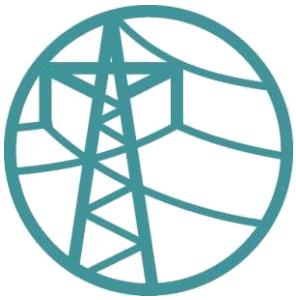


WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Krzysztof Kołodziej

Różne frameworki do budowy UI na podstawie tego samego API

Projekt inżynierski

Opiekun pracy:
dr inż. Tomasz Rak

Rzeszów, rok 2024

Spis treści

1. Wstęp

W świecie nowoczesnego front-endu istnieje wiele frameworków i bibliotek, które umożliwiają tworzenie interfejsów użytkownika (UI). Każde z tych narzędzi ma swoje mocne i słabe strony, a także specyficzne zastosowania. Wybór odpowiedniego narzędzia zależy od rodzaju projektu, wymagań biznesowych oraz preferencji zespołu deweloperskiego. Choć wiele rozwiązań oferuje podobne funkcjonalności, różnią się one podejściem do organizacji kodu, skalowalnością, filozofią projektowania aplikacji, wsparciem społeczności, czy też jakością dokumentacji.

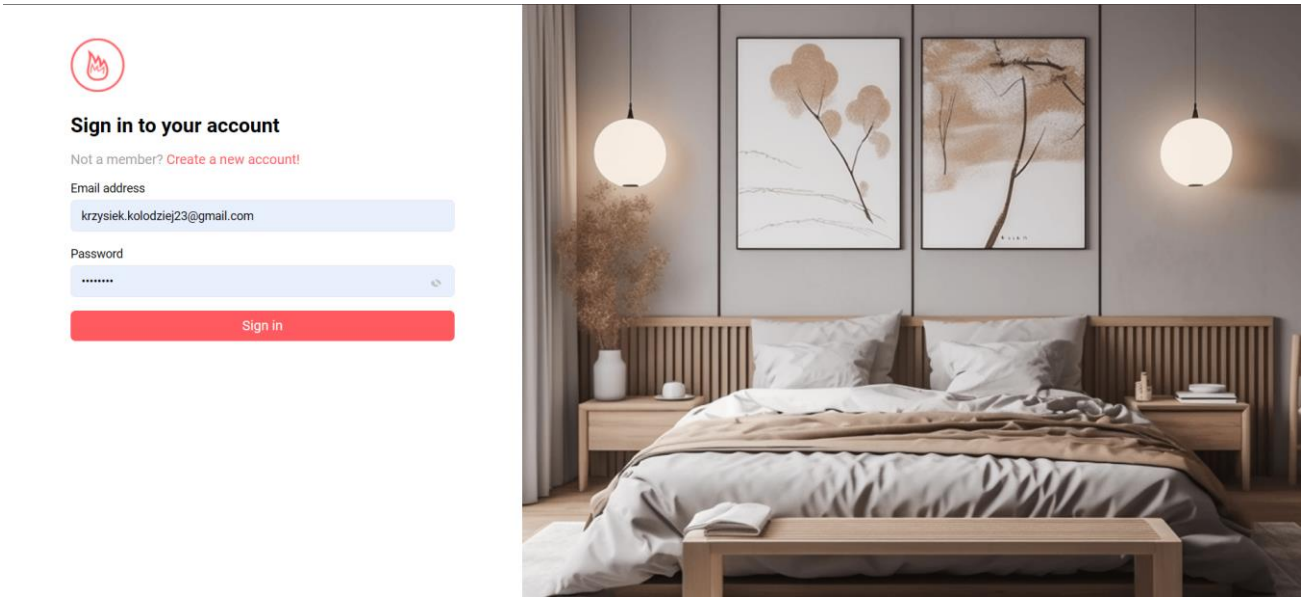
W ramach tego projektu powstanie aplikacja zbudowana zarówno w React, jak i w Vue. Dzięki temu będzie można zweryfikować popularne opinie na temat tych technologii i sprawdzić, jak wypadają w praktyce.

2. Cel pracy

Celem pracy jest analiza, jak różne narzędzia do tworzenia UI, takie jak React, Angular, Vue.js, mogą być wykorzystane do budowy funkcjonalnych i estetycznych interfejsów użytkownika, korzystając z identycznych danych i usług dostarczanych przez wspólne API. Praca przedstawiać będzie szczegółowy przegląd wybranych frameworków UI, omawiając ich architekturę, modele danych, sposoby renderowania oraz zarzadzanie stanem. Kluczowym elementem pracy jest implementacja przykładowego projektu – aplikacji webowej, w której ten sam backend (Docker) oparty na API jest wykorzystywany do budowy kilku różnych interfejsów użytkownika przy użyciu różnych frameworków. Praca ta analizuje i porównuje wydajność, skalowalność, łatwość utrzymania i inne kluczowe aspekty każdego z zastosowanych rozwiązań. Projekt oczywiście można stworzyć przy użyciu samego JavaScriptu, natomiast stosowanie frameworków znacząco wpływa na przyspieszenie rozwoju, skalowalność, bezpieczeństwo oraz wiele innych ważnych czynników.

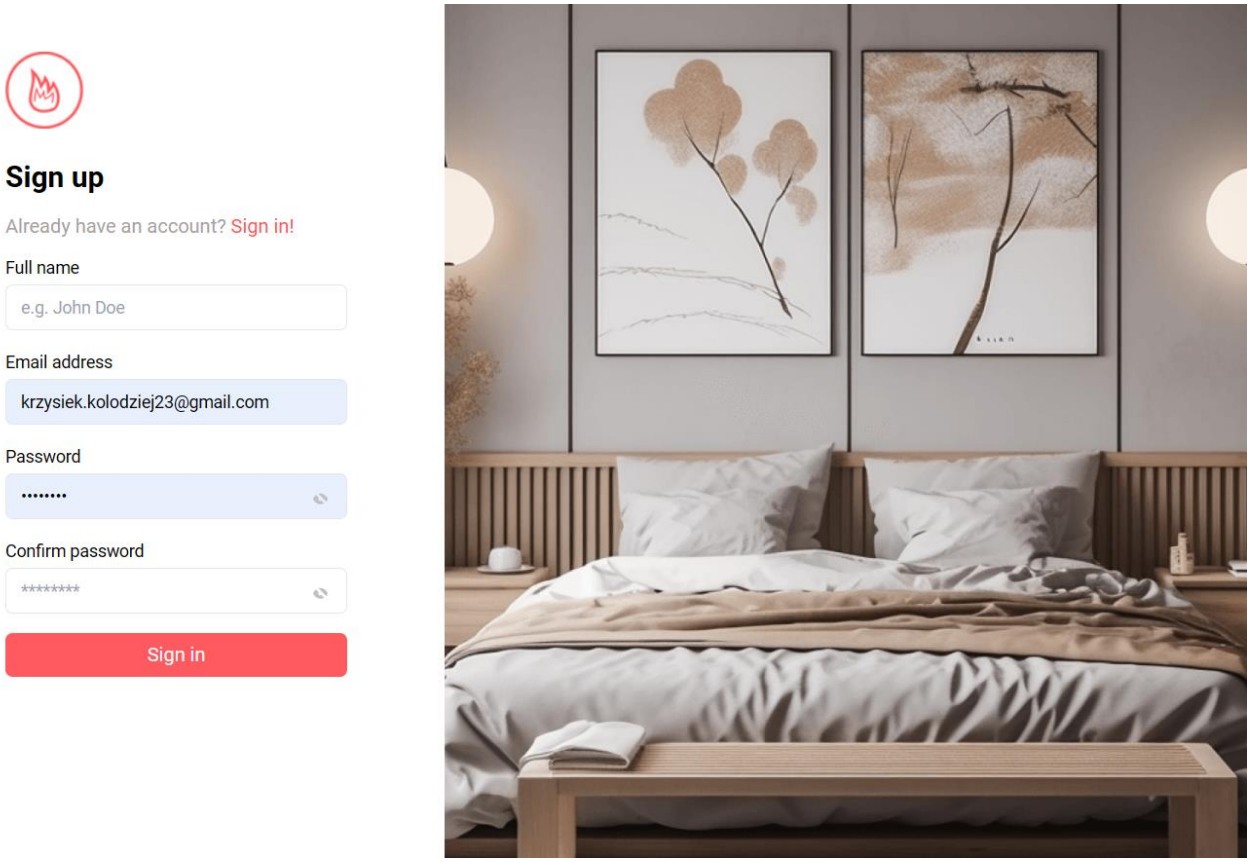
3. Opis aplikacji

Firebnb to nowoczesny system rezerwacji hoteli, który umożliwia użytkownikom zarówno dodawanie własnych ofert noclegowych, jak i rezerwowanie miejsc na wybrane daty. Jedną z kluczowych funkcjonalności jest autoryzacja i rejestracja użytkowników. Każdy użytkownik może zarządzać swoimi rezerwacjami, a także wyszukiwać oferty noclegowe wprowadzając odpowiednie kryteria. Aplikacja została zaprojektowana w sposób responsywny, z myślą o prostocie i wygodzie użytkowania.



Rysunek 1 - ekran logowania

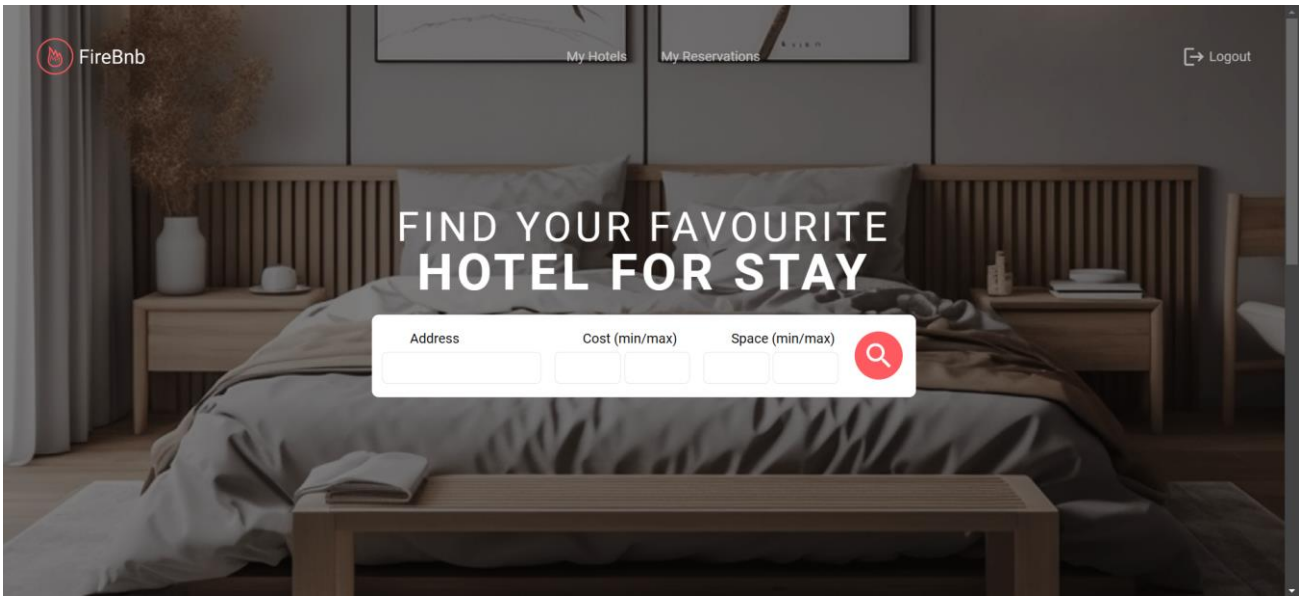
Ekran logowania jest pierwszym, co użytkownik widzi po wejściu na stronę. Może utworzyć konto klikając “Create a new account!” lub zalogować się, jeśli utworzył je już wcześniej.



Rysunek 2 - ekran rejestracji w wersji na tablety

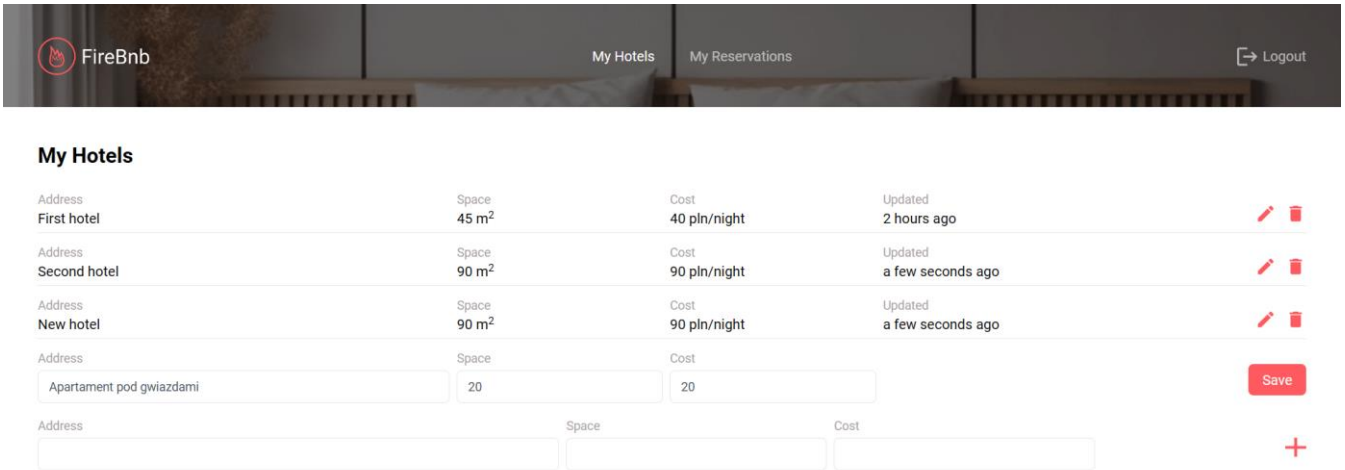
Po wejściu na stronę rejestracji, użytkownik ma możliwość wypełnienia formularza rejestracyjnego, który zawiera pola na dane takie jak imię i nazwisko, adres e-mail, hasło oraz

potwierdzenie hasła. Prawidłowo przesłany formularz poprzez wciśnięcie przycisku „Sing in” przekierowuje użytkownika z powrotem na stronę logowania.



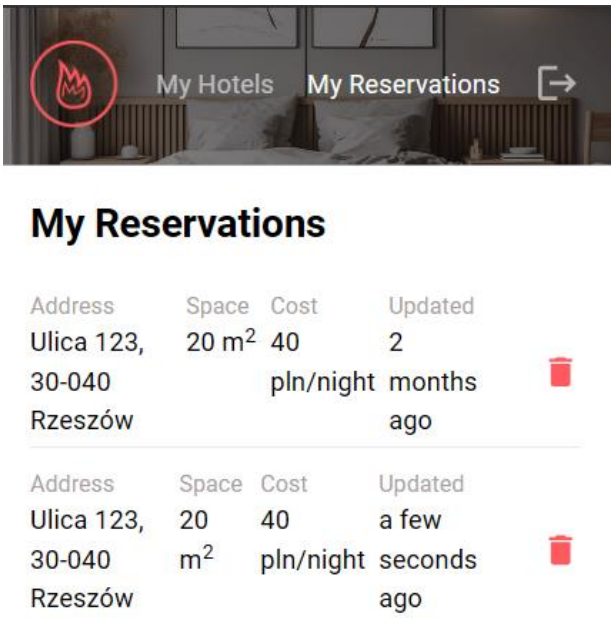
Rysunek 3 - strona główna aplikacji

Po pomyślnym zalogowaniu się, użytkownik zostaje przekierowany na stronę główną aplikacji Firebnb. Na tej stronie ma dostęp do kluczowych funkcji, takich jak wyszukiwanie i przeglądanie dostępnych noclegów. Użytkownik może także nawigować do sekcji „My Hotels”, gdzie zarządza swoimi ofertami noclegowymi, lub do sekcji „My Reservations”, aby przeglądać i zarządzać swoimi dokonanymi rezerwacjami. Dodatkowo, z poziomu strony głównej użytkownik ma możliwość wylogowania się, co kończy jego sesję w aplikacji.



Rysunek 4 - zarządzanie noclegami użytkownika

Na stronie „My Hotels” użytkownik ma możliwość zarządzania swoimi ofertami noclegowymi w pełnym zakresie. Może dodać nową ofertę hotelową, edytować istniejące oferty, wprowadzając zmiany w szczegółach takich jak przestrzeń, koszt czy adres, lub całkowicie usunąć wybraną ofertę. Warto zaznaczyć, że usunięcie noclegu powoduje również automatyczne usunięcie wszystkich powiązanych z nim rezerwacji.



Rysunek 5 – zarządzanie rezerwacjami użytkownika w wersji na smartfony

Na stronie „My Reservations” użytkownik może przeglądać wszystkie swoje aktywne rezerwacje i w razie potrzeby anulować wybraną z nich. Po dodaniu nowej rezerwacji, automatycznie pojawia się ona na tej stronie.

