**Testowanie**

Testowanie to ważny element przy tworzeniu infrastruktury informatycznej,

Pozwalający na tworzenie kodu wysokiej jakości, zminimalizowanie ilości możliwych błędów przy przyszłym użytkowaniu oprogramowania. Testowanie naszej aplikacji zostało podzielone na kilka etapów. Mobilna wersja została sprawdzona manualnie przy pomocy programu Android Studio.

Pobrana została najnowsza wersja za pomocą systemu kontroli wersji Git i sprawdzona pod kątem potencjalnych błędów, wykorzystując emulgatory różnych urządzeń mobilnych.

Kolejnymi technologiami użytymi przy testach są:

Postman to darmowe narzędzie rozwijane od ponad 10 lat, cieszący się największą popularnością na rynku oprogramowania API. Aplikacja jest rozwijana przez wiele lat przez wielu twórców, którzy odpowiadają na sugestie użytkowników, dodając nowe funkcjonalności. Postman to bardzo potężne narzędzie, które poprawia wygodę testowania oprogramowania. Służy głównie do wysyłania żądań http, pozwala na tworzenie kolekcji i folderów z zapytaniami, które usprawniają pracę ułatwiając współpracę z innymi.Graphical user interface, text, application, email

Description automatically generated

Rysunek Przykładowe żądanie http GET. Źródło: Opracowanie własne

Swagger jest użytecznym frameworkiem pozwalającym na wizualizowanie i korzystanie z API aplikacji, co przydatne jest przy tworzeniu dokumentacji. Skonfigurowanie swaggera jest bardzo proste, ułatwiając projektowanie interfejsu API. Dokumentacja jest aktualizowana automatycznie co jest jedną z większych korzyści tego oprogramowania. Swagger to framework, który pozwala wizualizować i korzystać z API aplikacji, przy okazji tworząc dokumentację. Podstawowa konfiguracja Swaggera nie jest zbyt trudna, trwa zwykle nie więcej niż dwadzieścia minut. Swagger sprawia, że projektowanie interfejsu API jest znacznie prostsze.

Główna funkcjonalnością Swagger w naszej pracy było sprawdzenie prawidłowości w działaniu kontrolerów, weryfikowanie potencjalnych błędów jak np. przechodzenie żadąń wstawiających nieprawidłowe dane.

Graphical user interface, application

Description automatically generated

Rysunek Przykładowe żądanie http GET. Źródło: Opracowanie własne

**Testy jednostkowe**

Testy jednostkowe znajdują swoje zastosowanie w sprawdzania poszczególnych funkcjonalności programu. Przeprowadza się je w celu znalezienia potencjalnych błędów w pojedynczych fragmentach programu, aby zweryfikować czy żadny błędny komponent nie zaburza funkcjonowania zgodnego z wymaganiami. Wykonywane są w trakcie rozwijania aplikacji by jak najszybciej wykryć błędnie napisane fragmenty kodu.

Wśród ich zalet możemy wymienić:

* Poprawienie jakości napisanego kodu
* Wykrycie błędów w kodzie przed dopuszczeniem go na produkcję
* Ułatwienie w zrozumieniu funkcjonalności programu

W języku Java testy zautomatyzowane testy jednostkowe tworzone są przy pomocy frameworku JUnit. Udostępnia on funkcjonalności takie jak odpowiednie środowisko testerskie, automatyczne uruchomianie testów, asercje do porównywania oczekiwanych wyników z rzeczywistymi, a także adnotacje, którymi oznacza się metody do testowania kodu. Logika testów w wyżej wymienionym frameworku polega na tworzeniu metod testujących odnoszących się do rzeczywistych metod, sprawdzających czy rzeczywiste działają prawidłowo i zgodnie z oczekiwaniami i wymaganiami. Testowa metoda porównuje wynik oczekiwany, z wynikiem rzeczywistym zwracanym przez metodę. Głównym testowanym aspektem przez testy jednostkowe były serwisy. Na rysunku nr. 3 i 4 znajdują się przykładowe testy wspomnianych wyżej serwisów.

JaCoCo plugin to wtyczka umożliwiająca wygenerowanie raportu z pokrycia kodu źródłowego testami. Każdy Pull Request wyświetla pokrycie procentowe projektu na moment stworzenia PRa, co możemy zobaczyć na rysunku nr. 5.

Text

Description automatically generated

Rysunek Przykładowy test jednostkowy serwisu. Źródło: Opracowanie własne

Text

Description automatically generated with medium confidence

Rysunek Przykładowy test jednostkowy serwisu. Źródło: Opracowanie własne.

Graphical user interface, text, application, email

Description automatically generated

Rysunek Pokrycie testów w Pull Request na GitHub. Źródło: Opracowanie własne

**Manualne testy integracyjne**

Do wykonania manualnych testów API wykorzystywane było narzędzie Swagger ze względu na prostotę użytkowania. Przy jego pomocy, wysyłając żądania HTTP sprawdzaliśmy prawidłowość zaimplementowanych funkcjonalności.

Przykładowy proces testowania polega na wysłaniu żądania GET na adres wybranej do testowania końcówki.

W odpowiedzi otrzymujemy listę wszystkich karetek, której zawartość możemy porównać ze stanem bazy danych w celu zweryfikowania poprawności działania.

Graphical user interface, application

Description automatically generated

Rysunek Przykładowe żądanie HTTP GET. Źródło: Opracowanie własne

Testowanie aplikacji mobilnej zostało przeprowadzone przy pomocy Android Studio. Oprogramowanie to umożliwia uruchomienia emulatora urządzenia mobilnego z systemem Android. Uruchomiliśmy najnowszą wersję APK i manualnie sprawdziliśmy czy wszystkie funkcjonalności działają zgodnie z oczekiwaniami (Rysunek nr. 7).

A screenshot of a cell phone

Description automatically generated with medium confidence

Rysunek Aplikacja moblina w emulatorze Android Studio. Źródło: Opracowanie własne