ChatGPT

Audyt JavaScript projektu TransLogix

accordion-faq.js

- **Poziom profesjonalny:** Kod akordeonu FAQ jest zwięzły i poprawny pod względem ARIA. Przy otwieraniu sekcji skrypt ustawia `aria-expanded="true"` na przycisku i usuwa atrybut `hidden` z panelu, a przy zamykaniu odwrotnie – dzięki temu użytkownicy czytników ekranu są informowani o stanie sekcji [1] [2]. Dodatkowo dodawana/usuwana jest klasa `.is-open` do stylowania wizualnego stanu.
- **Poziom profesjonalny:** Zaimplementowano płynne animacje wysokości panelu przy otwieraniu/zamykaniu. Wykorzystanie `panel.offsetHeight` (wymuszenie reflow) przed zmianą `panel.style.maxHeight` zapewnia płynne przejście z aktualnej wysokości do 0 [3]. To świadczy o dbałości o szczegóły animacji bez użycia ciężkich bibliotek.
- **Usprawnienie:** Brak obsługi klawiatury poza kliknięciem przycisku myszą. Jeśli `.accordion__button` jest elementem `<button>`, to Enter/Space zadziała domyślnie, co już zapewnia podstawową dostępność. Można jednak rozważyć dodanie obsługi strzałek do nawigacji między pytaniami FAQ lub zamykanie otwartej sekcji klawiszem Escape – nie jest to krytyczne, ale poprawiłoby dostępność.
- **Usprawnienie:** Kod nie przewiduje otwarcia więcej niż jednej sekcji jednocześnie (co wydaje się zamierzone – zawsze zamyka inne sekcje przy otwarciu nowej [4] [5]). Jeżeli wymagane byłoby jednoczesne otwarcie wielu FAQ, należałoby zmodyfikować logikę `closeAll()`. Obecne podejście jest jednak czytelne i odpowiednie dla typu treści FAQ.

aria-current.js

- **Poziom profesjonalny:** Funkcja poprawnie ustawia atrybut `aria-current="page"` dla bieżącej strony w menu nawigacji i stopce. Skrypt porównuje bieżący URL z `href` linków (uwzględniając także root `/` oraz `index.html` dla strony głównej) i dodaje atrybut do pasującego elementu [6]. Dzięki temu aktywna pozycja menu jest właściwie oznaczona dla dostępności, co jest dobrym standardem.
- **Usprawnienie:** Skrypt wybiera tylko pierwszy pasujący link (`matches[0]`) i ignoruje ewentualne kolejne [6]. Jeśli ten sam URL występuje w nawigacji i stopce jednocześnie, tylko pierwszy zostanie oznaczony `aria-current`. Zalecane byłoby oznaczanie **wszystkich** wystąpień bieżącej strony (np. zarówno w menu głównym, jak i stopce) – to drobna poprawka zwiększająca spójność.
- **Usprawnienie:** Porównywanie ścieżek mogłoby być bardziej niezawodne. Obecne podejście dzieli `pathname` po "/" i porównuje samą nazwę pliku lub pełną ścieżkę [7]. Warto upewnić się, że linki w HTML są spójne (np. zawsze względne bez `./`). Jeśli w przyszłości pojawią się inne formaty linków (np. z parametrami lub `#`), funkcja może wymagać rozbudowy o dodatkowe przypadki.

compact-header.js

- **Poziom profesjonalny:** Implementacja zwijania nagłówka przy przewijaniu jest prosta i wydajna. Skrypt sprawdza pozycję scrolla i dodaje/usuwa klasę `.header--compact` gdy użytkownik przewinie więcej niż 40px [8]. Stan jest przechowywany w zmiennej (`isCompact`), więc **klasa**

dodawana jest tylko raz przy przekroczeniu progu, co zapobiega powtarzającemu się manipulowaniu DOM przy każdym pikselu scrollu [9].

