UNIWERSYTET GDAŃSKI

Krzysztof Torończak
Nr albumu: 285771

Wiktor Szymulewicz
Nr albumu: 285787

Bartosz Urbanowicz
Nr albumu: 288486

Tomasz Wojciechowski
Nr albumu: 285774

Szachowa platforma edukacyjna Passed Pawn - rozwój z uwzględnieniem praktyk DevOps i wykorzystaniem AI

Kierunek studiów: Informatyka
Poziom studiów: I stopnia - licencjackie
Profil studiów: praktyczny
Forma studiów: studia stacjonarne

Praca dyplomowa licencjacka
wykonana pod kierunkiem
dr Jakub Neumann, prof. UG

Gdańsk, 2025 r.

Spis treści

1. Wstęp **7**
- 1.1. Wprowadzenie w problematykę nauki gry w szachy . . . 7
- 1.2. Cel pracy . . . 7
- 1.3. Istniejące podobne rozwiązania rynkowe . . . 8
  - 1.3.1. Chessable . . . 8
  - 1.3.2. Chessly . . . 8
  - 1.3.3. lichess.org . . . 9
  - 1.3.4. Podsumowanie istniejących rozwiązań . . . 9

2. Analiza wymagań systemu **10**
- 2.1. Wymagania funkcjonalne . . . 10
  - 2.1.1. Rejestracja i logowanie użytkownika . . . 10
  - 2.1.2. Przeglądanie kursów . . . 10
  - 2.1.3. Szczegóły kursu . . . 10
  - 2.1.4. Uczenie się z kursu . . . 10
  - 2.1.5. Tworzenie recenzji . . . 11
  - 2.1.6. Tworzenie kursów . . . 11
- 2.2. Wymagania niefunkcjonalne . . . 11
- 2.3. Opis aktorów . . . 11
- 2.4. Diagram przypadków użycia trenera . . . 13
- 2.5. Diagram przypadków użycia ucznia . . . 14
- 2.6. Diagram przypadków użycia części wspólnych trenera i ucznia . . . 15
- 2.7. Zewnętrzne systemy, na których polega aplikacja . . . 16
  - 2.7.1. Stripe . . . 16
  - 2.7.2. Cloudinary . . . 16
  - 2.7.3. MistralAI . . . 16

3. Projekt interfejsu użytkownika **17**
- 3.1. Proces projektowania UI . . . 17
- 3.2. Wybrane narzędzia projektowania interfejsu użytkownika . . . 17
- 3.3. Ograniczenia bibliotek UI . . . 19
- 3.4. Autorskie komponenty i widoki UI . . . 20
- 3.5. Znaczenie oryginalnego projektu interfejsu użytkownika . . . 26
- 3.6. Problemy i stracone zasoby . . . 27

4. Architektura Passed Pawn **28**
- 4.1. Przykłady powszechnie stosowanych rodzajów architektur systemów . . . 29
  - 4.1.1. Monolit . . . 29
  - 4.1.2. SOA (Service-Oriented Architecture) . . . 29
  - 4.1.3. Mikroserwisy . . . 30
  - 4.1.4. Big Ball of Mud - antywzorzec . . . 31
- 4.2. Wybór i projekt architektury Passed Pawn . . . 34
- 4.3. Single Page Application (SPA) vs Multi Page Application (MPA) . . . 39
  - 4.3.1. Szybkość interakcji . . . 39
  - 4.3.2. SEO (Search Engine Optimization) . . . 39

4.3.3. Początkowy czas ładowania . . . 40
4.3.4. Użycie zasobów klienta . . . 40
4.3.5. Obciążenie serwera . . . 40
4.3.6. Bogate doświadczenie użytkownika . . . 40
4.3.7. Efektywny development . . . 40
4.4. Wykorzystane protokoły komunikacyjne . . . 41
4.4.1. SSE . . . 41
4.4.2. HTTP, HTTP + TLS . . . 41

5. Aplikacja przeglądarkowa typu SPA **42**
5.1. Porównanie najpopularniejszych frameworków/bibliotek SPA – tabela . . . 42
5.2. Wybory technologiczne związane z interfejsem użytkownika . . . 43
5.2.1. Porównanie frameworków pod względem stylowania – tabela . . . 43
5.2.2. Podejście do stylowania . . . 44
5.2.3. Sposób enkapsulacji stylów . . . 45
5.2.4. PrimeNG jako wybrana biblioteka UI . . . 45
5.3. Cechy Angulara oraz ich wpływ na implementację . . . 45
5.3.1. Silnie narzucone założenia projektowe . . . 45
5.3.2. TypeScript . . . 45
5.3.3. System Dependency Injection . . . 46
5.3.4. Rozbudowane wsparcie dla formularzy . . . 46
5.3.5. Dyrektywy atrybutowe (attribute directives) . . . 47
5.3.6. Rozwinięty i ekspresywny data binding (wiązanie danych) . . . 47
5.3.7. Wsparcie dla programowania reaktywnego - RxJS . . . 48
5.3.8. Detekcja zmian . . . 48
5.3.9. Wbudowany klient HTTP . . . 49
5.4. Wybory technologiczne związane z kontrolą dostępu . . . 50
5.4.1. KeycloakJS, Keycloak-Angular . . . 50
5.4.2. Guardy . . . 50
5.5. Struktura implementacji . . . 51
5.6. Logika szachowa . . . 52

6. Główna aplikacja serwerowa **54**
6.1. Cel i rola aplikacji serwerowej . . . 54
6.2. ASP.NET Core — Powód wybrania technologii . . . 54
6.3. Architektura aplikacji . . . 54
6.3.1. Wzorce projektowe . . . 54
6.3.2. N-tier architecture . . . 54
6.4. Warstwy aplikacji . . . 55
6.4.1. API . . . 55
6.4.2. Business Logic . . . 55
6.4.3. Data Access . . . 55
6.4.4. Models . . . 55
6.4.5. Wykres warstw . . . 55
6.5. Interfejs API . . . 55
6.5.1. REST . . . 55
6.5.2. Swagger . . . 56

6.6. Przykłady operacji . . . 57
6.6.1. Pobieranie lekcji z użyciem ID . . . 57
6.7. Obsługa błędów i walidacja . . . 58
6.7.1. Global exception handler . . . 58
6.7.2. Walidacja . . . 58
6.8. Komunikacja z bazą danych — EntityFramework Core . . . 58

7. Moduł AI jako narzędzie doskonalenia UX 59
7.1. Cel wprowadzenia modułu . . . 59
7.2. System RAG - rozwiązanie pozwalające na wykorzystanie LLM . . . 59
7.3. Implementacja serwerowej aplikacji AI . . . 60
7.3.1. Python . . . 60
7.3.2. Wykorzystanie LangChain do utworzenia RAG Chain . . . 60
7.3.3. Wektorowa baza danych . . . 61
7.3.4. API . . . 62
7.4. Analiza dostępnych modeli AI . . . 62
7.5. Diagram sekwencji modułu AI . . . 62

8. Infrastruktura systemu 63
8.1. Podział infrastruktury na tradycyjną/lokalną i chmurową . . . 63
8.2. Wybór serwera dedykowanego OVH oraz domeny internetowej . . . 64
8.2.1. Oferta i specyfikacja serwera dedykowanego . . . 64
8.2.2. Oferta i specyfikacja domeny internetowej . . . 65
8.3. Oprogramowanie serwera dedykowanego . . . 65
8.4. Kubernetes - wybrane narzędzie orkiestracji systemu . . . 65
8.4.1. Elastyczności Kubernetes . . . 66
8.4.2. Niezawodności i skalowalność Kubernetes . . . 67
8.4.3. Uzasadnienie wykorzystania Kubernetes . . . 67
8.5. K3s jako wybrana dystrybucja K8s . . . 67
8.6. Najważniejsze komponenty niestandardowe klastra . . . 68
8.6.1. Longhorn . . . 68
8.6.2. Nginx Ingress Controller . . . 69
8.6.3. Cert-manager . . . 69
8.6.4. Inne niestandardowe elementy klastra . . . 70
8.7. Infrastruktura rozwoju i wdrażania aplikacji . . . 70
8.7.1. Serwer Jenkins . . . 70
8.7.2. Usługi GitHub . . . 70
8.8. Serwer Keycloak . . . 71
8.9. Wykorzystanie dobrych praktyk Infrastructure as Code (IaC), GitOps . . . 71
8.9.1. Geneza IaC. . . . 71
8.9.2. Definicja i znaczenie IaC . . . 71
8.9.3. Zakres wykorzystania praktyk IaC . . . 72
8.9.4. Geneza GitOps . . . 73
8.9.5. Definicja i znaczenie GitOps . . . 73
8.9.6. Zakres wykorzystania praktyk GitOps . . . 73

9. Automatyzacja procesów, podejście DevOps 75

9.1. Metodologia rozwoju i utrzymywania aplikacji Passed Pawn . . . 75
9.1.1. Zastosowanie praktyk DevOps . . . 76
9.1.2. Zastosowanie praktyk CI/CD . . . 76
9.2. Jenkins - główne narzędzie do automatyzacji procesów . . . 77
9.2.1. Najpopularniejsze alternatywne rozwiązania do Jenkinsa . . . 78
9.2.2. Zakres zastosowania Jenkinsa . . . 78
9.2.3. Uzasadnienie wyboru Jenkinsa . . . 78
9.3. Podział odpowiedzialności za pipeline-y . . . 79
9.4. Integracja Jenkins — Kubernetes . . . 82
9.5. GitHub Actions - automatyzacja testów . . . 83
9.6. Git hooki - walidacja kodu . . . 83
9.6.1. Uzasadnienie użycia hooka pre-commit . . . 84
9.6.2. Husky - agnostyczna platformowo nakładka na git hooki . . . 84
9.6.3. ESLint - statyczna analiza kodu . . . 84
9.6.4. Stylelint - statyczna analizy kodu . . . 84

10. Bezpieczeństwo systemu **85**
10.1. Centralny serwer autoryzacyjny Keycloak . . . 85
10.2. Metodologia zabezpieczania systemu . . . 86
10.3. Ogólne schematy i mechanizmy bezpieczeństwa . . . 87
10.3.1. Same Origin Policy . . . 87
10.3.2. HTTP + TLS . . . 87
10.4. Bezpieczeństwo aplikacji klienckiej . . . 87
10.4.1. Zabezpieczenia przed XSS . . . 87
10.4.2. Zabezpieczenia przed kradzieżą filmów . . . 88
10.4.3. Zarządzanie tokenami dostępu przez serwer Keycloak i adapter Keycloak . . . 88
10.5. Bezpieczeństwo aplikacji serwerowych . . . 89
10.5.1. SQL injection . . . 89
10.5.2. Walidacja danych . . . 89
10.5.3. Walidacja RBAC . . . 90
10.6. Bezpieczeństwo infrastruktury systemu . . . 90

11. Organizacja pracy nad projektem **92**
11.1. Pair programming . . . 92
11.2. Recenzja kodu (Code review) . . . 93
11.3. Kanban . . . 93
11.4. Burza mózgów . . . 94
11.5. Komunikacja w zespole . . . 95

12. Testowanie i walidacja systemu **96**
12.1. Testy API . . . 96
12.1.1. Testy manualne . . . 96
12.1.2. Testy automatyczne . . . 96
12.1.3. Github Actions . . . 97
12.1.4. Rider . . . 97
12.2. Testy end-to-end . . . 98

13. Podsumowanie projektu i potencjał do dalszego rozwoju **100**

14. Podział pracy w zespole **101**

Załączniki

Do pracy, w postaci pliku zip, został dołączony kod pochodzący z głównych gałęzi (main) repozytoriów git wykorzystanych w pracy. Ponadto, wgląd w repozytoria dostępny jest na platformie GitHub: https://github.com/passed-pawn-dev

Aplikacja webowa w wersji deweloperskiej dostępna jest pod adresem:
https://pp-dev.passed-pawn.com

1. Wstęp

1.1. Wprowadzenie w problematykę nauki gry w szachy

Szachy od wieków są jedną z najpopularniejszych gier planszowych na świecie. Ich początki sięgają VI wieku naszej ery, jednak w ostatnich latach gra przeżywa prawdziwy renesans. Dzieje się tak z powodu zyskującej na popularności formie rozgrywki przez Internet. Według szacunków, na świecie w szachy może grać nawet 600 milionów osób.[1] Wraz ze wzrostem popularności tej strategicznej gry, naturalnie rośnie zapotrzebowanie na skuteczne i łatwo dostępne sposoby na naukę szachów. Potencjał tego rynku potęguje również fakt, że gracz potrzebuje edukacji i rozwoju na każdym poziomie zaawansowania. Całkowicie nowi gracze muszą nauczyć się podstawowych zasad, takich jak ruchy figur i ogólne zasady rozgrywki. W momencie, w którym opanują podstawy, przychodzi czas na naukę bardziej złożonych strategii i motywów. Zaawansowani zaś uczą się na pamięć debiutów i końcówek. Nawet na poziomie zawodniczym, gracze muszą cały czas poznawać nowe warianty i uczyć się odpowiadać na nie, w zależności od tego, co może zagrać ich przeciwnik.

Wielu początkujących szachistów, błędnie zakłada, że wystarczy im doskonalenie swoich umiejętności podczas rozgrywania partii. Takie podejście może prowadzić do utrwalania złych nawyków, rosnącej liczby porażek, i w efekcie zaniechania dalszej gry. Gracze mogą również spotkać się z szeroką ofertą lekcji indywidualnych prowadzonych przez trenerów szachowych. Ten rodzaj nauki, choć może okazać się bardzo skuteczny, wiąże się z poświęceniem czasu i znacznych środków finansowych. Pojawia się zatem zapotrzebowanie na platformę, która umożliwi mniej obciążającą budżet naukę przez Internet, na którą gracz będzie mógł poświęcić tyle czasu, ile sam uzna za potrzebne. Jednocześnie platforma powinna oferować kompleksową i skuteczną edukację szachową.

1.2. Cel pracy

Celem pracy jest stworzenie aplikacji webowej służącej do efektywnej nauki gry w szachy poprzez wykorzystanie kursów szachowych tworzonych przez doświadczonych trenerów zarejestrowanych w systemie. Aplikacja ma za zadanie spełniać potrzeby dwóch rodzajów użytkowników — trenerów (dalej nazywanych również coachami) i uczniów (nazywanych także studentami). Uczniowi powinna ona umożliwiać zakup kursów jak najlepiej pasujących do jego potrzeb. Powinien on mieć ponadto możliwość dokładnego wyszukania interesującego go kursu, podejrzenia jego wybranych elementów oraz nabycia go. W zakupionym kursie uczeń powinien mieć dostęp do różnego rodzaju interaktywnych elementów dedykowanych grze w szachy, wspomagających naukę, które pozwolą mu efektywnie przyswajać i utrwalać wiedzę we własnym tempie. Powinien mieć również możliwość oceny kursu, by podzielić się swoją opinią z innymi użytkownikami.

Trener powinien mieć możliwość tworzenia kursów, o ustalonej przez niego cenie, objętości i zakresie materiału. Udostępnione narzędzia do tworzenia interaktywnych elementów mają być proste, intuicyjne, ale też elastyczne i oferujące wiele funkcjonalności. Oprócz tego, umożliwienie trenerowi możliwość edycji i modyfikacji utworzonych kursów, lekcji czy elementów jest pożądaną częścią aplikacji.

Oprócz zrealizowania przedstawionych założeń funkcjonalnych, równie ważnym celem pracy jest także realizowanie projektu w sposób umożliwiający jego proste utrzymanie, swobodną modyfikację

[1] Chess.com. *How Many Chess Players Are There In The World?* `https://www.chess.com/article/view/how-many-chess-players-are-there-in-the-world`. Data dostępu: 2025-06-23. 2017.

i rozwój. W drodze osiągnięcia tego celu stoi dokonywanie świadomych i przemyślanych decyzji technologicznych i architektonicznych, przy jednoczesnym wykorzystaniu współczesnych, dobrych standardów wytwarzania oprogramowania. Mowa przede wszystkim o zastosowaniu podejścia DevOps i silnym wykorzystaniu praktyk Infrastruktury jako Kodu (ang. Infrastructure as Code - IaC), GitOps oraz CI/CD.

Ostatnim ważnym tematem, który praca ma na celu zgłębić, jest efektywne wykorzystanie mechanizmów AI w celu usprawnienia funkcjonowania zaprojektowanej aplikacji webowej. W dzisiejszych czasach sztuczna inteligencja już wpłynęła na funkcjonowanie wielu branż i stanowi duże pole posuwających się w szybkim tempie innowacji. Wobec obecnej rewolucji AI nie można przejść obojętnie i dlatego zaadoptowanie w aplikacji funkcjonalności opartych o sztuczną inteligencję stanowi ukłon ku obecnemu kierunkowi i tematyce postępujących innowacji.

1.3. Istniejące podobne rozwiązania rynkowe

1.3.1. Chessable

Chessable to edukacyjna platforma szachowa stworzona w 2016 roku, przejęta przez Play Magnus Group w 2019 roku i wcielona w ekosystem Chess.com w 2022 roku. Platforma umożliwia naukę gry w szachy poprzez interaktywne kursy z ćwiczeniami na szachownicy oraz materiałami wideo, a także pozwala na śledzenie postępów i statystyk użytkownika.

Główną wadą platformy Chessable jest nieintuicyjny interfejs użytkownika, w szczególności panel tworzenia i zarządzania kursami użytkownika. Widoczne są błędy logiczne w implementacji i strukturze portalu, których przykładem jest dodawanie lekcji do kursu. Dodatkowo niedopracowana warstwa prezentacji pogarsza odbiór aplikacji — brakuje spójności interfejsu użytkownika między widokiem dostępnych szkoleń szachowych a formularzem ich tworzenia. Platforma szachowa PassedPawn operuje na ujednoliconej warstwie wizualnej, gdzie korzystanie z platformy jest możliwie jak najprostsze i przystępne dla nowych użytkowników. Z drugiej zaś strony dużą zaletą aplikacji jest bezpośrednia integracja i wsparcie przez platformę Chess.com.

Serwis Chessable posiada rozbudowaną strategię monetyzacji, która umożliwia zakup pojedynczego kursu, lub model freemium, opierający się na korzystaniu z platformy bez opłat, mając dostęp do ograniczonej puli funkcjonalności portalu. Inną możliwością jest okresowa subskrypcja PRO, która oferuje zaawansowane statystyki i analizę postępów. Abonament daje również dostęp do większej liczby opcji w edytorze kursów. Z kolei PassedPawn umożliwia zakup pojedynczych szkoleń. Strategia generowania przychodów jest prosta i czytelna, klient wie, za co płaci i co dostaje.

1.3.2. Chessly

Chessly to edukacyjna platforma szachowa stworzona przez amerykańskiego szachistę i osobowość internetową Levy'ego Rozmana. Od startu platformy w 2021 roku zarejestrowało się na niej ponad 900 000 użytkowników.[2] Serwis oferuje na ten moment ponad 50 kursów, a model monetyzacji oparty jest na miesięcznej subskrypcji. Kursy składają się z interaktywnych elementów i opisów oraz filmów.

Zasadniczą różnicą pomiędzy PassedPawn i Chessly jest fakt, że platforma ta nie oferuje możliwości tworzenia i sprzedawania kursów zwykłym zarejestrowanym użytkownikom. Kursy są tworzone tylko przez założyciela oraz jego pracowników. Powoduje to znacznie mniejszy potencjał na ilość kursów

[2]Chessly. *Chessly main page*. `https://chessly.com/`. Data dostępu: 2025-06-23. 2025.

dostępnych na platformie. Rozwiązanie proponowane w PassedPawn, umożliwiające każdemu stworzenie własnego kursu, powoduje, że ich liczba może szybko wzrastać. Wadą takiego rozwiązania jest utrudniona kontrola jakości, natomiast ten problem może być częściowo rozwiązany przez dostępny na opisywanej platformie system recenzji kursów, które może wystawić każdy użytkownik.

Model monetyzacji opierający się na miesięcznej subskrypcji może nie odpowiadać wielu użytkownikom. Jedna płatność zapewnia dostęp do wszystkich kursów, natomiast nie ma możliwości wykupienia dostępu do konkretnego kursu. Powoduje to, że jeżeli użytkownik jest zainteresowany np. konkretnym otwarciem lub strategią, nadal musi on zapłacić pełną cenę, do której wliczone są nieinteresujące go kursy. Problematyczny jest też fakt, że po anulowaniu subskrypcji użytkownik traci dostęp do kursów. W PassedPawn raz kupiony kurs będzie już zawsze dostępny dla użytkownika.

1.3.3. lichess.org

lichess.org jest darmową, otwarto źródłową platformą szachową. Choć jej główną funkcjonalnością jest rozgrywanie partii pomiędzy graczami, posiada również kilka narzędzi edukacyjnych. Narzędzia te możemy podzielić na te oferowane bezpośrednio przez platformę oraz te, które mogą dodawać wszyscy użytkownicy. Platforma bezpośrednio oferuje szeroki zestaw łamigłówek szachowych, podzielonych na motywy, o różnym poziomie trudności. Dostępne są również proste zadania, które ułatwiają naukę ruchów figur i podstawowych motywów taktycznych. Użytkownicy mogą dodawać tzw. opracowania. Są to zestawy przykładów z interaktywną szachownicą i opisami oraz łamigłówek, które gracz musi rozwiązać sam.

Choć wszystkie opracowania są darmowe, nie są one zazwyczaj zbyt obszerne. W przeciwieństwie do kursów w PassedPawn składają się one zwykle z kilku przykładów opisujących pojedynczy wariant lub motyw strategiczny. Fakt, że cała zawartość jest darmowa, jest zaletą dla uczniów, ale może być dużą wadą dla trenerów. Trener może zarabiać wyłącznie poprzez oferowanie lekcji indywidualnych. W PassedPawn trenerzy będą zarabiać bezpośrednio na wrzucanych przez siebie kursach.

lichess.org jest stowarzyszeniem non-profit. Aplikacja nie wyświetla żadnych reklam i jest oparta na otwartym oprogramowaniu. Finansowanie pochodzi wyłącznie z dobrowolnych darowizn użytkowników platformy. Taki model biznesowy zasadniczo różni się od modelu PassedPawn, który koncentruje się na osiąganiu zysku.

1.3.4. Podsumowanie istniejących rozwiązań

Istniejące szachowe platformy edukacyjne oferują wiele możliwości dla trenerów i uczniów, ale każda z nich ma też znaczące wady. Szczególnie nieciekawie wybór wygląda dla tutorów szachowych. Dostępne rozwiązania tworzenia własnych szkoleń są niedopracowane. Platforma Chessable, która posiada jedną z najbardziej rozbudowanej puli funkcjonalności zarządzania własnymi kursami, jest nieintuicyjna w obsłudze. Aplikacja Chessly nie oferuje możliwości tworzenia własnych szkoleń, a serwis lichess oferuje jedynie sposób na ogłoszenie lekcji indywidualnych. Z kolei PassedPawn wypełnia lukę na rynku aplikacji szachowych, oferując równie rozbudowane wsparcie narzędziowe zarówno dla trenera, jak i ucznia. Platforma jest przejrzysta i prosta w użyciu. Stworzone środowisko umożliwia rozwój biznesowy tutorów, jak i pogłębianie wiedzy przez zafascynowanych szachistów, kreując rozwój pełnej pasji i wzajemnego wsparcia społeczności.

2. Analiza wymagań systemu

2.1. Wymagania funkcjonalne

Platforma Passed Pawn została zaprojektowana z myślą o zarówno początkujących, jak i bardziej zaawansowanych graczach, chcących rozwinąć swoje umiejętności szachowe. W tym celu system oferuje szereg funkcjonalności umożliwiających naukę, ćwiczenie oraz śledzenie postępów. Inni użytkownicy mają możliwość tworzenia i udostępniania materiałów do nauki.

2.1.1. Rejestracja i logowanie użytkownika

Aplikacja umożliwia tworzenie konta użytkownika oraz logowanie przy użyciu loginu i hasła. Użytkownicy mają swoje role, Student (uczeń) lub Coach (trener), a sesja użytkownika jest bezpiecznie przechowywana w przeglądarce.

2.1.2. Przeglądanie kursów

Uczeń ma możliwość przeglądania kursów, które są dostępne do nauki. Użytkownik wyraźnie widzi, ile zapłaci za kurs, oraz czego dotyczy. Widzi też, jaki jest zalecany poziom zaawansowania ucznia.

Aplikacja oferuje szeroki wybór filtrów, aby jak najbardziej uprościć tę czynność. Filtry są bardzo responsywne i reagują, jak tylko użytkownik coś zaznaczy.

2.1.3. Szczegóły kursu

Użytkownik ma również opcje zobaczyć szczegóły wybranego kursu. Są tutaj widoczne elementy, jakie są zawarte w kursie, kto jest autorem, oraz jakie ma recenzje. Tutaj użytkownik również ma opcje kupna kursu bądź dodania do swojej listy, jeżeli jest darmowy.

2.1.4. Uczenie się z kursu

Jeżeli kurs został kupiony bądź dodany do listy, użytkownik może zacząć studiowanie. Każdy kurs zawiera lekcje, a każda lekcja zawiera swoje elementy. Platforma wspiera cztery różne rodzaje elementów i każda lekcja może mieć różne elementy w dowolnej ilości.

- Zagadka — Problem szachowy, który ma tylko jedno rozwiązanie. Użytkownik dostaje dostęp do interaktywnej szachownicy i ma za zadanie wykonać sekwencję ruchów, które wydają się najlepsze w danej pozycji. Jeżeli ruch jest nie poprawny, zostaje automatycznie cofnięty i użytkownik musi ponieść kolejną próbę. Jest to idealny sposób na naukę specyficznych debiutów albo znajdywanie ciekawych wzorów taktycznych.
- Przykład — Użytkownik widzi szachownicę, na której jest już przygotowana pozycja, oraz opcjonalnie strzałki i zaznaczone kolorem pola. Oprócz tego użytkownik otrzymuje tekst z wyjaśnieniem, co się dzieje na planszy.
- Wideo — Użytkownik dostaje dostęp do filmu, co umożliwia wyjaśnienie dowolnego konceptu, w wygodnym formacie dla obu stron.
- Quiz — Zapytanie, na które użytkownik musi odpowiedzieć poprzez wybranie jednej z kilku opcji. Może to być proste pytanie tekstowe, ale jest też możliwość wyświetlenie szachownicy i zadanie pytanie na jej temat, na przykład „jaki jest najlepszy ruch w tej pozycji?”.

2.1.5. Tworzenie recenzji

Każdy uczeń może zostawić recenzje pod kursem, z którego się uczył. Każda recenzja wymaga oceny w skali 1-5 oraz opcjonalnie komentarza. Inni użytkownicy mogą przeglądać komentarze na widoku szczegółów kursu, a średnia ocen jest wyraźnie wyświetlana.

2.1.6. Tworzenie kursów

Użytkownicy zarejestrowani jako trenerzy (Coach) uzyskują możliwość tworzenia kursów. Aplikacja udostępnia zaawansowany i intuicyjny edytor, który umożliwia proste tworzenie dowolnego rodzaju elementu lekcji.

2.2. Wymagania niefunkcjonalne

Przedstawiając wymagania niefunkcjonalne mowa o takich wymaganiach, które nie definiują samych funkcjonalności systemu. Dotyczą natomiast takich zagadnień, jak wydajność, bezpieczeństwo, niezawodność, użyteczność, prostota użycia czy skalowalność. Wymagania funkcjonalne mogą wpływać na wymagania niefunkcjonalne i odwrotnie. Z tego powodu zarówno określenie wymagań funkcjonalnych, jak i niefunkcjonalnych jest ważne, by dobrze przeanalizować i ocenić pomysł na system przed rozpoczęciem jego wdrażania lub przed całościowym jego wdrożeniem. System może (ale nie musi) być w stanie działać nawet w przypadku niespełnienia postawionych mu wymagań niefunkcjonalnych.

- Bezpieczeństwo — system powinien zapewnić, że wrażliwe dane użytkowników będą odpowiednio zabezpieczone, a płatne kursy nie będą możliwe do wykradnięcia
- Użyteczność — interfejs użytkownika powinien być intuicyjny, ale równocześnie prosty i efektywny w obsłudze. Zarówno uczeń, jak i tutor powinni łatwo zrozumieć i opanować dostępne dla nich funkcjonalności
- Skalowalność — system powinien umożliwiać łatwy rozwój nowych funkcjonalności, bez konieczności wprowadzania dużych zmian do istniejącego systemu.
- Przenaszalność — przeniesienie wdrożenia systemu z jednego środowiska sprzętowego na inne nie powinno stanowić problemu i powinno odbyć się bez przerwy w dostawie usług
- Niezawodność — platforma powinna być dostępna o każdej porze dnia i nocy. Jakość usług (czasy ładowania, brak występowania błędów) musi być zawsze na podobnym, zadowalającym dla użytkownika poziomie.

2.3. Opis aktorów

Jako że aplikacja jest platformą uczenia się na podstawie kursów, istnieją 2 główni aktorzy - trener (Coach) i uczeń (Student). Jedyni aktorzy aplikacji to użytkownicy, gdyż żaden zewnętrzny program nie używa Passed Pawn.

Trener

Trener to użytkownik, który ma zaawansowaną wiedzę szachową i pełni rolę mentora dla uczniów. Aplikacja udostępnia wiele narzędzi dla trenera, które można wykorzystać do budowy przeróżnych kursów szachowych. Użytkownik otrzymuje pełną swobodę — to on decyduje, jak kurs powinien wyglądać oraz z jakich elementów się składać. Trener ma również możliwość pobierania opłat za swoje kursy, ale może też publikować je bezpłatnie.

Najważniejsze funkcjonalności dostępne dla trenera:

- Tworzenie i edycja kursów
- Tworzenie i edycja lekcji
- Tworzenie elementów lekcji
- Zarządzanie i monitorowanie swoich kursów oraz ich zawartości

Cechy charakterystyczne trenera:

- Zaawansowana wiedza o szachach
- Chęć i doświadczenie w nauczaniu innych
- Chęć zarabiania na kursach (opcjonalna)

Uczeń

Uczeń jest użytkownikiem, który używa aplikacji w celu poszerzania swojej wiedzy i umiejętności szachowych. Może to być zarówno kompletny nowicjusz, który dopiero uczy się zasad gry, jak i gracz średnio zaawansowany, pragnący rozwijać swoje umiejętności strategiczne i taktyczne oraz pogłębiać znajomość debiutów, gry środkowej i końcówek. Uczeń korzysta z materiałów przygotowanych przez trenera i ma możliwość kontaktu z mentorem w razie jakichkolwiek wątpliwości.

Najważniejsze funkcjonalności dostępne dla ucznia:

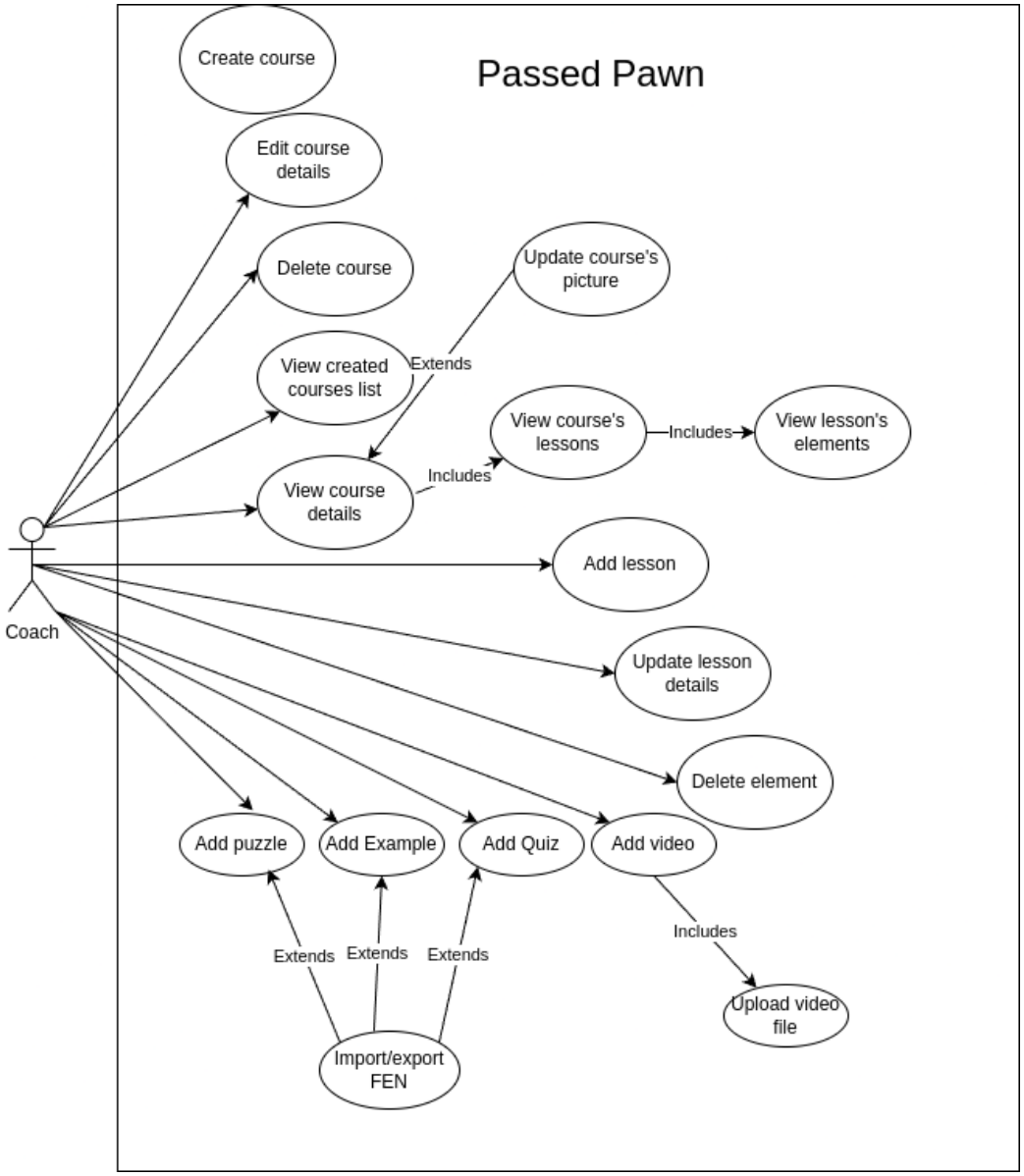
- Przeglądanie i kupno kursów
- Studiowanie na podstawie zawartości kursu
- Tworzenie elementów lekcji (jeśli dozwolone)
- Zarządzanie i monitorowanie swoich kursów oraz ich zawartości

Cechy charakterystyczne uczeń:

- Chęć nauki i poszerzania wiedzy o szachach

2.4. Diagram przypadków użycia trenera

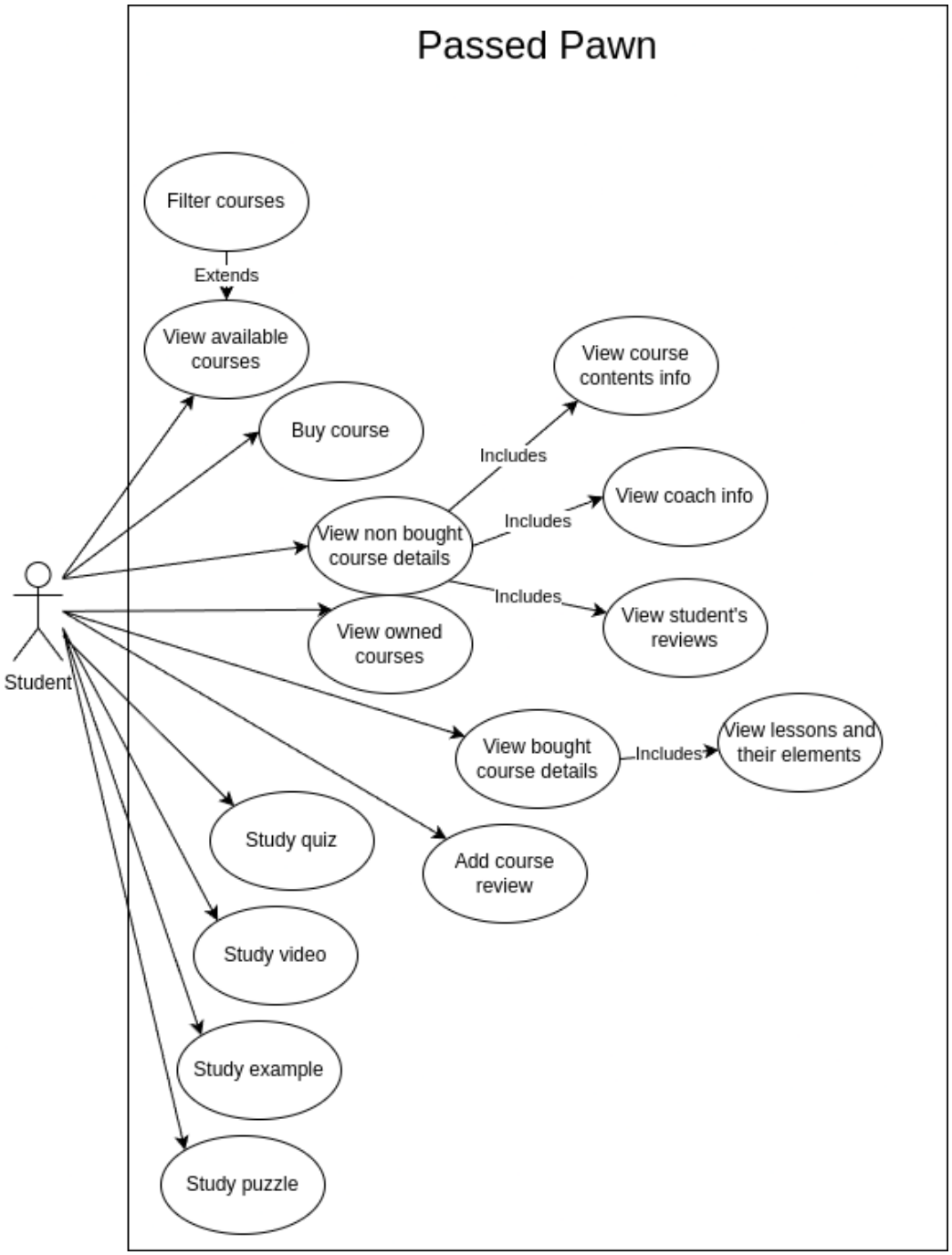
Diagram przedstawia przypadki użycia systemu Passed Pawn z perspektywy trenera, który może tworzyć i zarządzać kursami szachowymi. Trener dodaje lekcje oraz ich elementy (np. quizy, filmy, przykłady), a także otrzymuje możliwość usunięcia ich.



Rysunek 1: Diagram przypadków użycia trenera

2.5. Diagram przypadków użycia ucznia

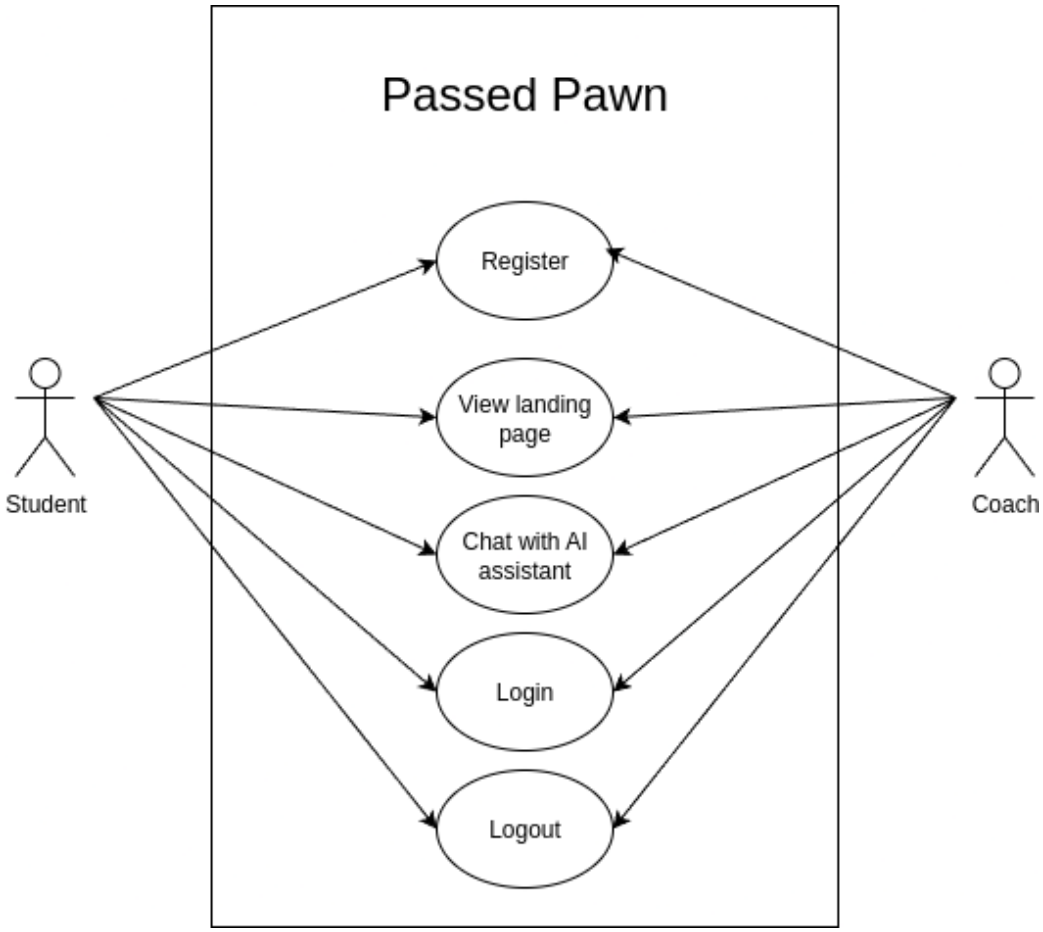
Diagram przedstawia przypadki użycia systemu Passed Pawn z perspektywy ucznia, który może przeglądać i kupować kursy, a także wystawiać opinie po ich ukończeniu. Uczeń ma dostęp do zawartości lekcji (quizów, filmów, przykładów i zadań) oraz informacji o trenerze. System umożliwia też filtrowanie dostępnych kursów i podgląd ich szczegółów przed zakupem.



