TESTER OPROGRAMOWANIA

1. **Proces testowy.**

Analiza

i projektowanie

Implementacja i wykonanie

Ocena kryteriów zakończenia

I raportowanie

Raportowanie błędów

Analiza statusu

Weryfikacja wyników

Tworzenie raportów

z testów

Implementacja testów

Wykonanie testów

Scenariusze testowe

Dane testowe

Kontrola aktualnej sytuacji

Planowanie kolejnych zadań

Planowanie

i nadzór

Czynności zamykające

testy

1. **Rodzaje testów.**

**Testowanie manualne** - to testowanie ręczne czyli takie, w którym nie biorą udziału żadne narzędzia automatyzujące, a sam tester pełni rolę użytkownika końcowego.

**Testowanie automatyczne** – testowanie za pomocą narzędzi.

**Testowanie biało skrzynkowe** – testowanie oparte na analizie wewnętrznej struktury modułu lub systemu (wgląd w kod programu).

**Testowanie czarno skrzynkowe** – testowanie funkcjonalne lub niefunkcjonalne, bez odniesienia do wewnętrznej struktury modułu lub systemu (nie mamy wglądu do kodu). Coś „wchodzi”, „coś” wychodzi.

**Testy jednostkowe (modułowe)** - testowanie wyizolowanych fragmentów, na najniższym poziomie.

To np. zbiór testów (prób), które weryfikują, czy jednostka kodu (np. klasa, serwis itp.) działa zgodnie z oczekiwaniami.

**Testy funkcjonalne** – testowanie oparte na analizie specyfikacji funkcjonalnej modułu lub systemu (testy czarno skrzynkowe).

**Testy eksploracyjne** – nieformalna technika projektowania testów, w której tester projektuje testy podczas ich wykonywania oraz wykorzystuje informacje zdobyte podczas testowania do projektowania nowych i lepszych testów (ISTQB).

**Testy integracyjne** – testowanie wykonywane w celu wykrycia defektów w interfejsach i interakcjach pomiędzy modułami lub systemami (działanie modułu w aplikacji).

**Testy użyteczności** - testowanie mające na celu określenie, w jakim stopniu oprogramowanie jest zrozumiałe, łatwe do nauczenia, łatwe w użyciu oraz atrakcyjne dla użytkowników w określonych warunkach (ISTQB).

**Testy akceptacyjne** - testowanie formalne uwzględniające potrzeby użytkownika, wymagania i procesy biznesowe przeprowadzane w celu umożliwienia użytkownikowi, klientowi lub innemu uprawnionemu podmiotowi ustalenia, czy zaakceptować system lub moduł (ISTQB).

**Testy regresyjne** - kiedy defekt zostaje znaleziony i naprawiony wtedy oprogramowanie powinno być ponownie przetestowane aby upewnić się, że defekt został usunięty. Taką czynność nazywamy testami potwierdzającymi.

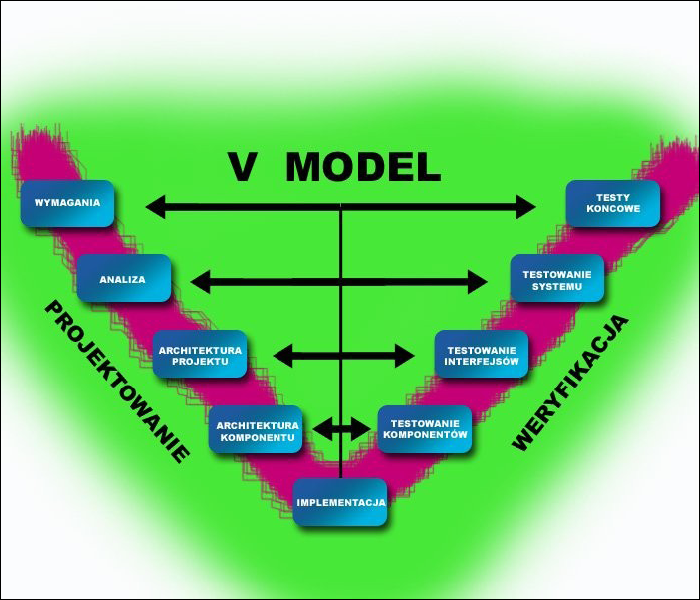
**Test dymny (smoke test)** – podzbiór wszystkich zdefiniowanych/zaplanowanych przypadków testowych, które pokrywają główne funkcjonalności modułu lub systemu, mający na celu potwierdzenie, że działają kluczowe funkcjonalności programu, bez zagłębiania się w szczegóły (ISTQB).

**Testy wydajnościowe** - testowanie wydajności to próba obciążenia serwera, bazy danych oraz samej aplikacji w oparciu o przygotowane scenariusze użycia (np. wyszukiwanie, zakładanie konta, korzystanie z forum, itd.) oraz o wygenerowanych wirtualnych użytkowników, którzy dane scenariusze "wykonują". Wynikiem testu jest raport o niewydajnych elementach i rekomendacje modyfikacji. Wskazujemy krytyczne punkty systemu negatywnie wpływające na wydajność końcowego produktu (testerzy.pl). Mogą uszkodzić produkt.

