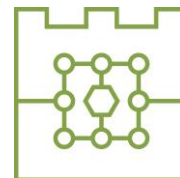




**Politechnika Krakowska
im. Tadeusza Kościuszki**

Wydział Informatyki i Telekomunikacji



Bartosz Szymański

Numer albumu: 148976

**Praktyczny przykład implementacji bezpiecznego
przeniesienia aplikacji na architekturę mikroserwisów
w chmurze AWS**

**A practical example of an implementation
of a monolithic application secure migration to AWS
cloud microservices architecture**

**Praca magisterska
na kierunku Informatyka
specjalność Cyberbezpieczeństwo**

Praca wykonana pod kierunkiem:
dr Barbary Borowik

Kraków, 2024

Spis treści

1	Wstęp	4
1.1	Cel pracy	4
1.2	Motywacja	4
1.3	Założenia i spodziewane wyniki	5
2	Ewaluacja zastanej architektury aplikacji	7
2.1	Logika biznesowa i baza danych	7
2.2	Warstwa widoku	12
3	Planowanie architektury mikroserwisowej w chmurze AWS	18
3.1	Wybór odpowiednich usług do utrzymania aplikacji	18
3.1.1	Amazon VPC - Virtual Private Cloud (ang. prywatna, wirtualna chmura)	18
3.1.2	Amazon ECS - Elastic Container Service (ang. elastyczny serwis kontenerów)	18
3.1.3	Amazon RDS - Relational Database Service (ang. zarządzany serwis relacyjnych baz danych)	19
3.1.4	Amazon MQ - Amazon Managed Message Broker Service (ang. zarządzany broker komunikatów dostarczany przez Amazon)	19
3.1.5	Amazon IAM - Identity and Access Management (ang. zarządzanie tożsamością i dostępem)	19
3.1.6	Amazon S3 - Simple Storage Service (ang. prosty magazyn danych)	20
3.1.7	Amazon CloudWatch	20
3.1.8	Amazon Route 53	21
3.1.9	Amazon Elastic Load Balancing	21
3.1.10	ACM - AWS Certificate Manager (ang. menadżer certyfikatów AWS)	21
3.1.11	Amazon API Gateway (ang. Bramka API Amazon)	22

3.2	Opracowanie planu migracji	22
3.2.1	Domain-Driven Design	22
3.2.2	Zakres migracji	23
3.2.3	Podejście Event Storming	23
4	Przebieg migracji	26
4.1	Faza pierwsza	26
4.2	Faza druga	26
4.3	Faza trzecia	27
4.4	Faza czwarta	28
4.5	Faza piąta	28
4.6	Faza szósta	29
4.7	Faza siódma	29
4.8	Faza ósma	29
4.9	Podsumowanie przebiegu migracji	29
5	Bezpieczeństwo	35
5.1	Poziom zabezpieczeń przed migracją	35
5.2	Bezpieczeństwo architektury chmurowej	36
5.3	Bezpieczeństwo komunikacji	36
5.4	Zarządzanie danymi	37
5.5	Bezpieczeństwo aplikacji	38
5.6	Podsumowanie	43

1. WSTĘP

1.1 Cel pracy

Celem niniejszej pracy dyplomowej jest zbadanie procesu migracji systemu monolitycznego do architektury mikroservisowej opartej o usługi chmury obliczeniowej dostawcy Amazon Web Services (w skrócie AWS) pod kątem ulepszenia zabezpieczeń omawianego systemu. System objęty badaniem, to: „System do internetowego wspomagania pacjenta i lekarza”, który został w całości zaprojektowany i zaimplementowany w formie monolitycznej przez autora pracy dyplomowej jako projekt inżynierski w celu ukończenia studiów inżynierskich pierwszego stopnia. Wybór systemu do przeprowadzenia analizy motywowany jest znajomością jego architektury, wpasowującą się doskonale w temat pracy oraz motywacja do jego dalszego rozwijania i ulepszania w zakresie cyberbezpieczeństwa.

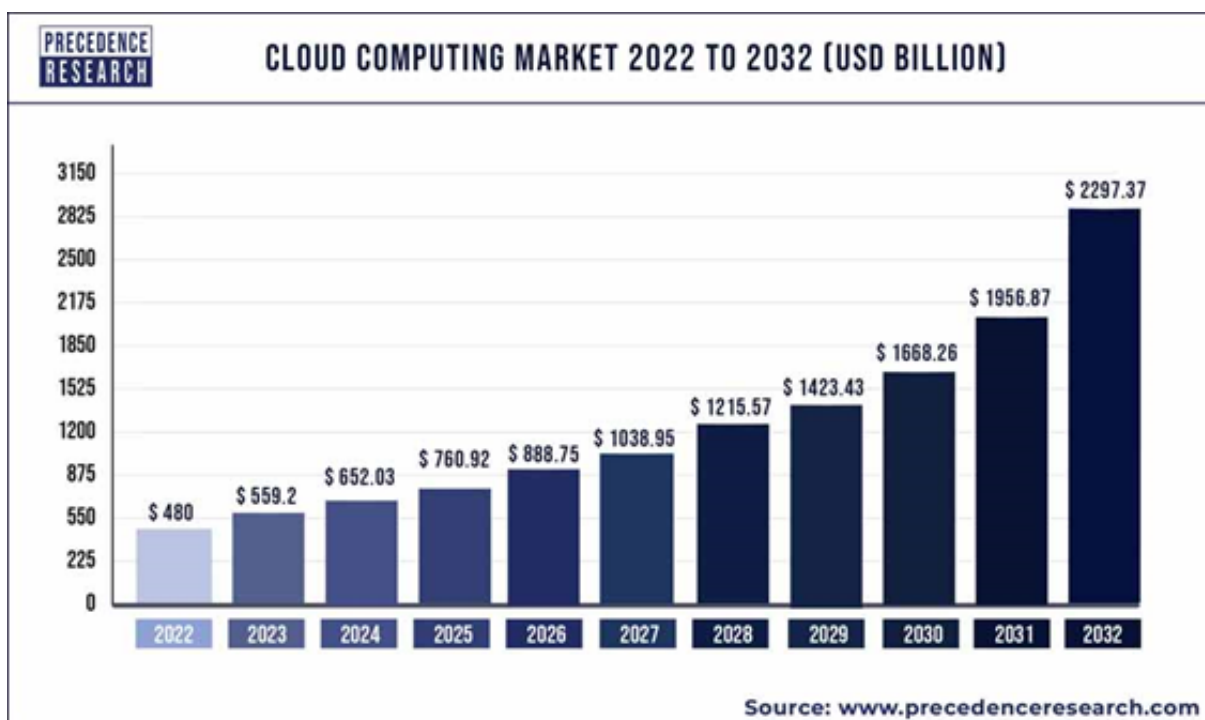
Analiza procesu migracji oraz rozwinięcia systemu zabezpieczeń systemu ma wykazać jak dobrze usługi chmurowe ochraniają swoich usługobiorców i ich oprogramowania przed popularnymi atakami hakerskimi oraz jak duży wpływ na bezpieczeństwo systemu ma jego architektura. Narzędzia używane przez autor pracy to przede wszystkim oprogramowanie wirtualizujące systemy operacyjne Docker, język programowania PHP, język programowania JavaScript, zestaw narzędzi programistycznych Symfony oraz Vue.js, dodatkowo usługi AWS, takie jak: Elastic Compute Cloud (w skrócie EC2), Relational Databases (w skrócie RDS), Route 53, Code Pipeline, Code Build, Code Deploy, Elastic Cache, Simple Storage Service, Elastic Container Registry, Lambda. Również narzędzia do analizy ruchu sieciowego, takie jak: Wireshark.

Praca ta omówia aspekty takie jak: ewaluacja zastanej architektury aplikacji, planowanie architektury mikroservisowej w chmurze AWS, ocena zagadnień bezpieczeństwa podczas migracji, implementacja i wdrażanie mikroservisów w chmurze AWS, ocena i analiza wyników. Każdy wymieniony etap pracy jest szczegółowo omówiony, odzwierciedlając wiedzę i doświadczenie akademickie jak i zawodowe autora.

1.2 Motywacja

Motywacją do podjęcia tematu pracy są doświadczenia autora na tle zawodowym, które przyspieszyły naukę w zakresie obliczeń chmurowych. Jednak największym motywatorem do wykonania badania jest kontynuacja pracy akademickiej, dążenie do ciągłego rozwoju wyprodukowanego przez siebie oprogramowania i wdrażanie wiedzy pozyskanej w czasie studiów

drugiego stopnia, tak by umiejętnie zabezpieczać oprogramowanie. Dodatkowym motywato-rem jest również fakt, iż technologia chmurowa każdego roku zdobywa coraz większy udział na rynku usług hostingowych, a także staje się dzięki wzrastającej konkurencyjności przystępniej-sza dla mniejszych firm lub prywatnych odbiorców. Serwis Precedence Research przewiduje, iż w nadchodzących ośmiu latach udział rynkowy technologii chmurowych zwiększy się około pięciokrotnie do wartości 2297 miliardów dolarów [22]. Patrz rysunek 1.1.



Rys. 1.1. Wykres przedstawiający prognozę wzrostu udziału rynkowego technologii chmurowej

1.3 Założenia i spodziewane wyniki

Rezultatem niniejszej pracy dyplomowej jest rozebranie monolitycznego systemu na mikroserwisy z pojedynczą odpowiedzialnością logiczną. Zasada działa aplikacji nie ulega zmianie, dla użytkownika zewnętrznego wprowadzenie zmiany powinno odbyć się bez odczuwalnej różnicy. Samo rozczłonkowanie systemu odbywa jak najbardziej atomicznie się da przy rozsądnym nakładzie pracy pozwalającym ukończyć tę pracę w terminie jej obrony. Autor zakłada, iż poprawie ulegnie bezpieczeństwo danych przechowywanych w bazie danych poprzez ukrycie bazy danych przed dostępem z zewnątrz przy wykorzystaniu wirtualnej sieci prywatnej, także większa granularność aplikacji poprawi bezpieczeństwo wdrożenia zmian poprzez usystematyzowanie procesu jaki za tym stoi poprzez wykorzystanie technologii „Pipeline” (ciąg

zautomatyzowanych procesów dostarczania nowego oprogramowania). Również autor odświeżył wersję zależności z jakich składa się projekt, tak by rozwiązać problemy luk bezpieczeństwa wynikających z użycia przestarzałych bibliotek. Dodatkowym poziomem ulepszenia zabezpieczeń aplikacji będzie przeszukanie kodu w pod kątem znalezienia luk logicznych lub twardo zakodowanych dostępu, takich jak hasła do bazy danych lub inne, z których aplikacja korzysta.

2. EWALUACJA ZASTANEJ ARCHITEKTURY APLIKACJI

Zgodnie z stanem aplikacji z dnia 11.06.2022r. projekt “System do internetowego wspomagania pacjenta i lekarza” oparty jest o kilka składowych jakie należy wyróżnić, a są nimi:

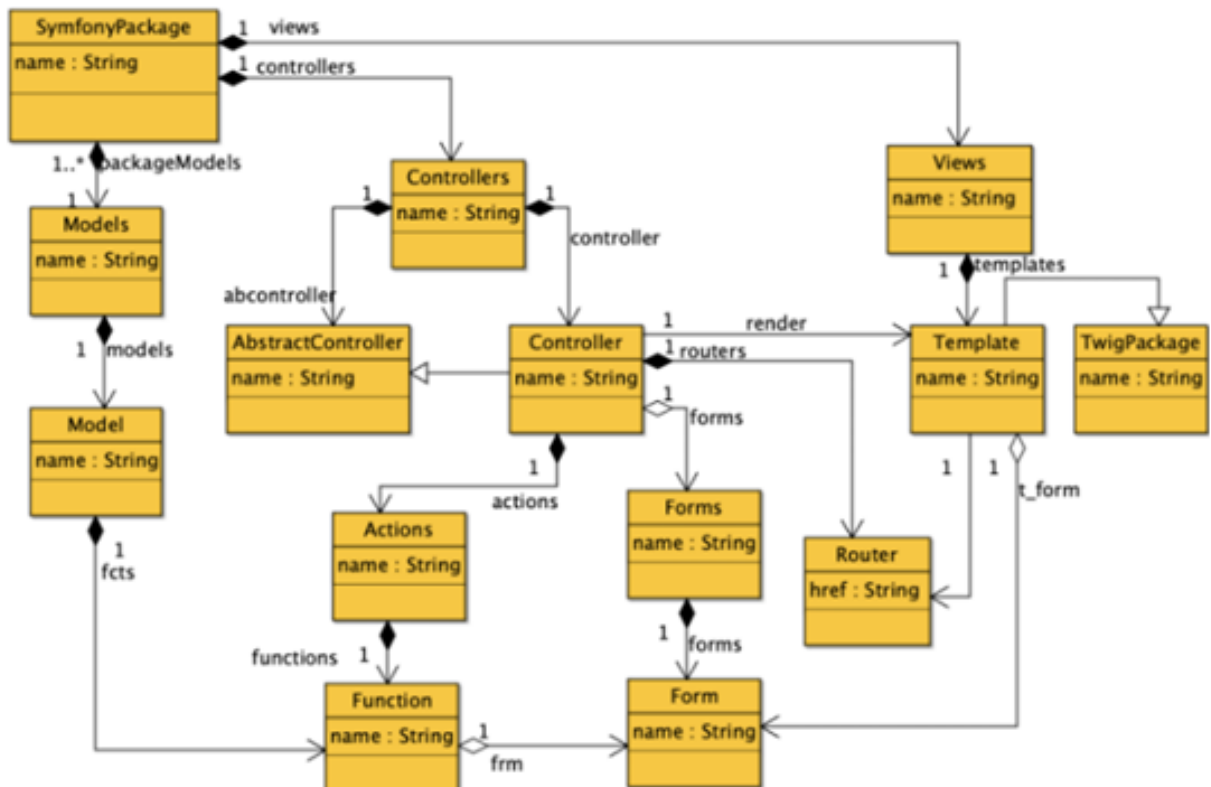
- baza danych oprata o silnik MySQL,
- trzon logiki biznesowej oparty o zestaw narzędzi Symfony w wersji 5.4,
- warstwa widoku aplikacji oparta o silnik Twig oraz Vue.js,
- silnik nginx jako narzędzie do przekazania zapytania klienckiego do logiki biznesowej.