4. REST API

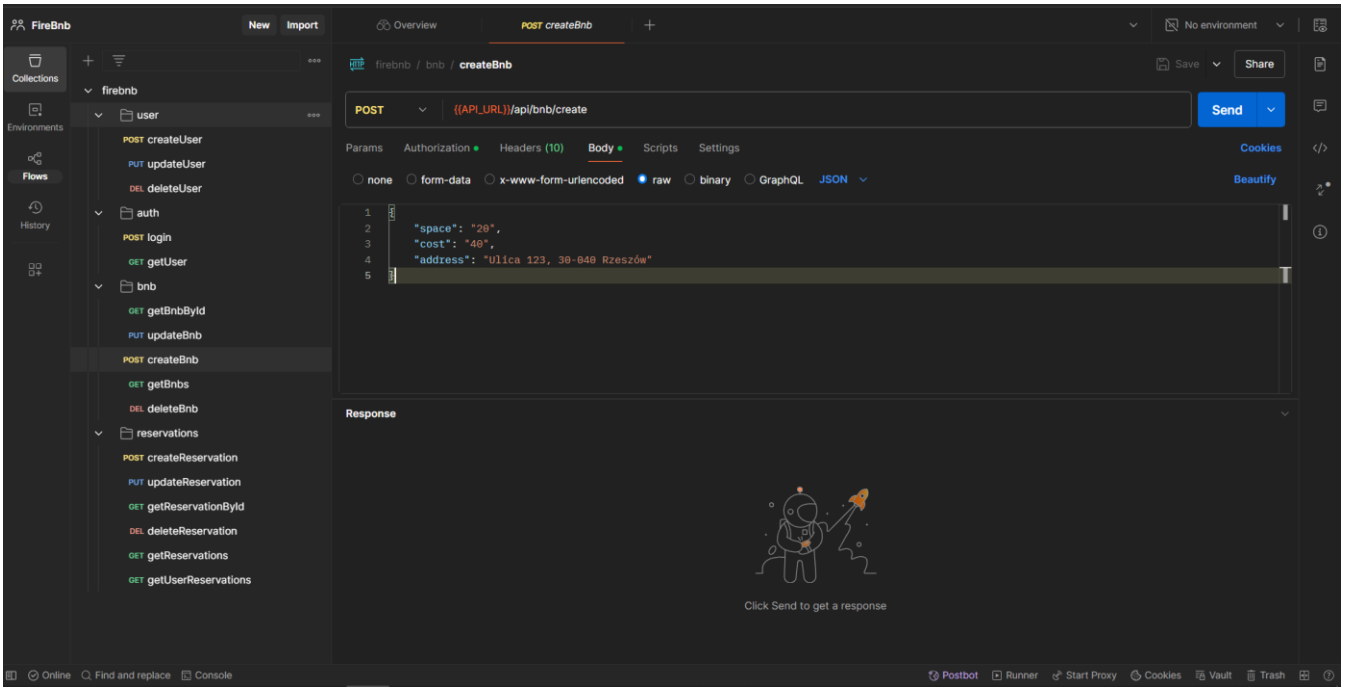
4.1. Wykorzystanie istniejącego projektu inżynierskiego oraz wybór technologii backendowej

Projekt opiera się na wcześniejszej pracy inżynierskiej innego studenta (Pelechowicz, J. (2024). REST API w wybranych frameworkach.), który skoncentrował się na budowie API w wybranych frameworkach. Tamta praca skupiała się wyłącznie na backendzie, implementując różne API oraz zapewniając konteneryzację przy użyciu Docker Compose. Ten projekt jest rozwinięciem tej koncepcji

dodatkowo o interfejs użytkownika. W każdej aplikacji, zarówno frontendowej jak i backendowej znajduje się osobny Dockerfile, który umożliwia konfigurację środowiska, którą następnie można uruchomić na każdym środowisku. Docker Compose ułatwił zarządzanie wieloma kontenerami oraz umożliwił uruchomienie ich wszystkich jednocześnie jednym poleceniem. Do projektu wykorzystano backend zaimplementowany w Express.js działający w środowisku Node.js. Warto jednak zaznaczyć, że API, na którym bazowano nie było idealne – podczas jego analizy wykryto kilka drobnych błędów, które wymagały naprawy. Wystąpiły między innymi błędy składniowe, czy też niedopracowania w logice niektórych funkcji. W celu wykonania przykładowej aplikacji, konieczne było również rozszerzenie backendu o dodatkowe funkcjonalności. Dodano endpoint, który zwraca rezerwacje noclegów dla aktualnie zalogowanego użytkownika. Niektóre z endpointów nie znalazły zastosowania w aplikacji, natomiast ich działanie zostało również przetestowane i naniesiono kilka poprawek na ewentualne niedociągnięcia.

4.2. Testowanie API

Aby sprawdzić poprawność działania oraz zgodność API z wymaganiami projektu, niezbędne było przeprowadzenie szczegółowych testów. W tym celu wykorzystano narzędzie Postman, które jest uniwersalnym środowiskiem do testowania i dokumentowania interfejsów API. Postman wspiera różne typy żądań (GET, POST, PUT, DELETE) i pozwala na precyzyjne dostosowanie parametrów wysyłanych zapytań.



Rysunek 6 - interfejs aplikacji Postman

4.3. Dostosowanie funkcjonalności API

W projekcie zastosowano wiele endpointów z istniejącego API, natomiast nie wszystkie znalazły swoje zastosowanie. W trakcie rozwoju projektu i implementacji, zidentyfikowano, że początkowa wersja nie obejmowała wszystkich funkcjonalności potrzebnych do prawidłowej implementacji projektu. Potrzeby użytkowników oraz wymagania frontendowe, czasami mogą ujawniać się dopiero podczas pracy nad systemem. Dodano w ten sposób endpoint `/api/reservation/list/user`, którego zadaniem jest „wyciągnięcie" informacji na temat rezerwacji aktualnie zalogowanego użytkownika. W Tabela 1 przedstawiono szczegółowe informacje na temat dostępnych endpointów, ich wymagania oraz status wykorzystania:

Kategoria	URL	Metoda	Ciało	Autentykacja	Opis
Użytkownik	api/user/create ✔	POST	full_name, password, email	-	Tworzenie nowego użytkownika. Wymaga unikalnego adresu email.
Użytkownik	api/user/update ✘	PUT	full_name, password, email	Token JWT	Aktualizacja danych aktualnie zalogowanego użytkownika. Jedynie właściciel może edytować dane.
Użytkownik	api/user/delete ✘	DELETE	email	Token JWT	Usunięcie konta użytkownika.
Autentykacja	api/auth/login ✔	POST	email, password	-	Utworzenie tokenu JWT dla użytkownika.
Autentykacja	api/auth/me ✔	GET	-	Token JWT	Pobranie danych o zalogowanym użytkowniku.
Noclegi	api/bnb/create ✔	POST	space, cost, address, user_id	Token JWT	Utworzenie nowego noclegu. Przypisanie noclegu do użytkownika.
Noclegi	api/bnb/index/<id> ✘	GET	id(podane w URL)	-	Pobranie informacji o wybranym noclegu.
Noclegi	api/bnb/list?query ✔	GET	cost_min, cost_max, space_min, space_max, address_like, user_id	-	Filtrowanie noclegów na podstawie różnych parametrów. Parametry podawane w query
Noclegi	api/bnb/update ✔	PUT	id, space, cost, address	Token JWT	Aktualizacja danych noclegu. Tylko właściciel noclegu może dokonywać zmian.
Noclegi	api/bnb/delete/<id> ✔	DELETE	id (podane w URL)	Token JWT	Usunięcie noclegu z bazy danych. Tylko właściciel noclegu może usunąć.

Rezerwacje	api/reservation/create ✔	POST	start_date, end_date, bnb_id	Token JWT	Złożenie nowej rezerwacji noclegu.
Rezerwacje	api/reservation/update ✘	PUT	id, start_date, end_date	Token JWT	Aktualizacja danych rezerwacji. Tylko właściciel może dokonywać zmian
Rezerwacje	api/reservation/index/<id> ✘	GET	id(podane w URL)	-	Pobranie szczegółów pojedynczej rezerwacji.
Rezerwacje	api/reservation/list?query ✘	GET	bnb_id, start_date, end_date	-	Pobranie listy rezerwacji dla określonego noclegu w zadanych ramach czasowych. Parametry podawane w query
Rezerwacje	api/reservation/list/user ✔ +	GET	start_date, end_date	Token JWT	Pobranie listy rezerwacji dla aktualnego użytkownika
Rezerwacje	api/reservation/delete/<id> ✔	DELETE	id(podane w URL)	Token JWT	Usunięcie rezerwacji. Tylko właściciel rezerwacji może usunąć.

Tabela 1 - definicja endpointów REST API

✔ - endpoint wykorzystany

✘ - endpoint niewykorzystywany

\+ - endpoint dodany do oryginalnego API na potrzeby aplikacji

5. Wykorzystanie Tanstack Query do komunikacji z API na przykładzie wybranych endpointów

5.1 O Tanstack Query

Komunikacja pomiędzy frontendem a API może być realizowana na wiele różnych sposobów. Jednym z nich jest wykorzystanie zewnętrznej biblioteki Tanstack Query, która jest wydajną, szeroko stosowaną bibliotekę w większości popularnych frameworków. Oto kilka powodów dlaczego warto po nią sięgać:

- Uproszczone pobieranie danych z interfejsu REST API – biblioteka ma wbudowane mechanizmy służące do obsługi złożonych scenariuszy tj. paginacja, filtrowanie, sortowanie, możemy łatwo pobierać dane i je aktualizować, bez konieczności ręcznego zarządzania żądaniami i odpowiedziami.

- Cachowanie i optymalizacja zapytań – dzięki automatycznemu cachowaniu, Tanstack Query przechowuje wyniki zapytań, co minimalizuje liczbę zapytań do API i poprawia

wydajność aplikacji. Dodatkowo dane mogą być automatycznie aktualizowane (refetching), co zapewnia ich aktualność,

- Łatwość mutacji danych – oprócz pobierania danych, biblioteka upraszcza również obsługę mutacji danych (dodawanie, aktualizowanie, usuwanie) poprzez wbudowane mechanizmy,

- Zarządzanie asynchronicznymi danymi – Tanstack Query automatyzuje obsługę danych przychodzących z API, można wykorzystać gotowe, sprawdzone rozwiązania, zamiast ręcznie implementować logikę.

5.2 Konfiguracja klienta Tanstack Query

Aby móc korzystać z Tanstack Query, musimy najpierw utworzyć instancję queryClient, a następnie umieścić ją wewnątrz providera, który owija nasz główny komponent. Dzięki temu mamy dostęp do Tanstack Query wewnątrz całej aplikacji..

Listing 1 – konfiguracja klienta Tanstack Query (App.tsx)

```
export const App = () => {
  const queryClient = new QueryClient();

  return (
    <QueryClientProvider client={queryClient}>
      <FireBnbApps />
    </QueryClientProvider>
  );
};

```

5.3 Pobieranie danych przy użyciu useQuery

Aby zaprezentować, jak działa mechanizm zapytań (query) oraz jakie korzyści płyną z jego wykorzystania, posłużymy się przykładem pobrania listy rezerwacji aktualnie zalogowanego użytkownika z endpointu GET `api/reservation/list/user` w aplikacji React.

Listing 2 – niestandardowy hook useUserReservations służący do pobrania listy rezerwacji (reservation.ts)

```
export function useUserReservations() {
  return useQuery({
    queryKey: ["user-reservations"],
    queryFn: () => client("reservation/list/user"),
  });
}
```

Hook useQuery zwraca obiekt zawierający wiele pól, m.in. data, isError, isSuccess, status, isInitialLoading itp. (https://tanstack.com/query/latest/docs/framework/react/reference/useQuery) które bardzo ułatwiają zarządzanie stanem aplikacji i dostosowywanie interfejsu do aktualnych warunków.

Wartość queryKey jest unikalnym identyfikatorem zapytania i pomaga zarządzać danymi w cache, pozwalając śledzić Tanstack Query, czy dane zostały już pobrane, czy wymagają ponownego zapytania itp. Z kolei queryFn to funkcja odpowiedzialna za wykonanie zapytania do API - w tym przypadku jest to client, funkcja ułatwiająca obsługę zapytań HTTP.

Listing 3 – wykorzystanie utworzonego hooku useUserReservations (my-reservations.tsx)

```
export const MyReservations = () => {
      const { data: myReservations } = useUserReservations();

      {myReservations?.data?.map(
            (reservation: ReservationType, index: number) => (
              <div
                key={index}
                className={`grid gap-2 grid-cols-[5fr_1fr] place-items-center
justify-items-end ${
                  index !== myReservations?.data?.length - 1 &&
                  "border-b border-stone-200"
                }`}
              >
                <div className="py-1 grid gap-2 grid-cols-[4fr_2fr_2fr_2fr] w-
full">
                  <div className="flex flex-col">
                    <p className={`${styles.paragraph2} text-stone-400`}>
                      Address
                    </p>
                    <p className={styles.paragraph}>
                      {reservation?.bnb?.address}
                    </p>
      ...

```

W Listing 3 komponent MyReservations używa hook useUserReservations, aby pobrać listę rezerwacji zalogowanego użytkownika z API. Następnie wyświetla otrzymane dane w układzie siatki (grid) dostosowując odpowiednio style.

5.4 Operacje na danych przy użyciu useMutation

Mutacje w Tanstack Query służą do wykonywania operacji zmieniających dane na serwerze, takich jak tworzenie, aktualizacja czy usuwanie zasobów. W przeciwieństwie do zapytań (query), które służą do pobierania danych, mutacje umożliwiają bezpośrednią ingerencję w stan aplikacji poprzez modyfikację danych na serwerze.

Kiedy mutacja jest wykonywana, Tanstack Query automatycznie zarządza stanem operacji, zwracając informacje takie jak isLoading, isSuccess, czy isError, co ułatwia obsługę interfejsu w czasie trwania operacji. Dodatkowo, po zakończonej mutacji, możemy odświeżyć dane w cache za pomocą

mechanizmu invalidacji, co pozwala na automatyczne aktualizowanie widoków aplikacji po zakończeniu operacji.

Jednym z przykładów mutacji Tanstack Query w omawianym projekcie jest implementacja endpointu DELETE `api/reservation/delete/<id>`.

Listing 4 - przykład mutacji na podstawie usuwania rezerwacji (my-reservations.tsx)

```
1
2   export const MyReservations = () => {
3     const { mutate: deleteReservation } = useDeleteReservation();
4     ...
5     const handleDeleteReservation = async (reservation: ReservationType) => {
6       await deleteReservation(reservation?.id);
7     };
8     ...
9
10  export function useDeleteReservation() {
11    const queryClient = useQueryClient();
12
13    return useMutation({
14      mutationFn: (reservationId: string) =>
15        client(`reservation/delete/${reservationId}`, {
16          method: "DELETE",
17        }),
18      onSuccess: () => {
19        queryClient.invalidateQueries({ queryKey: ["user-reservations"] });
20        toast.success("Reservation deleted successfully");
21      },
22    });
23  }
24
```

W kodzie komponentu MyReservations wykorzystano hook useDeleteReservation (linia 3), który odpowiada za usuwanie rezerwacji. Hook ten używa funkcji useMutation z Tanstack Query, aby wykonać zapytanie HTTP typu DELETE pod adres api/`reservation/delete/${reservationId}`. Gdy rezerwacja zostanie usunięta, dzięki funkcji onSuccess (linia 18) następuje odświeżenie danych listy rezerwacji za pomocą invalidateQueries (linia 19) z kluczem user-reservations, co powoduje, że aktualna lista rezerwacji użytkownika zostaje odświeżona. Dodatkowo wyświetla się powiadomienie o sukcesie operacji dzięki wywołaniu funkcji toast.success (linia 20).

5.5 Różnice w Tanstack Query pomiędzy różnymi frameworkami

Użycie Tanstack Query w React i Vue jest bardzo podobne, ponieważ biblioteka ta dostarcza spójny interfejs niezależnie od frameworku, w którym jest stosowana. Zarówno w React, jak i w Vue, podstawowe operacje, takie jak pobieranie danych za pomocą useQuery lub zarządzanie mutacjami z useMutation, są niemalże identyczne. Różnice wynikają głównie z architektury samego frameworka,

np. React korzysta z hooków, natomiast w Vue stosuje się kompozycję funkcji (setup), natomiast ogólna logika i sposób zarządzania stanem danych w Tanstack Query pozostaje spójna.

5.6 Implementacja funkcji client, wykorzystywanej podczas tworzenia niestandardowych hooków Tanstack Query

Listing 5 – funkcja client ułatwiająca wykonywanie zapytań HTTP (utils.ts)

```
1
2   export async function client(
3     endpoint: string,
4     {
5       data: requestData,
6       headers: customHeaders,
7       ...customConfig
8     }: ClientOptions = {}
9   ) {
10    const accessToken = getAccessToken();
11    const headers: Record<string | "Authorization", string> = {};
12
13    if (accessToken) headers.Authorization = `Bearer ${accessToken}`;
14
15    if (requestData) headers["Content-Type"] = "application/json";
16
17    const config = {
18      method: requestData ? "POST" : "GET",
19      body: requestData ? JSON.stringify(requestData) : undefined,
20      headers: {
21        ...headers,
22        ...customHeaders,
23      },
24      ...customConfig,
25    };
26
27    const response = await window.fetch(`${apiURL}/${endpoint}`, config);
28
29    if (response.status === 401) return Promise.reject({ message: "Please re-
30  authenticate." });
31
32    const data = await response.json().catch(() => null); // catch in case of empty
33  response
34    if (response.ok) return data;
35    else return Promise.reject(data);
36  }
37
```

Funkcja client jest asynchroniczną funkcją JavaScript, która służy do wykonywania zapytań HTTP do serwera. Jest to uniwersalne podejście do komunikacji z API, umożliwiające elastyczne dostosowanie nagłówków (linie 20-23) i ciała zapytania (linia 19) oraz zapewniające automatyczną autentykację poprzez umieszczenie w nagłówku „Authorization” tokena JWT (linia 13).

6. Implementacja aplikacji w React.js

6.1. O React.js

6.1.1. Opis

React jest jednym z najpopularniejszych frameworków frontendowych (a dokładniej biblioteką) i ma wiele zalet, które sprawiają, że programiści i firmy często wybierają go do tworzenia aplikacji. Programowanie w React opiera się na komponentach, co oznacza, że aplikacje są budowane z małych, wielokrotnego użytku fragmentów interfejsu użytkownika (UI), które mogą posiadać własny stan i logikę.

6.1.2. Hooki

Hooki w React umożliwiają zarządzanie stanem oraz efektami ubocznymi w komponentach. Dzięki nim React cechuje się sporą elastycznością, umożliwiając tworzenie bardziej zwięzłego oraz modularnego kodu. Jednym z najważniejszych hooków jest useState, który pozwala na dodanie lokalnego stanu do komponentów funkcyjnych. Drugim niezbędnym jest useEffect, który pozwala na wywoływanie efektów ubocznych, tj. pobieranie danych, czy też manipulowanie drzewem DOM. Innym ważnym hookiem jest useContext, który pozwala na dzielenie się globalnym stanem pomiędzy komponentami. Oprócz nich istnieją także useReducer, useCallback, useMemo, useRef i wiele innych, które ułatwiają programistom pracę nad aplikacją.

Listing 6 - przykład użycia useEffect w aplikacji (my-hotels.tsx)

```
1
2   useEffect(() => {
3     const handleKeyDown = (event: KeyboardEvent) => {
4       if (event.key === "Escape") {
5         setEditedBnb(null);
6       }
7     };
8
9     window.addEventListener("keydown", handleKeyDown);
10
11    return () => {
12      window.removeEventListener("keydown", handleKeyDown);
13    };
14  }, []);
15
```

W tym przypadku useEffect jest używany w celu dodania (linia 9) i usunięcia (linia 12) nasłuchiwacza zdarzeń dla klawisza "Escape" w oknie przeglądarki. W przypadku naciśnięcia tego klawisza, stan editedBnb zostaje ustawiony na null (linia 5). useEffect może zwracać wartość (wtedy mówimy o funkcji sprzątającej), która jest wywoływana podczas odmontowania komponentu, lub gdy

useEffect jest wywoływany ponownie. Zapobiega to ewentualnym wyciekom pamięci i zapewnia, że nasłuchiwacz nie jest aktywny po zakończeniu działania komponentu. Pusta tablica (linia 14) jako drugi argument oznacza, że efekt zostanie wywołany tylko raz, zaraz po pierwszym renderowaniu komponentu.