**Testy obciążeniowe** - testowanie, podczas którego system pracuje przy dużej ilości danych (ISTQB). Mogą uszkodzić produkt.

**Testowanie bezpieczeństwa** – testowanie mające na celu określenie bezpieczeństwa oprogramowania.

1. **Model V.**

****

**Model V** – jest to opis czynności cyklu życia wytwarzanego oprogramowania od specyfikacji wymagań do pielęgnacji. Model V ilustruje jak czynności testowe mogą być integrowane z każdym elementem cyklu życia wytwarzania oprogramowania.

1. **Tworzenie testów.**

Nazewnictwo

1 2 3 4

Przypadek testowy

Dane testowe

Scenariusz testowy

Przypadek użycia

**Ad. 1:** - jak coś powinno działać (opis).

**Ad. 2:** - jest to zbiór przypadków testowych, czyli kroków służących do sprawdzenia systemu.

**Ad. 3:** - zbiór danych wejściowych, wstępnych warunków wykonania, oczekiwanych rezultatów i końcowych warunków wykonania opracowany w określonym celu lub dla warunku testowego, jak wykonanie pewnej ścieżki programu lub zweryfikowanie zgodności z konkretnym wymaganiem

**Ad. 4:** - dane, które istnieją (przykładowo w bazie danych) przed wykonaniem testu, i które mają wpływ na testowany moduł lub system, lub na które wywiera wpływ testowany moduł lub system (dane użyte w scenariuszu).

**Przykład:**

*Przypadek testowy*

|  |  |  |  |
| --- | --- | --- | --- |
| Lp. | **Instruction (polecenie)** | **Expected resault (oczekiwany rezultat)** | **Actual resault**  **(aktualny rezultat)** |
| 1 | Kliknij przycisk „Zaloguj” | Aplikacja wyświetli ekran logowania | Aplikacja wyświetliła ekran logowania |
| 2 | Uzupełnij pole „Login” danymi | Użytkownik uzupełni pole „Login”, „Hasło” | Pole zostało uzupełnione loginem i hasłem użytkownika. |
| 3 | Kliknij przycisk „Zaloguj” | Użytkownik loguje się do aplikacji. | Użytkownik zalogował się do aplikacji. |

1. **Tytuł: [AA-XX] (JIRA)** Poprawne logowanie do aplikacji za pomocą email-a.
2. Opis: (skrócony opis tego co robi test – czego oczekujemy, czego dotyczy).
3. **Wymagania wstępne:** (np. wymagania, które muszą być spełnione do rozpoczęcia testu)

- aktywne konto założone za pomocą email-a.

4. **Dane testowe:** (wszystko, co jest potrzebne do testów)

- login: [test@wp.pl](mailto:test@wp.pl),

- hasło: zaq123.

5. **Oczekiwany rezultat:**

- po podaniu prawidłowego loginu i hasła użytkownik zostaje zalogowany do aplikacji.

6. **Aktualny rezultat:**

- po podaniu prawidłowego loginu i hasła użytkownik zalogował się do aplikacji.

*Zgłoszenie błędu*

**Tytuł: [AA-XX][Logowanie]** – (sekcja) Można zalogować się bez hasła do aplikacji.

**Krótki opis:** (np. dwuzdaniowy opis błędu).

**Zgłaszający: Przypisanie:**

**Priorytet: Waga/Szkodliwość:**

**Załączniki: Środowisko:** - wersja aplikacji,

- system,

- urządzenie,

- ważne informacje

Opis: (3 sekcje)

**Kroki:**

1. Otwórz aplikację.
2. Kliknij przycisk „Logowanie”.
3. Bez wprowadzania loginu i hasła kliknij przycisk „Zaloguj”.

**Oczekiwany rezultat (to, co powinno się stać):**

**-** wyświetla się informacja o błędnym logowaniu, np. „Niepoprawny login lub hasło”.

**Aktualny rezultat:**

- użytkownik zalogował się do aplikacji bez podania loginu i hasła.

1. **REDMINE**

**Do uzupełnienia**

1. **MR BUGGI 3**

**Do uzupełnienia**

1. **TESTLINK**

**Do uzupełnienia**

1. **POSTMAN**

POSTMAN jest bezpłatnym narzędziem, które możemy wykorzystać do testów API. Jest to narzędzie intuicyjne i proste w obsłudze oraz o całkiem sporych możliwościach. Program jest do pobrania tu: <https://www.getpostman.com/> . Po zainstalowaniu jest gotowy do użycia.

*Przykład testowania API.*

Dokumentacja API Opisane poniżej API pozwala dodawać i przeglądać trasy rowerowe z opisem punktów, przez które przebiega trasa, opisem, kolor, nawierzchnię itp.