Wyróżnione komponenty są ściśle ze sobą połączone. Logika biznesowa oraz baza danych są nierozrwalne, logika biznesowa posiada klasy encji, które bezpośrednio przekładają się na tabele w bazie danych. Logika biznesowa również decyduje o tym jaki widok jest obecnie wyświetlany, a warstwa widoku bazuje na tym, co dostarczył kontroler Symfony. Cały zestaw wykorzystuje narzędzie nginx do tego, by otrzymać zapytanie od klienta. W niniejszym rozdziale zostaną przedstawione kolejno wyżej wymienione komponenty z szerszym opisem co do każdego z nich.

2.1 *Logika biznesowa i baza danych*

Symfony jako framework, który za zadanie ma ułatwić rozwój oprogramowania internetowego w projekcie z pracy inżynierskiej autora niniejszej pracy spełnił swoją rolę niejako przyczyniając się do skrócenia czasu potrzebnego na rzecz implementacji założeń diagramów użyć, odzwierciedlenie domen każdego z aktorów systemu, a także diagramów przepływu danych. Narzędzia takie jak wbudowane kontrolery z dostępnym API do zapytań, czy formularze do walidacji danych wejściowych do systemu, router odpowiadający na zapytania, czy szablony wspierane przez silnik Twig to tylko niektóre z narzędzi jakie zostały wykorzystane przy okazji implementacji. Wyczerpujący schemat udogodnień oferowanych przez Symfony znajduje się na rysunku 2.1.

W celu odzwierciedlenia domen aktorów, w projekcie została wykorzystana biblioteka Doctrine, wspierana przez Symfony. Samo Doctrine to narzędzie do manipulacji bazą danych przy wykorzystaniu obiektów PHP [27]. By zainicjować dane w systemie bez konieczności manualnego wprowadzania ich autor wykorzystał dedykowaną ku temu bibliotekę wykorzystującą mechanizm fabryki danych testowych (ang. Fixtures Factory). Sam mechanizm fabryki danych



Rys. 2.1. Schemat metamodelu Symfony [28]

testowych w procesie tworzenia oprogramowania umożliwia uruchomienie systemu z wypełnioną bazą danych, tak by móc przeprowadzać wymagane akcje i sprawdzić poprawność logiki biznesowej [23]. Fabryka kreująca dane testowe została przedstawiona na listingu 2.1.

```

1 final class ClinicFactory extends ModelFactory
2 {
3     #[ArrayShape([
4         'name' => "string",
5         'country' => "string",
6         'city' => "string",
7         'email' => "string",
8         'zipCode' => "string",
9         'streetAddress' => "string"
10    ]) protected function getDefaults(): array
11    {
12        return [
13            'name' => self::faker()->company(),
14            'country' => 'Polska',
15            'city' => 'Kraków',

```



```

16         'email' => self::faker()->email(),
17         'zipCode' => $this->getRandomCracowPostCode(),
18         'streetAddress' => self::faker()->streetAddress(),
19     ];
20 }
21
22     protected function initialize(): self
23     {
24         // see https://symfony.com/bundles/ZenstruckFoundryBundle/current/index.
25         // html#initialization
26         return $this
27         // ->afterInstantiate(function(Clinic $clinic): void {})
28         ;
29     }
30
31     protected static function getClass(): string
32     {
33         return Clinic::class;
34     }
35
36     /**
37      * @throws Exception
38      */
39     private function getRandomCracowPostCode(): string
40     {
41         $first = '3';
42         $secondArray = ['0', '1'];
43         $second = $secondArray[array_rand($secondArray)];
44         $result = sprintf('%s%s-', $first, $second);
45         for ($i = 0; $i < 3; $i++) {
46             $result .= random_int(0, 9);
47         }
48
49         return $result;
50     }
51 }

```

Listing 2.1: Przykładowa fabryka danych testowych w Symfony

Do weryfikacji poprawności danych wykorzystano wbudowany w Symfony mechanizm

formularzy. Każdy formularz jest tworzony na potrzeby obsługi konkretnego obiektu encji lub specyficznego rodzaju danych, jak na przykład numer PESEL. Formularz przeznaczony do walidacji encji zawiera pola odpowiadające właściwościom tej encji oraz określa typy tych pól. Każde pole może mieć przypisany szereg ustawień, z których najważniejsze jest ustawienie przechowujące tablicę ograniczeń dotyczących wartości przekazywanych do danego pola.

Wykorzystując ten mechanizm, można na poziomie walidacji danych przeprowadzić weryfikację, czy np. encja z podanymi danymi, które mają być unikalne, znajduje się już w bazie danych. Przykładowy formularz został przedstawiony na listingu 2.2.

```
1 abstract class AbstractUserFormType extends AbstractType
2 {
3     public function __construct(protected readonly EntityManagerInterface
4         $entityManager)
5     {
6
7     public function buildForm(FormBuilderInterface $builder, array $options
8         ): void
9     {
10         $builder
11             ->add('email', EmailType::class, [
12                 'required' => true,
13                 'constraints' => [
14                     new NotBlank(),
15                     new Email(),
16                     new Callback([
17                         'callback' => function ($email,
18                             ExecutionContextInterface $context) {
19                             $user = $this->entityManager->getRepository(
20                                 User::class)->findOneBy(['email' => $email])
21                             ;
22                             if ($user) {
23                                 $context->addViolation('Istnieje już
24                                     użytkownik o podanym adresie e-mail');
```

```

24     })
25     ->add('password', RepeatedType::class, [
26         'required' => true,
27         'type' => PasswordType::class,
28         'constraints' => [new NotBlank()],
29     ])
30     ->add('firstName', TextType::class, [
31         'required' => true,
32         'constraints' => [new NotBlank()],
33     ])
34     ->add('secondName', TextType::class, [
35         'required' => true,
36     ])
37     ->add('lastName', TextType::class, [
38         'required' => true,
39         'constraints' => [new NotBlank()],
40     ])
41     ->add('agreeTerms', CheckboxType::class, [
42         'required' => true,
43         'constraints' => [
44             new IsTrue(),
45         ],
46     ]);
47 }
48 }

```

Listing 2.2: Formularz danych rejestracji

Do pobierania danych z bazy danych wykorzystano mechanizm repozytorium, który jest ściśle powiązany z odpowiednią encją. Przykładowo, encja `PatientData` posiada swoje repozytorium `PatientDataRepository`. Repozytorium powinno być zbudowane w sposób możliwie najbardziej generyczny, operując wyłącznie na encjach lub danych pochodzących z encji. Przykładowy kod repozytorium został przedstawiony na rysunku 2.3.

Ostatnim z mechanizmów Symfony, który został wykorzystany przez autora niniejszej pracy do zbudowania pierwotnej wersji systemu do internetowego wspomagania pacjenta i lekarza, jest mechanizm walidatora. Jest to część aplikacji współpracująca z formularzami, która umożliwia niestandardową weryfikację danych. Przykładowy walidator został przedstawiony na rysunku 2.4.

Powyżej przedstawiono elementy zestawu narzędzi programistycznych Symphony, które zostały wykorzystane w budowaniu projektu. Zastosowanie tych rozwiązań pozwoliło autorowi skorzystać ze sprawdzonych mechanizmów, które stworzyły bezpieczne środowisko do implementacji założeń biznesowych. Proces implementacji z pewnością byłby trudniejszy bez ich obecności.

2.2 Warstwa widoku

Rozdział ten opisuje sposób implementacji warstwy widoku aplikacji. Do przeprowadzenia procesu implementacji zostały wykorzystane narzędzia takie jak:

- Twig, który jest silnikiem szablonów dla systemów opartych o język programowania PHP,
- Vue.js, który dedykowany jest tworzeniu interaktywnych interfejsów użytkownika.

Wybór tych narzędzi został podyktowany personalnymi preferencjami autora pracy. Szablony Twig składają się z kilku sekcji, które zostały zdefiniowane przez autora w celu usprawnienia procesu tworzenia. Bazowy szablon przedstawiony jest na rysunku 2.5 i zawiera on bloki takie jak:

- `title` - blok zawierający tytuł strony,
- `stylesheets` - blok zawierający odwołania do arkuszy stylów,
- `body` - blok zawierający treść strony,
- `javascripts` - blok zawierający odwołania do skryptów JavaScript.

Na rysunku 2.6 przedstawiono przykładowy kod szablonu, który dziedziczy po szablonie bazowym.

```

abstract class AbstractUserFormType extends AbstractType
{
    public function __construct(protected readonly EntityManagerInterface $entityManager)
    {
    }

    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('email', EmailType::class, [
                'required' => true,
                'constraints' => [
                    new NotBlank(),
                    new Email(),
                    new Callback([
                        'callback' => function ($email, ExecutionContextInterface $context) {
                            $user = $this->entityManager->getRepository(User::class)->findOneBy(['email' => $email]);
                            if ($user) {
                                $context->addViolation('Istnieje już użytkownik o podanym adresie e-mail');
                            }
                        }
                    ])
                ],
            ],
        )
            ->add('password', RepeatedType::class, [
                'required' => true,
                'type' => PasswordType::class,
                'constraints' => [new NotBlank()],
            ])
            ->add('firstName', TextType::class, [
                'required' => true,
                'constraints' => [new NotBlank()],
            ])
            ->add('secondName', TextType::class, [
                'required' => true,
            ])
            ->add('lastName', TextType::class, [
                'required' => true,
                'constraints' => [new NotBlank()],
            ])
            ->add('agreeTerms', CheckboxType::class, [
                'required' => true,
                'constraints' => [
                    new IsTrue(),
                ],
            ],
        );
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'csrf_protection' => false,
        ]);
    }
}

```

Rys. 2.2. Zrzut ekranu przedstawiający przykładowy formularz w Symfony

```

class PatientDataRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, PatientData::class);
    }

    public function findPatientsDataByDoctorDataInAppointment(DoctorData $doctor)
    {
        $qb = $this->createQueryBuilder('pd');

        return $qb->join('pd.appointments', 'a')
            ->where($qb->expr()->eq('a.doctor', ':doctor'))
            ->setParameter('doctor', $doctor)
            ->getQuery()
            ->getResult();
    }
}

```

Rys. 2.3. Zrzut ekranu przedstawiający przykładowe repozytorium w Symfony

```

class WorkingDayValidator extends ConstraintValidator
{
    public function validate($value, Constraint $constraint): void
    {
        /* @var $constraint WorkingDay */
        if (!$constraint instanceof WorkingDay) {
            throw new UnexpectedTypeException($constraint, WorkingDay::class);
        }

        $start = $value['start'] ?? null;
        $end = $value['end'] ?? null;
        if (!$start && !$end) {
            return;
        }

        if (!is_array($value)) {
            throw new UnexpectedTypeException($value, 'array');
        }

        if (!$this->isValidWorkingDay($value)) {
            $this->context->buildViolation($constraint->message)
                ->setParameter('{{ value }}', $this->getInvalidValue($value))
                ->addViolation();
        }
    }

    private function isValidWorkingDay(array $workingDay): bool
    {
        $start = $workingDay['start'] ?? null;
        $end = $workingDay['end'] ?? null;

        if (($start && !$end) || (!$start && $end)) {
            return false;
        }

        return $end > $start;
    }

    private function getInvalidValue($value): string
    {
        return sprintf(
            '%s i %s',
            $this->getInvalidPartialValue($value, 'start'),
            $this->getInvalidPartialValue($value, 'end')
        );
    }

    private function getInvalidPartialValue($value, $key): string
    {
        $emptyResult = 'brak godziny oraz minuty';
        $format = 'H:i';
        /** @var DateTime $partialValue */
        $partialValue = $value[$key] ?? null;

        return $partialValue ? $partialValue->format($format) : $emptyResult;
    }
}

```

Rys. 2.4. Zrzut ekranu przedstawiający przykładowe walidator w Symfony

```

<!DOCTYPE html>
<html lang="pl-Pl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>{% block title %}{% endblock %}</title>
  {% block stylesheets %}
    <link href="https://cdn.jsdelivr.net/npm/@mdi/font@4.x/css/materialdesignicons.min.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/vuetify@2.x/dist/vuetify.min.css" rel="stylesheet">
    {{ encore_entry_link_tags('app') }}
  {% endblock %}
</head>
<body>
  {% block body %}{% endblock %}
  {% block javascripts %}{% endblock %}
</body>
</html>

```

Rys. 2.5. Zrzut ekranu przedstawiający przykłady szablon Twig


```
{% extends 'base.html.twig' %}

{% block title %}Strona główna | SDIWPIŁ{% endblock %}

{% block stylesheets %}
    {{ parent() }}
    {{ encore_entry_link_tags('home_page') }}
{% endblock %}

{% block body %}
    <div id="main"></div>
{% endblock %}

{% block javascripts %}
    <script type="text/javascript">
        window.state = {{ state|raw }}
    </script>
    {{ encore_entry_script_tags('home_page') }}
{% endblock %}
```

Rys. 2.6. Zrzut ekranu przedstawiający przykładowy szablon dziedziczący po szablonie bazowym

3. PLANOWANIE ARCHITEKTURY MIKROSERWISOWEJ W CHMURZE AWS

Niniejszy rozdział pokrywa zagadnienie planowania architektury mikroserwisowej oraz wykorzystanych usług potrzebnych do realizacji celu uruchomienia platformy. Nie istnieje mapa działań jakie należy wykonać by zgodnie z sztuką podzielić aplikację monolityczną na zbiór mikroserwisów. Istnieją natomiast wskazówki, którymi można się posługiwać by ułatwić doprowadzenie tego przedsięwzięcia do końca, a są nimi:

- identyfikacja mikroserwisów,
- dbanie o szczególną troskę w procesie ekstrakcji modułów, które są kandydatami na mikroserwisy,
- wprowadzenie modelu heksagonalnego aplikacji. [29]

3.1 Wybór odpowiednich usług do utrzymania aplikacji

W wyborze usług potrzebnych do użycia dla architektury mikroserwisowej, autor korzystał z własnych doświadczeń w pracy z platformą AWS. W związku z powyższym powody wyboru konkretnej usługi zostaną przedstawione w podrozdziale poświęconym na rzecz opisanie motywacji doboru.