Listing 7 - przykład użycia useState w aplikacji (login.tsx)

```
1
2  export const Login = () => {
3   ...
4   const [isLoading, setIsLoading] = useState(false);
5   const { mutate } = useLogin({
6     onMutate: () => {
7         setIsLoading(true);
8     },
9     onError() {
10        setIsLoading(false);
11        toast.error("Invalid email or password!");
12    },
13    onSuccess: () => {
14        setIsLoading(false);
15    },
16  });
17  ...
18  return (
19  ...
20      <Button isLoading={isLoading} className="w-full">
21          Sign in
22      </Button>
23  ...
24
```

Hook useState jest używany do zarządzania stanem lokalnym komponentu. W Listing 7 tworzy on zmienną stanu isLoading oraz funkcję setIsLoading, która pozwala na aktualizację tego stanu. Początkowo stan ten jest ustawiony na false, co oznacza, że process logowania nie jest aktywny. Stan ten służy do śledzenia, czy aplikacja jest w trakcie wykonywania żądania logowania, przyjmuje wartość true podczas próby logowania i false po zakończeniu tego procesu. Wartość stanu jest przekazywana do komponentu Button, aby zmieniać styl przycisku na ładujący się, po wysłaniu zapytania do API i w oczekiwaniu na odpowiedź. Customowe hooki, tj. występujący w tym przykładzie autorski useLogin, zostaną omówione w kolejnym przykładzie.

Listing 8 - przykład tworzenia niestandardowych hooków (auth.ts)

```
1
2  export const { ..., useLogin, ...} = configureAuth<
3    ...
4    LoginCredentials,
5    ...
```

```
>({
  ...
  loginFn: async (data: LoginCredentials) => {
    const response = await client("auth/login", {
      method: "POST",
      data,
    });

    setAccessToken(response?.token);

    return { email: data.email };
  },
  ...
});

```

W React istnieje możliwość tworzenia customowych hooków, które pozwalają na łatwe zarządzanie złożoną logiką oraz wielokrotne jej użycie w różnych miejscach aplikacji. W przedstawionym kodzie customowy hook useLogin służy do obsługi logowania w aplikacji. Jego implementacja jest konfigurowana wewnątrz funkcji configureAuth, gdzie między innymi definiowana jest funkcja loginFn, odpowiedzialna za wykonywanie żądania logowania do serwera. Funkcja configureAuth pochodzi z bilbioteki react-query-auth i ułatwia konfigurację autoryzacji w aplikacji.

Funkcja loginFn działa asynchronicznie, wysyłając dane logowania (obiekt LoginCredentials) do serwera przy użyciu klienta HTTP (client). W przypadku powodzenia, odpowiedź z serwera zawiera token dostępu, który jest następnie zapisywany do localStorage przy pomocy funkcji setAccessToken.

Listing 9 - przykład użycia useRef w aplikacji (homepage.tsx)

```

export const Homepage = () => {
  ...
  const modalRef = useRef(null);
  ...
  useOnClickOutside(modalRef, () => {
    setBookMeModalOpen(false);
  });
  ...
  return (
    <>
      ...
          {bookMeModalOpen && (
            <div
              ref={modalRef}
            >
              ...
            </div>
          )}
        ...
```

```
    </>
  );
};

```

useRef to hook w React, który pozwala tworzyć odniesienia do elementów DOM lub innych wartości, które utrzymują się przez cały cykl życia komponentu, ale nie wywołują ponownego renderowania, gdy ich wartość się zmienia. Najczęściej używa się go do przechowywania referencji do elementów DOM, tak jak w tym przykładzie, ale może także służyć do przechowywania jakiejkolwiek zmiennej, której zmiana nie powinna powodować rerenderowania komponentu.

W podanym przykładzie, useRef jest używany do utworzenia referencji do modalnego okna (modalRef). Wartość ta jest przekazywana jako atrybut ref do elementu div, który reprezentuje modal. modalRef początkowo jest ustawione na null, co oznacza, że referencja będzie pusta do momentu, aż React przypisze do niej element DOM po jego renderowaniu. Dodatkowo w kodzie jest używana funkcja useOnClickOutside (pochodząca ze zbioru hooków usehooks-ts), która służy do wykrywania kliknięć poza modalem. Funkcja ta przyjmuje referencję modalRef oraz callback, który zostanie wywołany w momencie kliknięcia poza modalem. W tym przypadku callback zamyka modal, ustawiając stan bookMeModalOpen na false.

6.1.3. Zarządzanie stanem

6.1.3.1. Przekazywanie danych przez propsy

React stosuje jednokierunkowy przepływ danych, co oznacza, że dane przepływają w jednym kierunku – od komponentów rodzica do komponentów dziecka przez propsy (wartości przekazywane do komponentów przez atrybuty). To ułatwia śledzenie i debugowanie danych w aplikacji, a także zapewnia bardziej przewidywalne zachowanie komponentów.

Listing 10 - przekazywanie danych w dół poprzez propsy (homepage.tsx)

```

<Button
    variant="primary-inverted"
    onClick={() => handleBookMeClicked(bnb)}
>
   Book me!
</Button>

```

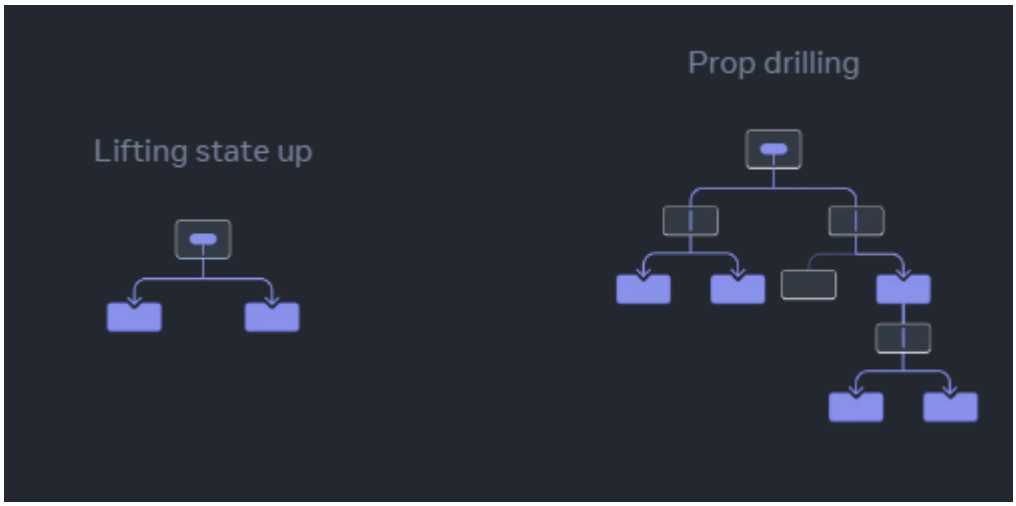
W Listing 10 do komponentu Button przekazywane są dwa propsy:

- **variant**: Określa styl przycisku, w tym przypadku wartością jest "primary-inverted". To pozwala komponentowi Button na odpowiednie dostosowanie wyglądu w zależności od przekazanego wariantu.
- **onClick**: Jest to funkcja obsługi zdarzenia, która zostanie wywołana po kliknięciu przycisku. W tym przypadku wywoływana jest funkcja handleBookMeClicked(bnb), która przekazuje zmienną bnb jako argument.

6.1.3.2. Stan lokalny a problem prop drillingu

W Listing 7 przedstawiono zarządzania stanem za pomocą hooku useState, który pozwala komponentom przechowywać i aktualizować stan lokalny. Jest to doskonałe rozwiązanie w przypadku prostych interakcji i komponentów, które nie wymagają współdzielenia stanu z innymi elementami aplikacji, a co za tym idzie przekazywania propsów przez wiele warstw aplikacji.

W bardziej złożonych aplikacjach, zarządzanie stanem lokalnym może prowadzić do problemów, zwłaszcza gdy trzeba przekazywać stan między wieloma komponentami lub obsługiwać złożoną logikę (prop drilling).



Rysunek 7 - problem prop drillingu

6.1.3.3. Stan globalny

Aby rozwiązać problem z Rysunek 7, React oferuje mechanizm kontekstu przy użyciu hooku useContext. Pozwala on na tworzenie globalnego stanu, który może być współdzielony między różnymi komponentami, bez konieczności ręcznego przekazywania propsów przez kolejne warstwy drzewa komponentów.

Listing 11 - przykład tworzenia stanu globalnego przy użyciu useContext (przykład z dokumentacji React)

```
1
2   const CountContext = React.createContext();
```

```

const CounterProvider = ({ children }) => {
  const [count, setCount] = useState(0);

  return (
    <CountContext.Provider value={{ count, setCount }}>
      {children}
    </CountContext.Provider>
  );
};

const DisplayCount = () => {
  const { count } = useContext(CountContext);
  return <p>Count: {count}</p>;
};

const IncrementButton = () => {
  const { setCount } = useContext(CountContext);
  return <button onClick={() => setCount(c => c + 1)}>Increment</button>;
};

const App = () => (
  <CounterProvider>
    <DisplayCount />
    <IncrementButton />
  </CounterProvider>
);
```

W Listing 11 komponenty DisplayCount oraz IncrementButton mają dostęp do stanu globalnego za pomocą useContext, bez konieczności przekazywania propsów przez kolejne poziomy drzewa komponentów.

6.1.3.4. Zaawansowane zarządzanie stanem

W przypadkach, gdy stan aplikacji staje się bardziej złożony, a jego zmiany wymagają bardziej skomplikowanej logiki (np. wiele typów działań na stanie), możemy skorzystać z hooku useReducer. Jest on podobny do useState, ale oferuje bardziej przewidywalne zarządzanie stanem, używając reducera – funkcji, która przetwarza akcje i aktualizuje stan na podstawie typu akcji.

Listing 12 - przykład zaawansowanego zarządzania stanem (przykład z dokumentacji React)

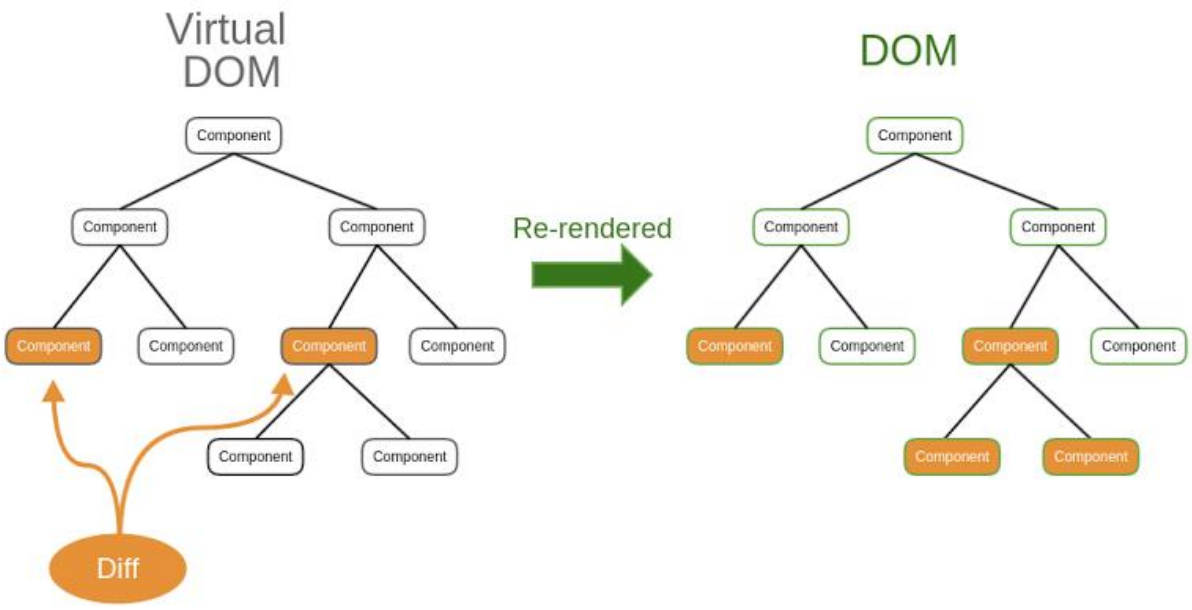
```

function reducer(state, action) {
  if (action.type === 'incremented_age') {
    return {
      age: state.age + 1
    };
  }
  throw Error('Unknown action.');
}
```

```
export default function Counter() {
  const [state, dispatch] = React.useReducer(reducer, { age: 42 });

  return (
    <>
      <button onClick={() => {
        dispatch({ type: 'incremented_age' })
      }}>
        Increment age
      </button>
      <p>Hello! You are {state.age}.</p>
    </>
  );
}
```

W Listing 12 komponent React korzysta z hooku useReducer do zarządzania stanem, gdzie reducer aktualizuje wiek (age) na podstawie typu akcji. Nierozpoznany typ funkcji powoduje pojawienie się błędu w aplikacji. Funkcja dispatch wywoływana po kliknięciu przycisku powoduje zwiększenie wieku o 1, co automatycznie aktualizuje wyświetlany tekst w komponencie.

6.1.4. Virtual DOM

React używa Virtual DOM, czyli wirtualnej reprezentacji drzewa DOM, do optymalizacji procesu renderowania. Zamiast bezpośrednio manipulować rzeczywistym DOM-em, React najpierw dokonuje zmian w Virtual DOM, a następnie efektywnie synchronizuje te zmiany z rzeczywistym DOM-em. Zamiast dokonywać na całym drzewie, React wprowadza zmiany tylko w miejscach, które uległy zmianie, lub w ich bezpośrednich dzieciach (Rysunek 8). Dzięki temu aplikacje często działają szybciej niż te, które używają tradycyjnego podejścia do zarządzania DOM-em.



Rysunek 8 - sposób działania Virtual DOM (*https://jayeshinde.medium.com/need-for-virtual-dom-19ebf6843d95*)

Listing 13 - Przykład użycia Virtual DOM w React (przykład własny, nieużywany w projekcie)

```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  const handleClick = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <h1>Licznik: {count}</h1>
      <button onClick={handleClick}>Zwiększ</button>
    </div>
  );
}

export default Counter;

```

Na początku React tworzy reprezentację Virtual DOM na podstawie struktury komponentu, a następnie renderuje ją w rzeczywistym DOM.

- Zmiana stanu:
  Gdy użytkownik kliknie przycisk, zmienia się stan count, co powoduje utworzenie nowej wersji Virtual DOM, uwzględniającej zaktualizowaną wartość licznika.

- Porównanie (Diffing):
  React porównuje nową wersję Virtual DOM z poprzednią, wykrywając różnice. W tym przypadku zmienia się tylko zawartość tagu <h1>, w którym wyświetlany jest licznik.

- Efektywna aktualizacja (Reconciliation):
  Na podstawie wykrytych zmian React aktualizuje jedynie zmodyfikowany fragment rzeczywistego DOM, czyli tekst w tagu <h1>, zamiast ponownie renderować cały komponent. Dzięki temu aplikacja działa szybciej i bardziej efektywnie.

6.1.5. Rozszerzenia plików

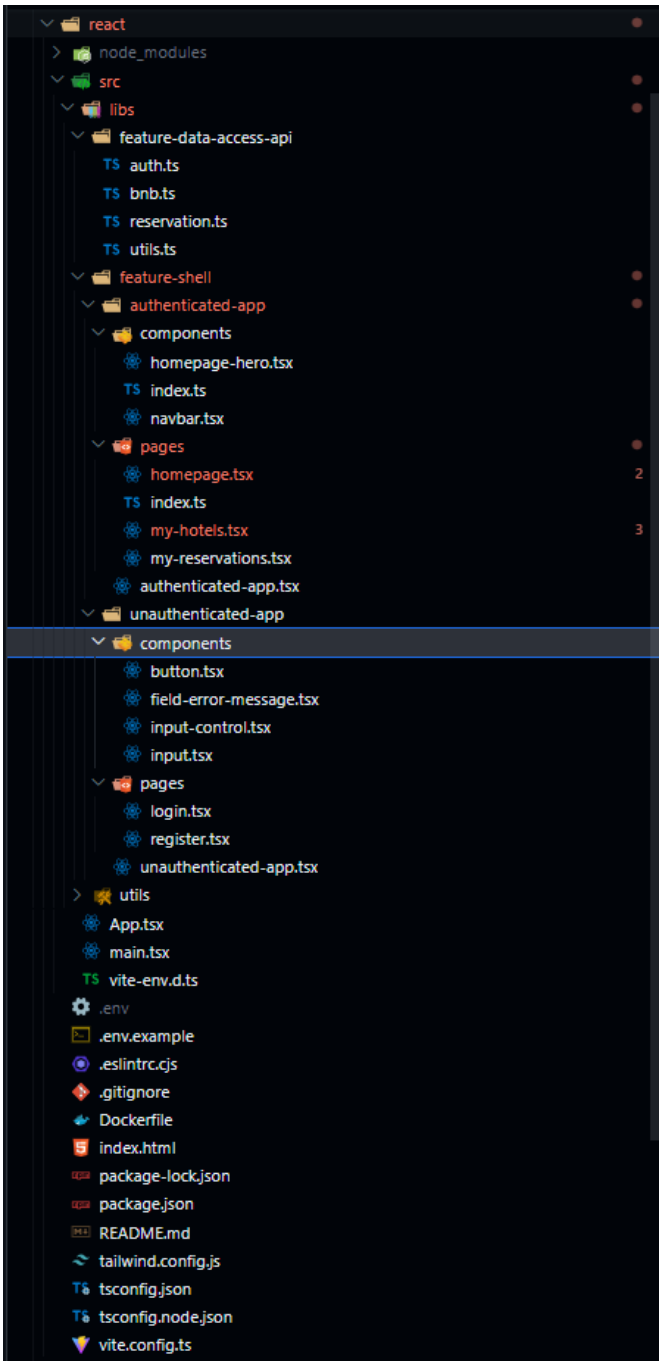
Pliki w React przyjmują 2 rozszerzenia:

- .jsx – komponenty Reactowe napisane przy użyciu składni JavaScript
- .tsx – rozszerzenie .jsx o możliwość pisania w Typescripcie

6.2. Struktura plików w projekcie

- **src -** główny katalog źródłowy aplikacji.
  - **libs -** folder zawierający moduły i biblioteki aplikacji:
    - **feature-data-access-api –** odpowiada za komunikację z API
    - **feature-shell -** struktura aplikacji podzielona na:
      - **authenticated-app**
        - **components**: komponenty UI dla użytkowników zalogowanych
        - **pages**: strony dostępne tylko dla zalogowanych użytkowników
      - **unauthenticated-app**
        - **components**: komponenty dla użytkowników niezalogowanych
        - **pages**: strony dostępne dla niezalogowanych użytkowników
    - **utils -** funkcje pomocnicze i zasoby:
      - **icons**: zbiór ikon używanych w interfejsie,
      - **schemas**: schematy walidacji danych, np. dla formularzy.