Przykładowy zestaw danych jaki zawiera jedna trasa:

**"number": "1",**

**"title": "trasa I niebieska (17 km)",**

**"viaPoints": "Długa lista miejscowości",**

**"trackDescription": "Malownicza trasa biegnąca przez bagna i mokradła", "trackColor": "#2e79f2ff"**

API zawiera dwa poziomy:

Otwarty - lista tras rowerowych (tablica JSON-ów)

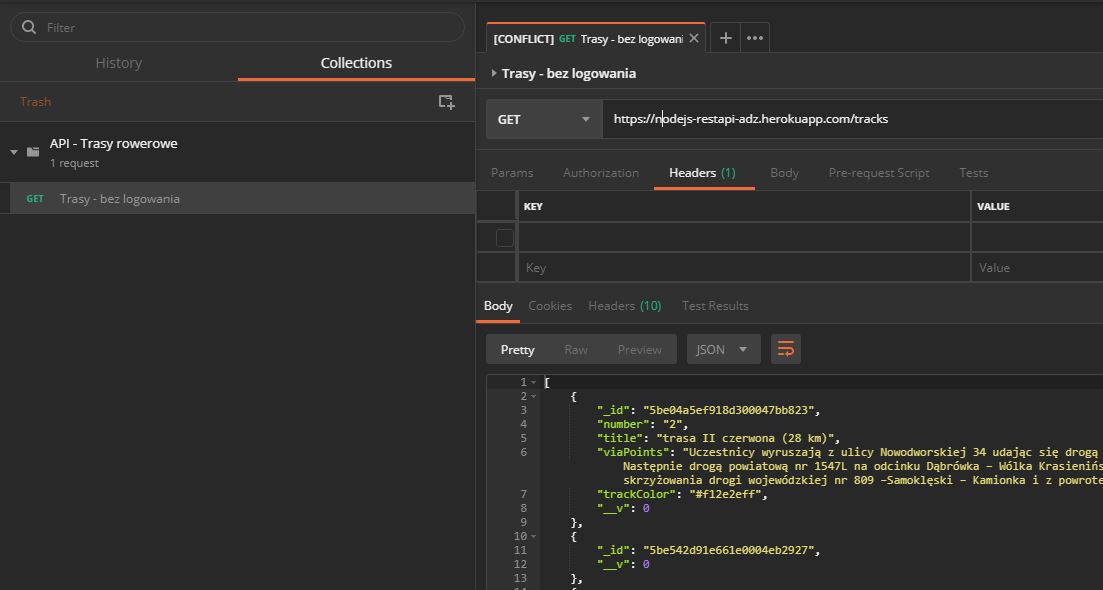
Zamknięty (dla zalogowanych) - dodawanie tras rowerowych

*Otwarty endpoint (bez logowania) - lista tras:*

**GET**

**https://nodejs-restapi-adz.herokuapp.com/tracks**

Zwraca tablicę JSON-ów z opisem tras rowerowych, które można dodać po rejestracji i zalogowaniu.

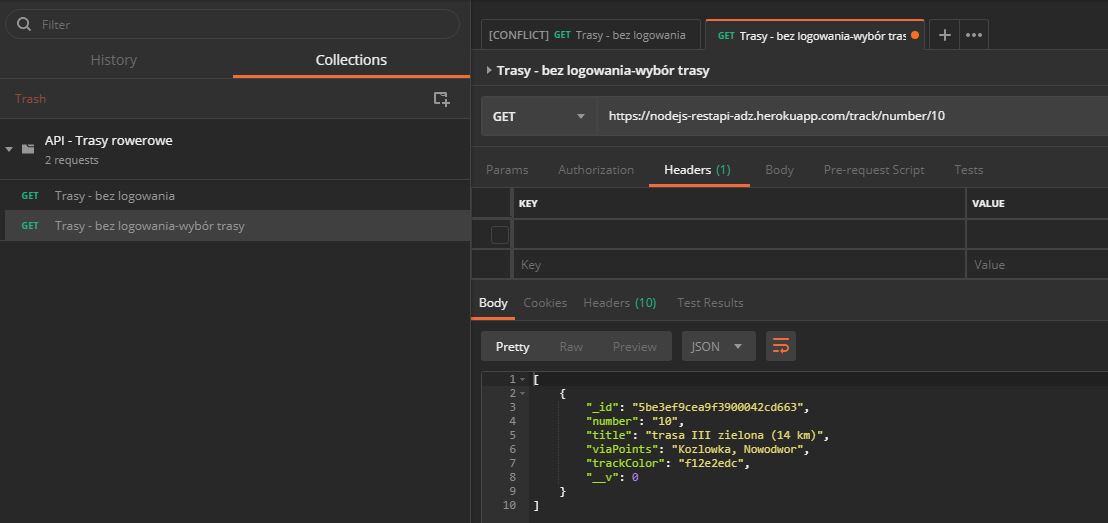


*Otwarty endpoint - wyszukiwanie tras po numerach.*

Dodatkowo API umożliwia wyszukiwanie tras rowerowych posługując się ich numerem:

**GET**

**https://nodejs-restapi-adz.herokuapp.com/track/number/10**

****

*Rejestracja*

Rejestracja użytkownika polega na wysłaniu zapytania POST z: imieniem, nazwiskiem, e-mailem i hasłem

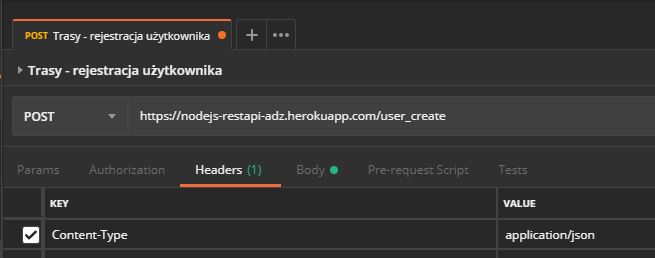
**POST**

**http://nodejs-restapi-adz.herokuapp.com/user\_create**

**HEADER**

**KEY: Content-Type,**

**VALUE: application/json**

****

**BODY**

example:

**{**

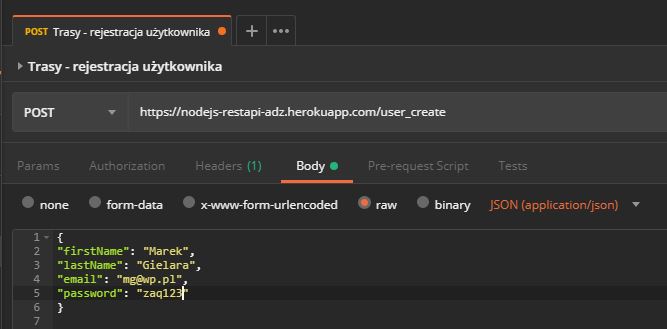
**"firstName": "Tomek",**

**"lastName": "Tomek",**

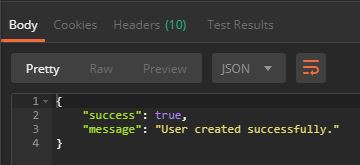
**"email": "tomek@wp.pl",**

**"password": "ttt"**

**}**



*Odpowiedź*



*Logowanie*

Następnie należy zalogować się wysyłając taki sam jak powyżej POST na

**POST**

**http://nodejs-restapi-adz.herokuapp.com/authenticate**

**HEADER**

**KEY: Content-Type**

**VALUE: application/json**

**BODY**

example:

**{**

**"firstName": "Imie",**

**"lastName": "Nazwisko",**

**"email": "email@wp.pl",**

**"password": "haslo"**

**}**

w odpowiedzi dostaniemy token:

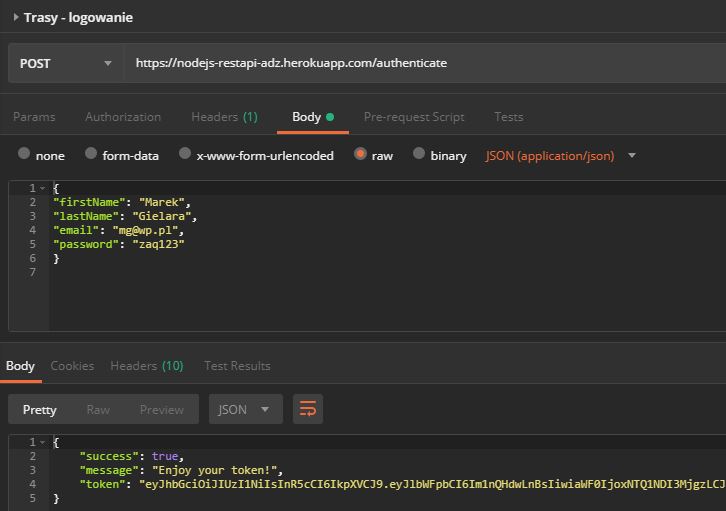
**{**

**"success": true,**

**"message": "Enjoy your token!",**

**"token": "eyxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"**

**}**



*Dodawanie tras z uwierzytelnieniem tokenem*

Token jest niezbędny do użycia end-pointu do dodawania tras:

