Spis treści

[WSTĘP 1](#_Toc198390154)

[1. ANALIZA WYMAGAŃ 2](#_Toc198390155)

[1.1. Wymagania biznesowe 2](#_Toc198390156)

[1.2. Diagram modeli domenowych 4](#_Toc198390157)

[1.3. Diagram przypadków użycia 6](#_Toc198390158)

[1.4. Diagram klas 8](#_Toc198390159)

[2. ARCHITEKTURA SYSTEMU 10](#_Toc198390160)

[2.1. Zarys architektury 10](#_Toc198390161)

[2.2. Bezpieczeństwo systemu 12](#_Toc198390162)

[2.2.1. Uwierzytelnienie przy użyciu Microsoft Entra ID 12](#_Toc198390163)

[2.2.2. Szyfrowane protokoły komunikacyjne przy użyciu TLS 15](#_Toc198390164)

[2.3. Frontend 17](#_Toc198390165)

[2.3.1. HTML 17](#_Toc198390166)

[2.3.2. CSS 18](#_Toc198390167)

[2.3.3. TypeScript 19](#_Toc198390168)

[2.3.4. Angular 19](#_Toc198390169)

[2.4. Backend 21](#_Toc198390170)

[2.5. Baza danych 21](#_Toc198390171)

# WSTĘP

# ANALIZA WYMAGAŃ

## Wymagania biznesowe

Budowę każdego systemu komputerowego, należy rozpocząć od ustalenia problemu, który ma on rozwiązać. Następnie należy dostrzec możliwe rozwiązania i wybrać jedno z nich. Powszechnie przyjętym sposobem na opis dlaczego organizacja implementuje system są tzw. wymagania biznesowe skupiające się na celach biznesowych[[1]](#footnote-1). Poniżej przedstawiona jest analiza wymagań biznesowych klienta.

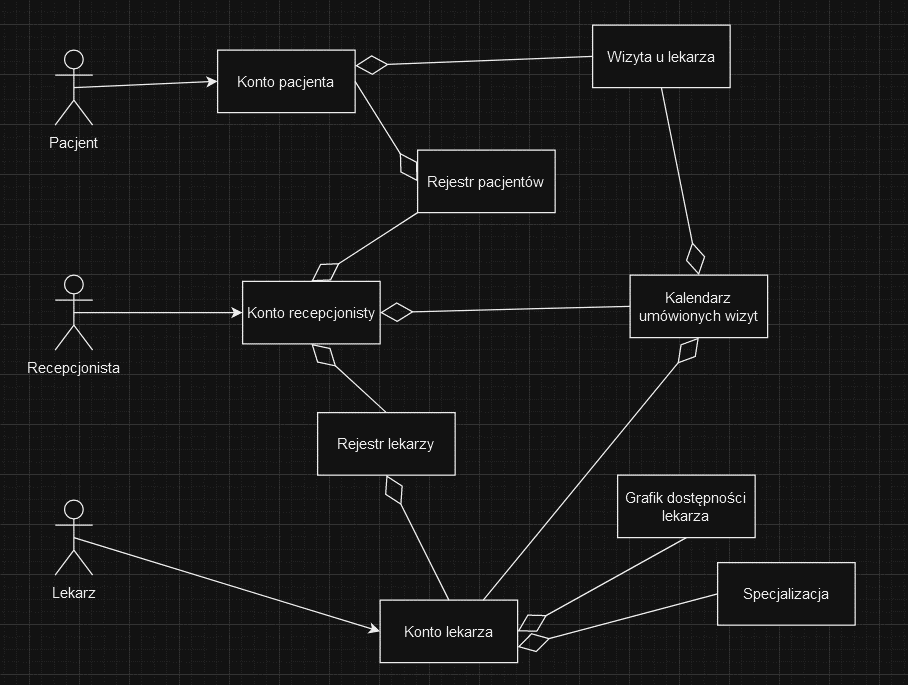
Przychodnia o nazwie *Great Health* potrzebuje systemu, który odciąży jej pracowników w rejestracji pacjentów, tym samym zwiększając wydajność, zmniejszając zapotrzebowanie na rekrutację nowej kadry oraz poprawiając konkurencyjność. Obecnie każdy pacjent musi się najpierw zarejestrować w przychodni poprzez stawienie się fizycznie w przychodni oraz przedstawienie swojego dowodu osobistego. Wówczas recepcjonista prowadząc rejestr pacjentów w formie papierowej, wpisuje nowo zarejestrowaną osobę na liście. Ponadto, pracownicy muszą prowadzić rejestr lekarzy. Podobnie jak z pacjentami, każdy lekarz musi być wpisany w papierowym rejestrze, a w przypadku zwolnienia, z niego wypisany. Oprócz wyżej wspomnianych rejestrów, prowadzony jest grafik dostępności lekarzy, ponieważ większość z nich pełni swoje obowiązki również w innych placówkach. Kiedy recepcjonista placówki ma już przed sobą rejestr pacjentów, lekarzy oraz grafik dostępności specjalistów, może on przyjmować osobiście stawionych pacjentów proszących o umówienie na wizytę z konkretnym lekarzem, o konkretnej godzinie. Informacje te są przechowywane w formie kalendarza. W przypadku jakichkolwiek zmian terminów, pracownik na recepcji musi wykreślić daną osobę z kalendarza i znaleźć jej nowy termin. Cały ten proces, jest żmudny i łatwo o popełnienie w nim błędu, co może skutkować pewnym zamieszaniem oraz opóźnieniami. Zaś z perspektywy samych lekarzy, również nie jest to najlepszy system zarządzania, ponieważ są oni informowani o każdej zmianie z pewnym opóźnieniem. W idealnej sytuacji, lekarze wiedzą kiedy mają pacjenta, a kiedy nie, najszybciej jak tylko się da, oraz nie muszą być o tym informowani przez pracowników z recepcji. Recepcjoniści powinni tylko i wyłącznie weryfikować tożsamość oraz obecność pacjentów gotowych na wizytę, a następnie przekierowywać ich do konkretnego gabinetu. Zaś z perspektywy pacjentów, najlepiej byłoby nie wychodzić z domu, aby umówić się do lekarza oraz mieć jasny obraz wolnych terminów na wybrany dzień.

