## Sformułowanie problemu

W obliczu stale rosnącej liczby aplikacji internetowych, istnieje potrzeba stworzenia niewielkiego, szybkiego i łatwego w instalacji frameworka do budowy interfejsów użytkownika opartych na JSX. Istniejące rozwiązania często są rozbudowane, co może wprowadzać dodatkową złożoność dla programistów oraz prowadzić do nadmiernego zużycia zasobów. W związku z tym, konieczne jest zaprojektowanie minimalistycznego narzędzia, które zapewni wydajność, prostotę użycia oraz szybkość działania, jednocześnie nie ograniczając funkcjonalności dostępnych dla deweloperów. Wymagane jest również zapewnienie prostego procesu instalacji, aby umożliwić szybkie rozpoczęcie pracy nad nowymi projektami bez zbędnego obciążenia związanego z konfiguracją i przygotowaniem środowiska.

## Wykorzystane technologie

JavaScript

Babel - kompilacja kodu. Konfiguracja poniżej:

|  |
| --- |
| {      "plugins": [        ["@babel/plugin-transform-react-jsx", {          "pragma": "MiniFramework.createElement",          "throwIfNamespace": false        }]      ]    } |

## Wymagania Funkcjonalne

Projektowany framework ma być lekki, reaktywny i elastyczny, zapewniając programistom wygodne narzędzie do tworzenia interfejsów użytkownika. Poniżej przedstawiono główne wymagania funkcjonalne:

* Lekkość: Framework powinien być niewielki pod względem rozmiaru, aby umożliwić szybkie ładowanie stron internetowych oraz minimalne zużycie zasobów.
* Reaktywność: Framework powinien automatycznie reagować na zmiany stanu komponentów, umożliwiając dynamiczne odświeżanie widoku w czasie rzeczywistym.
* Obsługa JSX: Framework powinien umożliwiać pisanie komponentów za pomocą składni JSX, co pozwoli na czytelniejszy i bardziej ekspresyjny kod.
* Zmienne w kodzie: Programiści powinni mieć możliwość korzystania ze zmiennych w kodzie komponentów, co umożliwi dynamiczne dostosowywanie się interfejsu do zmieniających się warunków.
* Generowanie list w czasie rzeczywistym: Framework powinien zapewniać łatwy sposób generowania list dynamicznie na podstawie danych, co umożliwi szybką aktualizację widoku w przypadku zmian w danych.

Powyższe wymagania mają na celu zapewnienie prostego i efektywnego narzędzia dla programistów, umożliwiając szybkie tworzenie interaktywnych interfejsów użytkownika bez zbędnego obciążenia związanego z konfiguracją czy złożoną implementacją.

## Struktura projektu

Projekt składa się z dwóch głównych modułów: MiniFramework oraz MiniComponent.

**MiniFramework** jest centralnym elementem projektu, dostarczającym podstawowe funkcje do obsługi JSX oraz zarządzania stanem komponentów.

* **createElement:** Funkcja createElement pozwala na tworzenie elementów JSX na podstawie podanego tagu, właściwości i dzieci. Jest to podstawowa funkcja do budowania struktury interfejsu użytkownika.render: Renderuje elementy JSX do rzeczywistego drzewa DOM.
* **useState:** Funkcja useState umożliwia zarządzanie stanem komponentów poprzez hooki. Jest to kluczowa funkcja w reaktywnym podejściu do tworzenia interfejsów użytkownika.
* **render:** Funkcja render odpowiedzialna jest za renderowanie elementów JSX do rzeczywistego drzewa DOM. Wspiera ona reaktywne odświeżanie widoku w przypadku zmiany stanu komponentów.
* update: Aktualizuje komponenty po zmianie stanu. W dużej części to ona odpowiada za reaktywne zmiany po stronie użytkownika
* **getApi:** Pobiera dane z API za pomocą zadanego URL-a i metody.
* **replaceValues:** Zastępuje wartości w kodzie JSX z tablicy danych. Pomaga w dynamicznym generowaniu list gdy zwykła funkcja map nie radzi sobie z reaktywnym odświeżaniem.

**MiniComponent** definiuje bazową klasę komponentu, która jest rozszerzana przez konkretne komponenty w aplikacji. Poniżej przedstawiono główne elementy MiniComponent oraz opisano ich funkcje:

* constructor: Konstruktor MiniComponent inicjalizuje właściwości (props) i stan (state) komponentu oraz wywołuje metody cyklu życia komponentu.
* willInit, didInit, didUpdate, willTerminate: Metody cyklu życia komponentu są wywoływane w określonych momentach, co pozwala na dostosowanie zachowania komponentu do potrzeb aplikacji.
* mainDiv: Metoda mainDiv jest przykładową funkcją, która zwraca nazwę klasy komponentu. Jest to przykład funkcji, która może być użyta w komponentach potomnych.
* mount: Metoda mount jest odpowiedzialna za renderowanie komponentu do drzewa DOM. Każdy komponent potomny musi zaimplementować tę metodę, aby był w stanie być wyrenderowany.

## Wdrożenie

Framework może być używany w aplikacjach internetowych wymagających dynamicznego renderowania interfejsu użytkownika. Wymaga skompilowania przy użyciu Babel do kodu JavaScript zrozumiałego przez przeglądarkę.

## Porównanie wielkości

\*Wykres nie uwzględnia kompilatora Babel

## Wnioski

Stworzenie własnego frameworka JSX pozwala na większą elastyczność w budowie interfejsów użytkownika oraz lepszą kontrolę nad kodem. Dodanie struktury komponentów umożliwia modularyzację i reużywalność kodu. Jednakże wymaga to większego nakładu pracy w porównaniu z korzystaniem z istniejących rozwiązań. Dzięki MiniFramework oraz MiniComponent możliwe jest skuteczne zarządzanie stanem komponentów oraz integracja z zewnętrznymi źródłami danych.