**POST**

**https://nodejs-restapi-adz.herokuapp.com/api/track\_create**

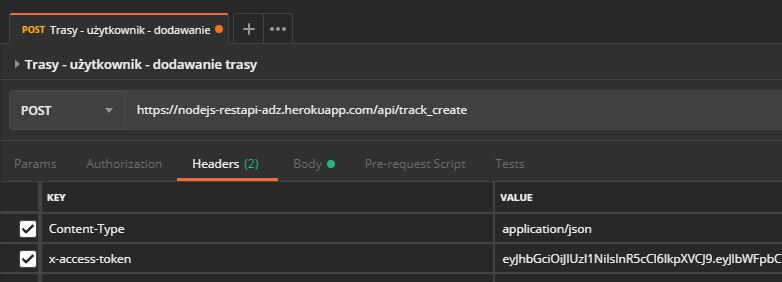
**HEADER**

**KEY: Content-Type**

**VALUE: application/json**

**KEY: x-access-token**

**VALUE: eyxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx**

****

**BODY**

example

**{**

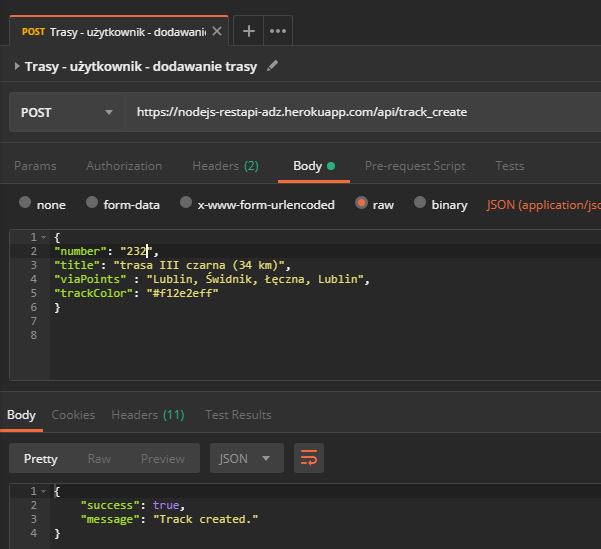
**"number": "232",**

**"title": "trasa III czarna (34 km)",**

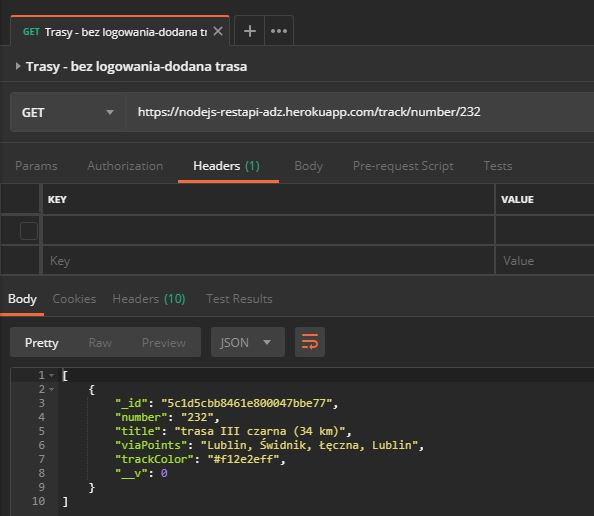
**"viaPoints" : "Lublin, Świdnik, Łęczna, Lublin",**

**"trackColor": "#f12e2eff"**

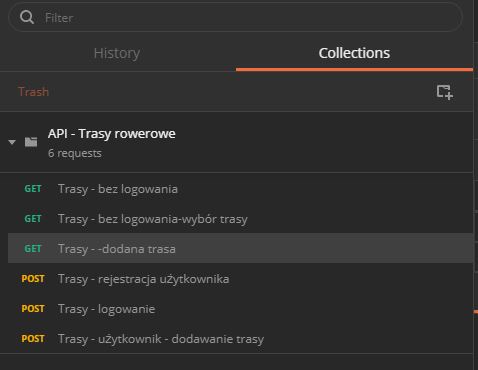
**}**

****

*Dodana trasa*

****

*Nasza kolekcja*

****

1. **JMeter**

Program JMeter służy do wykonywania testów wydajnościowych i obciążeniowych serwerów HTTP, FTP oraz zapytań do baz danych poprzez interfejs JDBC. Można testować zarówno elementy statyczne jak i dynamiczne: pliki, servlety Java, skrypty CGI, obiekty Java i inne. Możliwe jest także wykonywanie testów poprawności przy użyciu twierdzeń, aby dowiedzieć się czy strona zwraca właściwe wyniki. Można symulować różne poziomy obciążenia oraz badać wytrzymałość obciążeniową przy użyciu zdolności wielowątkowych programu udających wielu użytkowników. Wątki pozwalają także na jednoczesne testowanie różnych funkcji (zapisanych w oddzielnych thread groups). Rozszerzalność jest zapewniona poprzez moduły-wtyczki oraz opcję używania funkcji np. napisanych w JavaScript do modyfikacji danych wejściowych oraz elementów samplers, które są dostosowywalne dzięki obsłudye skryptów BeanShell. Własne komponenty zapisane w archiwach jar umieszczamy w podkatalogu /lib/ext lub ścieżce wskazywanej przez $JAVA\_HOME/jre/lib/ext. JMeter jest rozwijany przez projekt Apache Jakarta. Jest to projekt open source napisany całkowicie w Javie, dzięki czemu jest przenośny między platformami.

**Do uzupełnienia**

1. **Katalog Recorder (dodatek do FireFox-a)**

Najlepsza aplikacja do rejestrowania, odtwarzania i debugowania IDE Selenium. Eksportuje kod Selenium WebDriver. Zapewnia raporty, dzienniki, zrzuty ekranu. Fast & extensible.Katalon Automation Recorder to nowe IDE Selenium, które pomaga rejestrować akcje, przechwytywać elementy sieciowe w aplikacjach internetowych, odtwarzać zautomatyzowane przypadki testowe i szybko i łatwo tworzyć raporty.

