Politechnika Świętokrzyska w Kielcach

Wydział Elektrotechniki, Automatyki i Informatyki

Technologie Obiektowe

Projekt

Porównanie frameworków React i Vue

Wykonał:

Patryk Gubała

Jakub Hyla

Kielce, 23.06.2025 r.

Spis treści

1. Opis frameworków

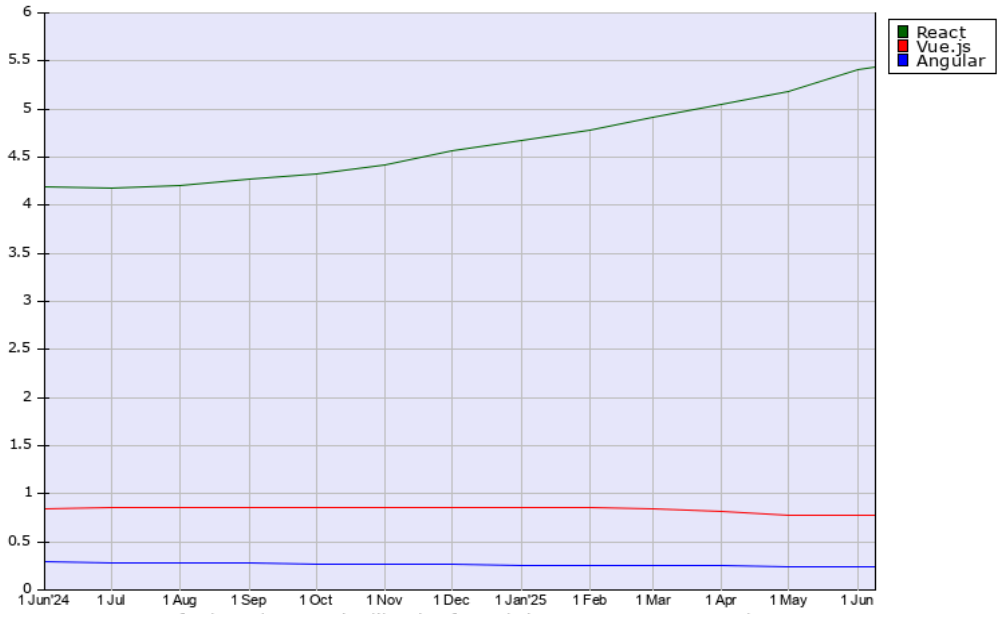
React to framework JavaScript do tworzenia interfejsów użytkownika, która zyskała ogromną popularność wśród programistów na całym świecie. Stworzony przez Jordana Walke'a podczas pracy w Facebooku, React stał się fundamentem wielu znanych aplikacji, takich jak Facebook, Instagram czy WhatsApp. Jego główną zaletą jest połączenie łatwości użycia, efektywności działania i elastyczności, co czyni go idealnym narzędziem do tworzenia nowoczesnych interfejsów użytkownika. Dzięki komponentowej architekturze, programiści mogą tworzyć złożone aplikacje, które są łatwe w utrzymaniu i rozwijaniu. React korzysta z wirtualnego DOM, co znacząco wpływa na wydajność i szybkość ładowania aplikacji, optymalizując interakcje z document object model. [1]

Vue, określany jako "progresywny framework" przez swojego projektanta Evana You, pojawił się na rynku w 2014 roku. Początkowo używany tylko w Chinach, ten popularny framework JavaScript jest obecnie powszechnie wykorzystywany na całym świecie do tworzenia intuicyjnych interfejsów użytkownika i aplikacji typu Single Page Applications (SPA). Vue obsługuje dwukierunkowe wiązanie i wykorzystuje wirtualny DOM. Głównym powodem popularności tego frameworka jest jego progresywny design, pozwalający programistom na ciągłe przenoszenie istniejących projektów. Można to osiągnąć, przesuwając każdą funkcję z osobna. Framework jest również znany z obszernej dokumentacji, co ułatwia naukę i adaptację do frameworka. Vue jest projektem open-source tworzonym przez społeczność, który jest stale ulepszany i rozwijany. BuzzFeed, Grammarly, Dribbble, Zoom, BMW i Nintendo to tylko kilka znanych marek korzystających z Vue. [1]

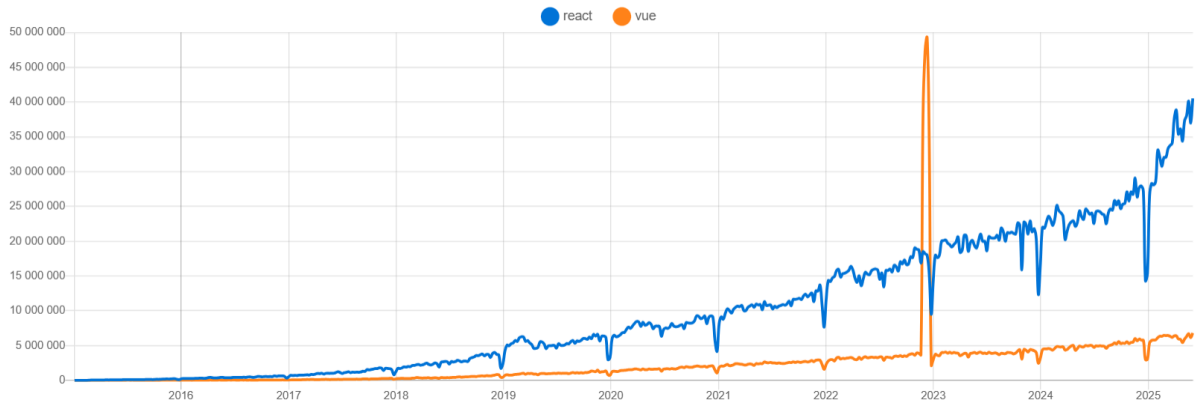
1.1 Popularność frameworków

React ma dużą społeczność, co przekłada się na większą liczbę tutoriali, kursów online, publikacji i prawie czterokrotnie większą liczbę zapytań w serwisie Stackoverflow. Większa społeczność oznacza również większy ekosystem zewnętrznych bibliotek, narzędzi i rozszerzeń. Wspierają ją wszystkie najważniejsze zintegrowane środowiska programistyczne. Facebook stworzył i utrzymuje tę bibliotekę, zapewniając jej długofalowe wsparcie. [1]

Chociaż Vue ma mniejszy udział w rynku, jego baza użytkowników stale rośnie. Nie posiada tak wielu zasobów, pakietów i bibliotek firm trzecich jak React (wiele z nich nie ma angielskiej lokalizacji). Vue jest obsługiwane przez wszystkie główne IDE, ale nie w tak szerokim zakresie jak React. [1]



Rys. 1 Trend użycia jako procent rynku w 2024-2025 [2]

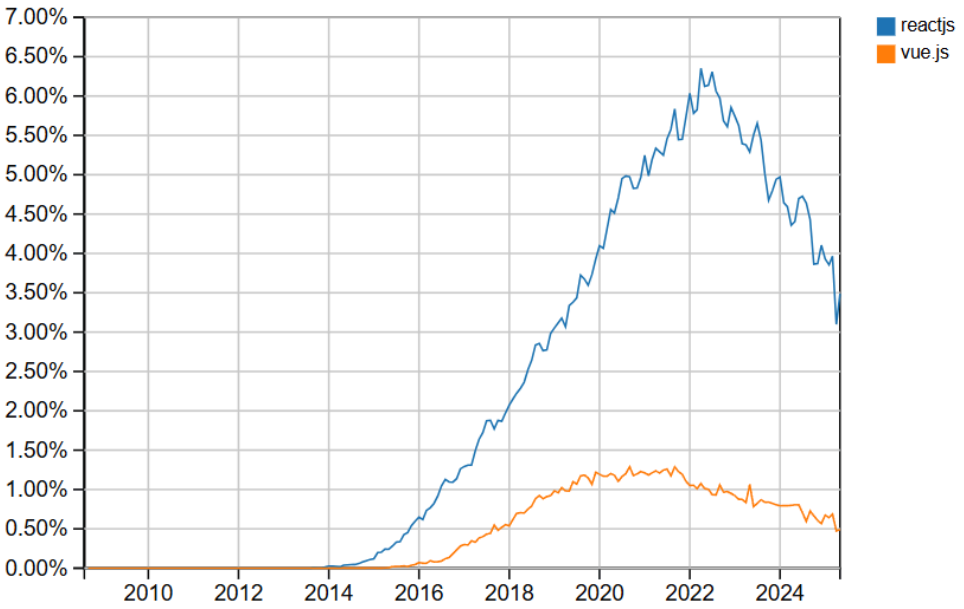


Rys. 2 Trend ilości pobrań paczek w npm [3]

Rys. 3 Popularność względem regionu [4]

	Technologia	Liczba głosów [%]
1.	Node.js	40,8
2.	**React**	**39.5**
3.	jQuery	21,4
4.	Next.js	17,9
5.	Express	17,8
6.	Angular	17,1
7.	ASP.NET CORE	16,9
8.	**Vue.js**	**15,4**
9.	ASP.NET	12.9
10.	Flask	12,9

Tab. 1 Popularność wg głosowania użytkowników na portalu stackoverflow [5]



Rys. 5 Procent zapytań na portalu stackoverflow [6]

1.2 Popyt na rynku pracy

Doświadczenie	Liczba ofert pracy	
	React	Vue
Praktyki / Staż	5	4
Junior	33	9
Mid	766	210
Senior	1315	281
Łącznie	2119	504

Tab. 2 Liczba ofert pracy w Polsce w zależności od doświadczenia na portalu justjoin [7]

Doświadczenie	Mediana zarobków (PLN brutto miesięcznie)			
	React		Vue	
	B2B	UoP	B2B	UoP
Junior	9 654	9 075	-	-
Mid	18 730	15 000	16 500	12 500
Senior	24 450	20 200	21 991	20 250

Tab. 3 Mediana zarobków w Polsce w zależności od doświadczenia wg portalu nofluffjobs [8]

Doświadczenie	Zakres zarobków (PLN brutto miesięcznie)	
	React	Vue
Junior	7 500 - 10 500	6 500 - 9 500
Mid	12 000 - 19 000	10 500 - 15 000
Senior	19 000 - 25 000	16 000 - 22 000
Lead	25 000 - 32 000	22 000 - 28 000

Tab. 4 Zakres zarobków w Polsce w zależności od doświadczenia wg portalu itentio [9]

2. Porównanie rozwiązań

2.1 Komponenty

Komponenty są podstawowym elementem architektury zarówno Reacta, jak i Vue. Oba frameworki wykorzystują komponentowy model budowania interfejsów użytkownika, jednak różnią się podejściem do ich definiowania, organizacji oraz komunikacji między nimi.

W React komponenty są zazwyczaj tworzone jako funkcje lub klasy w języku JavaScript lub TypeScript, często z użyciem JSX – rozszerzenia składni umożliwiającego wstawianie znaczników HTML bezpośrednio w kodzie JS. React opiera się na czystych funkcjach, a zarządzanie stanem i cyklem życia komponentu jest realizowane m.in. przez hooki (useState, useEffect, itp.).

