AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Raport końcowy testów aplikacji „ShoppingList”

Kierunek: *Informatyka i Systemy Inteligentne*
Specjalizacja*: Inżynieria Oprogramowania*
Przedmiot: *Testowanie i walidacja oprogramowania*
Rok: *2022/2023*

Prowadzący: *dr inż.*
Grzegorz Rogus

Autor:
Michał Pieniądz

1. Opis aplikacji

Aplikacja *ShoppingList* pozwala na zarządznie listami zakupów. Aplikacja dostępna jest tylko dla użytkowników zalogowanych. Proces logowania i rejestracji wykonuje użytkownik samodzielnie. Po zalogowaniu użytkownik ma możliwość definiowania list zakupów. Do każdej listy przypisuje się sugerowaną datę wykonania zakupu. Pojedynczy element listy zawiera nazwę, którą można wpisać samodzielnie lub wybrać spośród predefiniowanych popularnych propozycji, każdy elementy zawiera także ilość oraz gramaturę tego produktu. Możliwe jest dodanie zdjęcia jako elementu referencyjnego

2. Wykorzystane narzędzia i technologie

Aplikacja składa się z dwóch głównych części (backend i frontend).

- Frontend Angular CLI w wersji 14.2.7
- Backend Java Spring Boot w wersji 2.7.3.

Testy jednostkowe i integracyjne:

- Frontend:
  - Jasmine (5.1.0)
  - Karma (6.4.0)
- Backend:
  - Junit 5
  - Mockito

Testy systemowe end-to-end:

- Cypress (11.2.0)

Testy bezpieczeństwa:

- OWASP ZAP (2.12.0)

Testy wydajnościowe:

- Blazemeter (5.4.0)
- Jmeter (5.5)

3. Testy jednostkowe i integracyjne

W ramach testów jednostkowych i integracyjnych przeprowadzono testy:

- Frontend:
  - komponenty,
  - serwisy.
- Backend:
  - kontrolery,
  - serwisy,
  - repozytoria

Dla obu części aplikacji udało się osiągnąć wysokie pokrycie kodu testami:

- Frontend - blisko 99% pokrycia. Szczegółowy raport znajduje się w: *ShoppingList-Frontend/build/reports/coverage/report-html/index.html*

All files

98.98% Statements 292/295 **100%** Branches 23/23 **96.74%** Functions 119/123 **98.89%** Lines 269/272

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
app	100%	6/6	100%	0/0	100%	1/1	100%	6/6
app/components/login	95.83%	23/24	100%	1/1	85.71%	6/7	95.23%	20/21
app/components/main-navigation	92.85%	13/14	100%	2/2	80%	4/5	90.9%	10/11
app/components/register	95.65%	22/23	100%	1/1	88.88%	8/9	94.44%	17/18
app/components/shopping-list	100%	16/16	100%	1/1	100%	9/9	100%	14/14
app/components/shopping-list/add-list-item	100%	35/35	100%	2/2	100%	14/14	100%	34/34
app/components/shopping-list/edit-list-item	100%	42/42	100%	8/8	100%	15/15	100%	41/41
app/components/shopping-list/list-item	100%	12/12	100%	0/0	88.88%	8/9	100%	11/11
app/components/user-lists	100%	16/16	100%	3/3	100%	8/8	100%	14/14
app/components/user-lists/add-user-list	100%	16/16	100%	0/0	100%	7/7	100%	15/15
app/components/user-lists/edit-user-list	100%	14/14	100%	0/0	100%	7/7	100%	13/13
app/components/user-lists/user-list	100%	9/9	100%	0/0	100%	6/6	100%	8/8
app/models	100%	14/14	100%	0/0	100%	2/2	100%	14/14
app/services	100%	29/29	100%	2/2	100%	13/13	100%	27/27
app/services/auth	100%	18/18	100%	1/1	100%	9/9	100%	18/18
app/validators	100%	7/7	100%	2/2	100%	2/2	100%	7/7

- Backend – blisko 92% pokrycia. Szczegółowy raport znajduje się w: *ShoppingList-Backend/coverage/index.html*

