columns === []) {
21             return '';
22         }
23         $tableName = $table->tableName;
24         $definitions = array_map(
25             static fn (BaseColumn $column) => 'MODIFY
               COLUMN ' . $column->getSqlDefinition(),
26             $columns
27         );
28         return sprintf('ALTER TABLE %s %s;', $tableName,
               implode(', ', $definitions));
```

```

29     }
30 }

```

Listing 10.20: Klasa ChangeColumns

10.9.4 DropColumns Builder

`DropColumns` generuje instrukcje `ALTER TABLE ... DROP COLUMN`. Przetwarza tylko kolumny oznaczone flagą `deleted`:

```

1  <?php
2
3  declare(strict_types=1);
4
5  namespace Larafony\Framework\Database\Drivers\MySQL\Schema
6      \Builders;
7
8  use Larafony\Framework\Database\Base\Schema\Columns\
9      BaseColumn;
10 use Larafony\Framework\Database\Base\Schema\
11     TableDefinition;
12
13 class DropColumns extends \Larafony\Framework\Database\
14     Base\Schema\Builders\Builders\DropColumns
15 {
16     #[\NoDiscard]
17     public function build(TableDefinition $table): string
18     {
19         $columns = $table->columns;
20         $columns = array_filter(
21             $columns,
22             static fn (BaseColumn $column) => $column->
23                 deleted
24         );
25         if ($columns === []) {
26             return '';
27         }
28     }
29 }

```

```
23     $tableName = $table->tableName;
24     $modifiedColumns = array_map(
25         static fn (BaseColumn $column) => 'DROP COLUMN
26             ' . $column->name,
27         $columns
28     );
29     return 'ALTER TABLE ' . $tableName. ' ' . implode(' ',
30         $modifiedColumns);
31 }
```

Listing 10.21: Klasa DropColumns

Nota autora

Zwróć uwagę na spójność architektury: wszystkie buildery działają na tym samym `TableDefinition`, ale każdy patrzy na inne flagi. To pozwala na eleganckie rozdzielenie operacji ALTER TABLE na trzy niezależne części. Gdy wywołujesz `$schema->table('users', fn($t) => ...)`, `SchemaBuilder`:

1. Pobiera aktualną strukturę tabeli z bazy (metoda `getTable()`)
2. Wykonuje callback, który modyfikuje `TableDefinition`
3. Wywołuje wszystkie trzy buildery i łączy niepuste wyniki

Dzięki temu jeden callback może jednocześnie dodawać, modyfikować i usuwać kolumny – a system sam wygeneruje odpowiednie instrukcje SQL.

10.10 Testy jednostkowe

10.10.1 Testy klasy Grammar

```
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Tests\Database\MySQL;
6
7 use Larafony\Framework\Database\Drivers\MySQL\Grammar;
8 use Larafony\Framework\Database\Drivers\MySQL\Schema\
    ColumnDefinitions\IntColumn;
9 use Larafony\Framework\Database\Drivers\MySQL\Schema\
    ColumnDefinitions\StringColumn;
10 use Larafony\Framework\Database\Drivers\MySQL\Schema\
    TableDefinition;
11 use Larafony\Framework\Tests\TestCase;
12
13 class GrammarTest extends TestCase
14 {
15     private Grammar $grammar;
16
17     protected function setUp(): void
18     {
19         parent::setUp();
20         $this->grammar = new Grammar();
21     }
22
23     public function testCompilesCreateTable(): void
24     {
25         $table = new TableDefinition('users', [
26             'id' => new IntColumn('id', type: 'INT'),
27             'name' => new StringColumn('name', length:
                255, type: 'VARCHAR'),
28         ]);
29
30         $sql = $this->grammar->compileCreate($table);
31
32         $this->assertStringContainsString('CREATE TABLE
            users', $sql);
33         $this->assertStringContainsString('id', $sql);
34         $this->assertStringContainsString('name', $sql);
35     }
36
37     public function testCompilesAddColumns(): void
```

```
38     {
39         $table = new TableDefinition('users', [
40             'email' => new StringColumn('email', length:
41                 255, type: 'VARCHAR'),
42         ]);
43
44         $sql = $this->grammar->compileAddColumns($table);
45
46         $this->assertStringContainsString('ALTER TABLE
47             users', $sql);
48         $this->assertStringContainsString('ADD COLUMN',
49             $sql);
50         $this->assertStringContainsString('email', $sql);
51     }
52
53     public function
54     testCompilesAddColumnsReturnsEmptyForNoColumns():
55     void
56     {
57         $table = new TableDefinition('users', []);
58
59         $sql = $this->grammar->compileAddColumns($table);
60
61         $this->assertEmpty($sql);
62     }
63
64     public function testCompilesModifyColumns(): void
65     {
66         $emailColumn = new StringColumn('email', length:
67             100, type: 'VARCHAR');
68         $emailColumn->change();
69
70         $table = new TableDefinition('users', [
71             'email' => $emailColumn,
72         ]);
73
74         $sql = $this->grammar->compileModifyColumns($table
75             );
76
77         $this->assertStringContainsString('ALTER TABLE
78             users', $sql);
```

```
71         $this->assertStringContainsString('MODIFY COLUMN',
72             $sql);
73     }
74
75     public function
76     testCompilesModifyColumnsReturnsEmptyForNoModifiedColumns
77     (): void
78     {
79         $table = new TableDefinition('users', [
80             'email' => new StringColumn('email', length:
81                 255, type: 'VARCHAR'),
82         ]);
83
84         $sql = $this->grammar->compileModifyColumns($table
85             );
86
87         $this->assertEmpty($sql);
88     }
89
90     public function testCompilesDropColumns(): void
91     {
92         $emailColumn = new StringColumn('email', length:
93             255, type: 'VARCHAR');
94         $emailColumn->delete();
95
96         $table = new TableDefinition('users', [
97             'email' => $emailColumn,
98         ]);
99
100         $sql = $this->grammar->compileDropColumns($table);
101
102         $this->assertStringContainsString('ALTER TABLE
103             users', $sql);
104         $this->assertStringContainsString('DROP COLUMN
105             email', $sql);
106     }
107
108     public function
109     testCompilesDropColumnsReturnsEmptyForNoDeletedColumns
110     (): void
```

```
102     {
103         $table = new TableDefinition('users', [
104             'email' => new StringColumn('email', length:
105                 255, type: 'VARCHAR'),
106         ]);
107
108         $sql = $this->grammar->compileDropColumns($table);
109
110         $this->assertEmpty($sql);
111     }
112
113     public function testCompilesDropTable(): void
114     {
115         $sql = $this->grammar->compileDropTable('users',
116             false);
117
118         $this->assertStringContainsString('DROP TABLE',
119             $sql);
120         $this->assertStringContainsString('users', $sql);
121     }
122
123     public function testCompilesDropTableIfExists(): void
124     {
125         $sql = $this->grammar->compileDropTable('users',
126             true);
127
128         $this->assertStringContainsString('DROP TABLE',
129             $sql);
130         $this->assertStringContainsString('IF EXISTS',
131             $sql);
132         $this->assertStringContainsString('users', $sql);
133     }
134 }
```

Listing 10.22: GrammarTest

10.10.2 Testy IntColumn