W Vue komponenty są definiowane w plikach .vue, które zawierają trzy sekcje: <template>, <script> i <style>. Pozwala to na przejrzyste rozdzielenie struktury HTML, logiki oraz stylów. Vue wspiera również deklaratywne podejście do definiowania danych, metod i obserwatorów.

Poniżej przedstawione są przykłady w obu technologiach prostego komponentu licznika z przyciskiem inkrementującym wartość stanu.

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

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

  return (
    <div>
      <p>Licznik: {count}</p>
      <button onClick={() => setCount(count + 1)}>Zwiększ</button>
    </div>
  );
}

export default Counter;
```

Listing 1 Przykładowa struktura komponentu w React (.jsx)

```
<template>
  <div>
    <p>Licznik: {{ count }}</p>
    <button @click="count++">Zwiększ</button>
  </div>
</template>

<script>
export default {
  data() {
    return {
      count: 0
    };
  }
}
</script>
```

Listing 2 Przykładowa struktura komponentu w Vue (.vue)

W React podstawową metodą komunikacji między komponentami jest przekazywanie danych z komponentu nadrzędnego do podrzędnego poprzez props. Propsy są tylko do odczytu – komponent dziecko nie może ich bezpośrednio zmieniać. Jeśli komponent potomny ma wpływać na stan komponentu nadrzędnego, przekazuje się mu funkcję zwrotną (callback), która umożliwia wywołanie zmiany z poziomu dziecka. Ten sposób komunikacji jest prosty i czytelny, ale w bardziej złożonych strukturach komponentów może prowadzić do tzw. *prop drilling* – konieczności przekazywania propsów przez wiele warstw komponentów, nawet jeśli pośrednie komponenty ich nie używają.

```
function Parent() {
  const [message, setMessage] = useState("");
  const handleSend = (msg) => setMessage(msg);

  return (
    <div>
      <Child onSend={handleSend} />
      <p>Odebrano: {message}</p>
    </div>
  );
}

function Child({ onSend }) {
  return <button onClick={() => onSend("Wiadomość z
dziecka")}>Wyślij</button>;
}
```

Listing 3 Przykładowe użycie props z callback w React

Innym rozwiązaniem w React jest Context, który pozwala na udostępnienie danych wielu komponentom bez konieczności przekazywania ich przez propsy. Nadaje się do globalnych danych, takich jak ustawienia języka, motywu lub autoryzacji.

```
const UserContext = React.createContext();

function App() {
  return (
    <UserContext.Provider value={{ name: "Jan" }}>
      <Profile />
    </UserContext.Provider>
  );
}

function Profile() {
  const user = useContext(UserContext);
  return <p>Użytkownik: {user.name}</p>;
}
```

Listing 4 Przykładowe użycie Context w React

Podobnie jak w React, Vue wykorzystuje props do przekazywania danych z komponentu rodzica do dziecka. W drugą stronę – czyli do przekazywania informacji z komponentu dziecka do rodzica – Vue używa zdarzeń emitowanych za pomocą metody $emit.

```
<!-- Parent.vue -->
<template>
  <Child @send="handleSend" />
</template>

<script>
import Child from './Child.vue';

export default {
  components: { Child },
  methods: {
    handleSend(msg) {
      console.log("Odebrano:", msg);
    }
  }
}
</script>
```

```
<!-- Child.vue -->
<template>
  <button @click="$emit('send', 'Wiadomość z dziecka')">Wyślij</button>
</template>
```

Listing 5 Przykładowe użycie props i emit w Vue

Vue pozwala również na komunikację przy użyciu innych, bardziej zaawansowanych mechanizmów. Jednym z nich jest mechanizm *provide/inject*, który umożliwia automatyczne przekazywanie danych z komponentu nadrzędnego do dowolnie głęboko zagnieżdżonych komponentów potomnych – z pominięciem komponentów pośrednich, bez potrzeby jawnego ich przekazywania przez props. Dzięki temu można uprościć strukturę aplikacji, eliminując konieczność ręcznego przekazywania danych przez każdy poziom drzewa komponentów. Jest to przydatne np. w przypadku budowania bibliotek komponentów lub udostępniania wspólnego kontekstu bez użycia zewnętrznych narzędzi.

```
<!-- App.vue -->
<script>
export default {
  provide() {
    return {
      theme: 'dark'
    };
  }
}
</script>


<!-- NestedComponent.vue -->
<script>
export default {
  inject: ['theme'],
  mounted() {
    console.log("Motyw:", this.theme);
  }
}
</script>
```

Listing 6 Przykładowe użycie provide/inject w Vue

Różny dla obu technologii jest też cykl życia komponentu, czyli zestaw etapów, przez które przechodzi komponent w trakcie swojego istnienia – od stworzenia, przez aktualizacje, aż po usunięcie z DOM (ang. Virtual Document Object Model). Frameworki pozwalają nam reagować na te etapy, np. ładować dane przy montowaniu albo posprzątać zasoby przy odmontowaniu.

W Vue komponent przechodzi przez jasno zdefiniowane fazy cyklu życia. Możemy reagować na te etapy przy użyciu tzw. *lifecycle hooks* – specjalnych funkcji, które wykonują się automatycznie w odpowiednich momentach.

1. setup() – uruchamiany przed inicjalizacją komponentu (tu definiujemy logikę i dane)
2. onBeforeMount() – tuż przed zamontowaniem w DOM
3. onMounted() – po zamontowaniu w DOM (np. do pobierania danych)
4. onBeforeUpdate() – przed aktualizacją DOM po zmianie stanu
5. onUpdated() – po aktualizacji DOM
6. onBeforeUnmount() – przed zniszczeniem komponentu
7. onUnmounted() – po zniszczeniu (np. do czyszczenia timerów, listenerów)

```
import { onMounted, onUnmounted } from 'vue'

export default {
  setup() {
    onMounted(() => {
      console.log('Komponent został zamontowany')
    })

    onUnmounted(() => {
      console.log('Komponent został usunięty')
    })
  }
}
```

Listing 7 Przykładowe użycie lifecycle hooks w Vue

W React cykl życia komponentu nie jest podzielony na osobne, nazwane metody – tak jak ma to miejsce w Vue. Zamiast tego, reakcję na różne fazy osiągamy za pomocą hooka useEffect(), który umożliwia wykonywanie efektów ubocznych (np. pobieranie danych) po wyrenderowaniu komponentu. Kluczową rolę odgrywa tutaj tablica zależności, przekazywana jako drugi argument do useEffect, która to decyduje kiedy efekt powinien się ponownie wykonać.

1. render – funkcja komponentu jest wywoływana przy każdym odświeżeniu (np. zmianie stanu lub propsów). W tym momencie tworzona jest struktura JSX – drzewo elementów, które React następnie porównuje i aktualizuje w DOM.
2. useEffect(() => { ... }, [...]) – kod wewnątrz hooka useEffect wykonuje się po wyrenderowaniu komponentu. Efekt ten można dopasować do konkretnej fazy cyklu życia poprzez odpowiednie ustawienie zależności w tablicy:
    - [] – efekt wykona się tylko raz, przy pierwszym renderze,
    - [jakaśWartość] – efekt wykona się po pierwszym renderze i za każdym razem, gdy wartość jakaśWartość ulegnie zmianie,
    - brak drugiego argumentu – efekt uruchamia się po każdym renderze niezależnie od tego, co się zmieniło.
3. return () => { ... } wewnątrz useEffect – funkcja zwracana z efektu działa jako mechanizm czyszczenia. Wykonuje się:
    - tuż przed kolejnym wywołaniem efektu (jeśli zmieniły się zależności),
    - lub przy odmontowaniu komponentu.

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

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

  useEffect(() => {
    console.log(`Zmienna "count" zmieniła się: ${count}`);
    return () => {
      console.log(`Czyszczenie efektu dla count = ${count}`);
    };
  }, [count]);

  return <button onClick={() => setCount(count + 1)}>Kliknij
({count})</button>;
}
```

Listing 8 Przykładowe użycie useEffect w React

2.2 Routing

React wykorzystuje zewnętrzną bibliotekę react-router-dom, opartą na komponentach JSX i hookach. Vue korzysta z oficjalnego vue-router, który opiera się na konfiguracji obiektowej i centralnym zarządzaniu dostępem.

W Vue trasy definiuje się jako obiekty w tablicy, gdzie każda trasa zawiera path, komponent oraz opcjonalne dane w polu meta. Lazy loading komponentów odbywa się poprzez funkcję zwracającą import().

```
{
  path: '/admin',
  component: () => import('../pages/Admin/Admin.vue'),
  meta: { requiresAuth: true, roles: ['ADMIN'] }
}
```

Listing 9 Przykładowa definicja trasy w Vue

W React trasy opisuje się przy pomocy komponentów JSX zagnieżdżonych wewnątrz Routes.

```
<Route element={<PrivateRoute requiredRole="ADMIN" />}>
  <Route path="/admin" element={<Admin />} />
</Route>
```

Listing 10 Przykładowe definicja trasy w React

W Vue dostęp do trasy jest sprawdzany globalnie, za pomocą router.beforeEach. Funkcja ta ma dostęp do każdej trasy oraz jej metadanych, dzięki czemu może zweryfikować, czy użytkownik jest zalogowany i czy posiada odpowiednią rolę. Jeśli użytkownik nie spełnia warunków, jest przekierowywany na inną trasę.

```
router.beforeEach((to, from, next) => {
  const authStore = useAuthStore();

  if (to.meta.requiresAuth && !authStore.isAuthenticated) {
    return next({ name: 'Login' });
  }

  if (to.meta.guest && authStore.isAuthenticated) {
    return next({ name: 'Home' });
  }

  if (to.meta.roles && !to.meta.roles.some(role =>
authStore.hasRole(role))) {
    return next({ name: 'Unauthorized' });
  }

  next();
});
```

Listing 11 Kontrola dostępu do tras w Vue

W React analogiczną kontrolę dostępu realizuje komponent PrivateRoute. Wewnątrz komponentu sprawdzany jest stan autoryzacji (isAuthenticated) i ewentualna rola (hasRole). Jeśli użytkownik nie jest zalogowany lub nie posiada odpowiednich uprawnień, zostaje przekierowany za pomocą komponentu Navigate.

```
function PrivateRoute({ requiredRole }) {
  const { isAuthenticated, loading, hasRole } = useAuth();

  if (loading) return <div>Loading...</div>;
  if (!isAuthenticated) return <Navigate to="/login" replace />;
  if (requiredRole && !hasRole(requiredRole)) return <Navigate
to="/unauthorized" replace />;

  return <Outlet />;
}
```