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	95,2% (20/ 21)	95,1% (117/ 123)	91,7% (377/ 411)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
com.agh.shoppingListBackend.app	0% (0/ 1)	0% (0/ 2)	0% (0/ 2)
com.agh.shoppingListBackend.app.controllers	100% (4/ 4)	100% (21/ 21)	97,1% (99/ 102)
com.agh.shoppingListBackend.app.enums	100% (2/ 2)	100% (4/ 4)	100% (7/ 7)
com.agh.shoppingListBackend.app.models	100% (4/ 4)	100% (30/ 30)	100% (40/ 40)
com.agh.shoppingListBackend.app.security	100% (1/ 1)	100% (6/ 6)	100% (16/ 16)
com.agh.shoppingListBackend.app.security.jwt	100% (3/ 3)	100% (15/ 15)	77,4% (48/ 62)
com.agh.shoppingListBackend.app.security.services	100% (2/ 2)	92,3% (12/ 13)	81,8% (27/ 33)
com.agh.shoppingListBackend.app.services	100% (3/ 3)	92,9% (26/ 28)	98,3% (116/ 118)
com.agh.shoppingListBackend.app.utils	100% (1/ 1)	75% (3/ 4)	77,4% (24/ 31)

generated on 2022-12-04 00:07

3.1 Testy frontend:

Wszystkie poniższe testy zakładają wcześniejsze stworzenie testowanego elementu wraz z potrzebnymi zależnościami lub ich zastępnikami.

Wymagania dla środowiska

- Posiadanie node.js + npm kompatybilnej z bibliotekami z *package.json*
- Posiadanie lokalnej kopii kodu
- Posiadanie Angular CLI dla lokalnej instalacji node.js

Procedura testowania

- Uruchomienie komendy npm install w katalogu projektu
- Uruchomienie komendy ng test w katalogu projektu
- W celu wygenerowania raportu z pokrycia testami, uruchomienie *ng test --code-coverage*

LOGIN-COMPONENT

Nazwa:	LOGIN-COMPONENT - Prawidłowo wypełniony formularz powinien być prawidłowy
Warunki początkowe:	Brak
Dane testowe:	- nazwa użytkownika: „test” - hasło: „pass”
Procedura:	1. Wypełnienie formularza danymi
Warunki końcowe:	Formularz jest prawidłowy
Typ testu:	pozytywny

Nazwa:	LOGIN-COMPONENT - Przycisk Zaloguj powinien zawołać funkcję OnSubmit()
Warunki początkowe:	Brak
Dane testowe:	- nazwa użytkownika: „test” - hasło: „pass”
Procedura:	1. Wypełnienie formularza danymi. 2. Naciśnięcie przycisku zaloguj.
Warunki końcowe:	Funkcja OnSubmit() została zawołana.
Typ testu:	pozytywny

Nazwa:	LOGIN-COMPONENT - Tworzenie komponentu

Warunki początkowe:	- użytkownik jest niezalogowany
Dane testowe:	Brak
Procedura:	1. Utworzenie komponentu.
Warunki końcowe:	Component został prawidłowo utworzony.
Typ testu:	pozytywny

Nazwa:	LOGIN-COMPONENT - Pusty formularz powinien być fałszywy
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Pola formularza są puste
Warunki końcowe:	Formularz jest nieprawidłowy
Typ testu:	negatywny

Nazwa:	LOGIN-COMPONENT - Puste pole loginu powinno być nieprawidłowe
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Pole loginu jest puste
Warunki końcowe:	Pole jest loginu jest nieprawidłowe
Typ testu:	negatywny

Nazwa:	LOGIN-COMPONENT - Puste pole hasła powinno być nieprawidłowe
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Pole hasła jest puste
Warunki końcowe:	Pole jest hasła jest nieprawidłowe

Typ testu:	negatywny

Nazwa:	LOGIN-COMPONENT – Hasło powinno być typu password
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Odczytanie atrybutu type z pola password.
Warunki końcowe:	Pole powinno być typu ‘password’.
Typ testu:	pozytywny

Nazwa:	LOGIN-COMPONENT – Hasło powinno być widoczne po kliknięciu przycisku „pokaż hasło”
Warunki początkowe:	- Pole hasła jest typu password.
Dane testowe:	Brak
Procedura:	1. Kliknięcie przycisku „pokaż hasło”. 2. Odczytanie wartości atrubutu type z pola password.
Warunki końcowe:	Pole powinno być typu ‘text’.
Typ testu:	pozytywny

Nazwa:	LOGIN-COMPONENT – test OnSubmit() z pomyślną odpowiedzią serwisu logowania.
Warunki początkowe:	Brak
Dane testowe:	- nazwa użytkownika: „test” - hasło: „pass”
Procedura:	1. Wypełnienie formularza prawidłowymi danymi. 2. Wywołanie funkcji onSubmit(). 3. Zwrócenie pozytywnej wartości z serwisu logowania.
Warunki końcowe:	- Powinno nastąpić przekierowanie do list użytkownika. - isLoggedIn powinno mieć wartość true.
Typ testu:	pozytywny

Nazwa:	LOGIN-COMPONENT – test OnSubmit() z niepomyślną odpowiedzią serwisu logowania.

