ChatGPT

Audyt HTML projektu TransLogix – gotowość produkcyjna

Semantyka i struktura dokumentu HTML

Struktura strony opiera się na semantycznych elementach HTML5, co jest **dobrą praktyką**. Na każdej podstronie widoczny jest podział na **nagłówek (header)** z nawigacją, **główną zawartość (main)** oraz **stopkę (footer)**. Dzięki temu kod jest czytelny i daje przeglądarkom oraz czytnikom ekranu jasne punkty orientacyjne [1]. Użyto odpowiednich elementów sekcji, np. `<section>` do logicznego wydzielenia bloków treści, a w obrębie sekcji stosowane są semantyczne nagłówki (`<h1>…<h4>`).

Nagłówki i hierarchia: Każda strona (poza dynamiczną *service.html*) posiada **dokładnie jeden** `<h1>` z unikalną treścią, zgodny z tematem podstrony (np. *"Transport i logistyka szyte na miarę"* na stronie usług). Nagłówki kolejnych poziomów są używane w logiczny sposób – sekcje główne mają `<h2>`, a mniejsze komponenty (np. karty usług, FAQ) używają `<h3>` i niżej. Taka struktura jest prawidłowa i wspiera zarówno **czytelność kodu**, jak i **SEO** (wyszukiwarki przykładają większą wagę do słów kluczowych w nagłówkach niż w zwykłym tekście [2]). W jednym miejscu można rozważyć poprawę: w stopce listy statystyk użyto `<h3>` do wyróżnienia liczby (np. *612+* dostaw). Semantycznie nie są to nagłówki sekcji treści, a jedynie wyróżnione wartości, więc dla czystości hierarchii można by je zastąpić np. `<p>` lub `<span>` z klasą stylującą. Nie wpływa to jednak znacząco na odbiór treści.

Nawigacja: Główne menu jest zrealizowane wewnątrz elementu `<nav aria-label="Główna nawigacja">`, co jasno określa jego rolę. Elementy menu to linki w kontenerze `<div class="nav__links">`. Technicznie, brak listy `<ul>` z `<li>` w menu nie łamie standardu (nav może zawierać same linki), choć użycie listy uporządkowałoby strukturę w kodzie. Z perspektywy dostępności nie jest to błąd krytyczny. Ważne, że **każdy link jest tekstowy i zrozumiały** (np. *Start*, *Usługi*, *Kontakt*). Dodatkowo, w kodzie zadbano o opis linku do strony głównej poprzez `aria-label="Strona główna TransLogix"` na elemencie `<a>` marki – dzięki temu czytnik ekranu odczyta cel tego linku jednoznacznie, zamiast tylko nazwę firmy.

Sekcje i artykuły: W treści użyto `<section>` dla głównych bloków (np. sekcja *Dlaczego my*, sekcja *Flota* itp.), a wewnątrz niektórych umieszczono `<article>` dla samodzielnych elementów (np. karty floty, karty usług). Takie użycie jest zgodne z semantyką – `<article>` oznacza niezależny element, co pasuje np. do pojedynczej karty usługi czy pojazdu. Każdy artykuł ma swój tytuł (`<h3>`), co jest poprawne i pozwala na łatwą nawigację po tych elementach.

Podział na pliki i spójność: Wszystkie podstrony mają analogiczną strukturę szkieletu HTML, co wskazuje na przemyślane podejście. Różnice w kodzie `<head>` (omówione w dalszych sekcjach) są niewielkie. Warto docenić utrzymanie spójnego układu: na każdej stronie nagłówek i stopka są identyczne, a główna treść zaczyna się od sekcji z krótkim wprowadzeniem (tzw. *eyebrow* – mały opis, następnie główny nagłówek `<h1>` i paragraf lead). Taki schemat powtarza się (np. *Oferta -> Transport i logistyka szyte na miarę*, *Flota -> Sprzęt, który dowozi wyniki*, *Kontakt -> Porozmawiajmy o Twoim ładunku*). To nadaje serwisowi konsekwencji i profesjonalizmu.

Dynamiczna strona `service.html`**:** Ten plik stanowi szablon szczegółów usługi, ładowanych przez JS na podstawie parametru `?id`. W statycznym HTML sekcja ma tylko placeholder (*"Ładowanie szczegółów usługi..."*). Semantyka w tym pliku jest uboga – brak tu własnych nagłówków czy treści (poza komunikatem inicjalnym). To zrozumiałe, jeśli zawartość jest generowana dynamicznie. **W kontekście SEO i dostępności** może to być problem: robot indeksujący lub użytkownik bez JS zobaczy pustą stronę. Rozwiązaniem produkcyjnym może być albo **Server-Side Rendering** dla tych danych, albo przynajmniej mechanizm *prerendering* statyczny, tak by kluczowe informacje o usłudze (np. tytuł usługi jako `<h1>`) znalazły się w HTML. W przeciwnym razie podstrony usług mogą nie być zaindeksowane poprawnie. To istotna kwestia, jeśli szczegóły usług mają być widoczne w wynikach wyszukiwania.

Podsumowując, **semantyka HTML stoi na wysokim poziomie**. Kod używa właściwych elementów do ich zamierzonych celów, co zapewnia solidne podstawy pod **dostępność i SEO**. Jest to przykład trzymania się zasady "odpowiedni element do odpowiedniego zadania" [3] [2]. Struktura dokumentu jest logiczna i czytelna – zarówno dla deweloperów, jak i dla automatów przetwarzających HTML.

Dostępność (Accessibility)

Projekt wykazuje wiele dobrych praktyk z zakresu dostępności, co świadczy o profesjonalnym podejściu do warstwy front-end. Poniżej omówienie kluczowych aspektów:

- **Landmarki ARIA / HTML5:** Dzięki użyciu elementów takich jak `<header>`, `<main id="main">` i `<footer>` czy `<nav>` z odpowiednimi atrybutami, strona jest podzielona na regiony zrozumiałe dla czytników ekranu. Element `<main>` posiada atrybut `id="main"`, co pozwala innym mechanizmom (np. skip link) łatwo na niego wskazać. HTML5 sam dostarcza ról ARIA dla takich elementów – np. `<main>` jest traktowany jak region głównej zawartości, a `<nav>` jako nawigacja [1]. Nie ma potrzeby duplikowania ich rolami typu `role="main"` – autor poprawnie tego nie robi, unikając zbędnej redundancji.

- **Skip link (pomijanie do treści głównej):** Na początku kodu `<body>` każdej strony znajduje się link `<a class="skip-link" href="#main">Przejdź do treści głównej</a>`. To **bardzo ważny element dostępności**, umożliwiający użytkownikom klawiatury i czytników ekranu przeskoczenie od razu do głównej treści, omijając powtarzalne menu [4]. Implementacja jest poprawna – link prowadzi do kotwicy `#main` (na <main>), jest zapewne ukrywany CSS-em i pokazywany dopiero po fokusu (tego z kodu HTML nie widać, ale zakładamy typową implementację). Obecność skip linka świadczy o dojrzałości projektu, bo często jest on pomijany w mniej profesjonalnych realizacjach.

- **Nawigacja mobilna i przyciski:** Menu główne ma ukryty panel rozwijany (`<div id="mobile-nav" hidden>`), otwierany przez przycisk hamburgerowy `<button class="nav__toggle" aria-expanded="false" aria-controls="mobile-nav" aria-label="Otwórz menu">`. Atrybut `aria-controls` prawidłowo wskazuje na element panelu, a `aria-expanded` dynamicznie będzie zmieniany przez skrypt – domyślnie jest *false*, czyli menu zamknięte. Co ważne, przycisk ma **opis dostępności** (`aria-label="Otwórz menu"`), więc jest zrozumiały dla osób niewidzących. Gdy menu się otwiera, powinno nastąpić: zmiana `aria-expanded` na "true", usunięcie atrybutu `hidden` z panelu oraz ustawienie focusu na pierwszym elemencie menu (opcjonalnie zaimplementowanie *focus trap*, aby TAB poruszał się tylko w obrębie menu). W samym HTML widać przygotowanie pod te mechanizmy – to oznaka, że autor zna wzorce dostępności dla nawigacji. **Rekomendacja:** Dodać atrybut `aria-expanded="false"` również do samego panelu lub zarządzać stanem poprzez wspomniane właściwości – ale to już szczegóły implementacji JS. Ważne, że podstawa (powiązanie przycisku z panelem, opis) jest na miejscu.