Listing 12 Kontrola dostępu do tras w React

2.3 Zewnętrzne biblioteki

React i Vue nie dostarczają gotowego zestawu komponentów UI – ale oba posiadają wiele rozbudowanych bibliotek zewnętrznych, które zapewniają gotowe przyciski, formularze, tabele, panele boczne itp. Podejścia są podobne, ale różnią się pod względem stylu, integracji z frameworkiem i dostępnych opcji.

```
import { Button } from "@mui/material";

function MyButton() {
  return <Button variant="contained" color="primary">Zapisz</Button>;
}


import { Button } from "antd";

function MyButton() {
  return <Button type="primary">Zapisz</Button>;
}


import { Button } from "@chakra-ui/react";

function MyButton() {
  return <Button colorScheme="blue">Zapisz</Button>;
}
```

Listing 13 Porównanie implementacji przycisku w różnych bibliotekach UI w React

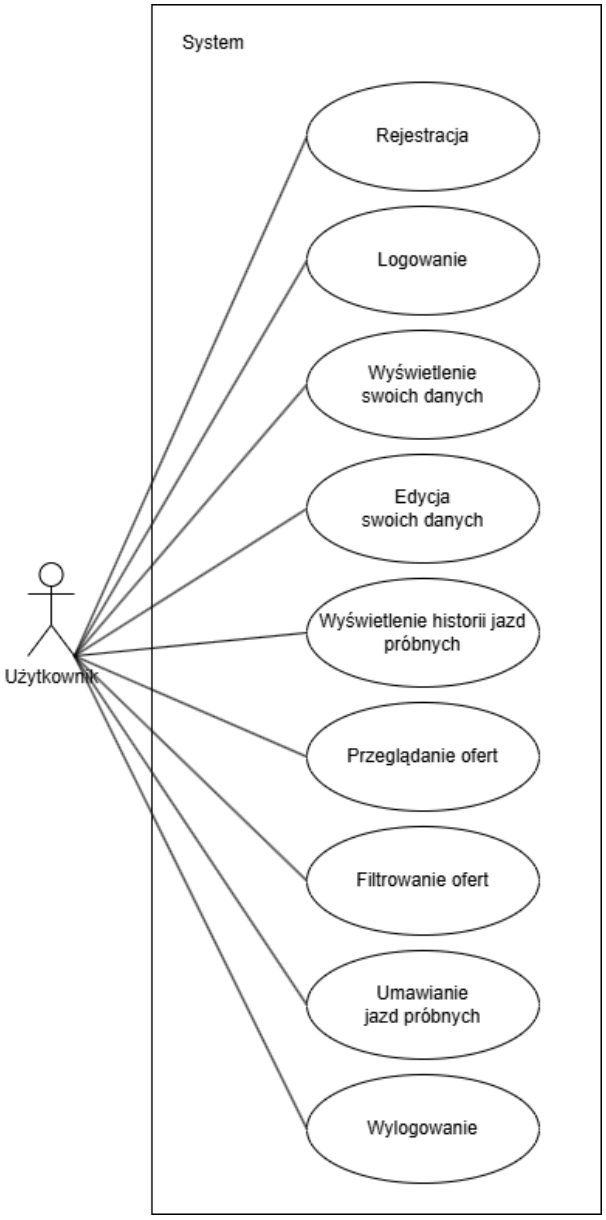
React	**Vue**
MUI	Vuetify
Chakra UI	Quasar
Ant Design	PrimeVue
Radix UI	Element Plus
Shadcn UI	Naive UI
Mantine	**Ant Design Vue**
Blueprint UI	Flowbite Vue 3

Tab. 5 Najpopularniejsze zewnętrzne biblioteki UI dla React i Vue

3. Utworzona aplikacja

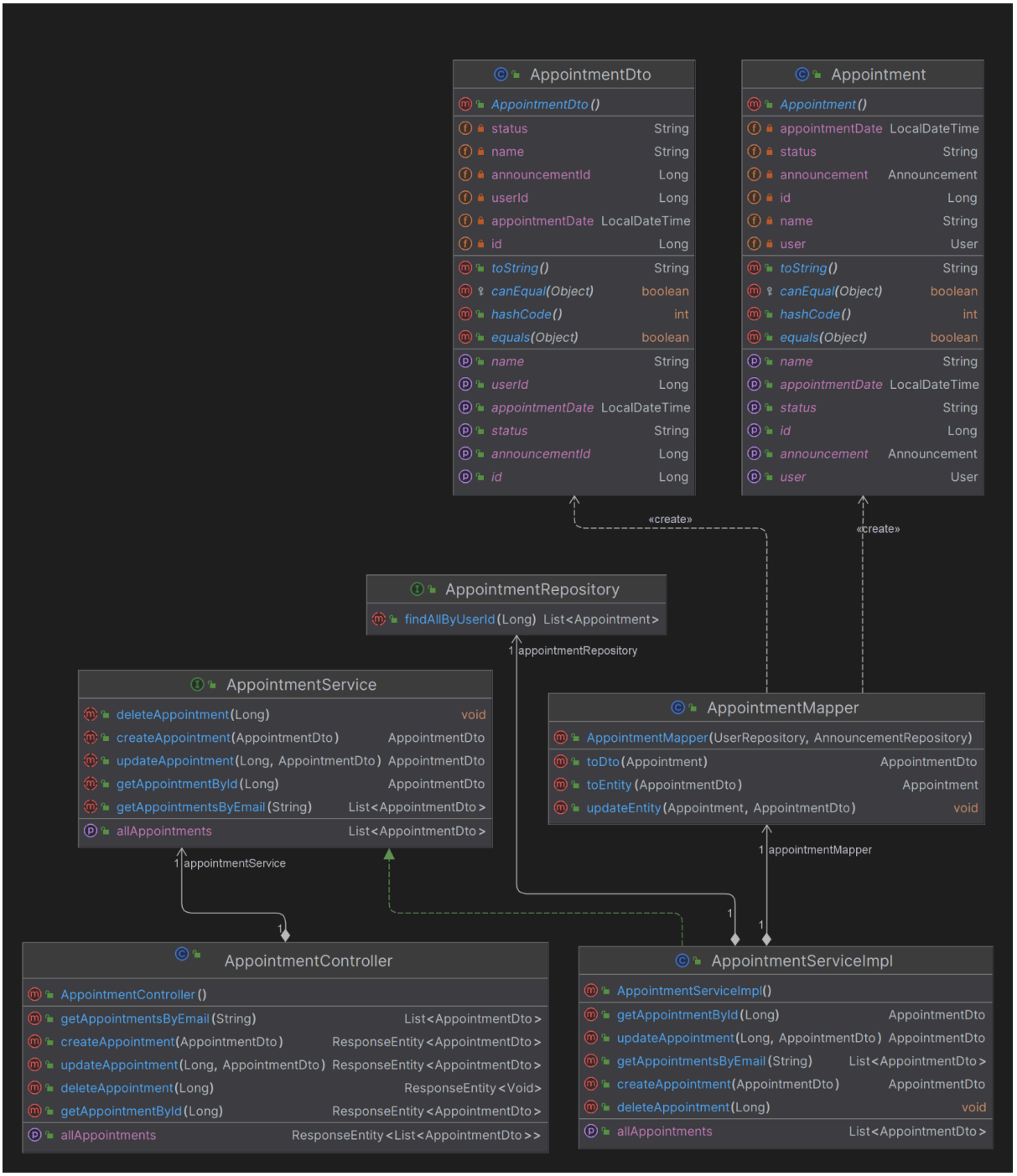
Na potrzeby przeprowadzenia porównań i testów powstała aplikacja komisu samochodowego, której część frontendowa została stworzona w języku JavaScript w dwóch wariantach: z wykorzystaniem Reacta oraz Vue. Backend aplikacji został napisany w Javie z użyciem frameworka Spring. Do przechowywania danych wykorzystano bazę danych MySQL, a do konteneryzacji zastosowano Dockera.