3.1.1 Amazon VPC - Virtual Private Cloud (ang. prywatna, wirtualna chmura)

Powody wyboru VPC dla celu realizacji architektury sieciowej:

- możliwość tworzenia wyizolowanych sieci w chmurze, co zapewnia wysoki standard bezpieczeństwa dla systemu, pozwala kontrolować ruch sieciowy przy pomocy reguł zapory sieciowej,
- możliwość kontrolowania adresacji IP, podsieci, tabel routingu i bram internetowych, co pozwala na dostosowanie sieci do indywidualnych potrzeb budowanego przez autora systemu. [21]

3.1.2 Amazon ECS - Elastic Container Service (ang. elastyczny serwis kontenerów)

Wybór ECS jako usługi pozwalającej na uruchomienie mikroserwisów jest uargumentowany niniejszymi powodami:

- ułatwione zarządzanie kontenerami poprzez wbudowane narzędzia do orkiestracji nimi,
- możliwość uruchomienia kontenerów bez konieczności zarządzania architekturą serwerową. [18]

3.1.3 *Amazon RDS - Relational Database Service (ang. zarządzany serwis relacyjnych baz danych)*

RDS jest idealnym wyborem do zarządzania bazami danych z następujących powodów:

- automatyzuje zadania typu tworzenie kopii zapasowych, aktualizacje oprogramowania oraz monitorowanie,
- posiada rozwiązania gwarantujące wysoką dostępność,
- oferuje szyfrowanie danych w spoczynku i w trakcie przesyłania oraz przy wykorzystaniu VPC do wyizolowania usługi zapewnia wysoki poziom bezpieczeństwa. [19]

3.1.4 *Amazon MQ - Amazon Managed Message Broker Service (ang. zarządzany broker komunikatów dostarczany przez Amazon)*

Z racji wykorzystania protokołu AMQP w projekcie migracji systemu, Amazon MQ jest odpowiednim wyborem do zarządzania komunikacją między mikroservisami z następujących powodów:

- umożliwia zautomatyzowanie konfiguracji, skalowania i zarządzania infrastrukturą brokerską,
- obsługuje popularne protokoły wiadomości, takie jak: AMQP, MQTT oraz STOMP, najbardziej kluczowym dla aplikacji systemu do internetowego wspomaganie pacjenta i lekarza jest AMQP,
- posiada wbudowane mechanizmy redundancji i automatycznego przełączania awaryjnego,
- oferuje szyfrowanie danych oraz izolację środowiska w VPC. [2]

3.1.5 *Amazon IAM - Identity and Access Management (ang. zarządzanie tożsamością i dostępem)*

Najważniejszymi powodami do wyboru IAM jako usługi zarządzającej dostępem są:

- precyzyjne zarządzanie dostępem do wszystkich zasobów AWS,

- umożliwienie definiowania szczegółowych polityk uprawnień dla użytkowników, grup i ról,
- zapewnia wysoki poziom bezpieczeństwa poprzez funkcje takie jak dwuskładnikowe uwierzytelnianie (MFA), tymczasowe poświadczenia oraz możliwość monitorowania działań użytkowników,
- IAM udostępnia centralne zarządzanie tożsamościami i uprawnieniami, przez co upraszcza administrację i dodaje przejrzystości zarządzania dostępem. [3]

3.1.6 *Amazon S3 - Simple Storage Service (ang. prosty magazyn danych)*

Przechowywanie danych aplikacji to kluczowe zadanie, jakie zostało postawione usłudze S3, a jej wybór został podyktowany następującymi powodami:

- usługa ta automatycznie skaluje się w górę lub dół, przez co umożliwia praktycznie nieograniczone ilościowo składowanie danych,
- oferuje wysoką trwałość danych na poziomie bliskim 100%, a także wysoką dostępność,
- zapewnia szyfrowanie danych, zarządzanie dostępem na poziomie obiektu oraz integrację z AWS IAM,
- umożliwia optymalizację kosztów na podstawie częstotliwości dostępu do danych i wymagań dotyczących trwałości. [20]

3.1.7 *Amazon CloudWatch*

Usługa o nazwie CloudWatch, w szerokim spektrum usług jakie oferuje platforma AWS, dedykowana jest monitorowaniu i logowaniu akcji wykonanych w samych usługach, jak i w kodzie aplikacji. To są powody jakie zostały uwzględnione przy podjęciu decyzji o skorzystaniu z niej:

- umożliwia centralne zarządzanie i analizę danych,
- oferuje zaawansowane możliwości monitorowania aplikacji i infrastruktury, a także konfigurację alarmów, które powiadamiają o problemach w czasie rzeczywistym,
- na podstawie logów zbieranych przez CloudWatch, można zautomatyzować uruchamianie akcji skalowania zasobów, czy też funkcji Lambda. [4]

3.1.8 *Amazon Route 53*

Jako, iż system modernizowany przez autora niniejszej pracy dyplomowej jest aplikacją internetową, to należy zapewnić mu dostęp do domeny, która będzie propagować jego usługi na cały świat. Usługą, w portfolio AWS, która zapewnia takie rozwiązania jest Amazon Route 53. Poniżej przedstawione są najważniejsze powody, które zostały uwzględnione przy dokonaniu wyboru:

- wysoka dostępność i niezawodność gwarantowane przez rozproszenie usługi DNS,
- elastyczność i skalowalność,
- zintegrowane funkcje geolokalizacji,
- integracja z innymi, ważnymi z punktu widzenia systemu usługami, takimi jak: Elastic Load Balancing i CloudFront. [16]

3.1.9 *Amazon Elastic Load Balancing*

Oferowana przez firmę Amazon usługa Elastic Load Balancing (ang. elastyczne balansowanie obciążeniem) jest kluczowym elementem każdej aplikacji z uwagi na rozproszenie obciążenia systemu. Niżej przedstawiono wszystkie powody wykorzystania omawianej usługi:

- automatycznie rozkładanie ruchu sieciowego między wieloma zasobami,
- poprawa dostępności aplikacji poprzez monitorowanie stanu zasobów i automatyczne przekierowanie ruchu do zdrowych instancji,
- zarządzanie ruchem na podstawie zawartości, geolokalizacji i opóźnienia między żądanym zasobem,
- integracja z AWS Certificate Manager do zarządzania certyfikatami SSL/TLS, a także ochrona przez atakami DDoS i zarządzanie ruchem HTTP/2,
- integracja z usługą Auto Scaling (ang. automatyczne skalowanie) oraz CloudWatch. [17]

3.1.10 *ACM - AWS Certificate Manager (ang. menadżer certyfikatów AWS)*

Narzędzie ACM to natywne rozwiązanie chmury AWS oferujące szereg udogodnień istotnych z punktu widzenia cyberbezpieczeństwa, a są nimi:

- automatyzacja procesu tworzenia, wdrażania i odnawiania SSL/TLS,

- integracja z Elastic Load Balancing (ELB), Amazon CloudFront i API Gateway (ang. bramka API),
- bezpłatne utrzymanie certyfikatów SSL/TLS dla domen zarządzanych przez AWS,
- łatwość użycia poprzez uproszczenie procesu zarządzania certyfikatami dzięki intuicyjnemu interfejsowi. [5]

3.1.11 *Amazon API Gateway (ang. Bramka API Amazon)*

Usługa Amazon API Gateway została stworzona z myślą o maksymalnym uproszczeniu tworzenia struktury API i łączenia zasobów w jedną całość. Autor niniejszej pracy dyplomowej wykorzystał ją, po uprzednim zgłębieniu jej zalet, a są nimi:

- integracja z Amazon Elastic Load Balancing,
- zarządzanie ruchem poprzez mechanizm Throttling Rules (ang. zasady tłumienia),
- łatwość tworzenia API i jego wdrożenia,
- monitoring operacji wykonywanych wewnątrz API. [1]

3.2 *Opracowanie planu migracji*

Migracja działającego systemu monolitycznego na architekturę mikroserwisów jest wyzwaniem. Wiąże się tym duży koszt wstępny jaki musi ponieść przedsiębiorstwo, by uruchomić szereg koniecznych procesów pozwalających przeprowadzić odpowiednio migrację. Samą migrację należy przeprowadzić w pełnej świadomości i po przeprowadzeniu uprzednio audytu opłacalności tego zabiegu. Zabieg ten wprowadza znaczne skomplikowanie do projektu i wymaga wykwalifikowanej kadry inżynierów, którzy będą w stanie zapewnić ciągłość działania aplikacji w fazach przejściowych, a przy okazji rozwijając nową funkcjonalność. W niniejszej pracy dyplomowej migracja zostanie przeprowadzona na systemie hobbystycznym, a co za tym idzie jego ewentualna przerwa w działaniu nie zadziała destrukcyjnie na ewentualne przedsiębiorstwo, które mogłoby ponieść kosztą niedziałającej platformy.

3.2.1 *Domain-Driven Design*

W skrócie DDD. Nazwa ta w języku polskim oznacza nacisk na prowadzenie rozwoju oprogramowania w kontekście jakiejś domeny. Jest to filozofia, która ma pomagać rozwiązywać problemy budowy oprogramowania dla skomplikowanych domen [24]. To podejście zakłada

podział systemu na komponenty oraz ich zachowania, aby te wiernie odzwierciedlały rzeczywistość. Po wstępnej fazie planowania obszarów systemu następuje przejście do fazy implementacji.

Zadaniem autora w niniejszej pracy dyplomowej było wykorzystanie potencjału jaki został stworzony w jego pracy inżynierskiej. Mianowicie w omawianej pracy system został podzielony w naturalny sposób pośród czterech aktorów, jakimi są: pacjent, lekarz, pracownik recepcji oraz pracownik administracji. W tego podziału wyłania się domena podziału. Każdy aktor ma swój zestaw czynności jakie w swojej domenie może wykonywać. Tak na przykład każdy użytkownik ma prawa do edycji swojego konta, usunięcia go, oraz podglądu danych jakie się w nim znajdują. Przyglądając się funkcjonalności wizyt wyłania się osobna zależność, która jest możliwa do wydzielenia z całości systemu, a dostęp mają do niej aktorzy tacy jak: pacjent, lekarz i pracownik recepcji. Każdy z aktorów ma też swój indywidualny zestaw danych, który jest niezależny od innego z aktorów, co kwalifikuje ten fakt na uwzględnienie go jako podział na cztery dodatkowe domeny. Osobną zależnością jest również sama wysyłka maili, która z punktu widzenia systemu jest istotna, ale bardzo mała porównując zakres czynności jakie wykonuje.

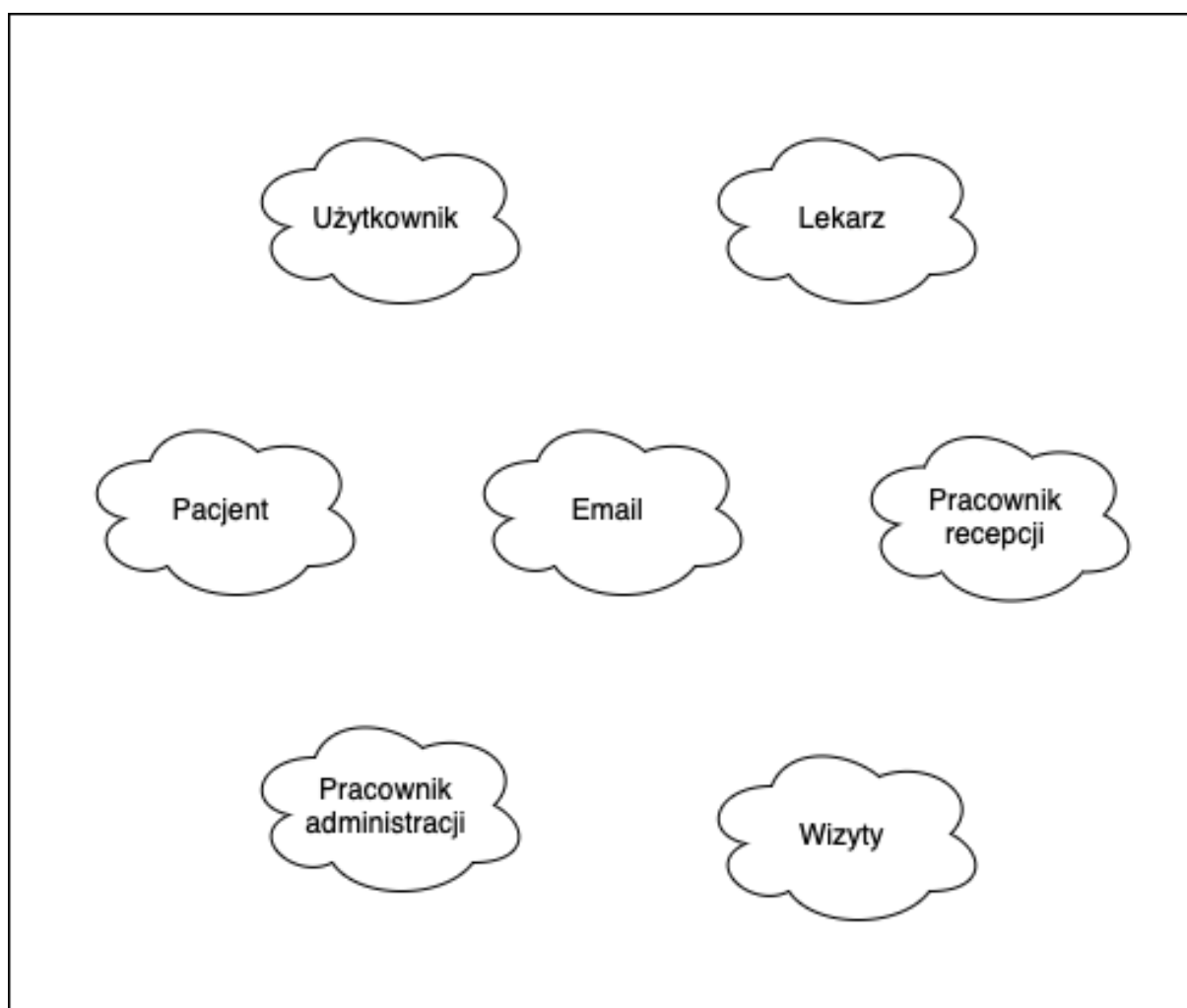
Dalsze działania wynikające z planowania będą bazować bezpośrednio na wytyczonym powyżej podziale. Sam podział nie jest nowy w kontekście całego systemu. Autor zgodnie z sztuką zbudował projekt, tak by w przyszłości móc go rozwinąć właśnie poprzez modyfikację w kierunku mikroservisów.