- **Przełącznik motywu (dark mode):** W nagłówku znajduje się przycisk zmiany trybu kolorów `<button class="theme-toggle" aria-pressed="false" aria-label="Przełącz tryb kolorów">`. Tu również zadbano o **właściwy opis funkcji przycisku** poprzez `aria-label`. Atrybut `aria-pressed="false"` sugeruje, że jest to przycisk typu *toggle* (włącz/wyłącz tryb) – po aktywacji skrypt zmieni `aria-pressed` na *true/false* zależnie od stanu. Obrazy ikon (słońce/księżyc) wewnątrz przycisku mają `alt=""` i `aria-hidden="true"`, więc nie będą mylnie odczytywane – to prawidłowe, bo tekstowy opis już istnieje. Całość wskazuje na zgodność z wzorcem dostępności dla przycisków przełączających stan.

- **Formularze i etykiety:** W kilku miejscach występują formularze (np. szybka wycena na stronie głównej i cenniku, oraz formularz kontaktowy na stronie Kontakt). Każdy **formularz jest właściwie ostylowany i oznaczony etykietami**:

- Użyto elementów `<label for="...">` powiązanych z polami `<input>` / `<select>` poprzez zgodne wartości `for` i `id`. Dzięki temu użytkownik czytnika słyszy opis pola, np. "*Dystans (km)*" dla pola odległości. Jest to fundamentalna praktyka dostępności [5] [6] – formularze w projekcie spełniają ten standard.

- Pola obowiązkowe mają atrybut HTML `required`, co informuje natywnie przeglądarkę i użytkownika (np. przez komunikaty walidacji). Dodatkowo, zastosowano mechanizm na komunikaty błędów: przy każdym polu jest `<div class="form__error" id="pole-error"></div>` oraz atrybut `aria-describedby="pole-error"` przy polu. Oznacza to, że jeśli skrypt umieści komunikat błędu w tym divie, zostanie on odczytany jako rozszerzony opis pola. To bardzo dobry wzorzec – zapewnia, że np. po błędnym wypełnieniu, czytnik powie użytkownikowi co jest nie tak.

    - **Uwaga:** Można rozważyć dodanie `role="alert"` lub `aria-live="polite/assertive"` do tych komunikatów błędów, by czytniki ogłaszały je automatycznie w momencie pojawienia. W projekcie zastosowano `aria-live="polite"` na elementach podsumowania wyceny i historii, więc autor zna tę technikę. Dla komunikatów błędów (jeśli walidacja jest po stronie JS) warto dodać np. `aria-live="assertive"` na kontener błędu, aby od razu go zakomunikować.

- Pola mają sensowne typy (`type="email"`, `type="tel"`, `type="number"`) co ułatwia ich obsługę na urządzeniach mobilnych (klawiatury kontekstowe) i walidację. Atrybuty typu `min`, `step` przy liczbach zostały ustawione – to dobrze, bo zapobiega np. wprowadzeniu 0 km dystansu itp.

- Przy polu telefonu zastosowano `type="tel"` – to ułatwienie (klawiatura numeryczna na telefonie), choć **dostępnościowo** można rozważyć dodanie `pattern` lub atrybutu `inputmode` dla poprawy ogłoszenia oczekiwanego formatu. To jednak drobny szczegół.

- Checkboxy i radio (tutaj checkboxy *Dodatki* i zgoda RODO) są również poprawnie etykietowane. Zgodę RODO rozwiązano sprytnie: etykieta obejmuje checkbox i tekst, a checkbox ma `required` – wymuszając zaznaczenie. Takie połączenie sprawia, że **brak zgody zasygnalizuje błąd**, co jest kluczowe prawnie i UX. Ten element jest zrozumiały dla czytników (tekst etykiety jest w `<label>` bez osobnego `for`, bo input jest wewnątrz label – to też poprawna metoda powiązania).

- **Mapa Google:** W sekcji kontakt jest osadzona mapa `<iframe>` z Google Maps. Autor zadbał o `title` tego iframe (`title="Mapa dojazdu do ul. ... Tarnów"`) oraz umieścił go w `<div class="map-embed" aria-label="Mapa dojazdu">`. To zapewnia opis kontekstu mapy. `aria-label` na divie nie zawsze jest wykorzystywane przez czytniki (div nie ma domyślnej roli), ale sam atrybut title wewnątrz iframe jest najważniejszy – oznacza treść mapy dla dostępności. Ewentualnie można by opakować mapę w `<figure>` z `<figcaption>` opisującym cel mapy, ale obecne rozwiązanie też przekazuje potrzebną informację.

- **Komponenty interaktywne (tabs, accordion, lightbox):** Projekt zawiera zaawansowane komponenty UI i widać, że zadbano o ich dostępność statyczną:

- **Zakładki (tabs)** na stronie *Usługi*: Struktura zakładek jest zgodna ze specyfikacją ARIA. Mamy kontener listy z `role="tablist"` i `aria-label="Zestawienia usług"`, wewnątrz przyciski `role="tab"` z atrybutami `aria-selected` (pierwszy true, reszta false), `aria-controls` wiążącymi je z panelami oraz unikalne `id` dla każdego taba. Panele mają `role="tabpanel"`, atrybut `aria-labelledby` wskazujący na swój tab oraz `tabindex="0"` i `hidden` (oprócz pierwszego). Taka konfiguracja spełnia wytyczne WAI-ARIA – użytkownik może klawiszami strzałek przełączać taby, a fokus przeniesie się do panelu (dzięki tabindex). Po implementacji JS zakładki będą dostępne z klawiatury i czytników praktycznie od razu. To **bardzo dobrze wykonany element** – na poziomie oczekiwanym od doświadczonego front-end developera.

- **Akordeon FAQ:** Każde pytanie jest zbudowane jako przycisk w nagłówku `<h3>` z `aria-expanded` i `aria-controls` oraz unikalnym ID panelu (np. faq1). Panel odpowiedzi ma `role="region"` (daje semantykę regionu) z `aria-labelledby` wskazującym ten przycisk i atrybutem `hidden`. Struktura jest poprawna – pytania są częścią struktury nagłówków (nagłówek sekcji FAQ ma `<h2>`, pytania `<h3>`), więc czytnik może je przeglądać na liście nagłówków. Przyciski posiadają w sobie tekst pytania, więc są czytelne także poza kontekstem. Po otwarciu skrypt powinien ustawić `aria-expanded="true"` i usunąć `hidden` z panelu. Byłoby idealnie dodać jeszcze `aria-disabled="true"` na otwartym pytaniu (by wskazać, że nie można otworzyć ponownie tego samego) lub zastosować mechanizm zwijania, ale to już szczegół UX. Ważne, że komponent spełnia wymagania dostępności – to kolejne potwierdzenie wysokiej jakości implementacji.

- **Lightbox galerii (Flota):** Kod HTML zawiera strukturę modalu do podglądu obrazków. `<div class="lightbox" role="dialog" aria-modal="true" aria-hidden="true" hidden>` definije modalne okno dialogowe, co jest poprawne – `role="dialog"` i `aria-modal="true"` oznajmiają czytnikowi, że pojawia się modal, a tło staje się niedostępne. Wewnątrz jest m.in. tytuł `<h3 class="lightbox__title">Galeria pojazdu</h3>`, przycisk zamykający z `aria-label="Zamknij"`, kontenery na obraz i przyciski Next/Prev (też z `aria-label`). **To pokazuje dbałość o detale:** nawet przyciski do przewijania obrazów mają ukryty tekst dla czytnika (zapewniony przez aria-label), choć wizualnie mogą być ikonami. Jedyna sugestia: żeby modal był w pełni samowystarczalny, warto dodać do kontenera dialogu atrybut `aria-labelledby="ID_tytułu"` wskazujący na ten nagłówek `<h3>`. Wtedy czytnik po otwarciu modalu mógłby odczytać nazwę dialogu (np. “Galeria pojazdu, dialog, zawartość…”). Aktualnie tego nie ma, więc użytkownik niewidzący usłyszy "dialog" bez nazwy kontekstowej. Jest to **łatwa poprawka** (dodanie id do h3 i aria-labelledby do `.lightbox`). Poza tym, pamiętać należy o zarządzaniu fokusem: po otwarciu modalu focus powinien przejść na np. pierwszą miniaturę lub przycisk zamknij, a poruszanie się TAB-em powinno ograniczyć do elementów

modalu (tzw. *focus trap*). Te kwestie są czysto JavaScriptowe – sam HTML daje solidną bazę do ich zaadresowania.