- **types**: definicje typów (dla w TypeScript) dla struktury danych w aplikacji.



Rysunek 9 - struktura plików w aplikacji React

6.3. Najważniejsze pliki

6.3.1. Plik main.tsx

Listing 14 - main.tsx odpowiedzialny za renderowanie aplikacji

```tsx
import ReactDOM from "react-dom/client";
import "../../../public/index.css";
import { App } from "./App";

ReactDOM.createRoot(document.getElementById("root")!).render(<App />);
```

```
6
```

W pliku main.tsx kluczową rolę odgrywa funkcja ReactDOM.createRoot, która służy do renderowania aplikacji React na stronie internetowej. ReactDOM to moduł, który zapewnia metody do interakcji z drzewem DOM w przeglądarce, umożliwiając osadzanie komponentów w rzeczywistym DOM-ie HTML. W tym pliku odbywa się import globalnych stylów dla aplikacji.

6.3.2. Plik App.tsx

Listing 15 - komponent FireBnbApps oraz komponent główny App

```
1  const FireBnbApps = () => {
2    const { data, isLoading } = useUser();
3
4    if (isLoading) return;
5    if (!data) return <UnauthenticatedApp />;
6    return <AuthenticatedApp />;
7  };
8
9  export const App = () => {
10   const queryClient = new QueryClient();
11
12   return (
13     <QueryClientProvider client={queryClient}>
14       <ToastContainer position="top-center" autoClose={2500} />
15       <FireBnbApps />
16     </QueryClientProvider>
17   );
18 };
```

W komponencie funkcyjnym FireBnbApps wywoływane są konkretne elementy, w zależności od stanu autoryzacji użytkownika. Plik App.tsx jest głównym komponentem aplikacji.

6.4. Uwierzytelnianie w aplikacji

Listing 16 - proces uwierzytelniania w aplikacji (auth.ts)

```
1
2  export const { useUser, useLogin, ...} = configureAuth<
3    User | null,
4    LoginCredentials,
5    ...
6  >({
7    userFn: async () => {
8      const token = getAccessToken();
9      if (!token) return null;
10
11     try {
12       return await client("auth/me");
13     } catch {
14       return null;
```

```
    }
  },
  loginFn: async (data: LoginCredentials) => {
    const response = await client("auth/login", {
      method: "POST",
      data,
    });

    setAccessToken(response?.token);

    return { email: data.email };
  },
  ...
});

...

const FireBnbApps = () => {
  const { data, isLoading } = useUser();

  if (isLoading) return;
  if (!data) return <UnauthenticatedApp />;
  return <AuthenticatedApp />;
};

...

export const Login = () => {
  ...
  const { mutate } = useLogin({
    ...
  });
  const handleFormSubmit = handleSubmit((values) => mutate(values));
  ...
  return (
    <div className="w-full flex">
      ...
        <form onSubmit={handleFormSubmit}>
          ...
        </form>
      </div>
      ...
    </div>
  );
};
```

W zależności od wyniku wywołania useUser w głównym komponencie aplikacji App.tsx, który sprawdza stan autoryzacji użytkownika, wyświetlana jest odpowiednia aplikacja: AuthenticatedApp dla zalogowanych użytkowników lub UnauthenticatedApp dla tych, którzy nie są zalogowani. Jeśli użytkownik nie jest zautoryzowany, odbywa się logowanie w aplikacji za pomocą komponentu Login,

który wykorzystuje mutację useLogin do przesyłania danych logowania. W ten sposób interfejs dynamicznie reaguje na stan użytkownika, zapewniając odpowiednie doświadczenie w zależności od stanu autoryzacji użytkownika.

6.5. Routing i nawigacja

Listing 17 - routing dla zalogowanego użytkownika (authenticated-app.tsx)

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <Homepage />,
  },
  {
    path: "/my-hotels",
    element: <MyHotels />,
  },
  {
    path: "/my-reservations",
    element: <MyReservations />,
  },
]);

export function AuthenticatedApp() {
  return <RouterProvider router={router} />;
}

```

AuthenticatedApp jest komponentem funkcyjnym, który dostarcza skonfigurowany router, co oznacza że zalogowani użytkownicy mogą korzystać z nawigacji po aplikacji według określonych ścieżek. Mechanizm ten realizowany jest przez bibliotekę react-router, która jest najpopularniejszym wyborem do routingu w aplikacjach React. Komponent AuthenticatedApp jest wykonany w sposób analogiczny, zmieniają się jedynie ścieżki.

Listing 18 - pobieranie aktualnego routa oraz nawigacja (navbar.tsx)

```
export const Navbar = () => {
  const navigate = useNavigate();
  const location = useLocation();
  ...
  return (
     <div className="w-full z-10 absolute top-0 left-0 p-5 md:p-10 flex items-
center justify-between text-white text-md">
      ...
      <div className="flex gap-5 md:gap-10">
        <button
          onClick={() => navigate("/my-hotels")}
          className={`hover:opacity-80 ${
```

```
            location.pathname !== "/my-hotels" && "opacity-60"
          }`}
        >
          My Hotels
        </button>
        ...
      </div>
      ...
    </div>
  );
};

```

Dzięki hookom z biblioteki React Router, aplikacje mogą łatwo zarządzać routingiem i dostosowywać UI w oparciu o bieżącą lokalizację. W Listing 18 przedstawiono:

- **useNavigate**: Hook, który umożliwia nawigację do innych ścieżek w aplikacji. W przykładzie, kliknięcie przycisku "My Hotels" wywołuje funkcję navigate, która przenosi użytkownika do trasy „/my-hotels”.

- **useLocation**: Hook, który zwraca obiekt reprezentujący aktualną lokalizację (routę) w aplikacji. Dzięki temu można łatwo sprawdzić, na jakiej stronie aktualnie znajduje się użytkownik, co pozwala na dynamiczną zmianę stylów lub zawartości komponentów, jak w przedstawionym przypadku zmiany nieprzezroczystości przycisku w navbarze.

6.6. Obsługa formularzy

Listing 19 - obsługa formularza rejestracji (register.tsx)

```

export const Register = () => {
  const { control, handleSubmit } = useForm<RegisterForm>({
    resolver: zodResolver(registerUserSchema),
  });
  const [isLoading, setIsLoading] = useState(false);
  const { mutate } = useRegister({
    ...
  });
  const handleFormSubmit = handleSubmit((values) => {
    const { password_confirmation, ...restValues } = values;
    mutate(restValues);
  });
  ...
  return (
    <div className="w-full flex">
      <div className="w-full flex flex-col space-y-3 px-20 py-10">
        ...
        <form onSubmit={handleFormSubmit}>
          <InputControl
```

```
            label="Full name"
            control={control}
            name="full_name"
            placeholder="e.g. John Doe"
          />
          <InputControl
            label="Email address"
            control={control}
            name="email"
            placeholder="mail@example.com"
          />
          <InputControl
            label="Password"
            control={control}
            name="password"
            type="password"
            placeholder="********"
          />
          <InputControl
            label="Confirm password"
            control={control}
            name="password_confirmation"
            type="password"
            placeholder="********"
          />
          <Button isLoading={isLoading} className="w-full">
            Sign in
          </Button>
        </form>
      </div>
      ...
    </div>
  );
};

```

W projekcie obsługę formularzy ułatwia biblioteka react-hook-form, która zapewnia prosty i efektywny sposób zarządzania stanem formularzy. Przykład użycia tej biblioteki przedstawia komponent rejestracji. Hook useForm zwraca funkcję handleSubmit, która służy do przesyłania danych formularza. Dzięki control, który również jest zwracany przez useForm, można łatwo integrować komponenty formularza z logiką zarządzania stanem, co upraszcza proces obsługi wartości i walidacji. Użycie zodResolver umożliwia integrację z biblioteką Zod do walidacji danych według określonych schematów, w tym przykładzie następuje walidacja przy użyciu registerUserSchema. To podejście pozwala na efektywne zarządzanie formularzami.

6.7. Stylowanie i responsywność

Listing 20 - stylowanie przycisku według przekazanych propsów (button.tsx)

```
function getButtonClassName({
  disabled,
  className,
  isLoading,
  variant,
  size = "medium",
}: AppButtonProps) {
  const baseClass = twMerge(
    "flex gap-2 rounded-md items-center justify-center text-white",
    size === "small" && "px-3 md:h-6 h-8 text-sm",
    size === "medium" && "px-4 md:h-9 h-12 text-md",
    size === "large" && "px-5 md:h-12 h-14 text-lg",
    disabled ? "bg-gray-400" : "bg-primary hover:opacity-80",
    variant === "primary-inverted" &&
      "bg-white text-primary border border-primary hover:bg-primary hover:text-
white"
  );

  const disabledClass = disabled
    ? "cursor-not-allowed"
    : "transition-colors duration-200";

  const loadingClass = isLoading
    ? "animate-pulse pointer-events-none touch-none"
    : "";

  return twMerge(baseClass, disabledClass, loadingClass, className);
}
```

Tailwind CSS to framework CSS, który stosuje podejście utility-first, co oznacza, że zamiast definiować style w osobnych klasach, dostarcza gotowych klas, które można bezpośrednio używać w trakcie tworzenia aplikacji. Dzięki temu programiści mogą szybko tworzyć i modyfikować style bez konieczności pisania własnego CSS.

W projekcie wykorzystano również tailwind-merge, które umożliwia łączenie kilku klas stylów w jeden ciąg, co ułatwia uniknięcie konfliktów pomiędzy klasami.

Przykład klas CSS w getButtonClassName:

- **rounded-md**: Ta klasa nadaje przyciskowi średnie zaokrąglenie rogów, co poprawia estetykę, (odpowiednikiem klasy w CSS jest: border-radius: 0.375rem).

- **bg-gray-400**: Używana, gdy przycisk jest zablokowany. Oznacza to, że jego kolor tła będzie szary, co wizualnie sygnalizuje użytkownikowi, że nie jest on aktywny,

- **bg-primary hover:opacity-80**: Stosowana dla aktywnego przycisku; definiuje kolor tła oraz efekt najechania, zmniejszający nieprzezroczystość, co dodaje interaktywności.

- **md:h-9**: Oznacza, że wysokość przycisku wynosi 9 jednostek (odpowiednikiem tej klasy w CSS jest height: 2.25rem;" tylko na średnich ekranach i większych, co ułatwia tworzenie responsywnych interfejsów.

Dzięki temu podejściu, Tailwind CSS upraszcza proces stylizacji aplikacji, pozwalając na łatwe dostosowywanie i szybkie wprowadzanie zmian w projekcie, co przekłada się na lepszą elastyczność i wydajność pracy.

6.8. Implementacja wybranego endpointu

W poprzednich rozdziałach dokumentacji omawiane były głównie funkcje wykorzystywane do implementacji endpointów GET i DELETE, natomiast nie został jeszcze przedstawiony przykład związany z aktualizacją zasobów w bazie danych. W tym rozdziale zaprezentowana zostanie implementacja endpointu PUT `api/bnb/update`, służącego do aktualizacji danych dotyczących noclegów aktualnie zalogowanego użytkownika. Endpoint umożliwia aktualizację takich informacji jak: adres, powierzchnia i koszt noclegu.

Proces tworzenia rozpoczyna się od utworzenia niestandardowego hooka useEditBnb, który wykonuje zapytanie do API. Kluczowym elementem jest formularz do edycji danych (np. adres, powierzchnia, koszt), z walidacją, aby zapobiec wysyłaniu niepełnych informacji. Po pomyślnej aktualizacji odświeżamy dane i informujemy użytkownika o sukcesie lub błędzie.

W samym komponencie wykorzystujemy hooki takie jak useState do przechowywania stanu edytowanego noclegu czy też useEffect do reagowania na zmiany stanu, np. po zakończeniu edycji. Formularze obsługujemy przy pomocy useForm z biblioteki React Hook Form, co umożliwia łatwe tworzenie i walidację formularzy. Komponent rozpoczyna się od części skryptowej, natomiast później renderujemy odpowiednią zawartość HTML strony. Zakończenie edycji noclegu, kończy się wyświetleniem odpowiedniego powiadomienia toast, z zewnętrznej biblioteki react-tostify.

Listing 21 - utworzenie niestandardowego hooku useEditBnb (bnb.ts)

```
1
2  export function useEditBnb() {
3   const queryClient = useQueryClient();
4
5   return useMutation({
6     mutationFn: (bnb: BnbType) =>
7       client("bnb/update", {
8         method: "PUT",
```

```
        data: bnb,
      }),
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ["bnbs"] });
      toast.success("Bnb edited successfully");
    },
    onError: () => {
      toast.error("Bnb edit failed");
    },
  });
 }

```

W utworzonym niestandardowym hooku useEditBnb funkcją, która zostaje wywołana podczas mutacji jest client, który przesyła żądanie pod wskazany adres z danymi edytowanego noclegu. W przypadku powodzenia, funkcja onSuccess odświeża listę noclegów, za pomocą invalidateQueries z queryClient, natomiast w przypadku błędu użytkownik zostaje o tym poinformowany poprzez odpowiednią notyfikację.

Listing 22 - implementacja endpointu PUT api/bnb/update (my-hotels.tsx)

```

export const MyHotels = () => {
  const { data: bnbs } = useBnbs();
  const myBnbs = bnbs?.filter(
    (bnb: BnbType) => bnb?.user_id === myUserData?.id
  );
  const { data: bnbs } = useBnbs();
  ...
  const { mutate: editBnb, isSuccess: isSuccessEdit, isPending } = useEditBnb();
  ...
  const [editedBnb, setEditedBnb] = useState<BnbType | null>(null);
  const {
    control: controlEdit,
    handleSubmit: handleSubmitEdit,
    watch: watchEdit,
    reset: resetEdit,
  } = useForm();

  const valuesEditBnb = watchEdit();
  ...
  const handleStartEditBnb = (bnb: BnbType) => {
    setEditedBnb(bnb);
    resetEdit({
      address: bnb?.address,
      space: bnb?.space,
      cost: bnb?.cost,
    });
  };

```

```
  const isEditBnbFormEmpty = () => {
    return (
      !valuesEditBnb.address && !valuesEditBnb.space && !valuesEditBnb.cost
    );
  };

  const onEditBnb = async () => {
    if (isEditBnbFormEmpty()) return;

    if (editedBnb?._id) {
      await editBnb({
        ...editedBnb,
        id: editedBnb._id,
        address: valuesEditBnb?.address || editedBnb?.address,
        space: valuesEditBnb?.space || editedBnb?.space,
        cost: valuesEditBnb?.cost || editedBnb?.cost,
      });
    }
  };

  useEffect(() => {
    if (isSuccessEdit) {
      resetEdit();
      setEditedBnb(null);
    }
    ...
  }, [isSuccessEdit, ...]);

  ...

  return (
    <>
      ...
      <div className="bg-white p-5 md:p-10">
        <p className={styles.heading}>My Hotels</p>
        <div className="flex flex-col space-y-2 pt-5">
          {myBnbs?.map((bnb: BnbType, index: number) => (
            <form key={index} onSubmit={handleSubmitEdit(onEditBnb)}>
              <div
                className={`grid gap-2 grid-cols-[5fr_1fr] place-items-center
justify-items-end ${
                  index !== myBnbs?.length - 1 && "border-b border-stone-200"
                }`}
              >
                <div className="py-1 grid gap-2 grid-cols-[4fr_2fr_2fr_2fr] w-
full">
                  <div className="flex flex-col">
                    <p className={`${styles.paragraph2} text-stone-400`}>
                      Address
                    </p>
                    {editedBnb?._id !== bnb?._id ? (
                      <p className={styles.paragraph}>{bnb?.address}</p>
```

```
                ) : (
                  <InputControl
                    fieldErrorMessage={false}
                    control={controlEdit}
                    name="address"
                    className="pb-2"
                  />
                )}
              </div>
              <div className="flex flex-col">
                <p className={`${styles.paragraph2} text-stone-400`}>
                  Space
                </p>
                {editedBnb?._id !== bnb?._id ? (
                  <p className={styles.paragraph}>
                    {bnb?.space} m<sup>2</sup>
                  </p>
                ) : (
                  <InputControl
                    fieldErrorMessage={false}
                    control={controlEdit}
                    name="space"
                    className="pb-2"
                  />
                )}
              </div>
              <div className="flex flex-col">
                <p className={`${styles.paragraph2} text-stone-400`}>
                  Cost
                </p>
                {editedBnb?._id !== bnb?._id ? (
                  <p className={styles.paragraph}>{bnb?.cost} pln/night</p>
                ) : (
                  <InputControl
                    fieldErrorMessage={false}
                    control={controlEdit}
                    name="cost"
                    className="pb-2"
                  />
                )}
              </div>
              {editedBnb?._id !== bnb?._id && (
                <div className="flex flex-col">
                  <p className={`${styles.paragraph2} text-stone-400`}>
                    Updated
                  </p>
                  <p className={styles.paragraph}>
                    {moment(bnb?.updatedAt).fromNow()}
                  </p>
                </div>
              )}
            </div>
```