Do uzupełnienia

1. **SQL**

Do uzupełnienia

1. **Scrum (Scrum Guide)**

Do uzupełnienia

1. **Chrome Developer Tools**

Do uzupełnienia

1. **Co to jest API, front-end, back-end, CMS, XMLHttpRequest (XHR)**

Application Programming Interface, **API** – sposób, rozumiany jako ściśle określony zestaw reguł i ich opisów, w jaki programy komputerowe komunikują się między sobą. Definiuje się go na poziomie kodu źródłowego dla składników oprogramowania, na przykład aplikacji, bibliotek, systemu operacyjnego.

**Front-end i back-end**

Początkowe oraz końcowe stadium procesu. Front-end jest odpowiedzialny za pobieranie danych od użytkownika oraz przekazanie ich do back-endu. Następnie back-end na podstawie tych danych wykonuje określone zadanie. Opcjonalnie front-end może pokazać użytkownikowi wyniki otrzymane od back-endu. Często stosowanym tłumaczeniem jest „fasada” i „wnętrze”.

**CMS -** [system zarządzania treścią](https://pl.wikipedia.org/wiki/System_zarz%C4%85dzania_tre%C5%9Bci%C4%85) (Content Management System) - aplikacja internetowa lub ich zestaw, pozwalająca na łatwe utworzenie serwisu WWW, jego aktualizację i rozbudowę

**XMLHttpRequest (XHR)** – obiekt języków skryptowych (np. JavaScript, JScript lub VBScript) przeglądarek internetowych umożliwiający przesyłanie żądań do serwera WWW za pomocą protokołu HTTP. Cechą charakterystyczną XMLHttpRequest jest możliwość wykonywania żądań już po załadowaniu się strony internetowej w trakcie interakcji z użytkownikiem. Otrzymane odpowiedzi serwera są wówczas wykorzystywane do modyfikacji załadowanego dokumentu. Możliwość asynchronicznego wykonywania żądań sprawia, że są one wykonywane w tle i nie przerywają interakcji użytkownika ze stroną, dynamicznie ją zmieniając. Treść odpowiedzi serwera najczęściej przekazywana jest w formatach XML, JSON lub czystym tekstem.

**Przykładowe kody odpowiedzi HTTP**

**200 - OK**

Zawartość żądanego dokumentu (najczęściej zwracany nagłówek odpowiedzi w komunikacji WWW Internetu).

**300 - Multiple Choices**

Wiele możliwości – istnieje więcej niż jeden sposób obsłużenia danego zapytania, serwer może podać adres zasobu, który pozwala na wybór jednoznacznego zapytania spośród możliwych

**400 - Bad Request**

Nieprawidłowe zapytanie – żądanie nie może być obsłużone przez serwer z powodu nieprawidłowości postrzeganej jako błąd użytkownika (np. błędna składnia zapytania)

**401 - Unauthorized**

Nieautoryzowany dostęp – żądanie zasobu, który wymaga uwierzytelnienia.

**404 - Not Found**

HTTP 404 lub Not Found - kod błędu odpowiedzi HTTP wskazującego na to, że klient jest w stanie komunikować się z serwerem, ale ten nie może znaleźć żądanego pliku, lub jest on skonfigurowany, aby nie spełnić żądania i nie wyświetlać informacji.

**500 - Internal Server Error**

Wewnętrzny błąd serwera – serwer napotkał niespodziewane trudności, które uniemożliwiły zrealizowanie żądania

**503 - Service Unavailable**

Usługa niedostępna – serwer nie jest w stanie w danej chwili zrealizować zapytania klienta ze względu na przeciążenie

1. **GIT**

Pobieramy Git-a ze strony <https://git-scm.com/> i instalujemy na kompie.

Podstawowe komendy

git init // Polecenie „git init” tworzy nowe repozytorium Git. Wykonywanie polecenia „git init” tworzy podkatalog .git w bieżącym katalogu roboczym.

git add (nazwa pliku lub katalogu) //

**Do uzupełnienia**

1. Java – Eclipse (Maven, Junit, Selenium)

**Do uzupełnienia**

1. Eclipse (Cucumber)

**Do uzupełnienia**

1. Ubuntu Server 16.04 (Docker, Jenkins)

**Do uzupełnienia**

1. SoapUI

**Do uzupełnienia**

1. **Rozmowa kwalifikacyjna – przykładowe pytania i odpowiedzi. Dzięki Paula ☺**
2. Różnica miedzy biało i czarno skrzynkowymi testami.

Biało skrzynkowe - wgląd do kodu - np. testy jednostkowe, czarno skrzynkowe - funkcjonalne, wydajnościowe.

1. Trójwarstwowy system aplikacji (Front-end, API, Back-end ).
2. Jaki jest powód wykonywania testów regresyjnych/regresywnych.

