PIK

DOKUMENTACJA KOŃCOWA

Spis treści

1. Założenia projektu

Założeniem projektu było stworzenie platformy e-commerce, która byłaby sklepem internetowym ze sprzętem technologicznym. Aplikacja miała korzystać z nowoczesnych technologii do pisania kodu oraz jego zarządzania, a także monitorowania pracy członków zespołu. Dodatkowo miały zostać napisane testy jednostkowe zarówno do backendu i frontendu dające pokrycie 70%. Projekt miał udostępniać bezpieczne logowanie przy użyciu loginu i hasła, oraz możliwość logowania za pomocą kont z innych serwisów społecznościowych.

2. Opis produktu

Stworzony sklep internetowy nosi nazwę Predator, która dobrze pasuje do ciemnego designu panującego na stronie oraz do sprzętu jaki sklep oferuje, czyli wysokiej klasy notebooki. Platforma zapewnia intuicyjny interfejs, który wygląda dobrze zarówno na urządzeniach mobilnych jak i biurowych. Komunikacja sieciowa jest zabezpieczona, dzięki czemu użytkownicy Predatora nie muszą się obawiać o bezpieczeństwo ich wrażliwych danych.

Użytkownik strony nie musi być zalogowany, aby mieć możliwość przeglądania katalogu sklepu, natomiast logowanie jest konieczne w przypadku dokonywania zakupu. Konto może być założone bezpośrednio na stronie Predator, ale jest również możliwość zalogowania się za pomocą konta Google i Facebook.

Przydatne linki

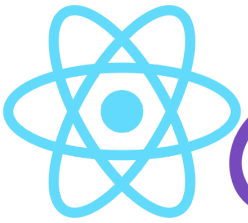
Działająca strona: https://pik-predator.herokuapp.com
Repozytorium kodu: https://github.com/ACharchula/PIK-predator
Nasz blog: https://predator-pik.blogspot.com/

3. Stos technologiczny

3.1. Backend

- Kotlin
- Spring
- JUnit

3.2. Frontend



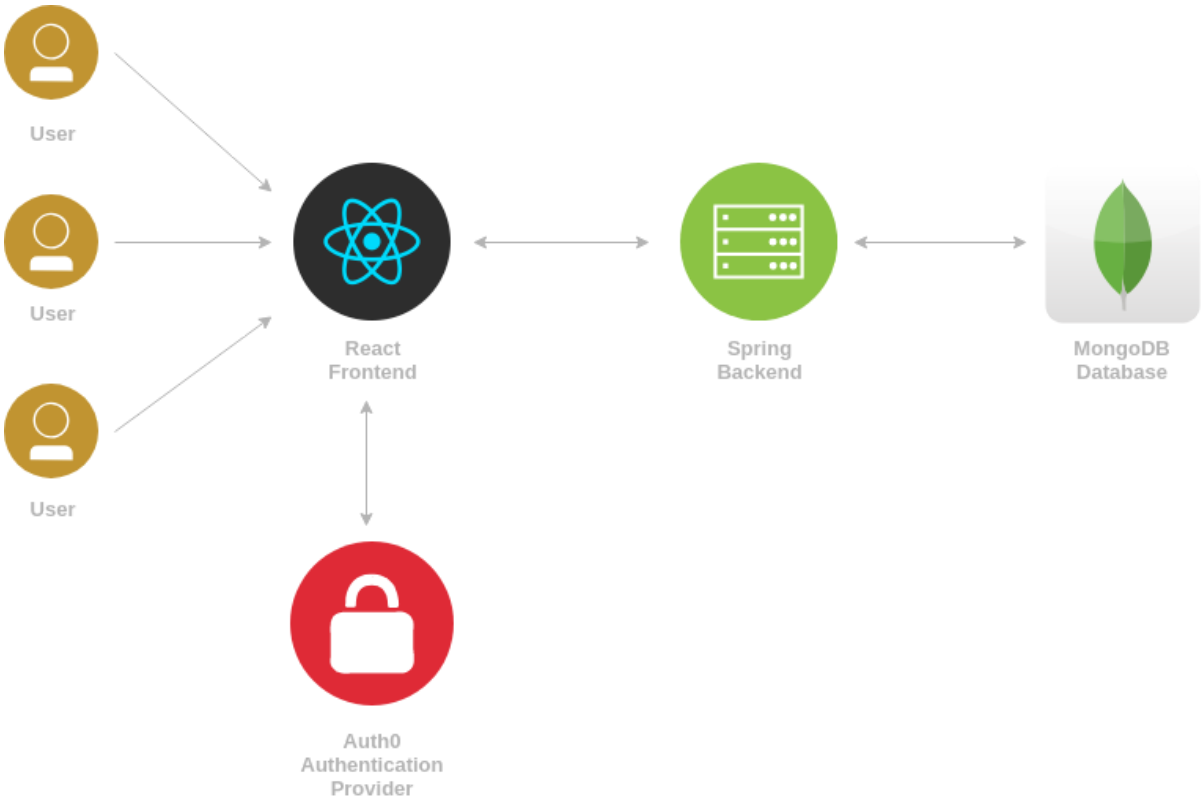
- React
- Redux
- Enzyme
- Sass
- śladowe ilości React Bootstrap