## Diagram modeli domenowych

Na podstawie wyżej opisanych wymagań biznesowych przychodni, należy przełożyć je na wymagania użytkownika[[2]](#footnote-2). Najlepiej zrobić to w postaci diagramów przypadków użycia. Jednakże najpierw pomocnym jest zacząć od zdefiniowania modeli domenowych, które tworzyć będą system[[3]](#footnote-3). Tworząc taki diagram, warto trzymać się pewnych zasad. Są to[[4]](#footnote-4):

* Koncentracja na obiektach występujących w domenie biznesowej
* Używanie agregacji w celu odzwierciedlenia relacji między obiektami
* Zorganizowanie obiektów wokół kluczowych abstrakcji w domenie biznesowej
* Wykorzystywanie modeli domenowych w celu utworzenia słownika projektu
* Utworzenie diagramu modeli domenowych przed diagramem klas tak, aby w tym drugim uniknąć wieloznaczności używanych terminów

Na podstawie powyższych rekomendacji utworzony został diagram modeli domenowych, który znaleźć można na rysunku 1.1. Wyszczególnionych zostało trzech aktorów – pacjent, recepcjonista oraz lekarz. Dla każdego z nich, utworzony został obiekt będącym kontem w systemie. Konto pacjenta zawiera listę wizyt u lekarzy oraz jest częścią rejestru pacjentów. Natomiast konto recepcjonisty ma dostęp do rejestru pacjentów oraz rejestru lekarzy. Pracownik na recepcji będzie również zarządzać kalendarzem umówionych wizyt, dlatego też jego konto powiązane jest z kalendarzem umówionych wizyt. Ostatnim kontem, jest konto lekarza, do którego przypisany jest grafik dostępności, zaś samo konto jest częścią rejestru lekarzy oraz kalendarza umówionych wizyt.



Rys. 1.1 Diagram modeli domenowych (opracowanie własne)

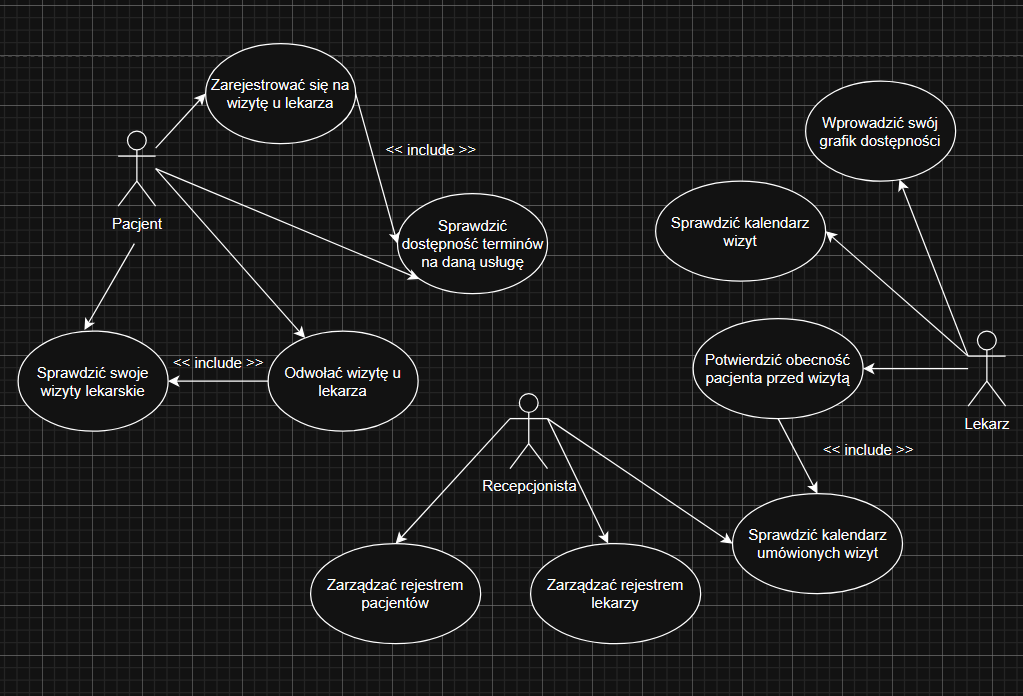
## Diagram przypadków użycia

Kiedy utworzony już został diagram modeli domenowych, można rozpocząć tworzenie diagramu przypadków użycia. Ten rodzaj diagramu pozwala uchwycić wymagania zachowania systemu tak, aby możliwym było go zaprojektowanie[[5]](#footnote-5). Ponadto, „k*ażdy przypadek użycia powinien zostać udokumentowany co najmniej trzema elementami:*

1. *opisem stanu początkowego (warunków początkowych),*
2. *scenariuszem,*
3. *opisem oczekiwanego stanu końcowego”[[6]](#footnote-6)*

Diagram przypadków użycia składa się z graficznego odwzorowania aktorów systemu oraz samych przypadków użycia, w których to aktorzy biorą udział[[7]](#footnote-7). Ponadto, diagram może zawierać również dwa dodatkowe typy relacji: *extend* oraz *include[[8]](#footnote-8)*. Pierwszy rozszerza przypadek użycia o scenariusz alternatywny. Drugi typ relacji zaś oznacza, że jeden z przypadków użycia składa się z jednego lub kilku innych scenariuszy