- **Inne drobne elementy dostępności:**

- Teksty linków i przycisków są zrozumiałe samodzielnie. Np. linki *"Szczegóły"* przy ofertach usług – wiadomo, że odnoszą się do konkretnej usługi (choć **można by** dodać `aria-label` zawierający nazwę usługi, aby link sam w sobie mówił np. "Szczegóły: Ekspres PL → DE", co pomaga, gdy czytnik wylistuje same linki na stronie). Obecnie link *Szczegóły* czytany bez kontekstu może być mniej jasny. To jednak **usprawnienie kosmetyczne** – w strukturze kodu link jest tuż po tytule usługi, więc większość czytników odczyta całość sensownie.
- W menu stopki linki do social mediów mają same ikony, ale zadbano o `aria-label="Facebook/Instagram/..."` na każdym `<a>`, zaś obrazki ikon mają pusty alt. To perfekcyjnie realizuje zasadę, że **link musi mieć tekst dostępny** (czy to bezpośrednio, czy poprzez aria-label) – tutaj jest to zrobione, unikając jednocześnie podwójnego czytania nazwy (gdyby alt nie był pusty, czytnik czytałby np. "Facebook, link, Facebook" [7]). Ten detal potwierdza profesjonalne podejście.

Podsumowując, **warstwa dostępności jest niemal produkcyjna**. Projekt zawiera wszystkie kluczowe elementy: poprawna semantyka, skip link, etykiety formularzy, opisy alternatywne, wykorzystanie ARIA do niestandardowych komponentów (tabs, accordion, dialog) – i robi to zgodnie ze standardami. Większość uwag to drobne usprawnienia (jak aria-labelledby w modalu czy ulepszone etykiety linków), które należą do tzw. *polish* rather than fundamental issues. Nie wykryto poważnych braków typu niedostępne formularze czy niezrozumiałe linki – wręcz przeciwnie, widać dbałość o szczegóły. To wskazuje, że projekt przygotowywał ktoś świadomy zasad WCAG i dobrych praktyk a11y.

SEO i metadane

Z punktu widzenia **SEO (Search Engine Optimization)** projekt został przygotowany bardzo solidnie. Zawiera on większość istotnych elementów wpływających na indeksowalność i prezentację w wynikach wyszukiwania:

- **Unikalne tytuły stron (`<title>`):** Każdy plik HTML ma własny, opisowy tytuł, zawierający zarówno nazwę sekcji strony, jak i markę *TransLogix*. Np. *"Usługi transportowe | TransLogix – KP_Code_ Demo"*, *"Cennik i kalkulator transportu | TransLogix – ..."*. Tytuły są zwięzłe, unikalne dla każdej podstrony i wykorzystują słowa kluczowe (np. usługi transportowe, flota pojazdów, szybka wycena). To dobra praktyka SEO – tytuł stanowi kluczowy czynnik rankingu i klikalności w wynikach. Drobna uwaga: dopisek *"KP_Code_ Demo"* sugeruje, że to wersja demo. Do środowiska produkcyjnego należałoby to usunąć lub zastąpić hasłem firmowym. W obecnej formie nie szkodzi SEO, ale może obniżać percepcję profesjonalizmu na zewnątrz (użytkownik widząc *Demo* w tytule może pomyśleć, że strona nie jest gotowa).

- **Meta description:** Każda strona posiada meta opis (`<meta name="description" content="...">`) w języku polskim, streszczający zawartość i zawierający ważne frazy. Przykładowo, strona główna ma opis: *"TransLogix to demo serwisu transportowo-logistycznego B2B. Łączymy przewozy drogowe, monitoring floty..."* itd. Opisy mieszczą się w zalecanej długości (ok. 150-160 znaków) i **wydają się unikalne oraz atrakcyjne** – zachęcają do kliknięcia, wymieniając atuty (np. monitoring 24/7, terminowe dostawy). Tak przygotowane meta description powinno pojawić się jako snippet w Google i poprawić CTR wyników. To element często pomijany przez początkujących – tutaj jest zrobiony bardzo dobrze.

- **Nagłówki treści a SEO:** Struktura nagłówków (omówiona wcześniej) nie tylko pomaga dostępności, ale też SEO. Główny nagłówek `<h1>` każdej strony zawiera kluczowe frazy (np. *Transport, który dowozi Twój biznes na czas* – zawiera "transport", *Usługi transportowe szyte na miarę* – zawiera "usługi transportowe" itd.). Podnagłówki `<h2>` rozwijają te tematy (np. "Wspieramy Twoją logistykę..."). Dzięki temu wyszukiwarka łatwo rozpozna tematykę każdej podstrony i powiąże słowa kluczowe z ważnymi elementami strony [2]. Nie zauważono nadużycia nagłówków (brak sztucznego upychania słów kluczowych) – treść wygląda naturalnie i informacyjnie, co jest zgodne z nowoczesnym SEO nastawionym na jakość contentu.

- **Linki kanoniczne (canonical):** W sekcji `<head>` każdej strony umieszczono `<link rel="canonical" href="...">` wskazujący na swoją właściwą URL (np. dla *services.html* kanoniczny to *.../services.html*). To zabezpiecza przed ewentualnym duplikowaniem treści pod różnymi adresami. W kontekście hostingu na Netlify, gdzie strony są dostępne zarówno z i bez `.html` w URL, linki te mówią wyszukiwarce, którą wersję uważać za główną. Kanoniczne adresy wydają się poprawne i wskazują na adresy produkcyjne (transport-project-01.netlify.app). **Spójność:** zauważono, że na stronie głównej canonical ma końcowy `/` (po domenie), a podstrony mają pełne nazwy plików `.html`. To jest OK. Potencjalnie, jeśli strona będzie przeniesiona na własną domenę lub inne URL, należy pamiętać o aktualizacji tych linków.

- **Meta robots:** `<meta name="robots" content="index,follow" />` jest obecny, co oznacza, że strona nie blokuje indeksacji (to domyślne ustawienie, ale jawne podanie nie szkodzi). W kontekście demo pewnie i tak chciałoby się indeksować, więc to w porządku.

- **Open Graph i Twitter Cards:** Wszystkie podstrony mają zestaw meta tagów **Open Graph** (og:*) oraz **Twitter Card**, co jest często pomijane w prostych projektach. Tutaj:

- Ustawiono og:type (wszędzie `website` – co jest poprawne dla stron firmowych, choć dla konkretnych podstron można by użyć og:article, ale nie jest to konieczne, typ `website` jest uniwersalny).
- Dodano og:title, og:description – z treściami odpowiadającymi meta title/description (czasem skrócone). To zapewni ładne podglądy przy udostępnianiu linków na Facebooku, LinkedIn itp.
- og:image wskazuje adres obrazka do udostępnień (1200x630). Alt dla og:image również jest dodany (np. *"TransLogix – transport and logistics B2B"*), co jest **dodatkowym smaczkiem** – ma znaczenie dla indeksacji obrazów i dostępności (np. Twitter używa tego alt).
- Twitter:card ustawiono na summary_large_image, co zgodne z rozmiarem obrazka i zapewni dużą kartę. Twitter:title/description są obecne, podobnie image.

Te tagi świadczą o dbałości o **social SEO** i branding. Jedyna niespójność to drobiazg: na stronie *Flota* zauważono inną ścieżkę obrazka Twitter (`og-img/og-1200x630.jpg` vs `og/translogix-og.jpg`). Prawdopodobnie chodzi o ten sam obraz, ale zapisany w innym miejscu. Warto ujednolicić, żeby nie było pomyłek w plikach – to jednak nie wpływa na SEO, tylko na pewność, że obraz się wczyta.