3.2.2 *Zakres migracji*

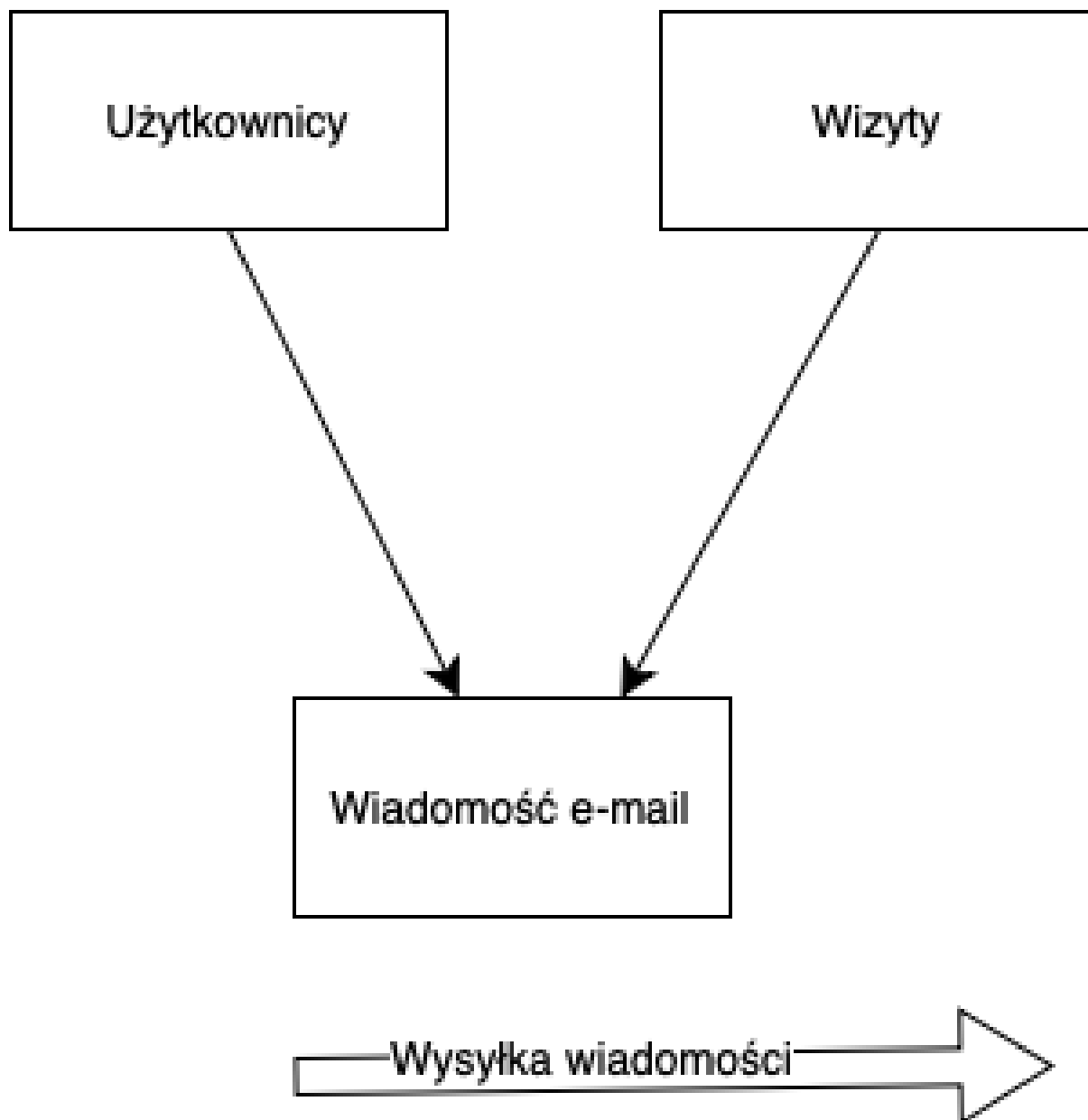
Według autora książki “Monolith to Microservices” należy zrozumieć w jakim zakresie szczegółowa ma być domena, którą próbuje się dzielić na mniejsze części, tak by było to rozsądne [25]. Patrz rysunek 3.1.

3.2.3 *Podejście Event Storming*

Podejście to polega na zgrupowaniu potencjalnych domen systemu za pośrednictwem akcji jakie wykonują. Tak na przykład w systemie do internetowego wspomaganie pacjenta i lekarza da się pogrupować czynności wykonywane w systemie pod kątem wysyłki wiadomości e-mail. Na (Wstaw referencję obrazka) widoczny jest schemat pokazujący, że poszczególne domeny, takie jak: użytkownicy, wizyty są zgrupowane do akcji wysyłania wiadomości drogą mailową. Całość potencjału omawianego podejścia została przedstawiona na rysunku 3.2.



Rys. 3.1. Zakres domenowy systemu do internetowego wspomagania pacjenta i lekarza



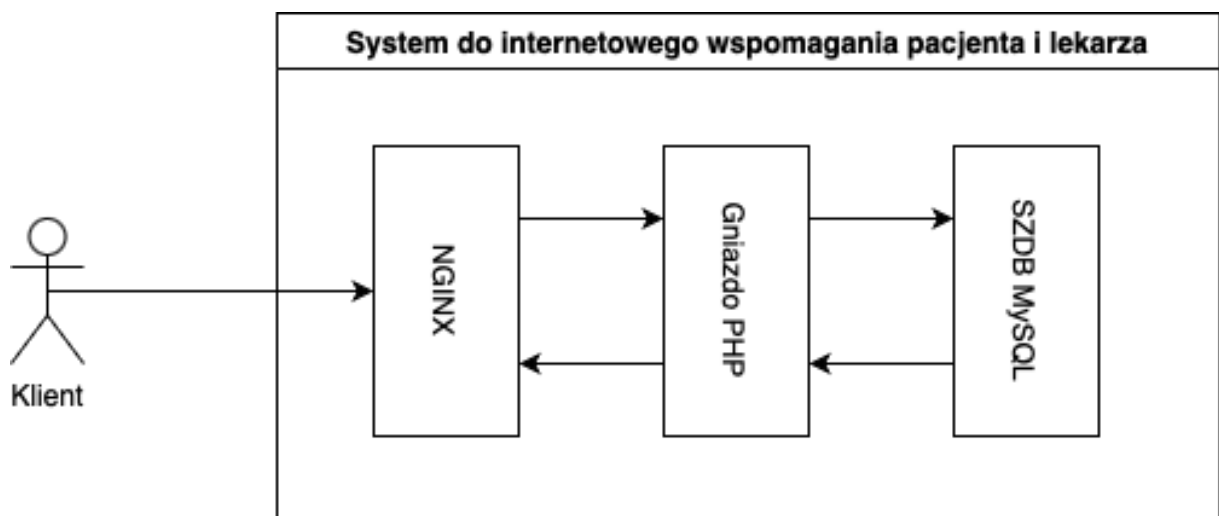
Rys. 3.2. Wysyłka wiadomości jest logicznie połączona dla tego modelu domeny, więc jej dalsza ekstrakcja może być trudna

4. PRZEBIEG MIGRACJI

W niniejszym rozdziale autor pracy dyplomowej przedstawił przebieg migracji, dzieląc go na osiem faz pośrednich. Przebieg ten jest odzwierciedleniem nakładu prac wykonanego przez autora na cel migracji systemu. Jest to połowa z potrzebnych działań praktycznych, ponieważ dalsza ich część to uruchomienie infrastruktury chmurowej w AWS.

4.1 Faza pierwsza

Po fazie planowania należy przejść do fazy, w której mając schemat podzielona zostanie całkowita migracja na fazy, tak by utrzymać ciągłość działania systemu. System uprzednio uruchomiony został w zvirtualizowanym środowisku Vagrant, gdzie miał zainstalowanie, które było proxy (ang. pośrednikiem) zapytań - nginx. W tym samym środowisku uruchomiony był również system do zarządzania relacyjną bazą danych - MySQL. Dodatkowo uruchomiona była tam również logika biznesowa przy pomocy gniazda PHP. Schematyczny obraz fazy pierwszej znajduje się na rysunku 4.1.

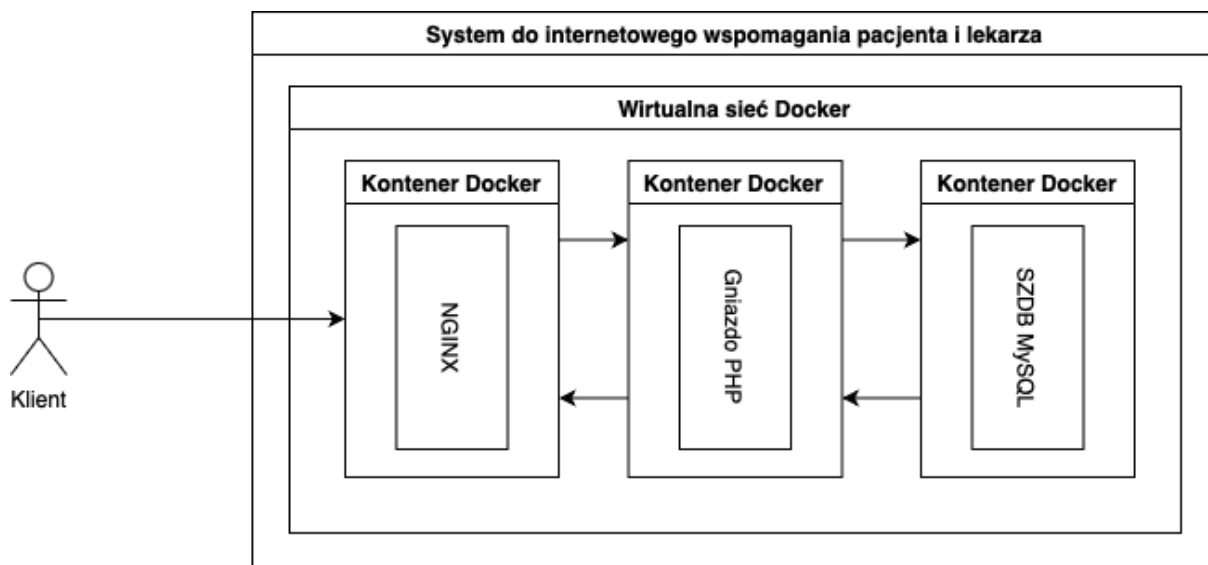


Rys. 4.1. Schemat architektury systemu do internetowego wspomagania pacjenta i lekarza w pierwszej fazie

4.2 Faza druga

Wspieranym oprogramowaniem wirtualizującym na środowisku lokalnym i w samym AWS jest Docker. W drugiej fazie migracji wykonano opisanie bazy danych, pośrednika zapytań

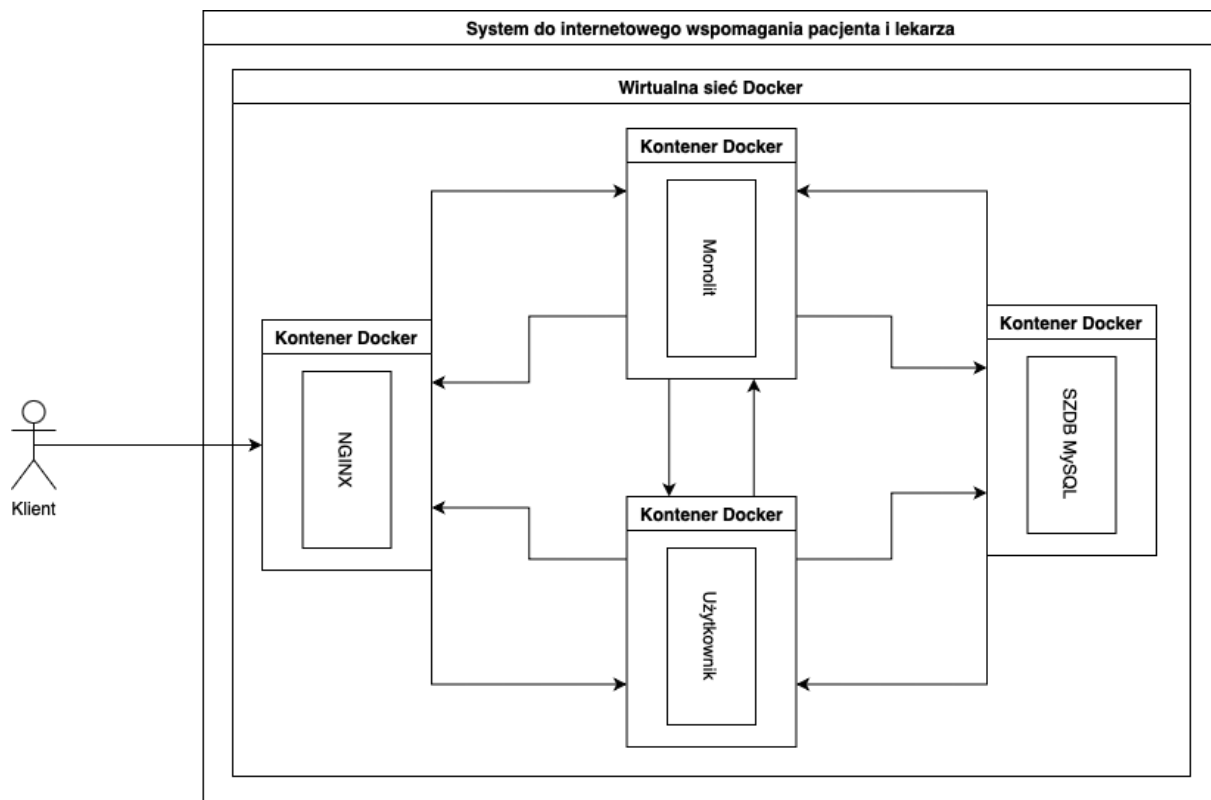
nginx i logiki biznesowej w osobnych, niezależnych kontenerach, skomunikowanych ze sobą wirtualną siecią. Omawiana zmiana widoczna na rysunku 4.2.



