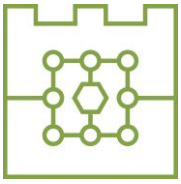
Politechnika Krakowska
im. Tadeusza Kościuszki

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 Barbara Borowik

Kraków, 2024

Spis treści

1. WSTĘP

1.1 Cel pracy

Celem niniejszej pracy dyplomowej jest zbadanie procesu migracji systemu monolitycznego do architektury mikroserwisowej 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.

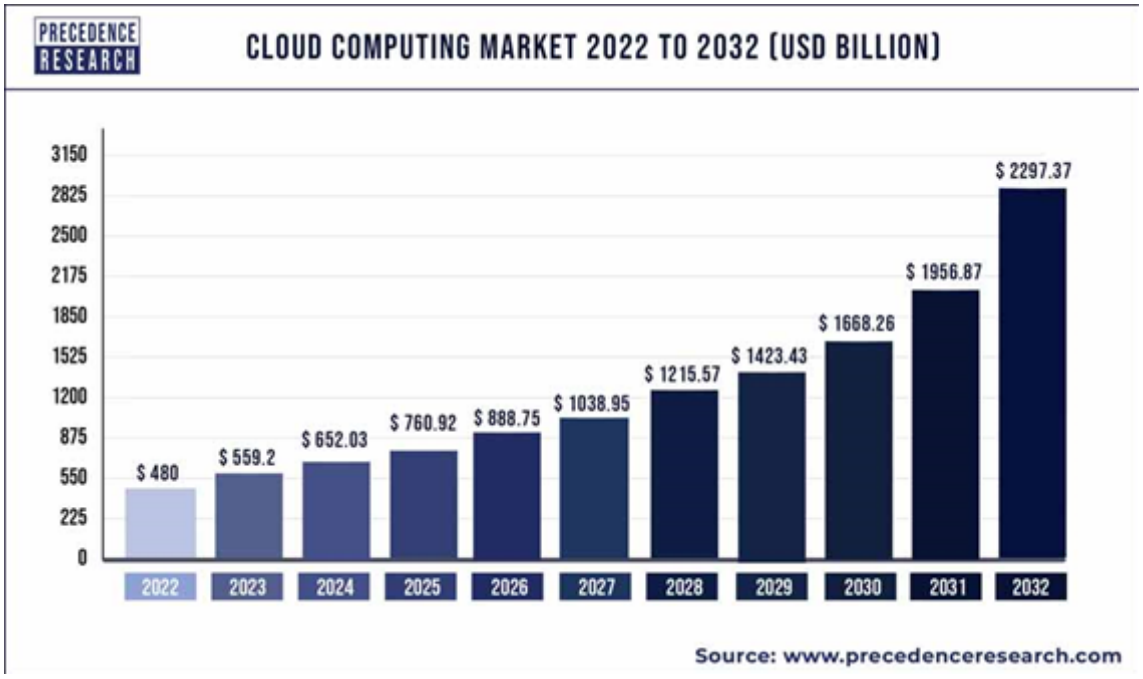
Analiza procesu migracji oraz rozwinięcia systemu zabezpieczeń systemu ma wykazać jak dobrze usługi chmurowe ochraniają swoich usługobiorców i ich oprogramowania przed popularnymi atakami hakerskimi oraz jak duży wpływ na bezpieczeństwo systemu ma jego architektura. Narzędzia używane przez autor pracy to przede wszystkim oprogramowanie wirtualizujące systemy operacyjne Docker, język programowania PHP, język programowania Java Script, 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 mikroserwisowej w chmurze AWS, ocena zagadnień bezpieczeństwa podczas migracji, implementacja i wdrażanie mikroserwisów 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 moty-

watorem 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 motywatorem jest również fakt, iż technologia chmurowa każdego roku zdobywa coraz większy udział na rynku usług hostingowych, a także staje się dzięki wzrastającej konkurencyjności przystępniejsza 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 [17]. 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 przechowy-

wanych 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 ustystematyzowanie 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 dostępów, 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:

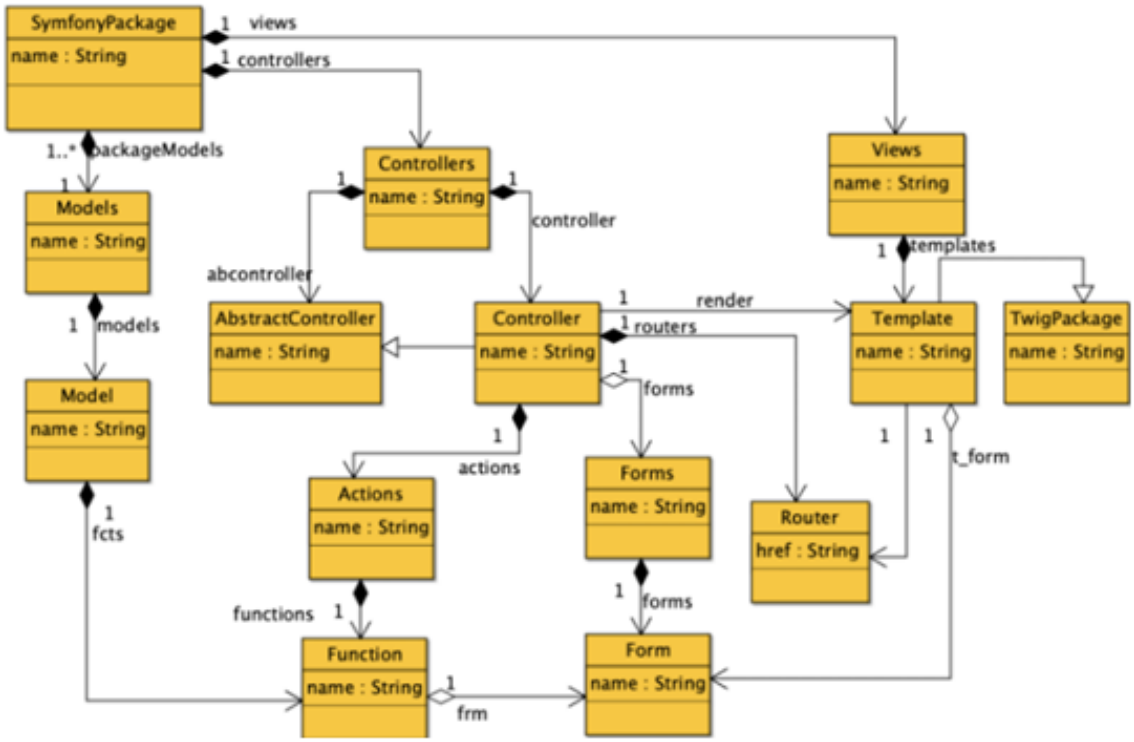
- baza danych oprata o silnik MySQL,
- trzon logiki biznesowej oparty o zestaw narzędzi Symfony w wersji 5.4,
- warstwa widoku aplikacji oparta o silnik Twig oraz Vue.js,
- silnik nginx jako narzędzie do przekazania zapytania klienckiego do logiki biznesowej.