Sprawdzamy czy nic nie zepsuliśmy po dodaniu czegoś więcej.

1. Czym są testy eksploracyjne i kiedy je robimy?

Testy bez dokumentacji, robimy je zawsze, szukamy i kombinujemy jak to zepsuć, im więcej tym lepiej. Klient tego nie widzi.

1. Elementy przypadku testowego?

Tytuł, wymagania wstępne, dane testowe, opis, kroki, a w nich - instrukcja i oczekiwany rezultat. Kroki są pod opisem albo osobno.

1. Defekt- zgłoszenie błędu, co zawiera?

Tytuł , krótki opis, środowisko, kto wykonał, przypisane, priorytet, waga/szkodliwość, załączniki ( screeny, logi itp.) kroki, oczekiwany rezultat, aktualny rezultat. Najważniejsze są : tytuł i kroki.

1. Czym jest waga i priorytet błędu?

Np. literówka w nazwie firmy- szkodliwość żadna, ale priorytet bardzo duży.

1. Retesty, a testy regresyjne (regresywne)?

Retesty- testy po usunięciu usterki, kiedy defekt zostaje znaleziony i naprawiony wtedy ponawiamy testy, aby upewnić się, że defekt został usunięty. Taką czynność nazywamy testami potwierdzającymi.

Testy regresywne - ponowne przetestowanie uprzednio testowanego programu po dokonaniu w nim modyfikacji, w celu upewnienia się, że w wyniku zmian nie powstały nowe defekty lub nie ujawniły się defekty w niezmienionej części oprogramowania. Testy takie są przeprowadzane po zmianach oprogramowania lub jego środowiska pracy.

1. Analiza ryzyka, kiedy sie wykonuje?

Kiedy musimy cos przetestować, jest dużo do zrobienia, nie wiemy, co i jak, nie mamy czasu, wtedy lepiej poświęcić czas, żeby zrobić analizę ryzyka bledów,- wpływ/waga.

1. Matryca ryzyka.

Ustala się ją w zależności od rodzaju projektu. Ustala się priorytety.

|  |  |  |  |
| --- | --- | --- | --- |
| Wpływ  Szansa | Mały | Średni | Duży |
| Niska | Niski | Niski | Średni |
| Średnia | Niski | Średni | Duży |
| Wysoka | Średni | Duży | Krytyczny |

1. Cykl życia błędu ( defect life circle).

Zostaje zlokalizowany, in progress-odrzucony-rejected, blocked- nie mam wystarczającej informacji, i może trafić do naprawy, później code review- czyli przegląd i sprawdzenie czy nadaje sie do testów, jak jest dobrze to jest improve/done/ok, jak nie to wraca to reopened.

- zgłoszenie błędu (New),

- priorytetyzacja błędu,

- przypisanie,

- bieżący status (in progress – w toku, odrzucony - rejected, blocked – nie ma wystarczającej informacji, i może trafić do naprawy),

- (developer) odtworzenie błędu i decyzja (rejected, blocked, naprawa),

- błąd zostaje naprawiony (zmiana w kodzie przechodzi przegląd) – code review ,

- wraca do testów (ready for QA),

- w testach (in QA),

- jeśli jest dobrze (aproved),

- jeśli nie (reopened) – przechodzi wszystko od początku

1. Czym są „smoke testy”

Testy krótkie powdrożeniowe, po produkcji, sprawdzające czy ogólnie system działa i nie wywali sie, zazwyczaj w nocy są robione.

1. Sanity testy vs. smoke testy?

Różnica- smoke testy, czy w ogóle to działa- happy scenario;), sanity- w dużej mierze na tym samym, ale tyczy sie nowych funkcjonalności.

1. Hot fix.

Poprawki na gorąco - jeżeli coś wrzuciliśmy na produkcje, ale nagle coś nam znikło, wtedy roimy hot fixy i wraca to, co było.

1. Rodzaje testów niefunkcjonalnych ?

Bezpieczeństwa, jednostkowe, użytecznościowe.

1. Metodyki zwinne ( Scrum, Canban).
2. Inne metodyki wytwarzania oprogramowania.

Waterfall, prince 2, pmi

1. Scrum.

Składa sie ze sprintów (okres czasu na wytworzenie jakiejś części systemu i dostarczenie jej, jakiejś nowej funkcjonalności, jest to metoda przyrostowa i iteracyjna, powinna dodawać jakąś wartość za każdym razem, trwają od 1 do 4 tygodni), w ramach sprintu mamy spotkania - daily (co zrobiłem, co zamierzam zrobić i jakie napotkałem problemy), planowanie, review-na początku sprintu, aby wszelkie ewentualne poprawki zgłaszać, moment na zadawanie pytań, retrospektywa - żeby usprawnić działanie zespołu, co powinniśmy zmienić żeby na przyszłość było lepiej, zazwyczaj są one stand up.

1. Czym są testy funkcjonalne.

Żeby sprawdzić czy dewelepowany system zachowuje sie tak jak powinien.