Rys. 4.2. Schemat architektury systemu do internetowego wspomagania pacjenta i lekarza w drugiej fazie

4.3 Faza trzecia

W fazie trzeciej migracji systemu należy zacząć od domeny użytkownika i wydelegować ją do osobnego mikroserwisu od reszty monolitu, dodatkowo w samym monolicie należy poczynić zmiany, które uzależniają wszystkie czynności należące do zarządzania kontem użytkownika generycznego przetransferować by podlegały walidacji oraz wykonaniu przez mikroserwis. Na schemacie pojawia się zmiana nazewnictwa z "Gniazdo PHP" na "Monolit", co od teraz symbolizować będzie całość logiki, ponieważ warstwa infrastruktury już nie jest istotna w tym momencie. Dodatkowo w oprogramowaniu pośredniczącym w zapytaniu między użytkownikiem a systemem - nginx należy poczynić zmianę konfiguracji, tak by ta przekazywała zapytania o użytkownika bezpośrednio do odpowiedniego kontenera z mikroserwisem w wirtualnej sieci Docker. W systemie do zarządzania relacyjną bazą danych należy utworzyć bazę danych odpowiadającą za tylko i wyłącznie dane wymagane do utrzymania w systemie przez mikroserwis użytkownika. Wyszczególniona powyżej zmiana została odzwierciedlona na rysunku 4.3.



Rys. 4.3. Schemat architektury systemu do internetowego wspomagania pacjenta i lekarza w trzeciej fazie

4.4 Faza czwarta

Faza czwarta migracji aplikacji zawiera w sobie dalszą modyfikację systemu. Początek zmian należy zawrzeć w konfiguracji oprogramowania nginx, gdzie należy przekierować cały ruch odpowiadający za obsługę użytkownika do mikroserwisu pacjenta. Dodatkowo utworzenie nowej bazy danych w systemie do zarządzania bazą danych MySQL. Niniejsza zmiana zwizualizowana została na rysunku 4.4.

4.5 Faza piąta

Kolejną fazą migracji jest część piąta, a ta będzie analogiczna do fazy czwartej. W jej skład będzie wchodzić utworzenie mikroserwisu dla pracownika recepcji oraz przekierowanie całego ruchu odpowiadającego za obsługę tej domeny do nowego mikroserwisu. Faza zostanie zakończona utworzeniem nowej bazy danych. Ta zmiana została przedstawiona na rysunku 4.5.

4.6 Faza szósta

W fazie szóstej dodano już ostatni mikroserwis odpowiadający za domeny związane z pracownikami lub użytkownikami. Utworzono zmiany w konfiguracji oprogramowania nginx oraz dodano bazę danych dla pracownika administracji. Zmiany zostały przedstawione na rysunku 4.6.

4.7 Faza siódma

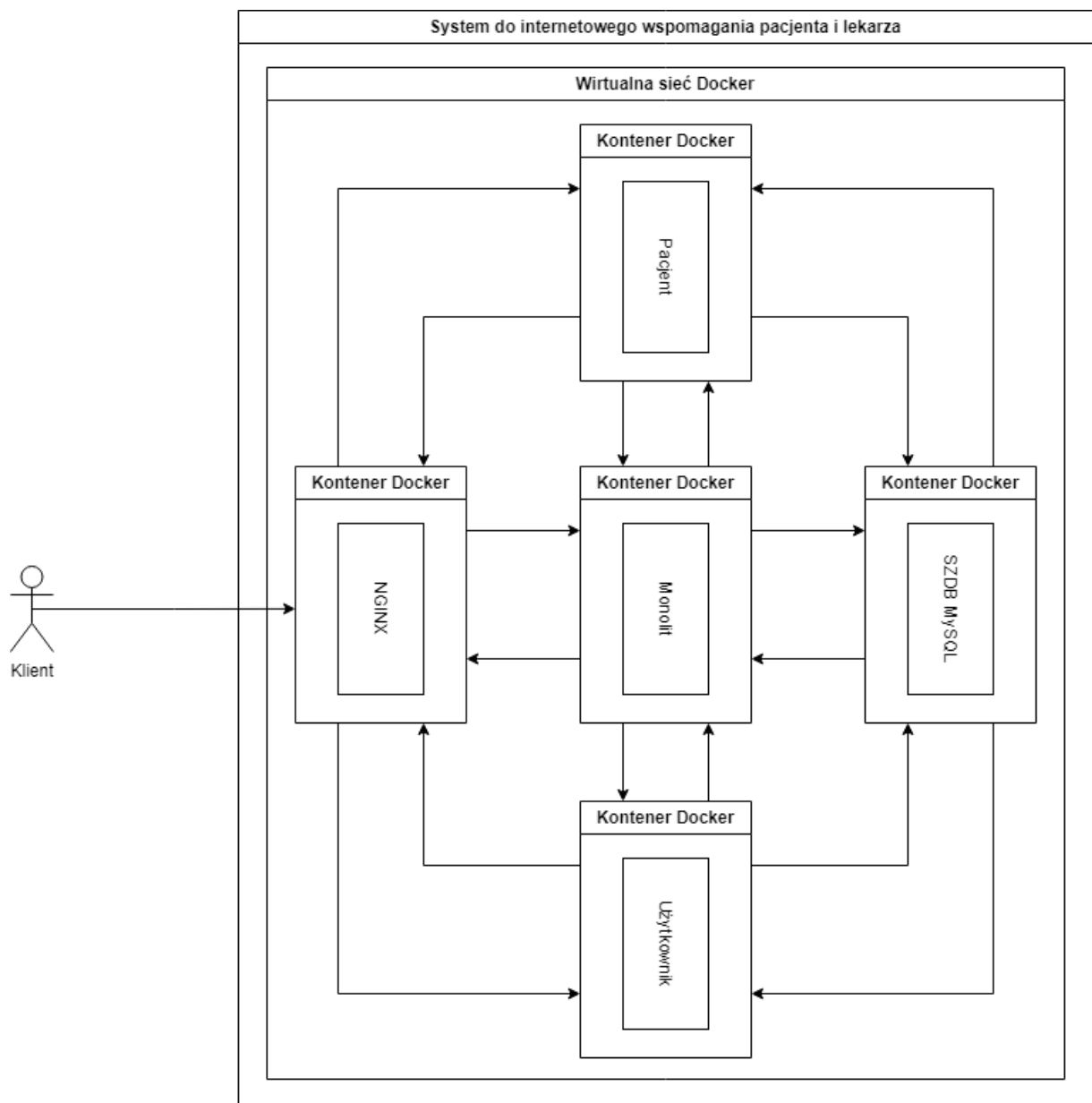
W fazie siódmej migracji dodano mikroserwis odpowiadający za obsługę wizyt. Proces technologiczny wygląda analogicznie do wcześniejszych faz. Wykonano zmianę konfiguracji oprogramowania nginx oraz utworzono nową bazę danych. Zmiany widoczne na rysunku 4.7.

4.8 Faza ósma

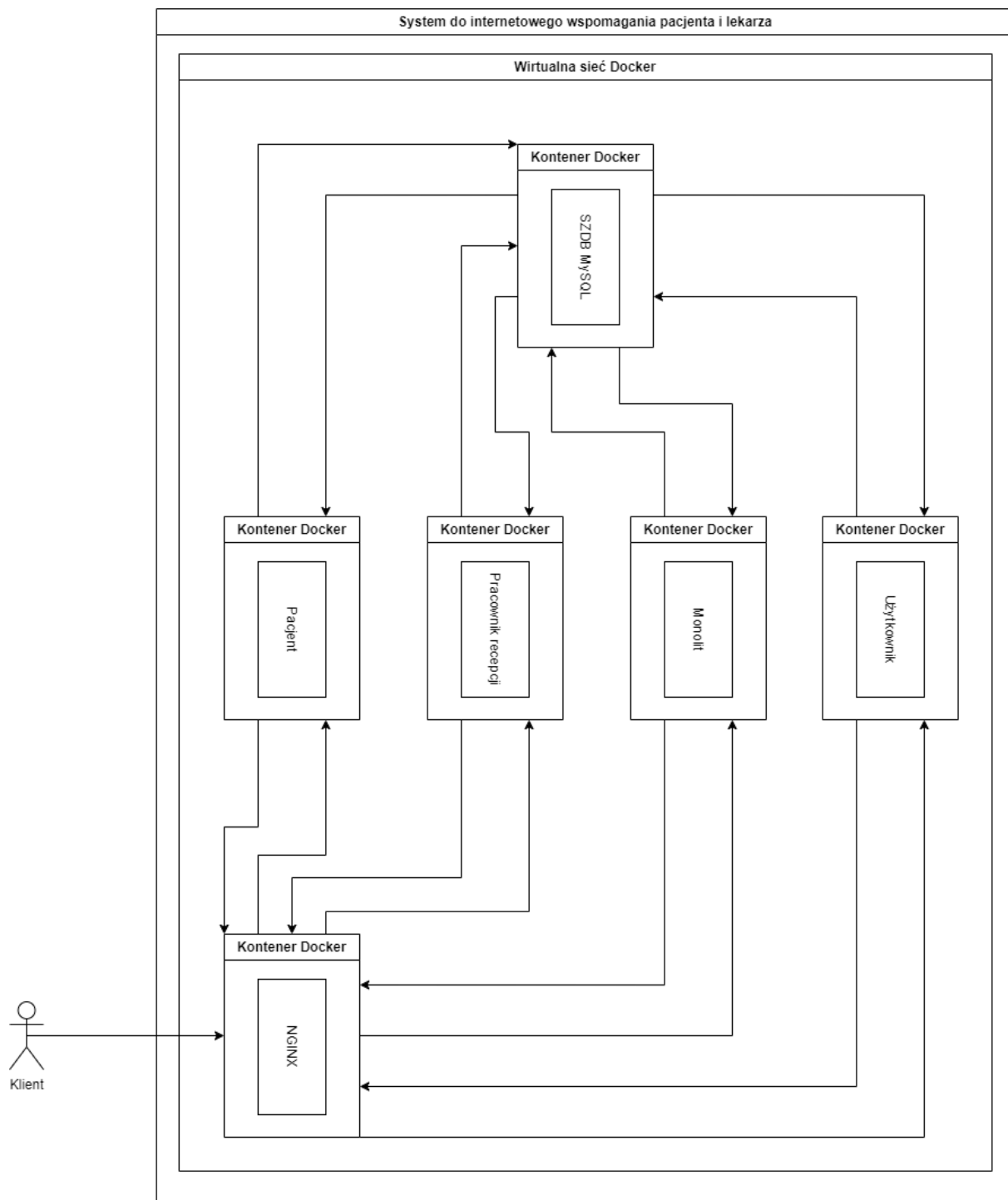
Ostatnia faza migracji to uruchomienie oprogramowania RabbitMQ, które będzie broke-rem komunikatów portu AMQP. Dodatkowo uruchomiono serwis wysyłki maili. Na schemacie nastąpiło usunięcie części systemu określonej jako monolit, a to dlatego, że cały system w obecnym stanie jest już niezależny od jednej całości. Całość przedstawiona na rysunku 4.8.

4.9 Podsumowanie przebiegu migracji

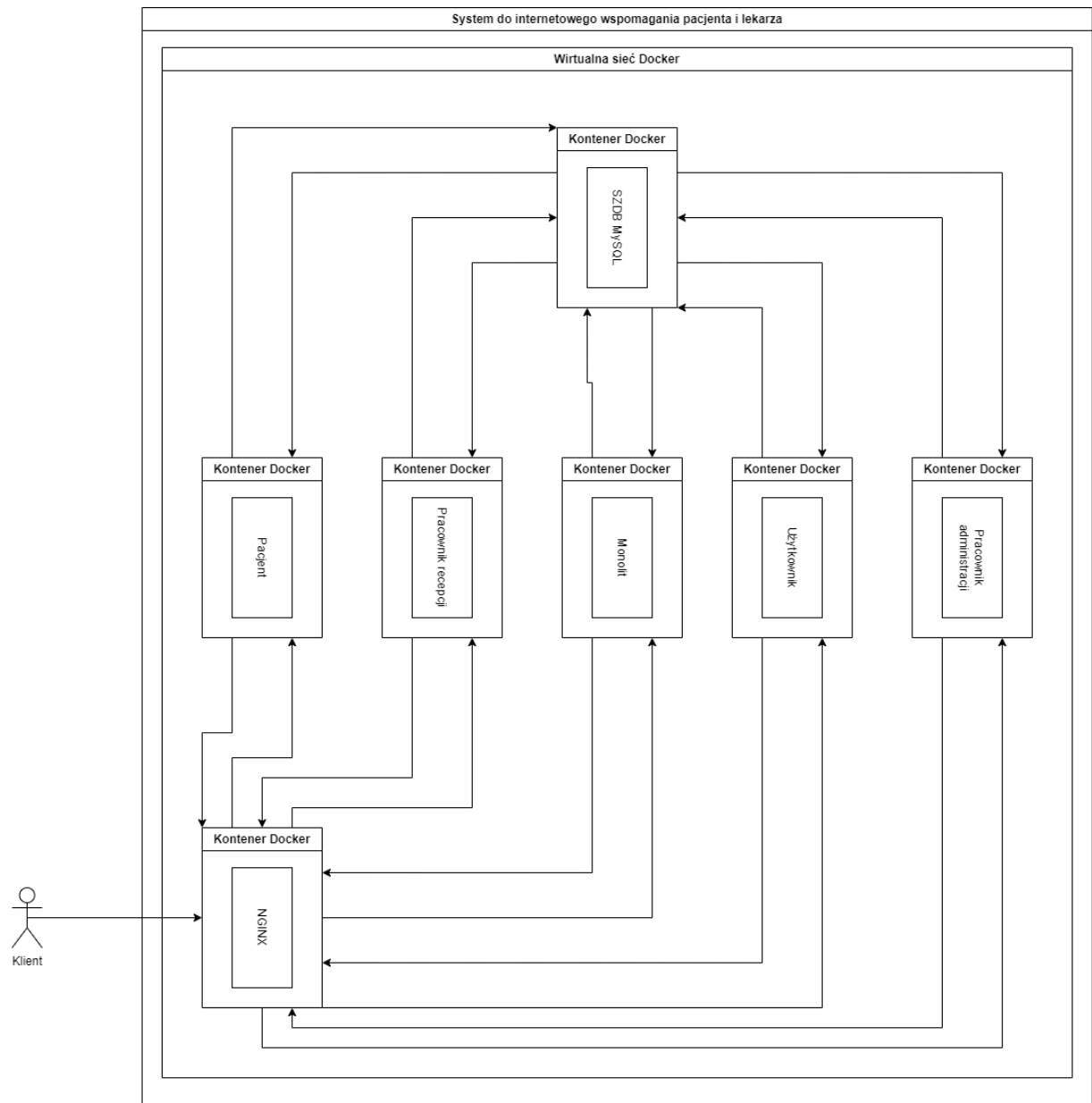
Proces stopniowego ewoluowania projektu monolitycznego w projekt mikroserwisowy dla systemu do internetowego wspomagania pacjenta i lekarza, jest procesem skomplikowanym. Czas poświęcony na dokonanie takiej migracji to w przybliżeniu 112h. Proces należał jednak do łatwiejszych przez wzgląd na niekomercyjne wykorzystanie systemu, a co za sobą wprowadza mniejsze ryzyko utraty danych lub poważnej awarii całej aplikacji.