- **Poziom profesjonalny:** Zastosowano `passive: true` przy nasłuchiwaniu scrolla [10], co poprawia wydajność (przeglądarka nie blokuje przewijania). Skrypt wywołuje też `handleScroll()` raz na starcie [11], aby nagłówek był od razu w odpowiednim stanie, np. gdy użytkownik odświeży stronę w połowie.
- **Usprawnienie:** Stały próg `40px` mógłby być wyprowadzony do konfiguracji lub obliczany dynamicznie (np. na podstawie wysokości nagłówka). Obecne rozwiązanie działa, lecz przy zmianie designu (innej wysokości nagłówka) wymagałoby aktualizacji kodu.
- **Usprawnienie:** Skrypt nie usuwa nasłuchu przy demontażu komponentu, ale w typowej implementacji (kod ładowany raz na stronę) nie jest to problemem. W kontekście SPA warto byłoby dodać mechanizm usuwania listenera scroll przy odpinaniu komponentu, jednak dla strony B2B typu statycznego nie jest to wymagane.

form.js

- **Poziom profesjonalny:** Walidacja formularzy została zaimplementowana kompleksowo i zgodnie z typowymi wymaganiami. Dla pól wymaganych sprawdzana jest niepustość, dla e-maili format przez regex, dla telefonów dopuszczenie określonych znaków, a dla pól liczbowych czy wartość > 0 [12] [13]. Błędy są komunikowane użytkownikowi poprzez elementy powiązane atrybutem `aria-describedby` – tekst błędu jest wstawiany do odpowiedniego elementu, a pole otrzymuje klasę `.invalid` [14]. Takie podejście jest zgodne z dobrymi praktykami dostępności (wiąże komunikat z polem).
- **Poziom profesjonalny:** Obsługa zdarzenia `submit` zapobiega domyślnemu wysłaniu formularza i najpierw wykonuje pełną walidację wszystkich pól [15]. Dopiero jeśli **żadne pole nie zawiera błędu**, skrypt wywołuje akcję sukcesu: albo callback `onValid` (np. kalkulacja) albo faktyczne przesłanie na serwer (`form.submit()`) lub reset formularza i pokazanie komunikatu sukcesu [16] [17]. Takie rozróżnienie pozwala użyć jednej funkcji walidującej do różnych formularzy (np. szybka wycena vs. formularz kontaktowy) – to elastyczne i przemyślane rozwiązanie.
- **Profesjonalne aspekty a11y:** Po pomyślnym "submit" formularza kontaktowego, skrypt wykrywa parametr `?success=1` i automatycznie wyświetla komunikat potwierdzenia oraz ustawia na nim focus [18]. Dzięki temu użytkownik korzystający z klawiatury lub czytnika od razu otrzymuje informację zwrotną. Również w przypadku błędów, focus przekierowywany jest na pierwsze pole formularza [19], aby ułatwić poprawienie (choć tu można poprawić – patrz niżej).
- **Do refaktoru (średni):** Przy wykryciu błędów formularza focus jest ustawiany zawsze na **pierwsze pole** formularza [19], niezależnie od tego, które pole zawiera błąd. Lepszym podejściem byłoby przesunięcie focusu na **pierwsze błędne pole**, aby użytkownik nie musiał tabulować w poszukiwaniu komunikatu. To drobna zmiana zwiększająca użyteczność w przypadku dłuższych formularzy.
- **Do refaktoru (średni):** Funkcja `validateField` nie sprawdza poprawności opcjonalnych pól liczbowych, jeśli nie są oznaczone jako required [13]. W efekcie, pole typu number z niepoprawną wartością (np. tekstem) przejdzie walidację jeśli nie ma atrybutu `required`. Warto rozważyć walidację wpisanej wartości liczbowej również dla pól opcjonalnych – jeśli użytkownik coś wpisał, powinno to być poprawne liczbowo, albo pole powinno być puste.
- **Usprawnienie:** Walidacja wymaganego checkboxa (zgoda) jest zaimplementowana tylko przy próbie submit (`requiredCheckbox` w kodzie) [20]. Brakuje analogicznej obsługi na blur/zmianę, przez co dopóki użytkownik nie spróbuje wysłać formularza, nie zobaczy komunikatu o braku zaznaczenia zgody. Nie jest to błąd krytyczny, ale dla spójności można by dodać np. nasłuch na `change` checkboxa wymaganej zgody, który wywoła `showError`/`clearError`.