- **Opis alternatywny obrazów (alt text):** Opisy alternatywne wpływają zarówno na dostępność, jak i pozycjonowanie obrazów w Google. W kodzie **wszystkie** `<img>` **posiadają atrybut** `alt` – co już jest osiągnięciem, bo brak alt to częsty błąd. Co więcej, alt jest stosowany zgodnie z zasadami:
- Obrazy czysto dekoracyjne mają alt **pusty** (`alt=""`) i/lub są oznaczone `aria-hidden="true"`, np. ikonki dekoracyjne, tło logo (bo nazwa TransLogix jest obok jako tekst). Zgodnie z MDN, obrazy dekoracyjne powinny mieć pusty alt, żeby czytnik je pominął [7] – tak zrobiono dla ikon i ozdobników.

- Obrazy zawierające informację mają opis. Przykładowo, w sekcji floty ikonki pojazdów mają alt typu *"Bus dostawczy"*, *"Chłodnia solo 18 palet"*, co opisuje co jest na obrazku. W karcie floty nawet zrobiono alt *"Bus dostawczy – otwórz galerię"* na miniaturce, sygnalizując użytkownikowi, że kliknięcie otworzy galerię – to ciekawy zabieg, może nie standardowy, ale pomocny (choć tę informację mógłby też przekazać np. tytuł lightboxa czy dodatkowy tekst linku; nie jest to błąd).
- Logo firmy w nagłówku i stopce: zamiast alt z nazwą (który dublowałby tekst obok), użyto alt="" i dodano tekst w `<span class="brand__text">TransLogix</span>`. Dodatkowo w stopce link brand ma `aria-label="Strona główna TransLogix"`. To idealne rozwiązanie – użytkownik słyszy pełną nazwę i kontekst (strona główna), a nie "logo-translogix-light.svg" czy inne niepożądane komunikaty.
- Jedynym obrazem bez własnego alt jest zapewne duży obraz tła (jeśli jest w HTML) – ale wygląda, że tła są poprzez CSS, a nie `<img>`, więc to nie dotyczy alt.

Sumarycznie, *atributy alt zostały wykonane wzorowo*. To korzystne i dla użytkowników (opis grafiki w razie jej braku) i dla robotów (Google Images może zindeksować te obrazy z przypisanym tekstem).

- **Dane strukturalne (omówione szczegółowo poniżej)** również pośrednio wpływają na SEO – pomagają wyszukiwarce zrozumieć kontekst (np. że strona kontaktowa to ContactPage z informacjami firmy, czy że oferta to CollectionPage listująca usługi). Obecność ich jest plusem.

- **Wydajność i UX a SEO:** Choć to nie audyt wydajności, warto wspomnieć drobiazg: atrybut `loading="lazy"` dodany do wielu obrazów (np. flota) poprawia szybkość wczytywania, co Google także bierze pod uwagę (Core Web Vitals). To potwierdza, że autor myślał całościowo. Takie smaczki (lazy loading obrazów) korzystnie wpływają na **SEO techniczne**.

Z perspektywy SEO brakuje właściwie tylko jednego elementu, który jest często zalecany: **wyróżnienia aktualnej pozycji w menu**. Dodanie do bieżącej strony w menu np. `aria-current="page"` lub klasy `.active` nie wpływa bezpośrednio na ranking, ale poprawia **UX**, co pośrednio pomaga SEO (użytkownik wie, gdzie jest, mniejsze prawdopodobieństwo pogo-sticking). Obecnie np. będąc na *Usługi*, link *Usługi* w menu nie jest wyróżniony ani oznaczony. To łatwa zmiana – warto ją wdrożyć, by nawigacja była bardziej klarowna.

Podsumowując, **projekt jest przygotowany pod SEO bardzo profesjonalnie**. Ma wszystkie meta tagi, opisy, poprawną strukturę nagłówków, linki kanoniczne, dane strukturalne i przemyślane teksty alt. To poziom znacznie przewyższający typowy kod juniora – wskazuje raczej na **świadome podejście mid/senior** do kwestii optymalizacji pod wyszukiwarki.

Struktura danych JSON-LD (Schema.org)

Implementacja **danych strukturalnych** w formacie JSON-LD na każdej podstronie to kolejny mocny punkt projektu. Zastosowano różne typy Schema.org odpowiednio do kontekstu strony, co wzbogaca semantykę strony dla wyszukiwarek i asystentów (tzw. rich results). Poniżej przegląd i ocena każdego z nich:

- **Strona główna (`index.html`):** W head znajduje się skrypt JSON-LD opisujący obiekt `"@type": "Organization"` – czyli firmę TransLogix jako całość. Zawiera nazwę, URL, logo oraz opis działalności, a także strukturę adresu (`PostalAddress`) i punkt kontaktowy (`ContactPoint` z telefonem, mailem, typem kontaktu *customer service* i językami). Jest to de facto wizytówka firmy.

- **Zgodność:** Tego typu dane zwykle umieszcza się na stronie głównej lub stronie "O nas", więc lokalizacja jest odpowiednia.
- Wszystkie wymagane pola Organization zostały ujęte (name, url, address, contactPoint). W contactPoint podano `availableLanguage` – dodatkowa informacja, rzadziej spotykana, ale poprawna jeśli firma obsługuje wiele języków.

- Mała nieścisłość: w komentarzu HTML opisano ten skrypt jako *"JSON-LD – ContactPage"*, co może sugerować pomyłkę – w rzeczywistości to nie jest ContactPage (ten typ jest użyty na stronie kontakt), tylko **Organization schema**. To drobnostka, komentarz nie wpływa na działanie, ale może mylić innych deweloperów. Warto skorygować opis na "Organization". Sam JSON-LD jest poprawny.

- **Strona *Usługi* (`services.html`):** Użyto typu `"@type": "CollectionPage"` z nazwą "Usługi transportowe – TransLogix". CollectionPage to odpowiedni schemat dla strony będącej listą wielu elementów (tutaj lista ofert/usług). W JSON-LD zawarto:

- opis strony (description),
- isPartOf → WebSite TransLogix (powiązanie, że ta strona jest częścią witryny firmy),
- publisher → Organization TransLogix (informacja o wydawcy strony).
- **Ocena:** Struktura logiczna i kompletna. Dodatkowo type CollectionPage jest trafny – wyszukiwarki rozpoznają, że to strona z kolekcją ofert.
- Można by rozbudować to o listę elementów kolekcji (np. enumeracja usług), ale nie jest to wymagane i często się tego nie robi, bo szczegóły są na stronach indywidualnych usług.

- Punktacja za użycie isPartOf i publisher: to są zalecane pola łączące z główną witryną i organizacją – projekt je ma, czyli zna dobre praktyki Schema.org.

- **Strona *Flota* (`fleet.html`):** Również typ `"CollectionPage"`, analogicznie do usług. Zawiera name "Flota pojazdów – TransLogix", opis, isPartOf (WebSite TransLogix) i publisher (Organization TransLogix z logo). Wszystko spójne ze stroną usług – dobry duplikat schematu kolekcji dla innego działu. Meta dane (tytuł, opis) są odpowiednie do floty.

- Jedyna różnica: w definicji favicon i manifestu w head tej strony jest lekkie odstępstwo (o czym dalej w spójności meta), ale to nie dotyczy JSON-LD. Sam JSON-LD jest **poprawny i przydatny** – np. Google może dzięki temu rozpoznać stronę floty jako sekcję witryny.

- **Strona *Cennik* (`pricing.html`):** Tutaj zastosowano ogólny typ `"@type": "WebPage"`, nazwany "Cennik usług – TransLogix". To zrozumiałe, bo cennik to po prostu strona informacyjna, nie lista (choć zawiera listing cen, ale raczej statyczny) ani nie artykuł. WebPage jest domyślnym typem schematu strony. Zawiera opis strony i publisher (Organization TransLogix).

- **Ocena:** Poprawnie. Dodatkowo, w przypadku WebPage często dodaje się `@id` lub `mainEntityOfPage`, ale nie jest to obowiązkowe. Jest minimalnie mniej informacji niż w CollectionPage, ale na cenniku nie potrzeba isPartOf (choć można by dodać). Generalnie, definicja jest w porządku.

- Ewentualnie możnaby doprecyzować subType WebPage – np. *AboutPage*, *FAQPage* itp. Dla cennika nie ma dedykowanego typu, więc ogólny WebPage jest właściwy.