3.3. Reszta narzędzi

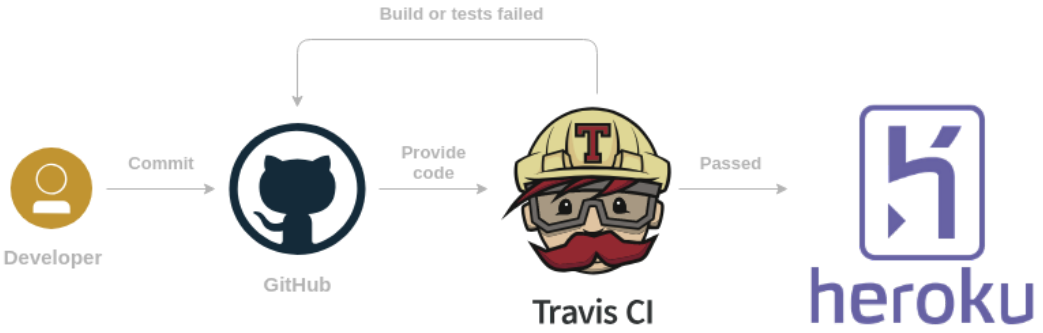
Hosting	Heroku
Baza Danych	MongoDB
Repozytorium Kodu	Github
Organizacja Pracy Zespołu	Github Issues
Szyfrowane Logowanie	Auth0
Płatności	Payment Request API

4. Architektura systemu

Zastosowana została typowa architektura 3-warstwowa z podziałem aplikacji na różne moduły odpowiedzialne za interfejs użytkownika, przetwarzanie danych i składowanie danych. Wybór bazy danych padł na nierelacyjną, dokumentową bazę danych MongoDB. Spowodowane jest to łatwością przetrzymywania danych, których schemat nie jest oczywisty i zawsze taki sam oraz wygodnym dodawaniem produktów, których podzespoły nie istniały jeszcze w naszej bazie, czyli łatwą skalowalnością. Komunikacja między warstwą interfejsu oraz przetwarzania danych odbywa się z wykorzystaniem REST.
Szyfrowanie logowania odbywa się za pomocą serwisu Auth0, który implementuje protokół oauth 2.0, pozwalający użytkownikom udostępniać aplikacjom i stronom trzecim informacje przechowywane u innych dostawców usług.



Deployment kodu na serwer Heroku jest zautomatyzowany dzięki narzędziu Travis. Uruchamia on testy, po których pozytywnym zakończeniu wysyła aplikację na heroku. Robi to sposób automatyczny przy pojawieniu się pull requesta na Githubie.



5. Endpointy REST API

5.1. Katalog

`GET /catalog` - endpoint służący do uzyskiwania produktów z katalogu. Zaimplementowane zostało wsparcie dla wyszukiwania oraz filtrowania po wybranych atrybutach produktu.

Wyszukiwanie

- realizowane za pomocą parametru `query`
- atrybuty brane pod uwagę podczas wyszukiwania
    - nazwa produktu
    - nazwa producenta
- przykład:

`/catalog?query=vivo`

Filtrowanie

- realizowane za pomocą zestawu parametrów konstruowanych na 2 sposoby
- przedział wartości
    - atrybut musi znajdować się w zadanym przedziale
    - przyrostki *From oraz *To
    - przykłady:

`/catalog?priceForm=1000&priceTo=2000` (cena od 1000 zł do 2000 zł)
`/catalog?ramSizeFrom=8&ramSizeTo=32` (pojemność pamięci RAM od 8GB do 32GB)

- lista akceptowanych wartości
    - atrybut musi przyjmować jedną z akceptowanych wartości
    - przyrostki *1, *2, *3 itd.
    - przykłady:

`/catalog?manufacturer1=Asus&manufacturer2=Lenovo` (producent Asus lub Lenovo)
`/catalog?hardDriveType1=SSD&hardDriveType2=HDD` (typ dysku SSD lub HDD)