3.1 Diagram przypadków użycia



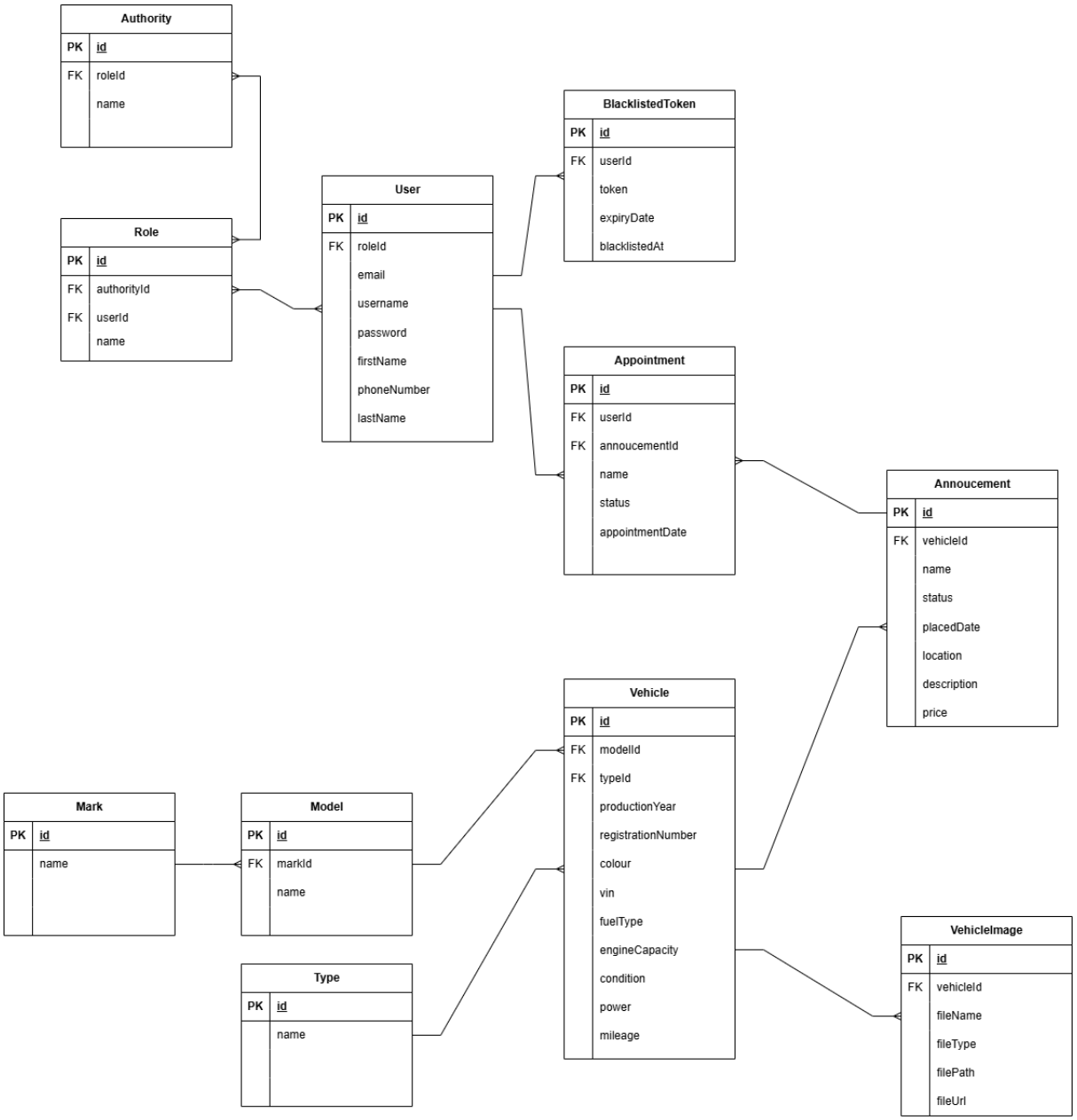
Rys. 6 Diagram przypadków użycia

3.2 Diagram klas



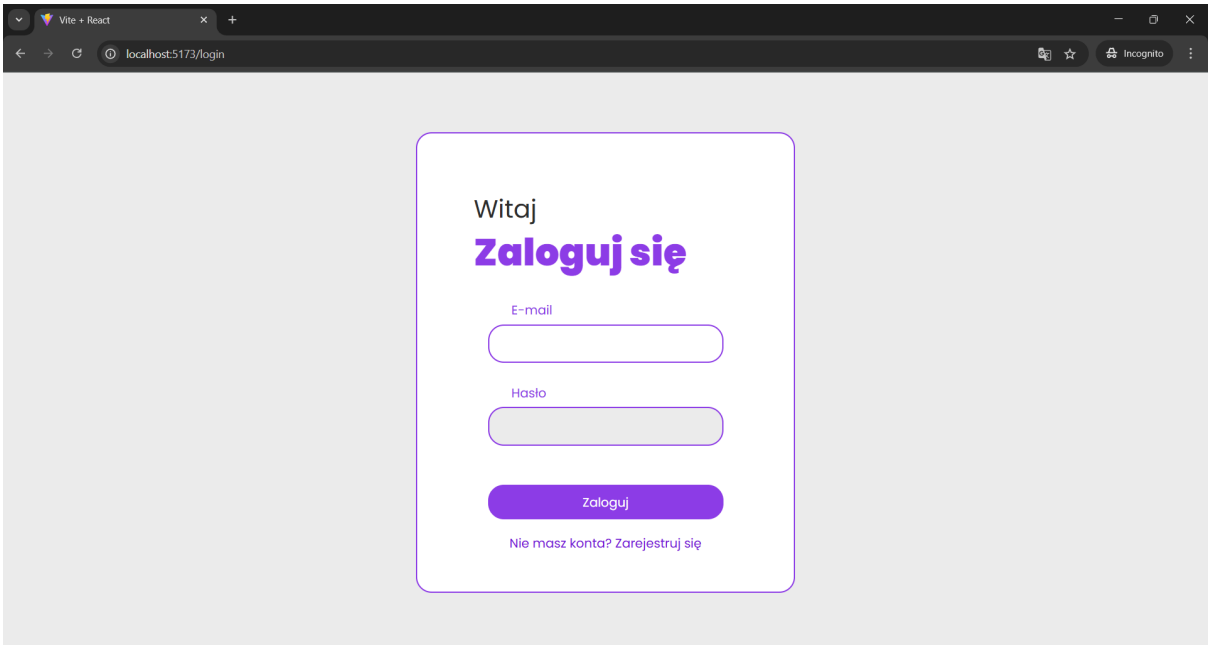
Rys. 7 Diagram klas (fragment)