- **Potencjalne ryzyko:** Funkcje kalkulatora (quick quote, pricing) i historii wycen dynamicznie wstawiają treści HTML na podstawie danych z formularza. Np. elementy historii wycen są tworzone przez składanie stringu HTML z wartości użytkownika (dystans, waga, itp.) [21] . Choć wartości pochodzą z pól formularza liczbowych i predefiniowanych opcji (więc ryzyko XSS jest minimalne), to jednak bezpośrednie użycie `innerHTML` z danymi użytkownika **może stanowić wektor XSS**. Dla pełnego bezpieczeństwa, rozważyć wykorzystanie `textContent` dla tekstów lub sanityzację wartości (np. przez konwersję na liczbę, co zresztą częściowo ma już miejsce). W obecnym kodzie założenie o zaufanych danych wejściowych jest akceptowalne, ale z punktu widzenia **enterprise** warto unikać takich konstrukcji.
- **Usprawnienie (polish):** Kod jest dość rozbudowany – zawiera zarówno logikę walidacji, jak i kalkulacje biznesowe (stawki) oraz zarządzanie UI (historia wycen). Mimo że jest czytelnie podzielony na funkcje pomocnicze, można rozważyć dalszy podział na moduły (np. osobno moduł `validation` i moduł `calculator/history`). Obecnie nie ma to wpływu na działanie, jest to sugestia na przyszłość dla zachowania jeszcze lepszej organizacji kodu.

gallery-filters.js

- **Poziom profesjonalny:** Skrypt filtrowania galerii floty działa prostolinijnie – wykorzystuje atrybuty `data-type` i `data-purpose` przypisane do elementów, aby filtrować je na podstawie wybranej kategorii [22] . Kliknięcie w przycisk filtra (chip) ustawia odpowiednią klasę `.is-active` i pokazuje/ukrywa elementy przez `item.style.display` [23] [24] . Takie podejście jest proste w utrzymaniu i zrozumiałe.
- **Poziom profesjonalny:** Po zastosowaniu filtra wywoływana jest funkcja `updateFleetSingleState()` sprawdzająca, ile kart pozostało widocznych i dodająca klasę `.is-single` do kontenera, jeśli została jedna [25] . To sprytny zabieg – prawdopodobnie styluje inny layout lub rozmiar karty, gdy jest tylko jedna w wyniku filtrowania. Dbałość o taki detal zwiększa czytelność interfejsu dla użytkownika.
- **Usprawnienie:** W kodzie znajduje się komentarz wskazujący na miejsce potencjalnej zmiany (`// ← TU` przy wywołaniu `updateFleetSingleState` w linii 20) [26] . Taki pozostawiony komentarz wygląda na tymczasowy znacznik użyty podczas debugowania. W kodzie produkcyjnym warto usunąć takie adnotacje.
- **Usprawnienie (dostępność):** Należy upewnić się, że elementy filtrujące (tzw. *chips*) są elementami interaktywnymi – np. `<button>` lub `<a>` – albo mają odpowiednie role i tabindex. Sam skrypt używa `addEventListener("click", ...)`, co zadziała nawet na `<div>`, ale wtedy elementy te nie są dostępne z klawiatury. Jeśli chipsy są `<button class="filter-chip">`, to jest w porządku. W przeciwnym razie (np. `<span>`) – warto dodać `tabindex="0"` i rolę `button`, aby użytkownicy klawiatury mogli z nich skorzystać.
- **Usprawnienie:** Po inicjalizacji od razu wywoływane jest `applyFilters("all")` [27] , dzięki czemu startowo pokazane są wszystkie elementy. Jednak kod nie wymusza podświetlenia domyślnego filtra "Wszystkie". Jeżeli HTML nie ustawia wstępnie `.is-active` na filtrze "all", to warto by to zrobić w JS (np. dodać `.is-active` dla pierwszego chipa na początku), aby stan UI był spójny (aktualnie wszystkie elementy są widoczne, ale żaden filtr nie zaznaczony jako aktywny, o ile nie zrobiono tego statycznie w HTML).

lightbox.js

- **Poziom profesjonalny:** Moduł lightbox jest rozbudowany i świadczy o wysokiej dbałości o szczegóły. Zaimplementowano blokadę scrolla tła po otwarciu modalu – ciało strony jest ustawiane na `position: fixed` z zachowaniem poprzedniego scrolla, dodawana jest klasa

`.no-scroll` do body [28], a po zamknięciu scroll przywracany [29]. Dzięki temu podczas przeglądania galerii strona w tle nie "przeskakuje".