Wyróżnione komponenty są ściśle ze sobą połączone. Logika biznesowa oraz baza danych są nierozerwalne, 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 rodziale 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, odwzwierciedlenie 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 [22]. By zainicjować dane w systemie



Rys. 2.1. Schemat metamodelu Symfony [23]

bez konieczności manualnego wprowadzania ich autor wykorzystał dedykowaną ku temu bibliotekę wykorzystującą mechanizm fabryki danych testowych (ang. Fixtures Factory). Sam mechanizm fabryki danych testowych w procesie tworzenia oprogramowania umożliwia uruchomienie systemu z wypełnioną bazą danych, tak by móc przeprowadzać wymagane akcje i sprawdzić poprawność logiki biznesowej [18].

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. [24]

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 opisania 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. [16]

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. [13]

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. [14]

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 odpowowiednim wyborem do zarządzania komunikacją między mikroserwisami 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 wspomagania 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. [15]

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 propagowoać 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. [11]

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. [12]

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 Throttiling 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 koszta niedziałającej platformy.

3.2.1 Domain-Driven Design

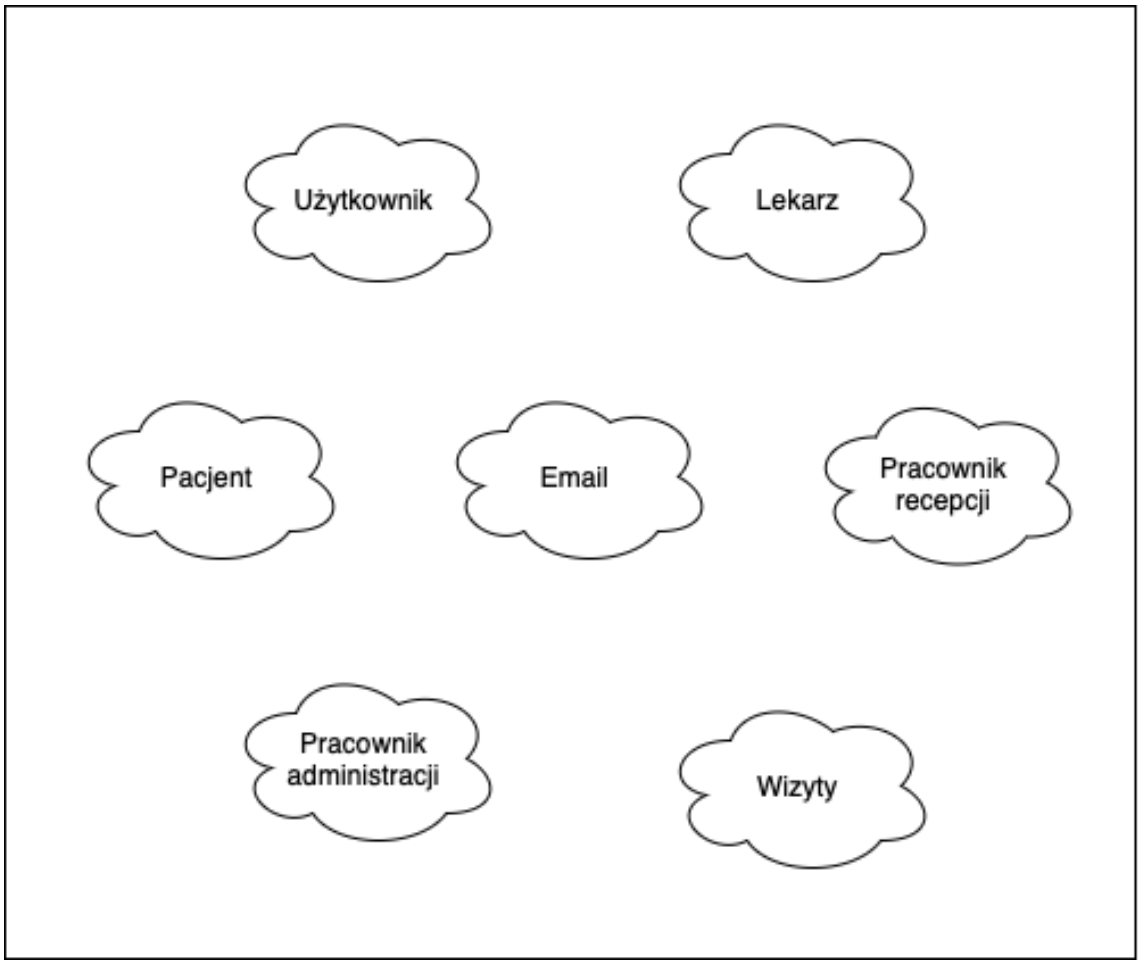
W skrócie DDD. Nazwa ta w języku polskim oznacza nacisk na prowadzenie rozwoju oprogramowania w kontekście jakiejś domeny. Jest to filozofia, która ma pomagać rozwiązywać problemy budowy oprogramowania dla skomplikowanych domen [19]. To podejście zakłada podział systemu na komponenty oraz ich zachowania, 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 mikroserwisów.