```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Larafony\Framework\Tests\Database\MySQL;
6
7 use Larafony\Framework\Database\Drivers\MySQL\Schema\
    ColumnDefinitions\IntColumn;
8 use Larafony\Framework\Tests\TestCase;
9
10 class IntColumnTest extends TestCase
11 {
12     public function testCreatesIntColumn(): void
13     {
14         $column = new IntColumn('id', type: 'INT');
15
16         $this->assertEquals('id', $column->name);
17         $this->assertEquals('INT', $column->type);
18     }
19
20     public function testGeneratesSqlDefinition(): void
21     {
22         $column = new IntColumn('age', type: 'INT');
23         $sql = $column->getSqlDefinition();
24
25         $this->assertStringContainsString('age', $sql);
26         $this->assertStringContainsString('INT', $sql);
27     }
28
29     public function testSupportsLength(): void
30     {
31         $column = new IntColumn('count', type: 'INT');
32         $column = $column->length(10);
33         $sql = $column->getSqlDefinition();
34
35         $this->assertStringContainsString('INT(10)', $sql)
36             ;
37     }
38
39     public function testSupportsUnsigned(): void
40     {
```

```
40     $column = new IntColumn('count', type: 'INT');
41     $column = $column->unsigned(true);
42     $sql = $column->getSqlDefinition();
43
44     $this->assertStringContainsString('UNSIGNED', $sql
45         );
46 }
47
48 public function testSupportsAutoIncrement(): void
49 {
50     $column = new IntColumn('id', type: 'INT');
51     $column = $column->autoIncrement(true);
52     $sql = $column->getSqlDefinition();
53
54     $this->assertStringContainsString('AUTO_INCREMENT'
55         , $sql);
56 }
57
58 public function testSupportsNullable(): void
59 {
60     $column = new IntColumn('optional', type: 'INT');
61     $column = $column->nullable(true);
62     $sql = $column->getSqlDefinition();
63
64     $this->assertStringContainsString('NULL', $sql);
65 }
66
67 public function testSupportsNotNullable(): void
68 {
69     $column = new IntColumn('required', type: 'INT');
70     $column = $column->nullable(false);
71     $sql = $column->getSqlDefinition();
72
73     $this->assertStringContainsString('NOT NULL', $sql
74         );
75 }
76
77 public function testSupportsDefault(): void
78 {
79     $column = new IntColumn('status', type: 'INT');
80     $column = $column->default(0);
```

```
78         $sql = $column->getSqlDefinition();
79
80         $this->assertStringContainsString('DEFAULT', $sql)
81         ;
82         $this->assertStringContainsString('0', $sql);
83     }
84 }
```

Listing 10.23: IntColumnTest

10.10.3 Testy TableDefinition

```
1  <?php
2
3  declare(strict_types=1);
4
5  namespace Larafony\Framework\Tests\Database\MySQL;
6
7  use Larafony\Framework\Database\Drivers\MySQL\Schema\
8      ColumnDefinitions\DateTimeColumn;
9  use Larafony\Framework\Database\Drivers\MySQL\Schema\
10     ColumnDefinitions\IntColumn;
11 use Larafony\Framework\Database\Drivers\MySQL\Schema\
12     ColumnDefinitions\StringColumn;
13 use Larafony\Framework\Database\Drivers\MySQL\Schema\
14     ColumnDefinitions\TextColumn;
15 use Larafony\Framework\Database\Drivers\MySQL\Schema\
16     IndexDefinitions\NormalIndex;
17 use Larafony\Framework\Database\Drivers\MySQL\Schema\
18     IndexDefinitions\PrimaryIndex;
19 use Larafony\Framework\Database\Drivers\MySQL\Schema\
20     IndexDefinitions\UniqueIndex;
21 use Larafony\Framework\Database\Drivers\MySQL\Schema\
22     TableDefinition;
23 use Larafony\Framework\Tests\TestCase;
24
25 class TableDefinitionTest extends TestCase
```

```
18 {
19     public function testCreatesTableDefinition(): void
20     {
21         $table = new TableDefinition('users');
22
23         $this->assertEquals('users', $table->tableName);
24         $this->assertEmpty($table->columns);
25     }
26
27     public function testIntegerReturnsIntColumn(): void
28     {
29         $table = new TableDefinition('users');
30
31         $column = $table->integer('age');
32
33         $this->assertInstanceOf(IntColumn::class, $column)
34             ;
35         $this->assertEquals('age', $column->name);
36         $this->assertEquals('INT', $column->type);
37     }
38
39     public function
40     testBigIntegerReturnsIntColumnWithLength(): void
41     {
42         $table = new TableDefinition('users');
43
44         $column = $table->bigInteger('id');
45
46         $this->assertInstanceOf(IntColumn::class, $column)
47             ;
48         $this->assertEquals('id', $column->name);
49     }
50
51     public function
52     testSmallIntegerReturnsIntColumnWithLength(): void
53     {
54         $table = new TableDefinition('users');
55
56         $column = $table->smallInteger('count');
```

```
54         $this->assertInstanceOf(IntColumn::class, $column)
55         ;
56     }
57
58     public function testStringReturnsStringColumn(): void
59     {
60         $table = new TableDefinition('users');
61
62         $column = $table->string('name', 100);
63
64         $this->assertInstanceOf(StringColumn::class,
65             $column);
66         $this->assertEquals('name', $column->name);
67         $this->assertEquals('VARCHAR', $column->type);
68     }
69
70     public function
71     testCharReturnsStringColumnWithCharType(): void
72     {
73         $table = new TableDefinition('users');
74
75         $column = $table->char('code', 5);
76
77         $this->assertInstanceOf(StringColumn::class,
78             $column);
79         $this->assertEquals('code', $column->name);
80         $this->assertEquals('CHAR', $column->type);
81     }
82
83     public function testTextReturnsTextColumn(): void
84     {
85         $table = new TableDefinition('posts');
86
87         $column = $table->text('content');
88
89         $this->assertInstanceOf(TextColumn::class, $column
90             );
91         $this->assertEquals('content', $column->name);
92         $this->assertEquals('TEXT', $column->type);
93     }
```

```
90
91     public function testMediumTextReturnsTextColumn():
          void
92     {
93         $table = new TableDefinition('posts');
94
95         $column = $table->mediumText('description');
96
97         $this->assertInstanceOf(TextColumn::class, $column
          );
98         $this->assertEquals('description', $column->name);
99         $this->assertEquals('MEDIUMTEXT', $column->type);
100     }
101
102     public function testLongTextReturnsTextColumn(): void
103     {
104         $table = new TableDefinition('posts');
105
106         $column = $table->longText('document');
107
108         $this->assertInstanceOf(TextColumn::class, $column
          );
109         $this->assertEquals('document', $column->name);
110         $this->assertEquals('LONGTEXT', $column->type);
111     }
112
113     public function testJsonReturnsTextColumn(): void
114     {
115         $table = new TableDefinition('settings');
116
117         $column = $table->json('data');
118
119         $this->assertInstanceOf(TextColumn::class, $column
          );
120         $this->assertEquals('data', $column->name);
121         $this->assertEquals('JSON', $column->type);
122     }
123
124     public function testDateTimeReturnsDateTimeColumn():
          void
125     {
```

```
126     $table = new TableDefinition('logs');
127
128     $column = $table->dateTime('created_at');
129
130     $this->assertInstanceOf(DateTimeColumn::class,
131         $column);
132     $this->assertEquals('created_at', $column->name);
133     $this->assertEquals('DATETIME', $column->type);
134 }
135
136 public function testTimestampReturnsDateTimeColumn():
137     void
138 {
139     $table = new TableDefinition('logs');
140
141     $column = $table->timestamp('updated_at');
142
143     $this->assertInstanceOf(DateTimeColumn::class,
144         $column);
145     $this->assertEquals('updated_at', $column->name);
146     $this->assertEquals('TIMESTAMP', $column->type);
147 }
148
149 public function testTimeReturnsDateTimeColumn(): void
150 {
151     $table = new TableDefinition('schedules');
152
153     $column = $table->time('start_time');
154
155     $this->assertInstanceOf(DateTimeColumn::class,
156         $column);
157     $this->assertEquals('start_time', $column->name);
158     $this->assertEquals('TIME', $column->type);
159 }
160
161 public function testDateReturnsDateTimeColumn(): void
162 {
163     $table = new TableDefinition('events');
164
165     $column = $table->date('event_date');
```