Rysunek 2: Diagram przypadków użycia ucznia

2.6. Diagram przypadków użycia części wspólnych trenera i ucznia

Diagram przedstawia funkcjonalności, które są dostępne zarówno dla trenera, jak i dla ucznia. Funkcjonalności nie jest za wiele, dotyczą głównie procesów zakładania i używania kont, oraz wbudowanego asystenta AI.



Rysunek 3: Diagram przypadków użycia części wspólnych trenera i ucznia

2.7. Zewnętrzne systemy, na których polega aplikacja

2.7.1. Stripe

Stripe jest platformą do obsługi płatności online, które umożliwia przyjmowanie płatności kartą, przelewami i innymi metodami. Oprócz czytelnego i wygodnego panelu admina Stripe oferuje również narzędzia do wielu technologii, które mocno upraszczają pracę programistom. Passed Pawn używa Stripe w celu otrzymywania płatności z zakupów kursów.

Narzędzia oferowane przez Stripe'a mocno wspomogły projektowanie zarówno aplikacji klienckiej, jak i serwerowej. Na przykład, Stripe jest w stanie wysyłać zapytania do API Passed Pawn za pomocą webhooków, a narzędzia zapewniły bezpieczeństwo i gwarancje, że zapytania nie są wysyłane przez osoby trzecie.

2.7.2. Cloudinary

Cloudinary to usługa chmurowa służąca do zarządzania zdjęciami i filmami. Passed Pawn używa obu z tych oferowanych usług, a Cloudinary zapewnia bezpieczeństwo i wydajność dla tych zasobów.

Narzędzia zapewnianie przez Cloudinary umożliwiły programistom zaprojektowanie całej logiki zapisywania i modyfikowanie zasobów bez potrzeby manualnego korzystania z API. Dotyczy to aplikacji serwerowej oraz klienckiej.

2.7.3. MistralAI

MistralAI jest firmą oferującą dostęp do dużych modeli językowych (LLM) działających w chmurze. Zapytania do tych modeli można wykonywać z użyciem udostępnionego API. W przeciwieństwie do wielu alternatyw MistralAI oferuje bardzo hojne darmowe funkcjonalności, które okazały się wystarczające dla potrzeb aplikacji Passed Pawn.

3. Projekt interfejsu użytkownika

Projektowanie interfejsu użytkownika to podstawa w świecie dzisiejszych aplikacji webowych. Jest to bezpośredni czynnik wpływający na „klikalność" i konwersje aplikacji, które są wyznacznikiem sukcesu biznesu.

Interfejs graficzny powinien być estetyczny, z drugiej zaś strony intuicyjny i wydajny, co może komplikować implementowana logika biznesowa. Zbudowanie spójnej całości to trudne zadanie, dlatego niezwykle ważne jest dokładne przeanalizowanie i zaprojektowanie UI tak, aby generował jak największy ruch na stronie. Rynek aplikacji webowych jest bardzo wymagający. W dorównaniu dzisiejszej konkurencji bardzo istotnym czynnikiem jest wspomniany wygląd interfejsu, jest to przysłowiowa okładka książki, na podstawie której mamy średnio 10 sekund, żeby przekonać użytkownika do pozostania na naszej stronie internetowej. Dopracowane UI to także jakość, którą bezpośrednio prezentujemy użytkownikowi, a co za tym idzie zaufanie, czyli kluczowy element układanki.

3.1. Proces projektowania UI

Projektowanie warstwy wizualnej zaczęło się od zbadania interfejsów konkurencji i wyciągnięcia z nich najlepszych elementów i zabiegów estetycznych, które warto wykorzystać. Z drugiej zaś strony analizowano błędy i aspekty wizualne, których powinno się unikać, aby zainteresować użytkownika produktem. Po ułożeniu i zatwierdzeniu przez zespół wartości i wzorców, którymi będzie się kierować, przyszedł czas na zaprojektowanie ogólnego zarysu strony.

Przed rozpoczęciem wspomnianego 2 etapu ustalono zbiór kluczowych funkcjonalności, tak zwane MVP, które będzie implementowane, aby wydzielić poszczególne elementy i ich ułożenie na stronie. Prace rozpoczęto od ogólnego szkieletu widoków z planszą, a następnie przystąpiono do projektowania mniej istotnych elementów, takich jak komponenty dostępnych kursów oraz profil trenera. Po ukończeniu całego procesu modelowania szkieletu wyłonił się uproszczony zarys strony.

Kolejny etap to dokładne zaprojektowanie pojedynczych komponentów i nałożenie ich na wcześniej przygotowany model, Po zaaplikowaniu całości ukazała się pierwsza wersja gotowego interfejsu, co naturalnie zostało poddane analizie i ewentualnym poprawkom.

W kolejnym etapie skupiono się na doborze ikon i grafik widocznych na stronie. Niszowy charakter rynku aplikacji szachowych wymusił samodzielne zaprojektowanie części ikon, ponieważ gotowe biblioteki UI nie były w stanie ich zapewnić.

Finałowym etapem procesu projektowania UI, po połączeniu ze sobą wszystkich elementów była dogłębna analiza stworzonego modelu. Ocenie podległ końcowy wyglądu interfejsu i przeprowadzono proste testy doświadczenia użytkownika (UX) na modelu strony. Po konsultacji zespołu i wprowadzeniu ostatecznych poprawek oficjalnie ukończono proces projektowania UI. Warto dodać, że na przestrzeni rozwoju aplikacji szachowej projekt UI ulegał lekkim zmianom i poprawkom. W przypadku dodania nowej funkcjonalności konieczne było przygotowanie dla niej wizualnej koncepcji i zintegrowanie z istniejącym modelem.

3.2. Wybrane narzędzia projektowania interfejsu użytkownika

Projekt UI aplikacji szachowej został przygotowany w programie Canva. Jest to popularna, internetowa platforma do tworzenia grafik i projektów wizualnych. Intuicyjność i prostota programu umożliwiła

natychmiastowe wdrożenie zespołu, który miał niewielkie doświadczenia w projektowaniu interfejsów użytkownika.

Praca zaczęła się od projektowania wspomnianych wcześniej niezależnych widoków, co w końcowej fazie złączyło się w całościowy projekt UI. Do realizacji projektu wykorzystano bogatą bibliotekę graficzną Canvy, która pozwala na wydajne wyszukiwanie i zaawansowaną edycję ikonek i wielokolorowych grafik. Na potrzeby aplikacji szachowej stworzono własne spersonalizowane ikonki i grafiki złożone z kształtów geometrycznych i ikon figur szachowych.

Z perspektywy czasu Canva wydaje się nie być najlepszym oprogramowaniem designerskim dla developerów aplikacji webowych. Główną wadą programu jest brak pełnego odwzorowania możliwości kaskadowego arkuszu stylów CSS w dostępnych narzędziach. Canva nie oferuje cieni (odpowiednik box-shadow w CSS) i ma ograniczone możliwości tworzenia gradientów, co znacząco wpłynęło na wydajność tworzonego modelu UI i wymusiło wspieranie się dodatkowym oprogramowaniem.

Alternatywnym rozwiązaniem byłoby użycie programu Figma, internetowego narzędzia do projektowania interfejsów (UI), które umożliwia szybkie tworzenie prototypów stron i aplikacji. W tworzeniu aplikacji internetowych ułatwia współpracę między projektantami a programistami. Proponowane oprogramowanie wypełnia luki narzędziowe, które pozostawiła Canva, umożliwiłoby również dokładniejsze zaprojektowanie modelu i wyłapanie elementów, które wymagały przebudowania już na etapie projektowania, zamiast nanosić poprawki w czasie implementacji klienckiej części aplikacji.

Kolejnym problemem Canvy, jest brak skalowalności. Oferowany szablon dla stron internetowych przypomina ten do tworzenia prezentacji i słabo się sprawdza w projektowaniu UI oprogramowania. Liniowe ułożenie (slajd jeden pod drugim) jest nieczytelne i niewydajne przy większych projektach. Świetnym rozwiązaniem dysponuje Figma, która umożliwia rozwijanie projektu w formie interaktywnej tablicy, umożliwiającej wizualne rozmieszczenie poszczególnych widoków aplikacji, co sprzyja skalowalności i wygodnej nawigacji.

Problemem Figmy jest tempo nauki programu, które jest znacznie wolniejsze, niż w przypadku Canvy, która wyróżnia się prostotą. Kluczowym jednak czynnikiem przy wyborze narzędzia okazała się wcześniejsza styczność z programem Canva, czego nie można powiedzieć o Figmie, w której zespół nie miał doświadczenia. W ostatecznym rozrachunku ciężko oszacować stracony czas przez napotkane ograniczenia narzędziowe Canvy, ponieważ nauka poruszania się i korzystania z Figmy również wymagałaby nieokreślonego nakładu pracy.[3][4]

[3]Canva. *How to Use Canva: A Beginner's Guide*. `https://www.canva.com/learn/how-to-canva-beginners-guide/`. Data dostępu: 2025-06-19. 2025.

[4]Figma. *Przeglądaj właściwości komponentów*. `https://www.canva.com/learn/how-to-canva-beginners-guide/`. Data dostępu: 2025-06-19. 2025.

Tabela 1: Porównanie narzędzi: Canva i Figma

Cecha	**Canva**	**Figma**
Zastosowanie w web designie	Podstawowe – dobre do prostych mockupów lub landing page'y	Profesjonalne narzędzie do tworzenia pełnych UI/UX projektów
Możliwość tworzenia interakcji	Ograniczone (linkowanie między stronami)	Zaawansowane prototypowanie z animacjami, interakcjami i stanami
Precyzja siatki i układu	Ograniczona kontrola, brak zaawansowanych narzędzi siatki	Precyzyjne narzędzia siatek, auto layout, komponenty
Współpraca zespołowa	Możliwa, ale uproszczona	Zaawansowana współpraca w czasie rzeczywistym
Eksport do kodu (np. HTML/CSS)	Brak bezpośredniego wsparcia	Możliwość integracji z narzędziami typu Zeplin, Anima, Figma to Code
Tworzenie komponentów wielokrotnego użytku	Ograniczone możliwości	Pełne wsparcie dla komponentów, wariantów i bibliotek
Responsywność projektów	Brak wsparcia dla responsywności	Obsługa auto layout i skalowania – projektowanie responsywne
Krzywa nauki	Bardzo łatwa, intuicyjna	Średnia – wymaga czasu na opanowanie zaawansowanych funkcji
Grupa docelowa	Początkujący, marketerzy, graficy do prostych projektów	Projektanci UI/UX, developerzy, zespoły produktowe

Tabela została opracowana na podstawie zewnętrznego źródła w postaci artykułu internetowego.[5]

3.3. Ograniczenia bibliotek UI

PrimeNG to świetne rozwiązanie umożliwiające wykorzystanie gotowych komponentów UI w swoim projekcie i pomimo ogromnej bazy jakościowych elementów biblioteka ma swoje ograniczenia.

- Aplikacja szachowa zawiera formularz pozwalający oceniać kursy szachowe. Priorytetem platformy jest przejrzystość i prostota, korzystanie z portalu ma być przyjemnością, dlatego pole formularza dla oceny kursu powinno iść za ideą wygody i prostoty. Niestety biblioteka nie oferuje gotowego komponentu (gwiazdek) do wystawienia oceny, który by umożliwił intuicyjne wystawianie recenzji.
- Pole formularza do wystawiania oceny to połowa sukcesu, potrzebujemy jeszcze komponentu, który będzie reprezentował średnią ocenę kursu w postaci gwiazdek. PrimeNG posiada taki komponent, ale nie obsługuje pół-gwiazdek potrzebnych w dokładniejszym przedstawieniu średniego wyniku, który zazwyczaj jest liczbą zmiennoprzecinkową.

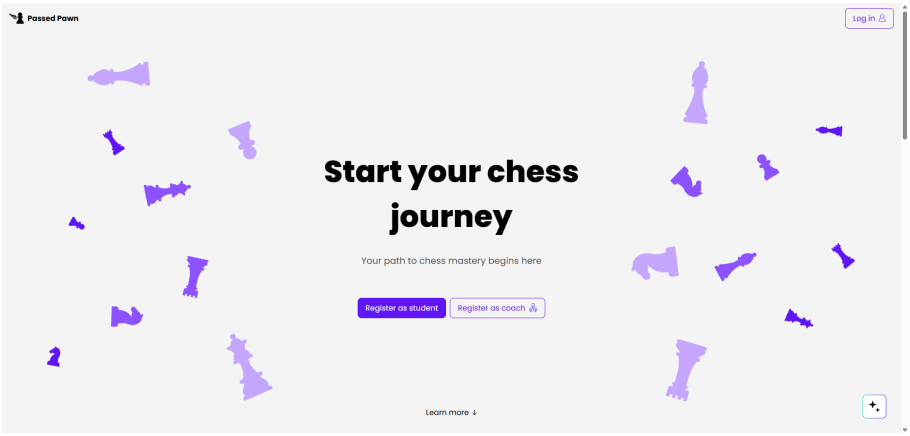
[5]Hitesh Sahni. *Canva vs Figma comparison: which design tool is right for you?* `https://hypegig.com/canva-vs-figma-comparison/`. Data dostępu: 2025-06-19. 2025.

- PrineNG nie posiada również komponentu (pola formularza), który by w czytelny sposób przedstawił poziomu trudności kursu. Tutaj także trzeba się posiłkować własną implementacją takiego elementu.

3.4. Autorskie komponenty i widoki UI

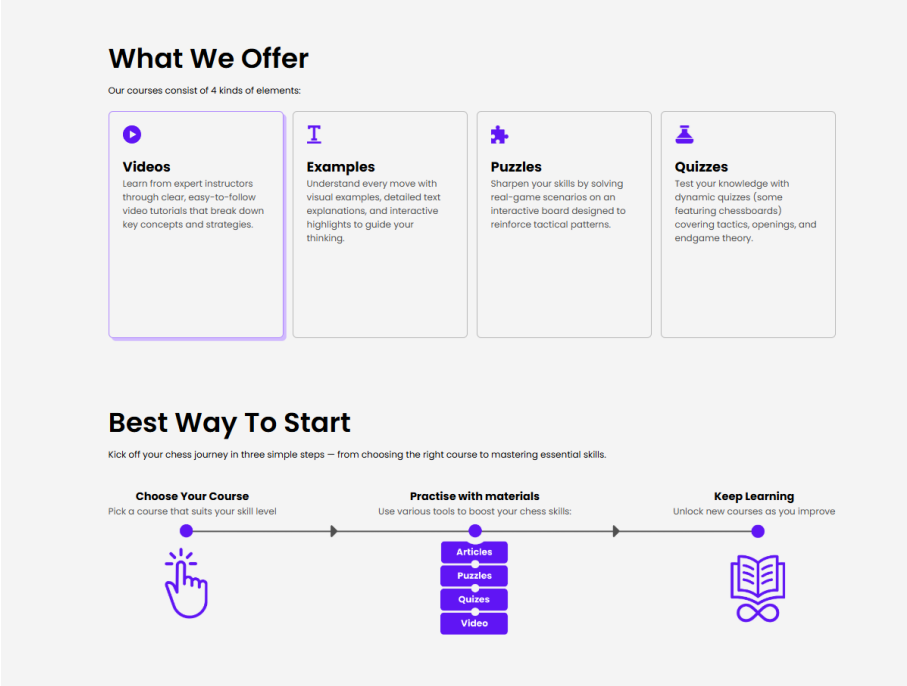
Korzystanie z aplikacji szachowej ma być połączeniem przyjemności z nauką, dlatego interfejs portalu jest możliwie jak najprostszy i intuicyjny. Dobrane kolory i styl elementów na stronie ma zainteresować użytkownika do skorzystania z produktu i pomóc w rozwoju jego umiejętności szachowych. Kluczowym widokiem jest pierwsza strona widziana przez potencjalnego klienta, czyli landing-page. Użytkownik skanuje stronę średnio kilka sekund, następnie podejmuje ważną decyzję czy jest zainteresowany dalszą eksploracją naszego produktu. Celem landing-page jest przedstawienie oferty w szybki i zrozumiały sposób.

Strona główna składa się z tematycznych grafik, treściwego nagłówka i 2 wyróżniających się przycisków (call to action), które mają zwrócić uwagę potencjalnego klienta.



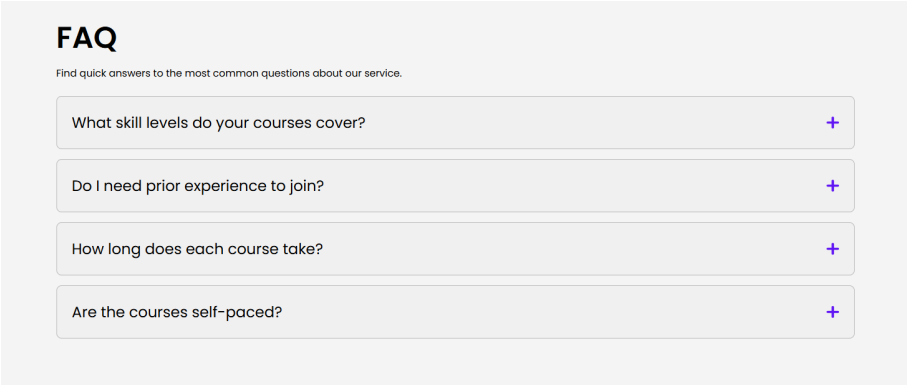
Rysunek 4: Strona startowa (landing page) aplikacji szachowej

Poniżej znajduję się sekcja ukazująca zawartość szkolenia i zalecany model nauki, na podstawie których użytkownik może poznać szczegóły oferowanych usług i strukturę kursów szachowych. Sekcje są minimalistyczne i zawierają niewielką ilość tekstu, zastąpioną ikonkami, aby wizualnie przedstawić proces i nie przytłaczać użytkownika zbędnymi informacjami.



Rysunek 5: Widok oferty szkolenia i proponowany model nauki

Na samym dole znajduje się sekcja pytań zadawanych przez zainteresowanych produktem użytkowników, które poruszają ważne i popularne tematy związane z oferowanym oprogramowaniem i automatyzują komunikację, wymaganą do poznania interesujących użytkownika szczegółów.

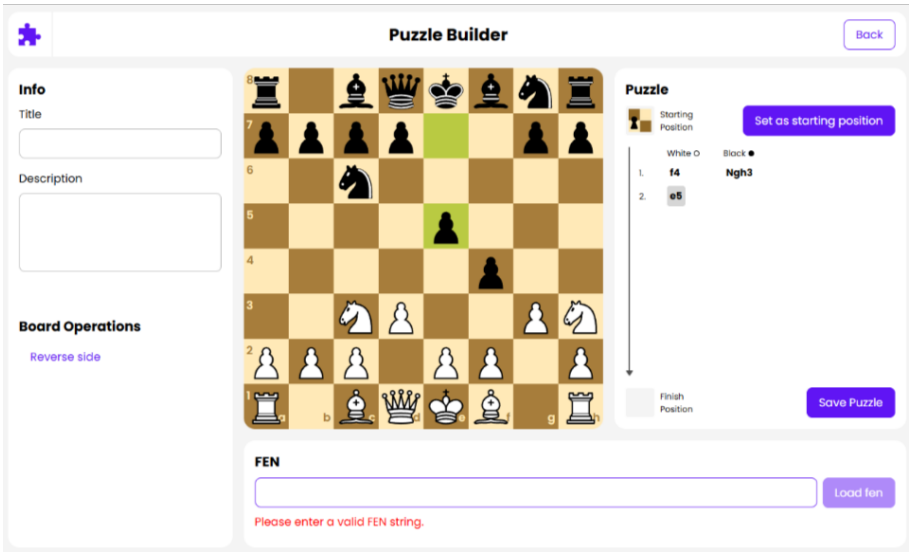


Rysunek 6: Strona startowa, sekcja pytań

Komponent Puzzle Builder — Widok na koncie trenera umożliwiający tworzenie zagadek do kursów szachowych. Komponent podzielony jest na 4 sekcje (logiczne całości):

- **Info i Board Operations** — zawiera pola tekstowe dla podstawowych informacji na temat łamigłówki oraz przycisk umożliwiający wykonywanie operacji na szachownicy.

- **Szachownica** — Centrum, najważniejszy element komponentu, reszta sekcji to podporządkowane narzędzia.
- **Fen** — element do wczytywania FEN na szachownicę w wersji podstawowej (istnieje druga rozbudowana wersja). FEN (Notacja Forsytha-Edwardsa), to notacja, która pozwala przedstawić pozycję na szachownicy w danym momencie gry w postaci tekstowej. Notacja składa się z części reprezentującej rozkład figur oraz części przedstawiającej inne dane jak np. możliwość roszowania, en passant, licznik ruchów itp. Element zawiera pole tekstowe i przycisk, który aplikuje wpisany FEN na planszę. Komponent sygnalizuje użytkownikowi błędny format FEN poprzez widoczną dezaktywację przycisku (przycisk traci kolor).
- **Puzzle** — Najważniejsza podporządkowana sekcja, która bezpośrednio współpracuje z szachownicą. Element prezentuje historię ruchów i pozwala na ustawienie pozycji startowej, jak i końcowej zagadki. Trener ma również możliwość interakcji z historią ruchów (cofanie się do wybranej pozycji). Interfejs opisywanej sekcji jest podzielony na 3 główne elementy ukazujące proces tworzenia łamigłówki. Idąc od góry 1 element to miejsce, w którym ukazany jest stan pozycji startowej. Kwadracik z ikonką oznacza, że pozycja startowa została wybrana, natomiast pusty kwadracik oznacza brak zaaplikowanej pozycji, analogicznie dla elementu ze stanem pozycji końcowej (na samym dole). Pośrodku mamy interaktywną tabelkę z historią wykonanych ruchów wraz ze strzałką ukazującą przebieg procesu tworzenia łamigłówki (wybranie pozycji startowej → ruchy na planszy → wybranie pozycji końcowej). Całość w prosty sposób wizualizuje stan parametrów tworzonej zagadki.



Rysunek 7: Widok komponentu do tworzenia łamigłówek

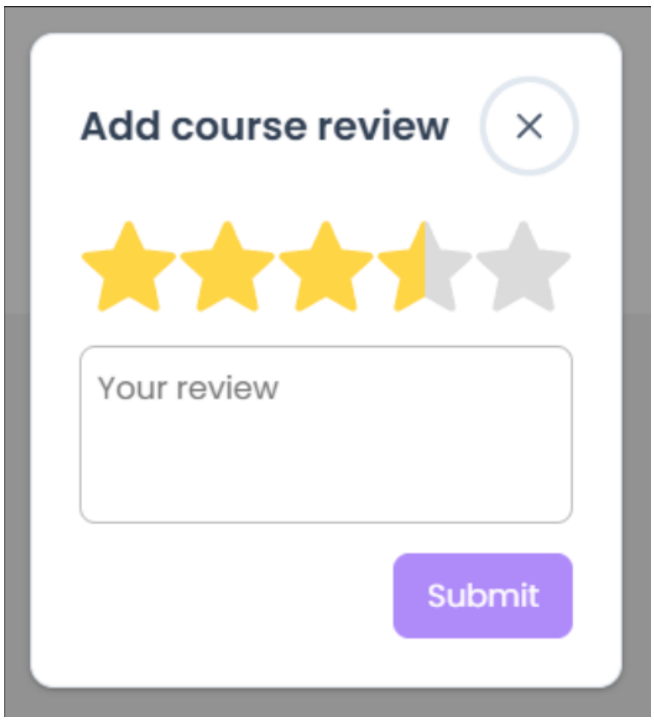
Komponent Quiz/Question and answers — komponent 3 etapu tworzenia quizu dla kursu szachowego znajdujący się w widoku dla trenera. Element podzielony jest na 4 (logiczne całości) sekcje:

- **Board Pieces i Board Operations** — zawiera listę figur, które można przeciągnąć na planszę oraz zestaw przycisków umożliwiający wykonywanie operacji na szachownicy. Sekcja skupia narzędzia, które bezpośrednio oddziałują z szachownicą i jej figurami.
- **FEN** — rozbudowana wersja komponentu FEN, wzbogacona o panel umożliwiający konfigurację notacji. Sekcja zawiera pola formularza własnej implementacji bez użycia biblioteki PrimeNG (pole typu checkbox).
- **Szachownica** — Centrum komponentu, reszta sekcji to podporządkowane narzędzia.

- **Quiz i Addings** — jest podzielony na 2 zakładki. Pierwsza z nich to Quiz, zawierający pola formularza — pytanie i odpowiedzi, które są wysoce interaktywne. Trener ma możliwość usuwania (czerwony kolor przycisku podkreśla krytyczność operacji), edytowania i przypisywania planszy do odpowiedzi (podobnie jak w przypadku puzzle builder). Kwadracik informuje o stanie odpowiedzi (czy ma przypisaną planszę). Z lewej strony panelu odpowiedzi znajdują się pola formularza (radio input), które trener zaznacza przy wyborze poprawnej odpowiedzi. Druga zakładka to Addings zawierająca pola tekstowe z wyjaśnieniem rozwiązania i podpowiedzią. Zawartość zakładek rozdziela narzędzia na bardziej istotne i częściej używane — Quiz oraz mniej istotne — Addings. Podział tworzy intuicyjny i wydajny interfejs dla trenera.

Rysunek 8: Widok komponentu tworzenia quizu, 3 etap

Komponent pola wejściowego wykorzystywany w formularzu do wystawiania i edycji opinii użytkownika na temat kursu szachowego. Biblioteka PrimeNG nie posiada gotowego odpowiednika. Intuicyjny interfejs pozwala na proste i szybkie wystawianie oceny. Użytkownik po najechaniu na gwiazdkę zobaczy odpowiadającą mu ocenę, która po kliknięciu zostanie zatwierdzona. Wystawianie oceny jest możliwe z dokładnością do pół gwiazdki. Podczas edycji oceny komponent wyświetla wcześniej zatwierdzoną wartość, jednocześnie umożliwiając zmianę oceny na nową.



Rysunek 9: Komponent do wystawiania recenzji kursu

Komponent prezentuje wystawioną ocenę kursu szachowego. Biblioteka PrimeNG oferuje gotowy element, który nie wspiera połowicznych gwiazdek używanych w aplikacji. W związku z tym w aplikacji szachowej zaimplementowano własny komponent, umożliwiający wyświetlanie ocen z dokładnością do pół gwiazdki. Dzięki temu możliwa jest pełna integracja systemu wystawiania i prezentowania recenzji.

Rysunek 10: Komponent prezentujący średni ocenę kursu

Komponent pola wejściowego dla poziomu trudności kursu szachowego. PrimeNG nie oferuje w swojej kolekcji odpowiedniego elementu spełniającego te wymagania. Komponent jest wykorzystywany w wyszukiwarce kursów szachowych, gdzie użytkownik ma możliwość filtrowania wyników na podstawie poziomu trudności, przedstawionego jako przedział wartości. Element umożliwia łatwe określanie granic tego zakresu, z możliwością ich edycji. Moduł został wyposażony w przycisk do resetowania ustawień, ponieważ edycja pozwala jedynie na rozszerzanie zakresu poziomu trudności, bez możliwości jego zawężenia. Na dole znajduje się fioletowy tekst informujący użytkownika o wybranym zakresie (w tym przypadku [od początkującego do eksperta])

Rysunek 11: Komponent pola wejściowego dla poziomu trudności kursu

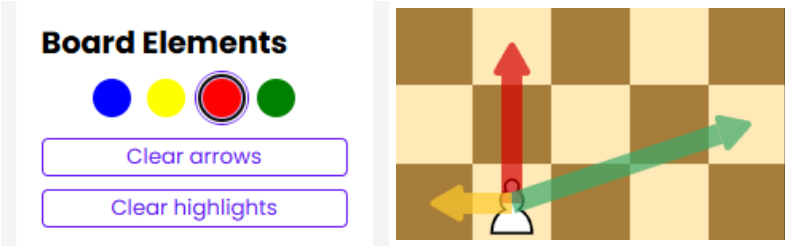
Diagram przedstawiający zawartość kursu szachowego znajdujący się na stronie szczegółowej. Komponent wizualizuje ilość poszczególnych elementów szkolenia poprzez długość odpowiedniego słupka wykresu. Diagram przedstawia dane w skali względnej, w której słupek o największej wartości stanowi punkt odniesienia (100% skali), a pozostałe słupki są proporcjonalnie do niego dostosowane. Wykres informuje użytkownika o różnorodności materiałów i pozwala ocenić atrakcyjność szkolenia.

Rysunek 12: Komponent, diagram przedstawiający zawartość kursu szachowego

Interaktywne strzałki na szachownicy, pozwalające wizualizować ruchy figur. Występują w widoku do tworzenia przykładów (1 z 4 atomowych elementów kursu), dostępne w 4 kolorach do dyspozycji trenera. Strzałka tworzona jest poprzez przytrzymanie prawego przycisku myszy, przeciągnięcie kursora do pola docelowego i zwolnienie przycisku. Aby usunąć konkretną strzałkę, należy wykonać ten proces ponownie, natomiast wyczyszczenie planszy ze wszystkich strzałek jest możliwe dzięki przyciskowi „Clear arrows".

Biblioteka PrimeNg nie oferuje implementacji wspomnianego komponentu, nawet jeśli istniałaby gotowy model, zintegrowanie go z niezależnie stworzoną szachownicą byłoby problematyczne.

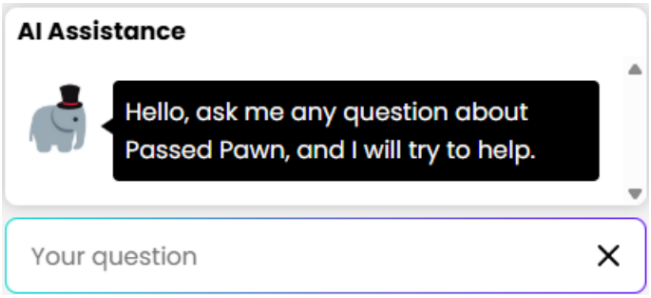
Mechanizm wyświetlania strzałek bazuje na prostych obliczeniach na wektorach w układzie kartezjańskim. Znając współrzędne [X, Y] 2 punktów (punkt startowy i docelowy) jesteśmy w stanie wyliczyć długość i kont padania strzałki na szachownicy.



Rysunek 13: Panel obsługi oraz system interaktywnych strzałek na szachownicy

Asystent AI, rozwijany komponent umożliwiający komunikację ze słonikiem Bobem (maskotka platformy szachowej), w celu uzyskania pomocy w obsłudze i poruszaniu się po aplikacji. Bob jest pomocnikiem, który odpowiada na pytania klientów. AI jest dostępne dla zalogowanych i niezalogowanych użytkowników w prawym dolnym rogu aplikacji szachowej.

Interfejs komponentu składa się z czatu, w którym wyświetlana jest konwersacja ze sztuczną inteligencją i pola tekstowego do zadawania pytań. Komponent pozwala użytkownikowi na kontrolę widoczności treści poprzez rozwijanie i zwijanie.



Rysunek 14: Asystent AI

3.5. Znaczenie oryginalnego projektu interfejsu użytkownika

Tworząc stronę internetową, zespół deweloperski skupia się na tworzeniu przejrzystej, wpisującej się w zasady czystego kodu (clean code) implementacji projektu. Z punktu widzenia deweloperów, którzy odpowiadają za rozwój projektu, jest to bardzo istotne, gdyż wpływa na wydajność ich pracy.

Jednak, jeśli spojrzymy na aplikacje z perspektywy klienta, kluczowy jest design, który jest odpowiedzialny za intuicyjność użycia i przejrzystości informacji. Użytkownik nie ocenia jakości kodu aplikacji serwerowej, nie ma ona dla niego żadnego znaczenia. Hierarchię priorytetów dla klienta strony internetowej można uprościć do 2 głównych aspektów: Aplikacja wspiera użytkownika w realizacji jego celów oraz umożliwia ich osiągnięcie w sposób szybki, intuicyjny i przyjazny.

Oryginalny design to wizytówka produktu, która wyróżnia go na rynku rosnącej konkurencji. Wygląd i sposób działania interfejsu jest głównym czynnikiem wpływającym na to, jak użytkownik zapamięta stronę. Kolejnym aspektem oryginalności jest minimalne wykorzystanie zewnętrznych bibliotek UI. Zaletą takiego podejścia oprócz niepowtarzalnego interfejsu, jest pełna niezależność i swoboda twórcza. Zespołu nie ogranicza zestaw gotowych częściowo konfigurowalnych komponentów używanych w tysiącach innych aplikacji.

Brak zewnętrznych bibliotek to również brak dodatkowych ciężkich zależności, które generują nadmiarowy kod i utrudniają optymalizację. Zaletą oryginalności naszego projektu jest elastyczność w modyfikacjach. Deweloperzy nie są ograniczani przez możliwości wspomnianych bibliotek UI i mogą się bez problemu dostosować do potrzeb użytkownika.

Tabela 2: Porównanie: samodzielny projekt UI vs gotowa biblioteka UI

Kryterium	**Samodzielny projekt UI**	**Gotowa biblioteka UI**
Wyjątkowość i oryginalność	W pełni unikalny wygląd i styl	Ograniczona indywidualność, szablonowy wygląd
Dopasowanie do potrzeb projektu	Maksymalna elastyczność i kontrola	Ograniczone do dostępnych opcji i konfiguracji
Wydajność i rozmiar paczki	Mniej zależności, lżejszy kod	Większy bundle przez nadmiar nieużywanego kodu
Szybkość wdrożenia (MVP)	Wymaga więcej czasu i pracy	Szybsze tworzenie prototypów
Skalowalność i modyfikacja	Łatwe dostosowanie do zmian i rozwoju projektu	Modyfikacje często trudne lub wymuszające obejścia
Wymagania techniczne / doświadczenie	Wymaga znajomości CSS, UX i front-endu	Niższy próg wejścia dla dewelopera
Spójność designu (zespół/większy projekt)	Trzeba samodzielnie dbać o standaryzację	Wbudowana spójność komponentów
Wsparcie społeczności / dokumentacja	Brak – tworzysz sam	Silne wsparcie i gotowe przykłady

3.6. Problemy i stracone zasoby

Decyzje dotyczące projektu zawsze podlegały dyskusji i walidacji przez wszystkich członków zespołu, co skutecznie podnosiło szanse na podejmowanie słusznych i racjonalnych kroków w kwestii dalszego rozwoju projektu.

Niestety zaprojektowanie interfejsu użytkownika w wersji 2.0 nie było uzasadnionym działaniem i przyczyniło się do utraty dziesiątek godzin pracy, które zespół powinien przeznaczyć na budowę kluczowych funkcjonalności. Projekt zakończył się na rozwoju tylko modelu landing-page z „wystylowanymi" komponentami i nowo dobranymi kolorami, zamieniając monochromatyczny fioletowy styl na nowe kolory (niebieski, pomarańczowy i czerwony). Całość modelu bardzo dobrze się prezentuje i spełnia postawione jej założenia. Wersja 2.0 będzie rozbudowywana w przyszłości i docelowo zastosowana w aplikacji szachowej, lecz nie zmienia to faktu, że podjęcie się tej pracy w okresie intensywnego rozwoju MVP aplikacji było nieprzemyślane.

Kolejnym błędem było projektowanie interfejsu użytkownika w słabo przystosowanym narzędziu, jakim jest Canva. Aplikacja Melanie Perkins jest słabo przystosowana do modelowania UI stron internetowych, ponieważ nie jest w stanie odzwierciedlić możliwości CSS (gradient i cienie). Wraz z rosnącym rozmiarem projektu spada efektywność i prostota poruszania się po aplikacji. Podczas rozwoju aplikacji szachowej zauważono również spadek wydajności Canvy (aplikacja się zacinała i potrzebowała dużo czasu na wczytanie projektu).

Następną komplikacją było wykorzystanie prostych komponentów PrimeNG, które zespół był w stanie samodzielnie zaimplementować (przyciski i niektóre pola formularza). Wybór biblioteki UI był uzasadniany zaoszczędzeniem czasu na implementacji dostępnych już komponentów, co poskutkowało utratą cennych minut na próbę modyfikacji gotowych elementów i w ostateczności zamianę na własnoręcznie napisany kod.

Podsumowując, większość napotkanych problemów dotyczyła utraty jednego zasobu, jakim jest czas, co wiązało się z wydłużeniem budowy aplikacji szachowej.

4. Architektura Passed Pawn

Architektura w odniesieniu do projektów IT może oznaczać bardzo różne rzeczy i można wymienić różne rodzaje architektur projektowanych w ramach danego przedsięwzięcia. W związku z tym należałoby przytoczyć ogólną definicję samego pojęcia architektury, która pozwoli zrozumieć uniwersalny sens, znaczenie i cechy, które łączą różne rodzaje architektur. Dr hab. inż. Jan Werewka, prof. WSEI definiuje architekturę jako "[...] sztuka i nauka planowania, projektowania oraz organizowania przestrzeni, struktur lub systemów, tak aby spełniały określone wymagania funkcjonalne, estetyczne i techniczne."[6] Profesor Werewka wyszczególnia przy tym następujące, najważniejsze cechy podstawowe architektury[7]:

- **Struktura**: Sposób, w jaki elementy systemu są rozmieszczone i jak są ze sobą powiązane.
- **Bloki konstrukcyjne (Building Blocks)**: W każdym systemie wyróżnia się jednostki pełniące określone funkcje przy budowie całego systemu.
- **Zasady projektowe**: Zestaw zasad panujących nad procesem projektowania i rozwoju, monitorowaniem i utrzymaniem zbudowanego obiektu.
- **Cele**: Architektura ma określone cele, którymi są: funkcjonalność, estetyka, skalowalność, wydajność, bezpieczeństwo, zgodność z wymaganiami użytkowników, itp.

W kontekście IT wyróżnia on natomiast następujące rodzaje architektur, które pomogą precyzyjnie określić rodzaj omawianej w punkcie architektury[8]:

- **Architektura Biznesowa (Business Architecture)** — Dotyczy struktury biznesowej organizacji, w tym procesów biznesowych, organizacji pracy, modeli biznesowych i produktów. Wyznacza cele biznesowe, motywację i strategię działania.
- **Architektura Informacyjna (Information Architecture)** — Koncentruje się na zarządzaniu i przepływie informacji w organizacji, obejmując modelowanie, organizację i cykl życia danych.
- **Architektura Aplikacji (Application Architecture)** — Określa sposób projektowania, integracji i zarządzania aplikacjami w organizacji, w tym podział na warstwy (prezentacja, logika biznesowa, dane).
- **Architektura Oprogramowania (Software Architecture)** — Skupia się na strukturze i organizacji kodu, obejmując podział na moduły, klasy, komponenty oraz wzorce projektowe (np. mikroserwisy, MVC).
- **Architektura Danych (Data Architecture**): Obejmuje strukturę, zarządzanie i przechowywanie danych w organizacji, w tym modele danych i strategie przechowywania (np. Big Data).
- **Architektura Technologiczna (Technology Architecture)** — Definiuje infrastrukturę technologiczną, w tym sprzęt, oprogramowanie i sieci, potrzebne do realizacji celów organizacji.
- **Architektura Integracji (Integration Architecture)** — Dotyczy sposobów integracji systemów, aplikacji i danych w organizacji, obejmując różne narzędzia i metodyki (np. middleware, ESB).
- **Architektura Systemu (System Architecture)**: Obejmuje projektowanie i organizację całych systemów informatycznych, ich komponentów i sposobu integracji.
- **Architektura Bezpieczeństwa (Security Architecture)** — Skupia się na zabezpieczeniu informacji, systemów i procesów w organizacji poprzez użycie polityk bezpieczeństwa i mechanizmów ochronnych.
- **Architektura Sieciowa (Network Architecture)** — Zajmuje się projektowaniem i zarządzaniem infrastrukturą sieciową, umożliwiającą komunikację między systemami i urządzeniami (np. LAN, WAN).
- **Architektura Chmury (Cloud Architecture)** — Obejmuje projektowanie i organizację zasobów IT w chmurze, w tym zarządzanie usługami SaaS, PaaS i IaaS oraz integracją chmury z lokalną infrastrukturą.