3.3 Schemat bazy danych

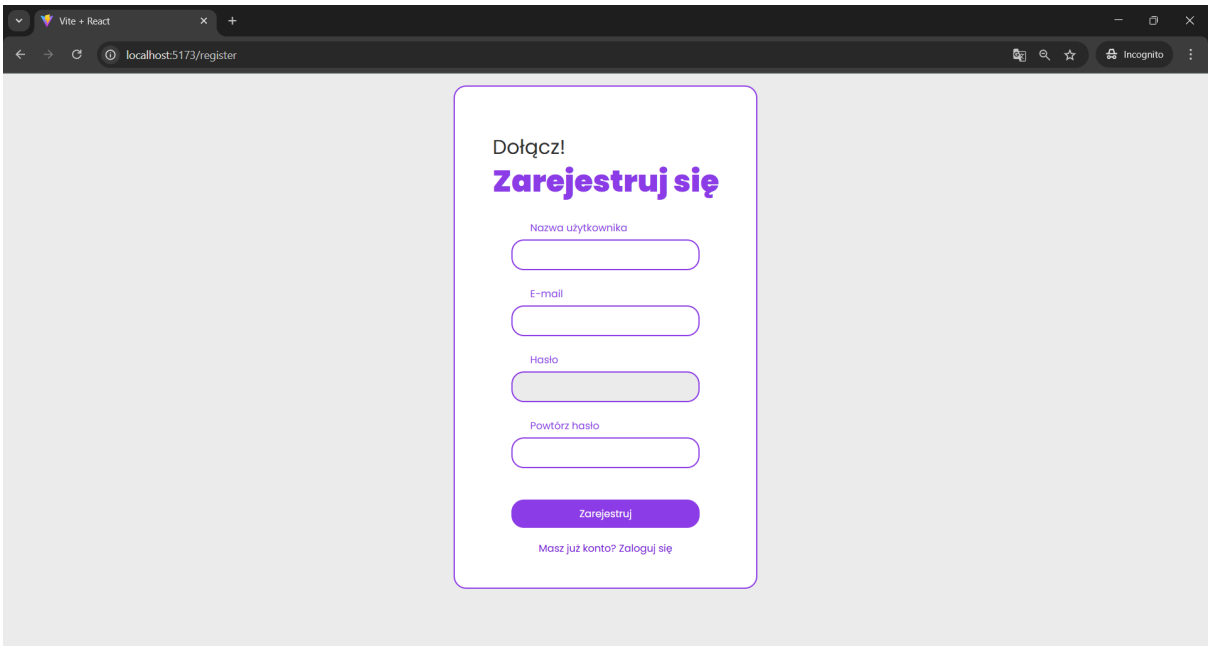


Rys. 8 Schemat bazy danych

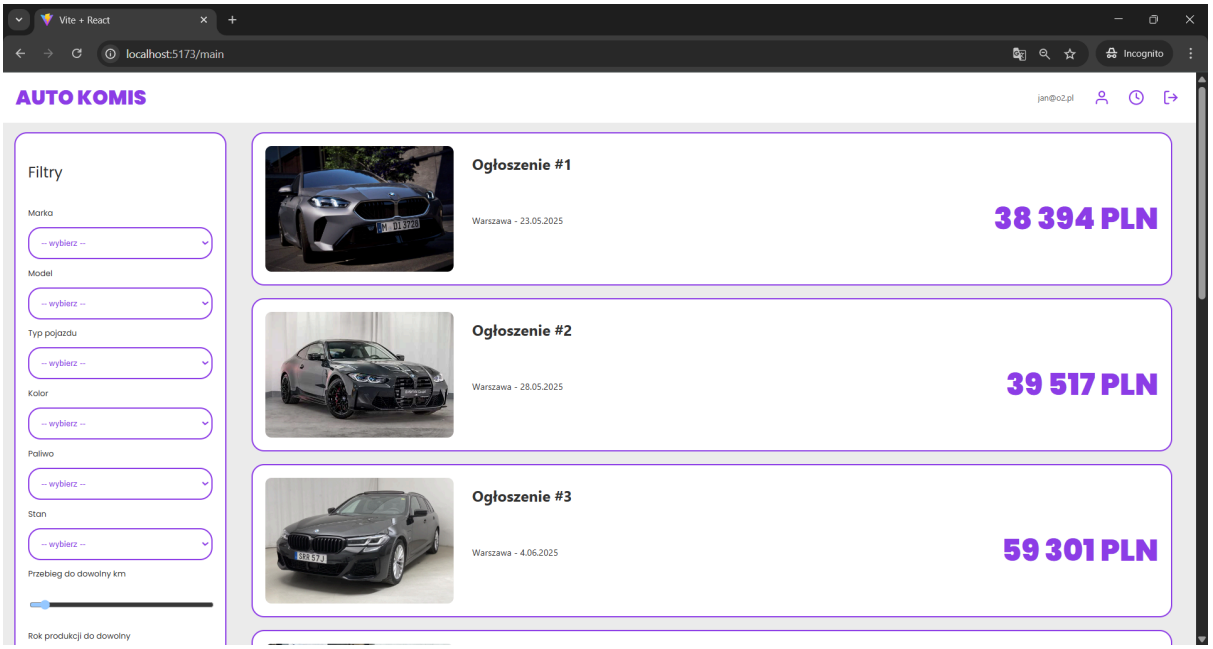
3.4 Interfejs



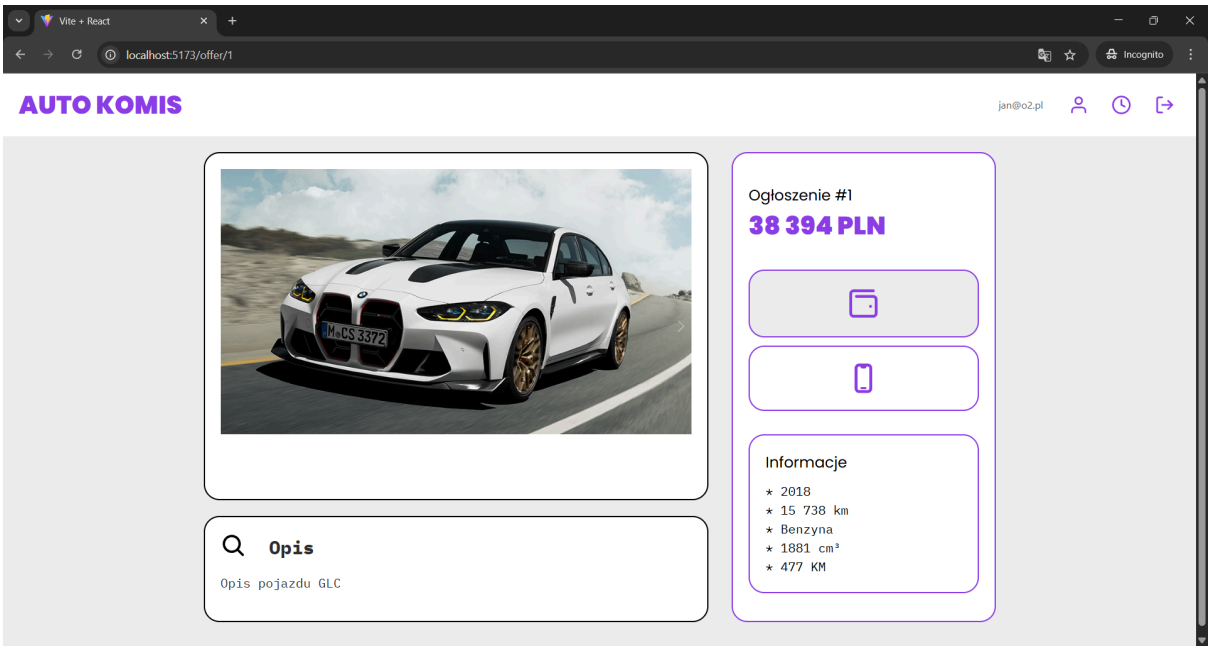
Rys. 9 Ekran logowania



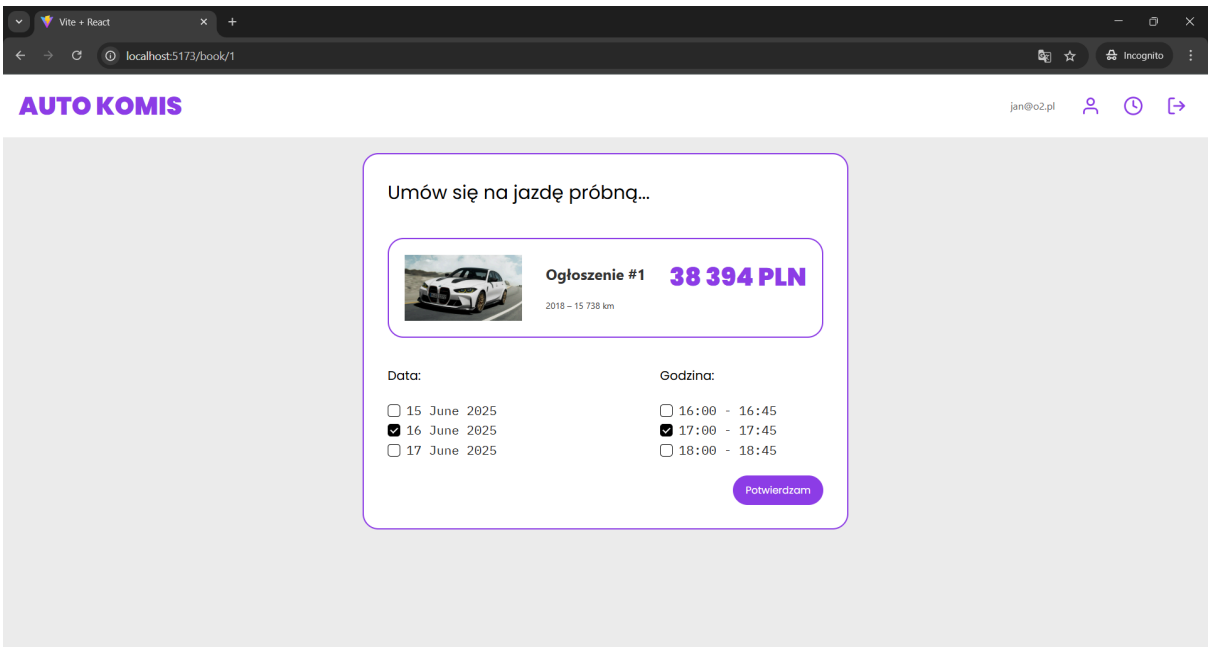
Rys. 10 Ekran rejestracji



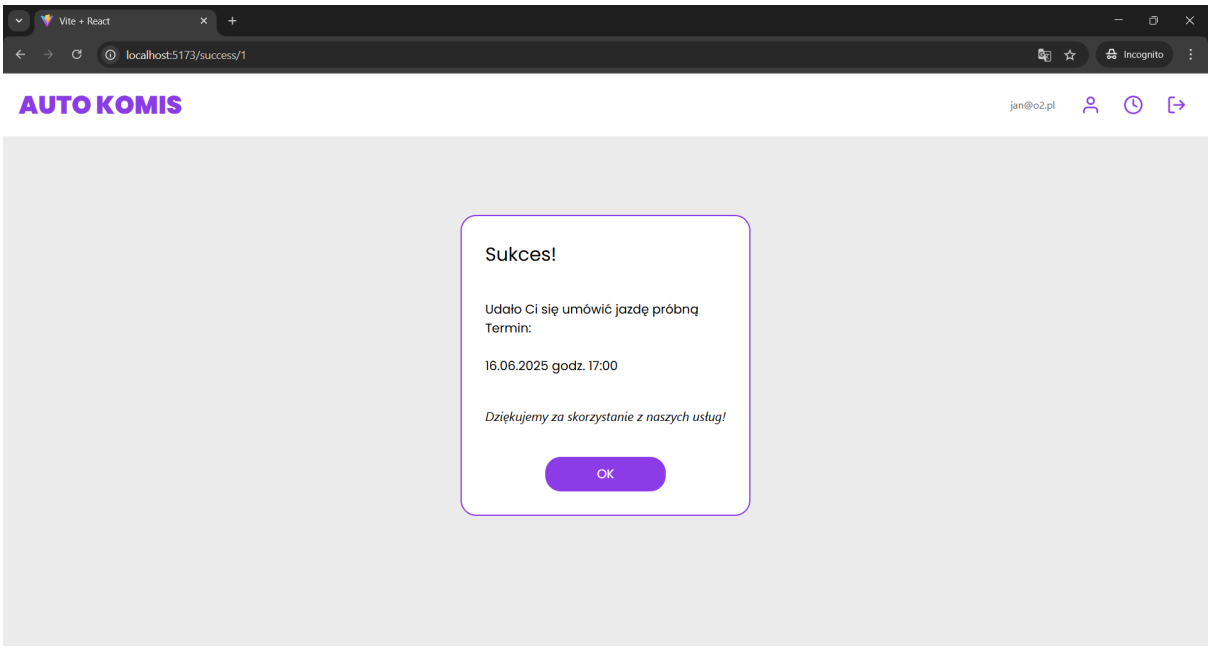
Rys. 11 Ekran główny



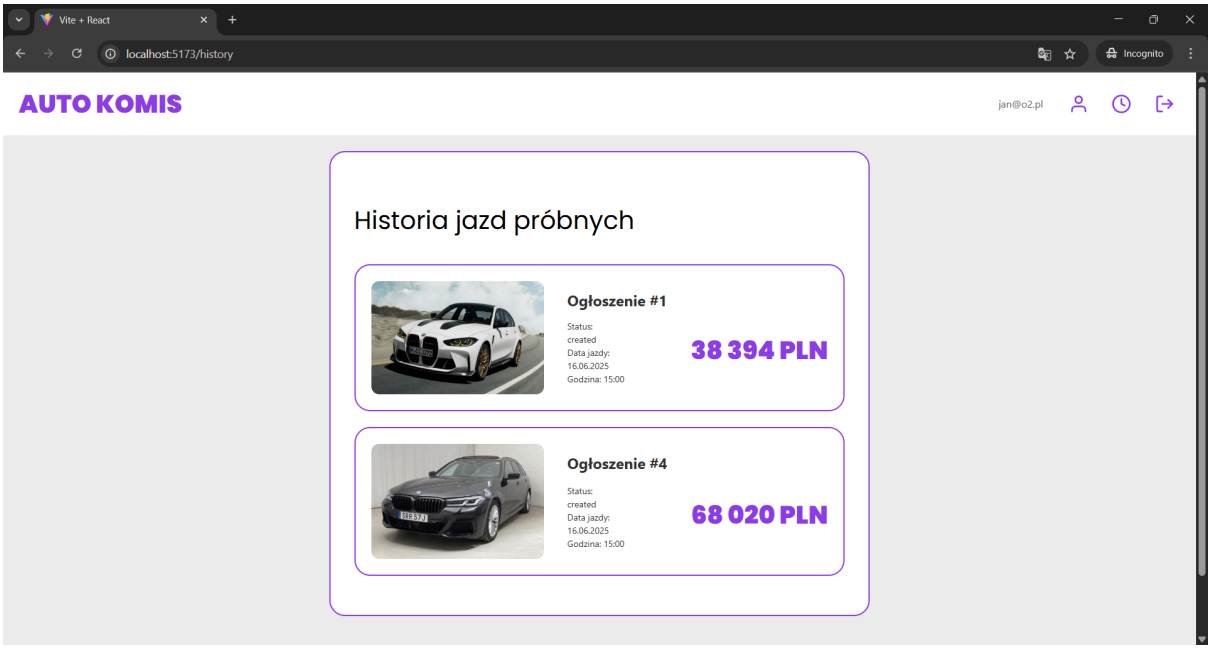
Rys. 12 Ekran oferty



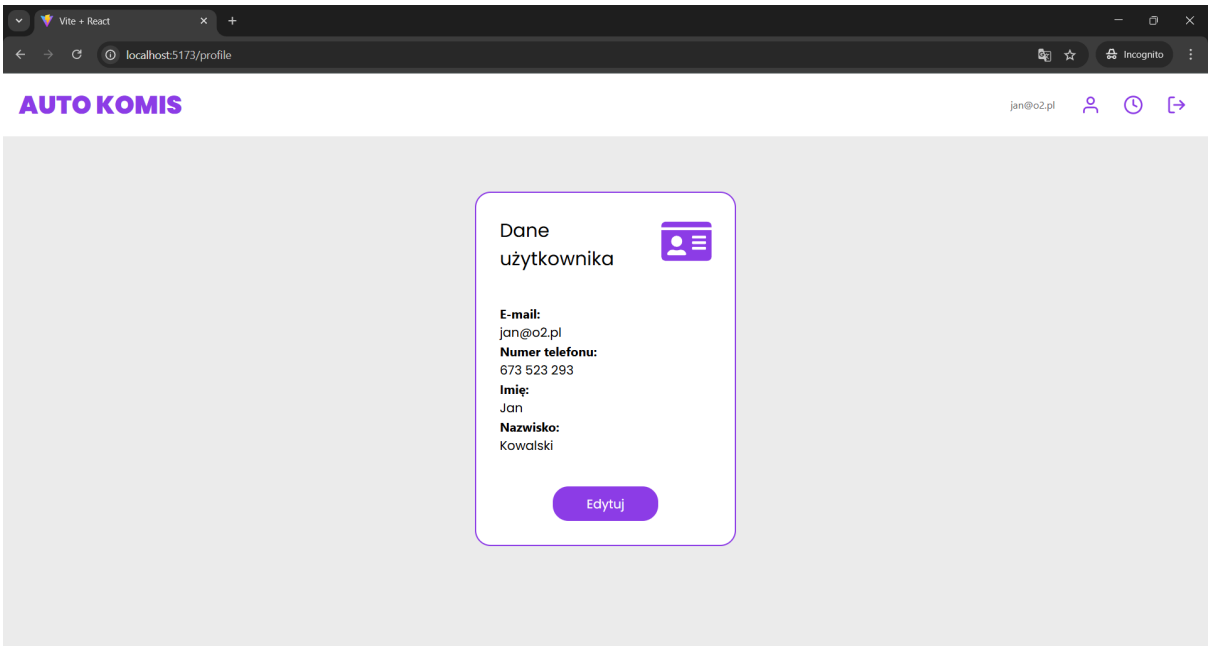
Rys. 13 Ekran umówienia jazdy próbnej



Rys. 14 Ekran sukcesu



Rys. 15 Ekran historii jazd próbnych



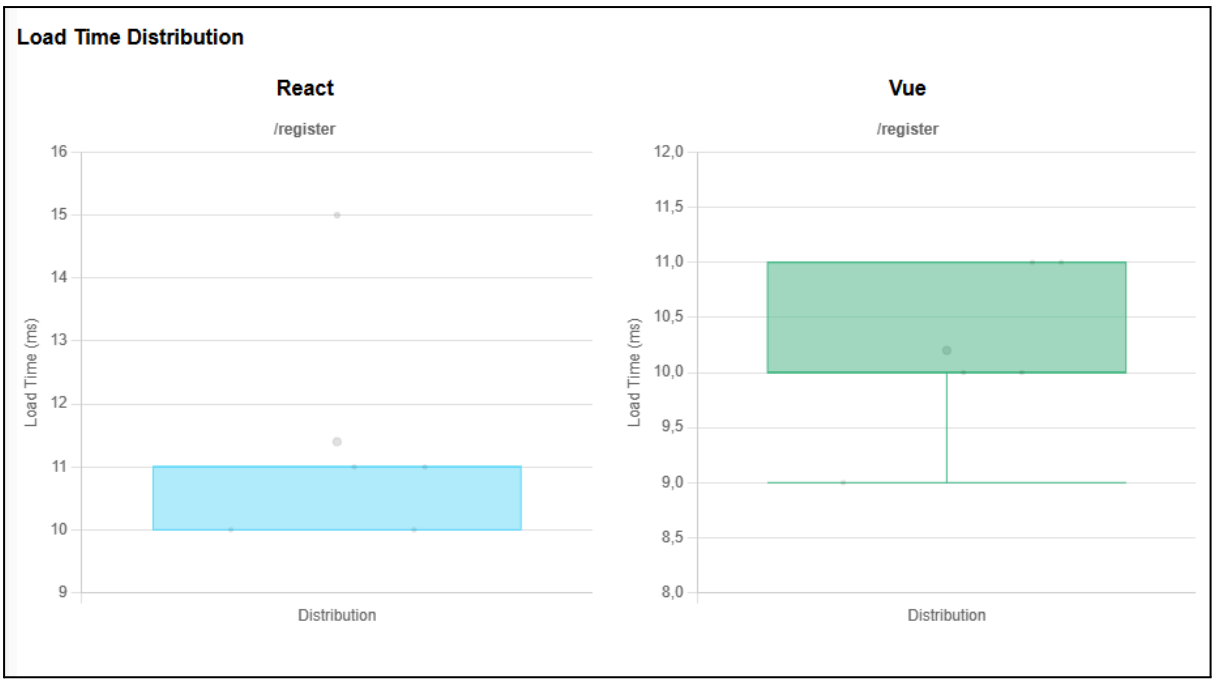
Rys. 16 Ekran profilu

4. Wyniki testów

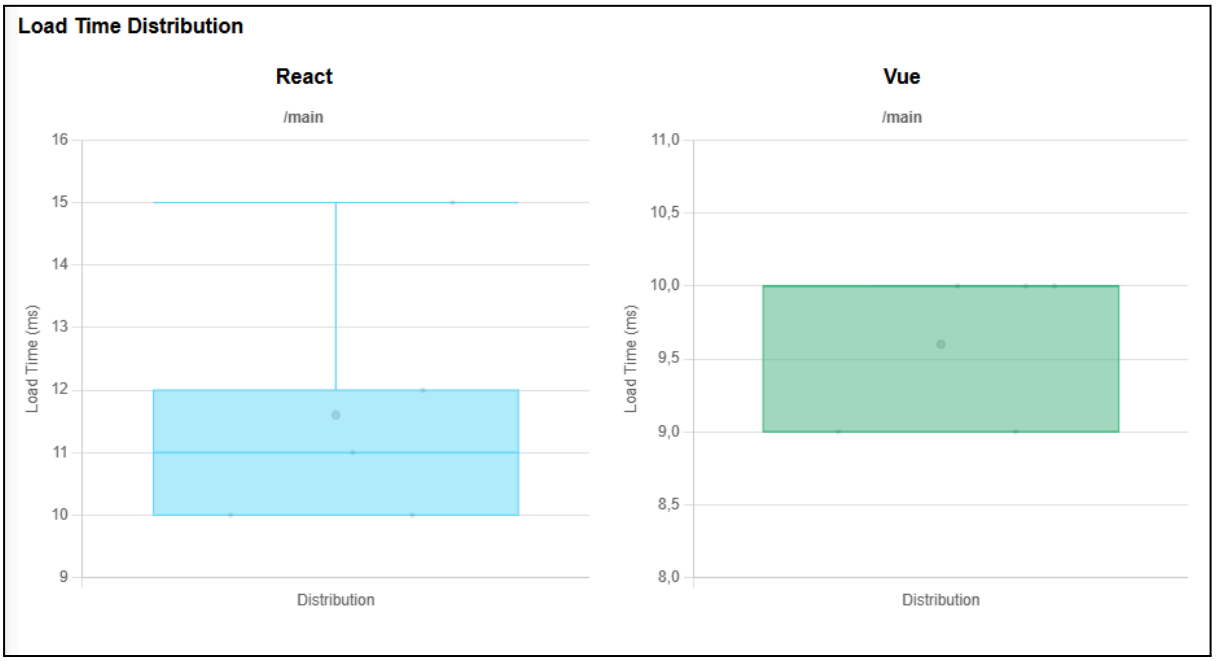
4.1 Czas ładowania strony