```
163         $this->assertInstanceOf(DateTimeColumn::class,  
164             $column);  
165         $this->assertEquals('event_date', $column->name);  
166         $this->assertEquals('DATE', $column->type);  
167     }  
168     public function testIndexReturnsNormalIndex(): void  
169     {  
170         $table = new TableDefinition('users');  
171  
172         $index = $table->index('email');  
173  
174         $this->assertInstanceOf(NormalIndex::class, $index  
175             );  
176         $this->assertEquals('users', $index->tableName);  
177         $this->assertEquals('email', $index->columns);  
178     }  
179     public function testIndexWithCustomName(): void  
180     {  
181         $table = new TableDefinition('users');  
182  
183         $index = $table->index('email', 'idx_users_email')  
184             ;  
185  
186         $this->assertInstanceOf(NormalIndex::class, $index  
187             );  
188         $this->assertEquals('idx_users_email', $index->  
189             indexName);  
190     }  
191     public function testPrimaryReturnsPrimaryIndex(): void  
192     {  
193         $table = new TableDefinition('users');  
194  
195         $index = $table->primary('id');  
196  
197         $this->assertInstanceOf(PrimaryIndex::class,  
198             $index);  
199         $this->assertEquals('users', $index->tableName);  
200         $this->assertEquals('id', $index->columns);
```

```
198     }
199
200     public function testPrimaryWithCompositeKey(): void
201     {
202         $table = new TableDefinition('user_roles');
203
204         $index = $table->primary(['user_id', 'role_id']);
205
206         $this->assertInstanceOf(PrimaryKey::class,
207             $index);
208     }
209
210     public function testUniqueReturnsUniqueIndex(): void
211     {
212         $table = new TableDefinition('users');
213
214         $index = $table->unique('email');
215
216         $this->assertInstanceOf(UniqueIndex::class, $index
217             );
218         $this->assertEquals('users', $index->tableName);
219         $this->assertEquals('email', $index->columns);
220     }
221
222     public function testUniqueWithCustomName(): void
223     {
224         $table = new TableDefinition('users');
225
226         $index = $table->unique('email', '
227             unique_user_email');
228
229         $this->assertInstanceOf(UniqueIndex::class, $index
230             );
231         $this->assertEquals('unique_user_email', $index->
232             indexName);
233     }
234
235     public function testSupportsMethodChaining(): void
236     {
237         $table = new TableDefinition('users');
```