[6] prof. WSEI dr hab. inż. Jan Werewka. *Rola i znaczenie architektury w obszarze IT*. `https://magazyn.wsei.edu.pl/rola-i-znaczenie-architektury-w-obszarze-it/`. Data dostępu: 2025-06-21. 2024.

[7] [Tamże].

[8] [Tamże].

Praca skupia się na analizie technicznej projektu Passed Pawn, więc Architektura Biznesowa oraz Informacyjna nie są przedstawione w pracy.

4.1. Przykłady powszechnie stosowanych rodzajów architektur systemów

4.1.1. Monolit

Architektura monolityczna stanowi tradycyjne podejście, w którym cały system jest rozwijany i wdrażany jako pojedyncza jednostka. Wszystkie komponenty, takie jak interfejs użytkownika, logika biznesowa i warstwa danych, są ze sobą ściśle powiązane i współdzielą bazę kodu. Kluczowym elementem architektury monolitycznej jest scentralizowanie wszystkich funkcji systemu w pojedynczej aplikacji.

Cechy charakterystyczne monolitu

- **Zwartość** — mimo podziału na moduły, całość aplikacji jest wciąż wdrażana jako pojedyncza aplikacja, która wymaga koordynacji rozwoju modułów
- **Modularność** — dobrze zbudowany monolit jest podzielony na logiczne, niezależne moduły, które mają jasno zdefiniowane interfejsy
- **Luźne powiązanie modułów** — w dobrze zaprojektowanym monolicie zmiany w jednym module mają minimalny wpływ na pozostałe, co ułatwia zarządzanie i rozwój systemem

Zalety

- **Proste wdrożenia** — istnienie pojedynczego artefaktu wdrożeniowego ułatwia wdrażanie systemu
- **Ułatwione testowanie** — skupienie kodu w jednym miejscu i pojedynczej jednostce, którą wystarczy uruchomić, może przesądzać o łatwym testowaniu małych i średnich wielkością systemów
- **Ułatwiona analiza** — skupienie kodu w jednym miejscu i spójnym spektrum technologicznym ułatwia analizę małych i średnich wielkością systemów
- **Niski narzut wydajnościowy** — brak występowania opóźnień komunikacyjnych między modułami, co wynika z komunikacji wewnątrzprocesowej, zamiast np. sieciowej

Wady

- **Kosztowne i nieelastyczne skalowanie** — skalowanie pojedynczych modułów jest niemożliwe albo utrudnione, więc domyślnym sposobem skalowania jest skalowanie całości, co może być niepożądane i generować niepotrzebne koszty
- **Zwiększona podatność i kaskadowość awarii** — błąd aplikacji w jednym module może wpłynąć na cały system
- **Siska elastyczność** — mniejsza elastyczność w doborze technologii dla poszczególnych części aplikacji

4.1.2. SOA (Service-Oriented Architecture)

SOA (Service Oriented Architecture), czyli Architektura Zorientowana na Usługi, to podejście do projektowania systemu, w którym system budowany jest jako zestaw luźno powiązanych i autonomicznych aplikacji/usług. W zamyśle każda usługa realizuje konkretne zadanie biznesowe, a komunikacja odbywa się za pośrednictwem standardowych protokołów i stylów komunikacji sieciowej, takich jak SOAP lub REST. Kluczowym aspektem tej architektury jest możliwość wielokrotnego wykorzystania funkcjonalności zawartych w serwisach.

Cechy charakterystyczne

- **Modularność i reużywalność** — usługi są projektowane jako niezależne moduły, które można ponownie wykorzystywać w różnych aplikacjach i kontekstach
- **Autonomia i elastyczność** — zmiany w jednej usłudze nie powinny wymagać modyfikacji innych, co umożliwia niezależne wdrażanie i zapewnia elastyczność
- **Hermetyzacja szczegółów implementacyjnych** — implementacja każdej usługi jest ukryta, a konsumentom ujawniany jest jedynie interfejs, co pozwala chociażby na dobór różnych technologii i języków programowania do potrzeb serwisu
- **Kompozycja zamiast integracji** — aplikacje są tworzone poprzez kompozycję autonomicznych usług

Zalety

- **Elastyczność i skalowalność** — ułatwione dodawanie, modyfikowanie i usuwanie usług bez zakłócania działania systemu, co pozwala na dostosowanie do zmieniających się wymagań biznesowych. Usługi mogą być wdrażane i skalowane niezależnie
- **Ponowne wykorzystanie komponentów** — skrócenie czasu i kosztów rozwoju dzięki wykorzystaniu istniejących, sprawdzonych komponentów.
- **Łatwa integracja** — ułatwia łączenie różnorodnych systemów wewnętrznych i zewnętrznych dzięki ustandaryzowanym protokołom komunikacyjnym
- **Zmniejszenie złożoności** — dzielenie systemów na mniejsze usługi obniża koszty utrzymania i zwiększa przejrzystość
- **Odporność i niezawodność** — awaria jednej usługi nie musi wpływać na cały system, co zwiększa odporność na awarie
- **Bezpieczeństwo** — możliwość szczegółowego zarządzania bezpieczeństwem na poziomie usług, co jest kluczowe w aplikacjach rozproszonych

Wady

- **Złożoność zarządzania** — wymaga odpowiedniego nadzoru architektonicznego i efektywnego modelu operacyjnego[9]
- **Większe wymagania infrastrukturalne** — Potrzeba infrastruktury do zarządzania usługami i ich komunikacją
- **Zwiększone opóźnienia sieciowe** — komunikacja między usługami obarczona jest dodatkowym narzutem związanym z przesyłaniem informacji po sieci
- **Trudność w odpowiedniej implementacji** — źle przemyślane, nieorganiczne i źle zarządzane kontrakty i granice serwisów mogą zmniejszać ich aspekt autonomiczności, jednocześnie pozostawiając wady w postaci większego skomplikowania w zarządzaniu oraz dodatkowego narzutu sieciowego wpływającego negatywnie na wydajność

4.1.3. Mikroserwisy

Architektura mikroserwisów polega na budowaniu systemu jako zbioru małych, niezależnych usług/aplikacji, które najczęściej komunikują się ze sobą poprzez sieć i mają wyraźnie zdefiniowane logiczne oraz fizyczne granice. Kluczowym elementem w tej architekturze jest autonomia i niezależność pojedynczych mikroserwisów, co pozwala na ich samodzielne wdrażanie bez koordynacji z innymi mikroserwisami.

Cechy charakterystyczne mikroserwisów

[9]Marcin Sobieraj. *Service Oriented Architecture - SOA*. `https://drogaarchitektait.pl/2020/08/service-oriented-architecture-soa/`. Data dostępu: 2025-06-21. 2020.

- **Niezależność i autonomia** — każdy mikroserwis jest z założenia niezależny i może być rozwijany, wdrażany i skalowany oddzielnie
- **Jedna odpowiedzialność** — każdy mikroserwis jest z założenia wyspecjalizowany do osiągnięcia pojedynczego celu o dobrze zdefiniowanym zakresie
- **Własna baza danych** — w architekturze mikroserwisowej, mikroserwisy często posiadają swoje własne bazy danych, co jest związane z dążeniem do jak największej niezależności od innych części systemu, innych mikroserwisów
- **Lekkie mechanizmy komunikacji** — mikroserwisy zazwyczaj komunikują się ze sobą lekkimi kanałami komunikacyjnymi takimi jak REST, gRPC lub poprzez kolejki komunikatów (Message Queues) i brokerów zdarzeń (Event Brokers) np. RabbitMQ, czy Apache Kafka, żeby jak najbardziej zniwelować narzut wydajnościowy związany z komunikacją sieciową i zredukować sprzężenie między mikroserwisami (tyczy się wykorzystania kolejek komunikatów i brokerów zdarzeń)
- **Technologiczna różnorodność** — zespoły mogą wybierać różne technologie i języki programowania do pracy z poszczególnymi mikroserwisami

Zalety

- **Skalowalność** — poszczególne mikroserwisy mogą być skalowane niezależnie w zależności od zapotrzebowania
- **Elastyczność i szybkość rozwoju** — niezależne zespoły mogą pracować nad różnymi mikroserwisami, co ułatwia rozwój dużych, złożonych systemów, które inaczej musiałyby koordynować wiele zespołów i prace dziesiątek albo nawet setek programistów
- **Odporność na awarie** — awaria jednego mikroserwisu nie wpływa na cały system
- **Swoboda technologiczna** — możliwość wyboru najlepszej technologii dla każdego mikroserwisu
- **Łatwiejsze utrzymanie** — mniejsze i bardziej wyspecjalizowane komponenty są łatwiejsze do zrozumienia i utrzymania

Wady

- **Złożoność zarządzania** — zarządzanie dużą liczbą niezależnych mikroserwisów może znacząco utrudniać analizę systemu jako całości i wprowadza całą gamę problemów związanych z ruchem sieciowym, co może być nieopłacalne przy małej i średniej wielkości projektach
- **Trudność w zachowaniu spójności danych** — zarządzanie spójnością danych w rozproszonym systemie może być znacznie utrudnione i wymaga często zastosowania skomplikowanych wzorców architektonicznych, np. systemu rozproszonych transakcji
- **Zwiększone opóźnienia sieciowe** — komunikacja między mikroserwisami obarczona jest dodatkowym narzutem związanym z przesyłaniem informacji po sieci
- **Trudność w odpowiedniej implementacji** — źle przemyślane, nieorganiczne granice mikroserwisów mogą eliminować ich aspekt niezależności i autonomiczności jednocześnie pozostawiając wady w postaci większego skomplikowania w zarządzaniu, debugowaniu oraz dodatkowego narzutu sieciowego wpływającego negatywnie na wydajność

4.1.4. Big Ball of Mud - antywzorzec

Big Ball of Mud (Wielka kula błota) to antywzorzec architektury systemu, który opisuje system pozbawiony wyraźnej struktury, składający się z połączonych w nieprzemyślany sposób komponentów bez zachowania wysokich standardów i świadomych konwencji w kodzie aplikacji. Kluczowym elementem tego typu architektury jest brak przemyślanego projektu i stosowanych konwencji, co prowadzi do chaosu i trudności w zarządzaniu.

Cechy charakterystyczne Big Ball of Mud

- **Brak jasnej struktury** — brak logicznego podziału na moduły i warstwy
- **Silnie splątany i sprzężony kod**: między wieloma miejscami w kodzie i jego zależnościami występują silne powiązania, co utrudnia jego analizę i modyfikację
- **Niewyraźne granice komponentów** — funkcjonalności są rozproszone w wielu miejscach, bez jasnego przypisania do konkretnych komponentów
- **Brak dokumentacji** — zazwyczaj brak lub niewystarczająca dokumentacja systemu.
- **Zależności cykliczne** — komponenty wzajemnie od siebie zależą, co utrudnia testowanie i utrzymanie

Dlaczego Big Ball of Mud to antyprzykład

- **Trudności w utrzymaniu i rozwoju** — Modyfikacje i rozszerzanie kodu są ryzykowne i wiążą się z wysokim prawdopodobieństwem zepsucia istniejących funkcjonalności.
- **Słaba skalowalność** — brak odpowiedniego ustrukturyzowania utrudnia efektywne skalowanie systemu.
- **Niska jakość kodu** — niespójność, duplikacja kodu i brak standardów obniżają jakość oprogramowania.
- **Mała elastyczność i utrudniona refaktoryzacja** — z powodu silnego sprzężenia części kodu i braku przemyślanej struktury, refaktoryzacja i dostosowanie systemu do zmieniających się potrzeb jest znacznie utrudnione i obarczone zwiększonym ryzykiem zepsucia istniejących funkcjonalności
- **Wysoki koszt operacyjny** — przedstawione wady systemu skutkują dużym kosztem utrzymania i rozwoju systemu

Architektura / Cecha	Monolit	SOA	Mikroserwisy	Big Ball of Mud
Struktura	Pojedyncza, zintegrowana aplikacja	Zbiór usług, które mogą być używane i ponownie wykorzystywane do wspierania aplikacji	Małe, niezależne usługi, każda skupiona na jednej funkcji biznesowej	Brak wyraźnej struktury, chaotyczny, mocno sprzężony i splątany kod
Rozwój	Prosty przy małej skali projektu rozwój, testowanie i wdrażanie	Bardziej złożony niż monolit, prostszy niż mikroserwisy	Złożony, wymaga koordynacji między usługami	Trudny i ryzykowny, brak modularności utrudnia zmiany
Skalowalność	Skaluje się jako całość	Usługi mogą być skalowane niezależnie	Usługi mogą być skalowane niezależnie	Bardzo niska, system trudny do rozdzielenia i skalowania
Wdrażanie	Łatwe wdrożenie, pojedynczy artefakt wdrożeniowy	Wymaga starannego zarządzania wersjami usług i zależnościami	Możliwe ciągłe wdrażanie, wymaga narzędzi orkiestrujących	Brak wyraźnego procesu wdrażania, często chaotyczne
Zestaw technologiczny	Jednolity w całej aplikacji	Może się różnić między usługami, ale rzadziej ze względu na wyzwania integracyjne	Może się znacznie różnić między usługami	Brak spójności, mieszanka technologii bez kontroli
Zarządzanie danymi	Scentralizowana baza danych	Współdzielone bazy danych lub osobne bazy dla każdej usługi	Osobna baza danych dla każdej usługi, promuje decentralizację danych	Brak jasnych granic, dane często globalne lub duplikowane
Komunikacja	Wywołania metod wewnątrz aplikacji	Komunikacja między usługami przez sieć (SOAP, REST)	Komunikacja między usługami często przez REST lub systemy kolejkowania komunikatów	Brak dobrze utrzymanych interfejsów, kod silnie sprzężony i splątany

Tabela 3: Porównanie architektur

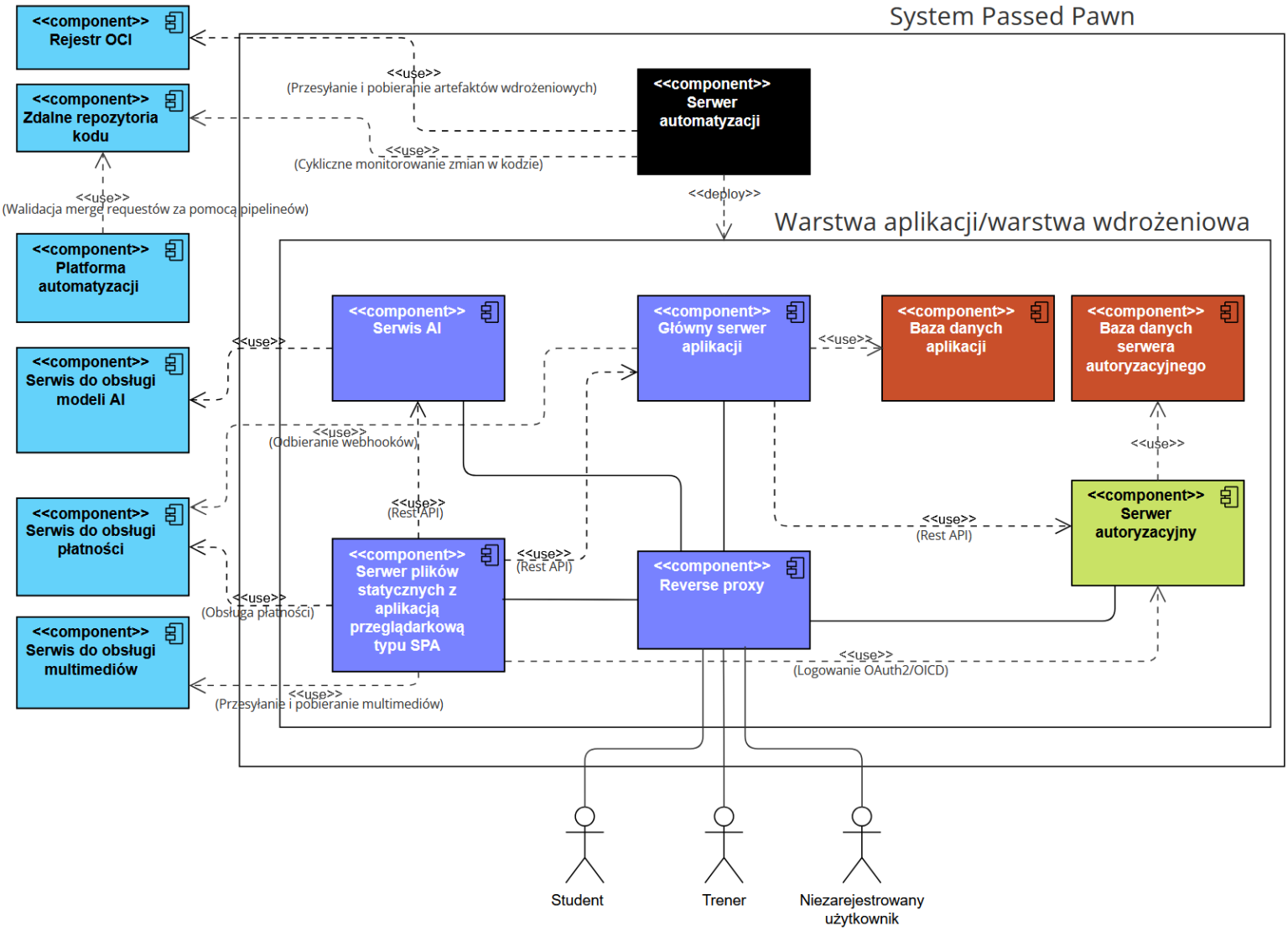
Tabela została opracowana na podstawie zewnętrznego źródła w postaci artykułu internetowego.[10]

[10]Pramitha Jayasooriya. *Monolithic vs. SOA vs. Microservices Architecture: A Java Perspective*. `https://medium.com/@lpramithamj/monolithic-vs-soa-vs-microservices-architecture-a-java-perspective-6d3d9fb26ac7`. Data dostępu: 2025-06-21. 2024.

4.2. Wybór i projekt architektury Passed Pawn

Architekturę systemu Passed Pawn można scharakteryzować jako leżącą gdzieś pomiędzy typową architekturą SOA a architekturą monolityczną. W miarę trafnym jest prawdopodobnie nazwanie jej **architekturą opartą o serwisy/usługi (ang. Service Based Architecture - SBA)** w związku z jej serwisowym/usługowym charakterem. Tym, co odróżnia ją od SOA jest fakt, że nie spełnia celów i założeń SOA. Podział na serwisy/usługi nie wynika w tym wypadku z potrzeby ani planu ich wielokrotnego użycia w różnych systemach i aplikacjach. Nie ma też na celu zapewnienia ich absolutnej autonomii i niezależnego wdrażania. Podział wynika z pragmatycznych pobudek strategicznych i taktycznych w projektowaniu systemu oraz proaktywnego podejścia do potencjalnych wyzwań jego rozwoju, utrzymania i wsparcia. Zaprojektowana architektura **zapewnia wysoką elastyczność w rozwoju, utrzymaniu i wsparciu** systemu na jego wczesnym etapie życia, jednocześnie mocno **minimalizując różnego rodzaju narzuty pracy, zasobów i wydajności** związanych z SOA i Mikroserwisami. W bieżącym stanie projektu nie ma potrzeby mocno formalnego i sztywnego zarządzania kontraktami między usługami/serwisami, a obecny podział usługi zapewnia szereg zalet. Przy obecnych wymaganiach projektu oraz kompetencjach zespołu, wdrażanie całego systemu jako jednej aplikacji monolitycznej mogłoby być zaś bardziej skomplikowane, a do tego mniej elastyczne.

Bardziej szczegółowe analizy z detalami implementacji, wpisujące się w konkretne rodzaje architektur, np. architektury bezpieczeństwa, wdrażania, aplikacji, przedstawione są w późniejszych sekcjach pracy. Poniżej przedstawiony jest natomiast podział ogólnej architektury na najważniejsze 14 części/komponentów projektu.



Rysunek 15: Diagram UML komponentów ogólnej architektury

1) Aplikacja przeglądarkowa typu SPA (Typescript, Angular, Nginx)
Rolę aplikacji przeglądarkowej, która stanowi dla użytkowników końcowych interfejs graficzny do interakcji z systemem, jest aplikacja typu SPA (Single Page Application) napisana we frameworku Angular w języku Typescript. Aplikacja dostarczana jest użytkownikom odwiedzającym stronę internetową w postaci plików statycznych przez serwer Nginx. Głównym zadaniem aplikacji jest zapewnienie użytkownikom jak najlepszego doświadczenia w interakcji z systemem.

Zależności:

- **Główny serwer aplikacji** — wykorzystanie udostępnianego interfejsu programistycznego REST do obsługi logiki biznesowej związanej z kursami szachowymi
- **Serwis AI** — wykorzystanie udostępnianego interfejsu programistycznego REST do obsługi logiki biznesowej związanej z asystentem AI
- **Serwer autoryzacyjny** — obsługa logowania OAuth2/OICD
- **Serwis do obsługi płatności** — obsługa płatności w aplikacji
- **Serwis do obsługi multimediów** — obsługa pobierania i przesyłania multimediów

2) Główny serwer aplikacji (REST Api, C#, ASP.NET Core)
Rolę głównego serwera aplikacji pełni serwer zaprojektowany w technologii ASP.NET Core w języku C# z interfejsem programistycznym REST. Do jego głównych zadań należy obsługa całej logiki biznesowej aplikacji związanej z obsługą kursów, egzekwowanie kontroli dostępu do zasobów aplikacji w odniesieniu do roli i tożsamości użytkownika (trener/uczeń/niezalogowany użytkownik) oraz komunikacja z bazą danych w celu persystencji danych.

Zależności:

- **Serwis do obsługi płatności** — odbieranie webhooków zawierających statusy dokonanych płatności
- **Baza danych aplikacji** — komunikacja z bazą przez protokół PostgreSQL w celu persystencji danych

3) Serwis AI (REST Api, Python, FastApi)
Rolę serwisu AI pełni serwer zaprojektowany w technologii FastApi w języku Python z interfejsem programistycznym REST opartym na komunikacji HTTP + JSON. Do jego głównych zadań należy obsługa logiki biznesowej związanej z działaniem asystenta AI w aplikacji.

Zależności:

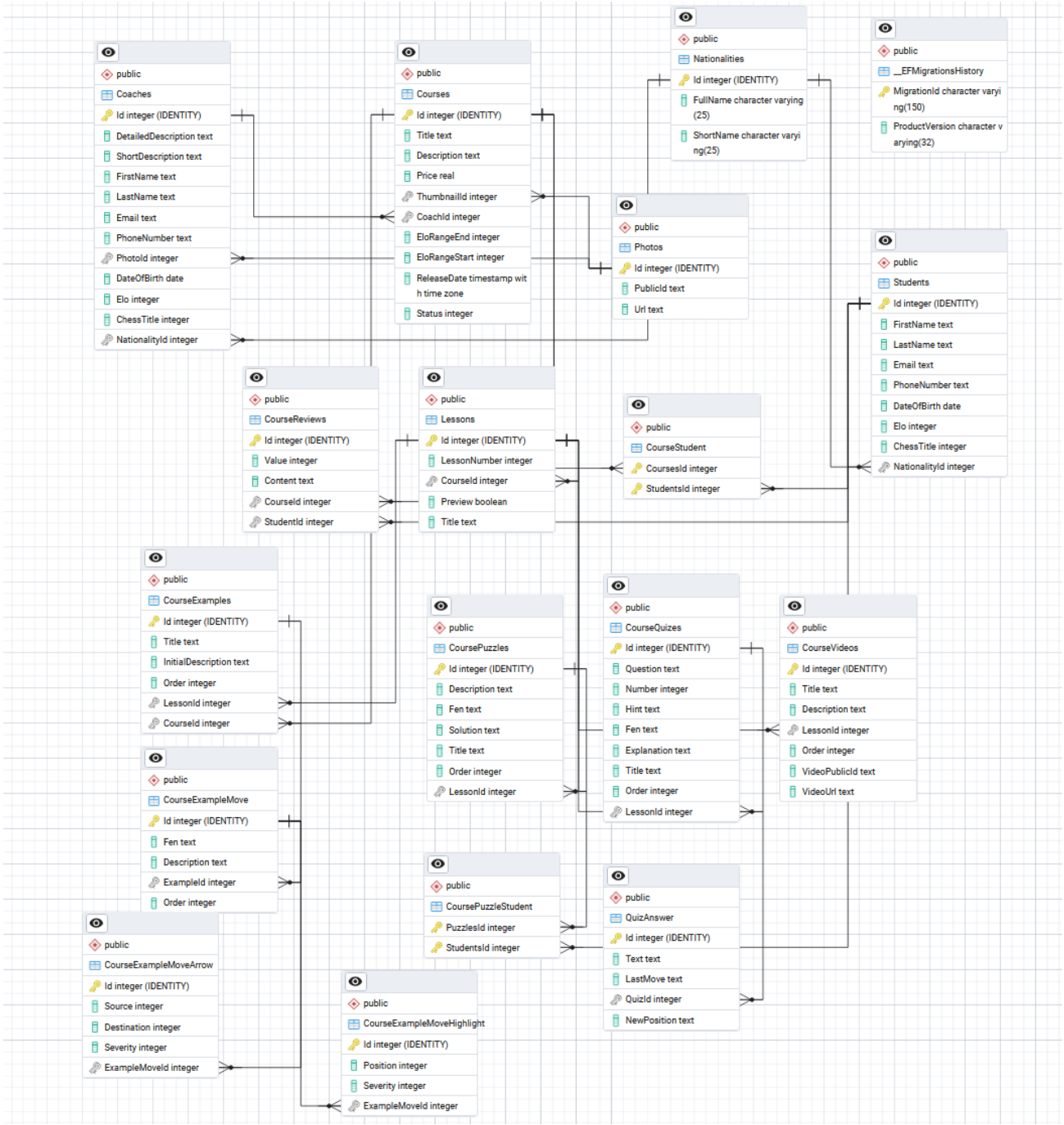
- **Reverse proxy** — obsługa pytań (nie mylić z zapytaniami) użytkowników w postaci żądań HTTP pochodzących z przeglądarek użytkowników końcowych
- **Serwis do obsługi modeli AI** — wykorzystanie REST API serwisu do obsługi pytań użytkowników przez wybrane modele AI

4) Serwer Autoryzacyjny (Keycloak)
Rolę serwera autoryzacyjnego pełni wewnętrznie zarządzany serwer Keycloak. Jego głównym zadaniem jest stanowienie centralnego punktu udostępniającego funkcjonalności do zarządzania tożsamością i dostępami w systemie w zgodzie ze współczesnymi standardami bezpieczeństwa.

Zależności:

- **Baza danych serwera autoryzacyjnego** — komunikacja z bazą przez protokół PostgreSQL w celu persystencji danych

5) Baza danych aplikacji (PostgreSQL)
Rolę bazy danych aplikacji pełni baza PostgreSQL, która zapewnia persystencję danych potrzebnych do obsługi logiki biznesowej aplikacji. Poniżej znajduje się diagram ERD bazy danych wygenerowany przez narzędzie pgAdmin.



Rysunek 16: Diagram ERD bazy danych aplikacji

6) Baza danych serwera autoryzacyjnego (PostgreSQL)

Rolę bazy danych serwera autoryzacyjnego pełni kolejna baza PostgreSQL, która zapewnia persystencję danych potrzebnych do obsługi funkcjonalności dostarczanych przez serwer autoryzacyjny. Znajdują się w niej m.in. dane potrzebne do autoryzacji i uwierzytelniania użytkowników. Architekturą bazy danych w sposób zautomatyzowany zarządza serwer autoryzacyjny.

7) Reverse proxy (Nginx)

Rolę reverse proxy pełni serwer Nginx, który funkcjonuje jako pośrednik w komunikacji z aplikacją, przekazując żądania użytkowników do odpowiednich serwisów warstwy aplikacji wewnątrz prywatnej

sieci systemu. Jest to jedyny wewnętrznie zarządzany komponent systemu, który dostępny jest z poziomu internetu. Jego główną rolą jest pełnienie funkcji centralnego punktu dostępu do systemu dodającego mechanizmy bezpieczeństwa (np. TLS/HTTPS) i optymalizacji wydajności w komunikacji z systemem.

Zależności:

- **Serwer plików statycznych z aplikacją przeglądarkową typu SPA** — pośredniczenie w komunikacji użytkowników z aplikacją SPA poprzez protokół HTTP
- **Główny serwer aplikacji** — pośredniczenie w komunikacji użytkowników z głównym serwerem aplikacji poprzez protokół HTTP
- **Główny serwer aplikacji** — pośredniczenie w wysyłaniu użytkownikom powiadomień za pomocą protokołu SSE (Server Sent Events)
- **Serwis AI** — pośredniczenie w komunikacji użytkowników z serwisem AI poprzez protokół HTTP
- **Serwer autoryzacyjny** — pośredniczenie w komunikacji użytkowników z serwerem autoryzacyjnym poprzez protokół HTTP

8) Serwer automatyzacji (Jenkins)
Rolę wewnętrznie zarządzanego serwera automatyzacji pełni serwer Jenkins wdrożony jako część Operatora Jenkins do Kubernetesa. Jego głównym zadaniem jest automatyzacja procesów budowania, wdrażania oraz wersjonowania artefaktów wdrożeniowych systemu.

Zależności:

- **Repozytoria kodu (Github Repositories)** — cykliczne monitorowanie kodu pod kątem wykrywania zmian i automatycznego budowania i przesyłania do rejestru nowych artefaktów wdrożeniowych, a następnie ich wdrażania
- **Rejestr OCI (Github Package Registry)** — przesyłanie zbudowanych artefaktów wdrożeniowych do przechowywania w rejestrze OCI w postaci Github Package Registry

9) Serwis do obsługi modeli AI (MistralAI)
Rolę serwisu do obsługi modeli AI pełni serwis MistralAI, który udostępnia interfejs REST API do odpytywania różnorodnych modeli AI.

10) Serwis do obsługi płatności (Stripe)
Rolę serwisu do obsługi modeli AI pełni serwis Stripe, który udostępnia funkcjonalności do przeprowadzania płatności w aplikacji webowej.

11) Serwis do obsługi multimediów (Cloudinary)
Rolę serwisu do obsługi modeli AI pełni serwis Cloudinary, który udostępnia infrastrukturę do przechowywania, przesyłania i pobierania multimediów.

12) Zdalne repozytoria kodu (Github Repositories)
Rolę zdalnych repozytoriów kodu pełnią Github Repositories. Oprócz przechowywania kodu zapewniają one szereg funkcjonalności ułatwiających zarządzanie zmianami w kodzie, wzmacniających bezpieczeństwo zarządzania kodem oraz tworzenie reguł, ograniczeń i egzekwowanych konwencji zarządzania kodem.

- **Platforma automatyzacji** — aktualizacja wersji głównej odnogi (main branch) repozytorium głównego serwera aplikacji jest zastrzeżone dla kandydatów (Merge Request), którzy spełnią wymogi walidacji wykonywanej przez testy automatyczne na platformie Github Actions

13) Platforma automatyzacji (Github Actions)
Rolę zewnętrznie zarządzanej platformy automatyzacji pełni Github Actions, której głównym zadaniem jest walidacja kandydatów nowych wersji systemu prezentowanych w tzw. Merge Requestach dzięki testom automatycznym.

14) Rejestr OCI (Github Package Registry)
Rolę rejestru OCI służącego do przechowywania i udostępniania przechowywanych artefaktów wdrożeniowych pełni Github Package Registry.

4.3. Single Page Application (SPA) vs Multi Page Application (MPA)

Do zbudowania aplikacji klienckiej rozważano dwa główne rozwiązania architektoniczne: **Single Page Application (SPA)** oraz **Multi Page Application (MPA)**. MPA jest bardziej tradycyjnym podejściem, w którym nawigacja pomiędzy różnymi widokami na stronie odbywa się poprzez renderowanie nowej strony za każdym razem, kiedy użytkownik chce zmienić widok. W SPA wszystko renderowane jest na tej samej stronie, co eliminuje potrzebę częstego przeładowywania. Sprawia to, że doświadczenie użytkownika jest bardziej płynne i zbliżone do natywnej aplikacji.

Oba podejścia mają swoje zastosowania w zależności od takich czynników jak np. poziom skomplikowania aplikacji czy interaktywność. Dlatego bardzo ważne było przeanalizowanie wymagań przed wyborem odpowiedniego rozwiązania. Poniżej przedstawiono najważniejsze aspekty, które wpłynęły na decyzję:[11]

4.3.1. Szybkość interakcji

W SPA nie trzeba ładować nowej strony z serwera przy każdej zmianie widoku (wystarczy pobrać potrzebne dane), dzięki czemu przejścia pomiędzy widokami są dużo szybsze i łagodniejsze (brak efektu przeładowania strony). W Passed Pawn aplikacji zarówno uczeń, jak i trener mają wiele widoków, pomiędzy którymi muszą często nawigować, dlatego SPA wydaje się naturalnym wyborem.

4.3.2. SEO (Search Engine Optimization)

W tradycyjnym SPA pozycjonowanie strony jest dużym wyzwaniem, ponieważ *crawlery* wykorzystywane przez wyszukiwarki najlepiej radzą sobie, gdy HTML jest od razu wypełniony odpowiednimi danymi. W SPA dane są dostępne dopiero po załadowaniu kodu JavaScript. MPA wypada tu lepiej, ale nowoczesne frameworki SPA oferują dwie techniki, które rozwiązują ten problem:

- SSR (Server Side Rendering): strony renderowane są na serwerze po otrzymaniu żądania
- SSG (Server Side Generation): renderowanie strony na serwerze przy budowaniu aplikacji

[11] Sehban Alam. *Single Page Applications (SPAs) vs. Multi-Page Applications (MPAs): Advantages and Challenges*. `https://medium.com/@sehban.alam/single-page-applications-spas-vs-multi-page-applications-mpas-advantages-and-challenges-df06bee3fed1/`. Accessed: 2025-06-17. 2024.

Niestety te techniki mogą być użyte tylko dla niektórych stron. W opisywanej aplikacji SEO ma znaczenie tylko dla strony głównej (landing-page), ponieważ reszta treści jest dostępna dopiero po zalogowaniu i wykupieniu kursu. Dlatego pozycjonowanie nie jest kluczowe.

4.3.3. Początkowy czas ładowania

Współczynnik rezygnacji i współczynnik konwersji to 2 najważniejsze metryki określające zainteresowanie użytkowników. Współczynnik rezygnacji odnosi się do procenta użytkowników, którzy odwiedzili stronę i opuścili ją bez wykonania żadnych interakcji. Natomiast współczynnik konwersji określa liczbę użytkowników, którzy wykonali określoną akcję (np. rejestrację na platformie). Bez SSR/SSG SPA może mieć dłuższy czas ładowania oraz większe opóźnienie, zanim strona stanie się interaktywna (TTI - Time To Interactive). Zmniejszenie TTI z 5,5s do 3,85s może zmniejszyć współczynnik rezygnacji o 10% oraz zwiększyć współczynnik konwersji o 57%.[12] MPA w tej kategorii wypada lepiej, ale SPA może zminimalizować ten problem przy zastosowaniu SSR/SSG na najważniejszych stronach.

4.3.4. Użycie zasobów klienta

SPA uruchamia znacznie więcej kodu JavaScript w przeglądarce klienta, co może powodować spadek wydajności i w konsekwencji gorsze doświadczenie użytkownika. Jednak opisywana aplikacja nie stosuje skomplikowanych animacji ani intensywnych obliczeń, więc problem ten nie jest bardzo istotny.

4.3.5. Obciążenie serwera

MPA przy każdej zmianie widoku musi pobrać wszystkie dane z serwera, co zwiększa jego obciążenie. Natomiast w SPA niektóre dane mogą być zachowane pomiędzy widokami. Problem ten może stać się istotny przy dużej liczbie użytkowników, ale warto go uwzględnić już na etapie planowania.

4.3.6. Bogate doświadczenie użytkownika

SPA oferuje bogatsze, bardziej natywne doświadczenie użytkownika niż MPA. Łatwiejsza jest implementacja animacji, interaktywnych elementów czy aktualizacji w czasie rzeczywistym. Jest to dla nas bardzo istotne, ponieważ interfejs użytkownika (zarówno ucznia, jak i trenera) opiera się na interaktywnych szachownicach, formularzach i innych dynamicznych elementach.

4.3.7. Efektywny development

Frameworki SPA (np. React, Angular, Vue) oferują wiele narzędzi wspierających efektywny development oraz organizację kodu. Dzięki architekturze komponentowej (CBA — Component-Based Architecture) możliwe jest tworzenie niezależnych, reużywalnych komponentów. Przy stosowaniu dobrych praktyk (np. zasada jednej odpowiedzialności — SRP), unikanie nadmiernych zależności oraz pisanie komponentów wielokrotnego użytku, proces developmentu staje się prostszy i bardziej efektywny.

Biorąc pod uwagę wszystkie wymienione aspekty, zdecydowano się na zastosowanie **Single Page Application**. Wybór ten znacząco ułatwił proces implementacji oraz zapewnił lepsze doświadczenie użytkownika.

[12] Edgemesh. *Why Time to Interactive (TTI) is the Most Consistent and Valuable Metric*. `https://edgemesh.com/blog/time-to-interactive-and-conversion-rate#heading-1`. Data dostępu: 2025-06-17. 2025.

4.4. Wykorzystane protokoły komunikacyjne

4.4.1. SSE

SSE (Server Side Events) jest protokołem opartym na nasłuchiwaniu wiadomości. W przeciwieństwie do podobnych alternatyw takich jak WebSockets bądź MQTT, SSE jest kompletnie jednostronny. Tak jak z nazwy wynika, klient nasłuchuje wiadomości pochodzących od serwera.

W aplikacji Passed Pawn taka komunikacja jednostronna ma jedno ważne zastosowanie. Ponieważ wszystkie płatności są przetwarzane przez zewnętrzny serwis, Stripe, po zakupie kursu jest lekkie opóźnienie, zanim zostanie faktycznie dodany do listy kursów ucznia w bazie danych. Z tego powodu, klient po zakupie kursu zaczyna nasłuchiwać na wiadomości SSE, a serwer wysyła taką wiadomość, kiedy kurs zostaje dodany do listy. Dzięki temu użytkownik jest w stanie zobaczyć powiadomienie.

Zastosowanie te kompletnie nie potrzebuje komunikacji dwukierunkowej, dlatego więc SSE okazało się idealnym protokołem pod takie zastosowanie.

4.4.2. HTTP, HTTP + TLS

HTTP jest zdecydowanie najczęściej używanym protokołem w systemie Passed Pawn. Z wyjątkiem jednego przykładu, gdzie użyty został SSE, cała komunikacja pomiędzy klientem a serwerem opiera się właśnie na protokole HTTP. Głównym powodem wybrania tego protokołu jest przestrzeganie konwencji REST oraz duże wsparcie przez używane przez system frameworki.

Oprócz komunikacji klient-serwer, HTTP jest też ekskluzywnie używany do komunikacji z zewnętrznymi serwisami, takimi jak Stripe czy Cloudinary.

Cała komunikacja, która jest wysyłana z i do serwera aplikacji używa HTTP + TLS (HTTPS), aby zapewnić poufność, integralność, i uwierzytelnienie.

Aplikacja Passed Pawn używa HTTP 2.0, ponieważ testy ujawniły, że wersja 1.1 powoduje niespodziewane zachowania przy użyciu SSE. Dodatkowo wersja 2.0 oferuje inne zalety, takie jak szybsze przetwarzanie zapytań i odpowiedzi.

5. Aplikacja przeglądarkowa typu SPA

Aplikacja kliencka stanowi interfejs dla użytkownika i jest jedynym sposobem na interakcje z jego funkcjonalnościami. Dla platformy edukacyjnej interakcja z użytkownikiem jest kluczowym elementem systemu, ponieważ bez odpowiedniego przedstawienia materiałów nie mają one żadnej wartości. Tworząc aplikację kliencką, należało zadbać, aby doświadczenie było jak najlepsze zarówno dla ucznia, jak i dla osoby tworzącej materiały. Trzeba było również zadbać o odpowiedni dobór technologii i rozwiązań, który umożliwił efektywny rozwój i dobrą skalowalność rozwiązania.

