1.1 Opis

Wymagania oprogramowania, które określają kryteria oceny jakości jego pracy. W przeciwieństwie do wymagań funkcjonalnych, które określają, co powinien robić system, wymagania niefunkcjonalne określają, jaki powinien być system.

1.2 Wymagania niefunkcjonalne

1.2.1 Wymagania dotyczące interfejsu

Interfejs użytkownika oprogramowania powinien być współpracować z dowolną współczesną przeglądarką internetową, np. Chrome, Mozilla, Safari czy Opera, przy pomocy której użytkownik będzie mógł uzyskać dostęp do aplikacji. Dodatkowo wymagana jest włączona obsługa JavaScript.

1.2.2 Interfejsy sprzętowe

Aby uruchomić aplikację mieć dostęp do sieci, dlatego cały asortyment techniczny wymagany do połączenia z siecią Internet będzie dla systemu interfejsem sprzętowym. Zaliczamy do niego m.in. modem, WAN-LAN, kabel ethernetowy, etc.

1.2.3 Interfejsy komunikacyjne

Portal do komunikacji przez internet użyje protokołu HTTPS i metod z nim powiązanych. Port potrzebny do połączenia to port 443 w protokole TCP. Urządzenie powinno również posiadać logiczny adres systemu w formacie IPv4, aby zapewnić identyfikację.

1.2.4 Wymagania operacyjne

- Warstwa klienta

Warstwa kliencka portalu został zaprojektowany przy użyciu tagów HTML i formatowania CSS. Framework Bootstrap adaptuje portal do wyświetlania na mniejszych i większych urządzeniach. Język JavaScript i biblioteka JQuery odpowiadają za poruszanie się po aplikacji i obsługę zdarzeń.

- Warstwa serwera

Warstwa serwera został napisana w języku Python z wykorzystaniem frameworka Django. Cały serwis opiera się na renderowaniu widoku po stronie klienta, wykorzystane jest do tego REST API napisane z wykorzystaniem biblioteki djangorestframework. Serwer wystawia API z którym łączy się klient i pobiera dane. Wszystko przechowywane jest w bazie danych SQLite z wykorzystaniem mechanizmu ORM.

Obsługa i działanie aplikacji

Aplikacja została zamieszczona na platformie Heroku, która jest platformą do hostowania aplikacji w chmurze. Dzięki integracji Heroku z GitHubem, proces wdrożenia kodu odbył się przy pomocy wbudowanego systemu łączenia konta GitHub z hostingiem, dzięki któremu pliki ze zdalnego repozytorium zostały automatycznie połączone z platformą.

Aby aplikacja działała poprawnie:

1.Utworzony został plik *Procfile*, w którym została zawarta deklaracja jakie polecenie należy wykonać aby uruchomić aplikację.

2. Utworzony został plik *requirements.txt* w celu poprawnego rozpoznania aplikacji w języku Python.

3. Dodany został skrypt *release-script.sh*, aby rozwiązać problem z nieprawidłową konfiguracją bazy danych.

Aplikacja posiada unikalną domenę *b00kswapp.herokuapp.com*, której używamy do kierowania żądań HTTP do odpowiedniego *dyno*. *Dyno* to izolowane i zwirtualizowane kontenery systemu Linux, które są przeznaczone do wykonywania kodu na podstawie polecenia określonego przez użytkownika.

Dyno dzielimy na:

- *Web dyno* – odbierają ruch http z routerow;
- *Worker dyno* – używane są do wykonywania zadań w tle, zadań czasowych i systemów kolejkowania.

Opis działania:

1. Aplikacja otrzymuje żądanie sieciowe;
2. Żądanie jest dostarczane do losowego web dyno;
3. Żądanie jest umieszczone w kolejce a web dyno zwraca użytkownikowi komunikat o powodzeniu;
4. Worker dyno odbiera żądanie z kolejki i wykonuje pracę;
5. Worker dyno może zatwierdzić wynik pracy w bazie danych, który może zostać zwrócony użytkownikowi w innym żądaniu.

Schemat działania aplikacji:

1. Wdrażanie kodu przez programistę
2. Aplikacja działająca na dyno
3. Programista zarządza aplikacją przez panel
4. Użytkownicy wysyłają żądania, które są obsługiwane przez aplikację.