```
134                 {editedBnb?._id !== bnb?._id ? (
135                   <div className="flex gap-2.5 text-primary pt-2">
136                     <button onClick={() => handleStartEditBnb(bnb)}>
137                       <IconEdit />
138                     </button>
139                     <button onClick={() => handleDeleteBnb(bnb)}>
140                       <IconDelete />
141                     </button>
142                   </div>
143                 ) : (
144                   <Button disabled={isEditBnbFormEmpty()} isLoading={isPending}>
145                     Save
146                   </Button>
147                 )}
148               </div>
149             </form>
150           ))}
151         </div>
152         ...
153       </div>
154     </>
155   );
156 }
```

Pobieranie danych (linia 3) wykorzystują niestandardowego hooka useBnbs, w którym wykorzystywana jest funkcja useQuery z biblioteki Tanstack Query. Następnie noclegi są filtrowane, aby uzyskać tylko te, które należą do aktualnego użytkownika (linie 4-6).

Użycie hooku useEditBnb (linia 9) jest potrzebne, aby uzyskać funkcje służącą do mutacji danych i potrzebne informacje na temat statusu tego procesu. Stan lokalny editedBnb przechowuje informacje o aktualnie edytowanym mieszkaniu (linia 11).

W liniach 12-17 znajduje się konfiguracja formularza edycji wykorzystująca metody z useForm, takie jak: control, reset, watch i handleSubmit. Wartości zwracane przez funkcję watchEdit() zapisujemy do zmiennej valuesEditBnb (linia 19), aby dynamicznie monitorować dane w formularzu.

Rozpoczęcie edycji następuje po kliknięciu przycisku IconEdit (linia 137), wywołana zostaje wtedy funkcja handleStartEditBnb (linie 21-28). Wewnątrz tej funkcji wartość edytowanego noclegu zostaje ustawiona na nocleg, na którym wywołano edycję, a następnie wartość formularza jest resetowana (resetEdit, linie 51-55), aby uniknąć problemów przy wielokrotnej edycji noclegów bez rerenderowania komponentów.

W liniach 30-34 znajduje się implementacja funkcji isEditBnbFormEmpty, służącej do sprawdzenia czy formularz edycji jest pusty, jest to przydatne w walidacji przed wysłaniem danych.

Podczas próby zakończenia edycji noclegu wywoływana zostaje funkcja onEditBnb (linie 36-48), w której następuje sprawdzenie, czy formularz nie jest pusty, a następnie jeśli nocleg który edytujemy ma przypisane ID, wywołujemy funkcję aktualizującą (editBnb), do której przekazujemy zaktualizowane informacje o noclegu.

Hook useEffect w liniach 50-56 reaguje na zmianę stanu isSuccessEdit. Gdy edycja zakończy się sukcesem, formularz zostaje zresetowany, a aktualnie edytowany nocleg ustawiony na null.

W wartości zwracanej przez JSX (linie 60-155) znajduje się wyświetlenie wszystkich noclegów użytkownika. Gdy edycja nie została jeszcze rozpoczęta, noclegi są wyświetlane w trybie podglądu, gdzie dane, takie jak adres, powierzchnia i koszt, są jedynie widoczne. W przypadku rozpoczęcia edycji, pola formularza przechodzą w tryb edycji, umożliwiając użytkownikowi zmianę danych noclegu. Każdy istniejący nocleg posiada obok siebie ikony, które umożliwiają edycję (ikonka ołówka, linia 137) oraz usunięcie noclegu (ikonka kosza, linia 140).

7. Implementacja aplikacji w Vue.js

7.1. O Vue.js

7.1.1. Opis

Vue to framework JavaScriptowy do tworzenia interfejsów użytkownika. Bazuje na standardowych technologiach HTML, CSS i JavaScript, oferując podobnie jak React deklaratywny model programowania oparty na komponentach. Został zaprojektowany jako elastyczny i progresywny framework, w zależności od potrzeb, Vue oferuje mechanizmy, które można używać na różne sposoby:

- Rozszerzanie statycznego HTML,
- Osadzanie jako komponenty Web Components na dowolnej stronie,
- Tworzenie aplikacji jednostronicowych (SPA),
- Renderowanie po stronie serwera (SSR) w projektach fullstack,
- Generowanie statycznych stron (SSG),
- Kierowanie aplikacji na platformy desktopowe, mobilne, WebGL, a nawet terminale

7.1.2. Wbudowane mechanizmy zarządzania stanem w Vue

Vue.js oferuje kilka wbudowanych mechanizmów do zarządzania stanem, które umożliwiają efektywne zarządzanie danymi w aplikacji. W przeciwieństwie do Reacta, gdzie zarządzanie stanem

opiera się na koncepcji hooków i kontekstu, Vue posiada własne rozwiązania, które są zintegrowane z jego architekturą.

Reaktywność to jedno z kluczowych pojęć w Vue, umożliwiające dynamiczne śledzenie i reagowanie na zmiany w danych aplikacji. Dzięki reaktywnym obiektom/wartościom, które są tworzone przez wbudowane funkcje, Vue podczas zmian w danych automatycznie aktualizuje powiązane z nimi elementy interfejsu użytkownika.

- reactive - funkcja pozwala na tworzenie obiektów reaktywnych, których zmiany są śledzone przez Vue. Umożliwia stworzenie obiektu, którego właściwości będą automatycznie aktualizować widok aplikacji, gdy ulegną zmianie.

Listing 23 - przykład użycia reactive w aplikacji Vue (homepage.vue)

```
1
2   let searchCriteria = reactive<BnbsSearchType>({});
3
```

W Listing 23 następuje stworzenie reaktywnego obiektu, który śledzi zmiany w jego właściwościach. Podczas tworzenia obiektu możemy również wskazać typ, z jakim zgodny musi być nasz obiekt. Początkowa zawartość obiektu została ustawiona jako pusta.

- ref – jest podobny do reactive, ale różni się tym, że jest używany w celu tworzenia pojedynczych wartości reaktywnych. Jest idealny do monitorowania prostych typów danych, takich jak liczby czy stringi.

Listing 24 - przykład użycia ref w aplikacji Vue (homepage.vue)

```
1
2   const selectedBnb = ref<BnbType | null>(null);
3
```

Zmienna selectedBnb będzie śledziła zmiany w swojej wartości. Gdy przypiszemy do niej nową wartość, np. obiekt BnB, Vue automatycznie zaktualizuje interfejs użytkownika wszędzie tam, gdzie ta zmienna jest używana.

- watch – funkcja, która pozwala na ścisłe monitorowanie określonych właściwości reaktywnych. Programista może precyzyjnie wskazać, które dane chce obserwować. W momencie, gdy obserwowana wartość się zmieni, wywołana zostaje funkcja zwrotna (callback), co umożliwia wykonanie dodatkowych operacji.

Listing 25 - przykład użycia watch w aplikacji Vue (my-hotels.vue)

```
1
2   watch(addressFieldAdd, (newValue) => {
```

```
  setAddressFieldAdd(newValue);
});

```

Wykorzystanie watch w Listing 25 umożliwia automatyczne reagowanie na zmiany w zmiennej addressFieldAdd i synchronizowanie jej z stanem w aplikacji za pomocą funkcji setAddressFieldAdd.

- watchEffect – W przeciwieństwie do watch, automatycznie monitoruje wszystkie reaktywne zależności użyte w funkcji. Jest to szczególnie przydatne, gdy zależymy od wielu danych i nie chcemy ręcznie określać, co dokładnie chcemy obserwować. Kiedy którakolwiek z tych zmiennych się zmieni, efekt zostanie ponownie wykonany.

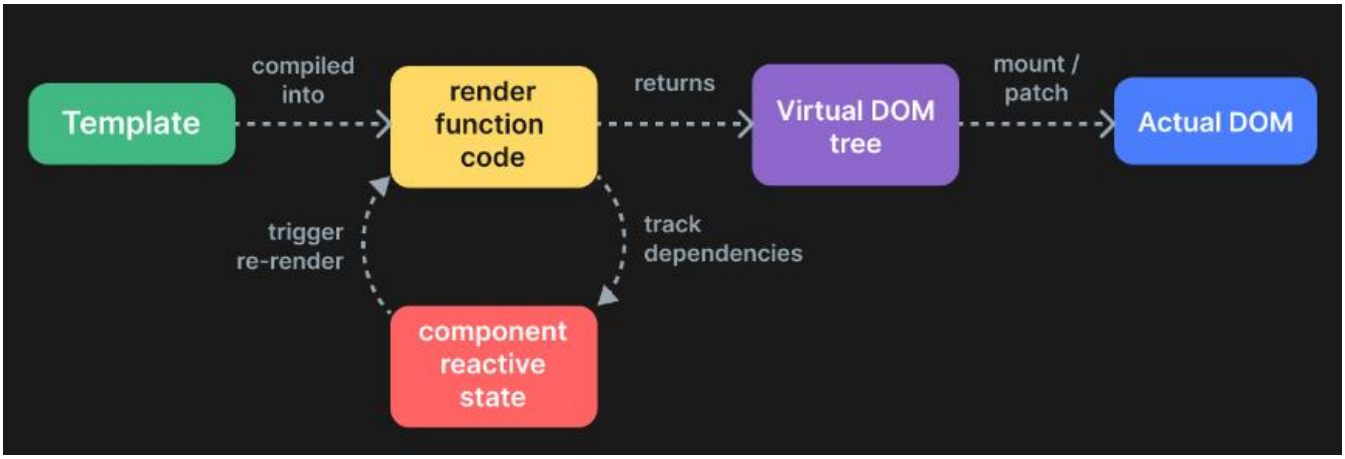
Listing 26 - przykład użycia watchEffect w aplikacji Vue (my-hotels.vue)

```

watchEffect(() => {
 if (bnbs.value && myUserData.value) {
   myBnbs.value = toRaw(bnbs.value).filter(
     (bnb: BnbType) => bnb.user_id === myUserData.value.id
   );
 }
});

```

W Listing 26 watchEffect automatycznie śledzi zależności i wykonuje efekty uboczne w momencie, gdy zmieniają się jakiekolwiek reaktywne zmienne, do których się odwołujemy.

7.1.3. Virtual DOM

Vue również korzysta z koncepcji Virtual DOM, co pozwala na optymalizację procesu renderowania. Zamiast bezpośrednio manipulować rzeczywistym DOM-em, Vue tworzy wirtualną reprezentację drzewa DOM. Dzięki tej optymalizacji, aplikacje stworzone w Vue są w stanie działać szybko, a ich interfejs użytkownika jest płynny.



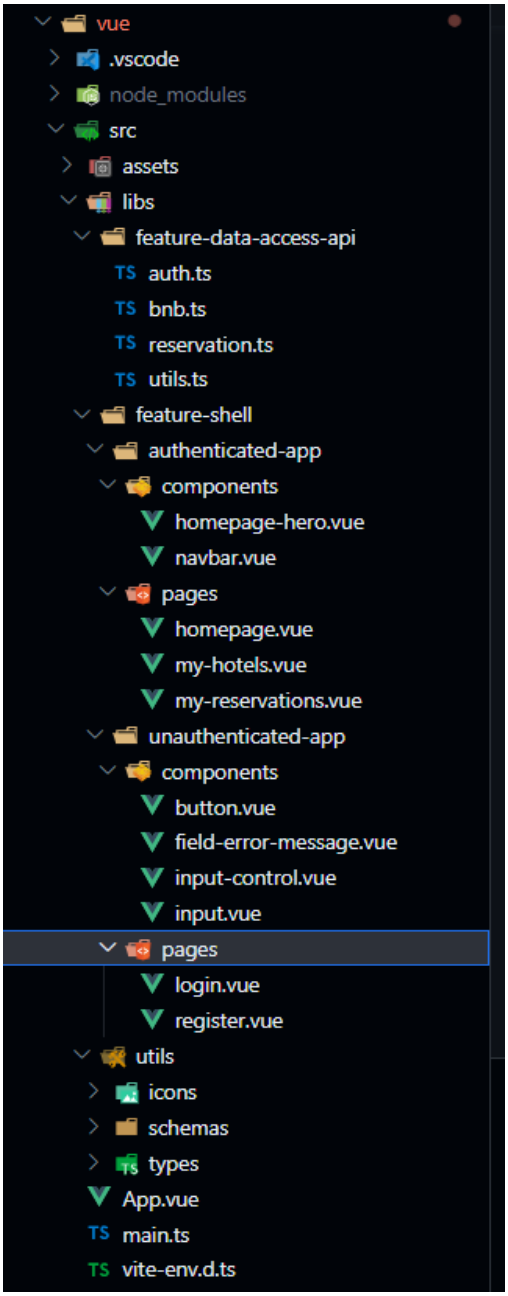
Rysunek 10 - sposób działania Virtual DOM w Vue (oficjalna dokumentacja Vue)

7.1.4. Rozszerzenia plików

Pliki Vue napisane w TypeScript mają rozszerzenie .vue z wbudowanym wsparciem dla TypeScript w sekcji skryptu, podczas gdy w React komponenty w TypeScript używają rozszerzenia .tsx dla plików z JSX oraz .jsx dla plików bez JSX.

7.2. Struktura plików w projekcie

Struktura plików w projekcie Vue została zorganizowana w sposób analogiczny do implementacji aplikacji w React, co ułatwia przejrzystość i zarządzanie kodem. Warto zauważyć, że w Vue każdy komponent jest definiowany w pliku z rozszerzeniem .vue, co różni się od podejścia w React, gdzie komponenty są zazwyczaj rozdzielane na pliki .jsx lub .tsx. Pomimo tych niewielkich różnic, ogólna struktura projektu pozostaje spójna. Głównym komponentem w projekcie jest App.vue, natomiast main.ts tworzy aplikację Vue.



Rysunek 11 - struktura plików w aplikacji Vue

7.3. Najważniejsze pliki

7.3.1. Plik main.ts

Listing 27 - main.ts odpowiedzialny za tworzenie aplikacji Vue

```
import { VueQueryPlugin } from "@tanstack/vue-query";
...

const unauthenticatedRoutes = [
  {
    path: "/",
    component: Login,
  },
  {
    path: "/login",
```

```
    component: Login,
  },
  {
    path: "/register",
    component: Register,
  },
];

const authenticatedRoutes = [
  {
    path: "/",
    component: Homepage,
  },
  {
    path: "/my-hotels",
    component: MyHotels,
  },
  {
    path: "/my-reservations",
    component: MyReservations,
  },
];

const token = localStorage.getItem("token");

 const router = createRouter({
  history: createWebHistory(),
  routes:
    token && token !== "undefined"
      ? authenticatedRoutes
      : unauthenticatedRoutes,
 });

 createApp(App)
  .use(VueQueryPlugin)
  .use(Vue3Toastify, {
    autoClose: 2500,
    position: "top-center"
  } as ToastContainerOptions)
  .use(VCalendar, {})
  .use(router)
  .mount("#root");

```

W Listing 27 main.ts mamy konfigurację i inicjalizację aplikacji Vue. Importowane są różne moduły i style, w tym VueQueryPlugin do zarządzania zapytaniami oraz vue-router do obsługi routingu. Definiowane są dwie grupy tras: jedna dla użytkowników niezalogowanych, zawierająca komponenty logowania i rejestracji, oraz druga dla zalogowanych, z komponentami takimi jak strona główna, moje hotele i moje rezerwacje. Aplikacja sprawdza, czy w localStorage znajduje się token, co

wpływa na wybór tras. Następnie tworzony jest router i aplikacja jest montowana w elemencie DOM o identyfikatorze #root.

W logice routingu istnieje wyraźna różnica między React a Vue. W Vue, vue-router pozwala na określenie podczas inicjalizowania aplikacji tylko jednego zbioru tras, co jest szczególnie widoczne w aplikacjach, w których chcielibyśmy dopasowywać trasy w zależności od stanu aplikacji, np. unauthenticated-app i authenticated-app. Z kolei React umożliwia dynamiczne dopasowywanie tras. W Vue logika routingu została przeniesiona na wyższy poziom, konfigurując odpowiednie trasy na podstawie danych z localStorage. Wnioskując, podejście do zarządzania trasami w Vue jest bardziej zcentralizowane, co może ułatwić organizację kodu, ale ogranicza elastyczność w porównaniu do dynamicznego podejścia w React.

7.3.2. Plik App.vue

Listing 28 - komponent główny App.vue

```
1
2  <template><RouterView /></template>
3
```

W pliku App.vue główny komponent aplikacji pełni rolę kontenera dla innych komponentów renderowanych w zależności od aktualnej trasy. RouterView to specjalny komponent z biblioteki vue-router, który umożliwia dynamiczne wyświetlanie komponentów na podstawie zdefiniowanych tras.

7.4. Uwierzytelnianie w aplikacji

W celu autentykacji stworzyliśmy funkcję useLogin (Listing 29 – funkcja useLogin służąca do zalogowania (auth.ts), która korzysta z mechanizmu mutacji do wysyłania żądania logowania. Funkcja używa useMutation, aby wykonać asynchroniczne żądanie HTTP POST do endpointu „auth/login” z danymi logowania (LoginCredentials). W przypadku błędu (np. niepoprawny e-mail lub hasło), wyświetlana jest informacja o błędzie za pomocą toast.error. Jeśli logowanie zakończy się sukcesem, token dostępu (token) jest zapisywany, a następnie strona jest odświeżana (window.location.reload).

Listing 29 – funkcja useLogin służąca do zalogowania (auth.ts)

```
1
2  export function useLogin() {
3    const mutation = useMutation({
4      mutationFn: async (data: LoginCredentials) => {
5        return await client("auth/login", {
6          method: "POST",
7          data,
8        });
9      },
10     onError() {
11       toast.error("Invalid email or password!");
```

```
    },
    onSuccess(data) {
      setAccessToken(data?.token);
      window.location.reload();
    }
  });

  return mutation;
}

```

Następnie używamy funkcji useLogin (Listing 30) w kodzie Vue, aby obsłużyć logowanie użytkownika. W komponencie szablonu wykorzystujemy formularz, który przy wysłaniu wywołuje funkcję handleFormSubmit, przypisaną do zdarzenia @submit.prevent. Dzięki temu obsługujemy wysłanie danych logowania bez przeładowania strony.

Listing 30 - użycie funkcji useLogin (login.vue)

```

<script setup lang="ts">
...
const { mutate } = useLogin();
const handleFormSubmit = handleSubmit((values) => {
  mutate(values);
});
...
</script>

<template>
  <div class="w-full flex">
    <div class="w-full flex flex-col space-y-3 px-20 py-10">
      ...
      <form @submit.prevent="handleFormSubmit" class="space-y-2.5">
        <InputControl
          label="Email address"
          name="email"
          placeholder="mail@example.com"
        />
        <InputControl
          label="Password"
          name="password"
          type="password"
          placeholder="********"
        />
        <Button :isLoading="isLoading" class="w-full"> Sign in </Button>
      </form>
    </div>
    ...
  </div>
</template>

```

7.5. Obsługa formularzy

Listing 31 - obsługa formularza rejestracji (register.vue)

```

<script setup lang="ts">
import { toTypedSchema } from "@vee-validate/zod";
import { useForm } from "vee-validate";
...