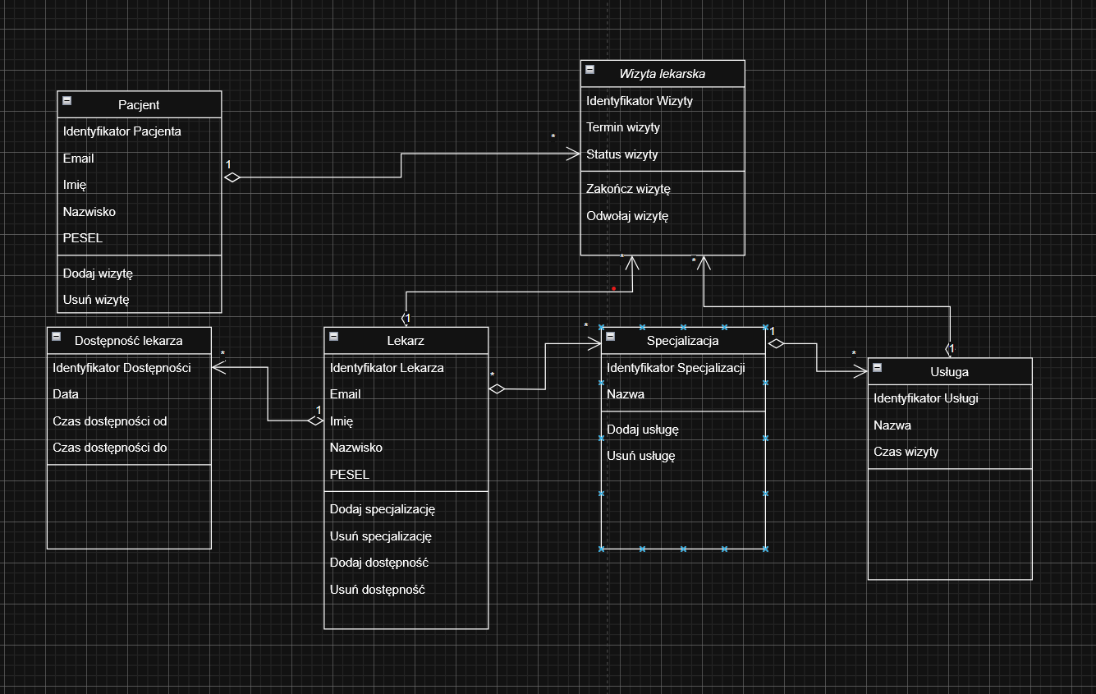
Na podstawie opisanych zasad, utworzonego diagramu modeli domenowych  
z rysunku 1.1, oraz opisu wymagań biznesowych, wykonany został diagram przypadków użycia przedstawiony na rysunku 1.2. Zawiera on trzech aktorów. Pierwszym z nich jest pacjent. Ma on możliwość umówienia się na wizyty u lekarza. Częścią tego przypadku użycia jest również sprawdzenie dostępności lekarzy. Pacjent ma również możliwość odwołać umówioną wcześniej wizytę u lekarza. W celu odwołania wizyty, użytkownik musi najpierw znaleźć swoją wizytę w systemie poprzez sprawdzenie swoich wizyt lekarskich. Kolejnym aktorem jest lekarz. Ma on możliwość wprowadzić swój grafik dostępności, sprawdzić kalendarz wizyt z pacjentami oraz zakończyć wizytę w systemie. Ostatni zaś aktor to recepcjonista. Jego zadaniem jest zarządzanie rejestrem lekarzy oraz rejestrem pacjentów. Ponadto, recepcjonista ma możliwość sprawdzenia kalendarza umówionych wizyt.



Rys. 1.2 Diagram przypadków użycia (opracowanie własne)

## Diagram klas

Zwieńczeniem analizy wymagań jest diagram klas przedstawiony na rysunku nr 1.3. „*Diagram klas to graficzny sposób odwzorowania klas zidentyfikowanych podczas zorientowanej obiektowo analizy oraz zachodzących między nimi relacji*”[[9]](#footnote-9). Analiza ta została wykonana na podstawie opisu wymagań biznesowych, diagramu modeli domenowych (rysunek nr 1.1) oraz diagramu przypadków użycia (rysunek nr 1.2). Wyodrębnionych zostało sześć klas. W skład każdej z nich wchodzi identyfikator, który będzie pomijany w następującym opisie. Będzie to atrybut szczególnie przydatny przy operacjach na bazie danych. Pierwszą klasą jest *Pacjent*. Składa ona się z następujących pól: *Email, Imię* oraz *Nazwisko*. Ponadto na obiektów klasy *Pacjent*, możliwe jest wywołanie metod *Dodaj wizytę* oraz *Usuń wizytę*. Kolejnym typem danych znajdującym się na diagramie jest *Wizyta lekarska*, która jednocześnie jest połączona z klasą *Pacjent* relacją agregacji w taki sposób, aby pacjent mógł posiadać wiele wizyt lekarskich. Atrybutami składającymi się na typ danych o nazwie *Wizyta lekarska* to *Termin wizyty* oraz *Status wizyty*. Ponadto, zadeklarowane zostały metody *Zakończ wizytę* i *Odwołaj wizytę*. Klasa *Wizyta lekarska* zagregowana jest również w typach danych o nazwach *Lekarz* oraz *Usługa*. Pierwszy z nich, zadeklarowane ma następujące pola: *Email, Imię, Nazwisko* i *Płeć*. Poza tym, występują tam takie metody jak - *Dodaj specjalizację*, *Usuń specjalizację, Dodaj dostępność*, czy *Usuń dostępność.* Natomiast *Usługa* jest typem danych złożonym m.in. z pól *Nazwa* oraz *Czas wizyty*. Oprócz klasy *Wizyta lekarska*, łączącej typy *Lekarz* i *Usługa*, na diagramie znajduje się również klasa o nazwie *Specjalizacja*. W jej skład wchodzi atrybut *Nazwa*. Klasa ta również posiada listę usług, stąd na diagramie zdefiniowana została relacja agregacji z typem *Usługa* oraz dwie metody – *Dodaj usługę* i *Usuń usługę*. Klasa *Specjalizacja* jest też zagregowana w typie *Lekarz*. Ostatnim typem danych znajdującym się na diagramie jest *Dostępność lekarza*. Podobnie do klasy *Specjalizacja*, jest ona zagregowana w typie *Lekarz*. Ponadto, w jej skład wchodzą następujące atrybuty – *Czas dostępności od* oraz *Czas dostępności do*.