```
234         $column = $table->string('email', 255);
235
236         // Verify method chaining is supported on returned
           columns
237         $this->assertInstanceOf(StringColumn::class,
           $column);
238     }
239 }
```

Listing 10.24: TableDefinitionTest

Nota autora

Testy kolumn sprawdzają wszystkie kombinacje opcji: nullable/not null, default values, auto_increment, unsigned. To ważne, bo błąd w generowaniu SQL może być trudny do wykrycia – baza danych nie zawsze rzuci błędem, czasem po prostu zinterpretuje zapytanie inaczej niż zamierzaliśmy.

10.11 Podsumowanie

W tym rozdziale zbudowaliśmy fundament DBAL – warstwy abstrakcji nad bazą danych:

- Interfejs `ConnectionContract` definiujący połączenie z bazą
- Wzorzec `Grammar` do kompilacji obiektowych definicji do SQL
- `TableDefinition` z fluent API do definiowania tabel
- Klasy kolumn: `IntColumn`, `StringColumn`, `DateTimeColumn`, `EnumColumn`
- Klasy indeksów: `PrimaryIndex`, `UniqueIndex`, `NormalIndex`
- `SchemaBuilder` jako główny orkiestrator operacji na schemacie
- Buildery SQL: `CreateTableBuilder`, `AddColumns`, `ChangeColumns`, `DropColumns`

Nota autora

Architektura oparta na **Grammar** może wydawać się bardziej skomplikowana niż proste generowanie SQL w jednej klasie. Ale ta inwestycja zwraca się wielokrotnie:

- Dodanie wsparcia dla PostgreSQL to nowy katalog w **Drivers/**, nie modyfikacja istniejącego kodu
- Testy są prostsze – możemy testować **Grammar** niezależnie od połączenia z bazą
- Fluent API jest czytelne dla programistów znających Laravela

W kolejnym rozdziale zajmiemy się **QueryBuilder** – klasą do budowania zapytań SELECT, INSERT, UPDATE i DELETE.

Czas na zapytania!

Spis listingów

1.1	Zasady SOLID	58
1.2	Przykład wzorca Budowniczy	62
1.3	Bazowy dekorator PSR-7	66
1.4	Dekorator MoveStatusDecorator	67
1.5	klasa UploadedFileFactory	68
1.6	klasa StorageFactory	69
1.7	Fasada Log	72
2.1	Interfejs ErrorHandler	96
2.2	Klasa DetailedErrorHandler	97
2.3	Klasa HtmlErrorFormatter	100
2.4	Aplikacja demonstracyjna z trasami błędów	103
2.5	Testy klasy DetailedErrorHandler	105
2.6	Testy klasy HtmlErrorFormatter	108
3.1	Interfejs Clock	114
3.2	Enum TimeFormat	116
3.3	Enum Timezone	117

3.4	Klasa SystemClock	118
3.5	Klasa FrozenClock	123
3.6	Klasa ClockFactory	128
3.7	Aplikacja demonstracyjna z zegarem	132
3.8	Testy klasy SystemClock	133
3.9	Testy klasy FrozenClock	136
3.10	Testy klasy ClockFactory	140
4.1	Request i response przy użyciu tablic superglobalnych	145
4.2	Klasa UriManager	147
4.3	Klasa Scheme	150
4.4	Klasa Authority	150
4.5	Klasa Query	151
4.6	Klasa UriFactory	152
4.7	Klasa StreamWrapper	154
4.8	Klasa StreamMetaData	155
4.9	Klasa StreamContentManager	157
4.10	Klasa StreamSize	158
4.11	Klasa Stream	159
4.12	Klasa InactivePropertyGuard	163
4.13	Klasa StreamFactory	164
4.14	Klasa FileMoveHandler	166
4.15	Klasa StreamMoveHandler	167

4.16	Klasa UploadedFileDecorator	169
4.17	Klasa ErrorValidatorDecorator	170
4.18	Klasa MoveStatusDecorator	171
4.19	Klasa PathValidatorDecorator	172
4.20	Klasa UploadedFile	173
4.21	Klasa UploadedFileFactory	175
4.22	Klasa Message	177
4.23	Klasa HeaderManager	180
4.24	Klasa Request	183
4.25	Klasa HeadersParser	185
4.26	Klasa QueryParamsManager	186
4.27	Klasa ParsedBodyParser	187
4.28	Klasa ParsedBodyManager	188
4.29	Klasa AttributesManager	189
4.30	Klasa UploadedFilesManager	190
4.31	Klasa UploadedFilesParser	191
4.32	Klasa ServerRequest	193
4.33	Enum StatusCode	198
4.34	Klasa ReasonPhraseMapper	200
4.35	Klasa StatusCodeFactory	201
4.36	Klasa Response	202
4.37	Klasa RequestFactory	203

4.38	Klasa <code>ServerRequestFactory</code>	205
4.39	Klasa <code>ResponseFactory</code>	207
5.1	Interfejs <code>ContainerContract</code>	212
5.2	Klasa <code>ContainerError</code>	213
5.3	Klasa <code>NotFoundError</code>	214
5.4	Interfejs <code>ReflectionResolverContract</code>	215
5.5	Klasa <code>ReflectionResolver</code>	217
5.6	Interfejs <code>AutowireContract</code>	222
5.7	Klasa <code>Autowire</code>	222
5.8	Interfejs <code>ArrayContract</code>	227
5.9	Klasa <code>ArrayGet</code>	227
5.10	Klasa <code>ArraySet</code>	228
5.11	Klasa <code>DotContainer</code>	229
5.12	Klasa <code>Container</code>	232
5.13	Interfejs <code>ServiceProviderContract</code>	235
5.14	Klasa <code>ServiceProvider</code>	236
5.15	Testy klasy <code>ReflectionResolver</code>	239
5.16	Testy klasy <code>DotContainer</code>	242
5.17	Testy klasy <code>Container</code>	243
6.1	Enum <code>HttpMethod</code>	249
6.2	Klasa <code>Route</code>	250
6.3	Klasa <code>RouteMatcher</code>	251

6.4	Klasa RouteCollection	254
6.5	Klasa RouteNotFoundError	256
6.6	Klasa ClosureRouteHandler	258
6.7	Klasa ClassMethodRouteHandler	258
6.8	Klasa InvokableClassRouteHandler	260
6.9	Klasa RouteHandlerFactory	262
6.10	Klasa ArrayHandlerFactory	264
6.11	Klasa StringHandlerFactory	265
6.12	Klasa Router	267
6.13	Klasa RouteServiceProvider	270
6.14	Plik public/index.php	271
6.15	Plik bootstrap/app.php	272
6.16	Testy klasy RouteMatcher	273
6.17	Testy klasy Route	278
6.18	Testy klasy Router	280
7.1	Klasa HttpClientConfig	288
7.2	Klasa CurlHttpClient	292
7.3	Klasa CurlOptionsBuilder	294
7.4	Klasa CurlHandleExecutor	298
7.5	Interfejs CurlWrapperContract	303
7.6	Klasa CurlWrapper	305
7.7	Klasa FakeCurlWrapper	306

7.8	Klasa ResponseParser	309
7.9	Klasa ResponseHeadersParser	311
7.10	Klasa StatusLineParser	312
7.11	Klasa NetworkError	314
7.12	Klasa TimeoutError	317
7.13	Interfejs MockHandler	318
7.14	Klasa CallbackMockHandler	319
7.15	Klasa MockHttpClient	321
7.16	Klasa HttpClientFactory	324
7.17	Testy klasy HttpClientFactory	327
8.1	enum LineType	337
8.2	klasa EnvironmentVariable	338
8.3	klasa ParsedLine	338
8.4	klasa ParserResult	339
8.5	klasa ValueParser	341
8.6	klasa LineParser	344
8.7	klasa DotenvParser	346
8.8	klasa EnvironmentLoader	347
8.9	klasa EnvReader	350
8.10	klasa ConfigBase	351
8.11	klasa ConfigServiceProvider	353
8.12	plik config/app.php	355

8.13	test DotenvParser	355
8.14	test EnvironmentLoader	360
9.1	enum ForegroundColor	367
9.2	enum BackgroundColor	368
9.3	enum Style	369
9.4	klasa OutputFormatterStyle	369
9.5	klasa InfoStyle	370