- **Strona *Kontakt* (`contact.html`):** Tutaj użyto typu `"@type": "ContactPage"`, co jest dokładnie tym, co powinna strona kontaktowa mieć. W JSON-LD widzimy:

- name “Kontakt – TransLogix”,
- description “Skontaktuj się z TransLogix – formularz kontaktowy i szybka wycena...”,
- url strony kontakt,
- publisher → Organization TransLogix (z logo i **w tym przypadku** zagnieżdżonym contactPoint – tu ciekawostka: w JSON-LD kontaktu `publisher` zawiera od razu `contactPoint` z danymi telefonu/email, czyli podobnie jak w Organization na stronie głównej). To trochę duplikacja – i tak już Organization na głównej ma te dane. Ale duplikacja w schema.org nie jest błędem; najwyżej mogło to być zorganizowane inaczej (np. ContactPage z mainEntityOfPage jako WebPage, ale obecna forma też jest akceptowalna).
- **Ocena:** ContactPage jest dobrym wyborem. Google lubi widzieć oznaczenie strony kontaktowej, może to przysłużyć się np. przy asystencie Google lub w Knowledge Panel (pokazuje link “Skontaktuj się”).

- Można by dodać `mainEntityOfPage: { "@type": "WebPage", "@id": ".../contact.html" }` zgodnie z niektórymi zaleceniami Google [8], ale brak tego nie powoduje błędu – obecna struktura jest czytelna i jednoznaczna.

- **Strona szczegółów usługi (`service.html`):** Zastosowano tu `"@type": "Service"` – to bardzo ciekawy i zaawansowany ruch. Strona `service.html` (ze szczegółami pojedynczej oferty transportowej) jest opisana nie jako strona, lecz bezpośrednio jako **usługa firmy**. W JSON-LD widzimy:

- name “Usługa transportowa – TransLogix” i opis (czyli ogólny opis strony usługi jako usługi),
- provider → Organization TransLogix (nazwa, strona, logo),
- areaServed: EU (działalność na terenie UE, fajny detal),
- availableChannel → ServiceChannel z numerem telefonu i URL do kontaktu.
- **Ocena:** To w zasadzie definiuje model oferty/serwisu świadczonego przez firmę, łącznie z tym, jak się skontaktować żeby z niej skorzystać. Dla wyszukiwarek może to być cenna informacja (np. w przyszłości Google może wyświetlać ofertę jako rich snippet usługi lokalnej).
- Ten JSON-LD nie zawiera informacji specyficznej dla konkretnej usługi (np. nazwy “Ekspres PL-DE” czy ceny), bo zapewne strona `service.html` obsługuje wiele usług dynamicznie. Tutaj być może przydałaby się generacja JSON-LD **dynamicznie** – np. po załadowaniu szczegółów usługi przez JS, można by uzupełnić dane (nazwa usługi, parametry, cena itp.) w strukturze Service. W obecnej formie jest to dość ogólny opis typu usługi. Niemniej jednak, już sama obecność typu Service i powiązanie z organizacją to ponadprzeciętna dbałość o **dane strukturalne**.
- Ponownie komentarz HTML mówi *“ServicePage”*, ale type to `Service` – drobna niezgodność w opisie, kod jest OK.

Ogólnie, **wszystkie skrypty JSON-LD są poprawnie sformatowane i osadzone**. Użyto unikalnych, adekwatnych typów schema.org tam, gdzie ma to sens, co jest oznaką dbałości o szczegóły. Dla pełnej zgodności z najlepszymi praktykami, można rozważyć: - Dodanie właściwości `mainEntityOfPage` do definicji Organization i ContactPage, wiążąc je z konkretnym WebPage (choć przy `isPartOf` i `publisher` można to uznać za opcjonalne). - Upewnienie się, że na stronie głównej znajduje się również definicja `"@type": "WebSite"` (ogólnie cała witryna). Czasami dodaje się JSON-LD:

```
{ "@context": "https://schema.org", "@type": "WebSite", "name":
"TransLogix", "url": "https://.../", "potentialAction": { "@type":
"SearchAction", ... } }
```

Tutaj nie ma definicji WebSite, ale pojawia się ona wewnątrz innych (isPartOf). To wystarczy – definicja WebSite implicitznie jest zawarta jako obiekt. Niemniej, jawna definicja WebSite (zwłaszcza jeśli jest wyszukiwarka na stronie, co tu nie dotyczy) bywa stosowana.

Podsumowując, **strukturę JSON-LD oceniam bardzo wysoko**. Wiele projektów w ogóle pomija dane strukturalne, a tu nie tylko je dodano, ale zrobiono to właściwie dla różnych typów stron. To zdecydowanie poziom *production-ready*, a nawet wykraczający przed to, bo implementacja danych strukturalnych często bywa "enterprise polish", czyli dodatkiem w dopieszczonych realizacjach.

Poprawność atrybutów HTML

W kodzie zwrócono uwagę na poprawne użycie atrybutów oraz ich zgodność ze specyfikacją. W dużej mierze atrybuty zostały użyte we właściwy sposób. Kilka kluczowych obserwacji:

- **Atrybut `lang` w `<html>`:** Ustawiono `lang="pl"` na elemencie `<html>`, co jest absolutnie wymagane dla poprawnej interpretacji języka przez przeglądarki i asystentów głosowych. Tutaj jest poprawnie (język polski). Nie ma wersji wielojęzycznej strony, więc nie ma atrybutów `hreflang` – nie są potrzebne.

- **Atrybuty ARIA:** Wspomnieliśmy już o wielu, ale reasumując ich poprawność:

- `aria-label` – użyty tam, gdzie trzeba zapewnić nazwę dla elementu bez tekstu (przyciski, linki z samą ikoną, kontenery filtrów). Wszystkie napotkane aria-label wyglądają sensownie (np. "Otwórz menu", "Przełącz tryb kolorów", nazwy social mediów).
- `aria-controls` – użyto poprawnie łącząc elementy (hamburger -> panel menu, akordeon -> panel FAQ, tab -> tabpanel).
- `aria-expanded` – inicjalizowane na false i zapewne zmieniane skryptem; nie zauważono miejsc, gdzie aria-expanded powinno być, a go brakuje (komponenty mają to na triggerach). Tylko po otwarciu modala lightbox można by dynamicznie dodać `aria-hidden="true"` reszcie strony lub manipulować atrybutami inert – ale to znowu zadanie dla JS poza statycznym HTML.
- `role` – zastosowano tylko tam, gdzie natywny element HTML nie istnieje lub jest niewystarczający:
    - role="dialog" na lightbox (bo to zwykły `<div>` pełniący rolę okna).
    - role="tabpanel/tablist" i "region" – bo wymagają tego komponenty budowane z div/button zamiast natywnych elementów.
    - Nie nadużyto ról typu `role="navigation"` czy `role="main"` na semantycznych elementach, co jest dobre (sam `<nav>` i `<main>` już to niosą).
    - `role="status"` na komunikacie sukcesu formularza (strona kontakt) – bardzo dobrze, bo role status/alert zapewniają, że sukces będzie ogłoszony przez czytnik jako komunikat. Tutaj dodatkowo hidden na tym divie jest do czasu pojawienia, co zapobiega czytaniu pustego elementu.
    - Sumarycznie ARIA została wykorzystana celowo i prawidłowo, bez widocznych błędów składni czy logiki.

- **Atrybuty dla formularzy:**

- `for` w label – jest, jak sprawdzono wcześniej.

- `id` i `name` w inputach – wszystkie pola formularza mają unikalne `id` i odpowiadający `name` (co istotne dla wysyłki, np. Netlify forms używa name).
- `required` – zastosowano sensownie do obowiązkowych pól, w tym do checkboxa akceptacji regulaminu.
- `type` – dobrany odpowiednio (email, number, tel, checkbox, select).

- `aria-describedby` – wskazuje id pustych elementów na błędy. Te id są unikalne i nie powtarzają się w obrębie strony. Uwaga: W formularzu kontakt, pole "Firma" ma `aria-describedby="company-error"`, choć nie jest required – czyli div na błąd istnieje mimo że pole opcjonalne. To nie przeszkadza; ewentualny błąd mógłby dotyczyć np. nieprawidłowego formatu (ale dla "Firma" raczej nie ma walidacji). Tak więc to przejaw ujednolicenia struktury formularza – nie wpływa negatywnie.