5.1. Porównanie najpopularniejszych frameworków/bibliotek SPA – tabela

Tabela 4: Porównanie frameworków: Angular, React i Vue

Kryterium	**Angular**	**React**	**Vue**
Data wydania	2016 (Angular 2+)	2011	2014
Wsparcie	Google	Meta	Społeczność
Typ	Framework	Biblioteka	Framework
Podejście	Opinionated (silnie narzucone założenia projektowe)	Elastyczny (pozwala samemu decydować nad organizacją projektu i konkretnymi rozwiązaniami)	Bardziej elastyczny niż Angular, ale nie tak jak React
Routing	Wbudowany (Angular Router)	Zewnętrzny (React Router, Next.js)	Zewnętrzny, ale oficjalnie wspierany (Vue Router)
Formularze	Template-driven: podejście deklaratywne, korzystające z dyrektyw Angulara, Reactive Forms: oparte na modelu, korzystające z FormControl i FormGroup	Manulana implementacja przy użyciu useState/useRef, zewnętrzne zależności (Formik, React Hook Form)	Manulana implementacja przy użyciu v-model, zewnętrzne zależności (vee-validate, vuelidate)
Zarządzanie stanem	Brak wbudowanego, można użyć serwisów lub NgRx, NGXS	Brak wbudowanego, można użyć Redux, Zustand	Brak wbudowanego, można użyć Vuex, Pinia
Model komponentów	Klasowe, z szablonem HTML	Obecnie rekomendowane funkcyjne	SFC (Single File Components)
Wsparcie testów	Wbudowane: Karma, Jasmine	Zewnętrzne: Jest, React Testing Library	Zewnętrzne: Vitest, Vue Test Utils
Wstrzykiwanie zależności	Wbudowane DI	Brak wbudowanego DI	Wbudowane DI, ale bardzo ograniczone

Tabela została opracowana na podstawie zewnętrznego źródła w postaci artykułu internetowego.[13] Zostały w niej porównane 3 najczęściej używane frameworki SPA.[14] Przedstawiono podstawowe informacje dotyczące historii i wsparcia, ogólną specyfikę, oraz bardziej szczegółowo aspekty technologiczne. W kolejnych punktach szerzej opisano czynniki decyzyjne dotyczące wyboru konkretnego frameworka.

5.2. Wybory technologiczne związane z interfejsem użytkownika

Przed rozpoczęciem projektu, zespół musi podjąć ważna decyzje nad wyborem technologii, które umożliwią wydajną budowę UI projektu. Jest to równie ważny etap, jak implementacja, ponieważ jej konsekwencje będą się ciągnąć przez resztę rozwoju aplikacji. Po wnikliwej analizie zespół zdecydował się na Angular jako frontend naszego portalu szachowego. Wybrany framework jest dobrze przystosowany do budowy dużych aplikacji. Angular wymusza silną strukturę i architekturę projektu, co w konsekwencji przerzuca się na jasne zasady organizacji kodu. Aplikacja jest podzielona na komponenty, które dzielą kod na mniejsze logiczne całości, co w rezultacie sprawia, że kod jest bardziej przejrzysty i ułatwia nawigację.

5.2.1. Porównanie frameworków pod względem stylowania – tabela

Tabela 5: Porównanie stylowania komponentów: Angular i React

Cecha	**Angular**	**React**
Struktura projektu	Style per component (.component.scss/css)	Dowolna: CSS Modules, inline styles, Tailwind
Scope (zakres stylów)	Domyślna enkapsulacja (scoped styles)	Brak domyślnego scoping — wymaga CSS Modules lub CSS-in-JS
Metody stylowania	• CSS / SCSS • Angular Material • Tailwind (opcjonalnie)	• CSS / SCSS • CSS Modules • Styled-components • Tailwind CSS • Emotion
Integracja z frameworkiem	Angular CLI automatycznie obsługuje style	Pełna dowolność, zależy od configa/bundlera
Biblioteki UI	Angular Material, NG-ZORRO, PrimeNG	Material UI, Ant Design, Chakra UI, Tailwind UI
Stylowanie dynamiczne	Trudniejsze (przez klasy/ngStyle)	Naturalne – można stylować bezpośrednio w JS
Filozofia	Rozdzielenie warstw: logika, HTML, styl	Komponent = logika + JSX + style (podejście holistyczne)

[13] BrowserStack. *Angular vs React vs Vue: Core Differences*. `https://www.browserstack.com/guide/angular-vs-react-vs-vue`. Data dostępu: 2025-06-18. 2025.

[14] Stackoverflow. *Tag Trends*. `https://trends.stackoverflow.co/tags=reactjs%2Cvue.js%2Cangular%2Csvelte%2Cangularjs%2Cvuejs3`. Data dostępu: 2025-06-19. 2025.

Tabela została opracowana na podstawie zewnętrznego źródła w postaci artykułu internetowego.[15]

5.2.2. Podejście do stylowania

Wybór języka stylów jest kluczowy z perspektywy implementacji wyglądu interfejsu użytkownika. Wyróżniamy 3 główne technologie cieszące się popularnością wśród deweloperów: CSS, SCSS i Tailwind. Klasyczny CSS, czyli język arkuszy stylów, który służy do opisu prezentacji elementów w dokumentach HTML, SCSS definiowany jako rozszerzenie języka CSS i Tailwind, nowoczesny framework CSS polegający na stylowaniu za pomocą gotowych klas. W przypadku projektu szachowego definiowanie dużej ilości klas tailwindowych stylów prowadzi do mieszania struktury HTML z wieloma selektorami, co w rezultacie produkuje chaotyczny i nieczytelny kod. Z kolei prymitywny CSS nieczytelnie reprezentuje zagnieżdżeniową strukturę HTML. Najstosowniejszą opcja okazał się SCSS, który pozwala na wspomniane definiowanie stylów z wizualizacją zagnieżdżeń danych klas. Rozszerzenie CSS pozwala na wygodne definiowanie zmiennych, umożliwiając ujednolicenie stylów w całej aplikacji. Stosowanie zmiennych zmniejsza ryzyko błędu zapisania wartości stylów z pamięci (np. koloru, szerokości komponentu) i pozwala na efektywne podmienianie wartości w całej aplikacji. SCCS jest również wyposażony w pętlę for, która automatyzuje pisanie powtarzalnych klas, zapewniając wygodne wystylowanie komponentu szachownicy.

Tabela 6: Porównanie systemów stylowania na frontendzie

Cecha / Technologia	**CSS**	**SCSS (Sass)**	**Tailwind CSS**
Składnia	Standardowa, prosta	Rozszerzenie CSS z dodatkowymi funkcjami	Użycie klas narzędziowych (utility classes) w HTML
Organizacja kodu	Ręczna	Zmienne, zagnieżdżenia, mixiny, funkcje	Wszystko przez klasy utility – zero plików CSS
Abstrakcja / logika	Brak logiki — wszystko pisane ręcznie	Tak – zmienne, warunki, pętle	Brak logiki – wszystko przez gotowe klasy
Łatwość użycia	Prosty dla początkujących	Wymaga nauki rozszerzeń	Wymaga znajomości klas i konfiguracji
Wydajność	Zależna od pisania ręcznego	Zoptymalizowana, ale większy CSS	Bardzo wydajny – używane klasy są treściwe i krótkie
Reużywalność	Trzeba pisać osobno	Wysoka dzięki mixinom i funkcjom	Wysoka przez klasy utility, ale mniej elastyczna
Konfiguracja	Brak	Minimalna (preprocesor)	Wymaga Tailwind config i build toola (PostCSS, Webpack)

[15] Gara Mohamed. *Angular vs. React (Part 1): style abstraction and encapsulation*. `https://medium.com/@gara.mohamed/angular-vs-react-part-1-style-abstraction-and-encapsulation-f55b336e0eaa`. Data dostępu: 2025-06-19. 2018.

Tabela została opracowana na podstawie zewnętrznego źródła w postaci artykułu internetowego.[16]

5.2.3. Sposób enkapsulacji stylów

Angular wspiera enkapsulację stylów – odnosi się to do sposobu, w jaki style CSS są stosowane do komponentów, umożliwia ukrywanie ich przed światem zewnętrznym. Framework oferuje 3 tryby enkapsulacji: domyślną Emulated – style ograniczają się do komponentu, None – brak enkapsulacji, ShadowDom – styl działa w ramach rzeczywistego ShadowDom – technologia webowa, która pozwala tworzyć izolowane drzewa DOM wewnątrz elementów HTML. Zastosowany w naszym projekcie domyślny tryb enkapsulacji – Emulated sprawia, że napisany SCSS jest przejrzysty i łatwiejszy w utrzymaniu. Aplikacja szachowa używa domyślnego trybu enkapsulacji, co zmniejsza ryzyko konfliktów klas CSS i przypadkowego nadpisywania stylów.

5.2.4. PrimeNG jako wybrana biblioteka UI

PrimeNG to biblioteka UI oferująca szeroki wybór gotowych komponentów dla interfejsu użytkownika, dedykowana dla Angulara. Umożliwia wybór motywu pasującego do aplikacji. Znacznie przyśpiesza proces developmentu, co pozwala na skupienie się na kluczowych funkcjonalnościach.[17] Tak jak opisano wcześniej, komponenty, w których należało uzyskać konkretny wygląd, zostały zaimplementowane ręcznie, natomiast nadal zdecydowano się użyć gotowych rozwiązań z biblioteki do stworzenia m.in. powiadomień „toast", okien modalnych, niektórych typów inputów, czy paginacji.

5.3. Cechy Angulara oraz ich wpływ na implementację

Angular to framework do tworzenia aplikacji webowych stworzony na podstawie AngularJS. Jest rozwijany przez Google, a nowe wersje są wypuszczane co około 6 miesięcy. Opisywana aplikacja korzysta z wersji 19, czyli najnowszej w momencie pisania pracy. Angular opiera się na języku TypeScript i umożliwia budowę złożonych aplikacji jednostronicowych (SPA). Jest stosowany do budowania średnich i dużych aplikacji webowych. Oferuje architekturę opartą na komponentach. W tej sekcji opisano wybrane cechy Angulara oraz wbudowane w niego rozwiązania, mające wpływ na implementację aplikacji przeglądarkowej.

5.3.1. Silnie narzucone założenia projektowe

Angular to *opinionated framework*, co w praktyce oznacza, że twórcy z góry narzucają pewne dobre praktyki, podziały, konwencje i rozwiązania architektoniczne. Przy dużych aplikacjach, w których rozwój zaangażowane jest wiele deweloperów, pozwala to uniknąć problemów wynikających z indywidualnych podejść i decyzji pojedynczych osób. Pomaga to również uniknąć niepotrzebnego debatowania nad wyższością danego rozwiązania i skupić się na istotnych aspektach działania aplikacji.

5.3.2. TypeScript

TypeScript to wysokopoziomowy statycznie typowany język programowania wykorzystywany w Angularze. Jest on nadzbiorem JavaScriptu, do którego dodaje wsparcie dla statycznego typowania oraz

[16] Sotiris Kourouklis. *SASS, CSS, or Tailwind: Which One Should You Choose?* `https://dev.to/sotergreco/sass-css-or-tailwind-which-one-should-you-choose-3c0o`. Data dostępu: 2025-06-19. 2025.

[17] PrimeNG. *PrimeNG Documentation*. `https://primeng.org/`. Data dostępu: 2025-06-17. 2025.

niektóre funkcje obiektowe jak dekoratory czy interfejsy. Cały framework Angular został zaprojektowany z myślą o użyciu TypeScriptu. Użycie tego języka w porównaniu do JavaScriptu zwiększa bezpieczeństwo i czytelność kodu, ułatwia rozwój dużych aplikacji oraz zapewnia dobrą integrację z obiektową architekturą Angulara.

5.3.3. System Dependency Injection

Angular oferuje rozbudowany, ale równocześnie prosty w użyciu system wstrzykiwania zależności (Dependency Injection). Użycie tego wzorca oferuje następujące korzyści:

- Luźne powiązania: klasy (w tym przypadku komponenty lub dyrektywy) nie tworzą samodzielnie swoich zależności, co pozwala na łatwą wymianę.
- Ułatwienie testowania: łatwe podmienianie zależności pozwala na proste *mockowanie* na potrzeby testów jednostkowych
- Zwiększona modularność kodu: serwisy mogą być używane w wielu komponentach lub dyrektywach
- Poprawa jakości kodu: łatwiejsza refaktoryzacja i lepsza czytelność

System Dependency Injection w Angularze oferuje rozbudowaną kontrolę, nad tym, gdzie zależność jest dostępna. Osiąga to dzięki kilku poziomom Injectorów (abstrakcja odpowiedzialna za dostarczanie zależności). Wyróżniamy m.in. Element Injector, który działa lokalnie dla komponentu lub dyrektywy i ich potomków, czy Environment Root Injector, który zawiera globalnie dostępne zależności.[18]

W React nie ma wbudowanego systemu DI. Można użyć w tym celu Context API, ale wiąże się to z wieloma ograniczeniami.

5.3.4. Rozbudowane wsparcie dla formularzy

Angular oferuje 2 podejścia do formularzy, z czego każde z nich ma inne zalety. W poniższej tabeli przedstawiono porównanie tych 2 podejść[19]

	Reactive	**Template-driven**
Model formularza	Zdefiniowany w klasie komponentu	Zdefinionowany w szablonie z użyciem dyrektyw
Podejście	Programistyczne	Deklaratywne
Walidacja	Walidacja w klasie poprzez użycie Validators	Walidacja w szablonie przy użyciu dyrektyw
Przypadki użycia	Skomplikowane lub duże formularze z dynamiczną zmianą pól albo rozbudowaną walidacją	Proste statyczne formy z mało rozbudowaną walidacją

W przeciwieństwie do Angulara, który oferuje aż dwa sposoby na tworzenie formularzy, każdy z wbudowaną walidacją, testowaniem oraz monitorowaniem stanu danego pola (dirty, touched, valid etc). React oraz Vue oferują co prawda możliwość implementacji prostych formularzy przy użyciu

[18] Miłosz Rutkowski. *Wszystko co musisz wiedzieć o Dependency Injection w Angularze*. `https://angular.love/pl/wszystko-co-musisz-wiedziec-o-dependency-injection-w-angularze`. Data dostępu: 2025-06-18. 2024.

[19] Angular. *Forms in Angular*. `https://angular.dev/guide/forms`. Data dostępu: 2025-06-18. 2025.

wbudowanych mechanizmów takich jak useState/useRef w React czy v-model w Vue, natomiast stwarza to konieczność korzystania z wielu zewnętrznych paczek, które mogą być trudne do dobrania oraz utrzymywania jako zależności.

5.3.5. Dyrektywy atrybutowe (attribute directives)

Dyrektywy atrybutowe to klasy stworzone przy użyciu dekoratora @Directive. Umożliwiają one działanie bezpośrednio na elementach drzewa DOM. Dzięki temu możemy operować na niższym poziomie abstrakcji, co może być przydatne w wielu scenariuszach, np. kiedy chcemy dynamicznie dodawać i usuwać elementy z drzewa lub zmieniać ich style czy inne właściwości. Istnieją również wbudowane dyrektywy, takie jak `ngClass`, która dynamicznie przypisuje klasy CSS, `ngStyle`, która bezpośrednio ustawia style, czy `ngModel`, która umożliwia dwukierunkowe wiązanie danych. Stosowanie własnych dyrektyw atrybutowych oferuje liczne zalety:

- Rozdzielenie odpowiedzialności (Separation of Concerns): logika odpowiedzialna za bezpośrednie manipulowanie drzewem DOM jest oddzielona od logiki komponentów
- Reużywalność: dyrektywy można bardzo łatwo przypisać do wielu komponentów lub elementów drzewa DOM. Przykładowo, w aplikacji klienckiej PassedPawn utworzono dyrektywy do wyświetlania różnokolorowych strzałek i podświetleń pól na szachownicy. Są one wykorzystywane w kilku szachownicach dostępnych w aplikacji. Wystarczy, że szachownica ma określoną strukturę elementów, aby można było skorzystać z dyrektywy.

```
<div class="chessboard"
    [appChessboardArrows]="arrows"
    [appChessboardHighlights]="highlights"
>
    @for (field of chessboardView; track $index) {
        <div class="chessboard-field">
        ...
        </div>
    }
</div>
```

- Ułatwione testowanie: dzięki oddzieleniu logiki z dyrektywy od komponentu łatwiej jest napisać testy jednostkowe, ponieważ nie ma potrzeby tworzenia całego komponentu.
- Kontrolowany i bezpieczny sposób manipulacji drzewem DOM
- Czysty i bardziej czytelny kod HTML: dyrektywy pozwalają na zwięzły i semantyczny opis zachowania, dzięki czemu kod jest bardziej czytelny i deklaratywny

5.3.6. Rozwinięty i ekspresywny data binding (wiązanie danych)

Data binding to koncept występujący w większości nowoczesnych frameworków webowych. Polega na połączeniu interfejsu użytkownika z danymi z kodu. W przypadku Angulara będzie to szablon i klasa komponentu. W Angularze wyróżniamy cztery rodzaje data binding'u, z czego trzy pierwsze są jednokierunkowe a ostatni dwukierunkowy:[20]

[20] Angular Minds. *Data Binding In Angular: Everything You Need to Know*. `https://www.angularminds.com/blog/data-binding-in-angular`. Data dostępu: 2025-06-18. 2025.

- *Interpolacja* — służy do prostego wyświetlania danych w widoku, przykładowo `<h3> course.title </h3>`
- *Property binding* — Ten sposób data bindingu można użyć w 2 celach. Służy do dynamicznego przypisania wartości do właściwości elementu DOM: może być to np. źródło obrazka `<img src="icons/edit.png" alt="edit" />`. Możemy również użyć go do przekazania wartości do komponentu - w tym celu musimy zadeklarować w klasie komponentu odpowiedni atrybut z dekoratorem @Input, np. `@Input() public rating: number = 0;`. Teraz możemy przekazać wartość do komponentu: `<app-star-rating [rating]="given` Ten sposób data bindingu może zostać również użyty na dyrektywach.
- *Event binding* - służy do przekazywania danych w drugą stronę niż property binding, czyli reagowania na zdarzenia DOM lub zdarzenia emitowane z komponentów. Również możemy użyć go dwojako, na elemencie DOM: `<button (click)="onDelete($event)">` oraz na komponencie: `<app-coach-course-tile (delete)="deleteCourse($event)"/>`. Żeby można było użyć event binding na komponencie, musimy zadeklarować atrybut z dekoratorem `@Output` i przypisać do niego instancję klasy `EventEmitter: @Output() public delete = new EventEmitter<string>();`. Następnie aby emitować zdarzenie wystarczy wywołać metodę `emit: this.delete.emit(this.course.id.toString());`
- *Two-way binding* (wiązanie dwukierunkowe) - ten sposób pozwala połączyć event binding oraz property binding poprzez zastosowanie dwóch rodzajów nawiasów: `<input[(ngModel)]="currentSeverity"/>`. Najczęściej jest wykorzystywany z użyciem `ngModel`, co zapewnia gotową implemetację. Ten mechanizm jest podstawą działania Template-driven Forms.

Składnia data bindingu w Angularze jest bardzo deklaratywna i ułatwia czytanie kodu. Widząc w szablonie odpowiednie nawiasy, od razu wiemy, z jakim wiązaniem mamy do czynienia i w którą stronę są przesyłane dane.

5.3.7. Wsparcie dla programowania reaktywnego - RxJS

RxJS (Reactive Extensions for JavaScript) jest biblioteką, która oferuje wsparcie dla reaktywnego programowania w JavaScript. Najważniejszym elementem RxJS jest `Observable`, który reprezentuje kolekcję przyszłych wartości lub wydarzeń. Oferuje również operatory takie jak `map`, `filter`, `reduce` i inne, które pozwalają na obsługę asynchronicznych wydarzeń w podobny sposób jak klasyczne kolekcje w JavaScript. To wszystko umożliwia alternatywne i nowoczesne podejście do programowania asynchronicznego, zastępujące wbudowany w JavaScript mechanizm `Promise`.[21]

5.3.8. Detekcja zmian

Mechanizm detekcji zmian jest potrzebny, żeby w momencie, w którym w komponencie zmienią się jakieś dane (na przykład zostanie dodany element do tablicy, zmieni się wartość liczbowa itp.), zmiany te zostały odzwierciedlone w widoku komponentu. Angular oferuje zaawansowany system detekcji zmian, który jest skuteczny i wydajny. System ten działa automatycznie, natomiast zrozumienie zasady jego działania może okazać się przydatne, żeby uniknąć błędów i wykorzystać jego pełny potencjał.

Angular przechwytuje różne asynchroniczne wydarzenia w przeglądarce i przy każdym z nich wywołuje mechanizm detekcji zmian. Mogą to być interakcje użytkownika, takie jak kliknięcia czy inputy oraz asynchroniczne operacje, takie jak np. `setTimeout()`. Mechanizm dodawania wywołania detekcji zmian do domyślnych implementacji jest realizowany przez bibliotekę Zone.js. Detekcja zmian może być również wywołana programistycznie na 2 sposoby: sprawdzenie zmian w bieżącym kom-

[21] Angular. *The RxJS library*. `https://v17.angular.io/guide/rx-library`. Data dostępu: 2025-06-17. 2023; RxJS. *RxJS Overview*. `https://rxjs.dev/guide/overview`. Data dostępu: 2025-06-17. 2025.

ponencie wywołując metodę `ChangeDetectorRef.detectChanges()` lub wywołanie całego cyklu detekcji zmian metodą `ApplicationRef.tick()`. Po wywołaniu detekcji zmian `ChangeDetector`, który jest instancjonowany w każdym komponencie przy jego tworzeniu sprawdza wszystkie dane, które zostały wykorzystane w szablonie. Zmiana jest wykrywana, jeżeli aktualna wartość jest inna od poprzedniej.

Istnieje również możliwość zmienienia sposobu działania detekcji zmian w zakresie komponentu. Możemy ustawić strategię detekcji na „OnPush" ustawiając następującą właściwość w dekoratorze `@Component`: `changeDetection: ChangeDetectionStrategy.OnPush`. Spowoduje to, że detekcja będzie automatycznie uruchamiana tylko w 3 przypadkach:

- Kiedy zmieni się referencja danych przekazywanych przez `@Input`
- Kiedy `EventEmitter` emituje wydarzenie
- Jeżeli w szablonie użyjemy *async pipe* i `Observable` wyemituje nową wartość

Mamy także do dyspozycji opcję całkowitego wyłączenia automatycznej detekcji zmian. Taka potrzeba może się pojawić, kiedy będzie to miało duże znaczenie dla wydajności. Wówczas wystarczy wywołać metodę `ChangeDetectorRef.detach()`, co spowoduje, że zmiany będzie można wykrywać tylko programistycznie.

Oferowany przez Angulara mechanizm wykrywania zmian jest wydajny, efektywny i niezawodny. Framework pozwala też na dość rozbudowaną kontrolę nad jego działaniem.[22]

5.3.9. Wbudowany klient HTTP

Komunikacja z aplikacją serwerową jest kluczową funkcją aplikacji klienckiej PassedPawn. Znaczna część tej komunikacji odbywa się przez protokół HTTP, dlatego jego obsługa i funkcjonalności z nią związane są istotną częścią wybieranego frameworka. W Angularze wbudowany jest klient HTTP w postaci singletona `HTTPClient`. Obsługuje on wszystkie dostępne metody HTTP. Aby skorzystać z klienta, wystarczy wywołać odpowiednią metodę, np.:
`return this.httpClient.get<{lessonCount: number}>('/api/Course/...')`.

Klient oferuje wiele przydatnych funkcjonalności, które znacznie ułatwiają pracę z tym protokołem w dużych projektach:

- Dostęp przez Dependency Injection — `HTTPClient` możemy udostępnić jako singleton w całej aplikacji dzięki metodzie `provideHttpClient()`. Dzięki temu możemy z niego korzystać w wielu serwisach lub bezpośrednio w komponentach.
- Integracja z RxJS - Metody klienta zwracają `Observable`. Dzięki temu request wysyłany jest, dopiero kiedy zasubskrybujemy zwracaną wartość (wywołanie metody `subscribe`). Zyskujemy też dostęp do wszystkich omówionych wcześniej przydatnych operatorów z RxJS.
- Automatyczna serializacja/deserializacja JSON — klient HTTP sam automatycznie przekształca nasze obiekty JavaScript na JSON przed wysłaniem requesta oraz JSON otrzymany w odpowiedzi na obiekt JavaScript. Usprawnia to znacznie proces komunikacji z aplikacją serwerową oraz sprawia, że kod jest czytelniejszy.

[22] Angular University. *Angular Change Detection - How Does It Really Work?* `https://blog.angular-university.io/how-does-angular-2-change-detection-really-work/`. Data dostępu: 2025-06-19. 2025.

- Interceptory — oferują bardzo użyteczny mechanizm przechwytywania żądań i odpowiedzi. Interceptor jest wywoływany przed wysłaniem żądania lub dostarczeniem odpowiedzi do adresata. Można w nim np. zmodyfikować żądanie lub odpowiedź, obsłużyć błędy lub dodać nagłówki. W PassedPawn stworzono interceptor, który dodaje do każdego żądania nagłówek z tokenem JWT w celu autoryzacji na serwerze. Istnieje opcja rejestracji wielu interceptorów, tworząc w ten sposób „łańcuch", w którym będą one wykonywane w kolejności ich deklaracji.

Klient HTTP Angulara jest użytecznym, sprawdzonym i kompletnym rozwiązaniem. Inne frameworki jak Vue czy React zmuszają użytkownika do polegania na wbudowanym w JavaScript narzędziu `fetch` lub korzystania z zewnętrznej zależności `axios`. Jednak rozwiązania te są dużo uboższe w funkcjonalności, dlatego deweloperzy muszą je implementować samemu, co zmniejsza efektywność i powoduje, że kod może znacznie różnić się z projektu na projekt.[23]

5.4. Wybory technologiczne związane z kontrolą dostępu

Wszystkie mechanizmy bezpieczeństwa realizowane są na serwerze, natomiast w aplikacji klienckiej potrzebujemy mechanizmu, który integruje się z dostawcą tożsamości (identity provider), oraz zapewnia, że w żądaniach znajdą się odpowiednie tokeny potrzebne do autentykacji. W tej sekcji opisane zostały wybrane rozwiązania oraz istniejące w Angularze mechanizmy kontroli dostępu.

5.4.1. KeycloakJS, Keycloak-Angular

Keycloak JS jest biblioteką JavaScript, która pozwala na zabezpieczenie aplikacji webowych poprzez działanie jako adapter do Keycloak. Używa protokołu OpenID Connect, aby połączyć się z instancją Keycloaka na serwerze.[24] `Keycloak-Angular` to „nakładka" na Keycloak JS, pozwalająca na łatwiejsze korzystanie z biblioteki przez udostępnienie różnych abstrakcji i narzędzi. Pozwala na łatwą konfigurację i stworzenie instancji serwisu `keycloak`. Serwis ten może być następnie używany jako singleton w całej aplikacji, dzięki wsparciu dla Dependency Injection Angulara.[25] wykorzystywany jest m.in. w serwisie `AuthService`, gdzie potrzebne są metody umożliwiające zalogowanie czy wylogowanie użytkownika oraz wyciąganie nazwy użytkownika, czy tokenu. Niezbędny jest również w `guardach`, gdzie sprawdzana jest rola aktualnie zalogowanego użytkownika.

5.4.2. Guardy

Wszystkie endpointy są odpowiednio zabezpieczone na serwerze, lecz nadal z punktu widzenia użytkownika nie możemy pozwolić na sytuację, w której znajdzie się on na stronie przeznaczonej dla innej roli. Tutaj z pomocą przychodzą nam *guardy* - mechanizm oferowany przez Angular Router. Przy implementacji guarda mamy do wyboru 2 opcje — guard funkcyjny lub klasowy. W PassedPawn użyto funkcji ze względu na zwięzłość i prostotę, natomiast oba podejścia są szeroko stosowane. Do zabezpieczenia strony użyto guarda CanActivate (czyli w funkcyjnej wersji CanActivateFn). Sprawdzany jest w niej aktualnie przechowywany token JWT, który pobrany jest z instancji Keycloaka. Z tokena tego możemy wyciągnąć informację o roli danego użytkownika i w konsekwencji zadecydować, czy

[23] Angular. *Understanding communicating with backend services using HTTP*. `https://angular.dev/guide/http`. Data dostępu: 2025-06-19. 2025.

[24] Keycloak JS. *Keycloak JS npm package page*. `https://www.npmjs.com/package/keycloak-js`. Data dostępu: 2025-06-17. 2025.

[25] Mauricio Vigolo. *Keycloak Angular npm package page*. `https://www.npmjs.com/package/keycloak-angular`. Data dostępu: 2025-06-17. 2025.

może mieć on dostęp do danej strony. Jeżeli użytkownik ma dostęp, wystarczy, że zwrócimy wartość `true`, w przeciwnym wypadku zwracamy `false`. Żeby użyć guardów, wystarczy przekazać tablicę do obiektu opisującego daną ścieżkę, np. `canActivate: [loggedInGuard, studentGuard]`.[26]

Angular oferuję szeroką gamę innych guardów:[27]

- CanActivateChild — podobny do CanActivate, ale zabezpiecza też potomne ścieżki
- CanDeactivate — zabezpiecza przed opuszczeniem komponentu. Przydatny na przykład, żeby uchronić użytkownika przed przypadkowym utraceniem niezapisanych danych w formularzu.
- Resolve — używany, żeby zacząć pobieranie danych jeszcze przed rozpoczęciem nawigacji
- CanLoad — uniemożliwia samo załadowanie kodu modułu, do którego użytkownik nie ma uprawnień
- CanMatch — bardziej nowoczesna alternatywa dla CanLoad, bardziej elastyczny i oferujący większą kontrolę

Guardy są niezwykle pomocnym mechanizmem oferowanym przez Angular Router, ponieważ pozwalają na kontrolę dostępu, optymalizację i polepszenie doświadczenia użytkownika w modularny, prosty i deklaratywny sposób.

5.5. Struktura implementacji

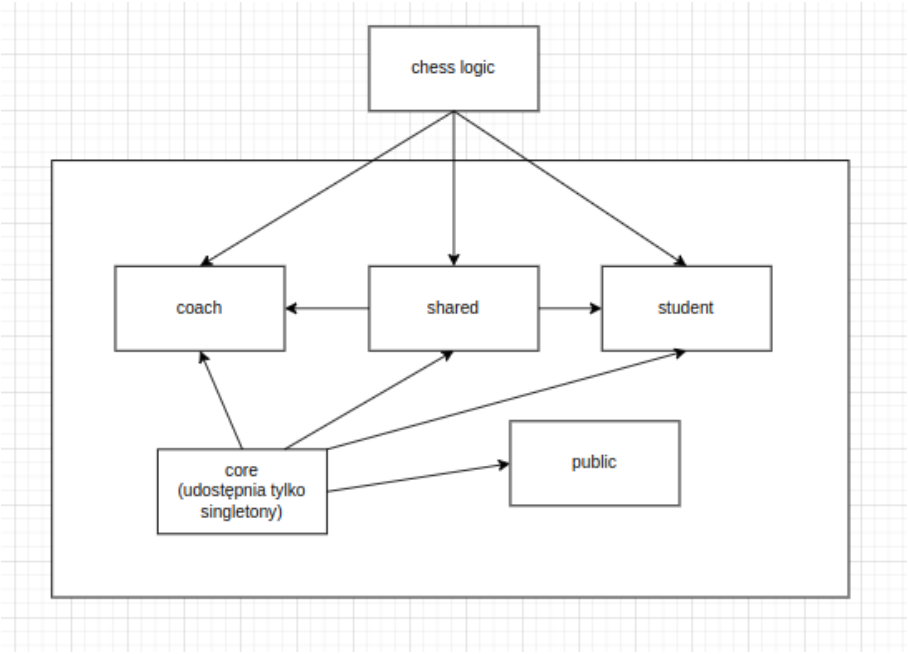
Implementacja aplikacji klienckiej podzielona została na 5 modułów. Wydzielono 2 najważniejsze niezależne od siebie moduły: *student* oraz *coach*. Oba z nich zawierają własne komponenty, modele, resolvery, serwisy etc. W obu jest też zdefiniowany własny routing. Dzięki temu można było zastosować *lazy loading* oraz zabezpieczyć całe moduły w jednym miejscu.

```
    {
        path: 'student',
        canActivate: [loggedInGuard, studentGuard],
        loadChildren: () =>
          import('./modules/student/student.routes').then((mod) => mod.
              STUDENT_ROUTES)
      },
      {
        path: 'coach',
        canActivate: [loggedInGuard, coachGuard],
        loadChildren: () =>
          import('./modules/coach/coach.routes').then((mod) => mod.
              COACH_ROUTES)
      },
```

Podział zasadniczej części kodu na te 2 moduły zdecydowanie ułatwił rozwój w momencie, w którym pojawiło się wiele elementów, które pełniły podobną funkcję, ale dla różnych typów użytkowników. Przykładowo zarówno coach, jak i student potrzebują widoku kursu, ale widoki te są zupełnie inne. Przyjęcie konwencji nazewniczej, w której nazwa każdego komponentu jest poprzedzona nazwą modułu (np. `StudentCourseComponent`), znacznie usprawniło nawigację i pozwoliło uniknąć pomyłek.

[26] Angular. *CanActivateFn Documentation*. `https://angular.dev/api/router/CanActivateFn`. Data dostępu: 2025-06-19. 2025.

[27] Raghuvardhan Karanam. *Route Guards in Angular*. `https://raghuvardhankaranam.medium.com/route-guards-in-angular-c2c01fe6167b`. Data dostępu: 2025-06-19. 2023.



Rysunek 17: Schemat modułów aplikacji klienckiej

Poza „głównymi” modułami wydzielono jeszcze 3 następujące moduły:

- *shared* — w tym module znajdziemy cały kod, który jest używany zarówno przez moduł *coach*, jak i *student*. Ważne jest, aby mieć pewność, że zdefiniowana tu logika będzie zawsze taka sama niezależnie od typu użytkownika. Znajdują się w nim m.in. dyrektywy umożliwiające wyświetlanie strzałek i podświetleń szachownicy, komponenty i serwisy związane z chat botem czy inne pomniejsze funkcjonalności takie jak wyświetlanie poziomu trudności kursu.
- *core* — ten moduł jest odpowiedzialny za kod, który nie jest współdzielony przez moduły *coach* i *student*, ale jest integralną częścią całej implementacji. Znajdują się w nim *gueardy, interceptory* oraz serwisy, które zawsze muszą być *singletonami* (w przypadku omawianej aplikacji jest to `authService`).
- *public* — tutaj umieszczono cały kod związany z niezalogowanym użytkownikiem, czyli komponenty *landing page'a* oraz strony *not found*, która wyświetlana jest, kiedy użytkownik wprowadzi nieprawidłowy adres url.

Poza głównym kodem źródłowym aplikacji podzielonym na moduły wydzielono jeszcze jedną część, niezależną od Angulara, napisaną w czystym TypeScripcie. Jest to logika szachowa, definiująca wszystkie potrzebne zasady gry w szachy takie jak dozwolone ruchy, logika szachowania, matowania, determinowania wyniku rozgrywki etc. Wydzielenie tej części poza kod reszty wprowadza jasny podział odpowiedzialności i ułatwia nawigację w projekcie.

Uproszczoną strukturę modułów i relacje między nimi przedstawiono na schemacie. Strzałka oznacza zależność danego modułu od innego — moduł, na który wskazuje, jest zależny od modułu, z którego wychodzi.

5.6. Logika szachowa

Logika implementująca zasady gry w szachy oraz wyświetlanie szachownicy została wydzielona od reszty aplikacji, aby nie wprowadzać niepotrzebnych powiązań i trzymać się zasady separacji odpo-

wiedzialności. Kod jest oparty na dostępnym publicznie poradniku na YouTube wraz z dołączonym repozytorium na GitHub.[28] [29] Oryginalny kod został szeroko zmieniony i przerobiony na potrzeby opisywanej platformy. Zostały dodane nowe funkcjonalności i mechanizmy, a istniejące struktury danych i klasy wraz z ich metodami i atrybutami również zostały również poddane modyfikacjom. Podjęte zostały próby kontaktu z autorem kodu, ale niestety skończyły się niepowodzeniem. Wskazane w opisie filmu na YouTube kanały komunikacji z autorem kodu są nieaktywne lub nie istnieją. W repozytorium GitHub znajduje się zapytanie o sposób licencjonowania kodu, natomiast na czas pisania pracy i projektowania aplikacji również ono pozostaje niezaadresowane.[30] W związku z brakiem jakiejkolwiek możliwości kontaktu z autorem wykorzystanego kodu, brakiem informacji o sposobie licencjonowania kodu, oraz ze względu na fakt, że kod został udostępniony w ramach poradnika, zostało założone, że jest on przeznaczony do ogólnego i nieogarniczonego użytku. W związku z tym, w formie zmodyfikowowanej i rozszerzonej, został on wykorzystany, by spełniać cel nie gry w szachy, lecz wsparcia obsługi elementów szachowych w kursach. Takie rozwiązanie daje pełną kontrolę rozszerzania, modyfikacji i implementacji kodu wykorzystywanego do spełniania wymagań funkcjonalnych platformy. Stanowi to wielki atut w porównaniu z użyciem zewnętrznych bibliotek zewnętrznymi, których nie znaleziono w formie, która mogłaby zaspokoić wszystkie zapotrzebowania funkcjonalne platformy, a które wiązałyby się ze znacznie trudniejszym rozszerzaniem i modyfikowaniem. Wprowadzałyby również niepotrzebne, niewykorzystywane funkcjonalności, które wpływałyby na gorszą wydajność aplikacji. Zastosowane rozwiązanie wprowadza więc więcej wstępnej złożoności i nakładu pracy, ale umożliwia lepszą optymalizację pod kątem wydajności oraz większą skalowalność, potencjał do rozszerzania i ogólną kontrolę nad rozwiązaniem.

[28] Roberts Tech. *Code a Chess Game with Stockfish API – JavaScript Tutorial*. `https://youtu.be/fJIsqZmQVZQ?si=6JttY01vB8CtVCzJ`. Data dostępu: 2025-06-30. 2024

[29] Roberts Tech. *Code a Chess Game with Stockfish API – JavaScript Tutorial*. `https://github.com/awsomeCStutorials/chess-game`. Data dostępu: 2025-06-20. 2024

[30] davehorner. *license?* `https://github.com/awsomeCStutorials/chess-game/issues/4`. Data dostępu: 2025-06-20. 2025.

6. Główna aplikacja serwerowa

6.1. Cel i rola aplikacji serwerowej

Aplikacja serwerowa jest jednym z ważniejszych elementów architektury całego systemu, działającym głównie jako pośrednik pomiędzy frontendem a bazą danych. Jej głównym celem jest realizacja logiki biznesowej, obsługa żądań przychodzących z aplikacji klienckiej oraz zarządzanie danymi przechowywanymi w systemie. Aplikacja zawsze pilnuje, żeby operacja była wykonana tylko i wyłącznie przez autoryzowanego użytkownika.

6.2. ASP.NET Core — Powód wybrania technologii

ASP.NET Core jest częścią platformy .NET zarządzanej przez firmę Microsoft i skupia się na tworzeniu aplikacji webowych. Framework ten jest wieloplatformowy oraz posiada otwarte źródło.

Technologia te została głównie wybrana z powodu doświadczenia wielu członków zespołu, ale dużym czynnikiem był też bogaty ekosystem, co ostatecznie okazało się dużą zaletą nad podobnymi alternatywnymi technologiami, takimi jak Spring Boot.

Zespół jest zadowolony z tej decyzji, gdyż aplikacja była budowana szybko oraz jest łatwo zarządzana, w dużej mierze dzięki licznym zainstalowanym paczkom z Nuget (menadżer paczek w .NET).

6.3. Architektura aplikacji

6.3.1. Wzorce projektowe

Aplikacja używa wiele wzorców projektowych, aby zarządzanie było jak najprostsze

- Dependency Injection — Aplikacja składa się z wielu serwisów oraz repozytoriów, i prawie wszystkie są dodawane do kontenera DI. Umożliwia to automatyczne wstrzykiwanie tych obiektów do klas, które je potrzebują. Automatyczna natura ułatwia zarządzanie kodem, a sam mechanizm dependency injection ułatwia testowanie, dzięki użyciu mocków.
- MVC (Model–View–Controller) — Jest to jeden z częściej używanych wzorców projektowych w aplikacjach ASP.NET Core. Model w kontekście API reprezentuje dane i logikę biznesową, controller obsługuje żądania HTTP, a view dane w formacie JSON, które są zwracane do użytkownika.
- DTO (Data Transfer Object) - to wzorzec projektowy służący do przenoszenia danych między warstwami aplikacji, np. między kontrolerem a warstwą logiki biznesowej. Dane zwracane do użytkownika również są w formacie DTO, ponieważ pozwala to kontrolować, jakie informacje są udostępniane na zewnątrz.

6.3.2. N-tier architecture

Architektura N-tier to podejście do projektowania aplikacji, które polega na logicznym podziale systemu na oddzielne części o konkretnych odpowiedzialnościach, takich jak obsługa żądań, logika biznesowa czy dostęp do danych.