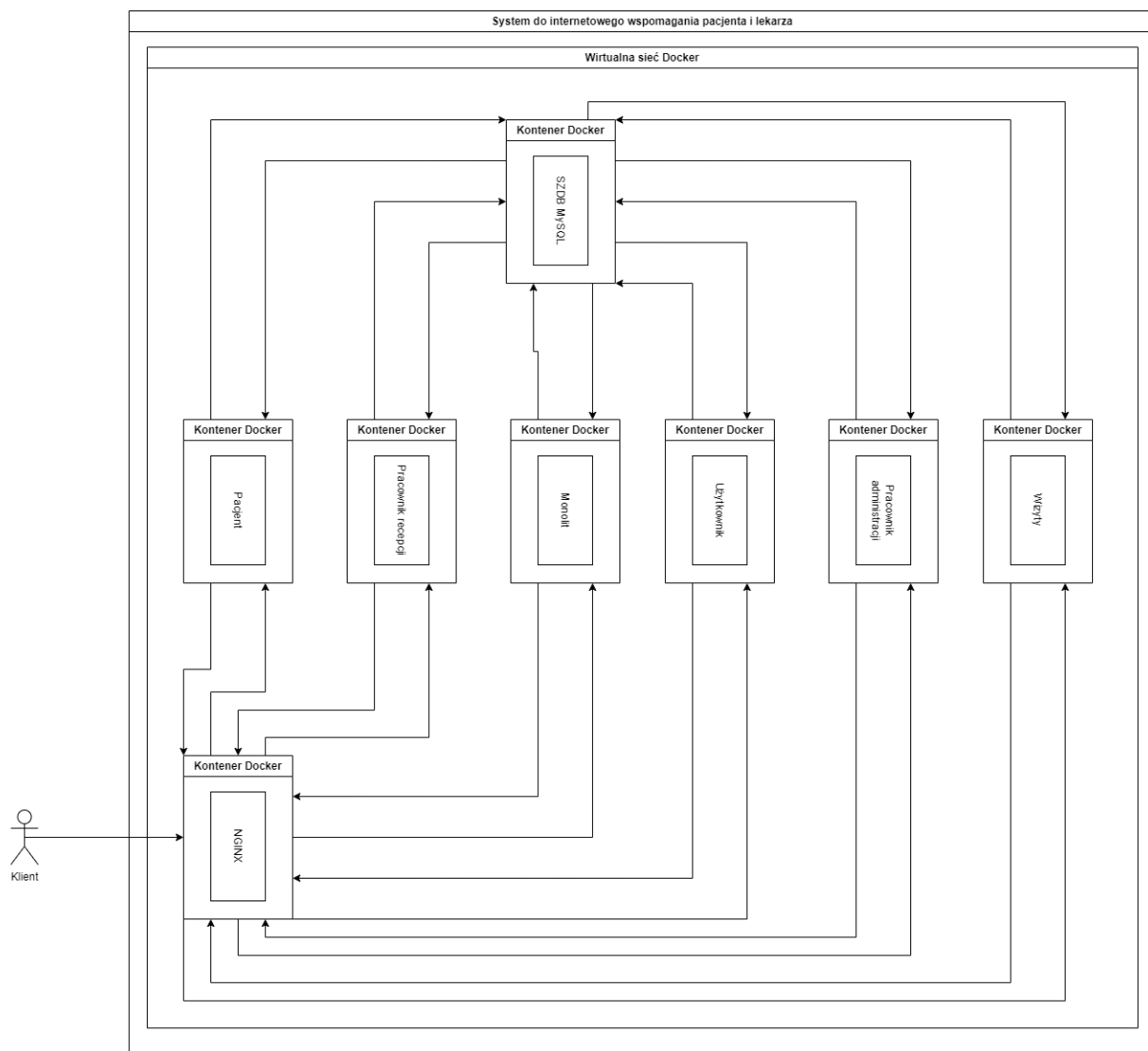
Rys. 4.4. Schemat architektury systemu do internetowego wspomagania pacjenta i lekarza w czwartej fazie



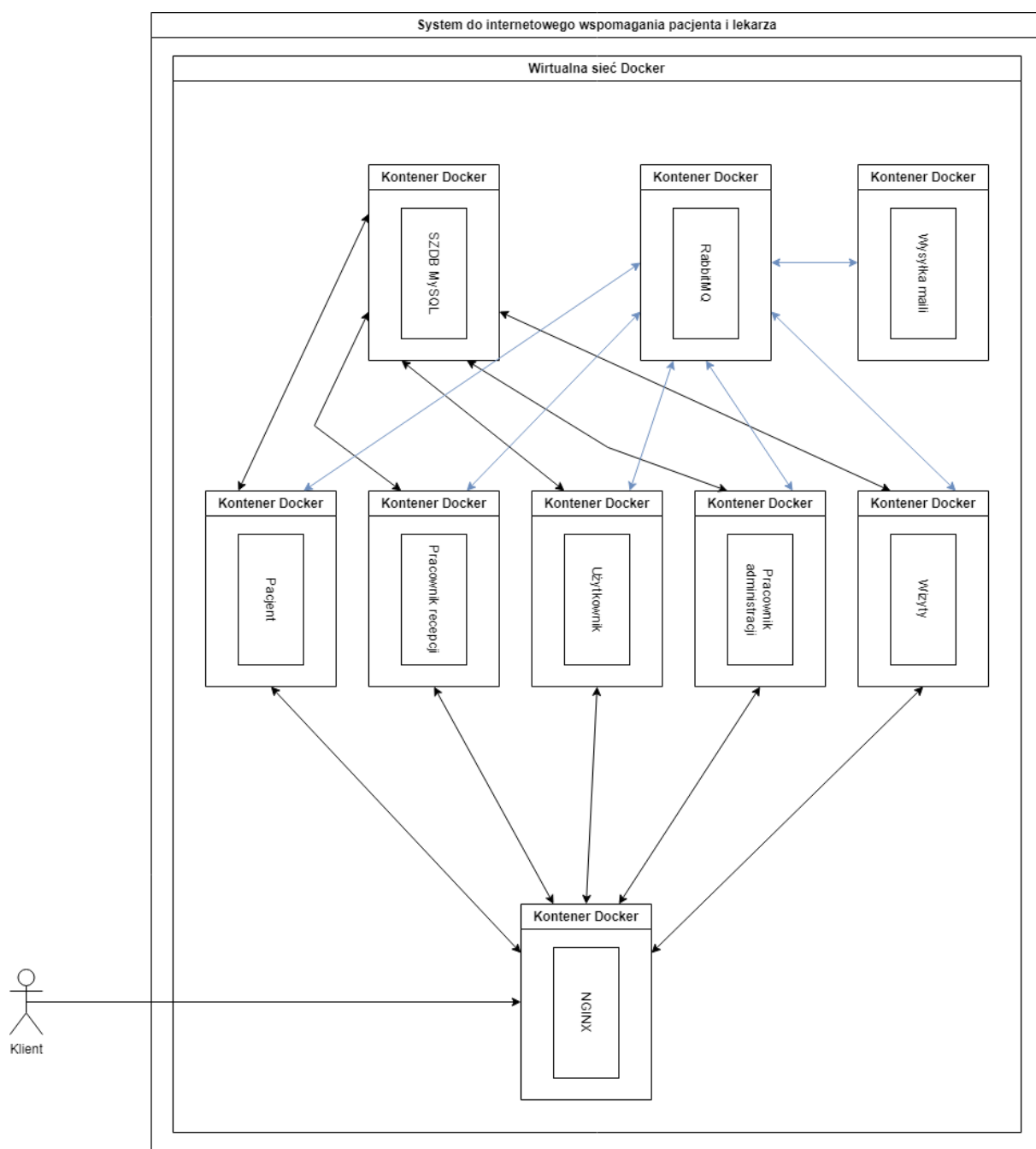
Rys. 4.5. Schemat architektury systemu do internetowego wspomagania pacjenta i lekarza w piątej fazie



Rys. 4.6. Schemat architektury systemu do internetowego wspomagania pacjenta i lekarza w szóstej fazie



Rys. 4.7. Schemat architektury systemu do internetowego wspomagania pacjenta i lekarza w siódmej fazie



Rys. 4.8. Schemat architektury systemu do internetowego wspomagania pacjenta i lekarza w ósmej fazie

5. BEZPIECZEŃSTWO

5.1 *Poziom zabezpieczeń przed migracją*

Niniejszy rozdział przedstawia stan poziomu zabezpieczeń przed migracją autora na system mikroserwisowy. Aspektem cechującym wszystkie systemy oparte o architekturę monolityczną jest fakt, że w przypadku awarii jednej usługi wewnątrz monolitu, niesie to ryzyko rozpropagowania awarii na cały system i zaburzenie jego działania. Ocena poziomu zabezpieczeń będzie krytyczna, ale złagodzona przez fakt, iż system nie działa w zakresie komercyjnym.

System do zarządzania bazą danych MySQL posiadał wyeksponowany na cały internet port, po którym potencjalny atakujący mógł przeprowadzić atak typu Brute Force w celu uzyskania dostępu do funkcji administracyjnych i potencjalnie pozyskanie danych przetrzymywanych w bazie danych. Atak typu Brute Force polega na próbie odgadnięcia hasła przez generowanie wszystkich możliwych kombinacji ciągu znaków. W teorii można złamać w ten sposób każde hasło [12]. Problem ten można rozwiązać na wiele sposobów, jednak autor tej pracy dyplomowej zastosuje ukrycie bazy danych w prywatnej sieci wewnętrznej, a dostęp do narzędzi administratorskich będzie się odbywać poprzez klucz SSH generowane przy pomocy algorytmów kryptograficznych takich jak RSA lub ECDSA.

Modernizowana aplikacja omawiana w tej pracy dyplomowej pierwotnie uruchomiona była na porcie HTTP 80, bez dodatkowych zabezpieczeń w postaci ważnego certyfikatu klucza publicznego. Certyfikat ten to informacja o kluczu publicznym podmiotu, która podpisana jest przez zaufaną stronę trzecią i jest niemożliwa do podrobienia [26]. Wynikało to z faktu, iż dla celów akademickich nie istotnym było utworzenie owego certyfikatu i zadbanie o komunikację użytkownika poprzez szyfrowanych kanał na porcie HTTP 443. Kolejnym krokiem do przeprowadzenia odpowiedniej migracji mającej na celu uwzględnienie cyberbezpieczeństwo systemu jest dokonanie zmiany na komunikację szyfrowaną.

Ostateczne problemy wynikające z pomniejszego zaniedbania autora to fakt, iż system nie otrzymywał regularnych aktualizacji zabezpieczeń. W tym celu by zwiększyć poziom bezpieczeństwa należy uruchomić procedurę aktualizacji bibliotek zewnętrznych wykorzystywanych przez framework (ang. zestaw narzędzi) Symfony oraz także przeprowadzić aktualizację wersji interpretera języka programowania PHP. Ten sam fakt dotyczy również bibliotek, z których korzystała warstwa widoku pod postacią zestawu narzędzi Vue.js.

5.2 Bezpieczeństwo architektury chmurowej

W dokumentacji narzędzia Amazon VPC podkreśla się, że dostarcza ono kluczowe rozwiązanie z punktu widzenia bezpieczeństwa każdego systemu opartego na obliczeniach chmurowych — izolację zasobów. Jest to możliwe dzięki logicznej izolacji wirtualnej sieci, którą definiuje użytkownik AWS. Narzędzie to oferuje również funkcje takie jak kreacja własnego zakresu adresów IP, tworzenie podsieci, konfiguracja tabel routingu oraz bramek internetowych, co pozwala na maksymalizację poziomu bezpieczeństwa i dostosowanie infrastruktury sieciowej do specyficznych potrzeb systemu. Wyżej wymienione funkcje udostępnione przez Amazon w narzędziu VPC zostały częściowo wykorzystane przez autora niniejszej pracy dyplomowej. Obecne rozwiązania sieciowe są znacznie bardziej dostosowane do wymagań systemu do internetowego wspomagania pacjenta i lekarza w porównaniu z pierwotną konfiguracją.

Aspekt kontroli dostępu do zasobów w chmurze jest zarządzany za pomocą narzędzia Amazon IAM. Jest to podstawowy wybór w zakresie konfiguracji kontroli dostępu, od którego zależy określanie uprawnień. Na pierwszy rzut oka IAM może wydawać się skomplikowany, jednak po zapoznaniu się z jego dokumentacją i poradnikami, ukazuje swoją prostotę oraz przewagę, jaką zyskuje użytkownik konfigurujący zasoby swojego projektu z uwzględnieniem bezpieczeństwa. IAM oferuje bardzo szerokie spektrum konfiguracji — od ogólnych ustawień po najbardziej szczegółowe, co pozwala precyzyjnie określić uprawnienia osób, które potrzebują dokonać zmian.

Jedną z zalet tego rozwiązania jest możliwość tworzenia tymczasowych haseł dostępu, co umożliwia dynamiczne uruchamianie poszczególnych części infrastruktury lub automatyzowanie procesu wdrożenia, jeśli użytkownik konfigurujący dąży do utworzenia oprogramowania w podejściu IaaS (Infrastructure as a Service).

Dodatkowo narzędzie to posiada wbudowany program do analizy dostępu oraz walidacji skonfigurowanej polityki uprawnień. Program ten zamieszcza porady, jak efektywnie zmniejszyć uprawnienia, aby nie były one nadmierne, a jedynie pozwalały na wykonanie niezbędnych zadań [3].

5.3 Bezpieczeństwo komunikacji

Naturalnym aspektem wymiany danych pomiędzy poszczególnymi komponentami systemu modernizowanego przez autora pracy jest uwzględnienie szyfrowanej komunikacji. Efekt ten uzyskuje się poprzez wykorzystanie protokołów TLS/SSL, które zapewniają poufność i in-

tegralność przesyłanych danych. TLS wykorzystuje szyfrowanie asymetryczne.

