Systemy bezpieczne i FTC

Wiarygodność systemów w pracy zespołowej - jak czynnik ludzki wpływa na różnice w implementacji specyfikacji

Szymon Bagiński
Artur Walasz

Prowadzący: Mgr inż. Tomasz Serafin

Styczeń 2018

Spis treści

Spis treści

1 Wstęp

1.1 Cel projektu

Wykorzystywanie nawet najlepszego oprogramowania, w którym zaimplementowano najbardziej zaawansowane technologie i najbezpieczniejsze algorytmy, nie jest w stanie zapewnić systemowi 100-procentowego bezpieczeństwa. Dzieje się tak dlatego, że w rozwoju i implementacji oprogramowania uczestniczą ludzie, którzy z natury mają skłonność do popełniania błędów. W rezultacie, ludzie, którzy są częścią systemu, zawsze będą najsłabszym ogniwem bezpieczeństwa systemu. Czynnik ludzki jest głównym powodem, dla którego udaje się tak wiele ataków na komputery i systemy.

Celem tego projektu było zbadanie wpływu czynnika ludzkiego na różne implementacje tej samej specyfikacji. W jego ramach stworzono dwie niezależne implementacje oprogramowania z jednolitym, ustandaryzowanym interfejsemm, sposobem testowania oraz wymaganiami funkcjonalnymi. Na ich podstawie wyciągnięto wnioski na temat tego, jak indywidualne podejście, interpretacja dostarczonych wymagań oraz wybór technologii wpływają na finalną wersję wytworzonego produktu.

1.2 Realizacja

Punktem wyjściowym do projektu było stworzenie wspólnej specyfikacji wymagań projektowych oraz zbioru testów akceptacyjnych [1] prostego serwisu aukcyjnego w architekturze klient-serwer.

Na potrzeby projektu zdefiniowano specyfikację interfejsu serwisu uwzględniającą następujące funkcje:

- Rejestracja użytkowników
- Logowanie użytkowników
- Tworzenie ofert typu:
  - akucja: “auction”
  - teraz: “buynow”
- Kupowanie / licytacja twoarów (ofert) przez użytkowników
- Przeglądanie ofert z filtracją
  - opis zawiera tekst zadany tekst
  - cena minimalna
  - cena maksymalna
  - ilość dostępnych przedmiotów
- Przeglądanie własnych ofert
- Obsługa własnych ofert:
  - modyfikacja ofert
  - usówanie ofert

Specyfikacja zawiera opis wszystkich wymaganych endpointów (dodatek A), w tym:

- adres url
- dozwolone metody HTTP
- wymagany format danych, które powinny zostać przesłane w zapytaniach POST
- oczekiwane statusy odpowiedzi w zależności od warunków

Dokładne wymagania dla konkretnych funkcji serwizu zostaną opisane w dalszej części dokumentacji. Porównane zostaną obydwie implementacjie pod kątem zgodności z dokumentacją oraz ze sobą wzajemnie.

2 Porównanie implementacji

2.1 Wybór technologii

Podstawową różnicą między dwiema powstałymi implementacjami jest użyta technologia. Każde z rozwiązań posiada oczywiście inny zestaw cech, niektóre rzeczy wykonuje się łatwiej kosztem innych. Każda z nich ma także inne ograniczenia, czy zachowania domyślne. Wybór ten zatem mocno rzutuje na różnice, które będą widoczne przy konkretnych funkcjach systemu, bądź na nakład pracy jaki trzeba włożyć przy spełnianiu konkretnych wymagań.

2.1.1 Implementacja 1

Do pierwszej implementacji postanowiono użyć języka programowania Rust [3]. Jest to kompilowalny język stworzony z myślą o tworzeniu stabilnych, bezpiecznych, i przede wszystkim wydajnych aplikacji.

Do stworzenia aplikacji, korzystającej z protokołu HTTP, skorzystano z biblioteki Rocket [2]. Jest to bardzo wygodne rozwiązanie. W połączeniu z menedżerem pakietów Cargo, pierwszą najprostszą aplikację można stworzyć w kilka minut. Rozszerzanie jej o kolejne funkcjonalności również nie przyspaża żadnych problemów. Jednocześnie zapewnia ona bezpieczeństwo danej aplikacji, elastyczność a przedwszystkim stabilność (m. in. poprzez "type safety"). W połączeniu z ogólnymi cechami języka Rust daje nam to duże prawdopodobieństwo, że skompilowany program, będzie działał bez problemów przez długi czas.