W Passed Pawn architektura to została użyta, ze względu na wielowarstwowe podejście. Ułatwia to zarządzanie kodem, szczególnie przy pracy w zespole.

6.4. Warstwy aplikacji

6.4.1. API

Warstwa odpowiedzialna za przyjmowanie żądań HTTP i zwracanie odpowiedzi. Zawiera kontrolery (zgodnie ze wzorcem MVC), przeróżne middleware, oraz plik wejściowy, który dodaje serwisy do kontenera DI, konfiguruje autoryzacje i inne middleware, oraz uruchamia aplikacje.

6.4.2. Business Logic

Wiele kontrolerów nie wykonuje wielu operacji, więc logika nie jest nigdzie przenoszona. Jednak dla bardziej skomplikowanych metod, logika jest umieszczana w serwisach wewnątrz tego projektu. Gwarantuje to, że kontrolery nie są bardzo przeciążone, co poprawia ich czytelność.

6.4.3. Data Access

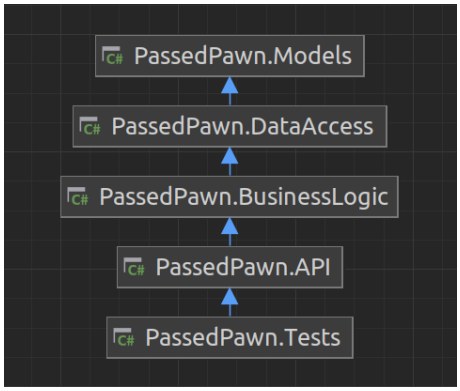
Wewnątrz tego projektu umieszczane są tak zwane entities, czyli modele ze wzorca MVC. Oprócz tego projekt zawiera też ApplicationDbContext, czyli klasę, która reprezentuje dane i encje wewnątrz bazy danych, oraz repozytoria, czyli serwisy wyłącznie odpowiedzialne za modyfikacje i pobieranie danych z bazy.

6.4.4. Models

Projekt zawiera przeróżne klasy, które przechowują dane. Większość modeli wykorzystywanych w aplikacji to DTO. Projekt zawiera również walidatory, które gwarantują, że klient nie wysyła danych, które API nie jest w stanie przetworzyć.

6.4.5. Wykres warstw

Schemat przedstawia warstwową strukturę aplikacji. Strzałki ilustrują kierunek dostępu — która warstwa może korzystać z której. Przykładowo, warstwa BusinessLogic nie ma dostępu do klas znajdujących się w warstwie API, natomiast API może korzystać z funkcjonalności warstwy BusinessLogic. Taki układ zachowuje porządek i czytelność aplikacji.



6.5. Interfejs API

6.5.1. REST

API zostało zaprojektowane w oparciu o styl architektoniczny REST (ang. Representational State Transfer). REST definiuje zbiór zasad, które zapewniają spójność i przewidywalność komunikacji między klientem a serwerem. Dzięki temu odpowiedzi są łatwe do interpretacji przez klienta, a cała struktura API staje się bardziej intuicyjna.

REST opiera się na konwencji kodów odpowiedzi HTTP:

2xx (np. **200 OK**, **201 Created**) — operacja wykonana pomyślnie,

4xx (np. **400 Bad Request**, **404 Not Found**, **403 Forbidden**) — błąd po stronie klienta, na przykład przesłane dane, których serwer nie jest w stanie przetworzyć,

5xx (np. **500 Internal Server Error**) — błąd po stronie serwera, np. wyjątek w aplikacji serwerowej albo tymczasowa awaria zewnętrznego serwisu.

Dzięki jednoznacznemu stosowaniu tych kodów klient może z łatwością określić, czy żądanie zakończyło się sukcesem, czy też napotkało problem, oraz jakiego rodzaju był to problem. Dodatkowo REST przewiduje intuicyjną strukturę adresów URL. Na przykład:

GET	*/api/course*	*// Pobierz wszystkie kursy*
GET	*/api/pcourse/1*	*// Pobierz kurs o ID = 1*
POST	*/api/course*	*// Dodaj nowy kurs*
PUT	*/api/course/1*	*// Zaktualizuj kurs o ID = 1*
DELETE	*/api/course/1*	*// Usuń kurs o ID = 1*

Tabela 7: Przykładowe żądania REST

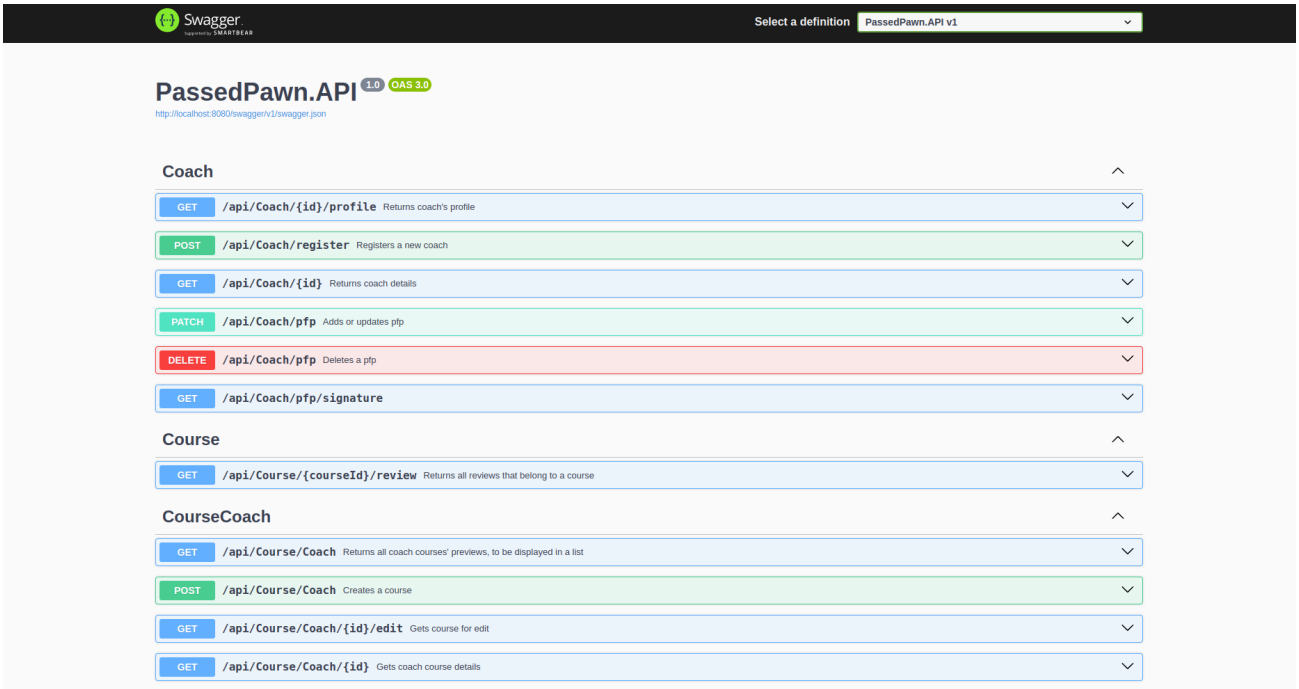
Taka struktura jest intuicyjna i ułatwia korzystanie z API przez klientów.

REST zakłada również, że API powinno być bezstanowe (stateless), co oznacza, że każde żądanie musi zawierać wszystkie informacje potrzebne do jego przetworzenia. Innymi słowami, serwer nie powinien przechowywać żadnego kontekstu między kolejnymi żądaniami, co ułatwia skalowanie i upraszcza architekturę.

Choć REST nie narzuca konkretnego formatu danych, w praktyce, jak i w Passed Pawn najczęściej wykorzystywanym jest JSON (JavaScript Object Notation), ze względu na jego czytelność i lekkość. Ponieważ wszystkie dane wysyłane z i do serwera są w formacie JSON, komunikacja jest zawsze intuicyjna.

6.5.2. Swagger

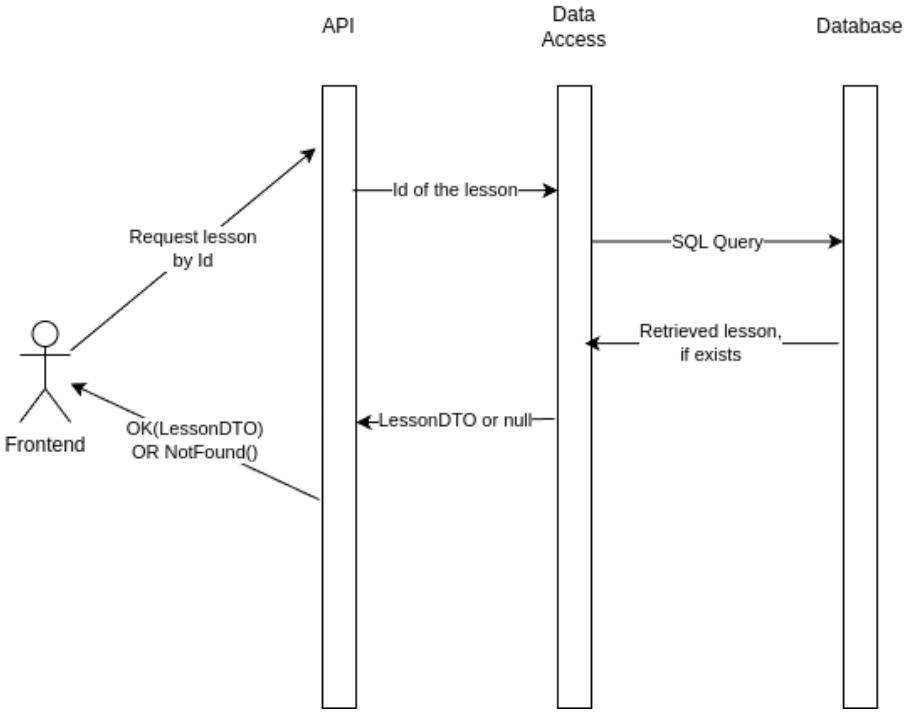
Dokumentacja aplikacji webowej zwłaszcza REST API powinna w przejrzysty sposób przedstawiać dostępne zasoby i ich funkcjonalność. Pozwala to kontrolować rozwój projektu i ułatwia komunikację w zespole, która jest kluczowa do wydajnego tworzenia oprogramowania. Z pomocą przychodzi Swagger — zestaw narzędzi i specyfikacja służąca do projektowania, opisywania, dokumentowania i testowania REST API. Obecnie jest częścią ekosystemu OpenAPI. Swagger to popularne narzędzie, które umożliwia wygenerowanie zwięzłej i spójnej dokumentacji (JSON). Technologia posiada również opcje wizualizacji dokumentacji poprzez wygenerowanie interfejsu UI (strony internetowej) dla zasobów API. Rozwiązanie OpenAPI umożliwia współpracę na poziomie technicznym zespołów deweloperskim frontendu i backendu, które muszą zintegrować budowane oprogramowanie w działającą całość. Technologia Swagger pozwoliła na zdefiniowanie specyfikacji zasobów API (endpointów i modeli danych), która była głównym punktem odniesienia dla zespołów odpowiedzialnych za warstwę klienta i serwera, co zniwelowało rozbieżności integrowanych projektów. OpenApi rozwiązało problem braku spójności między warstwami aplikacji szachowej. Dokumentacja API portalu szachowego obejmuje opisy poszczególnych ścieżek (endpointów) wraz ze zdefiniowaną listą zwracanych kodów odpowiedzi HTTP. Swagger uwzględnia typy zwracanych obiektów (schemas), używanych przez serwer.



6.6. Przykłady operacji

6.6.1. Pobieranie lekcji z użyciem ID

Diagram UML przedstawia operacje wykonywaną, kiedy frontend wyśle zapytanie o kurs o konkretnym ID. Jest to jedna z trywialniejszych operacji, na diagramie widać, że tylko dwie warstwy aplikacji są wykorzystywane.



6.7. Obsługa błędów i walidacja

6.7.1. Global exception handler

Niespodziewane wyjątki nie powinny szkodzić całej aplikacji. Z tego powodu wykorzystany jest middleware o nazwie GlobalExceptionHandler. Przechwytuje wyjątki i zwraca informacje użytkownikowi, że aplikacja jest tymczasowo niedostępna. Równocześnie, szczegóły wyjątku są zapisywane, co umożliwia programistom analizę i naprawę błędu.

6.7.2. Walidacja

Wszystkie dane przysłane przez klienta są walidowane, aby zapewnić, że wszystkie dane są przetwarzalne przez aplikację. ASP.NET Core zapewnia wiele gotowych walidatorów, jednak napisane zostały kilka walidatorów specyficznych dla aplikacji Passed Pawn.

6.8. Komunikacja z bazą danych — EntityFramework Core

Entity Framework Core to nowoczesny, wydajny ORM (Object-Relational Mapper) dla platformy .NET. Umożliwia on mapowanie obiektów klas C# na relacyjne tabele w bazie danych, co znacznie upraszcza dostęp do danych i ich modyfikację, gdyż użycie tego narzędzia kompletnie eliminuje potrzebę ręcznego pisania zapytań SQL.

W projekcie EF Core służył jako warstwa dostępu do bazy danych. Dzięki niemu możliwe było:

- Automatyczne, wymagające minimalnej konfiguracji tworzenie struktury bazy danych na podstawie klas modelowych (Code First),
- Obsługa relacji między encjami (np. jeden-do-wielu, wiele-do-wielu),
- Wykonywanie zapytań LINQ do bazy danych,
- Migracje umożliwiające ewolucję struktury bazy bez utraty danych,
- Asynchroniczne, nieblokujące zapytania do bazy danych.

Zaletą EF Core jest również wsparcie dla wielu dostawców baz danych — w tym przypadku użyto PostgreSQL, ale można łatwo przejść np. na SQL Server, SQLite czy MySQL, z minimalnymi zmianami w kodzie.

7. Moduł AI jako narzędzie doskonalenia UX

7.1. Cel wprowadzenia modułu

Przy implementacji aplikacji klienckiej zespół przyłożył dużą uwagę do zaprojektowania prostego i intuicyjnego interfejsu użytkownika. Jednak pomimo tego, że z perspektywy designera interfejs nie jest skomplikowany, niektórzy użytkownicy mogą mieć problem ze znalezieniem danej funkcji, lub mogą nie zdawać sobie sprawy, jakie funkcjonalności oferuje system. W celu minimalizacji tego problemu zdecydowano się na wprowadzenie asystenta AI w formie `chatbota`. Jest on dostępny z poziomu każdego widoku w rogu okna. Użytkownik w każdej chwili może wysłać do niego pytanie, a ten, przybierając rolę przyjacielskiego słonia, odpowie według własnej wiedzy o aplikacji. Dzięki temu rozwiązaniu użytkownik zawsze będzie miał szybki dostęp do informacji o interfejsie w zasięgu ręki. Chatbot otwiera też nowe możliwości dla niezarejestrowanych użytkowników, niezaznajomionych z celem i funkcjonalnościami platformy. Jeżeli informacje dostępne na landing page'u okażą się dla nich niewystarczające, mogą zadać asystentowi dowolne pytanie z tego zakresu.

7.2. System RAG - rozwiązanie pozwalające na wykorzystanie LLM

LLM (Large Language Model) to zaawansowany model sztucznej inteligencji wytrenowany na ogromnych zbiorach danych. Ma na celu rozumienie i generowanie tekstu w sposób jak najbardziej zbliżony do języka naturalnego.

Dużą wadą LLM jest to, że modele te uczą się na wielkich ilościach danych, a po wyszkoleniu nie jest łatwo rozszerzać ich wiedzy. Choć technicznie jest to możliwe dzięki procesowi zwanemu fine-tuningiem, to jednak ten proces jest kosztowny pod względem zasobów obliczeniowych i czasowych — w wielu przypadkach równie drogi, jak samo trenowanie modelu od podstaw co jest nierealistyczne dla większości osób bądź biznesów. Dlatego więc, osoby korzystające z LLM-ów muszą często dodawać kontekst w zapytaniach, co jest żmudne i często niewystarczające. Problemy te rozwiązuje mechanizm RAG (Retrieval-Augmented Generation).

RAG opiera się na pomyśle, że model LLM ma dostęp do swojej prywatnej bazy danych. Baza danych musi być wektorowa, ponieważ umożliwia to na bardzo szybkie wyszukiwanie powiązanych tekstów, głównie na podstawie semantyki zdania.

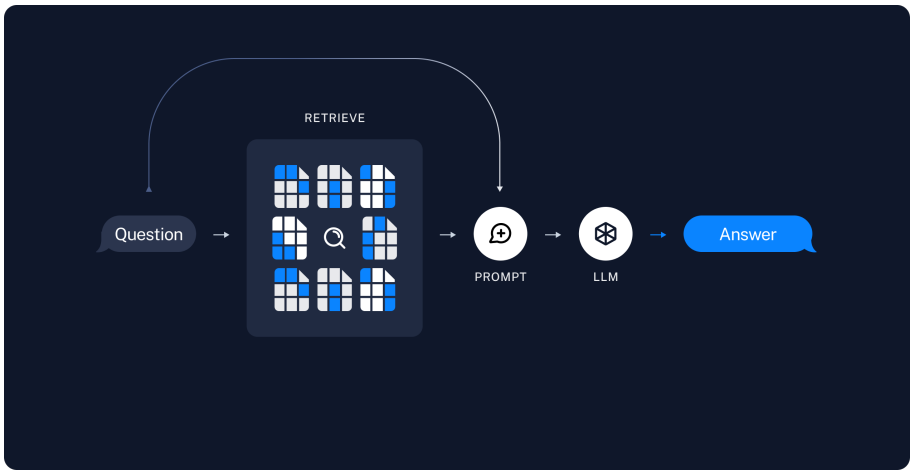
Przy każdym zapytaniu do systemu RAG, pytanie jest najpierw przekierowywane do wektorowej bazy danych. Wektory (czyli semantyka zdań) zapytania i zapisanych danych są porównywane, i najbliższe dane są pobierane z bazy danych. Następnie, zapytanie oraz wyciągnięte dane są równocześnie przekazywane do LLM, które, patrząc na dane, odpowiada na pytanie użytkownika.

Proces jest intuicyjny, wydajny oraz „niewidoczny" dla osoby pytającej, gdyż z perspektywy użytkownika model jedynie odpowiada na pytanie. Dużą zaletą jest też fakt, że nie trzeba modelu za każdym razem dotrenowywać, aby dodać nowe informacje, jedynie wystarczy zapisać odpowiednie dane w bazie danych.

Kolejną zaletą jest prywatność danych. Baza danych może postawiona lokalnie i nie jest nigdzie udostępniana.

Istotne jest też znacząco zmniejszone ryzyko halucynacji. RAG umożliwia na zweryfikowanie każdej odpowiedzi. LLM może być nauczone, że jeżeli odpowiedź na pytanie nie jest zawarta w dokumentach

z bazy, to po prostu odpowie, że nie. Daje to znacznie większą pewność, że użytkownik nie zostanie błędnie poinformowany.



Rysunek 18: Diagram działania RAG, udostępniony przez LangChain

Diagram pochodzi z dokumentacji python langchain[31].

Baza danych używana przez system RAG aplikacji PassedPawn zawiera wiedzę na temat strony, głównie przypadki użycia, ale również wygląd interfejsu użytkownika. Użytkownik zadający pytanie, na przykład „Jak mogę się zarejestrować jako uczeń", otrzymuje odpowiedź od LLM, która zawiera wszystkie instrukcje, jak użytkownik może się zarejestrować. Dlatego ważne jest, aby baza danych zawierała jak najwięcej informacji i aby zawsze była aktualna.

7.3. Implementacja serwerowej aplikacji AI

7.3.1. Python

Wysokopoziomowy język programowania ogólnego przeznaczenia. Jego składnia charakteryzuje się prostotą i klarownością. W PassedPawn został użyty przy implementacji chatbota, ze względu na szeroką gamę dostępnych bibliotek i narzędzi związanych ze sztuczną inteligencją.

7.3.2. Wykorzystanie LangChain do utworzenia RAG Chain

LangChain jest frameworkiem do budowania aplikacji opierających się na LLM-ach. Umożliwia integrację LLM-ów z innymi technologiami, takimi jak wektorowe bazy danych czy modele embeddingowe. Udostępnia również możliwość korzystania z API wielu LLM-ów jak np. GPT-4o, Gemini czy wykorzystany w PassedPawn MistralAI. To pozwala na implementację systemu RAG w formie RAG Chain, czyli połączonego systemu kroków. Aby zilustrować kroki, które wykonywane są w ramach „łańcucha" najlepiej posłużyć się przykładem kodu, w którym definiowany jest RAG Chain w przedstawionej implementacji:

[31] LangChain. *Build a Retrieval Augmented Generation (RAG) App: Part 1*. `https://python.langchain.com/docs/tutorials/rag/`. Data dostępu: 2025-06-24. 2024.

```
rag_chain = (
    {
        "context": retriever | format_docs,
        "question": RunnablePassthrough()
    }
    | RunnableLambda(build_prompt)
    | llm
    | StrOutputParser()
)
```

Warto zauważyć, że zdefiniowany RAG Chain składa się z 4 etapów, oddzielonych przez znaki „|". Dane płyną po kolei przez każdy etap, zgodnie z kolejnością definicji. Na początku do „łańcucha" wchodzi pytanie od użytkownika aplikacji klienckiej w formie łańcucha znaków. Na wyjściu otrzymujemy zaś odpowiedź na pytanie, również w formie string, które prześlemy z powrotem do klienta. Oto jak po kolei przetwarzane są dane:

- Etap 1: Główną rolą tego etapu jest znalezienie odpowiednich dokumentów w naszej wektorowej bazie danych za pomocą *retrievera*. Znajduje on dokumenty najlepiej semantycznie dopasowane do pytania. Następnie funkcja `format_docs` łączy je w jeden ciąg znaków, tworząc w ten sposób *context*. *Question* to po prostu pytanie, które dostaliśmy na wejściu, przekazane dalej bez zmian przez *RunnablePassthrough*.
- Etap 2: Ten etap jest odpowiedzialny za przekształcenie obiektu, otrzymanego z poprzedniego etapu w gotowy prompt dla modelu. Funkcja `build_prompt` łączy specjalne instrukcje, dotyczące „osobowości słonia", w którego ma wcielić się nasz model z kontekstem i pytaniem utworzonym w poprzednim etapie. Tak przygotowany prompt jest gotowy, aby przekazać go do następnego etapu.
- Etap 3: Tutaj mamy już do czynienia z klasycznym zapytaniem do LLM i wygenerowaniem odpowiedzi. Na podstawie promptu generuje on radę czy też wskazówkę dla użytkownika.
- Etap 4: Ten etap jest już tylko zabezpieczeniem, dzięki któremu jesteśmy pewni, że jako dane wyjściowe dostaniemy string. W większości przypadków model zwróci odpowiedź już w dobrym formacie, natomiast nie zawsze i nie we wszystkich modelach tak się dzieje, więc dobrą praktyką jest odpowiednie przetworzenie odpowiedzi na końcu łańcucha.

RAG Chain użyty w omawianej implementacji jest dość prosty, natomiast modularna budowa sprawia, że można go bardzo łatwo i inkrementacyjnie rozbudowywać o kolejne etapy. LangChain jest potężnym frameworkiem oferującym bardzo wiele funkcjonalności wykraczających poza zakres omawianego problemu, ale mimo tego budowa niewielkiego RAG Chaina jest relatywnie prosta i nieskomplikowana.

7.3.3. Wektorowa baza danych

ChromaDB to wektorowa baza danych stworzona specjalnie na potrzeby aplikacji działających z LLM-ami. Jest przystosowana do przechowywania *embeddingów*, czyli wektorowych reprezentacji zdań z naturalnego języka. Dzięki temu *retriever* po zwektoryzowaniu zapytania użytkownika (np. przy tworzeniu chatbota) może w bardzo efektywny sposób porównać zapytanie do dokumentów w bazie i zwrócić najbardziej podobne dokumenty. Znalezione dokumenty mogą być potem wykorzystane jako dodatkowy kontekst przy generowaniu odpowiedzi przez LLM.

7.3.4. API

FastAPI to nowoczesny i wydajny framework do tworzenia API w Pythonie. Jest łatwy do opanowania i pozwala na szybki development. Został użyty w module AI, aby wystawić API do chatbota. Do tej funkcjonalności nie potrzebujemy wielu endpointów ani skomplikowanej logiki, dlatego ważne było, aby rozwiązanie było proste i minimalistyczne, ale zarazem skuteczne.

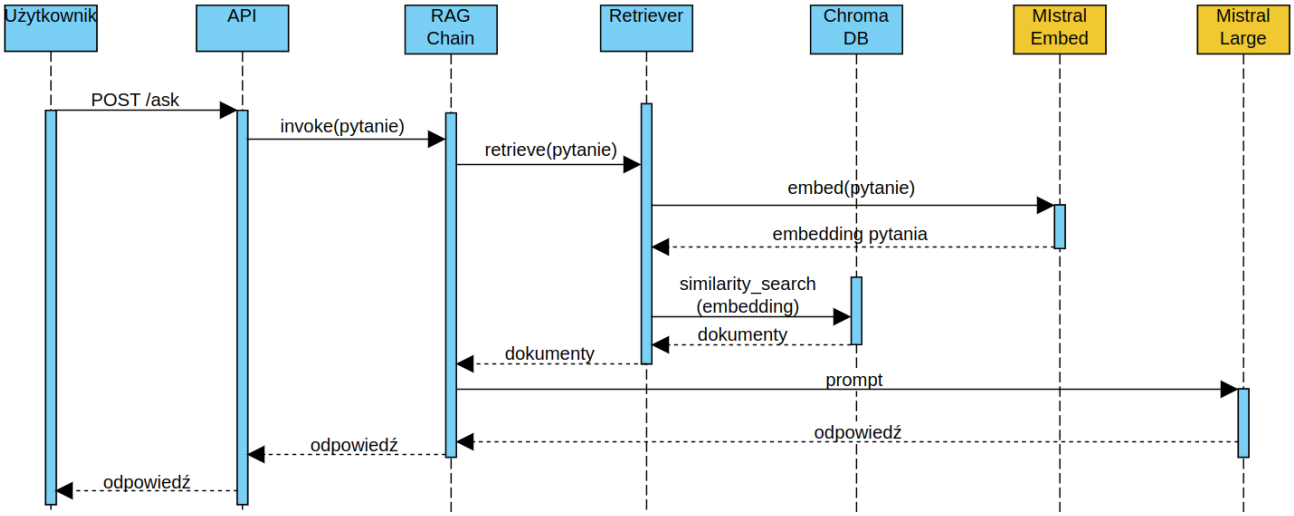
7.4. Analiza dostępnych modeli AI

W bibliotece LangChain mamy do wyboru integrację z wieloma modelami LLM, jednak zdecydowano się na wybór modeli oferowanych przez francuską firmę MistralAI. Omawiany system korzysta z 2 modeli:

- Mistral Large: jest to najnowszy i najbardziej zaawansowany model oferowany obecnie przez MistralAI. Według benchmarków jest drugim najlepszym modelem dostępnym obecnie przez API, po modelu OpenAIMistralAI. *Mistral Large, our new flagship model*. `https://mistral.ai/news/mistral-large`. Data dostępu: 2025-06-17. 2025
- Mistral Embed: model embeddingowy, czyli model wyspecjalizowany w zamienianiu zdań naturalnego języka w *embeddingi*. Wykorzystywany jest do embeddowania zarówno danych o aplikacji wprowadzonych do bazy danych, jak i pytań użytkownika, które porównywane są z danymi w bazie. MistralAI. *Models Overview*. `https://docs.mistral.ai/getting-started/models/models_overview/`. Data dostępu: 2025-06-17. 2025

Głównym czynnikiem decyzyjnym przy wyborze tych modeli była korzystna oferta w darmowym pakiecie. W miarę rozwoju aplikacji modele można łatwo podmienić na płatne alternatywy.

7.5. Diagram sekwencji modułu AI



Rysunek 19: Diagram sekwencji modułu AI. Kolor żółty oznacza serwisy spoza systemu.

8. Infrastruktura systemu

Infrastruktura systemu informatycznego to zestaw komponentów niezbędnych do działania i zarządzania usługami IT oraz środowiskami przedsiębiorstwa[32]. Składa się z elementów sprzętowych, takich jak serwery, urządzenia sieciowe i centra danych, oraz oprogramowania, w tym systemów operacyjnych i aplikacji, które współdziałają, aby zapewnić efektywną komunikację, wydajność i bezpieczeństwo systemu.[33]

Odpowiednie zaplanowanie i zaprojektowanie infrastruktury w zgodzie z potrzebami produktu jest ważne, ponieważ rzutuje na stabilność, skalowalność i bezpieczeństwo systemu, a tym samym na jego sukces biznesowy i konkurencyjność.

8.1. Podział infrastruktury na tradycyjną/lokalną i chmurową

Współcześnie infrastrukturę IT dzieli się na dwie podstawowe kategorie ze względu na poziom kontroli oraz bezpośredniość dostępu do zasobów: tradycyjną (lokalną) oraz chmurową[34]. Rozwiązania chmurowe zdobywają coraz większą popularność i często stanowią preferowany wybór ze względu na elastyczność i skalowalność.

Infrastruktura tradycyjna

Infrastruktura tradycyjna opiera się na fizycznych zasobach IT, które są zainstalowane i zarządzane lokalnie w siedzibie firmy lub w jej wyznaczonych obiektach. Tego typu infrastruktura obejmuje np. serwery, urządzenia sieciowe, komputery stacjonarne albo chociażby mikrokomputery takie jak Raspberry Pi, które pełnią funkcje obliczeniowe lub sterujące.

Zalety infrastruktury tradycyjnej

Zaletą infrastruktury tradycyjnej jest pełna i bezpośrednia kontrola nad sprzętem i danymi oraz możliwość dostosowania systemu do wymagań konkretnej organizacji/projektu. Ponadto, lokalne rozwiązania mogą oferować niższe opóźnienia i dają większą niezależność od zewnętrznych dostawców usług.

Wady infrastruktury tradycyjnej

Do wad infrastruktury tradycyjnej należą potencjalnie wysokie koszty utrzymania, w tym wydatki na sprzęt, energię i personel techniczny, oraz ograniczona skalowalność w porównaniu do wielu rozwiązań chmurowych. Oprócz tego, aktualizacje i rozbudowa infrastruktury mogą być czasochłonne i skomplikowane.

Infrastruktura chmurowa

Infrastruktura chmurowa to model dostarczania zasobów IT przez internet, gdzie użytkownicy korzystają z usług i zasobów bez konieczności posiadania fizycznego sprzętu na miejscu. Charakteryzuje się szerokim spektrum usług o różnym poziomie kontroli nad infrastrukturą — od serwerów dedykowanych i VPS, przez usługi przechowywania danych takie jak Amazon S3, Azure Blob Storage, Google Cloud Storage, aż po platformy multimedialne jak Cloudinary.

[32] IBM. *What Is It Infrastructure?* `https://www.ibm.com/think/topics/infrastructure`. Data dostępu: 2025-06-23. 2021.

[33] [Tamże].

[34] [Tamże].

Zalety Infrastruktury chmurowej

Zalety infrastruktury chmurowej to przede wszystkim elastyczność i łatwość skalowania zasobów w zależności od potrzeb, co pozwala na optymalizację kosztów i szybkie dostosowanie do zmieniających się obciążeń systemu lub wymagań biznesowych. Ponadto, rozwiązania chmurowe często oferują zaawansowane mechanizmy bezpieczeństwa i wysoką dostępność.

Wady infrastruktury chmurowej

Wadami infrastruktury chmurowej są potencjalne obawy dotyczące bezpieczeństwa i prywatności danych, a także uzależnienie od zewnętrznego dostawcy usług. Może to ograniczać kontrolę nad systemem i dostępność usług w przypadku awarii lub problemów z łącznością. Dodatkowo koszty długoterminowego użytkowania chmury mogą być trudne do przewidzenia i kontrolowania.

8.2. Wybór serwera dedykowanego OVH oraz domeny internetowej

W przypadku systemu Passed Pawn optymalną infrastrukturą ze względu na niską dojrzałość projektu jest taka, która pozwala jak najbardziej zminimalizować początkowy wysiłek jej obsługi i rozwoju, a przy tym pozostaje elastyczna w modyfikacji i narzuca jak najmniej konkretnych rozwiązań. W fazie budowania projektu od zera, wymagania funkcjonalne i niefunkcjonalne oraz założenia dotyczące architektury i wyglądu projektu mogą się łatwo i szybko zmieniać i w związku z tym elastyczność oraz jak największa kontrola posiadana nad systemem działa na jego korzyść. Jednocześnie, maksymalne ograniczenie pracy nad infrastrukturą pozwala skupić się na logice biznesowej systemu i dzięki temu w szybszym tempie weryfikować założenia dotyczące wymagań i architektury systemu oraz samego potencjału biznesowego projektu.

Za dobry środek między zapewnieniem aspektu maksymalnej kontroli a zminimalizowaniem ilości pracy nad infrastrukturą, można uznać w tym przypadku infrastrukturę w postaci serwera dedykowanego. **Serwer dedykowany** to fizyczny komputer w konkretnym centrum danych, którego zasoby sprzętowe (procesor, pamięć podręczna, pamięć masowa) są przeznaczone wyłącznie do użytku jednego klienta, zapewniając mu pełną kontrolę, wysoką wydajność, niezawodność i prywatność w użytkowaniu sprzętu, bez współdzielenia go z innymi użytkownikami. W tym samym czasie jest to rozwiązanie, które abstrahuje fizyczne zarządzanie sprzętem od użytkownika, co minimalizuje wstępną pracę nad przygotowaniem infrastruktury. Jest to więc rozwiązanie spełniające przestawione dla systemu Passed Pawn wymagania.

8.2.1. Oferta i specyfikacja serwera dedykowanego

Na potrzeby projektu wykorzystany jest serwer dedykowany KS-4 z oferty budżetowych/ekonomicznych serwerów dedykowanych od OVH.

Specyfikacja

- **Procesor** — Intel Xeon-E3 1230v6 - 4c/8t - 3.5 GHz/3.9 GHz
- **Pamięć podręczna** — 32 GB ECC 2133 MHz
- **Pamięć masowa** — 2×450 GB SSD NVMe, Soft RAID

W skład oferty wchodzi umowa o gwarantowanym poziomie usług (ang. Service Level Agreement - SLA), która zapewnia dostępność serwera na poziomie 99,9% czasu w skali miesiąca[35]. Koszt

[35]OVH. *Dedicated Server Service Level Agreement*. `https://us.ovhcloud.com/legal/sla/dedicated-`

wynajmu serwera wynosi 118,08 zł miesięcznie + dodatkowe, jednorazowe 94,71 zł za przygotowanie serwera. Są to ceny z wliczonym podatkiem VAT w wysokości 23%.

8.2.2. Oferta i specyfikacja domeny internetowej

W celu udostępnienia dostępu do aplikacji bez potrzeby wpisywania adresu IP serwera zakupiona jest domena internetowa z oferty OVH. W cenie 54,72 zł rocznie (wliczając 23% podatku VAT) zarejestrowana jest domena passed-pawn.com, a w skład oferty wchodzi również usługa DNSSEC zabezpieczająca domenę przed podatnością na cache poisoning. Oprócz tego, w skład oferty wchodzi opcjonalna obsługa domeny przez serwery DNS od OVH.

8.3. Oprogramowanie serwera dedykowanego

W tym miejscu znajduje się specyfikacja i opis najistotniejszych elementów oprogramowania serwera z poziomu systemu operacyjnego, które zostały zainstalowane i skonfigurowane, aby zapewnić bezpieczeństwo, stabilność oraz funkcjonalność środowiska serwerowego. Poniżej znajduje się konkretny wykaz najważniejszych technologii, narzędzi i konfiguracji wykorzystywanych do operowania na serwerze.

- **Debian 12 (Bookworm) z najnowszymi aktualizacjami** — stabilna i nowoczesna dystrybucja Linuxa, oferująca szerokie wsparcie architektur i bogaty zestaw pakietów
- **Zdalny dostęp SSH** — zabezpieczony poprzez dopuszczenie wyłącznie uwierzytelniania kluczem SSH dodanym na serwerze — zwiększa bezpieczeństwo poprzez eliminację logowania hasłem
- **Wyłączone logowania SSH na konto root** — dodatkowa warstwa zabezpieczeń, wymuszająca logowanie się na konto użytkownika z uprawnieniami sudo
- **Fail2ban** — narzędzie do ochrony przed atakami brute force, które monitoruje logi i blokuje podejrzane adresy IP
- **UFW (Uncomplicated Firewall)** — zapora sieciowa skonfigurowana do blokowania całego ruchu przychodzącego z wyjątkiem portów SSH, HTTP, HTTPS oraz portu Kubernetes, przy jednoczesnym zezwoleniu na cały ruch wychodzący
- **K3s** — lekka dystrybucja Kubernetes, umożliwiająca zarządzanie kontenerami w środowisku produkcyjnym
- **Helm** — menedżer pakietów dla Kubernetes, ułatwiający instalację i zarządzanie aplikacjami kontenerowymi
- **Docker** — platforma do tworzenia, dystrybucji i uruchamiania kontenerów, zapewniająca izolację i łatwość wdrażania aplikacji
- **OpenTofu** — narzędzie pozwalające na zarządzanie konfiguracją infrastruktury i zasobów w sposób deklaratywny

Tak skonfigurowane środowisko serwerowe zapewnia stabilność, bezpieczeństwo oraz elastyczność w zarządzaniu aplikacjami i usługami na serwerze dedykowanym.

8.4. Kubernetes - wybrane narzędzie orkiestracji systemu

Kubernetes (K8s) to zaprojektowana przez Google i certyfikowana przez CNCF (Cloud Native Computing Foundation) platforma o otwartym kodzie źródłowym służąca do orkiestracji zasobów systemów

`servers/`. Data dostępu: 2025-06-23. 2025.

zazwyczaj w formach kontenerów, ale również wirtualnych maszyn, która automatyzuje ich wdrażanie, skalowanie i zarządzanie[36]. Trudno wyszczególnić pojedynczą, kluczową/główną funkcję Kubernetesa, co wynika z jego wszechstronnego potencjału zastosowań i szerokiego spektrum dostarczanych wartości.

Zalety Kubernetes[37] [38]

- **Automatyzacja procesów:** automatyzuje wdrażanie, skalowanie i zarządzanie aplikacjami skonteneryzowanymi
- **Skalowalność:** umożliwia automatyczne, zarówno poziome, jak i pionowe, skalowanie aplikacji mogące dostosowywać się do zmieniającego się obciążenia
- **Wysoka dostępność:** oferuje mechanizmy zwiększające odporność na awarie poprzez mechanizmy automatycznego przełączania awaryjnego i równoważenia obciążenia
- **Samonaprawianie:** pozwala automatycznie restartować kontenery, które uległy awarii, zastępować je i wstrzymywać ruch do nich, dopóki nie będą gotowe go przyjąć
- **Przenaszalność:** umożliwia wdrażanie aplikacji w różnych środowiskach: on-premise, w chmurze publicznej i w środowiskach hybrydowych
- **Zarządzanie konfiguracją i sekretami:** pozwala na bezpieczne przechowywanie i zarządzanie wrażliwymi danymi oraz konfiguracją
- **Równoważenie obciążenia i odkrywanie usług:** może wystawiać kontenery za pomocą nazw DNS lub adresów IP i równoważyć ruch sieciowy
- **Automatyczne wdrożenia i wycofywania:** umożliwia mocno kontrolowane wdrażanie nowych wersji aplikacji i łatwe wycofywanie zmian
- **Rozszerzalność:** pozwala na dodawanie nowych funkcjonalności bez modyfikowania kodu źródłowego, dzięki Custom Resources i Operatorom

Wady Kubernetes

- **Stroma krzywa nauki:** do wykorzystania w skomplikowanych, złożonych lub wielkich skalą przypadkach biznesowych może wymagać znacznej wiedzy i doświadczenia, co wyklucza wykorzystanie przez osoby początkujące
- **Złożoność:** Kubernetes jest złożonym systemem, co może prowadzić do skomplikowanego zarządzania i rozwiązywania problemów w przypadku nieprzemyślanego/słabo kontrolowanego rozwoju klastrów
- **Potencjalnie nieefektywny dla małych projektów:** minimalny poziom wiedzy umożliwiający operowanie na Kubernetesie może być zbyt wysoki w porównaniu do oferowanych zalet dla małych start-upów, małych projektów lub prototypów

8.4.1. Elastyczności Kubernetes

Jedną z kluczowych cech Kubernetesa jest na pewno elastyczność. Kubernetes umożliwia dynamiczne zarządzanie zasobami i automatyczne skalowanie aplikacji kontenerowych, a nawet wirtualnych maszyn w różnorodnych środowiskach: On-premise, na wirtualnych serwerach prywatnych (VPS), na serwerach dedykowanych oraz w ramach dedykowanych usług u największych dostawców chmurowych, takich jak AWS (Elastic Kubernetes Service — EKS), Azure (Azure Kubernetes Service —

[36]Dokumentacja techniczna Kubernetes. *Kubernetes*. `https://kubernetes.io/`. Data dostępu: 2025-06-23. 2025.