- **Poziom profesjonalny:** Zapewniono **zarządzanie focusem** i obsługę klawiatury wewnątrz lightboxu. Przykłady:
- Po otwarciu focus jest przenoszony na przycisk zamknięcia lightboxu [30].
- Klawisz Escape zamyka albo powiększenie (zoom), albo cały lightbox (jeśli zoom nie jest aktywny) [31].
- Strzałki lewo/prawo zmieniają kategorię galerii [32].
- Tab jest obsłużony tak, by zogniskować focus w obrębie okna (tzw. *focus trap* dla modalu) – zarówno dla głównego dialogu [33], jak i dla trybu powiększenia zdjęcia [34]. To wszystko sprawia, że komponent jest w dużej mierze przyjazny dla użytkowników klawiatury.
- **Poziom profesjonalny:** Tryb powiększenia (zoom) zdjęcia został przewidziany i zrealizowany z użyciem oddzielnego overlay'a. Kliknięcie (a dokładniej – **podwójne kliknięcie/dotknięcie**) miniatury otwiera powiększony obraz w pełnym oknie [35] [36]. W tym stanie tło galerii ma `aria-hidden="true"`, a warstwa powiększenia `aria-hidden="false"`, co zapobiega czytaniu przez screen reader zbędnych elementów [37]. Użytkownik może zamknąć powiększenie klawiszem Escape lub klikając poza obrazem, a focus wraca wtedy do ostatniego elementu, który wywołał powiększenie [38] – to wzorowe zachowanie modalu.
- **Do refaktoru (średni):** Obsługa zmiany kategorii wykorzystuje przyciski "Prev"/"Next", ale brakuje możliwości zmiany aktualnie wyświetlanego zdjęcia w ramach tej samej kategorii. Miniatury są renderowane, jednak **brak obsługi kliknięcia pojedynczego** w miniaturę, aby zmienić zdjęcie główne. Użytkownik może jedynie obejrzeć miniatury lub dwuklikiem je powiększyć, ale nie przełączyć podglądu głównego na inne ujęcie. Być może to zamierzony minimalizm, ale z perspektywy UX warto dodać: pojedyncze kliknięcie miniatury ustawia `heroImg.src` na wybrane zdjęcie (lub umożliwić nawigację strzałkami góra/dół między miniaturami).
- **Uwaga (dostępność):** Powyższy brak wpływa też na dostępność – miniatury nie są elementami focusowalnymi (najprawdopodobniej to same obrazki `<img>` w kontenerach). Użytkownik klawiatury nie ma możliwości wybrać konkretnego obrazka do powiększenia (bo aby powiększyć, musi zrobić dblclick myszą). Rozwiązaniem mogłoby być opakowanie miniatur w przyciski (`<button>`) lub linki z `href`, co uczyni je dostępne dla klawiatury i pozwoli np. Enterem otworzyć powiększenie. Obecny stan to ograniczenie raczej **średniej wagi** – podstawowe funkcje działają, ale nie w pełni wykorzystano potencjał dostępności.
- **Usprawnienie:** Skrypt globalnie nasłuchuje `keydown` na dokumencie, aby reagować na Escape/strzałki nawet gdy focus jest na obrazku [31]. To poprawne, ale warto pamiętać o posprzątaniu tego nasłuchu przy ewentualnym odpinaniu komponentu. W typowej stronie multipage nie jest to problem (nasłuch istnieje tylko na tej podstronie galerii). W przypadku reużycia modułu lub SPA, można by dodać mechanizm czyszczenia eventów po zamknięciu lightboxu.
- **Usprawnienie:** Kod wykorzystuje podwójne zdarzenie do wykrycia podwójnego tapnięcia (mobilnego) i podwójnego kliknięcia (desktop). Istnieje teoretyczna możliwość, że na urządzeniach desktop obsługa `dblclick` i jednocześnie logika `pointerdown` mogą zadziałać oba, wywołując `openZoom` dwa razy. To raczej niszowy przypadek i przeglądarki najpewniej zapobiegną podwójnemu wykonaniu, ale zaznaczamy to jako detal do ewentualnego przetestowania. Ogólnie jednak modul lightbox jest napisany solidnie i z uwzględnieniem wielu scenariuszy.

nav.js

- **Poziom profesjonalny:** Mobilne menu nawigacji zostało zaimplementowane zgodnie z najlepszymi praktykami. Przycisk menu (hamburger) kontroluje widoczność panelu `.nav__panel` poprzez atrybut `aria-expanded` (false/true) i `aria-label` zmieniany

dynamicznie z "Otwórz menu" na "Zamknij menu" [39] [40]. Panel nawigacji otrzymuje `aria-hidden` gdy jest zamknięty, a także ma nadany rola `navigation` i własny aria-label ("Główne menu") [41]. To zapewnia, że nawigacja jest właściwie oznaczona dla technik asystujących.