3.2.2 Zakres migracji

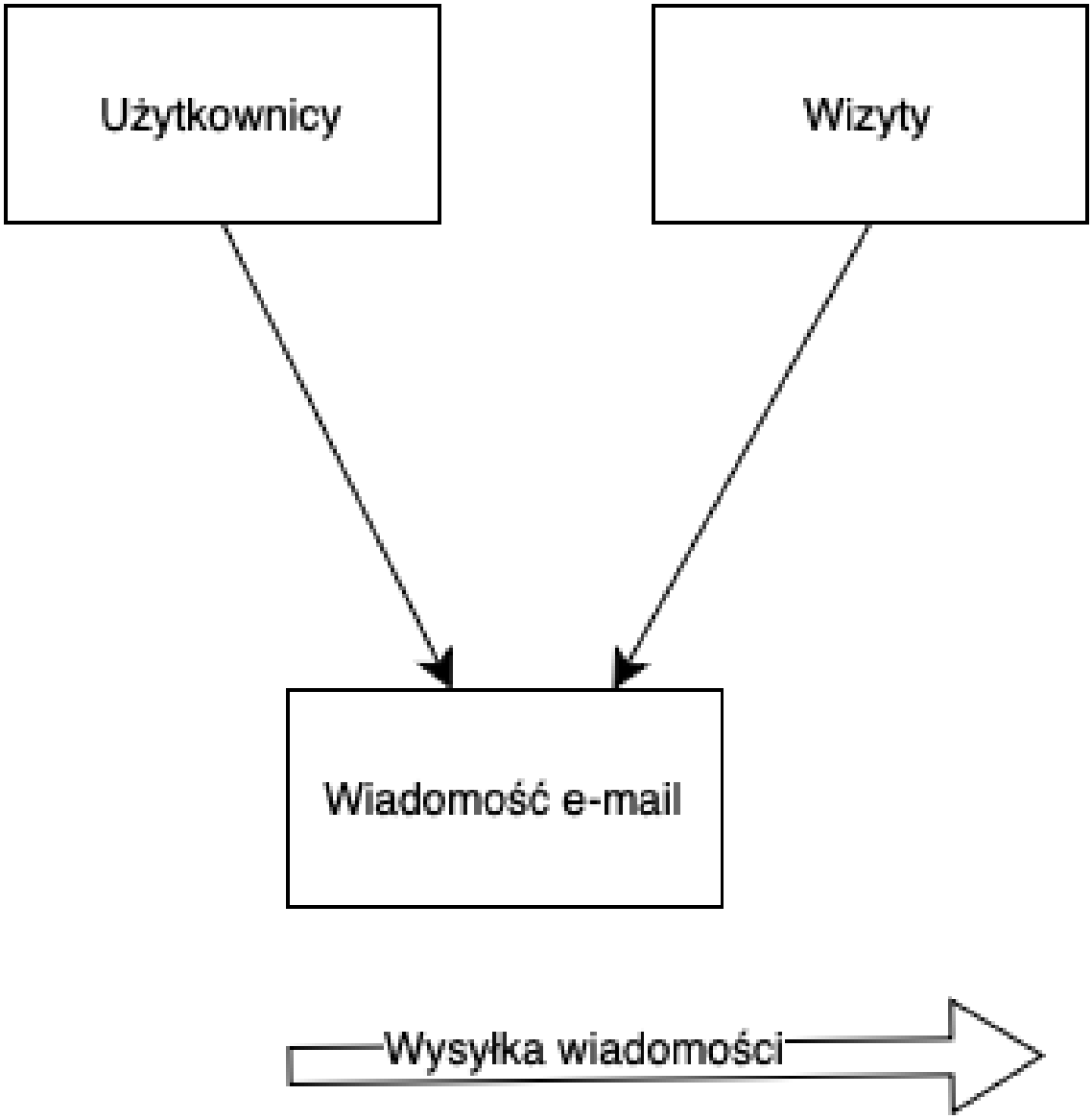
Według autora książki “Monolith to Microservices“ należy zrozumieć w jakim zakresie szczegółowa ma być domena, którą próbuje się dzielić na mniejsze części, tak by było to rozsądne [20]. Patrz Rysunek 3.1.



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

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 wspomagania 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.



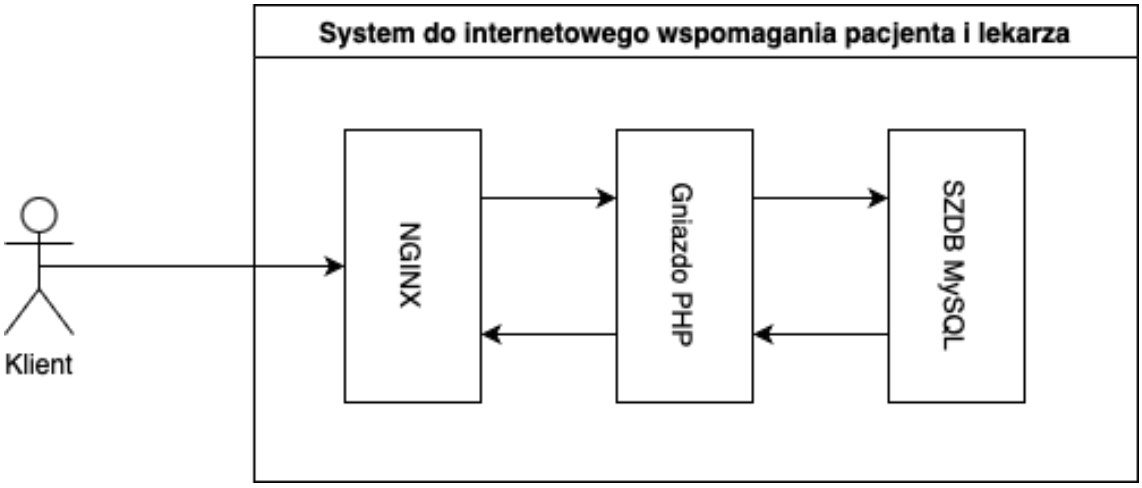
Rys. 3.2. Wysyłka wiadomości jest logicznie połączona dla tego modelu domeny, więc jej dalsza ekstrakcja może być trudna

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 zwirtualizowanym 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.