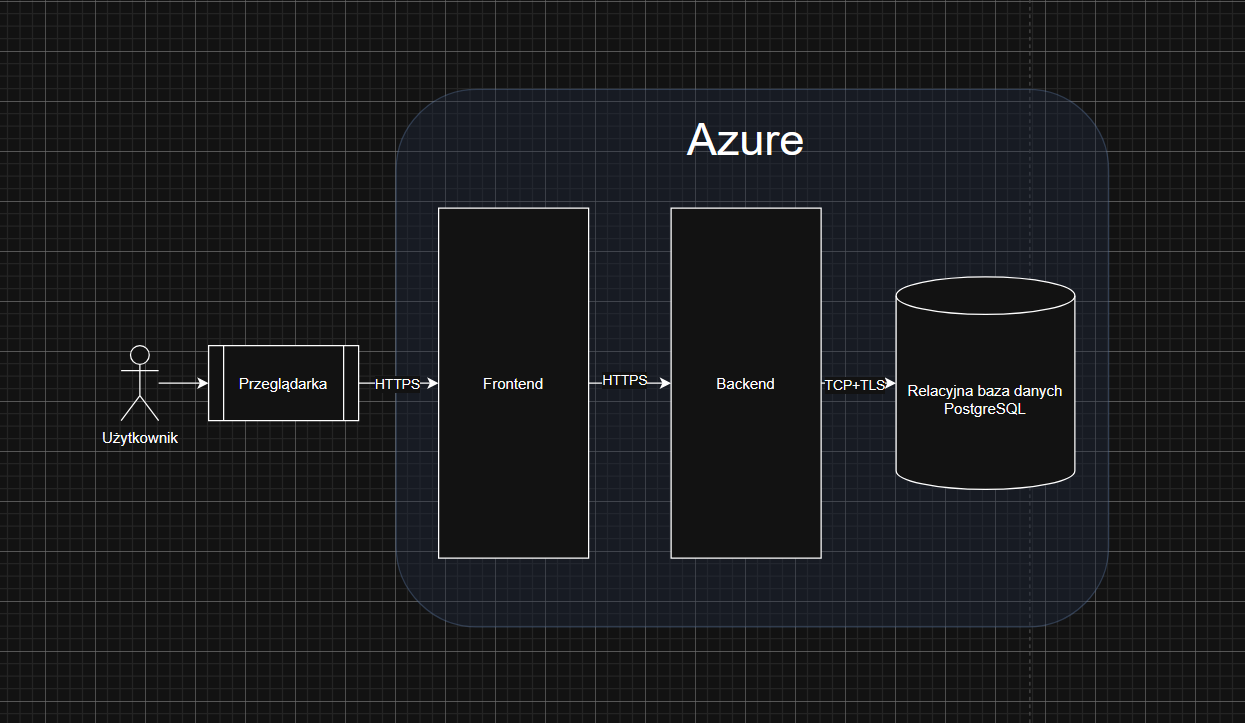


Rys. 1.3 Diagram klas (opracowanie własne)

# ARCHITEKTURA SYSTEMU

## Zarys architektury

System jest zbudowany z wykorzystaniem wzorca *Three-Tier Client Server Architecture[[10]](#footnote-10)*, który polega na podzieleniu całego systemu na trzy komponenty – warstwy prezentacji, warstwy logiki biznesowej oraz warstwy bazy danych. Każda z warstw jest osobną usługą sieciową. Pierwsza z nich, czyli warstwa prezentacji, jest również znana pod nazwą *frontend*, zaś druga pod nazwą *backend[[11]](#footnote-11).* Schemat w jakim pracują te trzy usługi polega na ich komunikacji między sobą - frontendu z backendem używając protokołu komunikacyjnego HTTPS oraz backendu z bazą danych używając protokołu TCP zaszyfrowanego protokołem TLS. Wizualizacja tej architektury trójwarstwowej przedstawiona jest na rysunku nr 2.1.



Rys. 2.1 Diagram architektury systemu (opracowanie własne)

Użytkownik wykorzystuje przeglądarkę do wyświetlenia warstwy prezentacji. Frontend generuje wizualizację danych, oraz umożliwia wykonanie na nich operacji. Kiedy użytkownik naciska któryś z wyświetlonych przycisków, generowane jest żądanie i przesyłane do warstwy logiki biznesowej. Backend wówczas wykonuje manipulacje na danych poprzez wysyłanie zapytań do bazy danych i na koniec zwraca zaktualizowane dane do frontendu, aby ten na koniec je wyświetlił użytkownikowi w przeglądarce. Żądania, które są generowane na frontendzie, wysyłane są z komputera użytkownika. Dlatego też, zarówno frontend, jak i backend są udostępnione w Internecie. Natomiast baza danych nie musi być dostępna z poziomu Internetu. Musi one jedynie znajdować się w tej samej sieci prywatnej, co backend. W ten sposób komunikacja backend – baza danych będzie przebiegać pomyślnie, a nikt spoza tej sieci nie będzie mógł wykonywać manipulacji na danych bezpośrednio się z nią komunikując, nawet jeżeli zna hasło, co jest dodatkowym zabezpieczeniem.

Głównymi zaletami architektury trójwarstwowej jest skalowalność, elastyczność oraz bezpieczeństwo[[12]](#footnote-12). Skalowalność odnosi się do możliwości zwiększania lub zmniejszania liczby instancji usługi osobno na każdym z trzech poziomów tak, aby obsłużyć wszystkich aktualnie korzystających z systemu użytkowników. Elastyczność zaś odzwierciedla fakt, że programiści mogą niezależnie pracować nad frontendem, backendem, a także bazą danych. Częstą praktyką jest również dzielenie zespołów na specjalistów z danej warstwy, to znaczy tzw. frontend developerów oraz backend developerów[[13]](#footnote-13). Natomiast zwiększone bezpieczeństwo wynika wprost z zastosowania zabezpieczeń na każdej warstwie aplikacji. Także wyżej wspomniane wyizolowanie bazy danych i umieszczenie jej w sieci prywatnej jest dobrym przykładem jak można skorzystać z wielowarstwowości architektury.

## Bezpieczeństwo systemu

System jest zabezpieczony na wielu poziomach. Przede wszystkim są to:

* Uwierzytelnienie przy użyciu Microsoft Entra ID
* Szyfrowane protokoły komunikacyjne przy użyciu TLS/SSL
* Uwierzytelnienie do bazy danych poprzez podanie nazwy użytkownika bazodanowego oraz hasła

Pierwsze dwa zostaną bardziej szczegółowo opisane w następujących podrozdziałach.