9.6	klasa OutputFormatter	371
9.7	klasa Output	373
9.8	atrybut AsCommand	380
9.9	atrybut CommandArgument	380
9.10	atrybut CommandOption	382
9.11	klasa Input	384
9.12	klasa InputParser	385
9.13	klasa ValueCaster	386
9.14	klasa Command	387
9.15	klasa CommandResolver	388
9.16	klasa ArgumentResolver	389
9.17	klasa OptionResolver	390
9.18	klasa CommandRegistry	392
9.19	klasa CommandDiscovery	394
9.20	klasa Kernel	397

9.21	klasa Console/Application	399
9.22	klasa ConsoleServiceProvider	401
9.23	plik bin/larafony	403
9.24	plik bootstrap/console.php	404
9.25	test klasy Output	405
9.26	test aplikacji konsolowej	409
9.27	przykładowa komenda BuildNotesCommand	410
10.1	Migracja tabeli users	414
10.2	Model User	416
10.3	Seeder UserSeeder	418
10.4	Interfejs ConnectionContract	422
10.5	Interfejs GrammarContract	424
10.6	Klasa MySQL Grammar	425
10.7	Klasa MySQL TableDefinition	427
10.8	Klasa BaseColumn	433
10.9	Klasa IntColumn	434
10.10	Klasa IntColumn	436
10.11	Klasa StringColumn	438
10.12	Klasa DateTimeColumn	439
10.13	Klasa EnumColumn	441
10.14	Klasa IndexDefinition	443
10.15	Klasa PrimaryIndex	445

10.16	Klasa SchemaBuilder (bazowa)	446
10.17	Klasa MySQL SchemaBuilder	447
10.18	Klasa CreateTableBuilder	451
10.19	Klasa AddColumns	452
10.20	Klasa ChangeColumns	454
10.21	Klasa DropColumns	455
10.22	GrammarTest	456
10.23	IntColumnTest	461
10.24	TableDefinitionTest	463
11.1	Klasa QueryDefinition	475
11.2	Enum QueryType	477
11.3	Enum JoinType	479
11.4	Enum OrderDirection	479
11.5	Klasa WhereClause (bazowa)	480
11.6	Klasa WhereBasic	481
11.7	Klasa WhereIn	482
11.8	Klasa WhereNull	483
11.9	Klasa WhereBetween	484
11.10	Klasa WhereLike	486
11.11	Klasa WhereNested	487
11.12	Klasa JoinClause	489
11.13	Klasa OrderByClause	490

<i>SPIS LISTINGÓW</i>	1526
11.14Klasa LimitClause	491
11.15Interfejs Query GrammarContract	492
11.16Klasa MySQL Query Grammar	493
11.17Klasa SelectBuilder	497
11.18Klasa InsertBuilder	499
11.19Klasa UpdateBuilder	499
11.20Klasa DeleteBuilder	501
11.21Klasa WhereBuilder	502
11.22Klasa JoinBuilder	503
11.23Klasa OrderByBuilder	504
11.24Klasa LimitBuilder	505
11.25Klasa QueryBuilder (bazowa)	506
11.26Klasa MySQL QueryBuilder	510
11.27QueryBuilderTest (fragment)	522
12.1 Interfejs MigrationContract	531
12.2 Klasa Migration	532
12.3 Klasa MigrationResolver	533
12.4 Klasa MigrationRepository (bazowa)	535
12.5 Klasa MySQL MigrationRepository	538
12.6 Klasa MigrationExecutor	539
12.7 Klasa MakeCommand	541
12.8 Stub migracji	544

12.9 Komenda MakeMigration	544
12.10Komenda Migrate	547
12.11Komenda MigrateRollback	549
12.12Komenda MigrateFresh	552
12.13Przykładowa migracja	554
12.14Konfiguracja bazy danych	556
12.15MakeMigrationTest	558
13.1 Fasada DB	570
13.2 Klasa PropertyObserver	572
13.3 Klasa Model	574
13.4 Klasa EntityManager	579
13.5 Klasa EntityInserter	580
13.6 Klasa EntityUpdater	581
13.7 Klasa ModelQueryBuilder	582
13.8 Klasa Relation	589
13.9 Atrybut HasMany	590
13.10Atrybut BelongsTo	591
13.11Klasa HasMany	591
13.12Klasa BelongsToMany	593
13.13Klasa BelongsToMany	594
13.14Klasa RelationDecorator	597
13.15Klasa EagerRelationsLoader	602

13.16Klasa BaseRelationLoader	605
13.17Klasa HasManyLoader	608
13.18Klasa BelongsToLoader	611
13.19Model User	613
13.20Atrybut CastUsing	618
13.21Klasa AttributeCaster	618
13.22Komenda MakeModel	620
13.23Stub modelu	623
13.24Klasa Seeder	624
13.25Komenda DatabaseSeed	625
13.26Klasa DatabaseManager	628
13.27Fasada Schema	633
13.28DatabaseServiceProvider	635
13.29ModelTest (fragment)	637
14.1 Klasa PlaceholderProcessor	647
14.2 Klasa Context	648
14.3 Klasa Message	649
14.4 Klasa Metadata	650
14.5 Klasa Logger	651
14.6 JsonFormatter	653
14.7 TextFormatter	655
14.8 XmlFormatter	656

14.9 Komenda DatabaseLogTable	658
14.10Stub migracji database_logs	659
14.11Model DatabaseLog	661
14.12CastUsing z castBack	662
14.13AttributeCaster z castBack	663
14.14Klasa ArrayCaster	665
14.15DatabaseHandler	667
14.16Klasa DailyRotator	668
14.17Klasa FileHandler	671
14.18Klasa LoggerFactory	673
14.19Konfiguracja logowania	674
14.20LogServiceProvider	675
14.21Fasada Log	676
14.22TextFormatterTest	680
14.23LogTest	685
15.1 Klasa MiddlewareStack	692
15.2 Klasa RouterMiddleware	694
15.3 Klasa Kernel	696
15.4 Klasa RouteParameter	698
15.5 Klasa ParsedRouteDecorator	699
15.6 Klasa Route	700
15.7 Klasa RouteMatcher	704

15.8 Klasa ParameterResolver	707
15.9 Klasa FunctionRouteHandler	710
15.10Klasa ClassMethodRouteHandler	711
15.11Klasa InvocableClassRouteHandler	713
15.12Klasa RouteBinding	716
15.13Klasa ModelBinder	716
15.14Klasa Router	720
15.15Klasa RouteGroup	723
15.16Klasa RouteMiddleware	726
15.17Atrybut Route	728
15.18Atrybut RouteGroup	729
15.19Atrybut RouteParam	729
15.20Atrybut Middleware	730
15.21Klasa AttributeRouteScanner	731
15.22Klasa AttributeRouteLoader	733
15.23Klasa ClassAttributesProcessor	735
15.24Klasa ParamAttributeProcessor	736
15.25Klasa RouteBuilder	738
15.26Test RouteMatcher	742
15.27Test Middleware	748
15.28Test Kernel	751
16.1 Interfejs ValidationRule	766

16.2 Klasa ValidationAttribute	767
16.3 Atrybut IsValidated	768
16.4 Atrybut Required	768
16.5 Atrybut MaxLength	769
16.6 Atrybut MinLength	770
16.7 Atrybut Min	771
16.8 Atrybut Email	772
16.9 Atrybut Confirmed	772
16.10Atrybut RequiredWhen	774
16.11Klasa ValidationResult	777
16.12Klasa ValidationError	778
16.13Klasa ValidationFailed	778
16.14Klasa AttributeValidator	780
16.15Klasa FormRequest	783
16.16Interfejs PreResolutionHookContract	786
16.17Klasa FormRequestTypeDetector	787
16.18Klasa FormRequestFactory	788
16.19Klasa FormRequestAwareHandler	793
16.20Klasa ArrayHandlerFactory	796
16.21Test FormRequest	799
16.22Test ValidationResult	804
16.23Test FormRequestAwareHandler	806

17.1 Plik konfiguracyjny views.php	815
17.2 Interfejs RendererContract	817
17.3 Interfejs ViewContract	818
17.4 Interfejs DirectiveContract	818
17.5 Klasa View	819
17.6 Klasa ViewManager	821
17.7 Klasa TemplateLoader	822
17.8 Klasa TemplateCompiler	824
17.9 Klasa bazowa Directive	828
17.10Dyrektywa If	831
17.11Dyrektywa Foreach	833
17.12Dyrektywa For	834
17.13Dyrektywa While	834
17.14Dyrektywa DoWhile	835