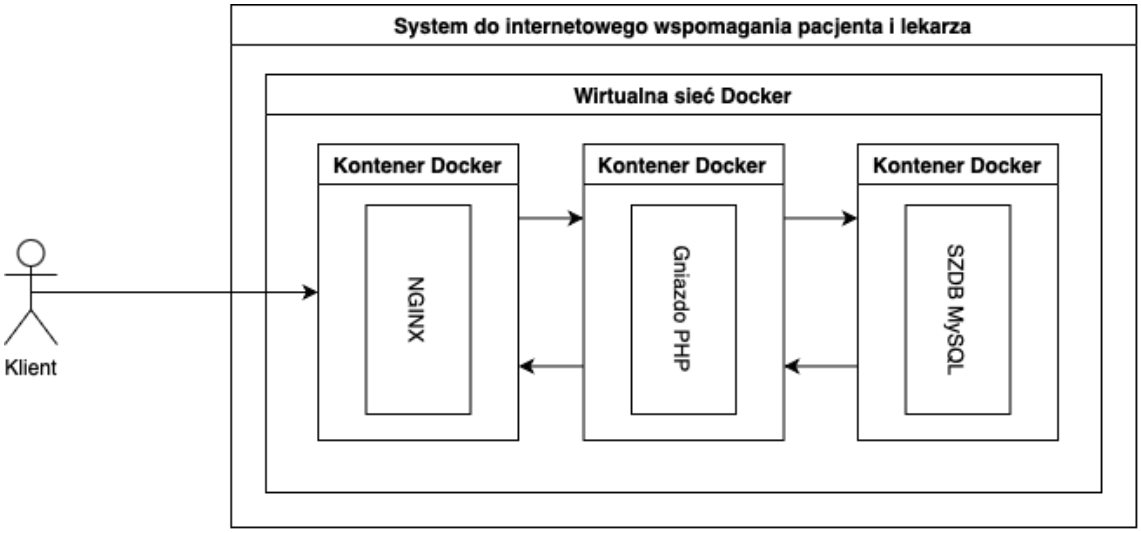


Rys. 4.1. Schemat architektury systemu do internetowego wspomagania pacjenta i lekarza w pierwszej fazie

4.2 Faza druga

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

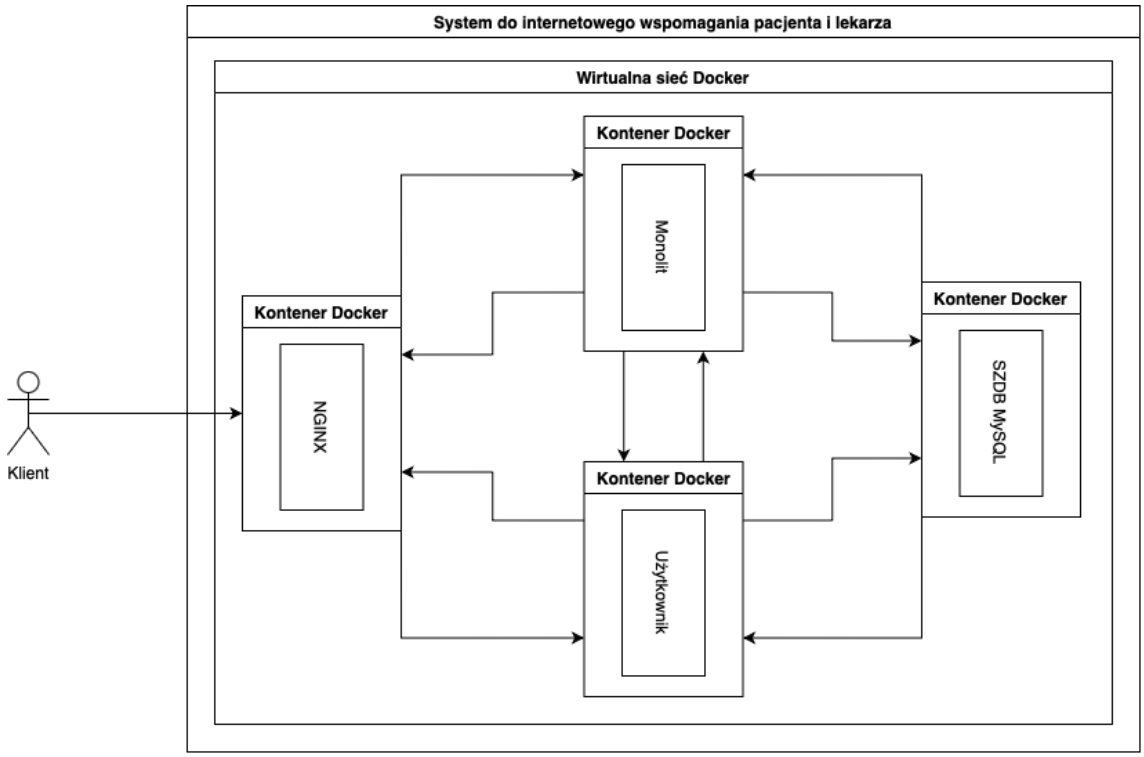
średnika zapytań nginx i logiki biznesowej w osobnych, niezależnych kontenerach, skomunikowanych ze sobą wirtualną siecią. Omawiana zmiana widoczna na Rysunku 4.2.



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 zwizualizowna 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