- **Atrybuty `alt` dla `<img>`:** Zostało to omówione – są obecne wszędzie. Alt pusty vs opisowy jest dobrany adekwatnie do kontekstu (zgodnie z wytycznymi [7]). Żadnego obrazka nie znaleziono bez alt – co jest zgodne z walidacją HTML (brak alt to błąd dla obrazka, chyba że role="presentation" – tu poszli drogą alt="").

- **Atrybuty linków i SEO:**

- `rel="noopener noreferrer"` dodano do wszystkich linków z `target="_blank"` (ikonki społecznościowe). To **ważne dla bezpieczeństwa i wydajności**, i jest często pomijane – tutaj wykonane prawidłowo.
- Linki wewnętrzne nie potrzebują rel i go nie mają, to ok.
- `tel:` i `mailto:` – w stopce kontaktowym `href="tel:+48533..."` i `href="mailto:..."` są poprawne. Dla tel dobrze użyto międzynarodowego formatu bez spacji, co zapewnia uniwersalność.

- W stopce wykryto jednak **błąd linku do FAQ**: odnośnik *FAQ* ma `href="#faq"` zamiast wskazywać na stronę kontaktu. Ponieważ sekcja FAQ jest na stronie kontakt, linki z innych podstron typu Start czy Usługi będą próbowały przewinąć do elementu o id `faq` na **tej samej stronie**, którego tam nie ma. To powoduje, że kliknięcie FAQ nic nie zrobi (poza dopisaniem `#faq` w URL). To **błąd** – należy zmienić ten link na `href="contact.html#faq"` aby zawsze przekierowywał na właściwą podstronę. To drobna korekta w HTML, ale istotna dla poprawnego działania na produkcji (inaczej użytkownicy poza stroną kontakt nie dostaną się do FAQ). Ten błąd zapewne przeoczono – reszta linków w stopce jest poprawna.

- **Konsystencja atrybutów meta i link:**

- W większości stron `<head>` powtarza te same meta tagi (charset, viewport, theme-color, color-scheme, manifest, icons). Natomiast zauważono niewielką niespójność:

    - Plik *fleet.html* i *service.html* używają innych ścieżek do faviconów/manifestu niż reszta. Np. `fleet.html` ma:

      ```
      <link rel="icon" type="image/x-icon" href="assets/favicons/
      favicon.ico" />
      <link rel="manifest" href="site.webmanifest" />
      ```

      podczas gdy inne strony mają:

```
<link rel="icon" type="image/png" href="/assets/icons/
favicon-96x96.png" sizes="96x96" />
<link rel="manifest" href="/assets/icons/site.webmanifest" />
```

Różnice:

- Ścieżki względne vs absolutne (z ukośnikiem na początku).
- Inna lokalizacja (`assets/icons/` vs `assets/favicons/`).
- Rodzaje favicon: na fleet.html użyto ico, 16x16,32x32 png i mask-icon Safari, a na innych tylko svg, ico i jeden png 96x96. To wskazuje, że część plików była tworzona w innym czasie lub przez inny generator.
- **Zalecenie:** Ujednolicić referencje do favicon i manifestu. W środowisku produkcji najlepiej upewnić się, że wszystkie strony ładują te same pliki (i że pliki rzeczywiście istnieją pod tymi ścieżkami). W przeciwnym razie, np. manifest może nie zostać znaleziony na podstronie (bo `site.webmanifest` jako ścieżka względna z `fleet.html` da URL `.../site.webmanifest`, zamiast `.../assets/favicons/site.webmanifest`). To kosmetyczne z punktu widzenia HTML semantyki, ale może wpływać na spójność doświadczenia (np. PWA).

- **Meta theme-color / color-scheme:** Ustawiono `color-scheme: light dark` i dynamiczne theme-color dla light/dark. To nowoczesne podejście zapewniające, że na urządzeniach mobilnych kolor paska adresu dostosuje się do trybu. Warto sprawdzić spójność – większość stron to ma, w *fleet.html* brakuje wersji `media="(prefers-color-scheme: light)"`/dark (jest tylko theme-color ogólny). To mały brak konsekwencji, choć nie krytyczny.

- **Walidacja HTML:** Nie przeprowadzaliśmy pełnej walidacji, ale pobieżny przegląd nie wykazał oczywistych błędów typu źle zagnieżdżone tagi czy brak zamknięcia. Struktura wydaje się poprawna. Atrybuty typu `viewport-fit=cover` w meta viewport to niestandardowe rozszerzenie dla iPhone X – dopuszczalne. `novalidate` przy formularzach jest celowe (wyłączenie domyślnej walidacji HTML na rzecz własnej).

- Jedna literówka w kodzie: w niektórych ścieżkach znalazł się podwójny ukośnik (np. `assets/img//logo/logo-translogix-light.svg` w index.html). Przeglądarki traktują `//` w ścieżce jako pojedynczy separator, więc obraz i tak się załaduje, ale warto poprawić te ścieżki dla schludności (może to być efekt łączenia ścieżek podczas generowania kodu). To kwestia porządku, nie wpływa na użytkownika w widoczny sposób.

- **Znaki diakrytyczne w treści:** Kod HTML ma deklarację UTF-8 i większość polskich znaków jest poprawnie wyświetlana (widzimy np. “Łączymy przewozy”, “Załadunek”, “pełną transparentność” – znaki ą, ł, ó są obecne). Jednak w sekcji FAQ tekst pytań/odpowiedzi nie zawiera polskich ogonków (np. jest *"Jak szybko mozecie..."* zamiast *"możecie"*, *"ladunek"* zamiast *"ładunek"*). To wygląda na niespójność wprowadzania treści – być może celowo pominięto diakryty ze względu na klawiaturę deweloperską lub kopiowano z innego źródła. **Z punktu widzenia HTML**, znaki są dopuszczalne ASCII, więc to nie łamie standardu. Natomiast **dla jakości projektu na produkcji** takie literówki są niedopuszczalne – należałoby uzupełnić polskie litery dla pełnego profesjonalizmu. Traktujemy to jako kosmetyk (nie wpływa na strukturę czy działanie strony, ale na postrzeganie przez użytkownika tak).

- **Identyfikatory i ich unikalność:** Id powtarzają się między stronami (np. zawsze jest `id="main"`, `id="mobile-nav"` na każdej podstronie), co jest naturalne – w obrębie jednej strony są unikalne, a poszczególne strony nie wczytują się jednocześnie. Warto zauważyć, że tam

gdzie w jednej stronie mogą być podobne komponenty, zadbano o unikalne id: np. w FAQ id pytań `faq1` - `faq6` są unikalne w ramach kontakt.html. W formularzach również każde id jest inne.

- Mały wyjątek to `id="serviceType"` pojawiające się w formularzu szybkiej wyceny i w formularzu kontakt – **na jednej stronie kontaktowej** mamy dwa pola o id="serviceType" (jedno w formularzu kontakt, drugie w osadzonej sekcji szybkiej wyceny, jeśli taka by tam była?). Trzeba to sprawdzić: na *contact.html* sekcja *Szybka wycena* jest raczej integralną częścią formularza kontakt (nie ma tam drugiego formularza kalkulatora, jak na stronie Cennik czy Start). Wydaje się, że na contact.html sekcja #quote to właśnie formularz kontakt (czyli nazewnictwo id "quote" jest tu mylące). W efekcie nie ma zduplikowanych elementów `serviceType` na jednej stronie. Jeśli jednak by kiedyś łączono komponent kalkulatora z formularzem kontakt na jednej stronie, należałoby zmienić np. id na unikalne. Obecnie wygląda na to, że nie dochodzi do konfliktu.

Reasumując, **atrybuty HTML zostały użyte prawidłowo i konsekwentnie**. Strona unika typowych błędów jak brak alt, niepowiązane label, niepoprawne aria, duplikaty ID czy niepoprawne typu input. Wszystko wskazuje na wysoki poziom staranności. Kilka drobiazgów do poprawy (niespójne linki, drobne duplikacje lub literówki) zostało wymienionych powyżej – to margines przy ogólnym obrazie zgodności ze standardami.