Warunki początkowe:	Brak
Dane testowe:	- odpowiedź z serwera ze statusem 401 - „Nieprawidłowa nazwa użytkownika lub hasło”.
Procedura:	1. Wywołanie funkcji onSubmit(). 2. Zwrócenie negatywnej wartości z serwisu logowania
Warunki końcowe:	- Nie powinno nastąpić przekierowanie. - Pola formularza powinny stać się nieprawidłowe. - Powinien zostać wyświetlony alert. - isLoggedIn powinien mieć wartość false.
Typ testu:	negatywny

Nazwa:	LOGIN-COMPONENT - test ngOnInit() kiedy użytkownik jest już zalogowany
Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	Brak
Procedura:	1. Wywołanie funkcji ngOnInit()
Warunki końcowe:	Użytkownik pomimo uruchomienia ekranu logowania powinien zostać przekierowany na swoje listy użytkowników.
Typ testu:	pozytywny

Nazwa:	LOGIN-COMPONENT – integracja – pomyślne logowanie
Warunki początkowe:	- użytkownik jest niezalogowany
Dane testowe:	- nazwa użytkownika: „test” - hasło: „pass”
Procedura:	1. Prawidłowe wypełnienie formularza. 2. Kliknięcie przycisku zaloguj.
Warunki końcowe:	- Serwis autentykacji powinien zostać zawołany. - Powinno zostać wysłane właściwe zapytanie do REST API - Użytkownik powinien zostać zapisany w storageService. - storageService.isLoggedIn() powinno zwracać true. - storageService.getUser() powinno zawierać dane zalogowanego użytkownika - Powinno nastąpić przekierowanie do list użytkownika.
Typ testu:	pozytywny

Nazwa:	LOGIN-COMPONENT – integracja – niepomyślne logowanie
Warunki początkowe:	- użytkownik jest niezalogowany
Dane testowe:	- nazwa użytkownika: „invalidName” - hasło: „invalidPass” - odpowiedź z serwera ze statusem 401 - „Nieprawidłowa nazwa użytkownika lub hasło”.
Procedura:	1. Prawidłowe wypełnienie formularza. 2. Kliknięcie przycisku zaloguj.
Warunki końcowe:	- Serwis autentykacji powinien zostać zawołany. - Powinno zostać wysłane właściwe zapytanie do REST API - Użytkownik nie powinien zostać zapisany w storageService. - storageService.isLoggedIn() powinno zwracać false. - storageService.getUser() nie powinno zawierać danych zalogowanego użytkownika, - Nie powinno nastąpić przekierowanie do list użytkownika. - Powinien zostać wyświetlony komunikat o niepomyślnym logowaniu.
Typ testu:	negatywny

REGISTER-COMPONENT

Nazwa:	REGISTER-COMPONENT - Tworzenie komponentu
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	2. Utworzenie komponentu.
Warunki końcowe:	Component został prawidłowo utworzony.
Typ testu:	pozytywny

Nazwa:	REGISTER-COMPONENT - Pusty formularz powinien być fałszywy
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	2. Pola formularza są puste

Warunki końcowe:	Formularz jest nieprawidłowy
Typ testu:	negatywny

Nazwa:	REGISTER -COMPONENT - Puste pole loginu powinno być nieprawidłowe
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	2. Pole loginu jest puste
Warunki końcowe:	Pole jest loginu jest nieprawidłowe
Typ testu:	negatywny

Nazwa:	REGISTER -COMPONENT - Puste pole hasła powinno być nieprawidłowe
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	2. Pole hasła jest puste
Warunki końcowe:	Pole jest hasła jest nieprawidłowe
Typ testu:	negatywny

Nazwa:	REGISTER -COMPONENT – za krótkie hasło powinno być nieprawidłowe
Warunki początkowe:	Brak
Dane testowe:	- hasło: „x”
Procedura:	1. Wpisanie za krótkiego hasła.
Warunki końcowe:	Pole hasła powinno być nieprawidłowe
Typ testu:	negatywny