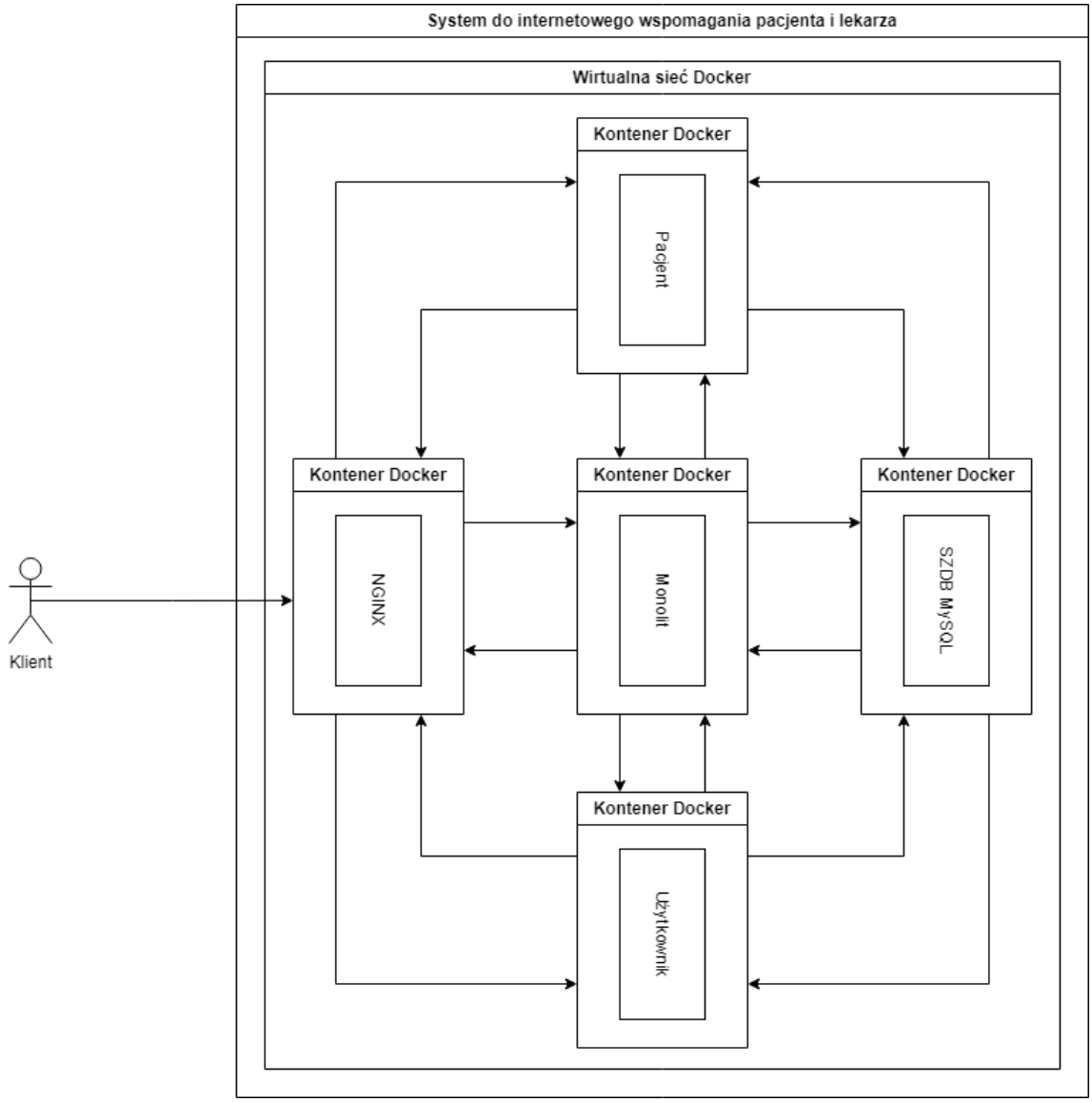
W fazie siódmej migracji dodano mikroserwis odpowiadający za obsługę wizyt. Proces technologiczny wygląda analogicznie do wcześniejszych faz. Wykonano zmianę konfiguracji oprogramowania nginx oraz utworzono nową bazę danych. Zmiany widoczne na Rysunku 4.7.

4.8 Faza ósma

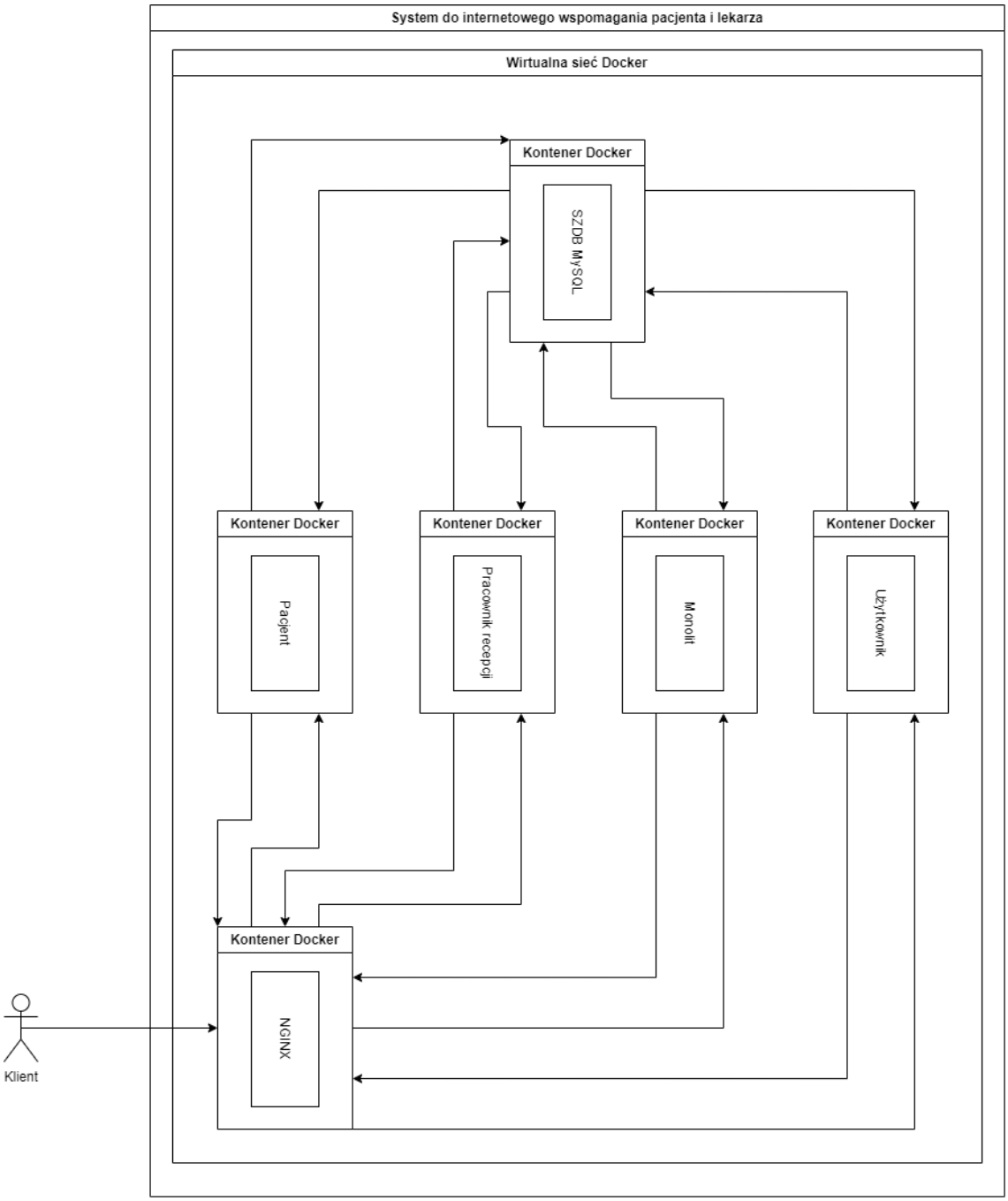
Ostatnia faza migracji to uruchomienie oprogramowania RabbitMQ, które będzie brokerem komunikatów portu AMQP. Dodatkowo uruchomiono serwis wysyłki maili. Na schemacie nastąpiło usunięcie części systemu określonej jako monolit, a to dlatego, że cały system w obecnym stanie jest już niezależny od jednej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