Spójność między podstronami

Ważnym aspektem gotowości produkcyjnej jest **spójność doświadczenia i kodu** na wszystkich podstronach serwisu. W projekcie TransLogix zadbano o to, by strony tworzyły jednolity system, choć znaleziono parę drobnych rozbieżności:

- **Spójność nagłówka i stopki:** Wszystkie podstrony korzystają z tego samego szablonu nagłówka (logo, menu, przyciski) oraz praktycznie identycznej stopki (sekcje: Oferta, Wsparcie, Dane firmy, Social media, dolny pasek prawny). Dzięki temu użytkownik czuje się na każdej podstronie tak samo – nie ma zmiany układu czy stylu. Również z perspektywy kodu jest to korzystne: powtarzalne elementy oznaczają, że prawdopodobnie zarządza tym jedna wspólna część (w projekcie statycznym może to być wynik powielenia kodu, ale i tak zachowano konsekwencję).
- Jednolitość obejmuje także obecność skip linka na wszystkich stronach (co jest spójne dla dostępności) oraz powtarzalne elementy jak przełącznik motywu.
- **Menu aktywne:** Jak wspomniano, brakuje oznaczenia aktywnej strony w menu (np. podświetlenie lub aria-current). To jedyny element spójności UX, który można dodać. Aktualnie użytkownik patrząc na menu nie widzi, na której stronie się znajduje, bo styl wszystkich linków jednakowy. Warto to ujednolicić w sensie wprowadzenia logiki aktywnego stanu. To małe usprawnienie.
- **Układ treści głównej:** Na każdej podstronie (oprócz dynamicznej service.html) sekcja otwierająca ma podobny układ: najpierw `<p class="eyebrow">` – krótki nadtytuł, potem główny nagłówek `<h1>`, potem `<p class="lead">` wprowadzający. To spójny schemat, nadający stronie rytm. Sekcje wewnątrz (oferta, flota, FAQ, cennik itp.) również stosują powtarzalną strukturę: kontener `.section__header` z `<p class="eyebrow">`, `<h2>`, `<p class="text-muted">` jako opis. Ta konsekwencja jest **bardzo dobra**, bo użytkownik szybko rozumie układ informacji na różnych stronach (np. zawsze sekcja zaczyna się małym nagłówkiem i tytułem).

- **Konsystencja meta-danych:** Omówione różnice w linkach do favicon/manifest to drobny problem spójności kodu (być może wynik użycia innego generatora na części stron). W produkcji powinno się dopilnować, by wszystkie strony wczytywały te same zasoby meta (favicony, manifest). Brak takiej spójności może skutkować np. inną ikonką na favikonie w zależności od podstrony, co jest niepożądane. Szczęśliwie, nazwy plików obrazków OG i Twitter są wszędzie takie same (poza flota, jak wspomniano), więc udostępnianie dowolnego linku powinno generować podobny podgląd.

- **Spójność danych kontaktowych:** W stopce dane firmy (adres, email, telefon) są identyczne na wszystkich stronach – co oczywiście jest konieczne, ale warto to odnotować jako prawidłowo zrobione. Niektóre strony (kontakt, index) dodatkowo prezentują kontakt w treści – i tam również te dane się pokrywają. To ważne dla wiarygodności (NAP – Name, Address, Phone – zgodny wszędzie). JSON-LD zresztą też powiela te informacje, więc jest dobrze.

- **Elementy interaktywne spójne:** Filtry i sortowanie na stronach *Usługi* i *Flota* są zrobione analogicznie (przyciski filtrów z klasą `.filter-chip`, aria-label na kontener), formularze szybkiej wyceny na index i pricing identyczne. To sugeruje, że komponenty były reużywane – co jest plusem.

- **Niespójność w nazewnictwie ID/odsyłaczy:** Pewnym zgrzytem jest wspomniane id="quote". Na stronie głównej i cennik jest sekcja szybkiej wyceny z id="quote". Na stronie kontakt id="quote" został użyty dla kontenera formularza kontaktowego (prawa kolumna). To nieco mylące – niby "szybka wycena" wypełnia się poprzez formularz kontaktowy. Funkcjonalnie może być ok, ale nazewnictwo mogłoby być lepsze (np. id="contact-form" dla formularza kontakt – zresztą taki id również jest nadany, a id="quote" jest na kontenerze z tym formularzem). Dla spójności semantycznej, rozważyłbym uniknięcie takich wieloznaczności – jednak to szczegół implementacyjny, raczej nie wpływający na użytkownika końcowego.

- **Wersja językowa i atrybuty globalne:** Wszystkie strony mają `lang="pl"`, co spójnie komunikuje język. Nie wprowadzono np. pojedynczej angielskiej strony bez zmiany lang (co czasem się zdarza i psuje spójność). Tutaj cały serwis jest wyraźnie PL.

Generalnie, projekt **prezentuje się jako spójny, jednolity produkt**. Użytkownik przechodząc między podstronami ma ciągłość nawigacji i wyglądu, a wyszukiwarki dostają konsekwentne informacje meta. Wskazane niespójności (favicon, link FAQ, drobiazgi w meta) są łatwe do poprawienia. Ich istnienie sygnalizuje, że integracja kodu mogła pochodzić z różnych etapów (np. początkowo inne favicony, potem zmieniono na svg, nie wszędzie zaktualizowano). Przed wdrożeniem produkcyjnym warto zrobić **przegląd finalny** tych elementów na wszystkich stronach, by 100% metadanych i linków było zunifikowanych.

Wnioski i rekomendacje

Na podstawie powyższego audytu można stwierdzić, że **projekt TransLogix jest bardzo blisko gotowości produkcyjnej pod kątem HTML**. Wiele elementów wykonano na poziomie profesjonalnym lub wręcz wykraczającym poza standardowe minimum. Poniżej podsumowanie mocnych stron oraz listę zaleceń podzielonych na kwestie krytyczne do poprawy i opcjonalne ulepszenia (*enterprise polish*):

Mocne strony projektu (poziom profesjonalny):

- **Semantyka HTML:** Poprawne użycie elementów strukturalnych (header, main, footer, section, article, nav). Struktura nagłówków logiczna, z jednym H1 na stronę i dobrym zachowaniem hierarchii. Kod jest czytelny i zorganizowany.
- **Dostępność (a11y):** Implementacja dostępności stoi na wysokim poziomie:
- Obecność skip linka do głównej treści [4] .
- Pełne i poprawne etykiety formularzy, opisy alternatywne obrazków [7] .
- Użycie ARIA do zapewnienia dostępności komponentów UI (nawigacja mobilna, karuzele tabów, akordeon FAQ, modal lightbox).
- Przemyślane opisy dla elementów interaktywnych (aria-label na przyciskach i linkach z ikonami).
- **SEO i metadane:**
- Każda podstrona ma unikalny i trafny `<title>` oraz meta description – gotowe do indeksacji.
- Zaimplementowano bogaty zestaw meta tagów (canonical, Open Graph, Twitter) co zapewni dobre udostępnianie w social media i brak duplikatów treści.
- Struktura treści sprzyja SEO (nagłówki z frazami, teksty opisowe, brak rażących błędów typu duplikaty H1).
- Lazy-loading obrazów i odpowiednie typy input to dodatkowe plusy techniczne.
- **Dane strukturalne (JSON-LD Schema.org):** Rzadko spotykana w takim stopniu implementacja:
- Użyto odpowiednich schematów (Organization, CollectionPage, ContactPage, Service, WebPage) tam gdzie to właściwe, co dostarcza wyszukiwarkom kontekstu o stronie [9] .
- Połączenie danych (publisher, isPartOf) wskazuje, że dane strukturalne zostały przemyślane całościowo.
- **Dbałość o detale front-end:** Projekt zawiera elementy świadczące o doświadczonym podejściu: kolorystyka dla trybu jasny/ciemny w meta, poprawne atrybuty `rel` dla linków, unikanie zbędnych atrybutów (np. brak `target="_blank"` tam gdzie niepotrzebne), wykorzystanie nowości (inputmode/tel, aria-live itp.). Kod HTML jest schludny, skomentowany w kluczowych miejscach (sekcje, wstawki JSON-LD) – co ułatwia utrzymanie.