Nazwa:	REGISTER -COMPONENT – za długie hasło powinno być nieprawidłowe

Warunki początkowe:	Brak
Dane testowe:	- hasło: 26 znaków „x”
Procedura:	1. Wpisanie za długiego hasła.
Warunki końcowe:	Pole hasła powinno być nieprawidłowe
Typ testu:	negatywny

Nazwa:	REGISTER -COMPONENT – prawidłowo wypełniony formularz powinien być prawidłowy.
Warunki początkowe:	Brak
Dane testowe:	- nazwa użytkownika: „test” - hasło: „pass”
Procedura:	1. Właściwe wypełnienie pól danymi.
Warunki końcowe:	Formularz powinien być prawidłowy.
Typ testu:	pozytywny

Nazwa:	REGISTER -COMPONENT – kliknięcie przycisku zarejestruj powinno zawołać onSubmit()
Warunki początkowe:	Brak
Dane testowe:	- nazwa użytkownika: „test” - hasło: „pass”
Procedura:	1. Właściwe wypełnienie pól danymi. 2. Kliknięcie przycisku zarejestruj.
Warunki końcowe:	Funkcja onSubmit() powinna zostać wywołana.
Typ testu:	pozytywny

Nazwa:	REGISTER-COMPONENT – Hasło powinno być typu password
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Odczytanie atrybutu type z pola password.

Warunki końcowe:	Pole powinno być typu ‘password’.
Typ testu:	pozytywny

Nazwa:	REGISTER -COMPONENT – Hasło powinno być widoczne po kliknięciu przycisku „pokaż hasło”
Warunki początkowe:	- Pole hasła jest typu password.
Dane testowe:	Brak
Procedura:	1. Kliknięcie przycisku „pokaż hasło”.
Warunki końcowe:	Pole powinno być typu ‘text’.
Typ testu:	pozytywny

Nazwa:	REGISTER -COMPONENT – test OnSubmit() z pomyślną odpowiedzią serwisu rejestracji.
Warunki początkowe:	Brak
Dane testowe:	- nazwa użytkownika: „test” - hasło: „pass”
Procedura:	1. Wypełnienie formularza prawidłowymi danymi. 2. Wywołanie funkcji onSubmit(). 3. Zwrócenie pozytywnej wartości z serwisu logowania.
Warunki końcowe:	- Powinien zostać wyświetlony alert o pomyślnym zarejestrowaniu - Powinno nastąpić przekierowanie do ekranu logowania.
Typ testu:	pozytywny

Nazwa:	REGISTER -COMPONENT – test OnSubmit() z niepomyślną odpowiedzią serwisu rejestracji.
Warunki początkowe:	Brak
Dane testowe:	- odpowiedź z serwera ze statusem 401 - „Something went wrong”
Procedura:	1. Wywołanie funkcji onSubmit(). 2. Zwrócenie negatywnej wartości z serwisu rejestracji.
Warunki końcowe:	- Nie powinno nastąpić przekierowanie. - Pola formularza powinny stać się nieprawidłowe. - Powinien zostać wyświetlony alert.
Typ testu:	negatywny

Nazwa:	REGISTER-COMPONENT - test ngOnInit() kiedy użytkownik jest już zalogowany
Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	Brak
Procedura:	1. Wywołanie funkcji ngOnInit()
Warunki końcowe:	Użytkownik pomimo uruchomienia ekranu rejestarcji powinien zostać przekierowany na swoje listy użytkowników.
Typ testu:	pozytywny

Nazwa:	REGISTER-COMPONENT – integracja – pomyślna rejestracja.
Warunki początkowe:	Brak.
Dane testowe:	- nazwa użytkownika: „test” - hasło: „pass”
Procedura:	1. Wypełnienie formularza danymi 2. Kliknięcie przycisku zarejestruj
Warunki końcowe:	- Powinien zostać zawołany serwis autentykacji - Powinno zostać wysłane właściwe zapytanie do REST API. - Powinien wyświetlić się komunikat o pomyślnej rejestarcji. - Powinno nastąpić przekierowanie na stronę logowania.
Typ testu:	pozytywny

Nazwa:	REGISTER-COMPONENT – integracja – niepomyślna rejestracja.
Warunki początkowe:	Brak.
Dane testowe:	- nazwa użytkownika: „test” - hasło: „pass” - odpowiedź z serwera ze statusem 401 - „Something went wrong”.
Procedura:	1. Wypełnienie formularza danymi 2. Kliknięcie przycisku zarejestruj
Warunki końcowe:	- Powinien zostać zawołany serwis autentykacji - Powinno zostać wysłane właściwe zapytanie do REST API. - Powinien wyświetlić się komunikat o niepomyślnej rejestarcji. - Nie powinno nastąpić przekierowanie na stronę logowania. - Pola formularza powinny być nieprawidłowe.
Typ testu:	negatywny