4.9 Podsumowanie przebiegu migracji

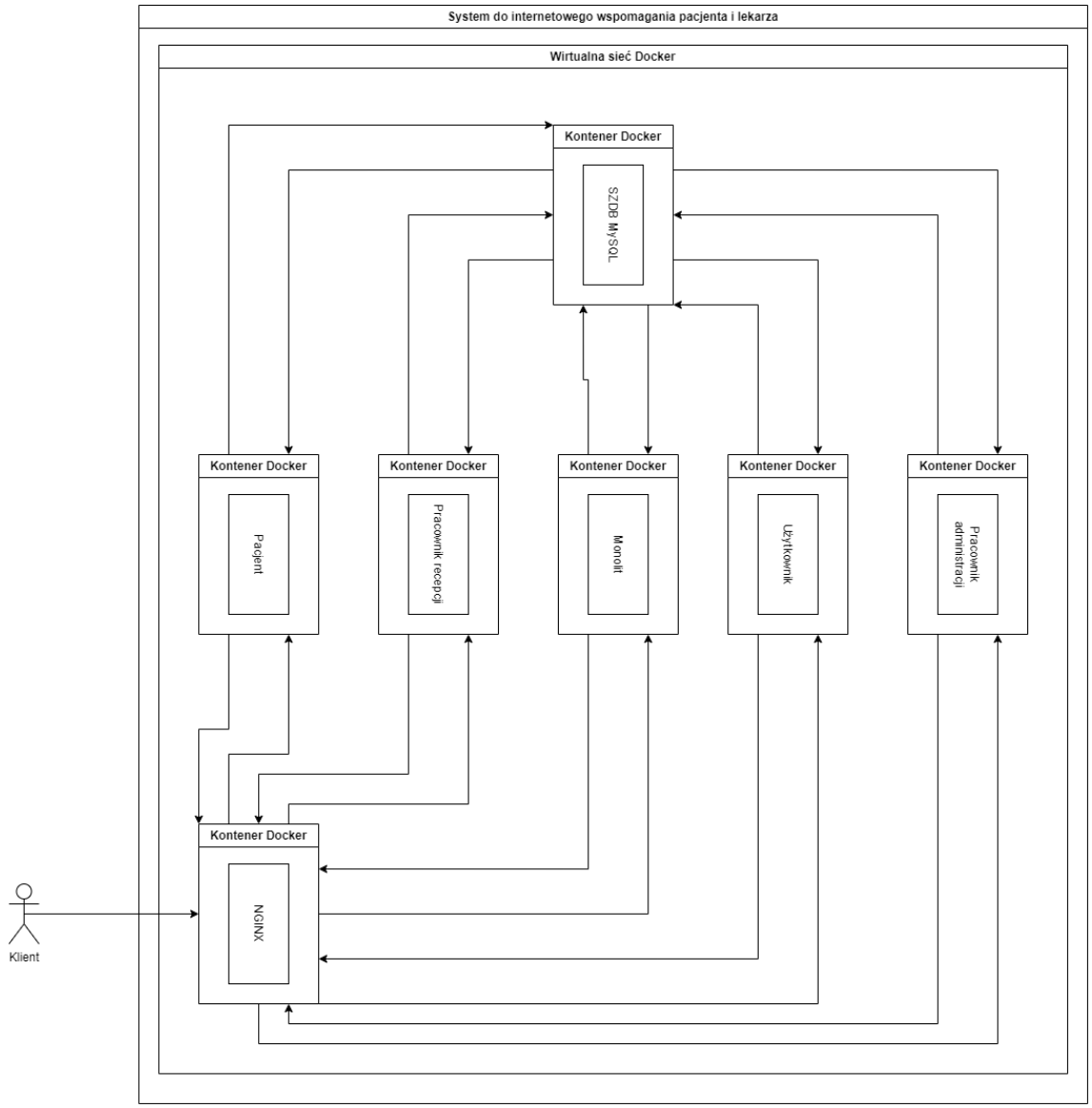
Proces stopniowego ewoluowania projektu monolitycznego w projekt mikroserwisowowy dla systemu do internetowego wspomagania pacjenta i lekarza, jest procesem skomplikowanym. Czas poświęcony na dokonanie takiej migracji to w przybliżeniu 112h. Proces należał jednak do łatwiejszych przez wzgląd na niekomercyjne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



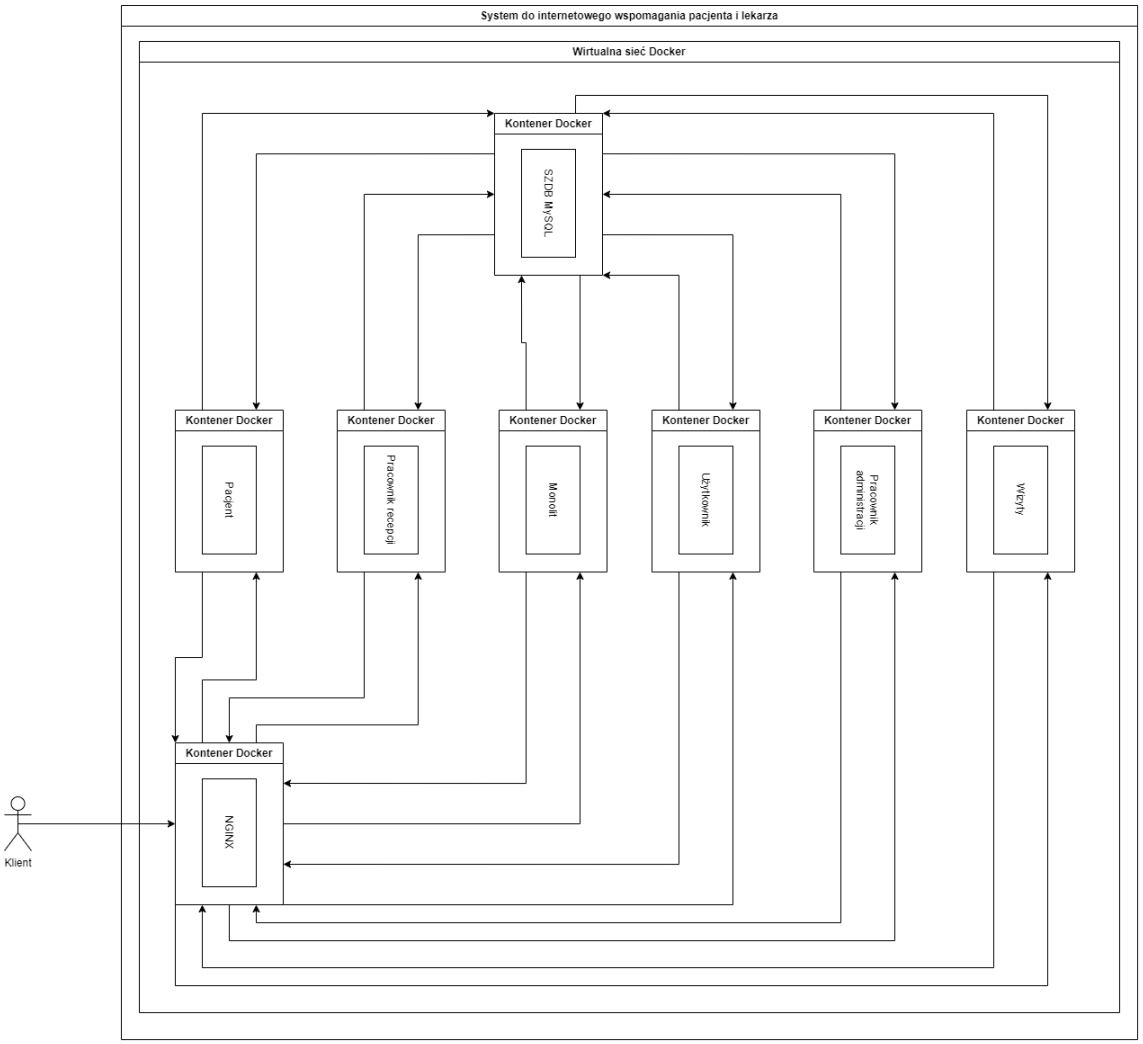
Rys. 4.4. Schemat architektury systemu do internetowego wspomagania pacjenta i lekarza w czwartej fazie