- **Poziom profesjonalny:** Skrypt przewiduje różne scenariusze zamknięcia menu:
- Kliknięcie w link wewnątrz panelu automatycznie zamyka menu (tylko na mobile – sprawdzane poprzez stan `aria-expanded`) [42].
- Kliknięcie poza obszarem menu również je zamknie [43].
- Naciśnięcie Escape zamknie menu i ustawi focus z powrotem na toggle (hamburger) [44]. To wszystko sprawia, że menu nie zostanie przypadkowo pozostawione otwarte i jest przyjazne w obsłudze (np. użytkownik szybko zamknie menu klawiszem Esc).
- **Poziom profesjonalny:** Dobrze rozwiązano dostosowanie do zmiany rozmiaru okna. Użyto `matchMedia("(min-width: 900px)")` i nasłuchiwania na zmianę (`mq.addEventListener("change", ...)`), aby przy przejściu na widok desktop automatycznie zdjąć `.nav__panel--open`, ustawić `aria-hidden=false` na panel (bo na desktopie menu jest po prostu widoczne) oraz usunąć klasę blokującą scroll na body [45] [46]. Zapewnia to spójny stan UI przy zmianie orientacji urządzenia lub rozmiaru okna przeglądarki – użytkownik nie zostanie z “ukrytym” menu na desktopie albo scroll zablokowanym po zamknięciu mobilnego menu.
- **Usprawnienie (dostępność):** Można rozważyć dodanie tzw. *focus trap* przy otwartym menu mobilnym, podobnie jak zrobiono to w lightbox. Obecnie focus po otwarciu pozostaje na przycisku toggle (lub przechodzi na pierwszy link, jeśli użytkownik naciska Tab). Użytkownik może wprawdzie przejść tabulatorem przez linki menu, ale jeśli przypadkiem wyjdzie focusem poza nie, to może trafić na elementy tła strony mimo zasłoniętego ekranu. Nie jest to bardzo krytyczne (mało prawdopodobne przypadkowe wyjście, a ekran tła i tak nieprzewijalny), jednak z punktu widzenia pełnej dostępności można by ograniczyć focus w obrębie otwartego menu.
- **Usprawnienie:** Podobnie jak w lightbox, tutaj także użyto klasy `.no-scroll` do blokady scrolla mobilnego zamiast pełnego `position: fixed`. Warto upewnić się, że to pokrywa wszystkie mobilne scenariusze (na większości urządzeń mobilnych `overflow: hidden` na body poprzez `.no-scroll` wystarczy). Ewentualne problemy (np. na iOS) rozwiązano już w lightbox przez `position: fixed`; można rozważyć ujednolicenie podejścia, choć jeśli nie ma błędów przewijania tła przy otwartym menu, to obecne rozwiązanie jest wystarczające.
- **Ogólnie:** Kod nawigacji jest czytelny, dobrze zoptymalizowany pod kątem UX i a11y. Rozdział odpowiedzialności jest właściwy – komponent menu nie zależy nadmiernie od innych modułów, współdzieli jedynie pewną konwencję (np. wykorzystanie tej samej klasy `.no-scroll` na body co lightbox, co jest pozytywne – uniknięto duplikowania różnych klas do blokady scrolla).

reveal.js

- **Poziom profesjonalny:** Wykrywanie i animacja pojawiania się sekcji na scroll zrealizowane jest za pomocą **IntersectionObserver**, co jest nowoczesnym i wydajnym podejściem [47]. Ustawiono sensowny threshold (0.15, czyli 15% pojawienia się elementu w viewport), po przekroczeniu którego klasa `.is-visible` jest dodawana i obserwacja danego elementu wyłączana (unobserve) [48]. Dzięki temu animacja uruchamia się tylko raz per element i nie obciąża niepotrzebnie głównego wątku po wykonaniu.
- **Poziom profesjonalny:** Zadbano o preferencje użytkownika dotyczące ruchu. Jeśli ustawiona jest preferencja `prefers-reduced-motion: reduce`, skrypt od razu dodaje wszystkim elementom klasę `.is-visible` i nie inicjuje IntersectionObserver [49]. Unika to animowania elementów dla osób, które tego nie życzą – duży plus za dostępność.
- **Usprawnienie:** Po ujawnieniu wszystkich elementów obserwator w zasadzie mógłby zostać odłączony (disconnect). Wprawdzie w kodzie po dodaniu klasy następuje `observer.unobserve(entry.target)`, więc z czasem nie obserwuje już nic, ale sam obiekt