[37]Dokumentacja techniczna Kubernetes. *Concepts/Overview*. `https://kubernetes.io/docs/concepts/overview/`. Data dostępu: 2025-06-23. 2024

[38]Dhaval Gajjar. *Why And Where To Use Kubernetes*. `https://www.opensourceforu.com/2024/02/why-and-where-to-use-kubernetes/`. Data dostępu: 2025-06-23. 2024

AKS) i Google Cloud (Google Kubernetes Engine — GKE). Jest w stanie również obsługiwać środowiska mieszane (hybrydowe), gdzie część aplikacji działa lokalnie, a część w chmurze. Kubernetes oferuje także znaczący potencjał rozszerzalności udostępniając ustandaryzowany sposób na rozszerzanie swoich podstawowych funkcjonalności przy pomocy Custom Resources (CR) i Custom Resource Definitions (CRD) oraz z wykorzystaniem udokumentowanego wzorca Operator. Otwiera tym samym szeroką gamę możliwości i ułatwia wprowadzania dobrych praktyk zarządzania infrastrukturą takich jak GitOps i Infrastructure as Code. Przykład wykorzystania części tego potencjału stanowi CloudNativePG, który jako Operator dla PostgreSQL na Kubernetes automatyzuje w deklaratywny i audytowalny sposób zarządzanie klastrami baz danych, w tym replikację, kopie zapasowe i wysoką dostępność, wszystko dzięki wykorzystaniu CR i CRD i wzorca Operator.

8.4.2. Niezawodności i skalowalność Kubernetes

Kubernetes wzmacnia niezawodność systemu dzięki mechanizmom automatycznego przywracania usług, równoważenia obciążenia oraz zarządzania replikacją, co minimalizuje ryzyko przestojów i awarii. Umożliwia skalowanie aplikacji zarówno w poziomie (poprzez dodawanie replik i węzłów), jak i w pionie (poprzez zwiększanie zasobów istniejących instancji), co pozwala efektywnie dostosować się do zmiennego zapotrzebowania. Ponadto, dzięki rozszerzeniom takim jak Operatory, Kubernetes wspiera również wysoką dostępność dla komponentów zewnętrznych, np. baz danych, zapewniając solidną i odporną infrastrukturę. W efekcie platforma ta jest kluczowym elementem budowy niezawodnych i skalowalnych środowisk produkcyjnych.

8.4.3. Uzasadnienie wykorzystania Kubernetes

Wybór Kubernetesa w przypadku logicznego zarządzania zasobami Passed Pawn uzasadniony jest jego elastycznością, łatwą rozszerzalnością, deklaratywnością, audytowalnością, bezpłatnym użytkowaniem oraz szerokim wsparciem ze strony środowiska open source.

Jako całość, Kubernetes stanowi idealny ekosystem, umożliwiający iteracyjne ulepszanie infrastruktury systemu niemal pod każdym względem — skalowalności, niezawodności, wysokiej dostępności, wydajności, przenaszalności, bezpieczeństwa oraz automatyzacji procesów. Kluczową zaletą jest przy tym możliwość pracy nad infrastrukturą małymi krokami, bez konieczności posiadania od razu szerokiej wiedzy ani podejmowania wszystkich decyzji na wczesnym etapie projektowania systemu.

Dzięki temu nawet przy ograniczonej znajomości samego narzędzia i jego rozszerzeń lub nie do końca sprecyzowanych wymaganiach dotyczących systemu i jego architektury można je stopniowo wdrażać, stosując proaktywne podejście do wyzwań, które mogą pojawić się w przyszłości. Kubernetes i jego rozszerzenia oferują bowiem gotowe, szeroko wspierane, sprawdzone i ustandaryzowane rozwiązania dla wielu potencjalnych problemów i potrzeb.

8.5. K3s jako wybrana dystrybucja K8s

Dystrybucje Kubernetes to gotowe do użycia pakiety, które ułatwiają instalację i zarządzanie klastrem Kubernetes, często oferując już skonfigurowane komponenty i różnorodne optymalizacje do konkrentych zastosowań. Różnią się między sobą zestawem wbudowanych funkcji, rozmiarem i wymaganiami zasobowymi.

K3s to lekka dystrybucja Kubernetes certyfikowana przez CNCF, zaprojektowana z myślą o małym

rozmiarze, małym zużyciu zasobów i prostym użytkowaniu[39]. Jest dostarczana jako pojedynczy plik binarny o rozmiarze mniejszym niż 100 MB, co osiąga nie zawierając potencjalnie niepotrzebnych modułów z oryginalnego Kubernetesa. Minimalne wymagania do uruchomienia klastra K3s to 2 GB pamięci RAM i 2 rdzenie CPU.[40]

Zalety:

- **Lekkość:** bardzo mały rozmiar i niskie zużycie zasobów, idealny do środowisk z ograniczonymi zasobami
- **Łatwość instalacji:** prosta instalacja i szybkie uruchamianie
- **Pełna zgodność z Kubernetes:** pełna kompatybilność z API Kubernetes, co pozwala na używanie standardowych narzędzi i manifestów

Wady:

- **Mniej funkcji niż pełen Kubernetes:** celowe usunięcie niektórych funkcji, które mogą być potrzebne w bardziej złożonych środowiskach produkcyjnych
- **Mniej opcji konfiguracji:** mniejsza elastyczność w konfiguracji niektórych komponentów w porównaniu do pełnej dystrybucji

Wybór K3s dla Passed Pawn jest uzasadniony jego lekkością, prostotą instalacji i zarządzania. K3s pozwala na szybkie wdrożenie i zarządzanie klastrem bez nadmiernego obciążania infrastruktury, jednocześnie zachowując pełną zgodność z API Kubernetes, co umożliwia wykorzystanie wszystkich standardowych narzędzi, wzorców i rozszerzeń w dalszym udoskonalaniu infrastruktury.

8.6. Najważniejsze komponenty niestandardowe klastra

8.6.1. Longhorn

Lognhorn w dużym skrócie jest narzędziem służącym do dynamicznej aprowizacji pamięci masowej dla Kubernetesa. Potrzeba i problematyka aprowizacji pamięci masowej dla Kubernetesa wynika z faktu, że kontenery są z natury efemeryczne i bez dedykowanej pamięci masowej ich dane znikają po zakończeniu działania, natomiast Kubernetes nie posiada wbudowanego mechanizmu dynamicznego zapewniania przestrzeni dyskowej chmurowej lub lokalnej. Kubernetes jest za to zgodny ze standarded CSI (ang. Container Storage Interface), czyli interfejsem określającym sposób integracji systemów zarządzania pamięcią masową z Kubernetesem. Lognhorn ma swój wbudowany sterownik CSI i dzięki niemu może dynamicznie zarządzać lokalną pamięcią dyskowa i chmurową dla Kubernetesa. W środowiskach kontenerowych zapewnienie trwałych woluminów (Persistent Volumes — PV), które mogą być dynamicznie tworzone i zarządzane, jest koniecznie aby aplikacje stanowe mogły przechowywać swoje dane niezależnie od cyklu życia Podów.

Longhorn jest preferowanym wyborem dla zasobów aplikacji, zwłaszcza tych stanowych, ponieważ w przeciwieństwie do wbudowanego w K3s narzędzia aprowizacji pamięci masowej (Local Path Provisioner), oferuje on funkcjonalności dbające o niezawodność i bezpieczeństwo danych, a także daje potencjał do implementacji mechanizmów zapewniania wysokiej dostępności do danych.

[39] Dokumentacja techniczna K3s. *K3s - Lighweight Kubernetes*. `https://docs.k3s.io/`. Data dostępu: 2025-06-23. 2025.

[40] Dokumentacja techniczna K3s. *Requirements*. `https://docs.k3s.io/installation/requirements`. Data dostępu: 2025-06-23. 2025.

Najważniejsze funkcjonalności Longhorn[41]

- Replikacja woluminów
- Migawki woluminów
- Backupy do zewnętrznych systemów (S3, NFS)
- Odzyskiwanie danych po awarii
- Klonowanie woluminów
- Szyfrowanie
- Intuicyjny dashboard zarządzania

8.6.2. Nginx Ingress Controller

Przed wyjaśnieniem czym jest Nginx Ingress Controller, należy pochylić się nad pojęciem Ingress w kontekście Kubernetesa. Ingress to wbudowany w Kubernetes typ zasobu, który zarządza zewnętrznym dostępem do usług w klastrze, zazwyczaj przez HTTP i HTTPS. Ingress działa jako router, kierując ruch do odpowiednich usług na podstawie reguł hosta i ścieżki[42].

Nginx Ingress Controller to kontroler Ingress wykorzystujący serwer Nginx jako reverse proxy i load balancer, automatycznie konfigurujący serwer Nginx zgodnie z deklaracjami zawartymi w zasobach typu Ingress[43].

Zalety:

- **Zaawansowany routing:** rozbudowane zdolności routingu (host, ścieżka, nagłówki)
- **Przydatne funkcjonalności:** wsparcie TLS i łatwe zarządzanie certyfikatami
- **Wysoka wydajność i niezawodność:** komponenty cert-manager zużywają stosunkowo niedużo zasobów, oferując przy tym mechanizmy wysokiej dostępności zwiększającej niezawodność cert-managera

Wady:

- **Złożoność:** większa złożoność konfiguracji w porównaniu do samodzielnie przygotowanego proxy nginx
- **Większe zużycie zasobów:** wyższe bazowe zużycie zasobów od lżejszych rozwiązań

Nginx Ingress Controller jest niezbędny do zarządzania złożonym ruchem przychodzącym do systemu i zapewnia elastyczność i skalowalność.

8.6.3. Cert-manager

Cert-manager to kontroler Kubernetes, wpisujący się we wzorzec Operator, automatyzujący zarządzanie certyfikatami SSL/TLS, ułatwiający ich uzyskiwanie, odnawianie i integrację z zasobami typu Ingress[44].

Zalety:

[41] Dokumentacja techniczna Longhorn. *Features*. `https://longhorn.io/docs/1.9.0/`. Data dostępu: 2025-06-23. 2025.

[42] Dokumentacja techniczna Kubernetes. *Ingress*. `https://kubernetes.io/docs/concepts/services-networking/ingress/`. Data dostępu: 2025-06-23. 2025.

[43] Repozytorium ingress-nginx od Kubernetes. *ingress-nginx*. `https://github.com/kubernetes/ingress-nginx`. Data dostępu: 2025-06-23. 2025.

[44] Dokumentacja techniczna cert-manager. *cert-manager*. `https://cert-manager.io/docs/`. Data dostępu: 2025-06-23. 2025.

- **Automatyzacja procesów:** możliwa automatyzacja cyklu życia certyfikatów
- **Przydatne integracje:** oferuje integracje z Ingress i popularnymi urzędami certyfikacji (np. Let's Encrypt)
- **Scentralizowane zarządzanie certyfikatami:** pełni funkcję scentralizowanego miejsca, w którym w sposób zautomatyzowany zarządzane są certyfikaty

Wady:

- **Złożoność:** początkowa konfiguracja może być skomplikowana

Certmanager jest kluczowy dla zapewnienia bezpiecznej komunikacji i automatyzacji zarządzania certyfikatami w Kubernetes.

8.6.4. Inne niestandardowe elementy klastra

- **Containerd:** lekkie środowisko uruchomiania i działania kontenerów zarządzające cyklem życia kontenerów
- **Flannel:** nakładka sieciowa (overlay network) do komunikacji między Podami na różnych węzłach
- **Network Policy controller:** kontroler polityk sieciowych implementujący zasady komunikacji między Podami
- **Spegel:** narzędzie do tworzenia lokalnego mirrora rejestru obrazów kontenerów, przyspieszające pobieranie obrazów i zmniejszające zależność od internetu
- **ServiceLB:** domyślny load balancer w K3s, który umożliwia korzystanie z usług typu LoadBalancer bez potrzeby posiadania load balancera od zewnętrznego dostawcy chmurowego

8.7. Infrastruktura rozwoju i wdrażania aplikacji

Infrastruktura związana z rozwojem i wdrażaniem aplikacji rozumiana jest w tym opisie jako wszystkie zasoby techniczne i narzędzia niezbędne do tworzenia, testowania, integracji oraz wdrażania oprogramowania. Jej najważniejszymi częściami są systemy zarządzania kodem źródłowym oraz narzędzia do automatyzacji procesów. Zaplanowanie i zidentyfikowanie tych elementów w kontekście konkretnego systemu jest kluczowe, ponieważ pozwala na efektywne zarządzanie procesem wytwarzania oprogramowania, minimalizuje ryzyko błędów oraz przyspiesza wdrażanie nowych funkcji i poprawek.

8.7.1. Serwer Jenkins

Z perspektywy projektowania infrastruktury systemu, wybór Jenkins **zapewnia dużą korzyść w postaci bezpośredniej kontroli** nad narzędziem automatyzacji **i działaniu w obrębie własnej, wybranej i kontrolowanej infrastruktury systemu**. Pozwala to maksymalnie dostosowywać działanie narzędzia automatyzacji do indywidualnych i zmieniających się potrzeb projektu. **Więcej na temat samego narzędzia Jenkins napisane jest w kontekście automatyzacji procesów związanej z rozwojem systemu i podejścia DevOps, czyli w punkcie 9.**

8.7.2. Usługi GitHub

GitHub, mimo alternatyw umożliwiających własny hosting, jest wyborem dla Passed Pawn ze względu na swoją **prostotę użytkowania, bogactwo funkcjonalności zarządzania przechowywanym kodem źródłowym i wspomagania automatyzacji oraz ze względu na dobrą znajomość zespołu projektowego Passed Pawn z platformą.**

Najważniejsze usługi i funkcjonalności GitHub wykorzystywane w Passed Pawn to:

- Hosting repozytoriów Git
- Pull/Merge Requests do przeglądu i integracji kodu
- GitHub Actions do automatyzacji testów
- Rejestr OCI do przechowywania artefaktów wdrożeniowych
- Integracja z narzędziami do analizy kodu — GitHub Copilot
- Mechanizmy (branch rules) egzekwowania ograniczeń we wprowadzaniu zmian w kodzie źródłowym, przestrzegania określonych protokołów wprowadzania zmian

Trzeba przy tym zaznaczyć, że wspomniane funkcjonalności są wykorzystywane w formie **Oprogramowania jako Usługi (Software as a Service - SaaS)**, co przekłada się na ograniczoną kontrolę nad nimi i uzależnienie od usługodawcy — GitHub.

8.8. Serwer Keycloak

Z perspektywy analizy infrastruktury, Keycloak, podobnie jak Jenkins, **zapewnia dużą korzyść w postaci bezpośredniej kontroli** nad narzędziem i **działaniu w obrębie własnej, wybranej i kontrolowanej infrastruktury systemu**. Pozwala to maksymalnie dostosowywać zarządzanie tożsamością i dostępami do indywidualnych i zmieniających się potrzeb projektu. **Więcej na temat samego narzędzia Keycloak napisane jest w kontekście bezpieczeństwa systemu, czyli w punkcie 10.**

8.9. Wykorzystanie dobrych praktyk Infrastructure as Code (IaC), GitOps

8.9.1. Geneza IaC.

Przed pojawieniem się praktyk IaC (pol. Infrastruktura jako Kod) i ich zyskaniem uznania w środowisku IT, tradycyjnym podejściem do zarządzania konfiguracją serwerów, sieci i innych komponentów infrastruktury było podejście manualne. Charakteryzowało się małym stopniem automatyzacji oraz deklaratywności konfiguracji i wysokim stopniem ręcznego oraz imperatywnego zarządzania poprzez krokowe wykonywanie przygotowanych instrukcji przez człowieka. Ten model wiązał i w dalszym ciągu wiąże się z podatnością na błędy, niekonsekwentnością osiąganych rezultatów oraz czasochłonnością. IaC pojawiło się jako odpowiedź na te problemy z zamysłem przeniesienia procesów zarządzania infrastrukturą na warstwę programistyczną, poprzez opisywanie pożądanych cech w plikach konfiguracyjnych używających języków takich jak YAML, JSON czy specjalistycznych DSL (Domain-Specific Language)[45]

8.9.2. Definicja i znaczenie IaC

Grzegorz Gnych, specjalista infrastruktury IT, definiuje pojęcie Infrastruktury jako Kodu w sposób: "[...] fundamentalna zmiana w podejściu do zarządzania infrastrukturą IT, gdzie zasoby techniczne są definiowane i zarządzane poprzez kod źródłowy, a nie ręczną konfigurację.". Lista potencjalnych wyzwań i popularnych problemów[46]

Zalety:

[45] Grzegorz Gnych. *Czym jest Infrastruktura jako kod (Infrastructure as Code)? – Kompendium wiedzy*. `https://nflo.pl/baza-wiedzy/czym-jest-infrastruktura-jako-kod-infrastructure-as-code-kompendium-wiedzy/#Integracja-z-Tenable-i-Rapid7`. Data dostępu: 2025-06-25.

[46] [Tamże].

- **Potencjał do automatyzacji** — zarządzanie infrastrukturą przez kod otwiera szerokie możliwości do standaryzacji i automatyzacji procesów zarządzania infrastrukturą, co zwiększa szybkość i niezawodność wdrożeń
- **Powtarzalność** — zwiększenie powtarzalności i spójność środowisk, co minimalizuje błędy ludzkie i różnice między środowiskami
- **Wersjonowanie** — umożliwienie prostego wersjonowania konfiguracji, co ułatwia śledzenie zmian, audyt i szybkie wycofywanie niepożądanych modyfikacji
- **Skalowalność** — łatwość skalowania infrastruktury oraz możliwość szybkiego tworzenia środowisk testowych i produkcyjnych
- **Ułatwiona kolaboracja** — usprawnienie współpracy zespołów developerskich i operacyjnych, poprzez ułatwienie analizy funkcjonującej oraz proponowanej infrastruktury, której pożądany stan opisany jest w kodzie

Wady:

- **Złożoność** — wyższa wstępna złożoność zarządzania kodem infrastruktury, wymaganie specjalistycznej wiedzy i umiejętności programistycznych w zespole
- **Bezpieczeństwo i ryzyko błędów** — błąd w kodzie, w połączeniu z automatyzacją, może błyskawicznie zniszczyć całą infrastrukturę poprzez np. przypadkowe usunięcie bazy danych
- **Dodatkowe problemy zarządzania stanem** — w przypadku narzędzi takich jak Terraform, sposób przechowywania stanu infrastruktury może ulec uszkodzeniu, np. jeśli wystąpiły pojedynczne zmiany infrastruktury wprowadzone ręcznie

Znajomość tych wad i zalet jest przydatna w kontekście analizy stopnia automatyzacji wdrażania zmian infrastruktury systemu, np. przy rozważaniu automatyzacji wdrażania zmian punktów krytycznych systemu, wymagających maksymalnej ostrożności i odpowiedniego audytu, czyli np. serwera autoryzacyjnego lub baz danych.

8.9.3. Zakres wykorzystania praktyk IaC

Zakres wykorzystania praktyk IaC dla Passed Pawn obejmuje:

- **Wbudowane zasoby Kubernetes (np. Deployment, Statefulset, Service, Ingress itp.)** — przechowywane w manifestach YAML i pakowane za pomocą tzw. Helm Chartów, które dodatkowo luzują sprzężenie zasobów z wybranymi aspektami konfiguracyjnymi
- **Longhorn** — zarządzany za pomocą manifestów YAML obsługujących oficjalny Helm Chart Longhorn
- **Nginx-Ingress-Controller** — zarządzany za pomocą manifestów YAML obsługujących oficjalny Helm Chart Nginx-Ingress-Controller od Kubernetes
- **Cert-Manager** — zarządzany za pomocą manifestów YAML obsługujących oficjalny Helm Chart Cert-Manager
- **Jenkins** — zarządzany za pomocą manifestów YAML obsługujących oficjalny Helm Chart dla oficjalnego Operatora Jenkins do Kubernetes
- **Keycloak** — zarządzany za pomocą rozszerzenia Keycloak Provider do języka HCL (Hashicorp Configuration Language)

Dzięki tak szerokiemu zastosowaniu praktyk IaC w przypadku Passed Pawn, zarządzania Infrastrukturą systemu jest wersjonowane, przejrzyste, audytowalne i powtarzalne.

8.9.4. Geneza GitOps

W okresie, gdy Kubernetes ugruntowywał swoją pozycję jako platforma orkiestracji kontenerów, wiele organizacji zaczęło eksperymentować w celu wypracowania wydajnych metod zarządzania konfiguracją, wdrażaniem oraz automatyzacją aplikacji, przy zapewnieniu stabilności i powtarzalności zdefiniowanych procesów. W roku 2017 koncepcję GitOps spopularyzowała firma Weaveworks, która wypracowała podejście oparte o praktyki Infrastructure as Code (IaC), w którym to podejściu Git pełni rolę jedynego, pojedynczego źródła prawdy o stanie systemu. Przedstawione podejście eliminowało gamę problemów wynikających z natury manualnego zarządzania infrastrukturą oraz wdrażaniem klastrów i aplikacji. Dodatkowo upraszczało zarządzanie wdrożeniami, ułatwiając audyt wprowadzanych zmian, dzięki centralnemu zarządzaniu kodem w Git.[47]

8.9.5. Definicja i znaczenie GitOps

Andrzej Ochman, ekspert z dziedziny inżynierii DevOps oraz Architektury rozwiązań, definiuje GitOps jako: "[...] podejście do zarządzania infrastrukturą i aplikacjami, w którym Git pełni rolę jedynego źródła prawdy. Oznacza to, że cała konfiguracja systemu — od definicji zasobów Kubernetesa, po polityki bezpieczeństwa — jest przechowywana w repozytorium Git. Zmiany w infrastrukturze są realizowane wyłącznie poprzez aktualizację kodu w repozytorium. Są też automatycznie synchronizowane z rzeczywistym stanem systemu."[48]

Zalety:

- **Centralizacja** — centralizacja zarządzania infrastrukturą poprzez pojedyncze, audytowalne repozytorium Git jako jedyne źródło prawdy
- **Przejrzystość** — pełna przejrzystość i kontrola zmian dzięki mechanizmom kontroli wersji i przeglądu kodu (pull requesty)
- **Automatyzacja** — automatyzacja synchronizacji stanu infrastruktury z repozytorium zapewnia spójność i wzmacnia niezawodność systemu
- **Audytowalność** — ułatwienie walidacji i audytów kodu infrastruktury dzięki historii zmian oraz możliwości szybkiego przywracania poprzednich wersji.
- **Ułatwiona kolaboracja** — podobnie jak w podejściu IaC, wsparcie kolaboracji w pracy nad infrastrukturą poprzez zwiększenie przejrzystości, audytowalności i analizy istniejącej i proponowanej infrastruktury

Wady:

- **Dostosowanie** — wymaga dostosowania procesów pracy zespołu do modelu zarządzania przez Git, np. poprzez udzielenie dostępu do zdalnych repozytoriów narzędziom automatyzującym
- **Złożoność** — początkowa złożoność wdrożenia i konfiguracji narzędzi GitOps może być wysoka
- **Synchronizacja** — wprowadzenie gamy potencjalnych wyzwań dotyczących synchronizacji stanu, np. w przypadku błędów w automatycznych procesach

8.9.6. Zakres wykorzystania praktyk GitOps

Zakres wykorzystania praktyk GitOps dla Passed Pawn obejmuje te same części infrastruktury systemu co w przypadku zastosowania praktyk IaC. Tam gdzie infrastruktura zarządzana jest w postaci kodu,

[47] Andrzej Ochman. *GitOps cz. I. Wprowadzenie do GitOps i jego rola w konteneryzacji*. `https://linuxpolska.com/pl/baza-wiedzy/blog/gitops-wprowadzenie-jego-rola-w-konteneryzacji-cz-i/`. Data dostępu: 2025-06-25. 2025.

[48] [Tamże].

tam jest ona również wersjonowana, przechowywana zdalnie w repozytoriach kodu oraz na bieżąco odzwierciedlana w stanie na serwerze dedykowanym od OVH. Jedynym wyjątkiem jest infrastruktura serwera Autoryzacyjnego Keycloak, gdzie zmiany nie są synchronizowane w sposób automatyczny, tylko w sposób ręczny, ale w dalszym ciągu są przechowywane jako kod. Wiąże się to ze szczególną ostrożnością w obcowaniu z narzędziem i ustanowieniem wymogu wprowadzania zmian przez człowieka w celu ułatwienia potencjalnie szybkiego wycofania tych zmian, zupełnego zrezygnowania z ich wdrażania lub minimalizowania powstałych szkód.

9. Automatyzacja procesów, podejście DevOps

Manualne zarządzanie procesami utrzymywania i wytwarzania oprogramowania może wiązać się z wyzwaniami i problemami takimi jak błędy ludzkie oraz słaba powtarzalność i skalowalność procesów. Praktykowanie IaC stanowi znaczący krok w kierunku minimalizacji imperatywnego stylu zarządzania infrastrukturą oprogramowania przez ręczne wykonywanie wielu instrukcji. Koniec końców jednak nawet Infrastruktura jako Kod musi kiedyś zostać zaaplikowana, a sam proces jej aplikowania może być mniej lub bardziej skomplikowany i składać się z mniejszej, bądź większej ilości kroków. Automatyzacja tego procesu jest de facto dopełnieniem wielkiej wizji wyeliminowania manualnego zarządzania infrastrukturą oprogramowania. Odpowiednia automatyzacja jest w stanie posunąć o krok dalej te same zalety, które zapewnia IaC, których istota leży właśnie w wyeliminowaniu czynnika ludzkiego z całego równania. Jest to, obok traktowania repozytorium jako jedynego źródła prawdy o infrastrukturze, dokładnie jeden z dwóch największych postulatów praktyk GitOps, które zostały przedstawione w poprzednim rozdziale. W wypadku tego rozdziału mowa natomiast nie tylko o automatycznej synchronizacji kodu infrastruktury z repozytorium, ale o ogólnej automatyzacji procesów rozwoju i utrzymania Passed Pawn.

Zalety automatyzacji procesów:

- **Przyspieszenie rozwoju w szerszej perspektywie** — znaczące skrócenie czasu realizacji wdrożeń i zmian dzięki eliminacji manualnych czynności
- **Redukcja błędów i powtarzalność procesów** — zmniejszenie ryzyka błędów ludzkich poprzez standaryzację i zapewnienie powtarzalności procesów
- **Spójność** — zapewnienie jednolitości środowisk i konfiguracji dzięki poprzez standaryzację i zapewnienie powtarzalności procesów
- **Skalowalność** — możliwość szybkiego dostosowania się do rosnących wymagań i zmian w infrastrukturze

Wady automatyzacji procesów:

- **Koszty wdrożenia** — potencjalnie wysokie nakłady czasowe i finansowe na początkową konfigurację i szkolenie osób odpowiedzialnych za automatyzację
- **Utrzymanie** — konieczność ciągłej aktualizacji i monitorowania skryptów automatyzujących, aby uniknąć błędów
- **Ryzyko błędów** — mówiąc o wdrożeniach, skutki błędów w automatyzacji mogą być szczególnie dotkliwe i obejmować wiele środowisk jednocześnie
- **Bezpieczeństwo** — narzędzia automatyzujące stanowią nowy, konieczny do uwzględnienia i zabezpieczenia wektor ataku i podatności systemu

9.1. Metodologia rozwoju i utrzymywania aplikacji Passed Pawn

Proces rozwoju i utrzymywania aplikacji Passed Pawn odbywał się z uwzględnieniem podejścia DevOps do wytwarzania oprogramowania oraz z zastosowaniem praktyk CI/CD (Continuous Integration, Continuous Deployment). Celem takiego podejścia i zastosowania takich praktyk jest umożliwienie efektywnego zarządzania infrastrukturą systemu przez członków zespołu developerskiego. Z uwagi na tło i doświadczenie developerskie wszystkich członków zespołu Passed Pawn, to właśnie ułatwianie pracy z częścią operacyjną systemu stanowi główny obiekt usprawnień idących z duchem DevOps.

9.1.1. Zastosowanie praktyk DevOps

Rozumienie pojęcia DevOps, od czasów jego pierwszego sformułowania w 2009 roku przez Patricka Debois, stało się rozmyte, a definicji tego pojęcia można znaleźć wiele i potrafią się między sobą bardzo różnić. Definicją uznawaną w procesie projektowania systemu Passed Pawn jest definicja pochodząca od autora pojęcia. W posłowiu drugiej edycji książki "The DevOps Handbook", Patrick Debois zdefiniował DevOps w następujący sposób: "[...] everything you do to overcome the friction between silos. All the rest is plain engineering."[49], czyli "[...] wszystko, co robi się, by przezwyciężyć tarcie między silosami. Cała reszta to zwykła inżynieria.". Mowa tutaj w domyśle o silosach w postaci domeny deweloperskiej/rozwojowej oprogramowania (Dev) oraz domeny operacyjnej (Ops). Autor dodaje, że ta definicja podkreśla, że samo budowanie technologii nie wystarcza do praktykowania DevOps – trzeba również mieć stały cel, którym jest usunięcie punktu tarcia. Autor tłumaczy, że ten punkt tarcia będzie przesuwał się razem z usuwaniem bieżących blokad. Ciągła ocena wąskiego gardła w procesie wytwarzania oprogramowania jest kluczowa. Zauważa on również, że w dzisiejszych czasach organizacje nieustannie optymalizują pipeline'y i automatyzację, ale nie pracują nad innymi punktami tarcia, które powodują blokady, a to jest właśnie obecnie jedno z głównych wyzwań wytwarzania oprogramowania[50].

Przykłady praktyk zastosowanych w myśl DevOps:

- Automatyzacja procesów budowania, testowania i wdrażania
- Ciągła integracja i ciągłe dostarczanie (CI/CD)
- Wspólna odpowiedzialność za działanie wdrożeń zespołów developerskich i operacyjnych
- Wykorzystanie infrastruktury jako kodu (IaC) do zarządzania środowiskami
- Kultura ciągłego uczenia się i doskonalenia procesów

Zastosowanie praktyk DevOps w rozwoju i utrzymywaniu projektu Passed Pawn umożliwia zwiększenie efektywności pracy zespołu, skrócenie czasu wdrożeń oraz poprawę jakości i stabilności aplikacji. Ponadto pomaga budować przystępną dla deweloperów metodologię pracy nad rozwojem infrastruktury systemu i procesami jego wdrażania oraz utrzymywania.

9.1.2. Zastosowanie praktyk CI/CD

CI, czyli Ciągła Integracja (ang. Continuous Integration) to praktyka inżynieryjna polegająca na częstym zatwierdzaniu zmian w kodzie we wspólnym repozytorium, najlepiej kilka razy dziennie, oraz automatycznym budowaniu tych zmian. Zmiany te są kompilowane razem z innymi równoległymi modyfikacjami w systemie, co pozwala na wczesne wykrywanie problemów z integracją między pracą wielu programistów uczestniczących w projekcie. Błędy kompilacji wynikające z nieudanej integracji traktowane są jako najwyższy priorytet dla całego zespołu, a praca zwykle wstrzymuje się do momentu ich naprawienia.

W połączeniu z automatycznym testowaniem, ciągła integracja umożliwia również testowanie zintegrowanej wersji, dzięki czemu możemy sprawdzić nie tylko, czy kod nadal się poprawnie kompiluje, ale także czy zachowuje poprawność funkcjonalną. Jest to również najlepsza praktyka w budowaniu solidnych i elastycznych systemów oprogramowania.[51]

[49] Gene Kim i in. *The DevOps Handbook*. 2st. Data dostępu: 2025-06-25. IT Revolution Press, 2016.

[50] [Tamże].

[51] Microsoft. *Continuous Integration and Continuous Delivery*. `https://microsoft.github.io/code-with-engineering-playbook/CI-CD/`. Data dostępu 2025-06-25. 2024.

CD, czyli Ciągłe Dostarczanie (ang. Continuous Delivery) rozwija koncepcję Ciągłej Integracji (CI), testując również wdrożenia zintegrowanego kodu w środowisku będącym repliką docelowego środowiska produkcyjnego. Pozwala to na wczesne wykrycie nieprzewidzianych problemów operacyjnych wynikających z wprowadzanych zmian oraz identyfikację luk w testach.

Celem tych praktyk jest zapewnienie, że główna gałąź kodu (main branch) jest zawsze gotowa do wdrożenia, co oznacza, że w razie potrzeby można pobrać aktualną wersję z tej gałęzi i wdrożyć ją na produkcji.[52]

Przykłady praktyk CI/CD:

- Automatyczne uruchamianie testów jednostkowych i integracyjnych przy zmianie kodu
- Automatyczne budowanie i pakowanie aplikacji
- Automatyczne wdrażanie na środowiska testowe
- Weryfikacja jakości i bezpieczeństwa kodu w pipelinie CI/CD

Zaadoptowanie CI/CD w rozwoju systemu Passed Pawn w postaci testów automatycznych oraz automatyzacji budowania i wdrażania aplikacji i innych komponentów systemu pozwoliło ułatwić w proces integrowania zmian z różnych części systemu i pochodzących od różnych członków zespołu. Automatyzacja pozwala zwiększyć częstotliwość scalania zmian oraz testowania kodu, dzięki odciążeniu członków zespołu z manualnych czynności. Automatyczne budowanie i wdrażanie aplikacji pomaga również w testach manualnych, ponieważ zapewnia istnienie środowiska deweloperskiego, które zawsze zawierało zsynchronizowaną z najnowszymi zmianami wersję aplikacji. Dzięki temu deweloperzy mają większą i częstszą szansę odnaleźć błędy w systemie.

9.2. Jenkins - główne narzędzie do automatyzacji procesów

Jenkins to otwartoźródłowy serwer automatyzacji, który umożliwia budowanie, testowanie i wdrażanie oprogramowania. Jego głównym zadaniem jest automatyzacja procesów, co pozwala na szybsze i niezawodne dostarczanie nowych funkcjonalności.

Zalety:

- **Otwartoźródłowy i darmowy** — dostępny bezpłatnie, co pozwala na szerokie zastosowanie bez dodatkowych kosztów licencyjnych
- **Ogromna liczba dostępnych wtyczek** — pozwala na integrację z wieloma narzędziami i rozszerzanie funkcjonalności
- **Elastyczność i możliwość dostosowania do różnych środowisk** — łatwość konfiguracji pod kątem specyficznych wymagań projektów
- **Szerokie wsparcie dla różnych języków programowania i narzędzi** — umożliwia pracę z wieloma technologiami i frameworkami
- **Duża społeczność użytkowników i szerokie wsparcie dokumentacyjne** — ułatwia rozwiązywanie problemów i rozwój systemu

Wady:

- **Złożoność konfiguracji i utrzymania** — wymaga doświadczenia i nakładów czasu na poprawne ustawienie i zarządzanie

[52] Microsoft, *Continuous Integration and Continuous Delivery*.

- **Potencjalne problemy z bezpieczeństwem przy niewłaściwej konfiguracji** — ryzyko wystąpienia luk bezpieczeństwa, jeśli system nie jest odpowiednio zabezpieczony
- **Słabe wsparcie wielkiej ilości wtyczek i rozszerzeń** — niektóre wtyczki mogą być przestarzałe lub słabo utrzymywane
- **Ograniczone możliwości bazowe — duża zależność od wtyczek i rozszerzeń** — podstawowa funkcjonalność jest ograniczona bez dodatkowych rozszerzeń

9.2.1. Najpopularniejsze alternatywne rozwiązania do Jenkinsa

GitLab Runner

GitLab Runner to agent wykonujący zadania CI/CD w ramach platformy GitLab. Umożliwia uruchamianie wielu zadań równocześnie, wspiera różne środowiska wykonawcze, w tym Docker i SSH, oraz oferuje automatyczne skalowanie na różnych chmurach i hipernadzorcach.

ArgoCD + Argo Workflows

Argo Workflows to silnik orkiestracji zadań działający natywnie na Kubernetes, który pozwala definiować złożone pipeline'y CI/CD. ArgoCD natomiast służy do ciągłego dostarczania i zarządzania aplikacjami na Kubernetes. Wspólnie stanowią nowoczesne rozwiązanie dla środowisk chmurowych i kontenerowych.

Usługi chmurowe: Azure DevOps, AWS CodePipeline, CodeBuild, CodeDeploy, GCP Cloud Build, Cloud Deploy, Cloud Test Lab

Mowa tutaj o usługach automatyzacji od największych dostawców chmurowych - Azure, AWS, GCP. Oferują kompleksowe, zintegrowane rozwiązania CI/CD w chmurze, umożliwiające automatyzację budowania, testowania i wdrażania aplikacji. Oferują skalowalność, integrację z innymi usługami chmurowymi oraz wsparcie dla różnych języków i platform, co ułatwia zarządzanie procesami DevOps w środowiskach wielochmurowych i hybrydowych.

9.2.2. Zakres zastosowania Jenkinsa

Jenkins jest wykorzystywany głównie do automatyzacji procesów budowania i wdrażania oprogramowania. Monitoruje repozytoria kodu źródłowego i uruchamia zadania w odpowiedzi na zmiany, co pozwala na szybkie wykrywanie błędów i utrzymanie wysokiej jakości kodu. Sprawdza się zarówno w małych zespołach, jak i w dużych organizacjach, które potrzebują elastycznego i rozbudowanego narzędzia CI/CD.

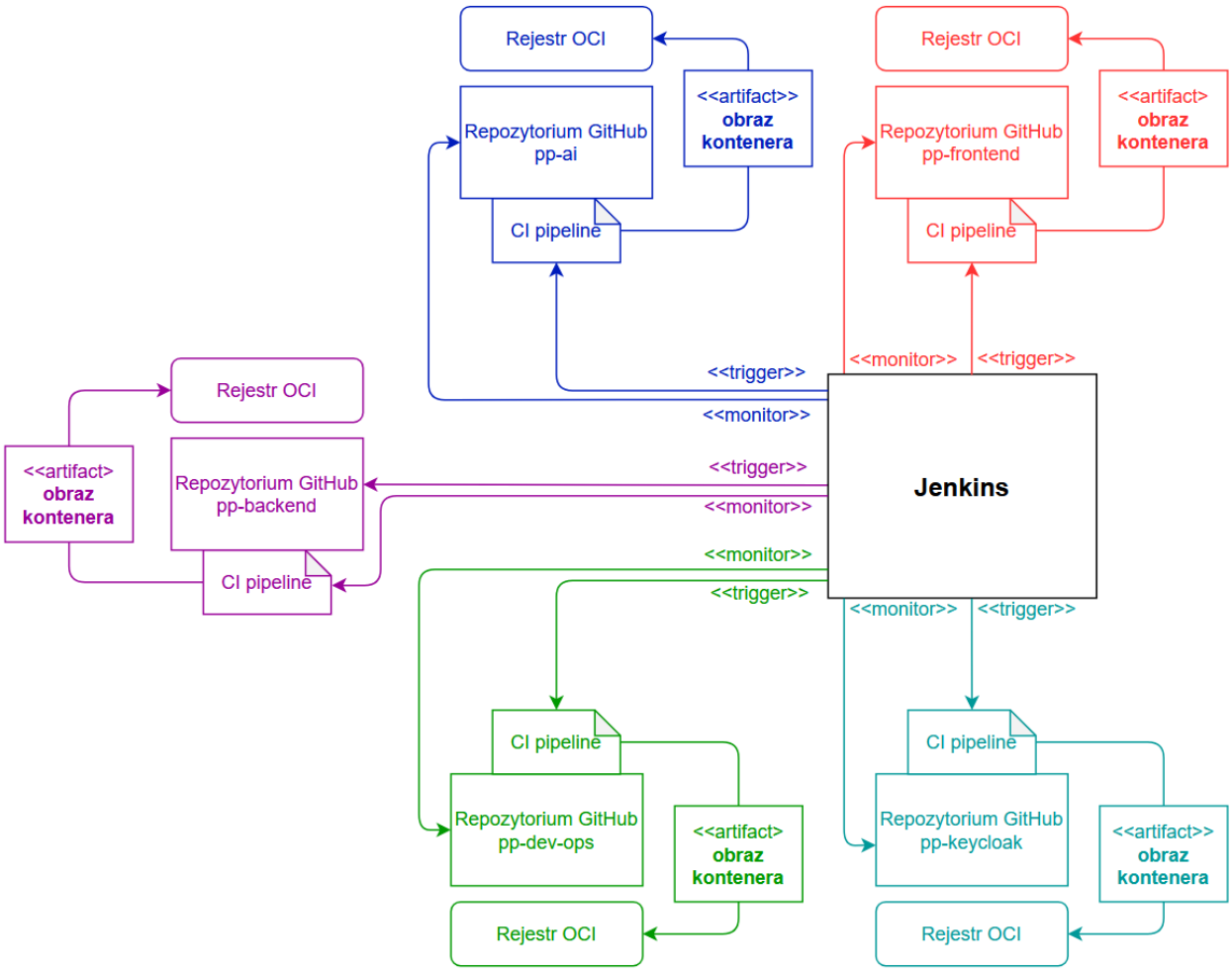
9.2.3. Uzasadnienie wyboru Jenkinsa

Jenkins stanowi główny serwer automatyzacji dla Passed Pawn, ze względu na swoją elastyczność, pełną kontrolę w zarządzaniu, szeroką dostępność materiałów szkoleniowych i dokumentacji technicznej oraz potencjał do automatyzacji testów o wyjątkowych i skomplikowanych potrzebach. Mimo pewnych wad, takich jak skomplikowana konfiguracja, czy malejące wsparcie wtyczek, jego dojrzałość i wciąż pozostała popularność czynią go solidnym wyborem na obecnym etapie rozwoju systemu.