## Uwierzytelnienie przy użyciu Microsoft Entra ID

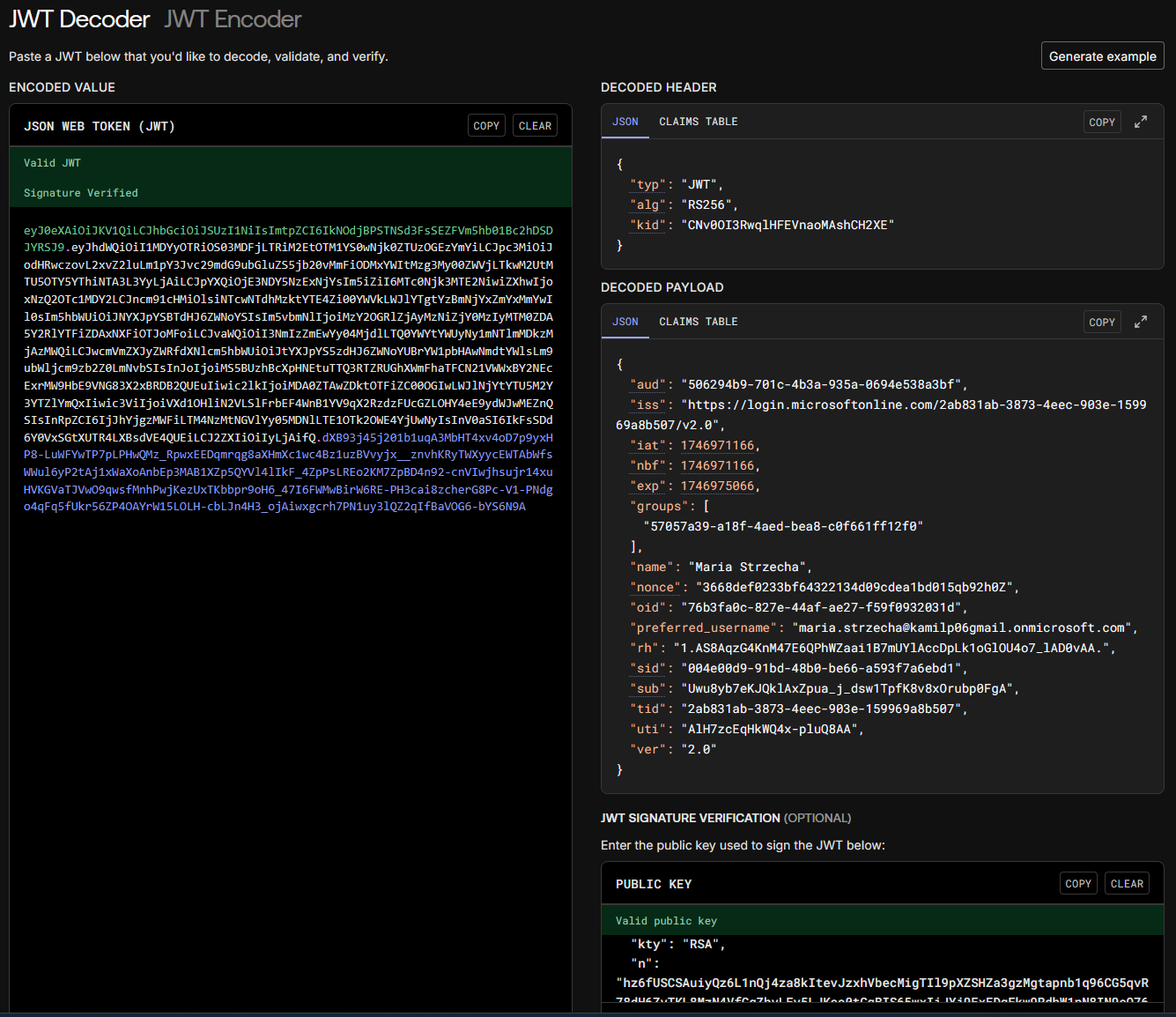
Pierwszym poziomem zabezpieczenia jest frontend. Kiedy użytkownik chce otworzyć jakąkolwiek stronę aplikacji w przeglądarce, zostaje automatycznie przekierowany na stronę logowania dostarczoną przez usługę Microsoft Entra ID z chmury Azure. Każdy, kto chce skorzystać z aplikacji musi mieć uprzednio utworzone konto w Microsoft Entra ID oraz dodany do odpowiedniej grupy. Grupy odzwierciedlają role, które odgrywają użytkownicy w systemie. W aplikacji *GreathHealth* są to pacjent, recepcjonista oraz lekarz. Po wprowadzeniu nazwy użytkownika oraz hasła, przeglądarka z powrotem przekierowuje użytkownika na stronę, którą chciał on odwiedzić. Po tym wydarzeniu, możliwe jest otwieranie wszystkich pozostałych udostępnionych stron poprzez panel nawigacyjny. Cały ten proces uwierzytelniania nosi nazwę *Authorization Code Flow with PKCE[[14]](#footnote-14)*. Schemat oraz kroki, które się na niego składają można zobaczyć na rysunku nr 2.2.



Rys. 2.2 Schemat działania uwierzytelnienia Authornization Code Flow with PKCE (https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow-with-pkce)

Kiedy użytkownik zostanie uwierzytelniony, frontend uzyskuje tzw. access token, który jest ciągiem znaków alfanumerycznych zawierającym dane dot. zalogowanego użytkownika. Dane te zapisane są zgodnie z formatem JSON Web Token (w skrócie JWT)[[15]](#footnote-15), który oprócz tego, że posiada dane dotyczące użytkownika, pozwala również na weryfikację źródła pochodzenia oraz integralności danych zakodowanych w tokenie. Weryfikacja ta możliwa jest dzięki temu, że token zawiera również podpis, który zweryfikować można za pomocą klucza publicznego dostarczonego przez Microsoft Entra ID[[16]](#footnote-16). Na rysunku nr 2.3 przedstawiony jest przykładowy token w formacie JWT wygenerowany z pomocą Microsoft Entra ID. Jak można zauważyć, składa się on z 3 części. Pierwsze dwie to nagłówek oraz dane użytkownika. Są one zakodowane przy użyciu algorytmu Base64[[17]](#footnote-17). Trzecia część to podpis umożliwiający weryfikację tokena.