17.15Dyrektywa Unless	836
17.16Dyrektywa Switch	837
17.17Dyrektywa Isset	839
17.18Dyrektywa Empty	840
17.19Dyrektywa Extend	841
17.20Dyrektywa Section	842
17.21Dyrektywa Yield	843
17.22Interfejs ComponentContract	844

17.23	Klasa Component	845
17.24	Dyrektywa Component	848
17.25	Dyrektywa Slot	853
17.26	Klasa AssetManager	854
17.27	Dyrektywa Stack	855
17.28	Klasa BladeAdapter	857
17.29	Klasa BaseAdapter	863
17.30	Klasa BladeSourceMapper	865
17.31	Klasa ViewServiceProvider	868
17.32	Test View	871
17.33	Test ViewManager	876
19.1	Przykład użycia options API	901
19.2	Przykład użycia compositions API	903
19.3	Przykład użycia compositions API ze składnią script setup	904
19.4	Implementacja klasy ResponseFactory	905
19.5	Implementacja klasy Inertia	908
19.6	Implementacja klasy Vite	913
19.7	Implementacja dyrektywy ViteDirective	917
19.8	Middleware InertiaMiddleware	919
19.9	Plik package.json	923
19.10	Plik vite.config.js	924
19.11	Główny widok inertia.blade.php	926

19.12	Plik app.js	927
19.13	Komponent Home/Index.vue	928
20.1	Interfejs ErrorHandler	935
20.2	Klasa CodeSnippet	936
20.3	Klasa TraceFrame	938
20.4	Klasa TraceCollection	940
20.5	Klasa Backtrace	941
20.6	Klasa DetailedErrorHandler	942
20.7	Klasa FallbackErrorHandler	947
20.8	Widok debug.blade.php	948
20.9	Widok 404.blade.php	957
20.10	Widok 500.blade.php	964
20.11	Klasa HtmlErrorFormatter	970
20.12	Klasa ErrorHandlerServiceProvider	973
20.13	Test DetailedErrorHandler	976
20.14	Test FallbackErrorHandler	980
21.1	Klasa PathNormalizer	986
21.2	Klasa TraceFrameFetcher	986
21.3	Klasa VariableFormatter	987
21.4	Klasa BaseFrameRenderer	989
21.5	Klasa FrameDetailsRenderer	990
21.6	Klasa FrameSourceRenderer	992

21.7 Klasa FrameVariablesRenderer	992
21.8 Klasa CodeSnippetRenderer	993
21.9 Klasa ConsoleHeaderRenderer	995
21.10Klasa ConsoleTraceRenderer	996
21.11Klasa ConsoleEnvironmentRenderer	998
21.12Klasa ConsoleHelpRenderer	999
21.13Klasa ConsoleCommandProcessor	1000
21.14Klasa ConsoleFrameRenderer	1002
21.15Klasa DebugSession	1004
21.16Klasa ConsoleRendererFactory	1005
21.17Klasa ConsoleRenderer	1009
21.18Klasa ConsoleHandler	1011
21.19Test ConsoleRenderer	1015
22.1 Szyfrowanie przy użyciu OpenSSL	1026
22.2 Użycie NONCE	1027
22.3 Sodium secretbox	1028
22.4 Sodium AEAD	1029
22.5 Generator kluczy	1030
22.6 Modyfikator pliku .env	1031
22.7 Komenda do generowania klucza	1032
22.8 Asercja EncryptionKeyExists	1033
22.9 Asercja KeyLengthIsValid	1033

22.10	Asercja Base64IsValid	1034
22.11	Asercja DataLengthIsValid	1035
22.12	Asercja DecryptionSucceeded	1036
22.13	Klasa EncryptionService	1037
22.14	Funkcja setcookie	1039
22.15	Klasa CookieOptions	1040
22.16	Klasa CookieManager	1042
22.17	Klasa SessionConfiguration	1044
22.18	Klasa SessionSecurity	1045
22.19	Klasa SessionManager	1046
22.20	Klasa FileSessionHandler	1051
22.21	Komenda SessionTable	1053
22.22	Stub migracji sesji	1055
22.23	Encja Session	1056
22.24	Klasa DatabaseSessionHandler	1058
22.25	Klasa SessionServiceProvider	1062
22.26	Testy szyfrowania	1064
22.27	Testy handlera plików	1067
22.28	Testy SessionManager	1070
23.1	Kontrakt TransportContract	1078
23.2	Kontrakt MailerContract	1079
23.3	Kontrakt MailHistoryLoggerContract	1080

23.4 Wyjątek TransportError	1081
23.5 Klasa Address	1082
23.6 Klasa Content	1083
23.7 Klasa Envelope	1084
23.8 Klasa Email	1086
23.9 Value Object MailUserInfo	1089
23.10 Value Object MailEncryption	1090
23.11 Value Object MailPort	1091
23.12 Klasa SmtplibConfig	1093
23.13 Klasa SmtplibConnection	1094
23.14 Klasa SmtplibCommand	1097
23.15 Klasa SmtplibResponse	1099
23.16 Klasa SmtplibResponseFactory	1101
23.17 Asercja NotConnected	1104
23.18 Klasa SmtplibTransport	1104
23.19 Klasa Mailable	1109
23.20 Klasa Mailer	1112
23.21 Klasa MailerFactory	1114
23.22 Komenda MailLogTable	1116
23.23 Stub migracji mail_log	1117
23.24 Encja MailLog	1119
23.25 Klasa MailHistoryLogger	1121

23.26	Test Address	1123
23.27	Test SmtplibConfig	1125
23.28	Test SmtplibCommand	1127
23.29	Test MailerFactory	1130
23.30	MailServiceProvider	1133
24.1	Encja User	1141
24.2	Encja Role	1147
24.3	Encja Permission	1150
24.4	Klasa UserManager	1151
24.5	Klasa RoleManager	1155
24.6	Klasa PermissionManager	1156
24.7	Fasada Auth	1158
24.8	AuthMiddleware	1162
24.9	AuthServiceProvider	1163
24.10	RegisterDto	1164
24.11	LoginDto	1165
24.12	RegisterController	1166
24.13	LoginController	1168
24.14	LogoutController	1170
25.1	Kontrakt StorageContract	1177
25.2	Klasa BaseStorage	1179
25.3	Klasa FileStorage	1185

25.4 Enum RedisEvictionPolicy	1191
25.5 Klasa RedisStorage	1192
25.6 Klasa CacheItemKeyValidator	1199
25.7 Klasa CacheItem	1200
25.8 Klasa CacheItemPool	1203
25.9 Klasa TaggedCache	1207
25.10 Fasada Cache	1211
25.11 Kontrakt StorageFactoryContract	1217
25.12 Klasa FileStorageFactory	1219
25.13 Klasa RedisStorageFactory	1220
25.14 Klasa MemcachedStorageFactory	1222
25.15 Klasa StorageFactory	1223
25.16 CacheServiceProvider	1225
26.1 Klasa StoppableEvent	1233
26.2 Klasa EventDispatcher	1233
26.3 Klasa ListenerProvider	1234
26.4 Atrybut Listen	1237
26.5 Klasa ListenerDiscovery	1238
26.6 EventServiceProvider	1241
26.7 Event UserRegistered	1245
26.8 Klasa WelcomeEmail	1246
26.9 Widok welcome email	1247

26.10	Klasa SendWelcomeEmailListener	1250
27.1	Kontrakt DataCollectorContract	1259
27.2	Klasa DebugBar	1259
27.3	DTO QueryInfoDto	1261
27.4	Klasa QueryCollector	1262
27.5	Klasa CacheCollector	1264
27.6	Klasa PerformanceCollector	1267
27.7	Klasa RequestCollector	1268
27.8	Klasa RouteCollector	1270
27.9	Klasa ViewCollector	1271
27.10	Klasa TimelineCollector	1273
27.11	Middleware InjectDebugBar	1281
27.12	DebugBarServiceProvider	1283
28.1	Kontrakt JobContract	1291
28.2	Kontrakt QueueContract	1291
28.3	Atrybut Serialize	1292
28.4	Klasa Job	1293
28.5	Klasa DatabaseQueue	1297
28.6	Klasa RedisQueue	1300
28.7	Klasa QueueWorker	1303
28.8	Klasa QueueFactory	1305
28.9	Komenda QueueWork	1306

28.10	Klasa FailedJobRepository	1308
28.11	Enum CronSchedule	1313
28.12	Klasa CronExpression	1314
28.13	Klasa ScheduledEvent	1317
28.14	Klasa Schedule	1319
28.15	Klasa ScheduleWorker	1321
28.16	Komenda ScheduleRun	1322