MAIN-NAVIGATION-COMPONENT

Nazwa:	MAIN-NAVIGATION-COMPONENT - Tworzenie komponentu
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Utworzenie komponentu.
Warunki końcowe:	Component został prawidłowo utworzony.
Typ testu:	pozytywny

Nazwa:	MAIN-NAVIGATION-COMPONENT – powinien prezentować nawigację zalogowanego użytkownika, kiedy użytkownik jest zalogowany
Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	Brak
Procedura:	1. Wywołanie ngOnInit()
Warunki końcowe:	- nawigacja powinna zawierać przyciski: „twoje listy” oraz „wyloguj” - nawigacja nie powinna zawierać przycisków: „zaloguj” oraz „zarejestruj”
Typ testu:	pozytywny

Nazwa:	MAIN-NAVIGATION-COMPONENT – powinien prezentować nawigację wylogowanego użytkownika, kiedy użytkownik jest wylogowany
Warunki początkowe:	- użytkownik jest niezalogowany
Dane testowe:	Brak
Procedura:	1. Wywołanie ngOnInit()
Warunki końcowe:	- nawigacja nie powinna zawierać przycisków: „twoje listy” oraz „wyloguj”

	- nawigacja powinna zawierać przyciski: „zaloguj” oraz „zarejestruj”
Typ testu:	pozytywny

Nazwa:	MAIN-NAVIGATION-COMPONENT – kliknięcie przyciska wyloguj powinno wywołać funkcję logout()
Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	Brak
Procedura:	1. Kliknięcie przycisku „wyloguj”
Warunki końcowe:	Funkcja logout() powinna zostać wywołana.
Typ testu:	pozytywny

Nazwa:	MAIN-NAVIGATION-COMPONENT – test funkcji logout()
Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	Brak
Procedura:	1. Wywołanie funkcji logout()
Warunki końcowe:	- sesja użytkownika powinna zostać wyczyszczona - użytkownik powinien zostać przekierowany na stronę logowania
Typ testu:	pozytywny

ADD-LIST-ITEM-COMPONENT

Nazwa:	ADD-LIST-ITEM-COMPONENT – tworzenie komponentu
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Stworzenie komponentu
Warunki końcowe:	Komponent tworzy się prawidłowo.
Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT - Pusty formularz powinien być fałszywy
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Pola formularza są puste
Warunki końcowe:	Formularz jest nieprawidłowy
Typ testu:	negatywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – Formularz z nazwą przedmiotu powinien być prawidłowy
Warunki początkowe:	Brak
Dane testowe:	- nazwa przedmiotu : „some item to buy name”
Procedura:	1. Pole nazwy przedmiotu zostaje wypełnione danymi
Warunki końcowe:	Formularz jest prawidłowy
Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – Puste pole autocomplete nazwy przedmiotu powinno zawierać wszystkie domyślne produkty.
Warunki początkowe:	- pole produktu jest puste - zdefiniowano domyślne produkty
Dane testowe:	Brak
Procedura:	1. Kliknięcie w pole nazwy produktu.
Warunki końcowe:	Liczba wyświetlonych propozycji jest równa ilości domyślnych produktów.
Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – Częściowo wypłenione pole autocomplete nazwy przedmiotu powinno odfiltrowane domyślne produkty.
Warunki początkowe:	- zdefiniowano domyślne produkty
Dane testowe:	- nazwa produktu: „mmm”