Frontend uzyskując access token z Microsoft Entra ID, uzyskuje tym samym dostęp do m.in. imienia i nazwiska, adresu email oraz grup, do których należy użytkownik. Imię i nazwisko wyświetlane są na pasku nawigacyjnym w celu poinformowaniu kto został uwierzytelniony. Natomiast adres email wykorzystywany jest do identyfikacji użytkownika w backendzie i przypisywaniu do niego danych przechowywanych w bazie danych. Grupy zaś informują o tym, jakie role zostały przypisane do użytkownika. Na podstawie ról, frontend jest w stanie określić jakie strony i funkcjonalności powinny zostać wyświetlone.



Rys. 2.3 Przykład tokena JWT wygenerowanego przez Microsoft Entra ID oraz zdekodowanego na stronie https://jwt.io

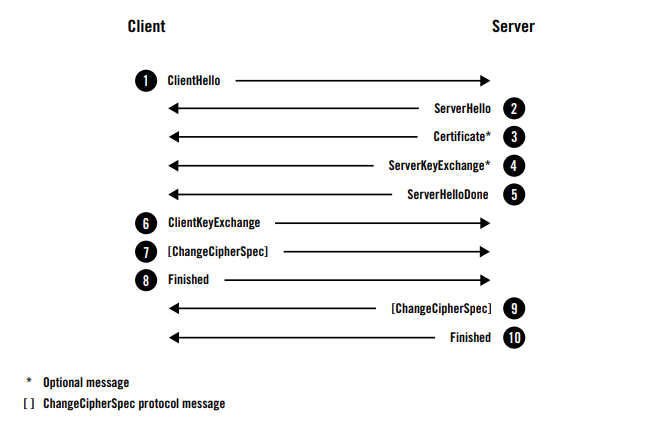
Frontend nie może być jedyną warstwą, która jest zabezpieczona przy użyciu Microsoft Entra ID. Backend, jako że wystawia API na zewnątrz, do Internetu, również musi weryfikować czy klient interfejsu (użytkownik) ma dostęp do wybranych funkcjonalności bądź danych. W tym celu, frontend przy każdym wysłanym żądaniu zawiera wspomniany token. Dzięki temu, backend może zweryfikować pochodzenie tokena. Ponadto, identyfikuje on tożsamość oraz sprawdza role zalogowanego użytkownika.

## Szyfrowane protokoły komunikacyjne przy użyciu TLS

Kolejnym komponentem zabezpieczenia systemu jest zastosowanie protokołu TLS w celu szyfrowania komunikacji między przeglądarką a usługami oraz między samymi usługami w systemie. Element ten jest konieczny w celu ochrony przed atakiem Man In The Middle (MITM). Tego typu atak polega na przechwyceniu komunikacji przez pośredniczący element sieciowy i jej odczycie lub modyfikacji tak, aby żadna ze stron o tym nie wiedziała[[18]](#footnote-18). W ten sposób atakujący może np. dokonać przelew z czyjegoś konta lub pozyskać dane poufne. Aplikacja *GreatHealth* wykorzystuje protokół HTTPS do ruchu sieciowego między przeglądarką a frontendem i backendem, który jest połączeniem protokołów HTTP oraz TLS[[19]](#footnote-19). Natomiast komunikacja pomiędzy backendem a bazą danych PostgreSQL wykorzystuje protokół TCP oraz TLS[[20]](#footnote-20). Pierwszy oraz drugi rodzaj ruchu sieciowego został zilustrowany na rysunku nr 2.1.

Zasada działania protokołu TLS opiera się na bezpiecznej wymianie utworzonego wspólnie przez obie strony klucza prywatnego, a następnie używania go do szyfrowania oraz deszyfrowania komunikatów. Proces tworzenia wspomnianego klucza prywatnego nazywany jest TLS Handshake i jest kluczowym elementem protokołu TLS[[21]](#footnote-21). Możemy wyróżnić następującego kroki.

1. Klient inicjuje TLS Handshake
2. Serwer odpowiada na zainicjowany TLS Handshake wraz z parametrami połączenia
3. Serwer wysyła swój certyfikat TLS tak, aby klient mógł zweryfikować jego poprawność
4. Serwer wysyła dane potrzebne do wygenerowania klucza prywatnego
5. Server oznajmia, że zakończył wysyłanie danych na tym etapie
6. Klient wysyła dane potrzebne do wygenerowania klucza prywatnego, szyfrując je kluczem publicznym serwera
7. Klient zaczyna wysyłać dane zaszyfrowane utworzonym kluczem prywatnym
8. Server zaczyna wysyłać dane zaszyfrowane utworzonym kluczem prywatnym



Rys. 2.4 Protokół TLS Handshake (Ivan Ristić, Bulletproof SSL and TLS, Feisty Duck Limited, s. 27)

Istotnym aspektem SSL Handshake jest to, że komunikacja już na tym etapie jest szyfrowana. Kiedy serwer wysyła swój certyfikat, zawiera on również klucz publiczny, za pomocą którego klient następnie szyfruje swoją kolejną wiadomość wysyłaną do serwera (pkt 6). Wiadomość ta zawiera pewną wartość, która połączona z poprzednią wartością wysłaną przez serwer, daje możliwość wygenerowania klucza prywatnego. Kiedy serwer otrzyma tę wiadomość, odszyfrowuje ją za pomocą swojego klucza prywatnego i jest w stanie wygenerować klucz prywatny po swojej stronie. Wówczas obie strony posiadają klucz prywatny, który następnie używany jest w komunikacji w celu ochrony przed atakiem Man In The Middle[[22]](#footnote-22).

## Frontend

Frontend jest usługą sieciową komunikującą się przy użyciu protokołu HTTPS. Przy użyciu serwera Nginx zwraca ona między innymi pliki JavaScript, CSS oraz HTML, które następnie służą do wyświetlenia aplikacji internetowej przez przeglądarkę. Komunikacja między frontendem a backendem również odbywa się przy pomocy protokołu HTTPS. Warto zauważyć, że to przeglądarka wykonuje kod źródłowy aplikacji frontendowej, a więc tak naprawdę komunikacja frontendu z backendem jest realizowana jako komunikacja przeglądarki z serwerem, ponieważ to w przeglądarce realizowany jest kod frontendu, zaś kod backendu na serwerze. Aplikacja została stworzona przy użyciu frameworka Angular[[23]](#footnote-23), gdzie kod źródłowy jest napisany w języku programowania TypeScript[[24]](#footnote-24), CSS[[25]](#footnote-25) oraz HTML[[26]](#footnote-26).