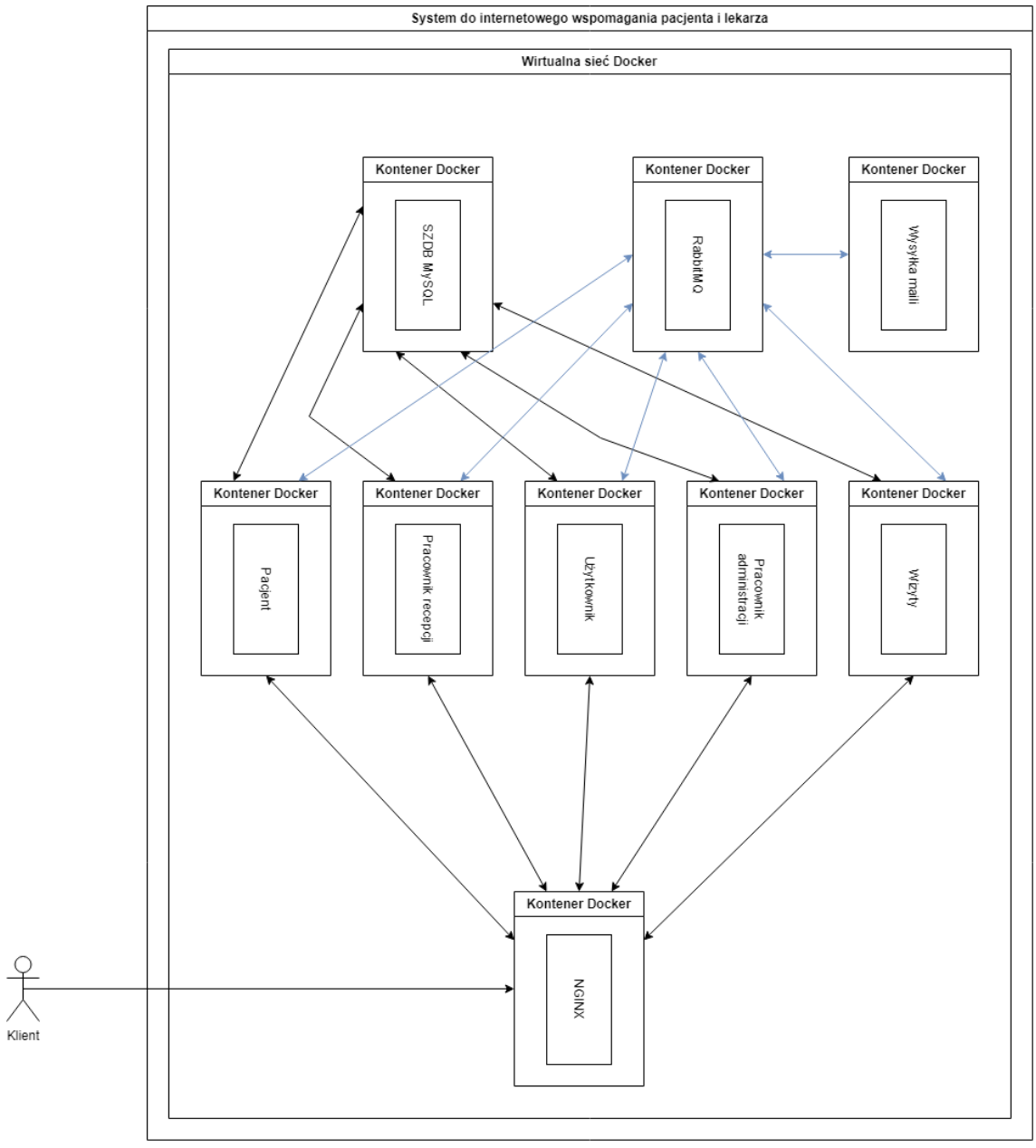
Rys. 4.5. Schemat architektury systemu do internetowego wspomagania pacjenta i lekarza w piątej fazie



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 [9]. 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 [21]. 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 integralność przesyłanych danych. TLS wykorzystuje szyfrowanie asymetryczne.

Szyfrowanie asymetryczne to rodzaj kryptografii, w którym jeden z kluczy jest kluczem publicznym [10]. 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 [14].

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 har-

monizacja prawa w ramach państw członkowskich Unii Europejskiej oraz umożliwienie swobodnego przepływu danych osobowych [8]. 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 [7].

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 [6]. Na Rysunku 5.1 przedstawiony przykład komunikatu, który dostarcza GitHub Dependabot.