`IntersectionObserver` wisi w pamięci do końca sesji. To drobiazg (garbage collector i tak go sprzątnie przy odświeżeniu strony), ale jako „polish” można by dodać np. licznik i `observer.disconnect()` gdy wszystkie elementy zostaną ujawnione.

- **Uwaga:** Użycie IntersectionObserver sprawia, że ta funkcjonalność nie zadziała w bardzo starych przeglądarkach (IE11 i starsze) bez polyfillu. W kontekście nowoczesnej strony B2B to akceptowalne założenie. Ewentualnie polyfill mógłby być warunkowo dołączony dla pełnej zgodności, ale prawdopodobnie nie jest to wymagane przez docelowych użytkowników.

service-detail.js

- **Poziom profesjonalny:** Moduł dynamicznie generuje stronę szczegółów usługi na podstawie danych w formacie JSON. Wykorzystuje `fetch` do pobrania `services.json` i wybiera właściwy obiekt na podstawie parametru w URL (`service` slug lub numeryczne `id`) [50]. To nowoczesne podejście, pozwalające zarządzać treścią poprzez dane zamiast duplikowania wielu plików HTML.
- **Poziom profesjonalny:** Zadbano o przyjazny komunikat, gdy żądana usługa nie zostanie znaleziona – wstawiany jest komunikat z linkiem powrotu do listy usług [51]. To lepsze niż pusty ekran. Ponadto tytuł strony jest aktualizowany dynamicznie na nazwę usługi [52], co poprawia SEO i użyteczność (użytkownik widzi w karcie przeglądarki odpowiednią nazwę).
- **Poziom profesjonalny:** Wygenerowany HTML jest semantyczny i zgodny z resztą strony. Na górze tworzona jest nawigacja okruszkowa `<nav aria-label="Okruszki">` z linkiem powrotu i oznaczeniem bieżącej pozycji `aria-current="page"` [53]. Dane usługi wstawiane są do elementów jak `<h1>` (nazwa), akapit `.lead` (opis), listy `<ul>` (korzyści) itp., dzięki czemu wygenerowana treść zachowuje strukturę tak dobrą, jak statyczny HTML.
- **Do refaktoru (średni):** Główne zastrzeżenie to użycie `innerHTML` do generowania dużych fragmentów UI z danymi bez żadnej filtracji/escapowania. W szablonie wstawiane są wartości jak `service.name`, `service.route` itp. bezpośrednio [54]. Zakładamy, że dane w `services.json` są zaufane (pochodzą od dewelopera). Jeśli jednak choć jedna wartość zawierałaby niespodziewany znak HTML (np. `"` w nazwie usługi, `<` w opisie), może to spowodować błędy wyświetlania, a w skrajnym przypadku potencjalny XSS. Przykładowo, wartość z cudzysłowem mogłaby przerwać atrybut `alt` obrazka [54]. Wrażliwość na takie dane jest **ryzykiem produkcyjnym** – niewielkim, jeśli dane są kontrolowane, ale wartym odnotowania. Dla bezpieczeństwa na poziomie “enterprise” zaleca się sanitizację danych przed wstawieniem (np. encja " zamiast `&quot;` w alt). Alternatywnie można budować elementy DOM metodami createElement + textContent, co eliminuje ryzyko interpretacji HTML.
- **Do refaktoru (średni):** Brak obsługi błędów sieci/przetwarzania JSON. Jeśli plik `services.json` nie załaduje się (błąd sieci) lub zwróci niepoprawny JSON, funkcja `initServiceDetail()` zakończy się wyjątkiem (unhandled Promise). Warto opakować fetch w blok try/catch i w razie błędu wyświetlić komunikat (np. *“Nie udało się załadować danych usługi”*). To ważne dla gotowości produkcyjnej – aplikacja powinna łaskawie obsłużyć sytuacje awaryjne zamiast milczeć.
- **Usprawnienie:** Gdy wywołano stronę service.html bez żadnych parametrów `?service=` lub `?id=`, obecny kod potraktuje `idParam` jako `NaN` (Number(null) -> 0) i najpewniej wyświetli *“Nie znaleziono usługi”*. Rozważanym usprawnieniem mogłoby być np. automatyczne przekierowanie na `services.html` w takiej sytuacji (skoro nie wskazano konkretnej usługi), ale to kwestia UX. Obecne zachowanie nie jest błędem – informuje poprawnie, że brak danych.
- **Ogólnie:** Kod jest czytelny i dobrze zorganizowany – użyto async/await, warunku wyjścia gdy nie ma kontenera (co pozwala bezpiecznie ładować skrypt na innych podstronach, gdzie nie zadziała), a także sprytnego mechanizmu wyboru usługi po slug **lub** id. Należy jedynie pamiętać