## HTML

HyperText Markup Language (w skrócie HTML)[[27]](#footnote-27) jest językiem znaczników, który definiuje treść strony internetowej wyświetlanej przez przeglądarkę. Kod źródłowy strony internetowej jest zapisywany w formie tekstowej w pliku o rozszerzeniu *.html*. Następnie po jego otwarciu przez przeglądarkę, zostaje wyświetlona treść strony internetowej. Kod źródłowy napisany w języku HTML ma formę hierarchiczną. Głównym komponentem występującym w tym języku jest znacznik. Najbardziej powszechnymi znacznikami są m.in. *body* (definiujący główną treść strony), *p* (definiujący paragraf), *h1* (definiujący nagłówek), *a* (definiujący odnośnik do innej strony), *div* (definiujący blok treści oddzielony nową linią) oraz *span* (definiujący blok treści nieoddzielony nową linią). Znaczniki mogą być zagnieżdżane, co sprawia, że cała struktura strony internetowej ma postać hierarchiczną. Przykładem kodu HTML oraz tego, jaki jest rezultat wyświetlenia go przez przeglądarkę można zaobserwować na rysunku nr 2.5.

Każdy rodzaj znacznika posiada pewną grupę atrybutów, które można im przypisać. Przykładem może być znacznik typu *img*, który definiuje obraz oraz atrybut *src* wskazujący plik graficzny. Innym przykładkę może być paragraf, czyli znacznik typu *p* wraz z atrybutem *style*, opisujący cechy stylistyczne paragrafu takie jak rozmiar czcionki bądź jej kolor.

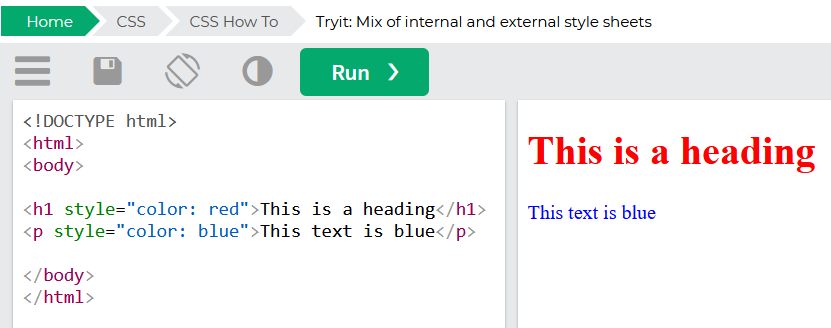


Rys. 2.5 Przykład kodu HTML oraz wyświetlenia go w przeglądarce (<https://udigroup.pl/blog/jezyk-html-co-to-jest-do-czego-sluzy-jak-wyglada/>)

## CSS

Cascading Style Sheets (w skrócie CSS)[[28]](#footnote-28) jest językiem uzupełniającym HTML, który służy do opisu warstwy prezentacji strony internetowej. Za pomocą języka CSS można panować nad rozmieszczeniem poszczególnych elementów na stronie, ich stylem oraz rozmiarem. Kod CSS można zastosować bezpośrednia w plikach *.html* przy użyciu atrybutu *style*, bądź jako oddzielny plik tekstowy o rozszerzeniu *.css* ładowany przez przeglądarkę przy pomocy znacznika *link* w pliku *.html* w znaczniku *head*.

Wyróżnia się 3 główne komponenty języka CSS. Pierwszym z nich jest selektor, który służy do wskazania elementów HTML, które zostaną objęte daną konfiguracją stylu. Wskazywać można rodzaje znaczników, konkretne elementy za pomocą identyfikatora lub na podstawie wartości atrybutów. Jednak najbardziej popularnym i polecanym selektorem jest selektor typu klasa, którą następnie można zastosować wybiórczo do wielu elementów HTML przy użyciu atrybutu *class* w wybranym znaczniku. Kolejnymi dwoma komponentami języka CSS są właściwości oraz wartości. Za ich pomocą definiować można cechy wyglądu elementów HTML. Dla przykładu przy użyciu właściwości *color* oraz wartości *red* można sprawić, że czcionka będzie czerwona. Przykład takie użycia zobrazowany jest na rysunku nr 2.6.



Rys. 2.6 Przykład użycia CSS (opracowanie własne przy użyciu strony <https://www.w3schools.com>)

## TypeScript

TypeScript**[[29]](#footnote-29)** jest językiem programowania, który jest rozszerzeniem języka JavaScript**[[30]](#footnote-30)**. Wprowadza on m.in. statyczne typowanie, które zdecydowanie zmniejsza liczbę błędów w kodzie oraz zapewnia dodatkową dokumentację. TypeScript jest kompilowany do języka JavaScript, który to kod jest wówczas wykonywany przez przeglądarkę. Kompilacja ta jest częścią procesu budowy projektu, co oznacza, że na etapie instalacji aplikacji istnieje już tylko skompilowany kod w języku JavaScript. Oba te języki wprowadzają dynamikę do strony internetowej. Pozwalają na zaprogramowanie akcji dostępnych na stronie oraz integrację z backendem, bądź innymi usługami sieciowymi.