Szyfrowanie asymetryczne to rodzaj kryptografii, w którym jeden z kluczy jest kluczem publicznym [13]. Dowolny użytkownik może użyć tego klucza do zaszyfrowania wiadomości, jednak tylko posiadacz drugiego, prywatnego klucza może ją odszyfrować. W uproszczeniu, tak przebiega komunikacja z wykorzystaniem protokołu TLS/SSL.

Z perspektywy modelu OSI, będącego standardem komunikacji komputerowej zaproponowanym przez ISO, TLS odgrywa kluczową rolę w warstwie prezentacji, co pozytywnie wpływa na zabezpieczenie warstwy najwyższej — warstwy aplikacji. W przypadku systemu modernizowanego przez autora, jest to warstwa związana z protokołem HTTP.

W gamie usług AWS, narzędziem zarządzającym i dostarczającym darmowe certyfikaty do szyfrowanej komunikacji jest AWS Certificate Manager (menedżer certyfikatów AWS). Usługa ta ułatwia centralne zarządzanie certyfikatami, zarówno tymi wygenerowanymi przez Amazon, jak i dostarczonymi z zewnątrz. Dodatkowo AWS Certificate Manager bezpiecznie przechowuje certyfikaty, co zwalnia użytkownika korzystającego z usług AWS z odpowiedzialności za weryfikację potencjalnych luk w zabezpieczeniach [5].

5.4 Zarządzanie danymi

Przechowywanie danych w zmodernizowanym systemie do wspomagania pacjenta i lekarza będzie zarządzane przez usługę Amazon RDS. Usługa ta oferuje szyfrowanie danych zarówno w spoczynku, jak i podczas wymiany, co pozytywnie wpływa na poziom zabezpieczeń systemu. Szyfrowanie danych w spoczynku realizowane jest przy użyciu kluczy dostarczonych przez użytkownika. Dane są szyfrowane niezależnie od tego, czy są aktywnie używane, czy przechowywane w kopiach zapasowych bazy danych [19].

Mając na uwadze dobro użytkowników systemu, a także regulacje Unii Europejskiej, szczególną uwagę należy zwrócić na zabezpieczenie danych zgodnie z rozporządzeniem o ochronie danych osobowych (RODO). Celem tego rozporządzenia jest harmonizacja prawa w ramach państw członkowskich Unii Europejskiej oraz umożliwienie swobodnego przepływu danych osobowych [10]. Amazon potwierdza, że usługi tej firmy są zgodne z RODO, zatem po odpowiednim skonfigurowaniu systemu aplikacja uruchomiona w chmurze obliczeniowej AWS będzie również zgodna z tym rozporządzeniem [8].

5.5 Bezpieczeństwo aplikacji

W niniejszym rozdziale omówione zostaną aspekty wewnętrzne bezpieczeństwa aplikacji, między innymi wykorzystanie narzędzia Dependabot, dostarczanego przez firmę GitHub. Narzędzie to oferuje trzy główne funkcjonalności:

- alerty informujące o podatnościach występujących w aplikacji,
- automatyczne tworzenie w repozytorium kodu Pull Request (ang. prośba o dodanie zmian) z usprawnieniami mającymi na celu likwidację podatności, jeśli jest to możliwe,
- automatyczne tworzenie w repozytorium kodu próśb o dodanie zmian z aktualizacją wersji, nawet jeśli nie wykryto podatności.

Aby skorzystać z tego narzędzia, konieczne jest posiadanie repozytorium na platformie GitHub. Autor tej pracy od początku tworzenia projektu korzysta z tej platformy do przechowywania kodu. Po utworzeniu repozytorium należy przejść do jego ustawień, a następnie w zakładce „Security” otworzyć sekcję „Code security and analysis”, gdzie znajduje się przełącznik umożliwiający uruchomienie Dependabota. Po włączeniu narzędzia, Dependabot automatycznie analizuje zależności i strukturę kodu, informując o potencjalnych problemach oraz sugerując ich rozwiązania [7]. Na rysunku 5.1 przedstawiony przykład komunikatu, który dostarcza GitHub Dependabot.

Autor, prowadząc rozpoznawanie w bazie kodu aplikacji, przeprowadził zarówno analizę pasywną, jak i aktywną. Pasywny rekonesans polegał na zgłębieniu wymagań funkcjonalnych systemu, aby odświeżyć pamięć o czynnościach, które system powinien zapewniać. Dodatkowo, w ramach pasywnej analizy, autor sprawdził domenę sdiwpil.com w rejestrze DNS i zweryfikował informacje oraz jakość zabezpieczeń.

Jednym z mankamentów, który został zidentyfikowany po wprowadzonych przez autora modyfikacjach, jest fakt, że system posiada jednego dostawcę DNS, którym jest Amazon. W celu uniezależnienia systemu od ewentualnej awarii AWS, autor rozważa możliwość rozlokowania wpisów DNS u różnych dostawców. Mimo to, autor wierzy, że Amazon dysponuje szeregiem zabezpieczeń mających na celu zapewnienie redundancji swoich zasobów, co daje pewność, że klienci mogą czuć się bezpiecznie, powierzając każdy detal swojego systemu w rękach specjalistów AWS.

Dalsza część analizy obejmowała rekonesans aktywny, który polegał na zidentyfikowaniu punktów końcowych API, wersji oprogramowania serwera, potencjalnych informacji o

modułach PHP, komponentów JavaScript oraz innych szczegółów związanych z architekturą aplikacji.

Aplikacja budowana i modernizowana przez autora nie posiada dokumentacji API, ponieważ jej API jest skierowane na współpracę z warstwą widoku, a nie na udostępnianie usług zewnętrznym klientom. Weryfikacja działania API może być zatem przeprowadzona poprzez symulację komunikacji między warstwą aplikacji a serwerem. Informacje, które można uzyskać bezpośrednio po uruchomieniu strony, obejmują punkty końcowe API związane z rejestracją i logowaniem użytkowników, zasoby statyczne, takie jak zdjęcia, pliki widoku, pliki HTML, CSS i JavaScript. Wymienione punkty końcowe są otwartymi i dostępnymi ścieżkami, które nie wymagają uwierzytelnienia użytkownika. Po zalogowaniu się na konta testowe dla poszczególnych aktorów systemu, ukazuje się pełne spektrum dostępnych ścieżek w systemie, które zostały zamieszczone w tabeli 5.1.

Tabela 5.1. Opis punktów końcowych systemu do internetowego wspomaganie pacjenta i lekarza

Punkt końcowy API	Dozwolona metoda HTTP	Krótki opis punktu	Aktor posiadający dostęp
/login/	POST	Przesłanie danych logowania i pobieranie tokena uwierzytelniającego.	użytkownik niezalogowany
/register/	POST	Przesłanie danych do utworzenia konta i pobieranie tokena uwierzytelniającego.	użytkownik niezalogowany
/patient/settings/get/	GET	Pobranie ustawień konta pacjenta.	pacjent
/patient/settings/update/	PUT i PATCH	Zapisanie ustawień konta pacjenta.	pacjent
/appointments/	GET	Pobranie listy umówionych wizyt.	pacjent, lekarz, pracownik recepcji, pracownik administracji

Punkt końcowy API	Dozwolona metoda HTTP	Krótki opis punktu	Aktor posiadający dostęp
/appointments/ create/	POST	Utworzenie nowej wizyty.	pacjent, lekarz, pracownik recepcji, pracownik administracji
/appointments/ get/id	GET	Pobranie szczegółów jednej wizyty.	pacjent, lekarz, pracownik recepcji, pracownik administracji
/appointments/ update/id	PUT i PATCH	Aktualizacja danych wizyty.	pacjent, lekarz, pracownik recepcji, pracownik administracji
/appointments/ delete/id	DELETE	Usunięcie danych wizyty.	pacjent, lekarz, pracownik recepcji, pracownik administracji
/doctor/ settings/get/	GET	Pobranie ustawień konta lekarza.	lekarz
/doctor/ settings/update/	PUT i PATCH	Zapisanie ustawień konta lekarza.	lekarz
/receptionist/ settings/get/	GET	Pobranie ustawień konta pracownika recepcji.	pracownik recepcji
/receptionist/ settings/update/	PUT i PATCH	Zapisanie ustawień konta pracownika recepcji.	pracownik recepcji
/admin/ settings/get/	GET	Pobranie ustawień konta pracownika administracji.	pracownik administracji
/admin/ settings/update/	PUT i PATCH	Zapisanie ustawień konta pracownika administracji.	pracownik administracji

Autor, kontynuując aktywny rekonesans systemu do internetowego wspomagania pacjenta i lekarza, uzyskał informacje dotyczące serwera pośredniczącego w zapytaniach oraz wersji protokołu HTTP używanej w komunikacji. W celu zdobycia tych danych, autor skorzystał z wbudowanego narzędzia w programie PHPStorm, które umożliwia wykonywanie zapytań HTTP za pośrednictwem swojego interfejsu. Odpowiedź na pytanie, jaki serwer jest pośrednikiem, została przedstawiona na rysunku 5.2, a jest to nginx.

Podążając za dobrymi praktykami, autor ukrył w konfiguracji nginx informacje takie jak wersja oprogramowania, aby potencjalny atakujący nie miał ułatwionego zadania poprzez wyszukiwanie luk zabezpieczeń specyficznych dla danej wersji. Dodatkowo został ukryty nagłówek X-Powered-By informujący o wersji PHP 8.3, aby uniemożliwić atakującemu celowanie w konkretne podatności związane z interpreterem PHP lub samym językiem programowania.

Kolejnym krokiem aktywnego rekonesansu będzie analiza plików źródłowych aplikacji. Pierwszym analizowanym plikiem będzie plik konfiguracyjny nginx.

Plik ten składa się z kilku kluczowych bloków. Jednak przed analizą samych bloków warto zwrócić uwagę, że konfiguracja zaczyna się od ustawienia `worker_processes 4;`, co oznacza, że nginx uruchamia cztery procesy wykonawcze. Cały plik konfiguracyjny został przedstawiony na rysunku 5.3.

Pierwszym blokiem w pliku konfiguracyjnym nginx jest blok `events` (ang. zdarzenia), który zawiera ustawienie `worker_connections 1024;`. Ustawienie to określa, ile maksymalnie połączeń może obsłużyć pojedynczy proces [6].

Sekcja `http`, z punktu widzenia cyberbezpieczeństwa systemu, zawiera ustawienia takie jak `limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;` [11], które definiują strefę ograniczenia liczby żądań. To ustawienie wykorzystuje adres IP klienta jako klucz (`$binary_remote_addr`) i tworzy strefę o nazwie „one” z rozmiarem pamięci ustawionym na 10 MB, co pozwala na 1 żądanie na sekundę z jednego adresu IP. Jest to szczególnie istotne z punktu widzenia ochrony przed potencjalnym atakiem DDoS (Distributed Denial of Service, rozproszona odmowa usługi), który ma na celu zablokowanie maszyny utrzymującej system w sieci [15]. Konfiguracja ta odrzuca nadmierną ilość zapytań z tego samego adresu IP, jeśli przekraczają one ustalone limity, co pomaga chronić system przed przeciążeniem i utratą dostępności.

Dalsze sekcje pliku konfiguracyjnego nginx nie zawierają specjalnych ustawień, które znacząco wpłynęłyby na bezpieczeństwo zewnętrzne. Niemniej jednak warto wspomnieć o po-

tencjalnych usprawnieniach, które mogą zostać wprowadzone, takich jak:

- dodanie większej liczby serwerów obsługujących dany upstream,
- rozszerzona konfiguracja logów, która może być rozbudowana o specyficzne ustawienia dla bloków location (lokalizacja),
- wprowadzenie wewnętrznej komunikacji za pośrednictwem portu HTTPS.

Kolejnym obszarem analizy są zależności używane w kodzie źródłowym systemu. Dla celów analitycznych tej pracy zostanie przeanalizowany jeden z plików `composer.json`, który określa zależności w poszczególnych mikroservisach. Taka analiza jest uzasadniona, ponieważ system opiera się na tym samym zestawie narzędzi w każdym mikroservisie. Mikroservisem, którego plik `composer.json` zostanie poddany audytowi, jest mikroservis autoryzacji.

Sekcją szczególnie wrażliwą z punktu widzenia bezpieczeństwa aplikacji jest sekcja `require`, która zawiera obiekt składający się z par klucz-wartość, gdzie klucz to nazwa zewnętrznej biblioteki, a wartość to wersja wykorzystywanej biblioteki. Cała sekcja jest widoczna na rysunku 5.4. W dniu 23 sierpnia 2024 roku, po uruchomieniu narzędzia `audit`, wbudowanego w program `Composer`, można sprawdzić, czy są dostępne sugestie dotyczące zabezpieczeń. W przypadku zmodernizowanego przez autora projektu nie występują żadne zalecenia, co wskazuje na wysoki poziom bezpieczeństwa projektu w tym zakresie, a rezultat przedstawiony jest na rysunku 5.5.

Aplikacja do internetowego wspomaganie pacjenta i lekarza wykorzystuje `Doctrine` jako pośredniczący mechanizm do połączenia z bazą danych. `Doctrine` zapewnia liczne ułatwienia, takie jak mapowanie obiektów na encje, mechanizmy usprawniające efektywność zapytań wykonywanych do bazy danych oraz wiele innych funkcji.

Jednym z kluczowych aspektów bezpieczeństwa, które `Doctrine` zapewnia, jest ochrona przed atakami typu `SQL injection`. Dzięki mechanizmom wiązania dynamicznych parametrów zapytań, `Doctrine` znacząco minimalizuje ryzyko tego rodzaju ataków [9]. Tylko świadome zrezygnowanie z korzystania z jego mechanizmów mogłoby otworzyć drogę do przeprowadzenia ataku, jednak jest to proces trudny do wykonania, gdyż wymagałoby to celowego wyłączenia wszystkich zabezpieczeń oferowanych przez `Doctrine`. Dlatego autor pracy ocenia poziom odporności na ataki `SQL injection` jako wysoki.

Sama dokumentacja `Doctrine` przedstawia przykłady, w jaki sposób unikać niebezpiecznych praktyk, które mogłyby prowadzić do tego rodzaju ataków [14]. Autor podkreśla, że w

zmodernizowanym systemie do wspomagania pacjenta i lekarza, jak również w jego pierwotnej wersji, nie stosowano podejścia, które mogłoby narażać system na atak typu SQL injection.