28.17	Klasa Dispatcher	1324
28.18	SchedulerServiceProvider	1326
29.1	Klasa Frame	1333
29.2	Klasa Encoder	1335
29.3	BaseFrameHead	1337
29.4	TinyFrameHead (payload do 125 bajtów)	1338
29.5	MediumFrameHead (126-65535 bajtów)	1339
29.6	LargeFrameHead (powyżej 65535 bajtów)	1340
29.7	Klasa Decoder	1341
29.8	AssertMinimumFrameLength	1343
29.9	AssertUnpackSuccess	1344
29.10	AssertSufficientBytes	1345
29.11	Klasa Handshake	1347
29.12	FiberEngine	1352
29.13	Klasa Server	1360

29.14	Klasa Connection	1367
29.15	EngineContract	1369
29.16	ConnectionContract	1370
29.17	ServerContract	1370
29.18	WebSocketServiceProvider	1371
29.19	WebSocketStart	1373
29.20	ReactEngine	1375
29.21	ReactConnection adapter	1380
29.22	ReactWebSocketServiceProvider	1382
29.23	ChatAIController	1384
29.24	ChatMessageListener	1385
29.25	ChatWebSocketProvider	1388
29.26	Chat/Index.vue	1390
30.1	McpServerFactory	1402
30.2	CacheSessionStore	1404
30.3	McpServiceProvider	1407
30.4	McpController	1410
30.5	Konfiguracja MCP	1412
30.6	Bootstrap z MCP	1413
30.7	McpStartCommand	1415
30.8	MakeControllerTool	1420
30.9	FrameworkDocsResource (fragment)	1422

30.10LearnLarafonyPrompt (fragment)	1432
31.1 DI - framework	1441
31.2 DI - CodeIgniter	1442
31.3 DI - Laravel 12	1443
31.4 DI - Symfony 8	1444
31.5 routing - framework	1445
31.6 routing - CI4	1446
31.7 routing - Laravel 12	1447
31.8 routing - Symfony 8	1448
31.9 DBAL - framework	1449
31.10DBAL - CI4	1450
31.11DBAL - Laravel 12	1451
31.12DBAL - Symfony 8	1452
31.13Console - framework	1453
31.14Console - CI4	1454
31.15Console - Laravel 12	1455
31.16Console - Symfony 8	1456
31.17View - framework	1457
31.18View - CI4	1458
31.19View - Laravel 12	1459
31.20View - Symfony 8	1460
31.21Mail - framework	1463

31.22Mail - CI4	1464
31.23Mail - Laravel 12	1464
31.24Mail - Symfony 8	1465
31.25Events - framework	1468
31.26Events - CI4	1469
31.27Events - Laravel 12	1470
31.28Events - Symfony 8	1471
31.29Jobs - framework	1472
31.30Jobs - CI4	1473
31.31Jobs - Laravel 12	1474
31.32Jobs - Symfony 8	1475
31.33Reverb client-to-server (produkcja)	1477
32.1 MonologLogger – wrapper Monologa	1482
32.2 MonologLoggerFactory – tworzenie loggerów z konfiguracji . . .	1483
32.3 MonologServiceProvider	1485
32.4 Konfiguracja Monologa	1486
32.5 Metody pomocnicze MonologLoggerFactory	1487
32.6 SymfonyTransport – adapter transportu	1488
32.7 SymfonyMailerServiceProvider	1490
32.8 SymfonyTransportFactory	1491
32.9 CarbonClock – implementacja interfejsu Clock	1492
32.10Rozszerzone możliwości CarbonClock	1494

32.11CarbonServiceProvider	1495
32.12GuzzleHttpClient – adapter PSR-18	1496
32.13GuzzleHttpClientFactory	1498
32.14Retry middleware	1499
32.15GuzzleServiceProvider	1500
32.16TwigAdapter	1501
32.17Szablon Twig	1503
32.18SmartyAdapter	1504
32.19TwigServiceProvider	1505
32.20PhpDebugBar – główny adapter	1507
32.21LarafonyCollectorAdapter	1509
32.22QueryCollector dla PHP Debug Bar	1510
32.23InjectPhpDebugBar middleware	1512
32.24PhpDebugBarServiceProvider	1513

Skorowidz

Skorowidz

AI integration, 1400
AI w pentestach, 762
ANSI escape code, 367
Apache, 85
Architektura WebSockets, 1332
array_find, 255
ArrayCaster, 665
ArrayHandlerFactory, 796
Asserts, 1343
Asymetryczna widoczność, 256
Atrybuty komendy, 380
Atrybuty kontrolera, 728
Atrybuty requestu, 189
Atrybuty walidacji, 766
AttributeCaster, 617, 663
autoloading, 76
Automatyczne bindowanie modeli, 715
Autoryzacja, 1140
Autowire, 222

Baza danych, 413
Bazowa komenda, 387
Blade, 813
BladeSourceMapper, 865
Bridge, 1481
Budowa URI, 150
Builder Pattern, 450
Burp Community, 761

Cache, 1175
Cache - FileStorage, 1185
Cache - RedisStorage, 1192

Cache - wprowadzenie, 1175
Cache widoków, 864
CacheSessionStore, 1404
Carbon, 123, 1492
castBack, 662
CastUsing, 617, 662
Ciasteczka - handler, 1039
Ciasteczka - jak to działa, 1025
Ciastka, 1024
Claude Code, 1417
ClientInterface, 287
Clock, 80, 112, 1492
code style, 78
Composer, 80, 881
Config, 335
config/database, 556
ConfigBase, 351
ConnectionContract, 422
Console Error Handler, 985
Container Interface, 211
Controller, 273
Cookies, 1024
Cron, 1311
Cron - parser wyrażeń, 1314
Crontab, 1311
CURL, 286
CurlHandleExecutor, 298
CurlHttpClient, 291
CurlOptionsBuilder, 294
CurlWrapperContract, 303

Database Abstraction Layer, 413

- DatabaseHandler, 658
- DatabaseManager, 628
- DatabaseServiceProvider, 628
- DBAL, 413, 1448
- Debug Bar, 1506
- DebugBar, 1258
- Debugowanie, 1506
- Dependency Injection, 210
- dependency injection, 77
- DetailedErrorHandler, 942
- DI, 210
- Dot Notation, 226
- Double Testing, 303
- DRY, 57, 198
- DTO, 764
- Eager Loading, 601
- EntityManager, 579
- Enum, 367
- Error Handler, 95, 934
- ErrorException, 99
- event dispatcher, 78
- event-driven architecture, 78
- Fabryka MailerFactory, 1114
- Fabryka ResponseFactory, 905
- Failed Jobs, 1308
- Fasada DB, 570
- Fasada Mailable, 1109
- Fatal Error, 99
- FiberEngine, 1349
- FileHandler, 668
- First-class Callable, 99
- Fluent API, 427, 472, 506
- Formatter logów, 653
- FormRequestAwareHandler, 786
- FrameHead, 1337
- Funkcja setcookie, 1039
- Grammar, 424
- Grupowanie tras, 723
- Guzzle, 1496
- Handlers obsługi wyjątków, 935
- HtmlErrorFormatter, 100
- HTTP, 77
- HTTP - DELETE, 764
- HTTP - GET, 763
- HTTP - PATCH, 764
- HTTP - POST, 763
- HTTP - PUT, 764
- HTTP Client, 286, 1496
- HTTP factory, 79
- HTTP vs WebSockets, 1331
- HttpClientConfig, 288
- HttpClientFactory, 324
- HttpMethod, 249
- Indeksy bazy danych, 443
- INDEX, 443
- Inertia.js, 896, 897
- Inertia.js - generowanie odpowiedzi, 905
- Inertia.js - mechanika, 898
- Inertia.js - protokół, 898
- Inertia.js - struktura odpowiedzi, 898
- Interaktywne menu, 378
- Interface Segregation, 96
- Interfejs SessionHandlerInterface, 1044
- Joby, 1290
- Joby - handler oparty na bazie danych, 1296
- Joby - handler oparty na bazie redis, 1299
- Joby - serializacja, 1292
- JoinType, 477
- Kernel, 696
- Kernel aplikacji konsolowej, 397
- Kolejki, 1290

- Kolumny bazy danych, 432
- Komenda database:seed, 625
- Komenda KeyGenerate, 1032
- Komenda Make, 541
- Komenda make:migration, 544
- Komenda make:model, 620
- Komenda migrate, 547