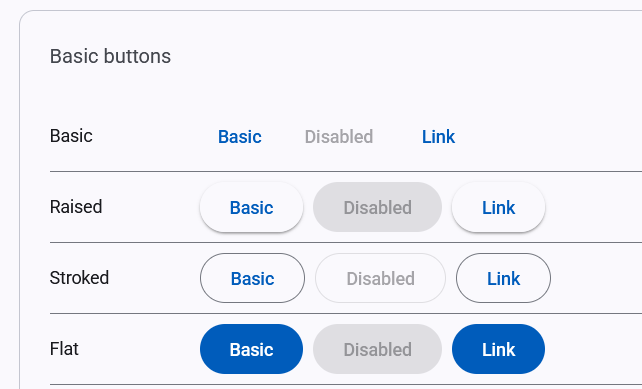
## Angular

Angular[[31]](#footnote-31) jest frameworkiem frontendowym napisanym w języku TypeScript przez firmę Google. Wprowadza on narzędzia do tworzenia aplikacji internetowych m.in. typu SPA (Single Page Application). Aplikacje tego typu w porównaniu do SSR (Server Side Rendering) charakteryzują się ogólnie szybszym działaniem, kosztem wolniejszego załadowania się strony na samym początku. Angular zapewnia projektowi spójną strukturę i wprowadza możliwość tworzenia komponentów, na które składają się plik z kodem HTML, plik z kodem CSS oraz plik z kodem TypeScript. Dzięki takiej strukturze, element ten jest spójnym tworem, gdzie kod HTML przedstawia treść, kod CSS wprowadza styl, natomiast kod TypeScript logikę biznesową. Komponent może definiować podstronę, ale również powtarzalny element, który jest wykorzystywany w wielu miejscach, w projekcie.

Ponadto, Angular wprowadza tzw. framework CSS o nazwie Angular Material[[32]](#footnote-32). Narzędzie to wprowadza grupę predefiniowanych komponentów dzielących ten sam styl, które można wykorzystać do zbudowania aplikacji. Przykładowymi komponentami, które wprowadza Angular Material są ikona, przycisk, panel nawigacyjny, kalendarz, czy też tekstowe pole formularza. Wygląd kilku z nich przedstawiony jest na rysunkach o numerach 2.7 oraz 2.8.



Rys. 2.7 Przykładowe pola formularza z Angular Material (https://material.angular.dev/components/input/overview)



Rys. 2.8 Przykładowe przyciski z Angular Material (https://material.angular.dev/components/button/overview)

## Backend

## Baza danych

1. Karl E Wiegers, Joy Beatty, Specyfikacja oprogramowania. Inżynieria wymagań. Wydanie III, Helion, 2014, s. 35 [↑](#footnote-ref-1)
2. Karl E Wiegers, Joy Beatty, Specyfikacja oprogramowania. Inżynieria wymagań. Wydanie III, Helion, 2014, s. 35 [↑](#footnote-ref-2)
3. Doug Rosenberg, Matt Stephens, Use Case Driven Object Modeling with UML. Theory and Practice. Appress, 2007, s. 25 [↑](#footnote-ref-3)
4. Doug Rosenberg, Matt Stephens, Use Case Driven Object Modeling with UML. Theory and Practice. Appress, 2007, s. 26 [↑](#footnote-ref-4)
5. Doug Rosenberg, Matt Stephens, Use Case Driven Object Modeling with UML. Theory and Practice. Appress, 2007, s. 49 [↑](#footnote-ref-5)
6. Jarosław Żeliński, Analiza Biznesowa. Praktyczne modelowanie organizacji, Helion, 201, s. 74 [↑](#footnote-ref-6)
7. Karl E Wiegers, Joy Beatty, Specyfikacja oprogramowania. Inżynieria wymagań. Wydanie III, Helion, 2014, s. 172 [↑](#footnote-ref-7)
8. Karl E Wiegers, Joy Beatty, Specyfikacja oprogramowania. Inżynieria wymagań. Wydanie III, Helion, 2014, s. 179 [↑](#footnote-ref-8)
9. Karl E Wiegers, Joy Beatty, Specyfikacja oprogramowania. Inżynieria wymagań. Wydanie III, Helion, 2014, s. 261 [↑](#footnote-ref-9)
10. Pethuru Raj, Anupama Raman, Harihara Subramanian, Architectural Patterns, Packt Publishing, 2017, s. 58 [↑](#footnote-ref-10)
11. https://www.geeksforgeeks.org/frontend-vs-backend/ [↑](#footnote-ref-11)
12. Pethuru Raj, Anupama Raman, Harihara Subramanian, Architectural Patterns, Packt Publishing, 2017, s. 60 [↑](#footnote-ref-12)
13. https://medium.com/buildingminds-technologies/importance-of-collaboration-between-frontend-and-backend-teams-4b05e8fd29f9 [↑](#footnote-ref-13)
14. https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow-with-pkce [↑](#footnote-ref-14)
15. https://learn.microsoft.com/en-us/entra/identity-platform/access-token-claims-reference [↑](#footnote-ref-15)
16. https://learn.microsoft.com/en-us/entra/identity-platform/security-tokens#token-endpoints-and-issuers [↑](#footnote-ref-16)
17. https://jwt.io/introduction [↑](#footnote-ref-17)
18. Avijit Mallik, Mhia Md. Zaglul Shahadat, Jia-Chi Tsou, Abid Ahsan, Man-in-the-middle-attack: Understanding in simple words, International Journal of Data and Network Science, artykuł na Research Gate, s. 80 [↑](#footnote-ref-18)
19. https://www.rfc-editor.org/rfc/rfc2818 [↑](#footnote-ref-19)
20. https://www.postgresql.org/docs/current/ssl-tcp.html [↑](#footnote-ref-20)
21. Ivan Ristić, Bulletproof SSL and TLS, Feisty Duck Limited, s. 25 [↑](#footnote-ref-21)
22. https://auth0.com/blog/the-tls-handshake-explained/ [↑](#footnote-ref-22)
23. https://angular.dev/ [↑](#footnote-ref-23)
24. https://www.typescriptlang.org/ [↑](#footnote-ref-24)
25. https://developer.mozilla.org/en-US/docs/Web/CSS [↑](#footnote-ref-25)
26. https://developer.mozilla.org/en-US/docs/Web/HTML [↑](#footnote-ref-26)
27. https://udigroup.pl/blog/jezyk-html-co-to-jest-do-czego-sluzy-jak-wyglada/ [↑](#footnote-ref-27)
28. https://webporadnik.pl/css-co-to-jest-i-jak-dziala-css-podstawowe-i-najwazniejsze-informacje-o-css/ [↑](#footnote-ref-28)
29. https://www.typescriptlang.org/ [↑](#footnote-ref-29)
30. https://www.w3schools.com/Js/ [↑](#footnote-ref-30)
31. https://angular.dev/overview [↑](#footnote-ref-31)
32. https://material.angular.dev/ [↑](#footnote-ref-32)