`GET /catalog/{productId}` - szczegóły produktu o podanym ID

Parametry

- productId - id produktu

Format odpowiedzi

- lista elementów struktury BasicProductInfo, zawierającej:
    - ID produktu
    - producenta
    - nazwę modelu
    - cenę
    - URL do zdjęcia
- możliwe kody HTTP
    - 200 - produkt znajduje się w bazie
    - 404 - nie znaleziono produktu w bazie

`GET /catalog/metadata/{attributeName}` - lista unikatowych wartości danego atrybutu zapisanych w bazie danych

Parametry

- attributeName - nazwa atrybutu (zwykły tekst)

Format odpowiedzi

- lista wartości typu String będących unikatowymi wartościami atrybutu
- możliwe kody HTTP
    - 200 - podany atrybut istnieje
    - 404 - podany atrybut nie istnieje (tzn. liczba jego wystąpień w dokumentach jest równa 0)

5.2. Koszyk

`GET /users/{userId}/cart` - pobranie zawartości koszyka

Parametry
- userId - ID użytkownika

Format odpowiedzi
- lista elementów struktury BasicProductInfo, zawierającej:
    - ID produktu
    - producenta
    - nazwę modelu
    - cenę
    - URL do zdjęcia
- możliwe kody HTTP
    - 200 - koszyk znajduje się w bazie i jest niepusty
    - 404 - koszyka nie znaleziono w bazie lub jest pusty

`POST /users/{userId}/cart` - dodanie produktów do koszyka

Parametry
- userId - ID użytkownika

Format odpowiedzi
- możliwe kody HTTP
    - 200 - koszyk istniał już wcześniej
    - 201 - utworzono koszyk

`DELETE /users/{userId}/cart` - usunięcie zawartości koszya

Parametry
- userId - ID użytkownika

Format odpowiedzi
- możliwe kody HTTP
    - 200 - koszyk istniał
    - 404 - koszyk nie istniał

`DELETE /users/{userId}/cart/{productId}` - usunięcie jednego produktu z koszyka

Parametry

- userId - ID użytkownika
- productId - ID produktu

Format odpowiedzi

- możliwe kody HTTP
    - 200 - podany produkt znajdował się w koszyku
    - 404 - podany produkt nie znajdował się w koszyku

`POST /users/{userId}/cart/checkout` - utworzenie zamówienia z zawartości koszyka

Parametry

- userId - ID użytkownika

Format odpowiedzi

- możliwe kody HTTP
    - 200 - koszyk istniał
    - 404 - koszyk nie istniał

5.3. Zamówienia

`GET /orders/{orderId}` - pobranie informacji o zamówieniu

Parametry
- orderId - ID zamówienia

Format odpowiedzi
- możliwe kody HTTP
    - 200 - znaleziono zamówienie
    - 404 - nie znaleziono zamówienie

`POST /orders/{orderId}` - oznaczenie zamówienia jako opłacone

Parametry
- orderId - ID zamówienia

Format odpowiedzi
- możliwe kody HTTP
    - 200 - znaleziono zamówienie
    - 404 - nie znaleziono zamówienie

`GET /users/{userId}/orders` - pobranie listy zamówień danego użytkownika

Parametry
- userId - ID użytkownika

Format odpowiedzi
- list elementów struktury SummaryOrderInfo zawierającej
    - ID zamówienia
    - listę struktur BasicProductInfo
    - łączną cenę zamówienia
    - datę złożenia zamówienia
- możliwe kody HTTP
    - 200 - lista zamówień może zawierać 0 lub więcej elementów

6. Podział prac

Dominik Wiszyński:

- dodanie search bara
- dodanie filtrów
- dodanie widoku dla wszystkich laptopów
- zapisanie struktury bazy danych
- zapisanie stacku technologicznego
- tworzenie testów dla komponentów z frontendu
- stworzenie bazy danych

Radosław Panuszewski:

- zaprojektowanie endpointów
- stworzenie zapytań do bazy danych
- stworzenie testów do zapytań
- stworzenie endpointu zwracającego informacje o wszystkich produktach
- stworzenie endpointu zwracającego informacje o konkretnym produkcie
- stworzenie przetworzenia zamówienia w backendzie
- zwracanie przefiltrowanych produktów
- stworzenie api dla koszyka
- zabezpieczenie wrażliwych ścieżek za pomocą Auth0
- testy backendu

Jakub Tokarzewski:

- zapisywanie koszyka do backendu
- tworzenie testów dla reducerów i actions
- dodanie możliwości dodania do koszyka
- stworzenie koszyka
- dodanie auth do reduxa
- poprawa wyglądu loginu
- dodanie Auth0 do aplikacji
- zaprojektowanie widoku produktów
- dodanie routingu do frontendu
- stworzenie footera
- dodanie bootstrapa
- stworzenie headera
- dodanie reduxa
- stworzenie bazy frontendu