9.3. Podział odpowiedzialności za pipeline-y

W proponowanej architekturze każde repozytorium jest odpowiedzialne za zarządzanie własnymi pipeline-ami CI, co pozwala zespołom/członkom zespołu na niezależne definiowanie i utrzymywanie procesów automatyzacji zgodnie z ich potrzebami. Jednocześnie wszystkie pipeline-y muszą stosować się do wspólnego, centralnie utrzymywanego szablonu, który zapewnia spójność oraz standaryzację deklarowania i przechowywania pipelineów CI, wymuszając, by znajdowały się w katalogu /cicd/pipelines/ci.jenkins.

Poniżej znajduje się diagram obrazujący podział odpowiedzialności za pipeline-y CI i schemat działania systemu automatyzacji w procesie wykrywania zmian w repozytoriach i uruchamiania pipeline-ów odpowiedzialnych za budowanie oraz wypychanie artefaktów wdrożeniowych do rejestrów OCI odpowiadających repozytoriom.



Rysunek 20: Diagram architektury pipline-ów CI

Serwer Jenkins monitoruje cyklicznie stan repozytoriów w celu wykrycia zmian na głównej odnodze (main branch). Jeżeli znajdzie zmiany, uruchamia odpowiadający danemu repozytorium pipeline CI, którego zadaniem jest zbudować i wypchnąć odpowiadający mu artefakt wdrożeniowy do odpowiadającego mu Rejestru OCI. Rolą osób odpowiedzialnych za zarządzanie danym repozytorium jest zdefiniować pipeline i dopilnować, by spełniał swój cel. Po pomyślnym ukończeniu działania

pipeline-u uruchamiany jest kolejny pipeline, za którego utrzymanie tym razem odpowiedzialne są osoby zarządzające repozytorium infrastruktury systemu, czyli pp-dev-ops. Pipeline ten uwzględnia nowo zbudowany i wypchnięty artefakt wdrożeniowy i buduje oraz wypycha zaktualizowany o wersję tego artefaktu Helm Chart aplikacji, czyli pojedynczą jednostkę wdrożeniową aplikacji przeznaczoną do wdrożenia na klastrze Kubernetes. To wszystko zapewnia odpowiednie wersjonowanie poszczególnych komponentów aplikacji oraz aplikacji jako całości. Wersja ostatecznej jednostki wdrożeniowej aplikacji ma następujący format:

<yyyy-MM-dd'T'HH-mm-ss>C<hash-ostatniego-commita-main-branch-pp-dev-ops>--<wersja-ostatniego-artefaktu-aplikacji-frontend>--<wersja-ostatniego-artefaktu-aplikacji-backend>--<wersja-ostatniego-artefaktu-aplikacji-ai>,

gdzie wersje artefaktów konkretnych aplikacji, to również

<yyyy-MM-dd'T'HH-mm-ss>C<hash-ostatniego-commita-main-branch-repozytorium>.

Przykładowa nazwa artefaktu wdrożeniowego z repozytorium pp-frontend (dodatkowo poszerzona o nazwę rejestru):

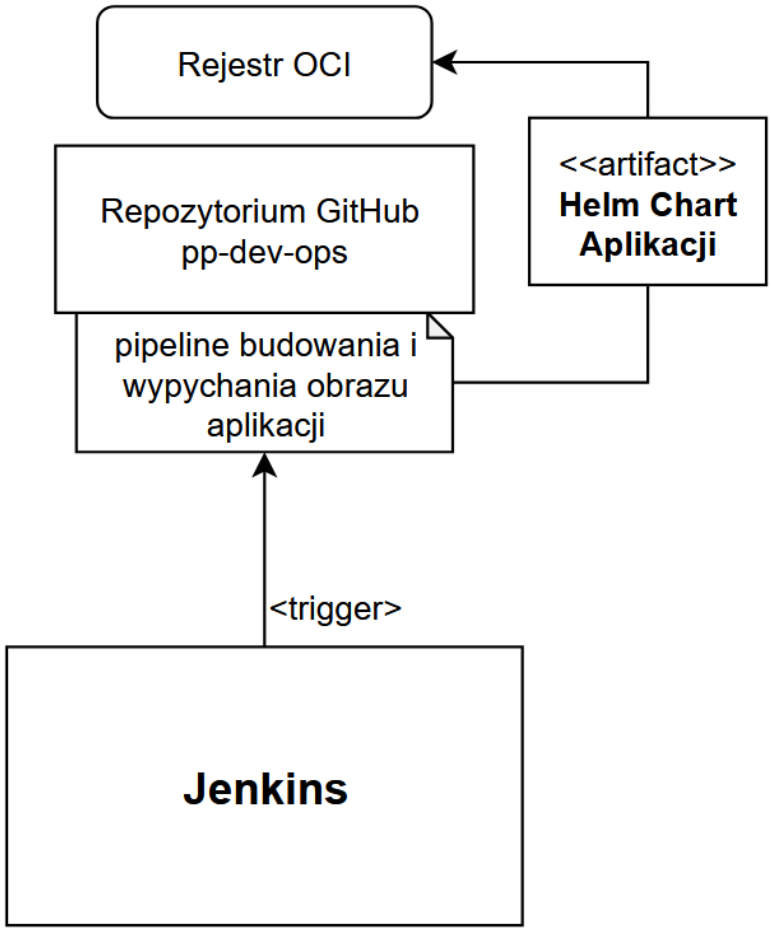
ghcr.io/passed-pawn-dev/pp-frontend/pp-frontend:2025-06-21T20-04-21C1efadfd

Przykładowa nazwa artefaktu wdrożeniowego całej aplikacji (również poszerzona o nazwę rejestru):

ghcr.io/passed-pawn-dev/pp-dev-ops/charts/pp-app:2025-06-22T14-24-13C1436ff7--2025-04-13T20-03-05C259dfac--2025-06-22T14-21-23C6dcc528--2025-05-29T09-44-55Czda321h

Przedstawiony sposób wersjonowania aplikacji jako całości oraz jej poszczególnych elementów osobno umożliwia łatwą analizę wersji kodu, co jest szczególnie przydatne w przypadku wystąpienia awarii bądź błędów.

Poniżej znajduje się diagram przedstawiający schemat budowania i wypychania obrazu całej aplikacji, co ma miejsce po każdym zakończonym sukcesem działaniu pipeline-u CI.



Rysunek 21: Schemat architektury budowania i wypychania obrazu aplikacji w formie Helm Chartu

Zalety przedstawionej architektury:

- **Autonomia zespołów** — pozwala na szybkie dostosowanie pipeline-ów do specyficznych potrzeb projektu bez konieczności oczekiwania na centralne zmiany
- **Standaryzacja** — centralny szablon wymusza jednolity sposób deklarowania pipeline-ów CI w każdym repozytorium
- **Skalowalność** — rozproszone zarządzanie pipelina-ami ułatwia skalowanie systemu wraz ze wzrostem liczby projektów i zespołów
- **Łatwiejsze utrzymanie** — aktualizacje i poprawki w szablonie są propagowane do wszystkich repozytoriów, co zmniejsza nakład pracy przy utrzymaniu spójności

Wady przedstawionej architektury:

- **Potencjalne konflikty** — zmiany w centralnym szablonie mogą wymagać dostosowań w wielu pipeline-ach, co może generować dodatkową pracę
- **Wymagana dyscyplina zespołów** — zespoły/członkowie zespołu muszą być świadomi i konsekwentnie stosować się do ustalonych standardów

Uzasadnienie: Podział odpowiedzialności, w którym każde repozytorium zarządza własnym pipeline'em, a jednocześnie korzysta z centralnego szablonu, łączy zalety elastyczności i standaryzacji. Pozwala to zespołom/członkom zespołu na szybkie reagowanie na zmiany i nowe pojawiające się wymagania, jednocześnie utrzymując spójność architektury procesów CI/CD w całym systemie. Takie podejście wspiera skalowalność i ułatwia zarządzanie dużą liczbą projektów, co jest kluczowe w perspektywie rozrostu systemu lub jego podziału na mniejsze części, np. w ramach migracji do architektury mikroserwisowej. Wdrożenie centralnego szablonu jako punktu odniesienia minimalizuje ryzyko rozproszenia standardów oraz ułatwia utrzymanie i rozwój infrastruktury automatyzacji. 1-2 zdania opisu diagramu

9.4. Integracja Jenkins — Kubernetes

Jenkins jest zaimplementowany przy wykorzystaniu Jenkins-Operatora, co pozwala na natywne zarządzanie instancjami Jenkins w środowisku Kubernetes z podejściem deklaratywnym i Infrastructure as Code (IaC). Zalety takiego rozwiązania to automatyczne skalowanie, łatwiejsze zarządzanie cyklem życia Jenkinsa, wbudowane praktyki bezpieczeństwa oraz integracja z Kubernetes i opcjonalnie chmurami publicznymi (AWS, Azure, GCP). Wadą może być większa złożoność początkowej konfiguracji oraz konieczność posiadania wiedzy o Kubernetes i operatorach. Kolejną przy tym zaletą jest zestaw prekonfigurowanych elementów, które dostarcza Operator, efektywnie ułatwiając i skracając pracę osobom mniej doświadczonym z Jenkinsem.

Alternatywnym rozwiązaniem jest uruchomienie Jenkinsa jako zwykłego deploymentu Kubernetes, co jest prostsze, ale nie oferuje pełnej automatyzacji zarządzania, skalowania i aktualizacji, które zapewnia operator, ani nie ma podobnych możliwości IaC, czy GitOps. To podejście w rozsądnym scenariuszu wymaga też zdobycia odpowiedniej wiedzy i doświadczenia z Jenkinsem, które pozwalałyby na jego samodzielne zabezpieczenie i skonfigurowanie do indywidualnych potrzeb.

Wykorzystanie Jenkins-Operatora jest uzasadnione podejściem GitOps i IaC, ponieważ operator pozwala na deklaratywne zarządzanie konfiguracją Jenkinsa, co ułatwia wersjonowanie, automatyzację i powtarzalność wdrożeń. Operator wspiera pełny lifecycle management Jenkinsa, co wpisuje się w praktyki GitOps, gdzie zmiany infrastruktury i konfiguracji są zarządzane przez repozytoria kodu, a automatyczne procesy zapewniają spójność środowisk. Ponadto, co zostało już zaznaczone, oferuje on szereg konfiguracji domyślnych, które zapewniają bezpieczeństwo, niezawodność i szeroką funkcjonalność automatyzacji.

Najważniejsze komponenty wbudowane Jenkins Operatora

- **Jenkins Custom Resource (CR)** — definiuje instancję Jenkinsa w Kubernetes, umożliwiając deklaratywne zarządzanie konfiguracją i parametrami działania Jenkinsa, co pozwala na łatwe utrzymanie i replikację środowisk
- **Controller Operatora** — odpowiada za monitorowanie i zarządzanie zasobami Jenkinsa w klastrze, automatycznie wdraża zmiany, aktualizacje przy jednoczesnym zapewnieniu bezpieczeństwa, dzięki odpowiedniej, domyślnej konfiguracji
- **Integracja z Kubernetes i chmurą** — operator wspiera mechanizmy backupu, obserwowalności i polityk bezpieczeństwa, a także integrację z usługami chmurowymi, co zwiększa niezawodność i ułatwia zarządzanie w środowiskach hybrydowych i multi-cloud

9.5. GitHub Actions - automatyzacja testów

GitHub Actions to narzędzie do automatyzacji przepływów pracy (workflow) w repozytoriach GitHub, które pozwala na definiowanie procesów takich jak budowanie, testowanie, pakowanie, wydawanie i wdrażanie aplikacji bezpośrednio w repozytorium.

Zalety:

- **Integracja z repozytoriami Github** — integracja bezpośrednio z GitHub, co ułatwia konfigurację i zarządzanie
- **Elastyczność i wieloplatformowość** — obsługa wielu języków i środowisk, elastyczność w definiowaniu workflow w plikach YAML
- **Kontrola środowiska wykonawczego** — możliwość korzystania z hostowanych runnerów lub własnej infrastruktury
- **Szerokie wsparcie i udostępnianie szablonów** — szeroka społeczność i gotowe akcje dostępne do użycia

Wady:

- **Ograniczone możliwości funkcjonalne**: mniej zaawansowane możliwości w porównaniu do narzędzi takich jak Jenkins, szczególnie przy bardzo złożonych pipeline'ach
- **Ograniczone możliwości konfiguracyjne**: ograniczenia w przypadku bardzo dużych i skomplikowanych projektów wymagających rozbudowanych konfiguracji

Alternatywy dla GitHub Actions

Do popularnych alternatyw należą: Jenkins, Azure Pipelines, GitLab CI i Argo Workflows. Jenkins wyróżnia się natomiast jako narzędzie open-source z szeroką gamą rozszerzeń i wsparciem dla bardziej złożonych pipeline'ów. Z tego powodu, w przypadku skomplikowania lub rozrośnięcia się wymagań dotyczących testowania systemu, testy automatyczne uruchamiane w GitHub Actions mogą zostać przeniesione na istniejącą już w systemie infrastrukturę Jenkins.

Zakres zastosowania GitHub Actions

GitHub Actions jest wykorzystywany w projekcie do automatycznego testowania głównej aplikacji serwerowej przed zatwierdzaniem wprowadzanych do niej zmian.

Uzasadnienie zastosowania GitHub Actions

GitHub Actions jest bardzo praktyczny w automatyzacji procesu walidacji i testowania aplikacji w projektach o umiarkowanej złożoności, ze względu na swoją prostotę obsługi i integrację z repozytoriami GitHub, w których na ten moment jest przechowywany kod systemu. W przypadku zrezygnowania ze zdalnych repozytoriów kodu w platformie GitHub lub w przypadku pojawienia się potrzeby obsługi bardziej złożonych pipeline'ów testowania, lepsze możliwości oferuje Jenkins, który jest już przystosowany do zaadoptowania bardziej złożonych procesów i integracji z różnymi platformami.

9.6. Git hooki - walidacja kodu

Git hooki to skrypty uruchamiane automatycznie w odpowiedzi na określone zdarzenia w repozytorium Git, umożliwiające dostosowanie i automatyzację procesów w cyklu życia oprogramowania.

Zalety:

- Możliwość automatycznego wykonywania kontroli przed wykonaniem operacji Git (np. commit, push).

- Zapewnienie spójności i jakości kodu na wczesnym etapie.

Wady:

- Hooki działają lokalnie i nie są domyślnie współdzielone w zespole, co wymaga dodatkowej konfiguracji.
- Mogą być pomijane lub wyłączane przez użytkowników.

9.6.1. Uzasadnienie użycia hooka pre-commit

Hook pre-commit pozwala na automatyczne uruchamianie testów lub linterów przed zatwierdzeniem zmian, co pomaga wykryć błędy i problemy z kodem jeszcze przed ich wprowadzeniem do repozytorium. Pomimo ograniczeń, takich jak konieczność konfiguracji u każdego programisty, pre-commit zwiększa jakość kodu i zapobiega wprowadzaniu błędów.

Hook pre-commit uruchamiany jest podczas pracy nad projektem aplikacji przeglądarkoweji uruchamia on np. tzw. lintery, które analizują statycznie kod przed każdym commitem, zapewniając jego spójność, jakość i zgodność z przyjętymi regułami i konwencjami. Aktualnie skonfigurowanych jest łącznie ponad 100 reguł i konwencji sprawdzanych i wymuszanych przed każdą wprowadzoną w formie commita zmianą w repozytorium.

9.6.2. Husky - agnostyczna platformowo nakładka na git hooki

Husky to popularna, platformowo agnostyczna, czyli nieprzywiązana do konkretnego środowiska-/systemu operacyjnego, nakładka na git hooki, która umożliwia łatwe nimi zarządzanie w projektach korzystających z pliku `package.json`. Działa na różnych systemach operacyjnych i integruje się z popularnymi narzędziami do lintowania i testowania. Dzięki temu pozwala na automatyzację procesów walidacji kodu w sposób prosty i wygodny.

9.6.3. ESLint - statyczna analiza kodu

ESLint to narzędzie do statycznej analizy kodu Typescript i JavaScript, które identyfikuje problemy i niezgodności ze stylem kodowania, pomagając utrzymać wysoką jakość kodu. Pozwala na konfigurowanie reguł i wykrywanie błędów jeszcze przed uruchomieniem aplikacji, co przyspiesza proces rozwoju i zmniejsza ilość błędów wykrywanych dopiero w późniejszych etapach testowania oprogramowania. Eslint jest wykorzystywany przy

9.6.4. Stylelint - statyczna analizy kodu

Stylelint to narzędzie do statycznej analizy kodu CSS i innych preprocesorów stylów typu SCSS, które pomaga wykrywać błędy i niezgodności w stylach, zapewniając spójność i jakość warstwy wizualnej aplikacji. Dzięki niemu można automatycznie wymuszać zasady formatowania i poprawności kodu stylów, co ułatwia utrzymanie projektu.

Podsumowując, GitHub Actions i git hooki (w szczególności pre-commit z Husky) tworzą efektywny ekosystem automatyzacji walidacji i testowania aplikacji, łącząc automatyzację na poziomie repozytorium z lokalną kontrolą jakości kodu. Na przyszłość, dla bardziej rozbudowanych pipeline'ów, warto rozważyć Jenkins jako narzędzie oferujące większe możliwości konfiguracyjne i skalowalność.

10. Bezpieczeństwo systemu

Bezpieczeństwo systemu można zdefiniować nie tylko jako technologie i techniczne mechanizmy zapewniania poufności i integralności dostępu do danych oraz funkcjonalności, ale również jako sam zbiór praktyk, których zadaniem jest ochrona infrastruktury IT. Mowa więc o szerokim zestawie czynności, metodologii oraz technicznych zabezpieczeń, które razem gwarantują zamierzony sposób działania systemu oraz jego odporność na niespodziewane incydenty i działania aktorów o złych intencjach.[53]

Bezpieczeństwo systemu jest krytycznym aspektem do uwzględnienia i zaprojektowania, ponieważ potencjalne zaniedbania w tej sprawie mogą skutkować stratami finansowymi, konsekwencjami prawnymi, czy utratą reputacji[54].

10.1. Centralny serwer autoryzacyjny Keycloak

Keycloak, jako otwartoźródłowe oprogramowanie do zarządzania tożsamością i dostępami (IAM - Identity and Access Management) wykorzystywany jest w przypadku Passed Pawn do uwierzytelniania i autoryzowania dostępu użytkowników do głównych zasobów aplikacji.

Najważniejsze funkcjonalności Keycloak:

- **Uwierzytelnianie użytkowników:** w tym logowanie SSO
- **Autoryzacja dostępu:** kontrola dostępu do zasobów systemu
- **Obsługa protokołów:** obsługa np. OAuth2, OpenID Connect, SAML
- **Zarządzanie użytkownikami i rolami:** możliwość definiowania ról i uprawnień
- **Wieloczynnikowe uwierzytelnianie (MFA):** zwiększenie poziomu bezpieczeństwa
- **Personalizacja i rozszerzalność:** dostosowanie do specyficznych potrzeb
- **Możliwa integracja z zewnętrznymi dostawcami tożsamości**: Keycloak oferuje prosty sposób implementacji logowania przy pomocy np. konta Google, Facebook, czy GitHub
- **Integracja z LDAP i Active Directory:** potencjalna synchronizacja danych użytkowników

Zalety Keycloak:

- **Open source i bezpłatność:** brak kosztów licencyjnych
- **Bogata funkcjonalność:** wsparcie dla współczesnych standardów branżowych
- **Łatwa integracja:** kompatybilność z różnymi aplikacjami i protokołami
- **Silne wsparcie:** aktywna społeczność open-source, regularne aktualizacje i szeroka dokumentacja
- **Solidne mechanizmy bezpieczeństwa**: Keycloak oferuje uznane i sprawdzone mechanizmy bezpieczeństwa, takie jak np. authorization code grant flow z PKCE

Wady Keycloak:

- **Zaawansowana konfiguracja i złożoność:** wymaga wiedzy technicznej do optymalnego ustawienia
- **Wymagania zasobów:** może być zasobożerny przy dużej skali użytkowników
- **Dostosowanie do specyficznych potrzeb:** nie wszystkie potencjalne potrzeby mogą być zaadresowane przez Keycloak

[53] Oxari Team. *Bezpieczeństwo systemów IT – czym jest i jak je skutecznie zapewnić?* `https://www.oxari.com/blog/bezpieczenstwo-systemow-it-czym-jest-i-jak-je-skutecznie-zapewnic`. Data dostępu 2025-06-25. 2025.

[54] [Tamże].

Alternatywne rozwiązania do zarządzania tożsamością i dostępami:

- **Okta:** komercyjne rozwiązanie chmurowe z szerokim wsparciem
- **Auth0:** platforma IAM z dużą elastycznością i integracjami
- **Microsoft Azure Active Directory:** popularne w środowiskach Microsoft

Uzasadnienie wykorzystania Keycloak:

Keycloak został wybrany ze względu na swoją ogólną elastyczność, oferowaną kontrolę i szeroką gamę dostarczanych funkcjonalności. Chcąc zachować aspekt własnego hostowania i zarządzania serwerem autoryzacyjnym, trudno wymienić drugie tak bogate w możliwości i szeroko wspierane, darmowe narzędzie jak Keycloak.

10.2. Metodologia zabezpieczania systemu

Projektowanie bezpieczeństwa systemu opiera się na możliwym stosowaniu wielu warstw zabezpieczeń, co jest solidnym podejściem do tematu bezpieczeństwa. Takie podejście zwiększa odporność systemu poprzez minimalizację różnego rodzaju szkód wynikających z potencjalnego wykorzystania różnych wektorów ataku przez złych aktorów albo nawet umożliwienia uzyskania korzyści bądź wpłynięcia w sposób negatywny na działanie systemu w przypadku złamania któregoś zabezpieczenia. Tego typu podejście jest podstawą w solidnych systemach, ponieważ minimalizuje podatność na ataki typu 0-day, które wykorzystują niezaktualizowane luki bezpieczeństwa, które w dłuższej perspektywie czasu są nie do uniknięcia.

Ponadto system projektowany jest z myślą o minimalizacji wektora ataków poprzez ograniczenie powierzchni narażonej na potencjalne zagrożenia. Przykładem może być np. ograniczenie pewnego spektrum funkcjonalności i dostępności do infrastruktury tylko do niezbędnych elementów, np. poprzez ograniczanie ruchu sieciowego lub dostępu do komponentów systemu z poziomu internetu.

Kolejnym ważnym elementem branym pod uwagę jest rozważny dobór i poszerzanie zależności oprogramowania. Unika się niepotrzebnych zależności oraz regularnie aktualizowane i monitorowane są wykorzystywane biblioteki, aby zapobiegać wprowadzaniu nowych luk bezpieczeństwa i poszerzaniu wektora ataków.

Zasada least privileged access (najmniejszych uprawnień) traktowana jest jako wszechobecna mantra przy planowaniu dostępu do zasobów systemu przez użytkowników końcowych, ale tak samo przez zasoby samego systemu w postaci jego komponentów. Jest ona konsekwentnie stosowana i rozważana przy projektowaniu systemu, aby użytkownicy i procesy miały dostęp wyłącznie do zasobów niezbędnych do realizacji swoich zadań. Ogranicza to potencjalne szkody wynikające z kompromitacji konta lub błędów użytkownika i również minimalizuje potencjalną powierzchnię ataków na system.

Oprócz tego ważne jest korzystanie z aktualnych wersji LTS (Long-Term Support) wykorzystywanego oprogramowania, co zapewnia stabilność i regularne aktualizacje bezpieczeństwa. Dzięki temu system jest chroniony przed znanymi podatnościami i ma zapewnione wsparcie producenta.

Świadomość stosowania innych, bardziej szczegółowych dobrych praktyk bezpieczeństwa jest również niezbędna w bardziej zniuansowanych tematach. Wiedzę na ten temat można zdobywać na uznanych forach takich jak OWASP oraz korzystając z narzędzi do walidacji kodu. Przykładem takiego narzędzia jest **Popeye**, którego część odpowiedzialności polega na analizie zasobów klastra Kubernetes pod kątem odpowiedniej konfiguracji RBAC (Role Based Access Control, czyli Dostęp oparty o role)

zapewniającej maksymalne bezpieczeństwo.

10.3. Ogólne schematy i mechanizmy bezpieczeństwa

10.3.1. Same Origin Policy

Same Origin Policy (pol. Polityka Jednakowego Pochodzenia) to mechanizm bezpieczeństwa wbudowany w przeglądarki, który ogranicza dostęp do zasobów znajdujących się na innej domenie niż strona źródłowa. Jej działanie polega na blokowaniu odczytu treści (lub zupełnym blokowaniu wysyłania zapytania do serwera) odpowiedzi od serwera bez jawnego zatwierdzenia przez serwer źródła, które chce odpowiedź odczytać. Ze względu na zachowanie kompatybilności ze starymi stronami, spektrum oraz sposób blokowania zapytań jest nieco zawiły. Co do zasady można natomiast przyjąć, że by zmaksymalizować surowość w egzekwowaniu bezpieczeństwa w ramach Same Origin Policy przez przeglądarkę, należy nie wprowadzać konfiguracji CORS na serwerze. CORS, czyli Cross Origin Resource Sharing, to mechanizm umożliwiający odstępstwa od reguł Same Origin Policy sterowany nagłówkami odpowiedzi serwera na zapytania pochodzące z przeglądarki. Jeżeli nie ma więc potrzeby wprowadzania odstępstw od mechanizmów bezpieczeństwa oferowanych przez przeglądarki, to nie powinno się ich po prostu definiować.

Passed Pawn, w komunikacji przeglądarka <-> serwer, używa serwera reverse proxy w postaci nginx. Wszystkie zapytania są do niego kierowane, co tworzy warstwę zapewniającą pojedyncze z perspektywy przeglądarki źródło pozyskiwania plików statycznych aplikacji typu SPA, a jednocześnie komunikacji z głównym serwerem aplikacji. To rozwiązanie pozwala nie konfigurować CORS i nie zezwalać na komunikację między źródłami, co w efekcie chroni użytkowników aplikacji przed atakami CSRF (Cross Site Request Forgery), których duża część opiera się na wykorzystaniu lepiej lub gorzej skonfigurowanego przez serwer CORS do przeprowadzenia ataku.

10.3.2. HTTP + TLS

TLS (Transport Layer Security) zapewnia 3 fundamentalne dla bezpieczeństwa użytkownika i aplikacji aspekty, czyli **poufność**, **integralność danych** i **uwierzytelnienie**. Dzięki pozyskaniu podpisanych certyfikatów TLS od ośrodka certyfikującego Let's Encrypt, użytkownik wchodzący na stronę Passed Pawn wie, że znajduje się na dobrej stronie, a nie na stronie, która pod Passed Pawn się podszywa. Ponadto komunikacja między przeglądarką i serwerem jest szyfrowana, co stanowi właśnie aspekt zapewniający poufność komunikacji. Oprócz tego, dzięki wykorzystaniu odpowiednich mechanizmów kryptograficznych, będących częścią funkcji zapewnionych przez protokół TLS, przeglądarka i serwer reverse proxy są w stanie zauważyć, kiedy ich komunikacja została zaburzona, a przesyłane dane naruszone. TLS chroni użytkownika i serwer przed spektrum ataków z kategorii MITM (Main in the Middle). By maksymalnie wykorzystać potencjał TLS, serwer reverse proxy przekierowywuje każde zapytanie HTTP, czyli bez wykorzystania protokołu TLS na HTTPS, czyli z wykorzystaniem TLS.

10.4. Bezpieczeństwo aplikacji klienckiej

10.4.1. Zabezpieczenia przed XSS

Cross-Site-Scripting (XSS), to metoda, która polega na umieszczeniu niechcianego kodu w aplikacji webowej przez niepowołaną osobę lub podmiot. Taki kod może posłużyć do wykradnięcia wrażliwych danych użytkownika lub wykonywania w jego imieniu niepożądanych akcji. Angular posiada

wbudowane mechanizmy, które zabezpieczają przed tym rodzajem ataku. Jednym z nich jest automatycznie sanityzowanie wartości wprowadzonych do szablonu. W przypadku zwykłej interpolacji, kod HTML nie jest w ogóle interpretowany, tylko przerabiany tak, aby wyświetlił się na stronie w surowej postaci. Natomiast kod, który powinien być interpretowany, np. przekazany do atrybutu `innerHtml`, `style` czy atrybutów związanych z url-ami, jest automatycznie sanityzowany. Usuwane są wszystkie potencjalnie niebezpieczne fragmenty, np. tagi `<script></script>`. W rzadkich przypadkach, kiedy musimy umieścić w aplikacji niebezpieczny kod, można użyć jednej z kilku dostępnych metod serwisu `DomSanitizer`, która pozwoli nam ominąć zabezpieczenia.[55]

10.4.2. Zabezpieczenia przed kradzieżą filmów

Poprzez zabezpieczenie aplikacji przed kradzieżą filmów nie jest rozumiane zupełne uniemożliwienie kradzieży filmów z platformy, ponieważ jest to praktycznie niemożliwe. Chodzi w istocie o utrudnienie i uczynienie jak najmniej wygodnym kradzieży filmów dla przeciętnego użytkownika. Jest to osiągane poprzez użycie odpowiednich atrybutów kodu HTML na znaczniku video. Mowa o atrybutach usuwających z interfejsu użytkownika przyciski pobrania filmiku. Oprócz tego, w skrypcie komponentu odpowiedzialnego za wyświetlanie filmiku przesłoniona jest reakcja na kliknięcie prawym przyciskiem myszy tak, aby użytkownik nie mógł otworzyć menu, w którym znajduje się opcja pobrania filmu. Równie ważnym aspektem jest pobieranie przez aplikację webową filmu z Cloudinary w postaci załącznika ładowanego bezpośrednio do pamięci przeglądarki, zamiast wyświetlanego za pomocą publicznego URL. Do pobrania filmiku wykorzystywany jest URL generowany specjalnie na potrzeby pobrania filmiku załącznika. URL po krótkim czasie wygasa i nie da się go ponownie wykorzystać. Ponadto jest on wykorzystywany tylko w kodzie skryptu strony, co czyni go znacznie trudniej dostępnym dla przeciętnego użytkownika, niż w przypadku osadzenia bezpośrednio w postaci url elementu <video>. Najbardziej realistycznymi przypadkami kradzieży filmu pozostają więc trzy scenariusze:

- Odnalezienie w zakładce sieci w narzędziach deweloperskich przeglądarki zapytania do Cloudinary o film w postaci załącznika i pobranie pliku binarnego, a następnie przekonwertowanie go na format przyjazny człowiekowi
- Szybkie odnalezienie i wykorzystanie URL pobrania załącznika w mało czytelnym, spakowanym kodzie skryptu strony
- Wykorzystanie własnego skryptu do wydobycia filmiku w postaci pliku binarnego z pamięci przeglądarki, a następnie przekonwertowanie go na format przyjazny człowiekowi
- Nagranie ekranu przeglądarki

W każdym, oprócz ostatniego z tych scenariuszy, użytkownik musi wykazać się nieprzeciętną wiedzą i umiejętnościami, żeby pobrać film umieszczony na stronie. Te scenariusze wymagają więcej pracy i wiedzy, niż ostatni scenariusz, natomiast zabezpieczenie aplikacji przed ostatnim scenariuszem jest jednak najtrudniejsze i wymaga zastosowania narzędzi DRM (Digital Rights Management) i opcjonalnie steganografii (jako mechanizm pozwalający zidentyfikować osoby nagrywające/pobierające filmiki), które są skomplikowane w wykorzystaniu.

10.4.3. Zarządzanie tokenami dostępu przez serwer Keycloak i adapter Keycloak

W aplikacji klienckiej adapter Keycloak przechowuje 3 rodzaje tokenów:

[55] Angular. *Preventing cross-site scripting (XSS).* `https://angular.dev/best-practices/security`. Data dostępu: 2025-06-24. 2025.

- Access Token: token JWT, który jest używany do autoryzacji przy dostępie do zasobów na serwerze. Domyślnie żyje 5 minut.
- Refresh Token: token, który pozwala odświeżyć access token po jego wygaśnięciu bez konieczności ponownego logowania.
- ID Token: token JWT zawierający różne informacje o użytkowniku. Część tych danych jest dostępna również w access tokenie.

Tokeny te są przechowywane w bezpośrednio w pamięci przeglądarki, aby uniknąć zagrożeń związanych z przechowywaniem ich w `localStorage`. Jednak nawet po odświeżeniu strony, kiedy tokeny zostają utracone, użytkownik nie musi logować się ponownie. Jest to możliwe dzięki 2 mechanizmom:

- *Session Cookie* (Cookie sesyjne) - jest ono ustawiane przez Keycloak w momencie zalogowania. Ponieważ jest to cookie HTTP-Only, ustawione przez Keycloak, aplikacja kliencka w ogóle nie ma do niego dostępu, co zapobiega jego wykradnięciu.
- *Silent Check SSO* - to mechanizm, który umożliwia adapterowi Keycloak ciche sprawdzenie sesji. Dzieje się to w specjalnym ukrytym *iframe*, w którym Keycloak wysyła żądanie o tokeny do serwera. Do żądania automatycznie dołączane jest Session Cookie. Na jego podstawie serwer decyduje, czy użytkownik dalej jest zalogowany i w przypadku powodzenia wysyła tokeny.

Logowanie przez Keycloak działa zgodnie z *Authorization Code Flow*, w połączeniu z *PKCE (Proof Key for Code Exchange)*. Przy próbie logowania aplikacja najpierw generuje *code_verifier*, czyli losowy ciąg znaków i *code_challenge*, czyli zahashowaną wersję *code_verfier*. Następnie aplikacja wysyła żądanie logowania, w którym przesyłany jest *code_challenge*. Serwer odpowiada przekierowując użytkownika na stronę logowania Keycloak. Jeżeli wpisane przez użytkownika dane są poprawne, użytkownik przekierowywany jest z powrotem, z otrzymanym od serwera *authorization_code*. Korzystając z kodu autoryzacyjnego, aplikacja natychmiast wysyła do serwera żądanie o tokeny, do którego dołącza również wygenerowany na początku *code_verifier*. Jeżeli kod jest poprawny, a zahashowany *code_verifier* jest taki sam, jak przesłany na początku *code_challenge*, serwer przesyła w odpowiedzi zestaw tokenów. W aplikacji klienckiej cały ten mechanizm implementowany jest przez adapter Keycloak, czyli Keycloak-js.[56]

10.5. Bezpieczeństwo aplikacji serwerowych

10.5.1. SQL injection

Aplikacja serwerowa używa EntityFramework Core jako ORM'a. Zapytania do bazy danych nigdy nie są wysyłane za pomocą ręcznie napisanego SQL, natomiast używane są zapytanie LINQ. Takie zapytania są parametryzowane, co uniemożliwia użytkownikom wstrzykiwanie własnego SQL.

10.5.2. Walidacja danych

Wszystkie dane wysyłane przez klienta są walidowane. Aplikacja używa zarówno walidatorów udostępnianych przez ASP.NET Core, jak i napisanych przez nas. Gwarantuje to, że użytkownik nie będzie w stanie przesłać danych, które mogłyby uszkodzić system, powodując niespodziewane zachowanie.

[56]Keycloak Team. *Securing applications and services with OpenID Connect*. `https://www.keycloak.org/securing-apps/oidc-layers`. Data dostępu 2025-06-26. 2025.

10.5.3. Walidacja RBAC

Wszystkie poprawne tokeny JWT, jakie użytkownicy używają w celu autoryzacji, zawierają informacje o roli użytkownika. Z tego powodu aplikacja serwerowa jest w stanie bardzo prosto i bezpiecznie zweryfikować, czy użytkownik jest uczniem, czy trenerem. Większość zapytań, jakie użytkownik może zadać serwerowi, jest zabezpieczona właśnie taką rolą, co daje pewność, że użytkownik nigdy nie otrzyma zasobów, do których nie ma dostępu.

10.6. Bezpieczeństwo infrastruktury systemu

W tej sekcji przedstawione są metody zabezpieczenia serwera dedykowanego od OVH, serwera automatyzacji Jenkins i jego pipeline-ów, ogólnego zabezpieczania klastra Kubernetes oraz zabezpieczania repozytoriów GitHub i domeny internetowej Passed Pawn

- **Logowanie SSH:**
  - Dostęp do serwerów możliwy wyłącznie za pomocą kluczy SSH.
  - Klucze SSH zabezpieczone dodatkowym hasłem (passphrase) dla zwiększenia bezpieczeństwa.
  - Wyłączenie logowania za pomocą hasła (PasswordAuthentication no) w konfiguracji SSH.
- **Firewall (UFW):**
  - Konfiguracja UFW (Uncomplicated Firewall) do blokowania wszelkiego ruchu przychodzącego.
  - Dopuszczanie ruchu przychodzącego tylko na niezbędnych portach HTTP, HTTPS, SSH i Kubernetes
  - Monitorowanie i logowanie prób nieautoryzowanego dostępu.
- **Fail2Ban:**
  - Automatyczne blokowanie adresów IP po wykryciu wielokrotnych nieudanych prób logowania.
  - Zapobieganie atakom brute-force i skanowaniu portów.
- **Zabezpieczenia organizacji, repozytoriów oraz rejestru OCI w GitHub:**
  - Użycie granularnych *deploy keys* do repozytoriów, ograniczających dostęp tylko do niezbędnych zasobów.
  - Konto serwera CI/CD z ograniczonymi uprawnieniami, wykorzystujące *Personal Access Token* (PAT) o minimalnych wymaganych uprawnieniach.
  - Wdrożenie surowych zasad i protokołów wprowadzania zmian w repozytorium, np. *branch protection rules*:
    - Wymuszanie *code review* przed scaleniem zmian.
    - Ograniczenie możliwości bezpośredniego pushowania do głównych gałęzi.
    - Automatyczne testy i ocena przez Github Copilot przed akceptacją zmian.
- **Wbudowany system RBAC i Network Policies w Kubernetes:**
  - Definiowanie ról i uprawnień (Role-Based Access Control) zgodnie z zasadą najmniejszych uprawnień.
  - Ograniczanie dostępu do zasobów Kubernetes tylko do autoryzowanych użytkowników i usług.
  - Implementacja *Network Policies* ograniczających komunikację między podami i zewnętrznym ruchem sieciowym.
- **Least Privileged Access oraz izolacja środowiska automatyzacji Jenkins:**
  - Konfiguracja kont Jenkins z minimalnymi uprawnieniami potrzebnymi do wykonania zadań.
  - Hermetyzacja środowiska Jenkins poprzez konteneryzację i izolację sieciową za pomocą Namespaceów, Network Policies i RBAC Kubernetes.

    - Ograniczenie dostępu do wrażliwych sekretów i kluczy w Jenkinsie.
- **Budowanie i wypychanie artefaktów:**
    - Wykorzystanie Kaniko do bezpiecznego budowania obrazów kontenerów bez potrzeby uruchamiania dockera w trybie root i bez potrzeby montowania gniazda środowiska uruchomieniowego kontenerów do agentów Jenkins.
    - Automatyczne wypychanie artefaktów do rejestru OCI GitHub z wykorzystaniem bezpiecznych tokenów dostępu.
- **Zabezpieczenie domeny internetowej za pomocą DNSSEC:**
    - Zapewnia integralność i autentyczność odpowiedzi DNS poprzez podpisy cyfrowe.
    - Chroni przed atakami typu DNS spoofing i cache poisoning, zapobiegając fałszowaniu rekordów DNS.
    - Wdrożenie DNSSEC dla zakupionej domeny zwiększa bezpieczeństwo użytkowników i poprawia zaufanie do serwisu.

11. Organizacja pracy nad projektem

Efektywna organizacja pracy zespołu programistycznego jest kluczowym elementem sukcesu każdego projektu informatycznego. W niniejszym rozdziale przedstawiono metody i narzędzia, które zostały wykorzystane podczas rozwoju aplikacji, w tym techniki pracy zespołowej, system zarządzania zadaniami oraz sposoby komunikacji, które umożliwiły sprawne i jakościowe realizowanie celów projektowych.