const { mutate } = useRegister();

const { handleSubmit } = useForm<RegisterForm>({
  validationSchema: toTypedSchema(registerUserSchema),
});

const handleFormSubmit = handleSubmit((values) => {
  const { password_confirmation, ...restValues } = values;
  mutate(restValues);
});
</script>

<template>
  <div class="w-full flex">
    <div class="w-full flex flex-col space-y-3 px-20 py-10">
      ...
      <form @submit.prevent="handleFormSubmit" class="space-y-2.5">
        <InputControl
          label="Full name"
          name="full_name"
          placeholder="e.g. John Doe"
        />
        <InputControl
          label="Email address"
          name="email"
          placeholder="mail@example.com"
        />
        <InputControl
          label="Password"
          name="password"
          type="password"
          placeholder="********"
        />
        <InputControl
          label="Confirm password"
          name="password_confirmation"
          type="password"
          placeholder="********"
        />
        <Button class="w-full"> Sign in </Button>
      </form>
    </div>
```

```
    ...
  </div>
</template>

```

W Vue do obsługi formularzy używamy biblioteki vee-validate, która w swoim działaniu przypomina react-hook-form implementowany w aplikacji React. Pozwala ona zarządzać formularzami w sposób zorganizowany, z walidacją i obsługą błędów.

W tym przypadku używamy funkcji useForm (linie 9-11), gdzie przekazujemy schemat walidacji, zdefiniowany przy pomocy biblioteki Zod, co zapewnia automatyczną weryfikację poprawności danych. W formularzu przypisujemy zdarzenie @submit.prevent (linia 23) do funkcji handleFormSubmit, która odpowiada za wysyłanie formularza i zapobiega domyślnemu odświeżeniu strony. Funkcja handleSubmit pobiera dane formularza, waliduje je i przetwarza, a następnie przekazuje oczyszczone wartości do funkcji mutate, obsługującej rejestrację użytkownika (linia 15).

W formularzu korzystamy z komponentów InputControl, które służą do wprowadzania danych użytkownika, takich jak pełne imię, adres e-mail, hasło i potwierdzenie hasła. Dzięki temu komponenty te obsługują zarówno walidację, jak i przekazywanie danych do formularza, co ułatwia zarządzanie danymi. Dodatkowo, podobnie jak w React, możemy dynamicznie zarządzać stanem ładowania za pomocą atrybutu isLoading na przycisku, co pozwala na wskazanie, że operacja jest w toku.

7.6. Stylowanie i responsywność

Listing 32 - stylowanie input-control według przekazanych propsów (input-control.vue)

```

<script setup>
...
const props = defineProps({
  name: String,
  placeholder: String,
  label: String,
  class: String,
  type: {
    type: String,
    default: "text",
  },
});
</script>

<template>
  <div :class="twMerge('grid gap-1', props.class)">
    <label :htmlFor="props.name" :class="styles.paragraph2">
      {{ props.label }}
    </label>
    <Input
```

```
      v-bind="props"
      :placeholder="props.placeholder"
      :error="props.error"
      :type="props.type"
      :name="props.name"
      class="w-full"
    />
  </div>
</template>
```

Podobnie jak w React, w Vue również korzystamy z Tailwind CSS do stylowania komponentów, a do zarządzania klasami CSS używamy narzędzia twMerge, które łączy klasy Tailwinda, eliminując konflikty i nadmiarowości. W Listing 32 jest przedstawiony komponent obsługujący kontrolę pól w formularzach, czyli wcześniej wspomniany InputControl.

Definiowanie propsów odbywa się przy użyciu defineProps (linie 4-13), dzięki czemu komponent InputControl staje się bardziej elastyczny i wielokrotnego użytku, ponieważ może przyjmować różne wartości takie jak name, placeholder, label, type czy class. W szablonie wykorzystano v-bind="props", co oznacza dynamiczne przypisanie do podrzędnego komponentu Input wartości, co znacząco upraszcza kod.

Całość komponentu łączy wygodę obsługi formularzy z elastycznym zarządzaniem stylami, co ułatwia jego wielokrotne użycie w różnych miejscach aplikacji.

7.7. Implementacja analogicznego endpointu do React w Vue

W tym rozdziale przedstawiona zostanie implementacja endpointu PUT `api/bnb/update` w Vue, analogicznie jak w rozdziale 6.8, dotyczącym implementacji tego endpointu w React, który służy do aktualizacji danych noclegów zalogowanego użytkownika, takich jak adres, powierzchnia i koszt noclegu. Implementacja w Vue zachowuje analogiczną funkcjonalność, jednak różni się strukturą kodu oraz wykorzystaniem specyficznych dla Vue narzędzi i bibliotek.

Listing 33 - utworzenie niestandardowego hooku useEditBnb (bnb.ts)

```

export function useEditBnb() {
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: (bnb: BnbType) =>
      client("bnb/update", {
        method: "PUT",
        data: bnb,
      }),
    onSuccess: () => {
```

```
      queryClient.invalidateQueries({ queryKey: ["bnbs"] });
      toast.success("Bnb edited successfully");
    },
    onError: () => {
      toast.error("Bnb edit failed");
    },
  });
}

```

Kod 2-19 jest praktycznie identyczny z implementacją React, z tą różnicą, że w Vue mutacja jest dostępna w ramach composable (kawałki kodu, które zawierają logikę aplikacji, możliwą do wielokrotnego wykorzystania w różnych komponentach Vue) zamiast niestandardowego hooka.

```
<script setup lang="ts">
import { hotelRoom, styles } from "@firebnb/public";
import moment from "moment";
import Navbar from "../components/navbar.vue";
import Button from "../../unauthenticated-app/components/button.vue";
import InputControl from "../../unauthenticated-app/components/input-control.vue";
import { ref, watch, onMounted, onUnmounted, toRaw, watchEffect } from "vue";
import {
  useBnbs,
  useDeleteBnb,
  useEditBnb,
  useAddBnb,
} from "../../../feature-data-access-api/bnb";
import { useUser } from "../../../feature-data-access-api/auth";
import { BnbType } from "../../../utils/types/bnb";
import { useField, useForm } from "vee-validate";
import { IconAdd, IconDelete, IconEdit } from "../../../utils/icons";
```

Od linii 1 rozpoczyna się sekcja <script>, czyli blok kodu, który zawiera logikę komponentu. Można w niej importować moduły, definiować zmienne, funkcje, korzystać z reaktywności i wiele innych. W liniach 2-17 znajdują się importy potrzebnych modułów.

```
const { data: bnbs } = useBnbs();
const { mutate: deleteBnb } = useDeleteBnb();
const { mutate: editBnb, isSuccess: isSuccessEdit, isPending } = useEditBnb();
const { mutate: addBnb, isSuccess: isSuccessAdd } = useAddBnb();
const { data: myUserData } = useUser();

const myBnbs = ref<BnbType[]>([]);
watchEffect(() => {
  if (bnbs.value && myUserData.value) {
    myBnbs.value = toRaw(bnbs.value).filter(
      (bnb: BnbType) => bnb.user_id === myUserData.value.id
    );
  }
});
```

```
32
```

Podobnie jak w React, dane są pobierane za pomocą niestandardowych hooków (composable, linia 18 i 22). Mutacje (linie 19-21) są używane do aktualizacji danych po stronie klienta oraz automatycznego odświeżania widoku po udanej operacji.

Kod 25-32 obsługuje filtrowanie, aby wybrać tylko noclegi należące do aktualnie zalogowanego użytkownika. Wykorzystywana jest funkcja watchEffect, która automatycznie reaguje na zmiany w zależnościach (tutaj bnbs i myUserData). Jest to podobne do Reactowego hooka useEffect, który również monitoruje zmiany w zależnościach i wykonuje kod w odpowiedzi na te zmiany. Różnica polega na tym, że watchEffect automatycznie śledzi używane zmienne, bez potrzeby jawnego ich deklarowania jako zależności, co upraszcza kod.

```
33 const editedBnb = ref<BnbType | null>(null);
34
35 const { resetForm: resetEdit } = useForm();
36 const { resetForm: resetAdd } = useForm();
37
38 const { value: addressFieldAdd, setValue: setAddressFieldAdd } =
39   useField("address_add");
40 const { value: spaceFieldAdd, setValue: setSpaceFieldAdd } =
41   useField("space_add");
42 const { value: costFieldAdd, setValue: setCostFieldAdd } = useField("cost_add");
43
44 const { value: addressFieldEdit, setValue: setAddressFieldEdit } =
45   useField("address_edit");
46 const { value: spaceFieldEdit, setValue: setSpaceFieldEdit } =
47   useField("space_edit");
48 const { value: costFieldEdit, setValue: setCostFieldEdit } =
49   useField("cost_edit");
50
```

Linia 33 tworzy reaktywną zmienną editedBnb, która przechowuje dane edytowanego noclegu lub wartość null, jeśli edycja jest wyłączona. Linie 35 i 36 tworzą funkcje resetEdit i resetAdd, które resetują formularze odpowiednio dla edycji i dodawania nowych noclegów. Kod 38-43 tworzy pola formularza dla dodawania nowego noclegu, każde z pól ma funkcję setValue, która umożliwia dynamiczną zmianę wartości danego pola. Linie 44-49 są analogiczne dla formularza edycji wybranego noclegu.

```
51
52 watch(addressFieldAdd, (newValue) => {
53   setAddressFieldAdd(newValue);
54 });
55 watch(spaceFieldAdd, (newValue) => {
56   setSpaceFieldAdd(newValue);
57 });
```

```
watch(costFieldAdd, (newValue) => {
  setCostFieldAdd(newValue);
});

watch(addressFieldEdit, (newValue) => {
  setAddressFieldEdit(newValue);
});
watch(spaceFieldEdit, (newValue) => {
  setSpaceFieldEdit(newValue);
});
watch(costFieldEdit, (newValue) => {
  setCostFieldEdit(newValue);
});

```

Kod 51-71 zawiera tworzenie funkcji watch, która monitoruje zmiany wartości pól formularza i automatycznie aktualizuje ich wartość za pomocą odpowiednich funkcji setValue. Dzięki temu dane w formularzu są zawsze zsynchronizowane z reaktywnymi zmiennymi. W React podobną funkcjonalność można zaimplementować za pomocą hooka useEffect.

```

const handleDeleteBnb = async (bnb: BnbType) => {
  await deleteBnb(String(bnb?._id));
};

```

Funkcja handleDeleteBnb asynchronicznie wywołuje mutację deleteBnb i usuwa nocleg na podstawie jego identyfikatora _id. Zmienna _id jest konwertowana na ciąg znaków za pomocą String().

```

const handleStartEditBnb = (bnb: BnbType) => {
  editedBnb.value = bnb;
  setAddressFieldEdit(bnb?.address || '');
  setSpaceFieldEdit(bnb?.space || '');
  setCostFieldEdit(bnb?.cost || '');
};

```

Funkcja handleStartEditBnb ustawia dane edytowanego noclegu w formularzu. Wartości pól są inicjalizowane na podstawie istniejących danych lub domyślnie jako puste ciągi znaków.

```

const isEditBnbFormEmpty = () => {
  return (
    !addressFieldEdit.value && !spaceFieldEdit.value && !costFieldEdit.value
  );
};

const submitEditBnb = async () => {
```

```
93    if (isEditBnbFormEmpty()) return;
94
95    if (editedBnb.value?._id) {
96      await editBnb({
97        ...editedBnb.value,
98        id: editedBnb.value._id,
99        address: addressFieldEdit.value || editedBnb.value?.address,
100       space: spaceFieldEdit.value || editedBnb.value?.space,
101       cost: costFieldEdit.value || editedBnb.value?.cost,
102     });
103   }
104 };
105
```

Kod 86-90 zawiera funkcję isEditBnbFormEmpty, która sprawdza, czy formularz edycji jest pusty. Jeśli żadne z pól nie ma wartości, funkcja zwraca true. Funkcja submitEditBnb (linie 92-105) sprawdza, czy formularz edycji nie jest pusty za pomocą isEditBnbFormEmpty(). Jeśli formularz zawiera dane i nocleg posiada identyfikator _id, wykonuje mutację edycji poprzez funkcję editBnb. Dane są aktualizowane na podstawie wartości formularza lub pozostają bez zmian, jeśli pole nie zostało zmienione.

```
106
107 const isAddBnbFormEmpty = () => {
108   return !addressFieldAdd.value || !spaceFieldAdd.value || !costFieldAdd.value;
109 };
110
111 const submitAddBnb = async () => {
112   if (isAddBnbFormEmpty()) return;
113
114   await addBnb({
115     address: addressFieldAdd?.value,
116     space: spaceFieldAdd?.value,
117     cost: costFieldAdd?.value,
118     user_id: myUserData.value?.id,
119   });
120 };
121
```

Funkcje isAddBnbFormEmpty oraz submitAddBnb (linie 106-121) są analogiczne, służą do wysłania danych formularza dodawania noclegu.

```
122
123 watch([isSuccessEdit, isSuccessAdd], ([isSuccessEdit, isSuccessAdd]) => {
124   if (isSuccessEdit) {
125     resetEdit();
126     editedBnb.value = null;
127   }
128   if (isSuccessAdd) {
129     resetAdd();
```

```
  }
});

```

Kod 122-132 obserwuje zmiany w wynikach mutacji edycji (isSuccessEdit) i dodawania (isSuccessAdd). Jeśli któraś z operacji zakończy się sukcesem, następuje zresetowanie odpowiedniego formularza.

```

const handleKeyDown = (event: KeyboardEvent) => {
  if (event.key === "Escape") {
    editedBnb.value = null;
  }
};

onMounted(() => {
  window.addEventListener("keydown", handleKeyDown);
});

onUnmounted(() => {
  window.removeEventListener("keydown", handleKeyDown);
});
</script>

```

Fnkcja handleKeyDown (linie 134-138) nasłuchuje naciśnięcia klawisza Escape, resetując stan edycji, gdy użytkownik chce anulować operację. Kod 140-146 dodaje nasłuchiwanie zdarzenia klawiatury po zamontowaniu komponentu (onMounted) oraz usuwa nasłuchiwanie zdarzenia po odmontowaniu komponentu (onUnmounted). W lini 147 następuje zakończenie części skryptowej </script>, po której zaraz pojawi się część szablonowa <template> zawierająca strukturę HTML komponentu.

```

<template>
  ...

  <div class="bg-white p-5 md:p-10">
    <p :class="styles.heading">My Hotels</p>
    <div class="flex flex-col space-y-2 pt-5">
      <form
        v-for="(bnb, index) in myBnbs"
        :key="index"
        @submit.prevent="submitEditBnb"
      >
        <div
          :class="[
            'grid gap-2 grid-cols-[5fr_1fr] place-items-center justify-items-end',
            index !== myBnbs?.length - 1 ? 'border-b border-stone-200' : '',
```

```
          ]"
        >
          <div class="py-1 grid gap-2 grid-cols-[4fr_2fr_2fr_2fr] w-full">
            <div class="flex flex-col">
              <p :class="[styles.paragraph2, 'text-stone-400']">Address</p>
              <p v-if="editedBnb?._id !== bnb._id" :class="styles.paragraph">
                {{ bnb?.address }}
              </p>
              <InputControl v-else name="address_edit" class="pb-2" />
            </div>
            <div class="flex flex-col">
              <p :class="[styles.paragraph2, 'text-stone-400']">Space</p>
              <p v-if="editedBnb?._id !== bnb._id" :class="styles.paragraph">
                {{ bnb?.space }} m<sup>2</sup>
              </p>
              <InputControl v-else name="space_edit" class="pb-2" />
            </div>
            <div class="flex flex-col">
              <p :class="[styles.paragraph2, 'text-stone-400']">Cost</p>
              <p v-if="editedBnb?._id !== bnb._id" :class="styles.paragraph">
                {{ bnb?.cost }} pln/night
              </p>
              <InputControl v-else name="cost_edit" class="pb-2" />
            </div>
            <div v-if="editedBnb?._id !== bnb._id" class="flex flex-col">
              <p :class="[styles.paragraph2, 'text-stone-400']">Updated</p>
              <p :class="styles.paragraph">
                {{ moment(bnb?.updatedAt).fromNow() }}
              </p>
            </div>
          </div>
          <div
            v-if="editedBnb?._id !== bnb._id"
            class="flex gap-2.5 text-primary pt-2"
          >
            <button @click="handleStartEditBnb(bnb)">
              <IconEdit />
            </button>
            <button @click="handleDeleteBnb(bnb)">
              <IconDelete />
            </button>
          </div>
          <div v-else>
            <Button :disabled="isEditBnbFormEmpty()" :isLoading="isPending">
              Save
            </Button>
          </div>
        </div>
      </form>
    </div>
```

Fragment kodu 153-215 przedstawia dynamiczny formularz do zarządzania noclegami użytkownika. Wykorzystuje pętlę v-for do iteracji po liście noclegów (myBnbs), generując dla każdego z nich osobny formularz. Formularze obsługują zarówno tryb podglądu, jak i edycji. W trybie podglądu dane takie jak adres, powierzchnia i koszt są wyświetlane jako tekst, natomiast w trybie edycji zastępowane są przez pola formularza umożliwiające wprowadzanie zmian.

Każdy formularz zawiera także sekcję z przyciskami do edycji i usuwania. Kliknięcie przycisku edycji przełącza dany formularz w tryb edycji, natomiast kliknięcie przycisku usuwania wywołuje funkcję usuwającą dany nocleg. Z kolei po przejściu w tryb edycji użytkownik może zatwierdzić zmiany, naciskając przycisk "Save".

Kod dynamicznie dostosowuje klasy CSS dla układu siatki, wykorzystując funkcje reaktywności Vue, takie jak v-if do warunkowego renderowania elementów. Formularz jest podzielony na sekcje odpowiadające poszczególnym polom danych, a dla każdego pola sprawdzane jest, czy użytkownik edytuje aktualny element, czy tylko go przegląda.

W porównaniu do React, Vue korzysta z dyrektyw takich jak v-for i v-if, które są wbudowane w składnię frameworka, podczas gdy w React podobne funkcjonalności realizuje się za pomocą metod JavaScript, takich jak map() i warunkowe renderowanie przez operatory logiczne. Różnica polega również na obsłudze klas CSS – Vue dynamicznie zarządza nimi przez :class, a w React stosuje się do tego biblioteki lub funkcje łączenia klas.