1. Czym są testy niefunkcjonalne?

Nie sprawdzają konkretnej funkcjonalności tylko aspekt systemu, np. bezpieczeństwo.

1. Dlaczego integracja implementarna jest lepsza niż big bang?

Szybciej można wykryć błędy, niższy zakres prac, jeśli po trochu to możemy to wkomponować w jakiś system prac, jeśli na big bang-na raz- trudno znaleźć błąd, trzeba wszystko przeszukiwać.

1. Typowy proces testowy- jak wygląda?

Planowanie i nadzór , analiza i projektowanie, implementacja, ocena, raportowanie i zakończenie, aktualizacja statusu. W skrócie - zaplanować co robimy, później piszemy do tego testy, bo zazwyczaj nie ma funkcjonalności, następnie testujemy, retesty, raportujemy, wyniki z tego co udało sie zrobić i aktualizujemy sytuacje żeby wiedzieć co robimy dalej.

- planujemy, co robimy,

- piszemy scenariusze,

- testujemy (retesty, poprawki),

- raporty (wyniki),

- aktualizacja sytuacji

1. Wartości brzegowe/graniczne ( boundary values)?

Jeśli np. mamy mieć maksymalnie 100 użytkowników, to wartości brzegowe będą to 100 i 101.)

1. Klasy równoważności.

Zbiory danych dla których dane systemu są takie same.

1. Z jakimi technologami pracowaliście?

Na daną chwile- języki programowania- java, eclipse-ide, techonologie- html, css, sql, programy- Jira, Postman, Redmine, Jmeter.

1. Testy UAT- kiedy, po co itd. - różnica pomiędzy testami alfa i beta.

Alfa- testy akceptacyjne po stronie dewelopera- beta, po stronie klienta. Testy UAT zazwyczaj wykonywanie przez klienta, robi sie po to żeby klient zaakceptował produkt (po to żebyśmy mieli podkładkę).

1. Grupy odpowiedzi (server response codes).

( 100, 2xx- ok, 3xxx- tez ok ale przekierowanie, 4xx -bledy po stronie klienta- przeglądarki, strony, 5xx bledy serwera).

1. Czy używaliśmy Firebug czy Chrome developer tools ?

TAK :d.

1. Jak zweryfikować czy problem jest frontowy FE czy backendowy BE?

Trzeba zajrzeć do API, albo w Networku- zobaczyć czy jakieś requesty idą. Jak jest FE to jak znajdzie w API dane które zwraca CI BE, jeśli w API już jest źle to na pewno jest to api albo BE.

1. Czy korzystaliśmy z narzędzi PROXY ?

Np. Charles Proxy.

1. Jak w prosty sposób wysłać request?

Np. Postman.

1. 3 główne typy aplikacji mobilnych.

Natywne - cross platformowe - responsywne.

1. Oprogramowania mobilne.

Android, iOs, Windows.

1. Czy lepsze jest testowanie na symulatorze czy w prawdziwym urządzaniu.

Na urządzeniu zdecydowanie, na symulatorze wielu rzeczy nie ma, nie działają.

1. Typowo mobilne rodzaje testów.

Np. multiply resolution.

1. Technologie do nawigacji.

GPS, gLONASS, Wi-fi, BTS.

1. SQL- podstawowe zapytania -

Do uzupełnienia

1. Co to jest „join”.

Zawiera elementy z dwóch tabel które do siebie pasują. Inner to jest zwykły join.

1. Distinct.

Unikalność (bez powtórzeń).

1. Co jest class.

Żeby zapewnić reużywalnosc kodu.

1. Jak rozumiemy USABLITY- jak zmierzyć?

Instalujemy kilka popularnych aplikacji podobnych do naszej i porównujemy jej funkcjonalności z naszą.

1. Czym są testy wydajnościowe?

Testowanie wydajności to próba obciążenia serwera, bazy danych oraz samej aplikacji w oparciu o przygotowane scenariusze użycia (np. wyszukiwanie, zakładanie konta, korzystanie z forum, itd.) oraz o wygenerowanych wirtualnych użytkowników, którzy dane scenariusze "wykonują". Wynikiem testu jest raport o niewydajnych elementach i rekomendacje modyfikacji. Wskazujemy krytyczne punkty systemu negatywnie wpływające na wydajność końcowego produktu. Mogą uszkodzić produkt.·

1. Czym są testy obciążeniowe - testowanie, podczas którego system pracuje przy dużej ilości danych (ISTQB). Mogą uszkodzić produkt.
2. Czym jest strategia testów?

Dokument opisujący jakie rodzaje testów wchodzą w zakres i jakiego rodzaju narzędzia są wykorzystywane)

1. Dlaczego studzienki kanalizacyjne są okrągłe?

Po pierwsze - okrągłej nie da sie wrzucić do środka, po drugie są najbardziej odporne na nacisk, najbardziej wytrzymałe.

1. Skąd czerpiesz wiedzę?

Z internetu i od innych ludzi z branży.