o wspomnianym bezpieczeństwie przy renderowaniu HTML z danych. Poza tym moduł ten znacząco ułatwia utrzymanie contentu usług w projekcie.

services-filters.js

- **Poziom profesjonalny:** Moduł listy usług z filtrami prezentuje dobre podejście do interaktywności bez frameworka. Dane usług są ładowane z tego samego `services.json` (zapewniona spójność z `service-detail.js`). Zastosowano *debouncing stanu* przez zapisywanie filtrów w `sessionStorage` – dzięki temu użytkownik powracający do listy nie traci swoich ustawień (filtr, zakres ceny itp. są wczytywane przez `loadState()` przy inicjalizacji) [55] [56]. To miły akcent "stateful UI" poprawiający UX.
- **Poziom profesjonalny:** Filtrowanie i sortowanie jest zaimplementowane wydajnie, operując na tablicy w pamięci. Funkcja `filterServices` stosuje kolejno wybrany filtr kategorii/typu, filtr ceny maksymalnej oraz sortowanie po cenie lub czasie [57] [58]. Wszystkie te operacje działają na kopii tablicy (`result = [...services]`), co zapobiega mutacji oryginalnych danych. Dzięki temu ponowne zmiany filtrów zawsze zaczynają od pełnej listy – prawidłowe podejście.
- **Poziom profesjonalny:** Integracja z UI jest dopracowana:
- Kliknięcie filtra (chip) ustawia stan i atrybuty ARIA (`aria-pressed`) dla przycisków, tak by aktualnie wybrany filtr był oznaczony semantycznie [59].
- Suwak ceny natychmiast aktualizuje wynik przy każdym przesunięciu (`input` event) i również pokazuje bieżącą wartość w sąsiednim elemencie tekstowym (wartość ceny) – to zapewnia użytkownikowi natychmiastową informację o kryterium filtrowania.
- Zmiana selektora sortowania odświeża listę i utrzymuje stan poprzez `sortSelect.value = state.sort` na starcie [60].
- Liczba wyników jest wyświetlana dynamicznie wraz z filtrowaniem [61], a także skrypt modyfikuje parametr `?filter=` w URL (za pomocą `history.replaceState`) aby można było łatwo udostępnić link do przefiltrowanej listy [62]. To poziom dopracowania wykraczający poza standard – **bardzo pozytywny aspekt**.
- **Do refaktoru (średni):** Podobnie jak w module service-detail, generowanie listy usług odbywa się przez składanie dużego stringu HTML z wykorzystaniem `innerHTML` [63]. Wartości takie jak `service.name`, `service.route` czy fragment opisu trafiają bezpośrednio do kodu. Jeśli którakolwiek z tych wartości zawierałaby niebezpieczne znaki, mogłoby to prowadzić do problemów z wyświetleniem lub potencjalnie XSS. Ryzyko znów jest niewielkie (dane z pliku są raczej statyczne i kontrolowane), ale z punktu widzenia bezpieczeństwa produkcyjnego, dobrze byłoby rozważyć użycie metod bezpiecznych (tworzenie elementów przez DOM API). Na przykład opis mógłby być dodany przez `textContent`, a nie przez string HTML, jeśli nie ma potrzeby formatowania HTML w nim.
- **Do refaktoru (średni):** Brak obsługi błędu ładowania JSON – analogicznie jak wyżej, warto dodać try/catch wokół `fetch`, aby w razie problemów z danymi aplikacja informowała użytkownika zamiast pozostawiać pustą listę.
- **Usprawnienie:** Kod inicjalizuje zmienną `state` łącząc domyślne wartości filtrowania z tymi z `sessionStorage` [56]. Domyślna maksymalna cena ustawiona jest na 10000 (lub wartość z suwaka w HTML). Jeśli w przyszłości pojawią się usługi z ceną powyżej tego pułapu, filtr może je niepotrzebnie ukryć. Warto upewnić się, że 10000 pokrywa rzeczywisty zakres cen (prawdopodobnie tak). Ewentualnie można dynamicznie ustawiać max zakresu na podstawie danych (np. `Math.max` z cen).
- **Usprawnienie:** W kodzie na końcu pliku znajduje się fragment, który niezależnie aktualizuje wyświetlaną wartość suwaka cen (`priceValue`) przy poruszaniu suwakiem [64]. Jest on poza funkcją `initServicesFilters`, więc wykona się od razu przy załadowaniu modułu. Działa to poprawnie, ale dla klarowności struktury można by ten fragment również umieścić wewnątrz