Autor, prowadząc rozpoznanie 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 X.X.

Tabela 5.1. Opis punktów końcowych systemu do internetowego wspomagania 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

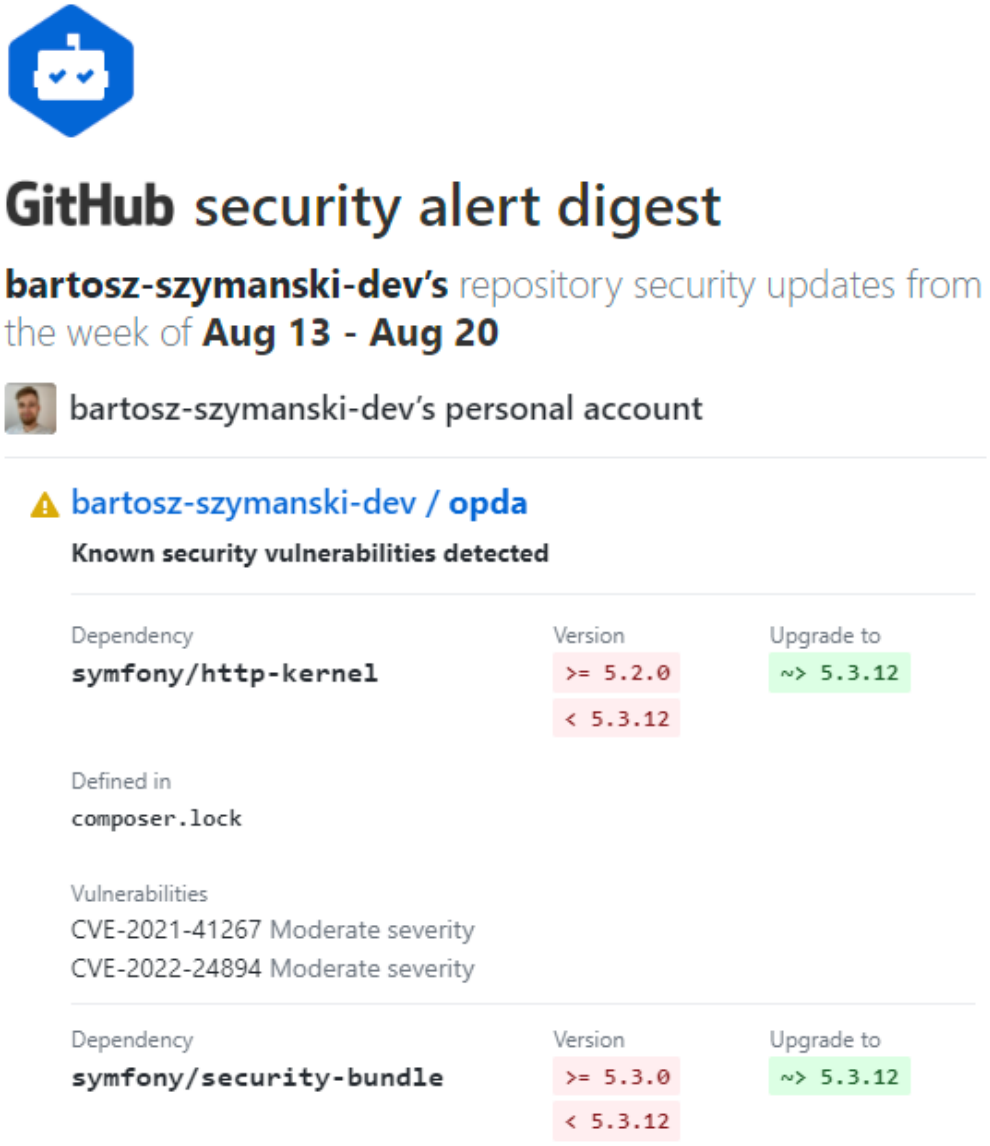
Punkt końcowy API	**Dozwolona metoda HTTP**	**Krótki opis punktu**	**Aktor posiadający dostęp**
/appointments/	GET	Pobranie listy umówionych wizyt.	pacjent, lekarz, pracownik recepcji, pracownik administracji
/appointments/ create/	POST	Utworzenie nowej wizyty.	pacjent, lekarz, pracownik recepcji, pracownik 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

Punkt końcowy API	**Dozwolona metoda HTTP**	**Krótki opis punktu**	**Aktor posiadający dostęp**
/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 X.X, 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.



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

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] 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.

[7] Gdpr - amazon web services (aws). https://aws.amazon.com/compliance/gdpr-center/. Data dostępu: 24 Sierpień 2024.

[8] Informacje ogólne - giodo. https://archiwum.giodo.gov.pl/1520143/j/pl. Data dostępu: 24 Sierpień 2024.

[9] (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.

[10] Post-quantum cryptography. https://www.nature.com/articles/nature23461. Data dostępu: 23 Sierpień 2024.

[11] Welcome - amazon route 53. https://docs.aws.amazon.com/Route53/latest/APIReference/Welcome.html. Data dostępu: 3 Sierpień 2024.

[12] Welcome - elastic load balancing. https://docs.aws.amazon.com/elasticloadbalancing/latest/APIReference/Welcome.html. Data dostępu: 3 Sierpień 2024.

[13] 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.

[14] 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.

[15] What is amazon s3? - amazon simple storage service. https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html. Data dostępu: 1 Sierpień 2024.

[16] 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.

[17] Cloud computing market. 2023. Data dostępu: 20 Styczeń 2024.

[18] 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.

[19] Scott Millett and Nick Tune. *Patterns, Principles, and Practices of Domain-Driven Design*. 2015. Data dostępu: 11 Sierpień 2024.

[20] Sam Newman. *Monolith to Microservices*. 2019. Data dostępu: 11 Sierpień 2024.

[21] Polski Komitet Normalizacyjny. *PN-I-02000. Technika informatyczna. Zabezpieczenia w systemach informatycznych. Terminologia*. 2002. Data dostępu: 14 Sierpień 2024.

[22] Fabien Potencier. *Symfony 5: The Fast Track*. 2020. Data dostępu: 6 Kwiecień 2024.

[23] M'hamed Rahmouni, Chaymae Talbi, and Soumia Ziti. Model-driven architecture; generating models from symfony. 2023. Data dostępu: 6 Kwiecień 2024.

[24] 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

Spis tabel