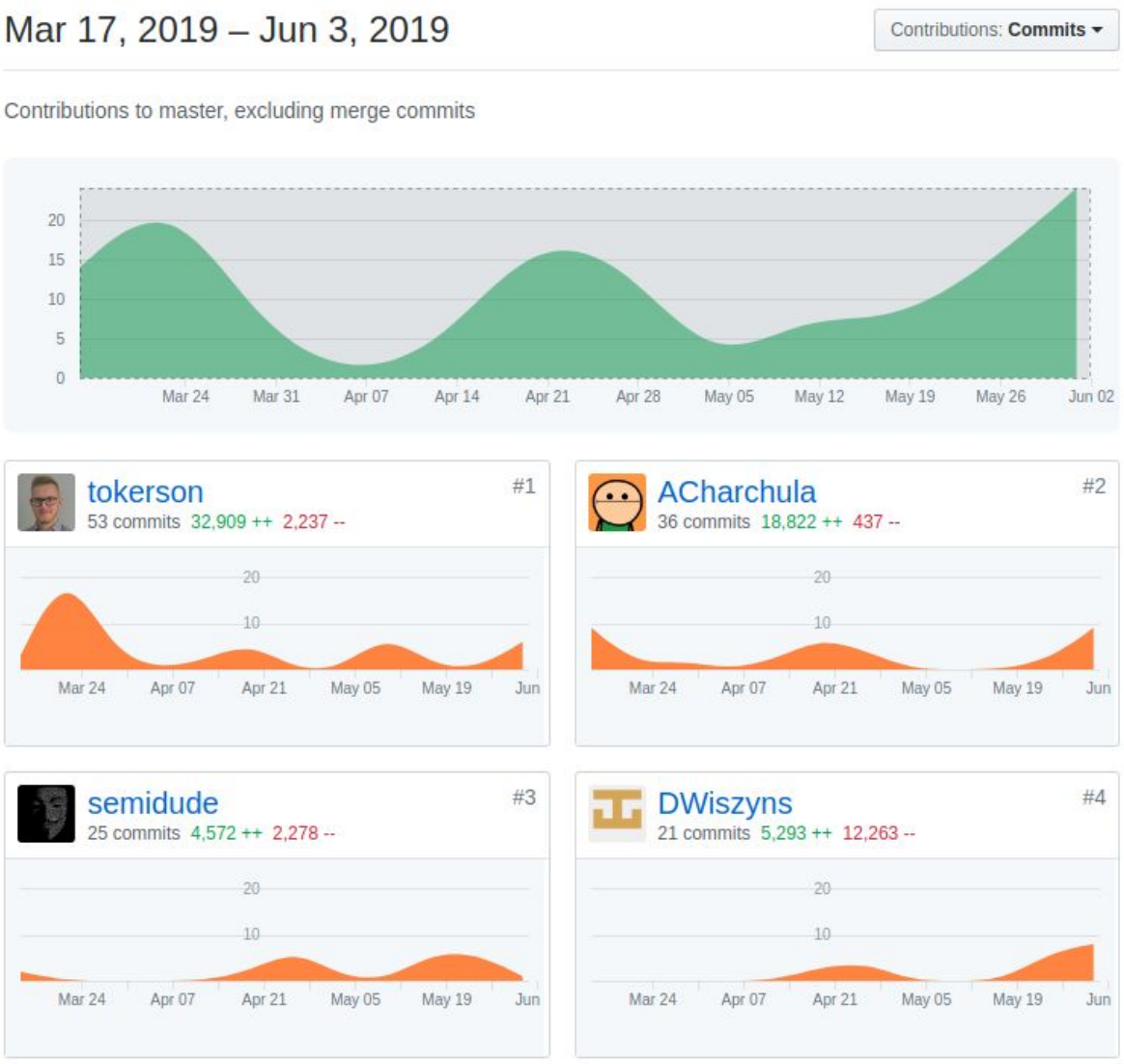
Antoni Charchuła:

- dodanie widoku użytkownika
- umożliwienie otrzymania backendowego api z przeglądarki
- dodanie paymentu - frontend
- dodanie widoku dla konkretnego produktu
- naprawa routingu w heroku
- dodanie Auth0 do aplikacji
- połączenie backendu z backendem

- stworzenie nexusa i połączenie z aplikacją
- konfiguracja heroku i CI travisa

7. Częstotliwość i intensywność kontrybuowania

Ilość dodanych linii kodu dla tokerson i ACharchula są niepoprawne ze względu na dodawanie node_modules.



8. Testy

Pokrycie kodu testami jednostkowymi Backend = 96%
Pokrycie kodu testami jednostkowymi Frontend = 20%