- Komenda migrate:fresh, 552
- Komenda migrate:rollback, 549
- Komenda schedule:run, 1322
- Komendy do migracji, 541
- Komponenty Blade, 844
- Komunikacja w czasie rzeczywistym, 1331
- kontener zależności, 77
- Kooperatywna wielozadaniowość, 1351
- larafony/mcp-assistant, 1419
- larafony/websocket-react, 1375
- Laravel, 365
- Laravel 12, 1439
- Laravel Boost, 1418
- Laravel Reverb, 1397
- Lazy Objects, 1352
- Logger, 645
- Logger PSR-3, 651
- logging, 76
- logowanie, 76
- Mail, 1487
- Maile, 1076
- Maile - IMAP, 1077
- Maile - Klasa Address, 1082
- Maile - Klasa Content, 1083
- Maile - Klasa Envelope, 1084
- Maile - SMTP, 1077
- MailHog, 1116
- MCP, 1400
 - porównanie frameworków, 1418
- MCP bezpieczeństwo, 1419
- MCP HTTP endpoint, 1410
- MCP implementacja, 1401
- MCP konfiguracja klienta, 1417
- MCP przepływ, 1419
- MCP Session, 1404
- mcp/sdk, 1401
- mcp:start, 1415
- McpServerFactory, 1402
- McpServiceProvider, 1407
- Message, 649
- MessageInterface, 177
- Metadane strumienia, 155
- Middleware, 691
- middleware, 79
- Middleware Inertia.js, 921
- Middleware routera, 694
- Middleware trasy, 726
- Migracje bazy danych, 530
- Migration, 531
- MigrationExecutor, 539
- MigrationRepository, 535
- MigrationResolver, 533
- MockHttpClient, 318
- Model, 573
- Model Context Protocol, 1400, 1479
- ModelQueryBuilder, 582
- Monolog, 1482
- MVC, 76
- Nginx, 87
- Node.js, 923
- NoDiscard, 231
- Notacja kropkowa, 336
- Object-Relational Mapping, 568
- Obsługa błędów, 95
- Obsługa wyjątków, 934, 985
- Opcje komendy, 380
- OpenAI integration, 1384
- OrderDirection, 477
- ORM, 568

- OWASP ZAP, 762
- Packagist, 881
- Parser .env, 336
- Parsowanie argumentów CLI, 383
- PDO, 422
- PHP 8.5, 231
- PHP Debug Bar, 1506
- PHP Fibers, 1349
- PHPstan, 89
- PHPStan - konfiguracja, 91
- PHPUnit, 89
- PHPUnit - konfiguracja, 91
- Pipe Operator, 298
- Pipe operator, 184
- Placeholdery PSR-3, 647
- Plik .env, 1031
- plik .env, 335
- plik bin/console, 403
- Porównanie frameworków - Aplikacje konsolowe, 1453
- Porównanie frameworków - Autoryzacja, 1466
- Porównanie frameworków - Cache, 1467
- Porównanie frameworków - Ciasteczka, 1461
- Porównanie frameworków - DBAL, 1448
- Porównanie frameworków - DI, 1441
- Porównanie frameworków - Eventy, 1468
- Porównanie frameworków - Joby, 1472
- Porównanie frameworków - Kolejki, 1472
- Porównanie frameworków - MCP, 1479
- Porównanie frameworków - middleware, 1445
- Porównanie frameworków - routing, 1445
- Porównanie frameworków - Sesja, 1461
- Porównanie frameworków - Websocket, 1476
- Porównanie frameworków - Widoki, 1457
- Porównanie frameworków - Wyjątki, 1460
- Porównanie frameworków - Wysyłka maili, 1462
- Porównanie frameworków - złożoność kodu, 1440
- Postman, 761
- Poziomy logowania, 646
- PRIMARY KEY, 443
- Problem N+1, 601
- Property Hooks, 258, 572
- PropertyObserver, 572
- Przetwarzanie jobów, 1303
- PSR, 75
- PSR-11, 77, 210, 211
- PSR-11 Exceptions, 213
- PSR-12, 78
- PSR-14, 78, 1231
- PSR-14 - Dispatcher, 1232
- PSR-14 - Event, 1231
- PSR-14 - Listener, 1232
- PSR-14 - Listener Provider, 1232
- PSR-15, 79, 251
- PSR-17, 79, 146
- PSR-18, 286, 287, 1496
- PSR-20, 80, 112, 115, 1492
- PSR-3, 76, 645, 1482
- PSR-4, 76
- PSR-6, 76, 1176
- PSR-7, 77, 146
- Publikacja pakietu, 881
- Query Grammar, 492
- QueryBuilder, 472
- QueryDefinition, 475
- QueryType, 477
- ReactPHP, 1375
- Redis, 1191, 1299
- Redis - instalacja, 1191
- Reflection Class, 214

- ReflectionNamedType, 221
- Refleksja, 214
- Rejestracja komendy, 392
- Relacje ORM, 589
- Request, 77, 145, 177
- Request Factory, 203
- request handling, 79
- RequestHandlerInterface, 251
- Response, 77, 145, 198
- RFC 3986, 146
- RFC 6455, 1332
- Rotacja logów, 668
- Route, 250
- RouteCollection, 253
- RouteMatcher, 251, 704
- RouteNotFoundError, 256
- Router, 267
- Router zaawansowany, 720
- RouteServiceProvider, 269
- Routing, 247
- Rozpoznawanie atrybutów i opcji, 388
- RunInSeparateProcess, 108
- SchemaBuilder, 445
- Seeder, 624
- Serializacja, 1293
- Service Provider, 235
- Sesja, 1024
- Sesja - handler oparty na bazie danych, 1053
- Sesja - handler oparty na plikach, 1050
- Sesja - jak to działa, 1025
- Session hijacking, 1060
- Silnik szablonów, 1501
- Smarty, 1501
- SOLID, 211
- Solid, 58
- SOLID – OCP, 153
- SPA, 897
- SplObjectStorage, 1349
- Status code, 198
- Statusy wgranego pliku, 170
- Stosy zasobów, 854
- Stream, 153
- Struktura aplikacji konsolowej, 366
- stty, 377
- Stub, 541
- Stub migracji, 543
- styl kodowania, 78
- Symfony, 365
 - brak natywnych WebSockets, 1397
- Symfony 8, 1439
- Symfony Mailer, 1487
- Szyfrowanie - generator kluczy, 1030
- Szyfrowanie - NONCE, 1027
- Szyfrowanie - OpenSSL, 1026
- Szyfrowanie - Sodium, 1028
- Szyfrowanie danych, 1026
- TableDefinition, 427
- Testowanie aplikacji konsolowej, 409
- Time Travel, 127
- Trasowanie, 247
- Trasowanie - parametry, 698
- Trasowanie zaawansowane, 691
- TTY, 377
- Twig, 1501
- Typy w PHP, 221
- UNIQUE, 443
- Uri, 146
- Uri\Rfc3986\Uri, 105
- UriInterface, 146
- VarDumper, 84
- Vhost, 85
- ViewServiceProvider, 868
- Vite, 857, 924
- Vue.js, 896

Vue3, 900
Vue3 - composition API, 902
Vue3 - dyrektywy, 900
Vue3 - options API, 901
Vue3 - szablony, 900

Walidacja po stronie klienta, 760
Walidacja po stronie serwera, 760
Walidator jako DTO, 764
WebSocket AI Chat, 1384
WebSocket Contracts, 1369
WebSocket Decoder, 1341
WebSocket Encoder, 1335
WebSocket handshake, 1332
WebSocket Opcode, 1332
WebSocket Server, 1360
websocket:start, 1373
WebSockets, 1331
 porównanie frameworków, 1397
WebSockets implementacja, 1333
WebSocketServiceProvider, 1371
Weryfikacja formularzy, 759
Wgrywany plik, 165
WHERE Clause, 480
Widoki, 813, 896
Wyjątki - debugowanie w konsoli, 1012
Wyjątki - klasa CodeSnippet, 936
Wyjątki - klasa TraceCollection, 939
Wyjątki - klasa TraceFrame, 938
Wyjątki HTTP Client, 313
Wysyłka maili, 1076
Wzorce Projektowe, 62
Wzorzec Bridge, 1481
Wzorzec Budowniczy, 62
Wzorzec Dekorator, 65, 165
Wzorzec Fabryka, 69, 203, 262
Wzorzec Fasada, 71
Wzorzec Grammar, 424
Wzorzec Kompozyt, 193
Wzorzec Strategia, 128, 165
Xdebug, 92
XSS, 102
YAGNI, 61
Zegar, 80