5.6 Podsumowanie

Bezpieczeństwo każdego systemu działającego w internecie w dużej mierze zależy od uwagi poświęconej podczas jego budowy oraz świadomości inżyniera implementującego konkretną architekturę. Niemniej jednak, obecnie dostępne na rynku rozwiązania i frameworki skutecznie chronią mniej doświadczonych użytkowników przed podstawowymi formami podatności. Tylko świadome odrzucenie tych mechanizmów może narażić system na ryzyko.

Oczywiście, powyższe stwierdzenie nie wyklucza możliwości odkrycia luk w innych obszarach, niż te omówione w rozdziale. Autor pracy wyraża chęć dalszego pogłębiania wiedzy zdobytej podczas analizy przeprowadzonej na potrzeby niniejszej pracy dyplomowej, mając świadomość, że tylko ciągły rozwój w tym zakresie oraz skuteczne wykorzystanie dostępnych narzędzi są kluczem do skutecznej ochrony systemu.



GitHub security alert digest

bartosz-szymanski-dev's repository security updates from the week of **Aug 13 - Aug 20**



bartosz-szymanski-dev's personal account

⚠️ bartosz-szymanski-dev / opda

Known security vulnerabilities detected

Dependency	Version	Upgrade to
symfony/http-kernel	>= 5.2.0	~> 5.3.12
	< 5.3.12	

Defined in
`composer.lock`

Vulnerabilities
CVE-2021-41267 Moderate severity
CVE-2022-24894 Moderate severity

Dependency	Version	Upgrade to
symfony/security-bundle	>= 5.3.0	~> 5.3.12
	< 5.3.12	

Rys. 5.1. Zrzut ekranu przedstawiający przykładowy komunikat dotyczący podatności dla wersji systemu do wspomagania pacjenta i lekarza przed migracją

```
POST https://sdiwpil.com/api/customer/register

HTTP/1.1 400 Bad Request
Server: nginx
Date: Thu, 22 Aug 2024 18:48:59 GMT
Content-Type: application/json
Content-Length: 196
Connection: keep-alive
Cache-Control: no-cache, private
X-Robots-Tag: noindex

{
  "errors": [
    {
      "message": "Ten formularz nie powinien zawierać dodatkowych pól.",
      "field": "user"
    },
    {
      "message": "Ta wartość nie powinna być pusta.",
      "field": "user_plainPassword"
    }
  ]
}
Response file saved.
> 2024-08-22T204859.400.json

Response code: 400 (Bad Request); Time: 3165ms (3 s 165 ms); Content length: 171 bytes (171 B)
```

Rys. 5.2. Zrzut ekranu przedstawiający przykładową odpowiedź od jednego z mikroservisów systemu do internetowego wspomaganie pacjenta i lekarza

```

worker_processes 4;

events {
    worker_connections 1024;
}

http {
    limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;

    upstream customer_service {
        server customer-service:8000;
    }

    upstream doctor_service {
        server doctor-service:8000;
    }

    upstream appointment_service {
        server appointment-service:8000;
    }

    upstream patient_service {
        server patient-service:8000;
    }

    server {
        listen 80;
        charset utf-8;
        access_log /var/log/nginx/access_log.log;
        error_log /var/log/nginx/error_log.log;
        server_tokens off;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_hide_header X-Powered-By;
        proxy_cache_bypass $http_upgrade;

        location = / {
            return 404;
        }

        location ~ ^/api/customer {
            rewrite ^/api/customer/(.*) /$1 break;
            proxy_pass http://customer_service;
        }

        location ~ ^/api/doctor {
            rewrite ^/api/doctor/(.*) /$1 break;
            proxy_pass http://doctor_service;
        }

        location ~ ^/api/appointment {
            rewrite ^/api/appointment/(.*) /$1 break;
            proxy_pass http://appointment_service;
        }

        location ~ ^/api/patient {
            rewrite ^/api/patient/(.*) /$1 break;
            proxy_pass http://patient_service;
        }

        location ~ ^/api/receptionist {
            rewrite ^/api/receptionist/(.*) /$1 break;
            proxy_pass http://receptionist_service;
        }

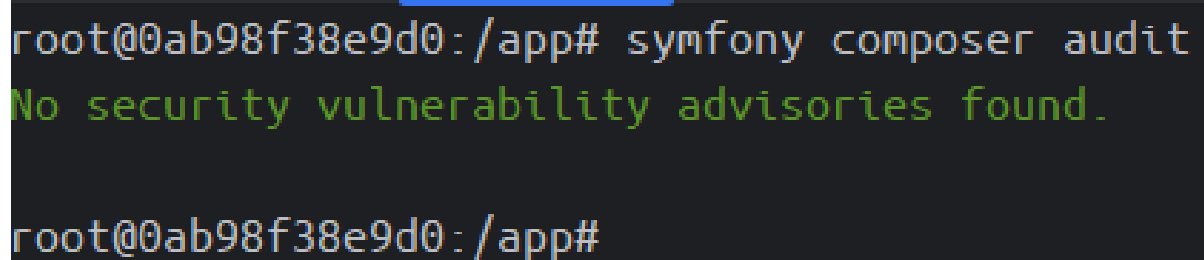
        location ~ ^/api/admin {
            rewrite ^/api/admin/(.*) /$1 break;
            proxy_pass http://admin_service;
        }
    }
}

```

Rys. 5.3. Zrzut ekranu przedstawiający konfigurację narzędzia nginx

```
"require": {
    "php": ">=8.2",
    "ext-ctype": "*",
    "ext-iconv": "*",
    "doctrine/dbal": "^3",
    "doctrine/doctrine-bundle": "^2.12",
    "doctrine/doctrine-migrations-bundle": "^3.3",
    "doctrine/orm": "^3.1",
    "friendsofsymfony/rest-bundle": "^3.7",
    "jms/serializer-bundle": "^5.4",
    "lexik/jwt-authentication-bundle": "*",
    "sentry/sentry-symfony": "^5.0",
    "symfony/amqp-messenger": "7.1.*",
    "symfony/console": "7.1.*",
    "symfony/dotenv": "7.1.*",
    "symfony/flex": "^2",
    "symfony/form": "7.1.*",
    "symfony/framework-bundle": "7.1.*",
    "symfony/messenger": "7.1.*",
    "symfony/monolog-bundle": "^3.10",
    "symfony/runtime": "7.1.*",
    "symfony/translation": "7.1.*",
    "symfony/validator": "7.1.*",
    "symfony/yaml": "7.1.*"
},
```

Rys. 5.4. Zrzut ekranu przedstawiający sekcję require z pliku composer.json mikroserwisu autoryzacji



```
root@0ab98f38e9d0:/app# symfony composer audit
No security vulnerability advisories found.

root@0ab98f38e9d0:/app#
```

Rys. 5.5. Zrzut ekranu przedstawiający wynik audytu zależności w pliku composer.json mikroserwisu autoryzacji

Bibliografia

- [1] Amazon api gateway features | api management | amazon web services. <https://aws.amazon.com/api-gateway/features/>. Data dostępu: 3 Sierpień 2024.
- [2] Amazon mq features | managed message broker service | amazon web services. <https://aws.amazon.com/amazon-mq/features/>. Data dostępu: 1 Sierpień 2024.
- [3] Amazon mq features | managed message broker service | amazon web services. <https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>. Data dostępu: 1 Sierpień 2024.
- [4] Apm tool - amazon cloudwatch - aws. <https://aws.amazon.com/cloudwatch/>. Data dostępu: 1 Sierpień 2024.
- [5] Aws certificate manager features - amazon web services (aws). <https://aws.amazon.com/certificate-manager/features/?nc=sn&loc=2>. Data dostępu: 3 Sierpień 2024.
- [6] Core functionality. https://nginx.org/en/docs/nginx_core_module.html#worker_processes. Data dostępu: 25 Sierpień 2024.
- [7] Dependabot quickstart guide - github docs. <https://docs.github.com/en/code-security/getting-started/dependabot-quickstart-guide#enabling-dependabot-for-your-repository>. Data dostępu: 24 Sierpień 2024.
- [8] Gdpr - amazon web services (aws). <https://aws.amazon.com/compliance/gdpr-center/>. Data dostępu: 24 Sierpień 2024.
- [9] Getting started with doctrine - doctrine object relational mapper (orm). <https://www.doctrine-project.org/projects/doctrine-orm/en/current/tutorials/getting-started.html#what-is-doctrine>. Data dostępu: 25 Sierpień 2024.
- [10] Informacje ogólne - giodo. <https://archiwum.giodo.gov.pl/1520143/j/pl>. Data dostępu: 24 Sierpień 2024.
- [11] Module ngx_http_limit_req_module. https://nginx.org/en/docs/http/ngx_http_limit_req_module.html. Data dostępu: 25 Sierpień 2024.

- [12] (nie)bezpieczny kod – brute force | lukasz-socha.pl - blog programisty i web developera. <https://web.archive.org/web/20220705122912/https://lukasz-socha.pl/php/niebezpieczny-kod-brute-force/>. Data dostępu: 14 Sierpień 2024.
- [13] Post-quantum cryptography. <https://www.nature.com/articles/nature23461>. Data dostępu: 23 Sierpień 2024.
- [14] Security - doctrine object relational mapper (orm). <https://www.doctrine-project.org/projects/doctrine-orm/en/current/reference/security.html>. Data dostępu: 25 Sierpień 2024.
- [15] Understanding denial-of-service attacks | cisa. <https://www.cisa.gov/news-events/news/understanding-denial-service-attacks>. Data dostępu: 25 Sierpień 2024.
- [16] Welcome - amazon route 53. <https://docs.aws.amazon.com/Route53/latest/APIReference/Welcome.html>. Data dostępu: 3 Sierpień 2024.
- [17] Welcome - elastic load balancing. <https://docs.aws.amazon.com/elasticloadbalancing/latest/APIReference/Welcome.html>. Data dostępu: 3 Sierpień 2024.
- [18] What is amazon elastic container service? - amazon elastic container service. <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html#welcome-terminology>. Data dostępu: 1 Sierpień 2024.
- [19] What is amazon relational database service (amazon rds)? - amazon relational database service. <https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html>. Data dostępu: 1 Sierpień 2024.
- [20] What is amazon s3? - amazon simple storage service. <https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html>. Data dostępu: 1 Sierpień 2024.
- [21] What is amazon vpc? - amazon virtual private cloud. <https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html>. Data dostępu: 1 Sierpień 2024.
- [22] Cloud computing market. 2023. Data dostępu: 20 Styczeń 2024.

- [23] L. da Silva and P. Vilain. Execution and code reuse between test classes. pages 99–106, Los Alamitos, CA, USA, Czerwiec 2016. IEEE Computer Society. Data dostępu: 11 Kwiecień 2024.
- [24] Scott Millett and Nick Tune. *Patterns, Principles, and Practices of Domain-Driven Design*. 2015. Data dostępu: 11 Sierpień 2024.
- [25] Sam Newman. *Monolith to Microservices*. 2019. Data dostępu: 11 Sierpień 2024.
- [26] Polski Komitet Normalizacyjny. *PN-I-02000. Technika informatyczna. Zabezpieczenia w systemach informatycznych. Terminologia*. 2002. Data dostępu: 14 Sierpień 2024.
- [27] Fabien Potencier. *Symfony 5: The Fast Track*. 2020. Data dostępu: 6 Kwiecień 2024.
- [28] M’hamed Rahmouni, Chaymae Talbi, and Soumia Ziti. Model-driven architecture; generating models from symfony. 2023. Data dostępu: 6 Kwiecień 2024.
- [29] Rhuan Rocha, Paulo Alberto Simoes, and Joao Carlos Purificação. *Java EE 8 Design Patterns and Best Practices*. 2018. Data dostępu: 1 Sierpień 2024.

Spis rysunków

1.1	Wykres przedstawiający prognozę wzrostu udziału rynkowego technologii chmurowej	5
2.1	Schemat metamodelu Symfony [28]	8
2.2	Zrzut ekranu przedstawiający przykładowy formularz w Symfony	13
2.3	Zrzut ekranu przedstawiający przykładowe repozytorium w Symfony	14
2.4	Zrzut ekranu przedstawiający przykładowe walidator w Symfony	15
2.5	Zrzut ekranu przedstawiający przykłady szablon Twig	16
2.6	Zrzut ekranu przedstawiający przykładowy szablon dziedziczący po szablonie bazowym	17
3.1	Zakres domenowy systemu do internetowego wspomagania pacjenta i lekarza .	24
3.2	Wysyłka wiadomości jest logicznie połączona dla tego modelu domeny, więc jej dalsza ekstrakcja może być trudna	25

4.1	Schemat architektury systemu do internetowego wspomaganie pacjenta i lekarza w pierwszej fazie	26
4.2	Schemat architektury systemu do internetowego wspomaganie pacjenta i lekarza w drugiej fazie	27
4.3	Schemat architektury systemu do internetowego wspomaganie pacjenta i lekarza w trzeciej fazie	28
4.4	Schemat architektury systemu do internetowego wspomaganie pacjenta i lekarza w czwartej fazie	30
4.5	Schemat architektury systemu do internetowego wspomaganie pacjenta i lekarza w piątej fazie	31
4.6	Schemat architektury systemu do internetowego wspomaganie pacjenta i lekarza w szóstej fazie	32
4.7	Schemat architektury systemu do internetowego wspomaganie pacjenta i lekarza w siódmej fazie	33
4.8	Schemat architektury systemu do internetowego wspomaganie pacjenta i lekarza w ósmej fazie	34
5.1	Zrzut ekranu przedstawiający przykładowy komunikat dotyczący podatności dla wersji systemu do wspomaganie pacjenta i lekarza przed migracją	44
5.2	Zrzut ekranu przedstawiający przykładową odpowiedź od jednego z mikroserwisów systemu do internetowego wspomaganie pacjenta i lekarza	45
5.3	Zrzut ekranu przedstawiający konfigurację narzędzia nginx	46
5.4	Zrzut ekranu przedstawiający sekcję require z pliku composer.json mikroserwisu autoryzacji	47
5.5	Zrzut ekranu przedstawiający wynik audytu zależności w pliku composer.json mikroserwisu autoryzacji	48

Spis tabel

5.1	Opis punktów końcowych systemu do internetowego wspomaganie pacjenta i lekarza	39
-----	--	----

Wykaz listingów

2.1	Przykładowa fabryka danych testowych w Symfony	8
-----	--	---

2.2	Formularz danych rejestracji	10
-----	--	----