11.1. Pair programming

Programowanie w parach (pair programming) to technika programowania, w której dwóch programistów wspólnie pracuje nad jednym zadaniem, siedząc przy jednym komputerze. Ostatnio popularność zyskuje zdalna praca w parach, która polega na udostępnieniu swojego ekranu drugiemu deweloperowi przez dedykowane aplikacje.

Zaletą opisywanej techniki pracy zespołowej jest przede wszystkim wytwarzanie lepszego oprogramowania i znajdowanie błędów na wcześniejszym etapie rozwoju projektu. Praca w parach wymusza na deweloperach prowadzenie żywej, nieustannej dyskusji, która prowadzi do znalezienia możliwie najlepszego rozwiązania, a także umożliwia dzielenie się cenną wiedzą, co prowadzi do ciągłego podnoszenia poziomu kompetencji programistycznych członków zespołu. Wyróżniamy 3 główne techniki programowania zespołowego:[57]

- **Ping Pong** — technika pracy w parach powiązana z programowaniem sterowanym testami (Test-Driven Development). Jeden deweloper pisze testy, drugi wytwarza oprogramowanie, które pomyślnie przechodzi te testy. Deweloperzy zamieniają się rolami, co sprawia, że technika jest efektywna i stabilna.
- **Kierowca-Nawigator (Driver-Navigator)** — technika programowania zespołowego w relacji kierowca-nawigator. Jeden deweloper pisze kod (kierowca), stosując się do poleceń, wskazówek drugiej osoby (nawigatora). Deweloper-kierowca ma możliwość kwestionowania i nadzorowania otrzymywanych treści, co w naturalny sposób prowadzi do dyskusji i znalezienia wspólnego rozwiązania.
- **Nieustrukturyzowane programowanie w parach (Unstructured Pairing)** — trzecia technika programowania w parach, występująca w przypadku, gdy praktykowana praca zespołowa nie wpisuje się w żadną z powyższych form. Wyróżnia się luźniejszą strukturą łączącą techniki Ping-Pong i Kierowca-Nawigator. Zamiana ról deweloperów nie jest sztywno narzucona, występuje w dogodnym, sensownym momencie.

Podczas budowy aplikacji szachowej zespół deweloperski pracował głównie w pojedynkę nad postawionymi przed nim wyzwaniami. Programowanie w parach było wykorzystywane głównie przy pracy nad kompleksowymi i priorytetowymi zadaniami, które wymagały dwóch par oczu do wydajnego i jakościowego rozwiązania problemu. Podczas długiej drogi rozwoju portalu szachowego zespół wykorzystywał głównie technikę nieustrukturyzowanego programowania w parach, dzięki luźnej i naturalnej formie udało się skutecznie i przyjemnie budować fundamenty aplikacji. Technika pracy zespołowej przyniosła owocne rezultaty przy rozwoju logiki szachowej i integracji systemu zarządzania dostępem — Keycloak z serwisem szachowym.

Zaletą stosowania opisanej techniki jest jej efektywność. Zespół prowadził wiele intensywnych dyskusji na temat architektury systemu i samych rozwiązań problemów dotyczących aplikacji, co przełożyło się na wytworzenie wydajnego i mniej wadliwego oprogramowania. Proces zaangażowania dodatkowego członka zespołu do pracy nad wspomnianymi kompleksowymi zadaniami był bardzo prosty, jeśli

[57] Codementor. *What is pair programming?* `https://www.codementor.io/pair-programming`. Data dostępu: 2025-06-19. 2025.

rozwiązanie zadania w pojedynkę było trudne lub zajmowało za dużo czasu (co w konsekwencji zagrażało utratą płynności rozwoju aplikacji szachowej), zespół decydował się na pracę w parach.

11.2. Recenzja kodu (Code review)

Recenzja kodu to proces w tworzeniu oprogramowania, którego celem jest sprawdzenie napisanego kodu i wykrycie potencjalnych błędów. Wyróżniamy kilka rodzajów recenzji kodu w tym (Pull Request Review), jest to funkcjonalność wykorzystywana w systemach kontroli wersji opartych o Git (Github, Gitlab). Deweloperzy tworzą (pull request) z zaproponowanymi zmianami w kodzie, które są następnie sprawdzane przez resztę członków zespołu.

Repozytorium aplikacji szachowej korzysta z opisywanej funkcjonalności, co umożliwia kontrolowanie wypychanego kodu i wykrywanie błędów na bardzo wczesnym etapie przed scaleniem zmian do głównej gałęzi. Recenzja kodu wymusza także zapoznanie się z rozwiązaniami zadań realizowanych przez innych członków zespołu, dzięki czemu wiedza o projekcie jest rozproszona, a zależność od jednej osoby zostaje ograniczona.

Kolejną zaletą funkcjonalności jest wymiana wiedzy w zespole i możliwość dyskusji nad proponowanym rozwiązaniem, co prowadzi do podejmowania wspólnych i w konsekwencji lepszych decyzji projektowych. Podczas rozwoju portalu szachowego udało się naprawić wiele błędów w implementacji, które zostały wykryte na etapie recenzji kodu, co świadczy o skuteczny działaniu procesu.[58]

11.3. Kanban

Rozwój aplikacji szachowej wiązał się z dużą liczbą zadań wymagających skutecznego zarządzania, dlatego zespół zdecydował się na wykorzystanie dedykowanego narzędzia — tablicy Kanban w aplikacji Trello. Kanban to framework do zarządzania projektem, który używa zasad Agile do planowania, wizualizacji i organizowania zadań projektowych. Jednym z narzędzi oferowanych przez framework jest tablica Kanban (Kanban Board), która pozwala na wizualizację wszystkich zadań, nad którymi pracują poszczególni członkowie zespołu. Tablica składa się z kolumn, które reprezentują konkretne etapy pracy (np. „Do zrobienia", „Recenzja kodu", „Zakończone"). Dzięki temu możliwe jest przejrzyste przedstawienie aktualnego stanu pracy całego zespołu.

Wykorzystanie frameworka umożliwiło zespołowi spójny i skuteczny rozwój projektu. Każdy napotkany błąd podczas pracy nad implementacją portalu szachowego zostawał natychmiastowo zgłaszany i zapisywany na tablicy, co przyczyniło się do uporządkowania pracy zespołu. Kluczowym usprawnieniem, które zapewniło Trello, jest właśnie porządek, czyli posegregowana i czytelna lista zadań.

Korzystanie z narzędzia do zarządzania projektem to podstawa w świecie dzisiejszego rozwoju oprogramowania. Zarządzanie złożonym projektem z dużą liczbą zadań wymaga systemu, który zapewni przejrzystość i kontrolę nad postępami, bez niego łatwo o chaos i brak koordynacji.[59]

[58] Browserstack. *What is Code Review?* `https://www.browserstack.com/guide/what-is-code-review`. Data dostępu: 2025-06-19. 2025.

[59] Coursera Staff. *What Is a Kanban Board?* `https://tinyurl.com/mryvypab`. Data dostępu: 2025-06-19. 2024.

Tabela 8: Porównanie pracy z tablicą Kanban i bez struktury zadań

Aspekt	**Praca z Kanbanem (np. Trello)**	**Praca bez określonej organizacji zadań**
Widoczność zadań	Jasna, przejrzysta wizualizacja wszystkich zadań	Brak ogólnego poglądu, zadania często są rozproszone
Podział etapów pracy	Zadania przypisane do konkretnych etapów (To Do / In Progress / Done)	Brak wyraźnego podziału — trudno określić, co jest w trakcie
Priorytetyzacja zadań	Możliwość ustawienia priorytetów, etykiet i terminów	Priorytety często są niejasne lub zapomniane
Współpraca zespołowa	Łatwe śledzenie postępów i przydziałów	Trudności z rozdzieleniem odpowiedzialności
Śledzenie postępu	Prosty podgląd, kto i nad czym pracuje	Postęp nie jest jednoznacznie widoczny
Eliminacja zatorów	Szybka identyfikacja przeciążeń w konkretnych etapach	Brak mechanizmu wykrywania problemów w przepływie pracy
Zarządzanie czasem	Pomaga planować i kontrolować czas realizacji	Często prowadzi do opóźnień i chaosu
Motywacja i porządek	Systematyczna praca, motywacja dzięki przesuwaniu kart	Ryzyko zgubienia lub zapomnienia o zadaniach

11.4. Burza mózgów

Przed napisaniem pierwszej linijki kodu trzeba mieć pomysł, zarys projektu, MVP, które będzie implementowane w przyszłości. Każdy projekt zaczyna się od starannie wypracowanego planu działania i przygotowania podstawowych funkcjonalności aplikacji. Potrzeba narzędzia, które pozwoli na udokumentowanie pomysłów zespołu.

Pracę nad projektem portalu szachowego rozpoczęto od zebrania pomysłów członków zespołu i wykorzystano do tego burzę mózgów — technikę pracy zespołowej opartą na kreatywnym myśleniu, której celem jest generowanie jak największej liczby pomysłów na rozwiązanie konkretnego problemu. Idealne rozwiązanie do szybkiego dokumentowania licznych, często chaotycznych wizji i sugestii członków zespołu, które po połączeniu przerodziły się w spójny model aplikacji szachowej. Pomysły bywały różne, niekiedy wręcz absurdalne, jednak intensywna i konstruktywna dyskusja w zespole pozwoliła wyłonić solidny zarys projektu.

Proces generowania dobrych pomysłów rozpoczynał się od indywidualnego zebrania i spisania idei oraz wizji przez każdego z deweloperów w formie krótkich opisów na platformie Miro. Następnie odbywała się zespołowa dyskusja, podczas której omawiano propozycje i wspólnie wyłaniano najlepsze rozwiązania. Proces przebiegał bardzo sprawnie i po ukształtowaniu się spójnej całości rozpoczęto prace deweloperskie.[60]

[60]Miro. *Jak przeprowadzić burzę mózgów*. `https://miro.com/pl/brainstorming/what-is-brainstorming/`. Data dostępu: 2025-06-19. 2025.

11.5. Komunikacja w zespole

Komunikacja, czyli niezbędny element udanego projektu informatycznego. Często niedoceniana i zaniedbywana przez zapracowanych, odizolowanych deweloperów, skupionych tylko i wyłącznie na wyznaczonym zadaniu. Komunikacja to klucz do wydajnej pracy zespołu, pozwala na żywą dyskusję i wspólne rozwiązywanie problemów.

Zespół rozwijający aplikację szachową nie miał problemów z wymianą informacji, luźna atmosfera ułatwiła komunikowanie o napotkanych trudnościach i potrzebie konsultacji w razie wątpliwości. Fundamentem komunikacji zespołu była prostota, niezbędny był jasny cel i klarowność przekazywanej informacji. Skupiono się na jakości wypowiedzi, a nie jej objętości.

Głównym kanałem przekazywania informacji były wiadomości tekstowe. Po wypchnięciu zmian do repozytorium i utworzeniu pull requesta deweloper informował resztę zespołu na wyznaczonym kanale tekstowym o możliwości przeglądu jego kodu. Wiadomość zawierała pomocne linki do repozytorium oraz tablicy Kanban, które ułatwiały szybkie odnalezienie się w gąszczu licznych zadań i zapewniały kontekst programistyczny.

W przypadku pracy nad złożonymi zadaniami wymagającymi konsultacji z resztą zespołu wykorzystywano kanały głosowe, które umożliwiły wydajny przekaz informacji i posłużyły za narzędzie do pracy w parach (pair programming). Wykorzystanie dedykowanych aplikacji (miro, trello) umożliwiło organizację pracy zespołu, która w połączeniu z dobrą komunikacją przerodziły się w szybki i stabilny rozwój projektu.[61]

[61] Ayoub Abidi. *The Role of Communication in Software Development*. `https://ayoub3bidi.medium.com/the-role-of-communication-in-software-development-2dbeb6cbe38c`. Data dostępu: 2025-06-19. 2023.

12. Testowanie i walidacja systemu

12.1. Testy API

API jest testowane głównie na dwa sposoby: manualnie oraz za pomocą automatycznych testów jednostkowych.

12.1.1. Testy manualne

Za każdym razem, gdy wprowadzana jest zmiana w kodzie, powiązane miejsce w aplikacji jest od razu testowane manualnie. Do testowania manualnego są używane dwa narzędzia:

- Postman: Popularne narzędzie do testowania API. Prawie wszystkie endpointy są zapisane i nowe mogą być łatwo dodawane w miarę tworzenia nowych funkcjonalności. Po wprowadzeniu zmiany, przed wdrożeniem na produkcję, są wysyłane zapytania do odpowiednich endpointów w różnorodny sposób. Przykładowo, jest sprawdzane, czy aplikacja zachowuje się odpowiednio w przypadku ucznia, trenera oraz niezalogowanego użytkownika. Testowanie jest wykonywane także na różnych zbiorach danych, zarówno dla przypadków poprawnych, jak i takich, które powinny zostać odrzucone. Choć testy te nie dają pełnej gwarancji poprawności działania, pozwalają na szybkie wychwycenie najbardziej oczywistych błędów.
- DevTools: API jest testowane również w połączeniu z frontendem, bezpośrednio w przeglądarce. Aplikacja jest używana w sposób, w jaki robiłby to użytkownik, a w zakładce „Network" w DevTools jest monitorowany ruch sieciowy. Weryfikowane jest, czy wszystkie zapytania do API mają odpowiednią strukturę, oraz czy odpowiedzi są zgodne z oczekiwaniami. Testy te zapewniają wysokie prawdopodobieństwo, że zdecydowana większość użytkowników będzie w stanie korzystać z aplikacji bez błędów.

12.1.2. Testy automatyczne

Testy automatyczne mają pewną wadę — przy zmianach w kodzie konieczna jest często modyfikacja także powiązanych testów, co bywa czasochłonne. Ponieważ testy manualne zapewniają już wysoki poziom wiarygodności, uznano, że dodatkowa pewność oferowana przez testy automatyczne nie uzasadnia nakładu pracy potrzebnego do ich utrzymania. Dlatego automatycznym testowaniem objęte zostały tylko te części aplikacji, które zawierają złożoną logikę lub są najczęściej używane przez użytkowników.

Do testów automatycznych jest używany framework xUnit — popularne narzędzie w ekosystemie .NET, przeznaczone do testowania aplikacji każdego rodzaju, w tym API. Dodatkowo wykorzystywana jest biblioteka Moq, umożliwiająca łatwe tworzenie moków serwisów.

Wszystkie serwisy wewnątrz API są projektowane jako interfejsy, które są wstrzykiwane do komponentów, które ich potrzebują. Jedną z zalet takiego podejścia jest możliwość łatwego ich mokowania. Dzięki temu kontrolery mogą być testowane niezależnie od logiki zawartej w serwisach.

Projekt jest podzielony na trzy główne części: Controllers, Services i Validators:

- Controllers: Testowane są kontrolery, a w szczególności endpointy najczęściej wykorzystywane przez użytkowników. Testy te zakładają poprawność działania serwisów i sprawdzają jedynie, czy kontrolery reagują właściwie. Zapewniają one wysoką wiarygodność, lecz ze względu na częste modyfikacje w kodzie kontrolerów, kod testów również musi być często aktualizowany. Każdy test przebiega według schematu:
    - Arrange: Przygotowywane są odpowiednie moki, które mają odpowiadać w ustalony sposób.

    - Act: Wywoływana jest metoda, która ma zostać przetestowana.
    - Assert: Sprawdzany jest rezultat działania metody: najważniejsze są status odpowiedzi oraz zwrócony obiekt, które muszą odpowiadać założonym przypadkom testowym.
- Services: Testowane są wyłącznie te serwisy, które zawierają skomplikowaną logikę. Jest to logika rzadko modyfikowana, lecz trudna do ręcznego testowania, ze względu na mnogość przypadków, które muszą być uwzględnione. Testy te umożliwiły podejście Test Driven Development, w którym najpierw były pisane testy, a dopiero później implementacja. Dzięki temu każda iteracja pokazywała, jakie przypadki wymagają jeszcze obsłużenia.
    - ClaimsPrincipalServiceTests: Testowana jest logika odnajdywania użytkownika w bazie danych na podstawie tokena JWT.
    - CourseServiceTests: Testowany jest serwis odpowiedzialny za tworzenie i modyfikację kursów. Najistotniejsze jest testowanie dodawania i przesuwania lekcji, ponieważ każda lekcja posiada pole Order. Wymagana jest gwarancja, że nie występują duplikaty wartości Order, a dodanie lekcji pomiędzy innymi powoduje odpowiednie przesunięcie pozostałych.
- Validators: Walidatory są odpowiedzialne za weryfikację poprawności danych wprowadzanych przez użytkownika. Zwykle są one proste, z wyjątkiem jednego walidatora, który sprawdza, czy podany FEN (ciąg znaków reprezentujący stan szachownicy) ma poprawny format. W tym przypadku używany jest zaawansowany wyrażeniem regularnym (regex), które wymaga dokładnego przetestowania. Są testowane liczne przypadki zarówno poprawnych, jak i błędnych FEN-ów.

12.1.3. Github Actions

Przy każdym wprowadzeniu zmian do GitHuba uruchamiany jest tzw. GitHub Action. GitHub Actions mają wiele zastosowań, jednak w repozytorium projektu używane są do automatycznego i szybkiego przeprowadzenia testów.

Cały proces jest wykonywany automatycznie za każdym razem, kiedy nowy kod zostanie wysłany do repozytorium. Dzięki temu możliwe jest szybkie sprawdzenie, czy aplikacja działa poprawnie po wprowadzonych zmianach.

W trakcie działania GitHub Action:

- Tworzone jest wirtualne środowisko z odpowiednimi zależnościami,
- Aplikacja jest budowana,
- Uruchamiane są wszystkie testy automatyczne.

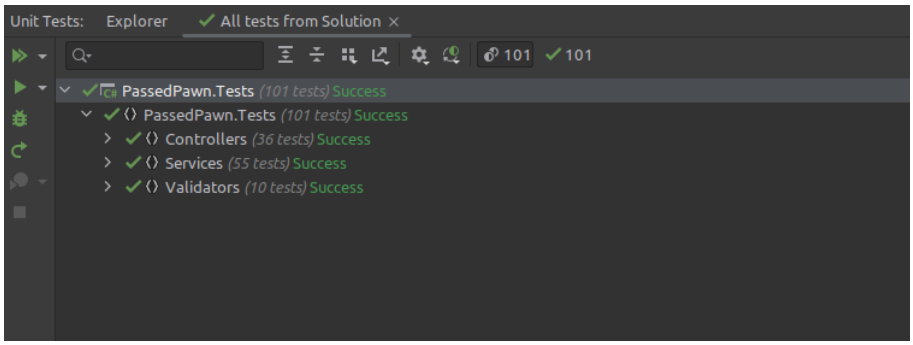
Na podstawie wyniku tych kroków, system automatycznie informuje o sukcesie lub błędzie. Podczas przeglądania kodu przez inną osobę, widoczny jest status ostatniego przebiegu akcji — dzięki czemu bardzo szybko można sprawdzić, czy wszystkie testy przeszły pomyślnie.

W przypadku błędów (np. nieprzechodzących testów), osoba odpowiedzialna za zmiany jest informowana e-mailem przez GitHuba o tym, że Action zakończył się niepowodzeniem. Dzięki temu problem może być szybko zauważony i naprawiony, zanim zmiany zostaną połączone z główną gałęzią projektu.

12.1.4. Rider

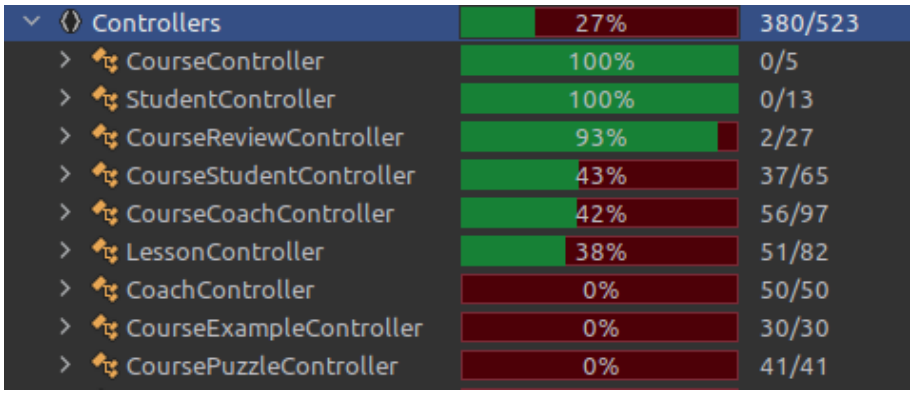
Rider to IDE wyprodukowane i wspierane przez JetBrains. Każdy programista odpowiedzialny za API projektu używa właśnie tego IDE, gdyż zapewnia wiele różnych narzędzi, oraz wspomagaczy w programowaniu. Wiele z tych narzędzi wspomaga testowanie.

- Zakładka Unit Tests: IDE automatycznie znajduje wszystkie testy w projekcie i wyświetla je w dedykowanej zakładce. W tym miejscu programista może uruchomić je wszystkie, lub tylko konkretne, jednym przyciskiem, a następnie zobaczyć, które przechodzą, a które nie.



Rysunek 22: Graficzne przedstawienie statusu testów

- DotCover: Narzędzie, które po uruchomieniu testów oblicza, które części aplikacji są przetestowane, a które nie. Ułatwia analizę przez programistę, jakie części aplikacji potrzebują więcej testów.



Rysunek 23: Graficzne przedstawienie pokrycia kodu przez testy

12.2. Testy end-to-end

Testowanie end-to-end to technika testowania, mająca na celu sprawdzenie całego przepływu aplikacji — od początku do końca. Tego typu testy weryfikują każdy komponent systemu i ich współpracę. W aplikacji Passed Pawn zostały przeprowadzone manualne testy end-to-end. Sporządzone zostały scenariusze, które opisują szczegółowo, jak należy przeprowadzić każdy test. Poniżej przedstawiono przykładowy scenariusz, dotyczący zakupu kursu przez ucznia.

Tabela 9: Przykładowy scenariusz testu end-to-end

Nazwa scenariusza	Zakup kursu przez użytkownika
Aktorzy	Użytkownik zalogowany jako uczeń
Wymagania wstępne	• Użytkownik posiada konto ucznia w systemie • Użytkownik jest zalogowany na konto • Użytkownik posiada kartę płatniczą z dostępnymi środkami
Stan początkowy	Zalogowany użytkownik znajduje się na stronie „My Courses”
Przebieg zdarzeń	1. Użytkownik klika w przycisk „Avalaible Courses” na pasku nawigacji 2. Użytkownik wyszukuje interesujący go kurs za pomocą wyszukiwarki 3. System wyświetla pasujące kursy 4. Użytkownik wybiera klika przycisk „Buy” na kafelku jednego z kursów 5. System wyświetla formularz na dane karty płatniczej 6. Użytkownik uzupełnia dane karty 7. Użytkownik klika przycisk „Pay” 8. System wyświetla powiadomienie o rozpoczęciu płatności 9. System wyświetla powiadomienie o powodzeniu płatności
sukcesu	• Użytkownik uzyskuje dostęp do kursu w zakładce „My Courses” oraz do wszystkich zawartych w nim lekcji i elementów • Zostało stworzone konto ucznia z danymi użytkownika

13. Podsumowanie projektu i potencjał do dalszego rozwoju

Przedstawiony system spełnia podstawowe założone wymagania funkcjonalne i niefunkcjonalne. W swojej obecnej formie umożliwia tworzenie kursów szachowych zawierających dedykowane dla gry w szachy elementy wspomagające proces nauki — quizy, przykłady i łamigłówki, które zawierają interaktywne szachownice z wizualnymi pomocami. Oferuje przy tym usprawnienia w stosunku do rozwiązań konkurencyjnych w postaci intuicyjnego i prostego interfejsu użytkownika, przejrzystego wglądu w zawartość kursów przez uczniów, asystenta AI oraz otwartego dla zainteresowanych tutorów dostępu do publikowania treści na platformie.

Dzięki zastosowaniu praktyk DevOps, IaC, GitOps i CICD, praca nad utrzymaniem, modyfikacją i rozwojem systemu jest wysoce audytowalna i stabilna, a przy tym wciąż efektywna. Stanowi to świadectwo działania zastosowanych praktyk i zaprojektowanej w przemyślany sposób architektury systemu. Mimo znacznego rozrostu, system pozostaje sprawny i przystępny w zarządzaniu. W obecnej formie aplikacja stanowi solidny produkt, który można rozbudować o dodatkowe funkcjonalności, takie jak:

- Śledzenie postępu nauki — elementy w kursie, które były już odwiedzone przez użytkownika, będą odpowiednio oznaczone. Dodatkowo na kafelku kursu znajdzie się informacja, jaki procent elementów został już odwiedzony
- Punkty doświadczenia, które uczeń będzie mógł wymienić na kolejne kursy — angażują użytkownika do dalszych interakcji z platformą i oferują system nagród, które motywują do osiągania kolejnych celów
- Integracja z zewnętrznymi API, np. platformy Chess.com — przy przykładzie z konkretną pozycją użytkownik będzie mógł zobaczyć wszystkie swoje gry, w których ta pozycja wystąpiła i przeanalizować swoje błędy.
- Popularny kontent (hot content) - panel wyróżniający popularne elementy kursów, które cieszą się pozytywnym odbiorem i dużą popularnością wśród użytkowników (pomógłby wyróżnić kontent warty uwagi).
- Tłumaczenia kursów z wykorzystaniem sztucznej inteligencji — pozwoli to na dotarcie do większej liczby uczniów, na przykład takich, którzy nie znają dobrze języka angielskiego. Umożliwi również tworzenie popularnych kursów trenerom z wielu państw, którzy wolą tworzyć je w ojczystym języku. Sztuczna inteligencja wykorzystywana do tego celu rozwija się w bardzo szybkim tempie i już teraz umożliwia wysokiej jakości tłumaczenia.
- Indywidualne lekcje z trenerem — funkcjonalność przewidująca lekcje z tutorem szachowym w formie video-rozmowy z dostępem do interaktywnej szachownicy i materiałów szkoleniowych. Uczeń miałby również możliwość gry w szachy z trenerem na wirtualnej szachownicy, wzbogaconej o wsparcie wizualne w postaci strzałek i podświetlanych pól.
- Generowanie egzaminu końcowego — funkcjonalność, która umożliwiłaby automatyczne stworzenie egzaminu dla ucznia na koniec przepracowanego kursu. Pomogłoby sprawdzić i utrwalić zdobytą wiedzę.
- Ranking użytkowników z największą liczbą punkty doświadczenia — mógłby zachęcać do regularnej nauki i częstszego korzystania z aplikacji.
- Dodawanie innych użytkowników do grona znajomych — umożliwi to wyświetlanie postępów użytkownika na tle osób z otoczenia. Dodatkowo może to skłonić ucznia do promocji platformy wśród swoich bliskich.

Bibliografia

[1] Chess.com. *How Many Chess Players Are There In The World?* https://www.chess.com/article/view/how-many-chess-players-are-there-in-the-world. Data dostępu: 2025-06-23. 2017 (cyt. na s. 7).

[2] Chessly. *Chessly main page*. https://chessly.com/. Data dostępu: 2025-06-23. 2025 (cyt. na s. 8).

[3] Canva. *How to Use Canva: A Beginner's Guide*. https://www.canva.com/learn/how-to-canva-beginners-guide/. Data dostępu: 2025-06-19. 2025 (cyt. na s. 18).

[4] Figma. *Przeglądaj właściwości komponentów*. https://www.canva.com/learn/how-to-canva-beginners-guide/. Data dostępu: 2025-06-19. 2025 (cyt. na s. 18).

[5] Hitesh Sahni. *Canva vs Figma comparison: which design tool is right for you?* https://hypegig.com/canva-vs-figma-comparison/. Data dostępu: 2025-06-19. 2025 (cyt. na s. 19).

[6] prof. WSEI dr hab. inż. Jan Werewka. *Rola i znaczenie architektury w obszarze IT*. https://magazyn.wsei.edu.pl/rola-i-znaczenie-architektury-w-obszarze-it/. Data dostępu: 2025-06-21. 2024 (cyt. na s. 28).

[7] Marcin Sobieraj. *Service Oriented Architecture - SOA*. https://drogaarchitektait.pl/2020/08/service-oriented-architecture-soa/. Data dostępu: 2025-06-21. 2020 (cyt. na s. 30).

[8] Pramitha Jayasooriya. *Monolithic vs. SOA vs. Microservices Architecture: A Java Perspective*. https://medium.com/@lpramithamj/monolithic-vs-soa-vs-microservices-architecture-a-java-perspective-6d3d9fb26ac7. Data dostępu: 2025-06-21. 2024 (cyt. na s. 33).

[9] Sehban Alam. *Single Page Applications (SPAs) vs. Multi-Page Applications (MPAs): Advantages and Challenges*. https://medium.com/@sehban.alam/single-page-applications-spas-vs-multi-page-applications-mpas-advantages-and-challenges-df06bee3fed1/. Accessed: 2025-06-17. 2024 (cyt. na s. 39).

[10] Edgemesh. *Why Time to Interactive (TTI) is the Most Consistent and Valuable Metric*. https://edgemesh.com/blog/time-to-interactive-and-conversion-rate#heading-1. Data dostępu: 2025-06-17. 2025 (cyt. na s. 40).

[11] BrowserStack. *Angular vs React vs Vue: Core Differences*. https://www.browserstack.com/guide/angular-vs-react-vs-vue. Data dostępu: 2025-06-18. 2025 (cyt. na s. 43).

[12] Stackoverflow. *Tag Trends*. https://trends.stackoverflow.co/tags=reactjs%2Cvue.js%2Cangular%2Csvelte%2Cangularjs%2Cvuejs3. Data dostępu: 2025-06-19. 2025 (cyt. na s. 43).

[13] Gara Mohamed. *Angular vs. React (Part 1): style abstraction and encapsulation*. https://medium.com/@gara.mohamed/angular-vs-react-part-1-style-abstraction-and-encapsulation-f55b336e0eaa. Data dostępu: 2025-06-19. 2018 (cyt. na s. 44).

[14] Sotiris Kourouklis. *SASS, CSS, or Tailwind: Which One Should You Choose?* https://dev.to/sotergreco/sass-css-or-tailwind-which-one-should-you-choose-3c0o. Data dostępu: 2025-06-19. 2025 (cyt. na s. 45).

[15] PrimeNG. *PrimeNG Documentation*. https://primeng.org/. Data dostępu: 2025-06-17. 2025 (cyt. na s. 45).

[16] Miłosz Rutkowski. *Wszystko co musisz wiedzieć o Dependency Injection w Angularze*. https://angular.love/pl/wszystko-co-musisz-wiedziec-o-dependency-injection-w-angularze. Data dostępu: 2025-06-18. 2024 (cyt. na s. 46).

[17] Angular. *Forms in Angular*. https://angular.dev/guide/forms. Data dostępu: 2025-06-18. 2025 (cyt. na s. 46).
[18] Angular Minds. *Data Binding In Angular: Everything You Need to Know*. https://www.angularminds.com/blog/data-binding-in-angular. Data dostępu: 2025-06-18. 2025 (cyt. na s. 47).
[19] Angular. *The RxJS library*. https://v17.angular.io/guide/rx-library. Data dostępu: 2025-06-17. 2023 (cyt. na s. 48).
[20] RxJS. *RxJS Overview*. https://rxjs.dev/guide/overview. Data dostępu: 2025-06-17. 2025 (cyt. na s. 48).
[21] Angular University. *Angular Change Detection - How Does It Really Work?* https://blog.angular-university.io/how-does-angular-2-change-detection-really-work/. Data dostępu: 2025-06-19. 2025 (cyt. na s. 49).
[22] Angular. *Understanding communicating with backend services using HTTP*. https://angular.dev/guide/http. Data dostępu: 2025-06-19. 2025 (cyt. na s. 50).
[23] Keycloak JS. *Keycloak JS npm package page*. https://www.npmjs.com/package/keycloak-js. Data dostępu: 2025-06-17. 2025 (cyt. na s. 50).
[24] Mauricio Vigolo. *Keycloak Angular npm package page*. https://www.npmjs.com/package/keycloak-angular. Data dostępu: 2025-06-17. 2025 (cyt. na s. 50).
[25] Angular. *CanActivateFn Documentation*. https://angular.dev/api/router/CanActivateFn. Data dostępu: 2025-06-19. 2025 (cyt. na s. 51).
[26] Raghuvardhan Karanam. *Route Guards in Angular*. https://raghuvardhankaranam.medium.com/route-guards-in-angular-c2c01fe6167b. Data dostępu: 2025-06-19. 2023 (cyt. na s. 51).
[27] Roberts Tech. *Code a Chess Game with Stockfish API – JavaScript Tutorial*. https://youtu.be/fJIsqZmQVZQ?si=6JttY01vB8CtVCzJ. Data dostępu: 2025-06-30. 2024 (cyt. na s. 53).
[28] Roberts Tech. *Code a Chess Game with Stockfish API – JavaScript Tutorial*. https://github.com/awsomeCStutorials/chess-game. Data dostępu: 2025-06-20. 2024 (cyt. na s. 53).
[29] davehorner. *license?* https://github.com/awsomeCStutorials/chess-game/issues/4. Data dostępu: 2025-06-20. 2025 (cyt. na s. 53).
[30] LangChain. *Build a Retrieval Augmented Generation (RAG) App: Part 1*. https://python.langchain.com/docs/tutorials/rag/. Data dostępu: 2025-06-24. 2024 (cyt. na s. 60).
[31] MistralAI. *Mistral Large, our new flagship model*. https://mistral.ai/news/mistral-large. Data dostępu: 2025-06-17. 2025 (cyt. na s. 62).
[32] MistralAI. *Models Overview*. https://docs.mistral.ai/getting-started/models/models_overview/. Data dostępu: 2025-06-17. 2025 (cyt. na s. 62).
[33] IBM. *What Is It Infrastructure?* https://www.ibm.com/think/topics/infrastructure. Data dostępu: 2025-06-23. 2021 (cyt. na s. 63).
[34] OVH. *Dedicated Server Service Level Agreement*. https://us.ovhcloud.com/legal/sla/dedicated-servers/. Data dostępu: 2025-06-23. 2025 (cyt. na s. 64).
[35] Dokumentacja techniczna Kubernetes. *Kubernetes*. https://kubernetes.io/. Data dostępu: 2025-06-23. 2025 (cyt. na s. 66).
[36] Dokumentacja techniczna Kubernetes. *Concepts/Overview*. https://kubernetes.io/docs/concepts/overview/. Data dostępu: 2025-06-23. 2024 (cyt. na s. 66).
[37] Dhaval Gajjar. *Why And Where To Use Kubernetes*. https://www.opensourceforu.com/2024/02/why-and-where-to-use-kubernetes/. Data dostępu: 2025-06-23. 2024 (cyt. na s. 66).

[38] Dokumentacja techniczna K3s. *K3s - Lighweight Kubernetes*. https://docs.k3s.io/. Data dostępu: 2025-06-23. 2025 (cyt. na s. 68).
[39] Dokumentacja techniczna K3s. *Requirements*. https://docs.k3s.io/installation/requirements. Data dostępu: 2025-06-23. 2025 (cyt. na s. 68).
[40] Dokumentacja techniczna Longhorn. *Features*. https://longhorn.io/docs/1.9.0/. Data dostępu: 2025-06-23. 2025 (cyt. na s. 69).
[41] Dokumentacja techniczna Kubernetes. *Ingress*. https://kubernetes.io/docs/concepts/services-networking/ingress/. Data dostępu: 2025-06-23. 2025 (cyt. na s. 69).
[42] Repozytorium ingress-nginx od Kubernetes. *ingress-nginx*. https://github.com/kubernetes/ingress-nginx. Data dostępu: 2025-06-23. 2025 (cyt. na s. 69).
[43] Dokumentacja techniczna cert-manager. *cert-manager*. https://cert-manager.io/docs/. Data dostępu: 2025-06-23. 2025 (cyt. na s. 69).
[44] Grzegorz Gnych. *Czym jest Infrastruktura jako kod (Infrastructure as Code)? – Kompendium wiedzy*. https://nflo.pl/baza-wiedzy/czym-jest-infrastruktura-jako-kod-infrastructure-as-code-kompendium-wiedzy/#Integracja-z-Tenable-i-Rapid7. Data dostępu: 2025-06-25 (cyt. na s. 71).
[45] Andrzej Ochman. *GitOps cz. I. Wprowadzenie do GitOps i jego rola w konteneryzacji*. https://linuxpolska.com/pl/baza-wiedzy/blog/gitops-wprowadzenie-jego-rola-w-konteneryzacji-cz-i/. Data dostępu: 2025-06-25. 2025 (cyt. na s. 73).
[46] Gene Kim i in. *The DevOps Handbook*. 2st. Data dostępu: 2025-06-25. IT Revolution Press, 2016 (cyt. na s. 76).
[47] Microsoft. *Continuous Integration and Continuous Delivery*. https://microsoft.github.io/code-with-engineering-playbook/CI-CD/. Data dostępu 2025-06-25. 2024 (cyt. na s. 76, 77).
[48] Oxari Team. *Bezpieczeństwo systemów IT – czym jest i jak je skutecznie zapewnić?* https://www.oxari.com/blog/bezpieczenstwo-systemow-it-czym-jest-i-jak-je-skutecznie-zapewnic. Data dostępu 2025-06-25. 2025 (cyt. na s. 85).
[49] Angular. *Preventing cross-site scripting (XSS)*. https://angular.dev/best-practices/security. Data dostępu: 2025-06-24. 2025 (cyt. na s. 88).
[50] Keycloak Team. *Securing applications and services with OpenID Connect*. https://www.keycloak.org/securing-apps/oidc-layers. Data dostępu 2025-06-26. 2025 (cyt. na s. 89).
[51] Codementor. *What is pair programming?* https://www.codementor.io/pair-programming. Data dostępu: 2025-06-19. 2025 (cyt. na s. 92).
[52] Browserstack. *What is Code Review?* https://www.browserstack.com/guide/what-is-code-review. Data dostępu: 2025-06-19. 2025 (cyt. na s. 93).
[53] Coursera Staff. *What Is a Kanban Board?* https://tinyurl.com/mryvypab. Data dostępu: 2025-06-19. 2024 (cyt. na s. 93).
[54] Miro. *Jak przeprowadzić burzę mózgów*. https://miro.com/pl/brainstorming/what-is-brainstorming/. Data dostępu: 2025-06-19. 2025 (cyt. na s. 94).
[55] Ayoub Abidi. *The Role of Communication in Software Development*. https://ayoub3bidi.medium.com/the-role-of-communication-in-software-development-2dbeb6cbe38c. Data dostępu: 2025-06-19. 2023 (cyt. na s. 95).

Spis rysunków

1 Diagram przypadków użycia trenera 13

2 Diagram przypadków użycia ucznia . . . 14
3 Diagram przypadków użycia części wspólnych trenera i ucznia . . . 15
4 Strona startowa (landing page) aplikacji szachowej . . . 20
5 Widok oferty szkolenia i proponowany model nauki . . . 21
6 Strona startowa, sekcja pytań . . . 21
7 Widok komponentu do tworzenia łamigłówek . . . 22
8 Widok komponentu tworzenia quizu, 3 etap . . . 23
9 Komponent do wystawiania recenzji kursu . . . 24
10 Komponent prezentujący średni ocenę kursu . . . 24
11 Komponent pola wejściowego dla poziomu trudności kursu . . . 25
12 Komponent, diagram przedstawiający zawartość kursu szachowego . . . 25
13 Panel obsługi oraz system interaktywnych strzałek na szachownicy . . . 25
14 Asystent AI . . . 26
15 Diagram UML komponentów ogólnej architektury . . . 35
16 Diagram ERD bazy danych aplikacji . . . 37
17 Schemat modułów aplikacji klienckiej . . . 52
18 Diagram działania RAG, udostępniony przez LangChain . . . 60
19 Diagram sekwencji modułu AI. Kolor żółty oznacza serwisy spoza systemu. . . . 62
20 Diagram architektury pipline-ów CI . . . 79
21 Schemat architektury budowania i wypychania obrazu aplikacji w formie Helm Chartu 81
22 Graficzne przedstawienie statusu testów . . . 98
23 Graficzne przedstawienie pokrycia kodu przez testy . . . 98

Spis tabel

1 Porównanie narzędzi: Canva i Figma . . . 19
2 Porównanie: samodzielny projekt UI vs gotowa biblioteka UI . . . 27
3 Porównanie architektur . . . 33
4 Porównanie frameworków: Angular, React i Vue . . . 42
5 Porównanie stylowania komponentów: Angular i React . . . 43
6 Porównanie systemów stylowania na frontendzie . . . 44
7 Przykładowe żądania REST . . . 56
8 Porównanie pracy z tablicą Kanban i bez struktury zadań . . . 94
9 Przykładowy scenariusz testu end-to-end . . . 99