Kwestie do poprawy przed wdrożeniem (błędy krytyczne):

1. **Poprawienie linku do sekcji FAQ w stopce:** Obecnie `href="#faq"` działa tylko na stronie kontakt, a na innych stronach jest linkiem donikąd. Należy zmienić na `href="contact.html#faq"` we wszystkich wystąpieniach w stopce, by zawsze prowadził do sekcji FAQ na stronie kontakt. To **istotna poprawka UX** – bez tego użytkownicy z innych podstron nie dotrą do FAQ.
2. **Zapewnienie treści statycznej lub prerender dla** `service.html`**:** Strona szczegółów usługi powinna mieć co najmniej własny tytuł (`<h1>`) i podstawowe dane w HTML, albo należy wdrożyć mechanizm prerenderowania dla botów. W obecnej formie jej SEO jest ograniczone (pusta treść). To **ważne**, jeśli ruch z Google na konkretne usługi jest pożądany.
3. **Ujednolicenie meta danych w** `<head>` **na wszystkich stronach:** Szczególnie dotyczy to linków do favicon i manifestu. Trzeba upewnić się, że każda strona odwołuje się do poprawnych ścieżek istniejących plików (preferowane jest używanie jednolitego zestawu ikon – obecnie rozbieżności między `/assets/icons/` a `/assets/favicons/`). Niespójność może skutkować drobnymi błędami (brak favikony na którejś stronie, niezaładowanie manifestu PWA). To **łatwe do poprawy** poprzez synchronizację sekcji `<head>` .
4. **Aria label dla modala (lightbox):** W dialogu lightbox warto dodać `aria-labelledby="..."` wskazujące na tytuł *"Galeria pojazdu"*. Bez tego użytkownik SR nie usłyszy tytułu okna dialogowego. Choć drobne, jest to **poprawka dostępności** – zalecana przed produkcją.
5. **Wyróżnienie bieżącej pozycji menu (ARIA/klasa aktywna):** Aby nawigacja była w pełni przyjazna i **intuicyjna**, dodaj atrybut `aria-current="page"` na link menu odpowiadający

otwartej stronie (oraz klasę CSS dla wizualnego podkreślenia). To uchroni przed dezorientacją użytkownika i jest standardem w dojrzałych stronach.

6. **Korekta tekstów z błędami znaków (polskie ogonki):** Choć to kwestia contentu, na produkcji absolutnie należy poprawić wszystkie wystąpienia typu "mozecie" -> "możecie" etc. Brak polskich znaków wygląda nieprofesjonalnie i może wpływać na wiarygodność strony w oczach klienta. To łatwa poprawka edycyjna.

Usprawnienia kosmetyczne i rekomendacje (tzw. *enterprise polish*):

- **Dodanie grupowania dla przycisków filtrów (ARIA):** Można opakować zbiory przycisków filtrów (na stronach Usługi i Flota) w element z `role="group"` i nadać mu aria-label (np. `<div role="group" aria-label="Filtruj usługi po kategorii">`). Obecnie jest tylko `aria-label` na divie bez roli. Dodanie roli spowoduje, że czytnik przedczyta nazwę grupy i liczbę elementów. To drobne ulepszenie ułatwiające zrozumienie, że przyciski stanowią zestaw opcji filtrujących.
- **Lepsze teksty linków "Szczegóły":** Rozważyć dodanie ukrytego tekstu w linkach szczegółów usługi, aby były unikalne. Np. `Szczegóły <span class="visually-hidden">usługi Ekspres PL-DE</span>`. Dzięki temu, w trybie listy linków, użytkownik widzi od razu, której oferty dotyczy dany link [10]. Aktualna implementacja nie jest zła, to jedynie poprawa UX dla niewielkiego procenta sytuacji.
- **Caption dla tabeli cennika:** Tabela cen ma aria-label="Pakiety cenowe", co jest poprawne. Alternatywnie (lub dodatkowo) można użyć `<caption>Pakiety cenowe</caption>` wewnątrz `<table>` dla natywnej semantyki tabeli. Caption jest czytane przez czytniki i prezentowane wizualnie (jeśli ostylowane). Aria-label załatwia sprawę, więc to czysto semantyczna kosmetyka.
- **Dodanie `scope` w tabeli cen:** Głowice tabeli (Pakiet, Co zawiera, Cena od) mogą mieć `scope="col"` w `<th>`, aby explicite zaznaczyć nagłówki kolumn. W małych tabelach nie jest to konieczne, bo i tak każdy <th> w thead domyślnie jest nagłówkiem kolumny. To drobny detal dostępnościowy (czytniki same to interpretują poprawnie, więc raczej kosmetyka).
- **Komentarze i nazewnictwo w JSON-LD:** Ujednolicić komentarze, by nie myliły typów (np. *ContactPage* vs *Organization* na index). Można też rozważyć przeniesienie powtarzalnego kodu Organization/publisher do jednego miejsca (redukcja duplikacji), aczkolwiek w JSON-LD powielenie nie szkodzi, a upraszcza pojedynczy snippet per strona.
- **Ewentualne dodanie schema dla WebSite i SearchAction:** Jeśli planowane jest dalsze wzbogacanie danych strukturalnych, można dodać definicję typu WebSite (ogólnie dla całej strony, z name i potencjalnie akcją wyszukiwania, jeśli byłaby wyszukiwarka). W kontekście firmy lokalnej można by też rozważyć schema LocalBusiness/Organization na stronie kontakt (z godzinami otwarcia, jeśli istotne). To jednak *nice-to-have*, obecny zakres JSON-LD jest już świetny.
- **Wydzielenie CSS klasy `.visually-hidden`:** Wspomniany pomysł na ukryte teksty wymaga klasy utility do ukrywania tekstu wizualnie (pozostawiając dla SR). W projekcie prawdopodobnie jest taka (skoro jest skip-link, pewnie ma zdefiniowane style). Jeśli nie, można dodać.
- **Walidacja W3C:** Po wprowadzeniu poprawek, warto przepuścić kod przez walidator W3C. Spodziewam się kilku ostrzeżeń raczej niż błędów (np. `viewport-fit` nieznane meta, czy duplikaty id o ile są). Dobrze jest osiągnąć czysty wynik walidacji dla perfekcyjnej oceny jakości.
- **Performance/Best Practices (wykracza poza HTML):** Choć nie dotyczy bezpośrednio HTML struktury – upewnij się, że atrybuty `alt` nie są zbyt długie (są ok), że elementy interaktywne mają wystarczający rozmiar klikalny itd. To już detale wykraczające poza audyt HTML, jednak patrząc globalnie warto je mieć na uwadze.

Ocena ogólna jakości HTML

Biorąc pod uwagę powyższe punkty, **jakość HTML projektu TransLogix oceniam bardzo wysoko**. Struktura i semantyka są zrealizowane na poziomie **production-ready**, z elementami świadczącymi o dojrzałości (ARIA, SEO, JSON-LD). Zidentyfikowane usterki są w większości drobne i łatwe do poprawienia – co jest naturalnym etapem przed finalnym wdrożeniem (żaden poważny projekt nie jest wolny od drobnych poprawek).

Jeśli przyjąć skalę developerską: - Kod zdecydowanie **nie nosi znamion pracy początkującego (juniora)** – brak tu typowych błędów jak nieprawidłowe nagłówki, brak alt, chaos w strukturze czy ignorowanie dostępności. - Zbliża się do poziomu **mid/senior**, a pod niektórymi względami (dane strukturalne, kompleksowe podejście do ARIA) ociera się o **“senior-ready”**. - Po wprowadzeniu rekomendowanych poprawek, można śmiało powiedzieć, że front-end HTML będzie **production-ready**, spełniając standardy jakości wymagane w profesjonalnych wdrożeniach.

Projekt wymaga jedynie niewielkiego *dopolerowania*, głównie w kategoriach spójności i drobnych poprawek dostępności/SEO. Krytycznych błędów wpływających na funkcjonalność czy indeksowanie jest bardzo niewiele (praktycznie tylko link FAQ). To znakomity wynik.

Końcowa konkluzja: HTML w projekcie TransLogix jest przygotowany z dużą starannością i zrozumieniem nowoczesnych standardów. Po zastosowaniu wskazanych poprawek serwis będzie w pełni gotowy do środowiska produkcyjnego, zarówno pod kątem technicznym, jak i jakościowym.

---

1 2 3 5 6 10 HTML: A good basis for accessibility - Learn web development | MDN
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Accessibility/HTML

4 7 What HTML features promote accessibility? - Learn web development | MDN
https://developer.mozilla.org/en-US/docs/Learn_web_development/Howto/Design_and_accessibility/HTML_features_for_accessibility

8 9 The right json schema markup for a contact page - Google Help
https://support.google.com/webmasters/thread/263524875