Zadanie kwalifikacyjne

Twoje zadanie polega na stworzniu aplikacji, która pozwala na rejestrację i logowanie użytkownika a zalogowanemu użytkownikowi wyświetla historię transakcji. Aplikacja powinna składać się z 4 ekranów:

1. Widok niezalogowanego użytkownika: na górze po lewej stronie ekranu umieść przyciski „Logowanie” i „Rejestracja”. Na środku strony możesz umieścić jakąś treść jako zachętę lub ozdobę
2. Rejestracja: formularz rejestracji nowego użytkownika
3. Logowanie: formularz logowania użytkownika
4. Widok zalogowanego użytkownika: na górze strony po prawej stronie ekranu umieść nazwę użytkownika i przycisk „Wyloguj”. Na środku ekranu umieść moduł historii transakcji.

Do stworzenia aplikacji użyj czystego Javascript, HTML i CSS. Zewnętrzne biblioteki i frameworki nie są dozwolone – nie używaj więc jQuery, Reacta, Boostrapa, SCSS, lodasha, TypeScripta itp. – chyba, że wyraźnie wskazano inaczej w danej części zadania. Ograniczenie technologii jest częścią zadania – na dalszym etapie, czyli w projekcie głównym, będziemy korzystać z wszelkich dobroci.

Wymagania:

- **Ogólne**
  - Językiem aplikacji jest język polski, językiem kodu (i komentarzy) jest język angielski.
  - Formularz rejestracji składający się z pól „Nazwa użytkownika”, „Hasło, „Email”, „Potwierdź email” i przycisku „Zarejestruj”
  - Formularz logowania składający się z pół „Nazwa użytkownika” i „Hasło” oraz przycisku „Zaloguj”
  - Górny pasek akcji („Logowanie”, „Rejestracja”, „Wyloguj”) powinien odróżniać się od reszty strony np. kolorem tła
  - Formularze powinny być wyśrodkowane w pionie i poziomie a pola formularza być w układzie kolumny z etykietami po lewej stronie
  - W górnym pasku akcji oraz w formularzach do pozycjonowania elementów użyj flexboxa (https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox) Do pozycjonowania formularzy możesz również użyć grida. (https://developer.mozilla.org/pl/docs/Web/CSS/CSS_Grid_Layout)
  - Zadbaj o estetyczny i spójny wygląd całości jak i poszczególnych elementów. Nie masz tutaj sztywnych ram ani konkretnych wymagań, ale też postaraj się nie przesadzić, jednocześnie demonstrując swoją wiedzę o CSS

  - Błędy walidacji pokazuj w kontekście pola, które zawiera błąd (pod lub obok pola, którego błąd dotyczy)
  - Cała aplikacja używa czcionki Open Sans, osadzonej przez <link />
  - Nie powinno być żadnych dodatkowych wymagań dotyczących „uruchomienia" aplikacji, poza otwarciem pliku index.html w przeglądarce

- **Rejestracja**
  - Po rejestracji, nowy użytkownik jest od razu zalogowany a zarejestrowani użytkownicy są przechowywani w przeglądarce tak, żeby można było zalogować się na wszystkie zarejestrowane konta, również po odświeżeniu strony - https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API
  - Nazwa użytkownika musi być unikalna a na jeden adres email nie można założyć więcej niż jednego konta
  - Walidacja:
    - pole „Email" musi mieć poprawny format adresu email,
    - pole „Potwierdź email" musi mieć taką samą wartość jak „Email",
    - pole „Nazwa użytkownika" musi mieć między 6 a 16 znaków, dozwolone znaki to litery i cyfry,
    - pole „Hasło" musi mieć co najmniej 6 znaków
    - walidacja pól następuje po wciśnięciu przycisku „Zarejestruj"
    - do walidacji możesz użyć własności HTML - https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation
  - W tym widoku widoczny u góry jest tylko przycisk „Logowanie"

- **Logowanie**
  - Sprawdź, czy użytkownik istnieje i czy hasło jest poprawne - jeśli nie, wyświetl stosowny błąd
  - Jeśli podany adres e-mail jest wolny, zachęć użytkownika do założenia konta
  - Jeśli użytkownik jest zalogowany, to po odświeżeniu strony nadal pozostaje zalogowany
  - W tym widoku widoczny u góry jest tylko przycisk „Rejestracja"

- **Widok zalogowanego użytkownika**
  - W tym widoku przyciski „Logowanie" i „Rejestracja" są niewidoczne
  - Ekran podzielony jest w poziomie na dwie części: górna – wykresy i dolna – historia transakcji
  - Ekran posiada dwa widoki – mobilny (do 768px szerokości włącznie) i desktop
  - Jeśli lista transakcji nie mieści się na ekranie, to powinna się przewijać w pionie, ale część z wykresami pozostaje widoczna
  - Widok nie może być szerszy niż ekran – niedopuszczalne jest przewijanie w poziomie
  - Dane potrzebne do wyświetlenia widoku aplikacja pobiera przy użyciu fetch API
    - GET z URL https://api.jsonbin.io/v3/b/63a092ab15ab31599e2045be
    - nagłówek autoryzacyjny **x-access-key** o wartości: $2b$10$5pBRUbFRKdKft/b8qSQ3leyPQgQ8CLXlvgoQA6GdpYvdWva.pOfGS
    - odpowiedź zawiera „transactions" oraz „transactionTypes"