```
216
217     <form
218       @submit.prevent="submitAddBnb"
219       class="grid grid-cols-[5fr_1fr] place-items-center justify-items-end"
220     >
221       <div class="py-1 grid gap-2 grid-cols-[4fr_2fr_2fr] w-full">
222         <div class="flex flex-col">
223           <p :class="[styles.paragraph2, 'text-stone-400']">Address</p>
224           <InputControl name="address_add" class="pb-2" />
225         </div>
226         <div class="flex flex-col">
227           <p :class="[styles.paragraph2, 'text-stone-400']">Space</p>
228           <InputControl name="space_add" class="pb-2" />
229         </div>
230         <div class="flex flex-col">
231           <p :class="[styles.paragraph2, 'text-stone-400']">Cost</p>
232           <InputControl name="cost_add" class="pb-2" />
233         </div>
234       </div>
235       <button
236         type="submit"
237         class="text-primary hover:opacity-80"
```

```
        :title="isAddBnbFormEmpty() ? 'First fill all fields' : undefined"
      >
        <IconAdd />
      </button>
    </form>
  </div>
</template>
```

Fragment kodu 217-243 przedstawia formularz do dodawania nowych noclegów. Formularz zawiera trzy pola — adres, powierzchnię oraz koszt — każde z nich renderowane przy użyciu komponentu InputControl, który jest niestandardowym komponentem do obsługi pól wejściowych z wbudowaną walidacją.

Cały formularz jest osadzony w siatce CSS, co zapewnia estetyczne rozmieszczenie elementów. Każde pole danych jest pogrupowane w sekcje z etykietami, które opisują jego funkcję. Formularz jest połączony z funkcją submitAddBnb, która zostaje wywołana po naciśnięciu przycisku "Submit", dzięki czemu dane są przesyłane tylko po wcześniejszej walidacji. Przycisk dodawania nowego noclegu jest dynamicznie dezaktywowany, jeśli pola formularza są puste, a dodatkowo pokazuje podpowiedź (tooltip) z komunikatem informującym użytkownika, że wymagane jest wypełnienie wszystkich pól.

W porównaniu do React, Vue używa tutaj dyrektywy @submit.prevent, która automatycznie zapobiega domyślnemu odświeżeniu strony, co w React musiałoby być obsłużone jawnie w kodzie, np. przez event.preventDefault() w funkcji obsługującej formularz.

Implementacja endpointu `api/bnb/update` w Vue jest bardzo podobna do tej w React, zarówno pod względem logiki, jak i struktury. Obie wersje opierają się na podobnych krokach, takich jak obsługa formularzy, zarządzanie stanem, walidacja danych oraz przesyłanie zaktualizowanych informacji do API. Różnice wynikają głównie ze składni oraz narzędzi specyficznych dla każdego frameworka.

7.8. Implementacja wybranego endpointu

W dokumentacji nie omówiono jeszcze szczegółowo endpointu DELETE, więc zostanie on omówiony w tym rozdziale. Endpoint api/reservation/delete/<id> pozwala na usunięcie informacji o rezerwacji aktualnie zalogowanego użytkownika. Implementacja rozpoczyna się od stworzenia funkcji, służącej do wysyłania zapytań do API.

Listing 34 - utworzenie funkcji useDeleteReservation (reservation.ts)

```

export function useDeleteReservation() {
  const queryClient = useQueryClient();

```

```
  return useMutation({
    mutationFn: (reservationId: string) =>
      client(`reservation/delete/${reservationId}`, {
        method: "DELETE",
      }),
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ["user-reservations"] });
      toast.success("Reservation deleted successfully");
    },
  });
}

```

Funkcja useDeleteReservation służy do usuwania rezerwacji poprzez HTTP metodę DELETE. Funkcja najpierw uzyskuje dostęp do instancji queryClient, a następnie wykorzystuje hook useMutation, który obsługuje usunięcie rezerwacji. mutationFn definiuje funkcję odpowiedzialną za wykonanie żądania DELETE na endpoint api/reservation/delete/<id>, gdzie <id> to identyfikator rezerwacji. Kiedy mutacja zakończy się sukcesem, prowadzi to do odświeżenia cache'owanej listy rezerwacji użytkownika poprzez invalidateQueries dla klucza ["user-reservations"]. To zapewnia, że dane na interfejsie użytkownika są aktualne. Na koniec, po pomyślnym usunięciu rezerwacji, wyświetlany jest pozytywny komunikat za pomocą toast.success.

Listing 35 - implementacja endpointu DELETE api/reservation/delete/<id> (my-reservations.vue)

```

<script setup lang="ts">
…
const { mutate: deleteReservation } = useDeleteReservation();

const handleDeleteReservation = async (reservation: ReservationType) => {
  await deleteReservation(reservation?.id);
};
</script>

<template>
  …
  <div class="bg-white p-5 md:p-10">
    <p :class="styles.heading">My Reservations</p>
    <div class="flex flex-col space-y-2 pt-5">
      …
      <div
        v-for="(reservation, index) in myReservations?.data"
        :key="index"
        :class="[
          'grid gap-2 grid-cols-[5fr_1fr] place-items-center justify-items-end',
          index !== myReservations?.data?.length - 1
            ? 'border-b border-stone-200'
            : '',
        ]"
```