`initServicesFilters` lub przynajmniej opatrzyć komentarzem, że dotyczy on jedynie UI (aktualizacja etykiety ceny). To drobna kwestia stylu – nie wpływa na działanie, bo zmienne `priceRange` i `priceValue` są dostępne w zakresie globalnym modułu.

- **Ogólnie:** Funkcjonalność filtrowania usług jest **dobrze zaprojektowana i wykonana**. Aplikacja pamięta wybory użytkownika, jest interaktywna i responsywna. Kod mógłby być nieco zabezpieczony na wypadek nieprzewidzianych danych, ale w zakładanym środowisku (dane statyczne od dewelopera) sprawuje się bez zarzutu.

Ocena końcowa warstwy JS

Warstwa JavaScript projektu TransLogix prezentuje **wysoki poziom wykonania**. Kod jest podzielony na moduły odpowiadające poszczególnym funkcjonalnościom strony, co świadczy o przemyślanej architekturze. Większość rozwiązań – od nawigacji mobilnej, przez akordeon FAQ, po zaawansowany lightbox – została wykonana profesjonalnie z dbałością o dostępność (ARIA, focus management) i wydajność (wydarzenia pasywne, IntersectionObserver, minimalna ingerencja w DOM).

Nie stwierdzono krytycznych błędów, które uniemożliwiałyby wdrożenie produkcyjne. Zidentyfikowane problemy mają charakter **średni lub kosmetyczny** – głównie dotyczą potencjalnych usprawnień: dodatkowe scenariusze dostępności (np. obsługa klawiatury dla galerii), pełniejsze zabezpieczenia XSS przy generowaniu HTML z danych oraz drobne ulepszenia UX. Wprowadzenie tych poprawek podniosłoby kod do poziomu **enterprise-grade**, jednak już w obecnej postaci jest on solidny.

Końcowa ocena: warstwa JS jest w znacznej mierze *production-ready*. Kod wygląda na tworzony przez doświadczonego developera – oceniam go na poziomie **mid+/senior**. Bez obaw można wdrażać go na produkcję, pamiętając o ewentualnych drobnych poprawkach, które dodatkowo podniosą jakość i trwałość rozwiązania.

---

1 2 3 4 5 accordion-faq.js
file://file_00000000a92871f5abf66d4e8d55e23c

6 7 aria-current.js
file://file_000000005820720eb02b7be09402eb89

8 9 10 11 compact-header.js
file://file_00000000fa5c720e989f44a8363676e5

12 13 14 15 16 17 18 19 20 21 form.js
file://file_0000000004e471f5a39dc841f9d4d76f

22 23 24 25 26 27 gallery-filters.js
file://file_000000005c48720eabc0391a1f0e80ec

28 29 30 31 32 33 34 35 36 37 38 lightbox.js
file://file_00000000a58871f59ee9d883ec903363

39 40 41 42 43 44 45 46 nav.js
file://file_00000000c4b0720e8a446ac93407ef93

47 48 49 reveal.js
file://file_0000000040f871f58f404fe8b2992292

50 51 52 53 54 service-detail.js
file://file_0000000016f071f5b05bb72c2292118c

55 56 57 58 59 60 61 62 63 64 services-filters.js

file://file_000000003d84720e9ca4ca5b29a08f92