2.1.2 Implementacja 2

<TODO> Artur, jak leci </TODO>

2.2 Rejestracja

Mogłoby się wydawać, że tak prosty *endpoint* jak rejestracja nie powinien spowodować dużych rozbieżności. Niemniej jednak niedociągnięcia specyfikacji pozostawiają pole do różnych interpretacji.
W ciele zapytania POST podczas rejestracji wysyłana jest struktura JSON z dwoma polami: mail oraz password. Wartość pola mail musi być oczywiście niepowtarzalna. Istniej zatem możliwość aby użyć jej jako identfikatora danego użytkownika, zarówno w bazie danych jak i w aplikacji. Tak postanowiono zrobić w implementacji 1., gdzie mail jest kluczem pierwotnym dla użytkownika. <TODO> Artur, jak z tym mailem u cb? </TODO>
Następnie należy rozpatrzyć przypadek, którego specyfikacja nie przewiduje. Gdy nie uda się poprawnie zinterpretować przesłanych danych jako JSON, bibliotek Rocket obsłuży taki przypadek wysyłając status 422 (Unprocessable Entity). <TODO> Artur, jak z tym u Cb, dodaj wniosek jak możesz </TODO>
Specyfikacja określa co powinno się stać gdy zapytanie zostanie wysłane poprawnie za wyjątkiem typu metody HTTP. W takim przypadku powinien zostać zwrócony status 405 (Method Not Allowed). W przypadku implementacji 1. skutkuje to bardzo dużym nakładem mało istotnej pracy. Rocket, gdy nie uda mu się dopasować żadnego istniejącego endpointu, wysyła domyślnie status 404 (Not Found). Dzieje się tak również w przypadku gdy nie znajdzie odpowiedniej metody HTTP, nawet jeśli reszta parametrów się zgadza. Aby otrzymać zgodność ze specyfikacją należałoby więc utworzyć endpointy dla wszystkich możliwych metod protokołu HTTP, których jedyną odpowiedzialnością byłoby wysłanie błędu o statusie 405. W implementacji 1. zdecydowano się nadpisać tylko metody get, put oraz delete. <TODO> Artur, a jak z tym u Cb? </TODO>

2.3 Logowanie

2.4 Tworzenie ofert

2.5 Kupowanie / licytacja

2.6 Przeglądanie ofert wraz z filtrowaniem

3 Podsumowanie

Odwołania

[1] *Opis testów funkcji systemu* [online], Data dostępu: 12.01.2019. `https://drive.google.com/drive/folders/194qvJLmUYSD427WfbAkm365Q3SCY1ZaY`.

[2] *Rocket - Simple, Fast, Type-Safe Web Framework for Rust* [online], Data dostępu: 12.01.2019. `https://rocket.rs/`.

[3] *Rust* [online], Data dostępu: 12.01.2019. `https://www.rust-lang.org/`.

A Interfejs serwisu aukcyjnego

```
 /registration
POST
{
    mail:     string,
    password: string
}

201 - Created (pomyślne utworzenie użytkownika)
400 - Bad Request (np. zły email)
409 - Conflict (konto istnieje)
500 - Internal Server Error
503 - Server Unavaiable (powód podany w opisie)

not POST
405 - Method Not Allowed
-----------------------------------------------------------------
/login
POST
{
    mail:     string,
    password: string
}

200 - OK, (JWT token in response)
401 - Unauthorized
404 - Not Found
500 - Internal Server Error
503 - Server Unavaiable (powód podany w opisie)

not POST
405 - Method Not Allowed
-----------------------------------------------------------------
/offers
POST
{
    type*:        [auction|buynow],
    description*: string,
    price*:       float, (cena min. - akcja; cena za sztukę - kup teraz),
    date*:        timestamp, (sekundy, tylko dla aukcji),
    amount*:      int (tylko dla kup teraz)
}

201 - added
{
    offer_id: int
}
400 - Bad Request (niepoprawny JSON, brakujące lub nadmiarowe pola)
401 - Unauthorized (niezalogowany użytkownik)
403 - Forbidden (zalogowany na nieuprawnione konto)
500 - Internal Server Error
503 - Server Unavaiable (powód podany w opisie)

GET
Dozwolone filtry w url:
- contains - pole description zawiera ciąg znaków
- price_min - minimalna cena towaru (i aukcji)
- price_max - maksymalna cena towaru (i aukcji)
- type - [auction/buynow] - typ oferty
- created_by_me: boolean

200 - (oferty w odpowiedzi - może być pusta "[]")
400 - Bad Request (niepoprawny filtr w url)
500 - Internal Server Error
503 - Server Unavaiable (powód podany w opisie)
-----------------------------------------------------------------
/offers/{id}
PATCH
{
    **fields_to_modify...
}

użytkownik może modyfikować:
```

```
- price, amount, description - dla ofert typu "buynow"
- price, description - dla ofert typu "auction"

202 - Accpepted
400 - Bad Request
403 - Unauthorized
404 - Not Found (nie znaleziono ofert)

DELETE
(no payload)
202 - Accepted
403 - Unauthorized
404 - Not Found (nie znaleziono oferty)

GET
(no payload)
200 - Ok (w odpowiedzi szczegóły aukcji)
{
     type:          "auction",
     description:   "opis",
     status:        [active / expired],
     last_bid:      float,
     customer_id:   int,
     expiration_ts: <timestamp>
}
200 - Ok (szczegóły oferty "buynow")
{
     type:        "buynow",
     description: string,
     price:       int,
     amount:      int
}
404 - Not Found
-----------------------------------------------------------------
/offers/{id}/buy
POST
{
     bid:    int (auction)
     amount: int (buynow)
}

202 - Accepted
400 - Bad Request (niepoprawny JSON, brakujące lub nadmiarowe pola)
401 - Unauthorized (niezalogowany użytkownik)
409 - Conflict:
     409 - {"minimal_bid": <minimalny bid>}
     409 - {"max_amout": <ilość dostępnych produktóœ>}
     409 - {"status": "expired"}
     409 - {"conflict": "unable to order own items"}
     409 - {"conflict": "you can not bid on the auction you win"}
500 - internal server error
503 - Server Unavaiable (powód podany w opisie)
```