- Historia transakcji zawiera tabelę transakcji (jeden wiersz – jedna transakcja), składającą się z (od lewej do prawej):
  - Widok mobilny – typ transakcji (ikona), opis, kwota
  - Widok mobilny – pełne szczegóły transakcji widoczne po kliknięciu wiersza, tylko jeden wiersz na raz może być „otwarty”
  - Widok desktop – data, typ transakcji (ikona), opis, kwota, saldo; dodatkowo pod opisem znajduje się nazwa typu transakcji
  - Ikony nie są dostarczone wraz z zadaniem– stwórz lub pobierz darmowe ikony, które nawiązują do typów transakcji
- Wykresy, w tej części użyj biblioteki Chart.js https://www.chartjs.org/, umieszczając ją na stronie przez CDN
  - Wykres kołowy (doughnut lub pie chart), prezentujący procentowy podział transakcji wg ich typu w ogóle transakcji, tj. jeśli dane zawierają 20 transakcji a 10 z nich to typ „Wydatki”, wykres pokazuje „Wydatki” jako 50% koła
  - Wykres słupkowy (bar chart), pokazujący saldo konta na koniec każdego dnia, zawartego w historii transakcji. Oś Y – saldo, oś X – dzień
  - Upewnij się, że dane widoczne po najechaniu kursorem na części wykresów są właściwe
  - W widoku mobilnym, jeden wykres zajmuje całą szerokość ekranu a użytkownik może przewijać między nimi lewo-prawo
  - W widoku desktop, wykresy znajdują się obok siebie i każdy zajmuje ok. połowę ekranu

Wymagania dodatkowe:

Czyli bonusowe punkty za bonusowe funkcjonalności. Możesz wypełnić wszystkie, możesz tylko niektóre, możesz nie robić żadnego.

- Strona musi być responsywna[1]
- Strona musi być dostępna przynajmniej w podstawowym zakresie[2], dotyczy formularzy (nawigacja klawiaturą, semantyka)
- Hasło użytkownika musi być przechowywane niejawnie – czyli nie plain textem, ale np. jako hash – ale sama metoda hashowania nie musi być bardzo ukryta
- Walidacja nazwy użytkownika: długość od 6 do 16 znaków, wartość może składać się tylko z liter, cyfr oraz znaków - _ [ ] \ / przy czym musi zawierać co najmniej 5 liter i jedną cyfrę
- Walidacja email musi uwzględniać popularne aliasy adresów, np. adres@gmail.com to to samo co adres+alias@gmail.com
- Walidacja realizowana z użyciem wyrażeń regularnych (tam, gdzie to ma sens) po stronie JS

[1] https://web.dev/responsive-web-design-basics/
[2] https://webaim.org/techniques/forms/

- Historia transakcji – w widoku mobilnym, transakcje są pogrupowane wg daty, tzn. wyświetla się data (dzień), pod nią transakcje z tego dnia, następnie kolejna data z historii a pod nią transakcje z tego dnia itd.
- Historia transakcji – dodaj możliwość filtrowania listy po typie transakcji oraz wyszukiwanie po opisie
- Wykres słupkowy salda – wartości poniżej zera mają kolor czerwony, wartości powyżej – zielony a linia „0 PLN" jest wyraźnie zaznaczona
- Stwórz kilka własnych zestawów danych (na podstawie JSONa w zadaniu) i przypisuj zestaw na stałe do każdego użytkownika losowo, podczas rejestracji konta
- Aplikacja stworzona jest jako Single Page Application, tzn jedna strona z wieloma widokami zarządzanymi przez JS
- Rozszerz zestaw danych o dodatkowe pola i zaprezentuj je w interfejsie (liczy się sens oraz prezentacja danej)
- Aplikacja posiada dwa języki interfejsu – PL i EN – między którymi użytkownik może się swobodnie przełączać za pomocą przycisku w górnym pasku akcji. Pamiętaj o dodaniu dwóch języków do danych (typy transakcji)

Zasoby:

Praktyczne wprowadzenie do CSS flexbox: https://flexboxfroggy.com/#pl

Praktyczne wprowadzenie do CSS grid: https://cssgridgarden.com/#pl

Przechowywanie danych – Storage API
https://developer.mozilla.org/pl/docs/Web/API/Web_Storage_API

Wyrażenia regularne: https://regexr.com/

Zasady:

Każde wymaganie (każdy punkt lub podpunkt) warte jest 1 pkt. Wymagania dodatkowe warte są 3 pkt każde, postaraj się jednak najpierw wypełnić wymagania podstawowe.

Nie musisz spełnić wszystkich wymagań, aby rozwiązanie zostało zaakceptowane. Ostatecznie liczy się ilość zdobytych punktów, gdyż będziemy wybierać najwyżej punktowane rozwiązania.

Na co będziemy dodatkowo zwracać uwagę:

- Czytelność i porządek w kodzie
- Zastosowanie wzorców i dobrych praktyk
- Użycie najnowszych technologii: semantyczny HTML, ES6+, CSS3+
- Nie korzystaj z pomocy typu GitHub Copilot czy ChatGPT. W kolejnym etapie sprawdzimy, czy znasz i rozumiesz swój kod.

2023 intive PATRONAGE

Na rozwiązania czekamy do **15 stycznia 2023** do północy. Jeżeli masz jakiekolwiek pytania lub wątpliwości co do wymagań, zapraszamy do kontaktu mailowego - patronage2023-javascript@intive.com

Swoje rozwiązanie dostarcz w jednej z dwóch postaci – link do Dysku Google, gdzie znajduje się plik .zip z rozwiązaniem, lub link do publicznego repozytorium na Github.

Powodzenia! 😊