Test automatycznie mierzy pełny cykl ładowania strony internetowej, rozpoczynając pomiar od momentu inicjacji nawigacji przez użytkownika (kliknięcie linku lub wpisanie URL) aż do całkowitego zakończenia ładowania wszystkich zasobów strony. Wykorzystuje przeglądarkowe API performance.timing do precyzyjnego obliczenia czasu między navigationStart a loadEventEnd, co daje dokładny pomiar w milisekundach rzeczywistego czasu oczekiwania użytkownika. Test jest wykonywany w kontrolowanych warunkach z wyłączoną pamięcią podręczną przeglądarki, aby zapewnić powtarzalne i wiarygodne wyniki pomiarów.

Rys. 17 Wykres pudełkowy dla ekranu login z zaznaczonym odrzuconym punktem dla czasu ładowania strony



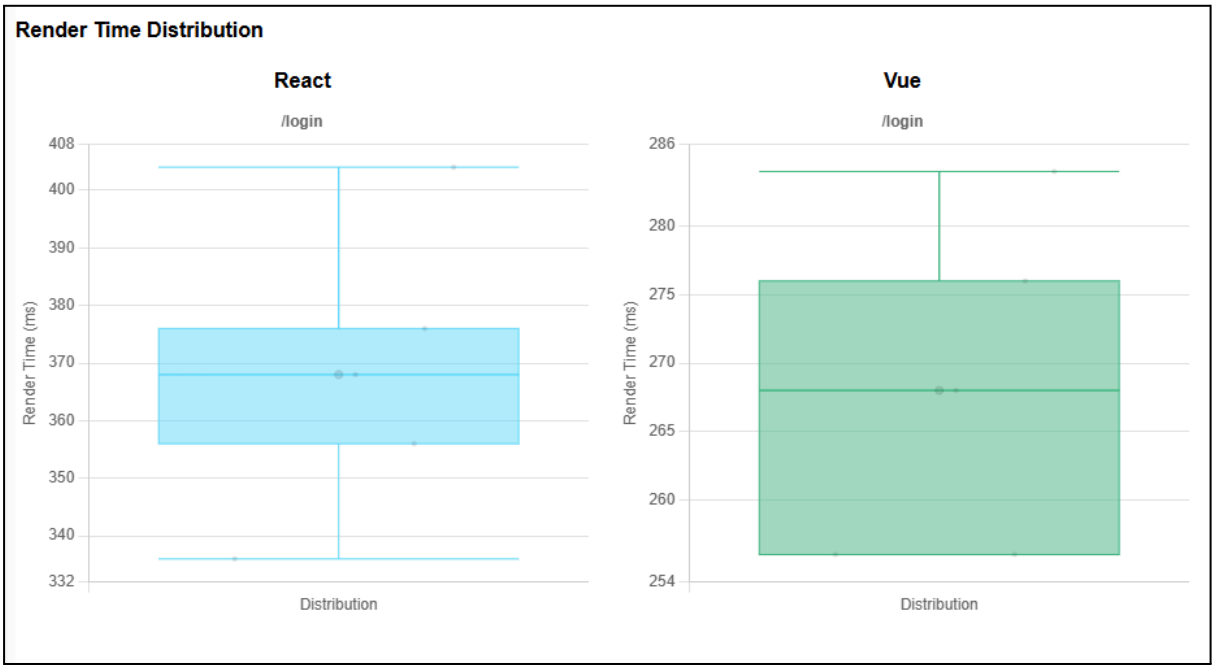
Rys. 18 Wykres pudełkowy dla ekranu register dla czasu ładowania strony



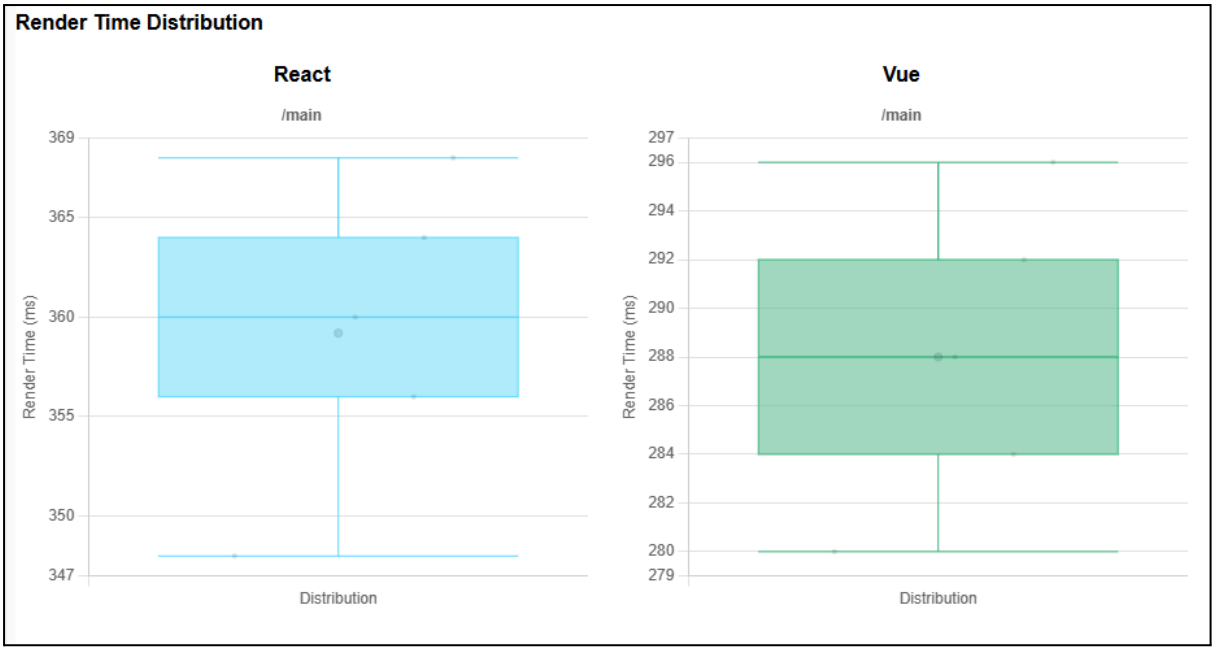
Rys. 19 Wykres pudełkowy dla ekranu main dla czasu ładowania strony