```
26        >
27          <div class="py-1 grid gap-2 grid-cols-[4fr_2fr_2fr_2fr] w-full">
28            <div class="flex flex-col">
29              <p :class="[styles.paragraph2, 'text-stone-400']">Address</p>
30              <p :class="styles.paragraph">
31                {{ reservation?.bnb?.address }}
32              </p>
33            </div>
34            <div class="flex flex-col">
35              <p :class="[styles.paragraph2, 'text-stone-400']">Space</p>
36              <p :class="styles.paragraph">
37                {{ reservation?.bnb?.space }} m<sup>2</sup>
38              </p>
39            </div>
40            <div class="flex flex-col">
41              <p :class="[styles.paragraph2, 'text-stone-400']">Cost</p>
42              <p :class="styles.paragraph">
43                {{ reservation?.bnb?.cost }} pln/night
44              </p>
45            </div>
46            <div class="flex flex-col">
47              <p :class="[styles.paragraph2, 'text-stone-400']">Updated</p>
48              <p :class="styles.paragraph">
49                {{ moment(reservation?.updatedAt).fromNow() }}
50              </p>
51            </div>
52          </div>
53          <button
54            class="text-primary pt-2"
55            @click="handleDeleteReservation(reservation)"
56          >
57            <IconDelete />
58          </button>
59        </div>
60      </div>
61    </div>
62  </template>
63
```

W Listing 35 komponent wyświetla listę rezerwacji użytkownika oraz umożliwia ich usuwanie za pomocą specjalnej ikonki kosza.

Każda rezerwacja jest wyświetlana za pomocą pętli v-for (linia 18). Obok każdej rezerwacji wyświetlane są informacje takie jak adres, powierzchnia (space), koszt noclegu (cost), a także czas ostatniej aktualizacji (updatedAt) (linie 27-52), który jest sformatowany przy użyciu biblioteki moment.js.

Obok tych informacji znajduje się przycisk z ikoną kosza <IconDelete /> (linia 57), który umożliwia użytkownikowi usunięcie danej rezerwacji. Przy kliknięciu na przycisk uruchamiana jest

funkcja handleDeleteReservation (linia 55), która wywołuje mutację usunięcia rezerwacji za pomocą useDeleteReservation (linie 6-8). Każdy przycisk jest związany z konkretną rezerwacją dzięki przekazaniu jej identyfikatora do funkcji deleteReservation, co pozwala na usunięcie właściwego wpisu.

8. Porównanie frameworków

8.1 Rozmiary projektów

Porównanie rozmiarów projektów w wersji deweloperskiej:

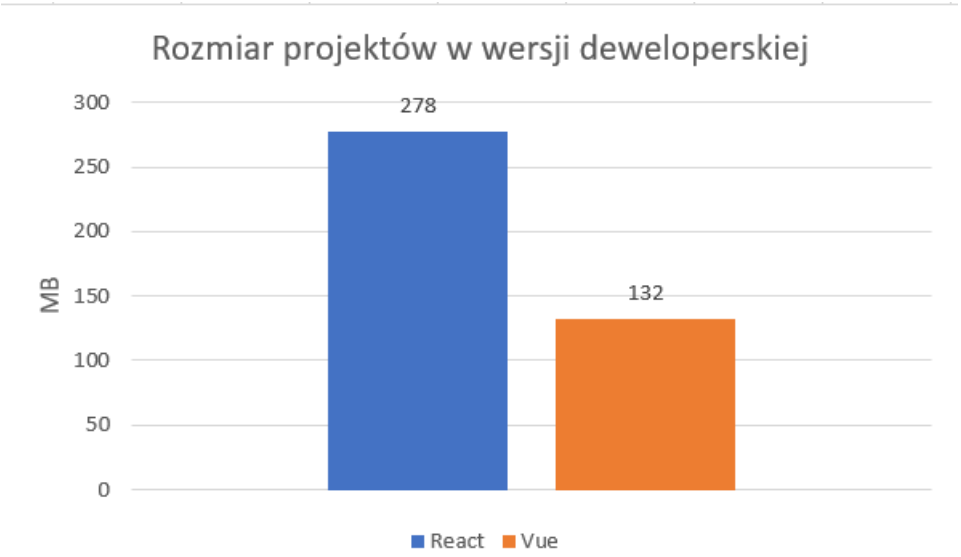


Figure 1 - porównanie rozmiarów projektów w wersji deweloperskiej [MB]

Porównanie rozmiarów projektów w wersji produkcyjnej:

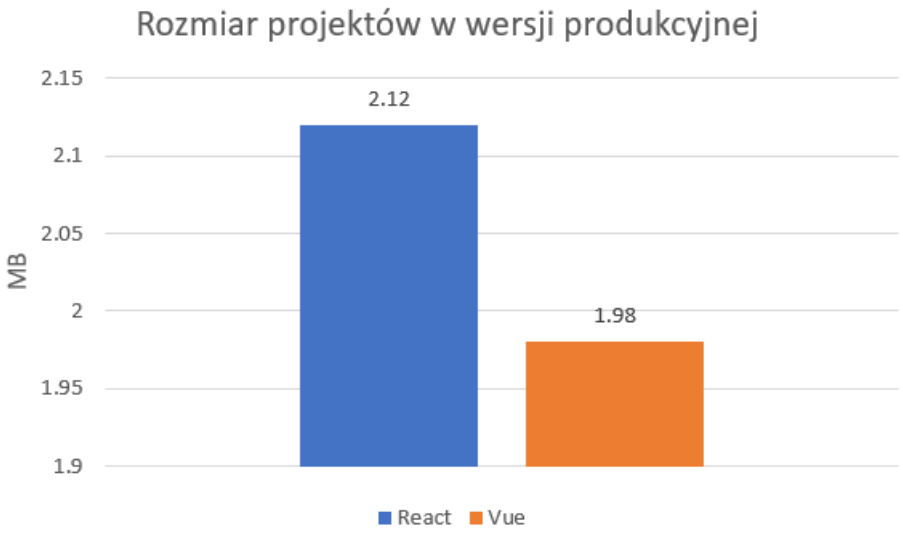


Figure 2 - porównanie rozmiarów projektów w wersji produkcyjnej [MB]

Rozmiar projektu w React i Vue, może różnić się z kilku powodów. Oto kilka kluczowych czynników, które mogą wpływać na różnice w rozmiarze między aplikacjami React i Vue:

Rozmiar Bibliotek i Frameworków:

- **React**: React jest dość rozbudowaną biblioteką i często wymaga dodatkowych bibliotek i narzędzi do pełnego wykorzystania jego możliwości (w projekcie został użyty m.in. react-router, react-hook-form, czy też react-tostify). Każda z tych bibliotek może zwiększać ogólny rozmiar pakietu.

- **Vue**: Vue jest bardziej modularny i często mniejsze rozmiary są wynikiem bardziej zintegrowanego podejścia do routingu, zarządzania stanem i innych funkcji. Vue ma również wbudowane biblioteki do zarządzania stanem i routingu, które mogą być bardziej kompaktowe.

Rozmiar Bundle (skompresowany plik, połączony z wielu plików, np. JS, wykorzystywany do optymalizacji czasu ładowania i wydajności aplikacji webowej)

- **React**: Często projekty React używają różnych narzędzi i bibliotek do zarządzania stanem i efektami, co może prowadzić do większych bundli.

- **Vue**: Vue często oferuje mniejsze bundla dzięki swoim optymalizacjom i efektywnemu zarządzaniu rozmiarem aplikacji (królem w tym rankingu jest Svelte, które posiada rozmiar bundle czasami nawet kilkukrotnie mniejszy od reszty!)

8.2 Dokumentacja i wsparcie społeczności

Dla React dostępność wielu darmowych bibliotek stworzonych przez innych użytkowników stanowi ogromny atut. Na przykład, w projekcie potrzebny był komponent DatePicker; zamiast implementować go od podstaw, w internecie znajduje się wiele w pełni konfigurowalnych wersji, których wykorzystanie znacząco przyspiesza proces pisania aplikacji. Dzięki temu, możemy skupić się na kluczowych funkcjonalnościach naszej aplikacji. Warto również wspomnieć o płatnych bibliotekach, które dostarczają jeszcze więcej funkcjonalności. Te rozwiązania często oferują dodatkowe opcje, wsparcie techniczne oraz bardziej zaawansowane komponenty.

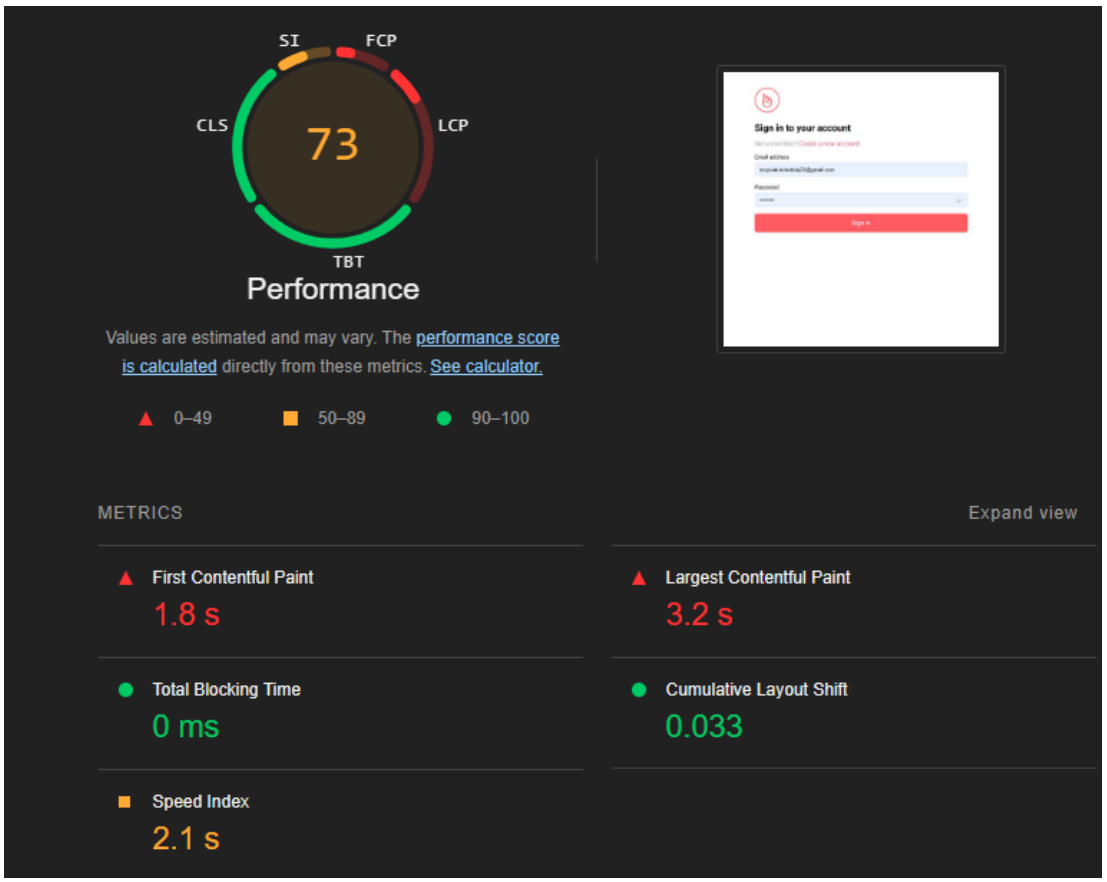
Vue.js, podobnie jak React, może pochwalić się ogromnym wsparciem społeczności i dynamicznie rozwijającą się bazą użytkowników. Choć React jest starszy i cieszy się dłuższą historią, Vue również zdołał zbudować silną i zaangażowaną społeczność. React z kolei, dzięki wsparciu korporacyjnemu (stworzony przez Facebooka) oraz jego ogromnej popularności wśród firm technologicznych, ma jedną z największych i najbardziej aktywnych społeczności na świecie. To prowadzi do tego, że liczba zasobów, wtyczek, bibliotek oraz narzędzi dla Reacta jest ogromna. Zarówno dla Vue, jak i Reacta, dostępnych jest wiele materiałów edukacyjnych, tutoriali, kursów

online, grup dyskusyjnych czy dedykowanych forów. Zasoby dla Reacta są jednak nieco bardziej rozbudowane, co wynika z jego dłuższego istnienia i większej liczby użytkowników. Pisząc aplikacje w Vue, liczba dostępnych bibliotek stworzonych przez społeczność była zauważalnie niższa w porównaniu do Reacta, a wiele bardziej zaawansowanych narzędzi czy komponentów często było płatnych, co czasami bywało ograniczeniem.

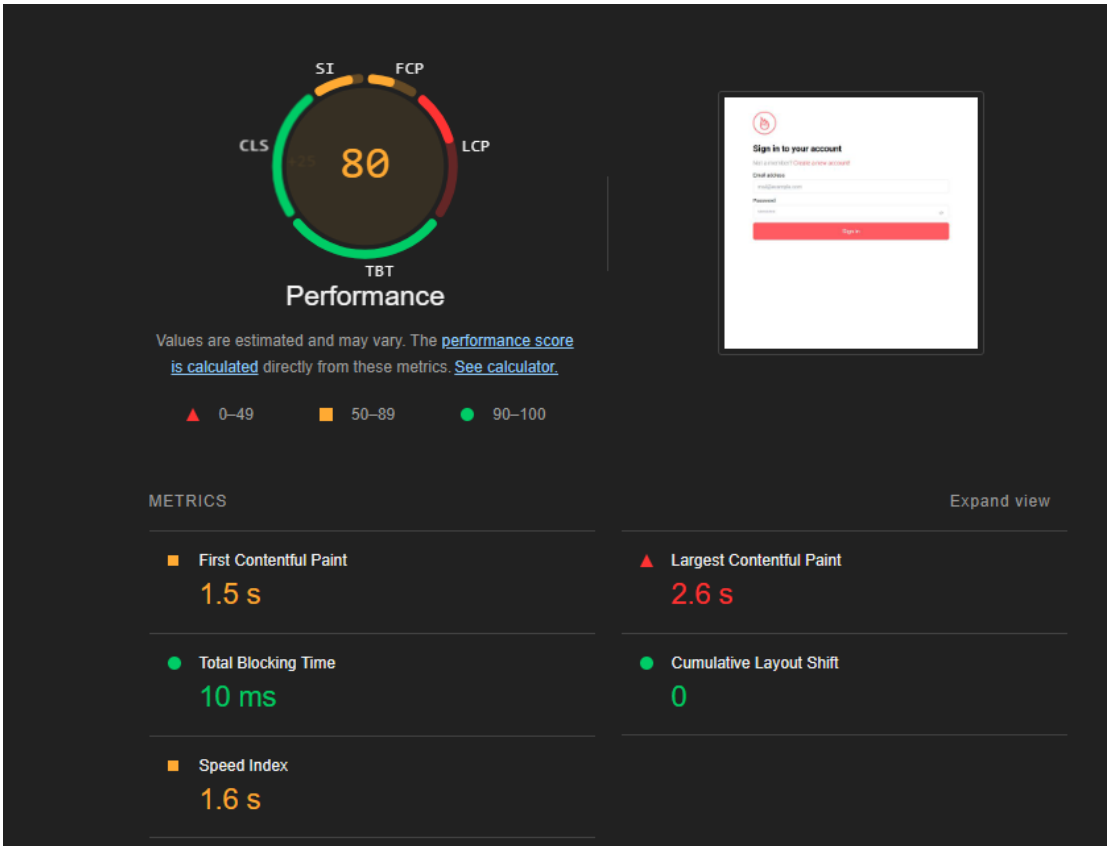
Podsumowując, oba frameworki oferują bogatą bazę materiałów edukacyjnych, kursów online oraz forów dyskusyjnych, jednak dla Reacta zasoby te są niceo bardziej rozbudowane. Warto jednak zauważyć, że Vue, mimo mniejszych zasobów, również dostarcza solidne wsparcie i z roku na rok zyskuje na popularności, co przekłada się na ilość i jakość materiałów w Internecie, czy też książek.

8.3 Porównanie wydajności przy pomocy narzędzia Lighthouse

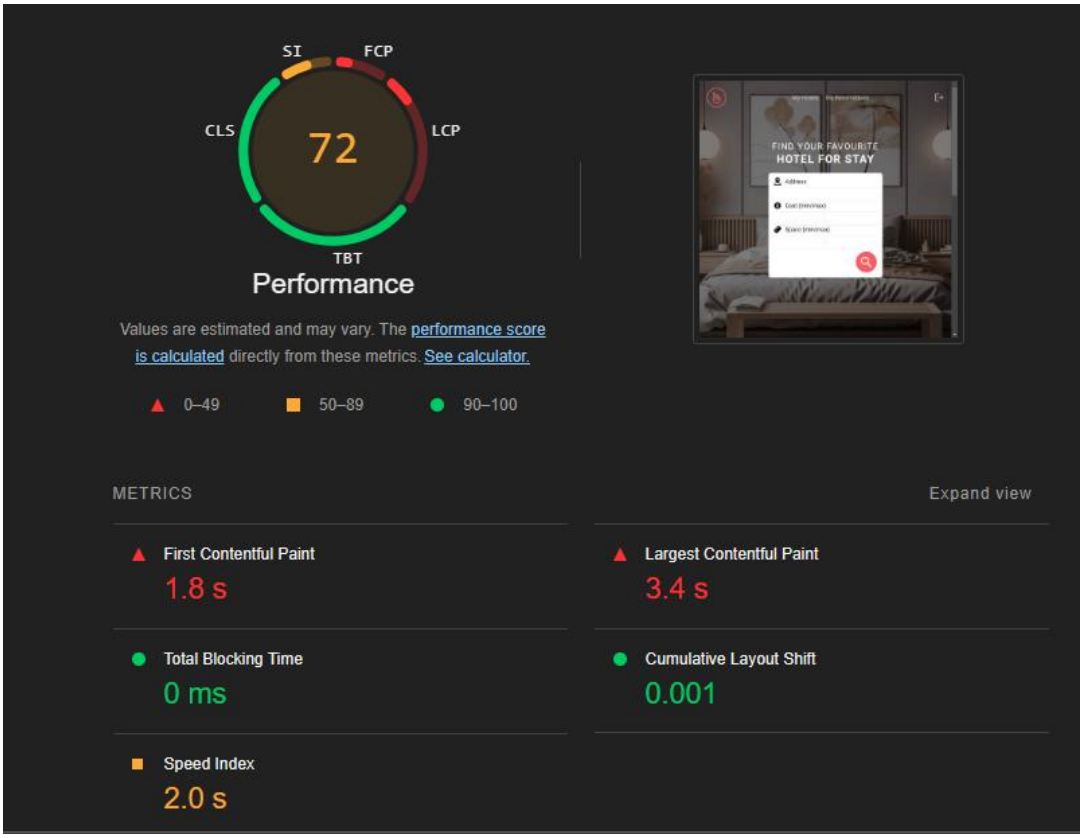
Lighthouse to zautomatyzowane narzędzie o otwartym kodzie źródłowym do audytowania aplikacji webowych, wbudowane w przeglądarki, takie jak Google Chrome. Umożliwia analizę różnych aspektów strony, między innymi: wydajności, dostępności, SEO, czy też najlepszych praktyk. Warto jednak zauważyć, że wyniki audytu nie zawsze są jednoznaczne, ponieważ na wydajność mogą wpływać różne czynniki. Lighthouse jest łatwy w użyciu - wystarczy otworzyć narzędzia deweloperskie w przeglądarce, przejść do zakładki "Lighthouse" i wygenerować raport.



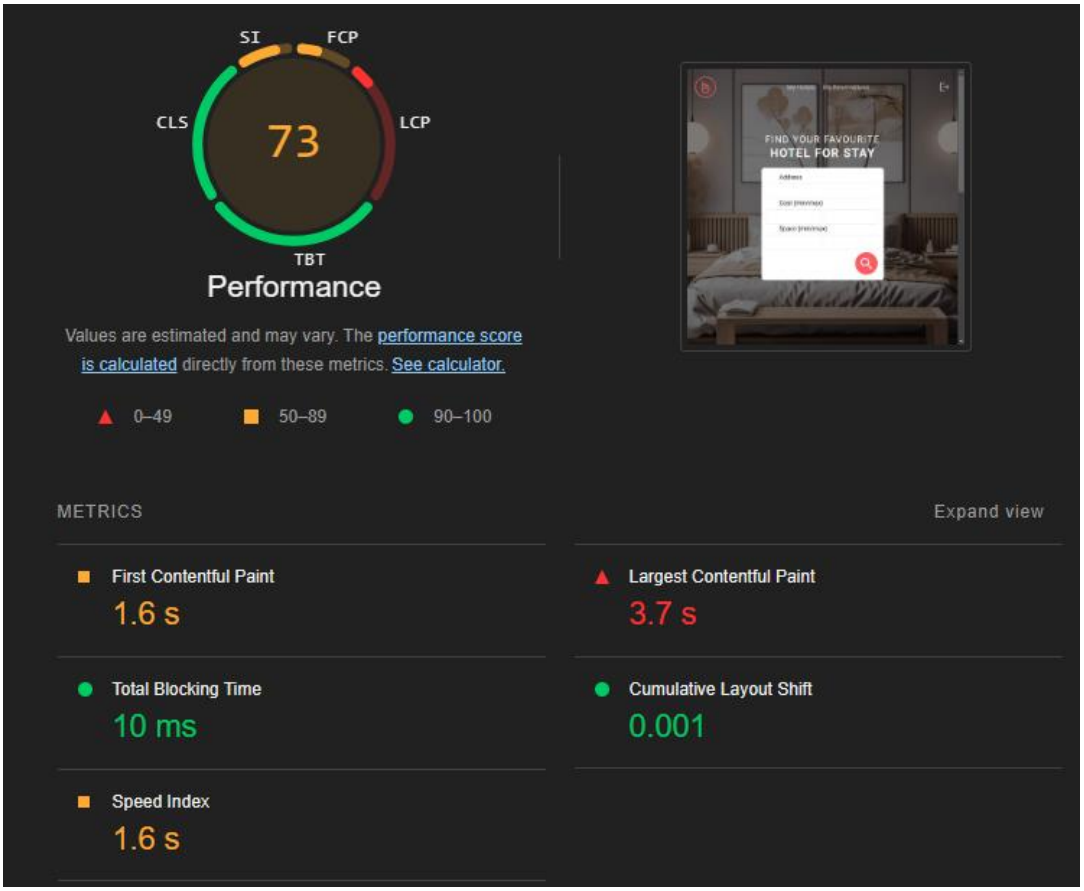
Rysunek 12 - wynik wydajności strony logowania w React



Rysunek 13 - wynik wydajności strony logowania w Vue



Rysunek 14 - wynik wydajności strony głównej w React



Rysunek 15 - wynik wydajności strony głównej w Vue

Obie aplikacje zostały napisane w bardzo podobny sposób i nie zostały specjalnie optymalizowane, aby poprawić wyniki w rankingu. W związku z tym wyniki można rozpatrywać jako naturalne różnice wynikające z użytych technologii.

- Cumulative Layout Shift (CLS): Obie aplikacje osiągnęły ten sam wynik, co sugeruje, że obie są stabilne pod względem zmian układu podczas ładowania strony.
- Speed Index (SI): Vue osiągnęło lepszy wynik (1.6s i 1.6s) w porównaniu do React (2.0s i 2.1s), co wskazuje, że Vue ładuje zawartość strony szybciej.
- First Contentful Paint (FCP): Vue uzyskało lepszy czas (1.6s i 1.5s) niż React (1.8s i 1.8s), co oznacza, że Vue szybciej wyświetla pierwsze elementy treści.
- Largest Contentful Paint (LCP): React jest minimalnie szybszy (3.4 s) w porównaniu do Vue (3.7 s), co sugeruje, że największy element treści jest renderowany szybciej w aplikacji React.

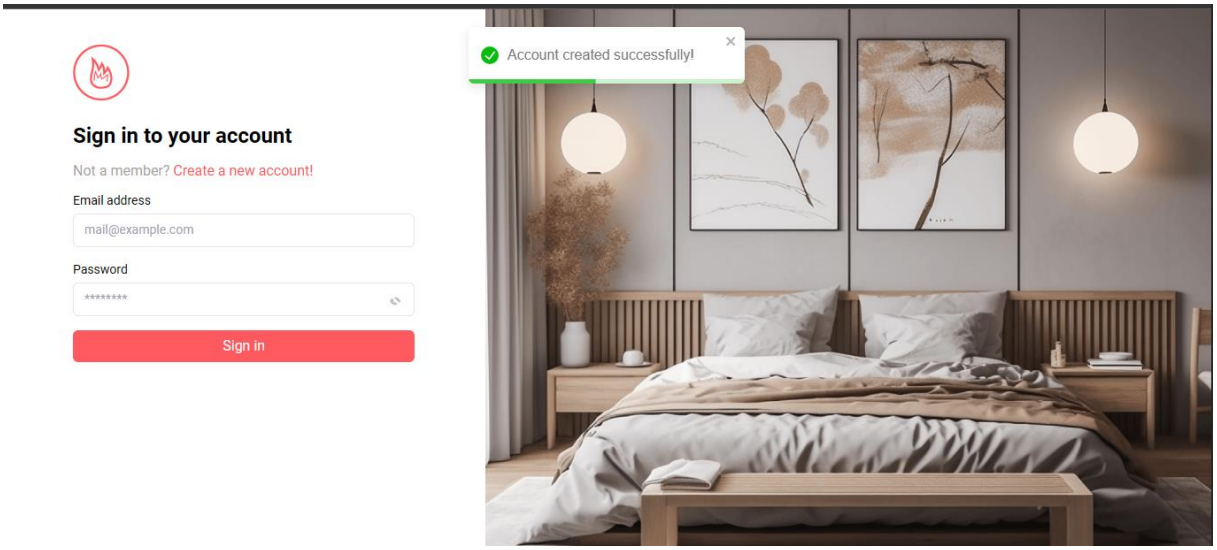
Podsumowując, Vue wykazuje lepsze wyniki w niektórych wskaźnikach wydajności, podczas gdy React jest nieznacznie szybszy w przypadku Largest Contentful Paint. To sugeruje, że Vue może

oferować lepsze wrażenia użytkownika pod względem czasu ładowania treści, ale różnice są na ogół niewielkie i zależne od konkretnej aplikacji oraz jej implementacji.

8.4 Różnice w routingu

Podczas tworzenia aplikacji w Vue pojawił się problem, polegającym na utracie powiadomień typu toast podczas nawigacji. Wynikał z tego, że Vue.js domyślnie usuwa i tworzy na nowo komponenty podczas nawigacji. Toast jest tworzony w komponencie, który jest usuwany podczas nawigacji, dlatego znika. W przeciwieństwie do Reacta, gdzie komponenty mogą pozostać w drzewie komponentów i tylko ich stan może się zmieniać, Vue.js zazwyczaj przeładowuje komponenty w całości.

Zamiast tworzyć powiadomienia typu toast w komponentach, które są usuwane podczas nawigacji, lepszą metodą jest implementacja globalnego systemu powiadomień. W tym celu powiadomienia są tworzone na najwyższym poziomie hierarchii aplikacji (w głównym pliku main.ts). Dzięki temu powiadomienia nie zmieniając się podczas podczas zmiany trasy.



Również warto zaczekać (await), aż nawigacja zostanie zakończone, ponieważ to również może powodować niechciane zachowanie powiadomień.

Listing 36 - rozwiązanie problemu ze znikającymi powiadomieniami (register.tsx)

```
async onSuccess() {
     await router.push("/");
     toast.success("Account created successfully!");
},
```

9. Podsumowanie

W ramach projektu stworzono tę samą aplikację z wykorzystaniem React i Vue, bazując na gotowym API, które wymagało drobnych poprawek. Praca z obiema technologiami była przyjemna i intuicyjna, choć różniły się one nieco pod względem organizacji kodu, podejścia do zarządzania stanem oraz filozofii projektowania komponentów. Pomimo tych różnic, oba frameworki pozwoliły osiągnąć niemal identyczne efekty końcowe, co dowodzi ich wszechstronności i elastyczności.

Vue wyróżnił się nieco lepszą wydajnością w tym projekcie, co jednak nie miało większego znaczenia przy mniejszej skali. W przypadku dużych aplikacji może to być czynnik wart rozważenia. React z kolei zaoferował bogatsze wsparcie społeczności, co przekłada się na łatwiejsze rozwiązywanie problemów i dostęp do wielu dodatkowych narzędzi oraz bibliotek. Wynika to z dłuższej obecności Reacta na rynku oraz jego popularności wśród deweloperów.

Obie technologie mają świetną dokumentację, co znacząco ułatwia naukę i pracę. Próg wejścia w przypadku obu frameworków jest podobny, choć Vue może być nieco bardziej intuicyjne dla osób zaczynających przygodę z front-endem ze względu na bardziej klasyczne podejście do szablonów.

Warto podkreślić, że zarówno React, jak i Vue dynamicznie się rozwijają. Nowe wersje przynoszą nie tylko usprawnienia wydajnościowe, ale również zmiany w podejściu do tworzenia aplikacji. W przyszłości różnice między nimi mogą się jeszcze bardziej zmniejszyć lub mogą również pojawić się nowe funkcjonalności, które zmienią sposób pracy z tymi narzędziami. Wybór między nimi powinien zależeć przede wszystkim od specyfiki projektu i preferencji zespołu, choć nie ma on aż tak wielkiego znaczenia jak jeszcze kilka/kilkanaście lat temu.

Ostatecznie oba frameworki to doskonałe narzędzia, które przy odpowiednim podejściu pozwalają na tworzenie nowoczesnych, skalowalnych i wydajnych aplikacji.

Spis listingów

Spis rysunków

Spis wykresów

Bibliografia

Dokumentacja React.js. (2024). *https://react.dev/*.
Jayesh Shinde. (2020). *https://jayeshinde.medium.com/need-for-virtual-dom-19ebf6843d95*.
Pelechowicz, J. (2024). *REST API w wybranych frameworkach.*
React Hook Form. (2024). *https://react-hook-form.com/docs*.
React Router. (2024). *https://reactrouter.com/en/main*.
Tailwind CSS. (2024). *https://v2.tailwindcss.com/docs*.
Tanstack Query. (2024). *https://tanstack.com/query/latest*.
VeeValidate. (2024). https://vee-validate.logaretm.com/v4/
Vue docs. (2024). https://vuejs.org/guide/introduction

Rzeszów 2024-01-05

Karta wyboru tematu pracy dyplomowej/projektu inżynierskiego
Wydział Elektrotechniki i Informatyki
Wybory zima 2023/24
data wyboru 2024-01-03
KRK/29/7757

Rodzaj pracy: **projekt inżynierski**

Temat pracy: **Różne frameworki do budowy UI na podstawie tego samego API**

Opis tematu:
Praca porównuje różne frameworki do budowy interfejsu użytkownika (UI), wykorzystując tego samego interfejsu API. Celem pracy jest analiza, jak różne narzędzia do tworzenia UI, takie jak React, Angular, Vue.js, mogą być wykorzystane do budowy funkcjonalnych i estetycznych interfejsów użytkownika, korzystając z identycznych danych i usług dostarczanych przez wspólne API. Praca przedstawiać będzie szczegółowy przegląd wybranych frameworków UI, omawiając ich architekturę, modele danych, sposoby renderowania oraz zarządzanie stanem. Kluczowym elementem pracy jest implementacja przykładowych projektów – aplikacji webowej, w której ten sam backend (Docker) oparty na API jest wykorzystywany do budowy kilku różnych interfejsów użytkownika przy użyciu różnych frameworków. Praca ta analizuje i porównuje wydajność, skalowalność, łatwość utrzymania i inne kluczowe aspekty każdego z zastosowanych rozwiązań.

STUDENT	Imię i nazwisko	**Krzysztof Kołodziej**
	Numer albumu	169562
	Kierunek	EF/AA-DI-3(05) Informatyka, studia stacjonarne, I stopnia
OPIEKUN PRACY	Tytuł, imię i nazwisko	dr inż. Tomasz Rak
	Jednostka organizacyjna	Katedra Informatyki i Automatyki (EA)