4.2 Czas renderowania

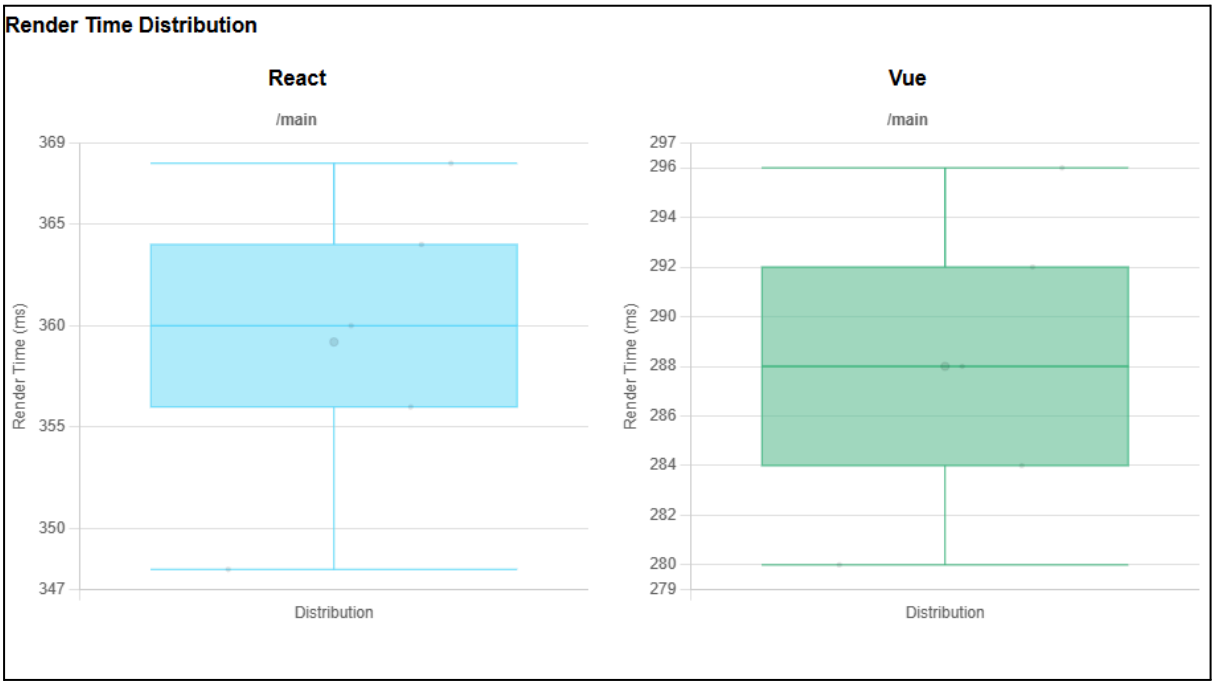
Test mierzy szybkość wizualnego wyświetlania treści na ekranie użytkownika, priorytetowo wykorzystując nowoczesne metryki first-contentful-paint (czas wyświetlenia pierwszej znaczącej treści) lub first-paint (czas pierwszego renderowania), a w przypadku ich niedostępności oblicza czas między navigationStart a domComplete. Dodatkowo implementuje zaawansowane pomiary specyficzne dla frameworków React i Vue, które wykorzystują ich wbudowane narzędzia deweloperskie do dokładnego pomiaru czasu renderowania komponentów. Test uwzględnia specyfikę różnych technologii frontendowych i dostarcza precyzyjnych danych o wydajności renderowania niezależnie od użytego frameworka.



Rys. 20 Wykres pudełkowy dla ekranu login dla czasu renderowania



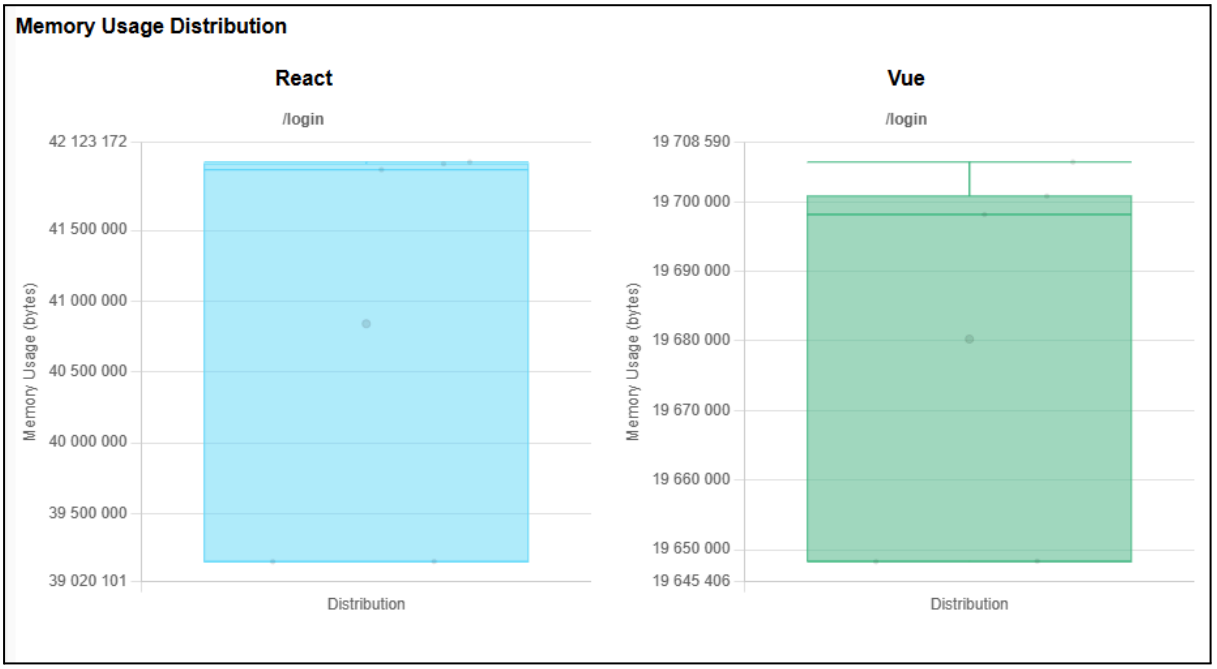
Rys. 21 Wykres pudełkowy dla ekranu register dla czasu renderowania



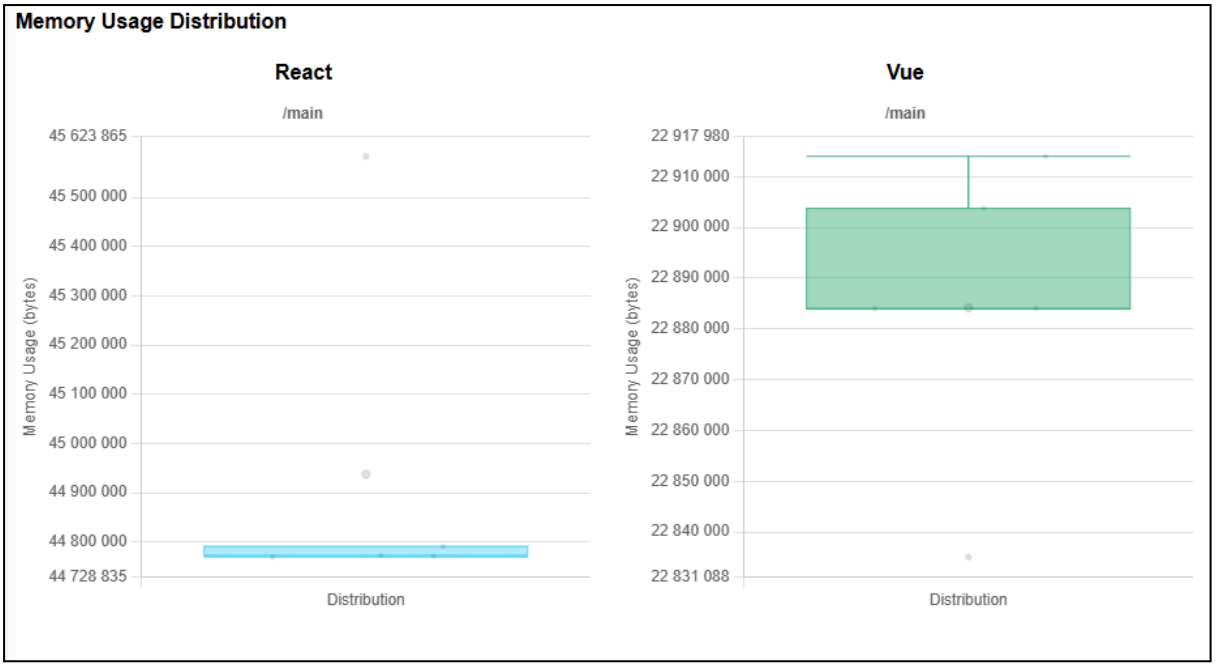
Rys. 22 Wykres pudełkowy dla ekranu main dla czasu renderowania

4.3 Zużycie pamięci

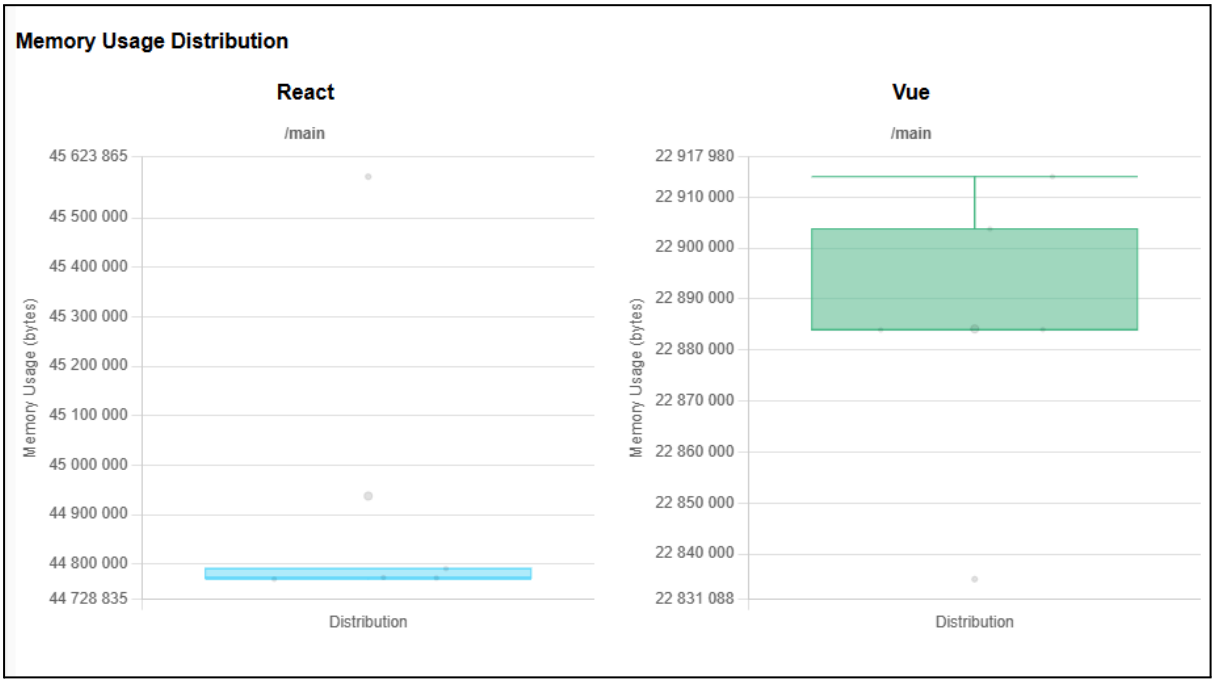
Test monitoruje rzeczywiste zużycie pamięci JavaScript przez aplikację w czasie rzeczywistym, wykorzystując przeglądarkowe API window.performance.memory.usedJSHeapSize, które zwraca aktualny rozmiar aktywnie wykorzystywanej sterty JavaScript w bajtach. Pomiar jest wykonywany po pełnym załadowaniu strony i ustabilizowaniu się aplikacji, co pozwala na uzyskanie reprezentatywnych danych o rzeczywistym zapotrzebowaniu pamięciowym. Test dostarcza kluczowych informacji o efektywności zarządzania pamięcią przez aplikację, co jest szczególnie istotne dla długotrwałych sesji użytkowników i aplikacji o dużej złożoności.



Rys. 23 Wykres pudełkowy dla ekranu login dla zużycia pamięci



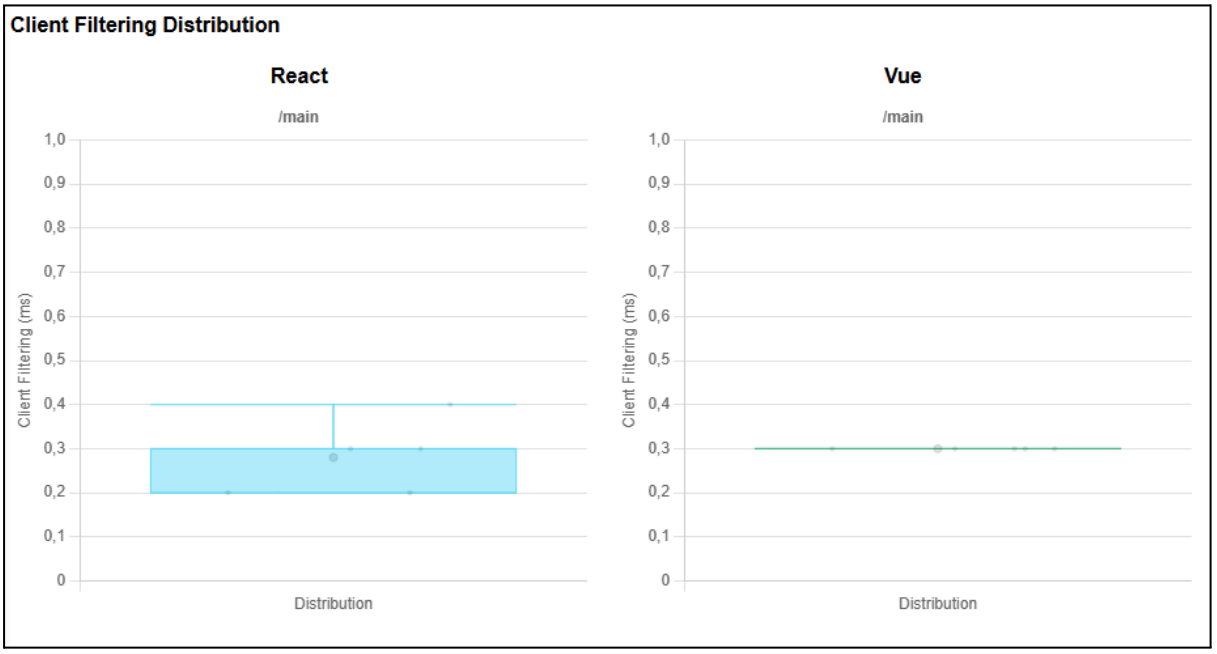
Rys. 24 Wykres pudełkowy dla ekranu register dla zużycia pamięci



Rys. 25 Wykres pudełkowy dla ekranu main dla zużycia pamięci

4.4 Filtrowanie po stronie klienta

Test wydajności operacji filtrowania danych jest wykonywany wyłącznie na stronie głównej aplikacji (/main) i symuluje realistyczne scenariusze przetwarzania dużych zbiorów danych po stronie przeglądarki. Generuje zestaw 10,000 testowych obiektów z różnorodnymi właściwościami (identyfikator, nazwa, kategoria, wartość liczbowa, status aktywności, rok, cena) i wykonuje złożone filtrowanie wielokryteriowe sprawdzające jednocześnie pięć warunków logicznych. Test może wykorzystywać dedykowane funkcje optymalizacyjne frameworka (runReactFilterTest dla React, runVueFilterTest dla Vue) lub standardowe JavaScript Array.filter(), mierząc precyzyjnie czas wykonania operacji za pomocą performance.now() i dostarczając danych o wydajności przetwarzania danych w środowisku przeglądarki.



Rys. 26 Wykres pudełkowy dla ekranu main dla czasu filtrowania po stronie klienta

4.5 Podsumowanie wyników

Route: /login

Metric	React (mean)	React (median)	Vue (mean)	Vue (median)	Difference	Winner
Load Time	10.25 ms	10.50 ms	10.25 ms	10.00 ms	0.00% slower	Tie
Render Time	368.00 ms	368.00 ms	268.00 ms	268.00 ms	37.31% slower	Vue
Memory Usage	38.95 MB	39.99 MB	18.77 MB	18.79 MB	107.52% slower	Vue

Rys. 27 Podsumowanie wyników testów dla ekranu login

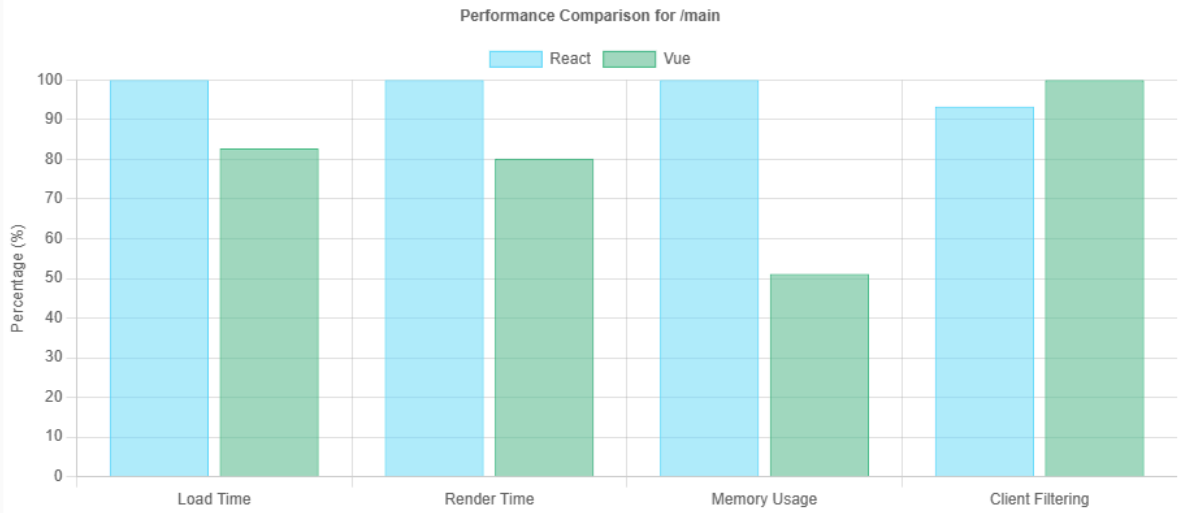
Route: /register

Metric	React (mean)	React (median)	Vue (mean)	Vue (median)	Difference	Winner
Load Time	10.50 ms	10.50 ms	10.20 ms	10.00 ms	2.94% slower	Tie
Render Time	365.60 ms	344.00 ms	256.00 ms	256.00 ms	42.81% slower	Vue
Memory Usage	38.48 MB	37.53 MB	18.80 MB	18.80 MB	104.70% slower	Vue

Rys. 28 Podsumowanie wyników testów dla ekranu register

Route: /main

Metric	React (mean)	React (median)	Vue (mean)	Vue (median)	Difference	Winner
Load Time	11.60 ms	11.00 ms	9.60 ms	10.00 ms	20.83% slower	Vue
Render Time	359.20 ms	360.00 ms	288.00 ms	288.00 ms	24.72% slower	Vue
Memory Usage	42.70 MB	42.70 MB	21.84 MB	21.83 MB	95.56% slower	Vue
Client Filtering	0.28 ms	0.30 ms	0.30 ms	0.30 ms	6.67% faster	React



Rys. 29 Podsumowanie wyników testów dla ekranu main

Summary

Metric	React (avg)	Vue (avg)	Difference	Winner
Load Time	10.78 ms	10.02 ms	7.65% slower	Vue
Render Time	364.27 ms	270.67 ms	34.58% slower	Vue
Memory Usage	40.04 MB	19.80 MB	102.23% slower	Vue
Client Filtering	0.28 ms	0.30 ms	6.67% faster	React

Rys. 30 Wykres pudełkowy przedstawiający podsumowanie wszystkich wyników

Bibliografia

[1] https://fireup.pro/pl/blog/vue-vs-react-ktory-z-nich-wybrac-i-dlaczego [dostęp 09.06.2025]

[2] https://w3techs.com/technologies/comparison/js-angularjs,js-react,js-vuejs [dostęp 09.06.2025]

[3] https://npmtrends.com/react-vs-vue [dostęp 09.06.2025]

[4] https://smartbees.pl/blog/vue-vs-react [dostęp 09.06.2025]

[5] https://survey.stackoverflow.co/2024/technology#most-popular-technologies-webframe [dostęp 09.06.2025]

[6] https://trends.stackoverflow.co/?tags=reactjs%2Cvue.js [dostęp 09.06.2025]

[7] https://justjoin.it/job-offers?experience-level=c-level&orderBy=DESC&sortBy=published [dostęp 09.06.2025]

[8] https://nofluffjobs.com/pl [dostęp 09.06.2025]

[9] https://itentio.com/blog/frontend-developer-salary-in-poland-2024/?utm_source=chatgpt.com [dostęp 09.06.2025]