Procedura:	1. Kliknięcie w pole nazwy produktu. 2. Wpisanie danych.
Warunki końcowe:	Liczba wyświetlonych propozycji jest mniejsza od ilości domyślnych produktów.
Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – Wybranie nazwy przedmiotu powinno zawołać onSelectionChanged().
Warunki początkowe:	- zdefiniowano domyślne produkty
Dane testowe:	- nazwa produktu: pierwszy produkt z domyślnych produktów
Procedura:	1. Kliknięcie w pole nazwy produktu. 2. Wybranie pierwszego produktu.
Warunki końcowe:	Funkcja onSelectionChanged() zostaje zawołana
Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – zmiana zdjęcia powinna wywoływać selectFiles()
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Wysłanie eventu o zmianie zawartości obrazka.
Warunki końcowe:	Funkcja selectFiles() zostaje wywołana
Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – kliknięcie przycisku dodaj powinno wywołać addItem()
Warunki początkowe:	Brak
Dane testowe:	- nazwa przedmiotu: „test”
Procedura:	1. Wypełnienie formularza danymi. 2. Kliknięcie przycisku „dodaj”.
Warunki końcowe:	Funkcja addItem() zostaje wywołana.
Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – kliknięcie przycisku anuluj powinno wywołać clearForm()
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Wypełnienie formularza danymi. 2. Kliknięcie przycisku „anuluj”
Warunki końcowe:	Funkcja clearForm() została wywołana
Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – ngOnInit() powinno ustawić domyślne wartości formularza
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Wywołanie funkcji ngOnInit()
Warunki końcowe:	- liczba sztuk powinna wynosić 1 - jednostka nie powinna być pusta
Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – addItem() powinno wywołać itemService z odpowiednimi danymi.
Warunki początkowe:	Brak
Dane testowe:	- nazwa przedmiotu: „any product” - ilość: 2 - jednostka: „kg”
Procedura:	1. Wypełnienie formularza wartościami 2. Wywołanie funkcji addItem()
Warunki końcowe:	Serwis itemService powinien zostać wywołany z odpowiednio uformowanym obiektem ItemRequest.
Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – po udanym dodaniu przedmiotu powinien zostać wysłany event refresh.
Warunki początkowe:	- udane dodanie przedmiotu

Dane testowe:	Brak
Procedura:	1. Wywołanie addItem()
Warunki końcowe:	Powinien zostać wyemitowany event w celu odświeżenia aktualnej listy.
Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – po nieudanym dodaniu przedmiotu powinien zostać wyświetlony komunikat.
Warunki początkowe:	Brak
Dane testowe:	- odpowiedź z serwera ze statusem 401 - „Something went wrong”
Procedura:	1. Wywołanie addItem()
Warunki końcowe:	Powinien zostać wyświetlony popup o nieudanym dodaniu nowego przedmiotu.
Typ testu:	negatywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – wywłoanie onSelectionChanged() powinno przetłumaczyć wybraną opcję domyślnego produktu
Warunki początkowe:	- Zdefiniowane tłumaczenie dla tego produktu
Dane testowe:	- odpowiedź z serwera ze statusem 401 - „Something went wrong”
Procedura:	1. Wywołanie onSelectionChanged()
Warunki końcowe:	Nazwa wybranego produktu powinna zostać przetłumaczona
Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – wywłoanie selectFiles() powinno ustawić nazwę zdjęcia
Warunki początkowe:	Brak
Dane testowe:	- plik o nazwie „testName”
Procedura:	1. Wywołanie selectFiles
Warunki końcowe:	W formularzu powinna zostać ustawiona odpowiednia nazwa zdjęcia.

Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – przycisk delete powinien być ukryty kiedy nie ma żadnego zdjęcia
Warunki początkowe:	- brak wybranego zdjęcia
Dane testowe:	Brak
Procedura:	1. Pobranie właściwości przycisku „usuń zdjęcie”
Warunki końcowe:	Przycisk powinien mieć wartość hidden równą true.
Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – przycisk delete powinien widoczny kiedy jest wybrane zdjęcie
Warunki początkowe:	- wybrane zdjęcie
Dane testowe:	Brak
Procedura:	1. Pobranie właściwości przycisku „usuń zdjęcie”
Warunki końcowe:	Przycisk powinien mieć wartość hidden równą false.
Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – przycisk delete powinien usuwać dane zdjęcia
Warunki początkowe:	- wybrane zdjęcie
Dane testowe:	Brak
Procedura:	1. Kliknięcie przycisku usuń zdjęcie.
Warunki końcowe:	- interfejs dodawania zdjęcia zostaje wyczyszczony
Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – integracja - pomyślne dodanie nowego przedmiotu
Warunki początkowe:	- użytkownik jest zalogowany - id listy jest zdefiniowane

Dane testowe:	- nazwa produktu: „chleb” - ilość: 1 - jednostka: COUNT - zdjęcie: kod obrazka w Base64
Procedura:	1. Wypełnienie formularza danymi. 2. Kliknięcie przycisku dodaj.
Warunki końcowe:	- serwis tworzenia przedmiotu zostaje wywołany - zostaje wysłane odpowiednie żądanie do REST API - interfejs dodawania zostaje wyczyszczony - zostaje wyemitowany event do odświeżenia listy zakupów
Typ testu:	pozytywny

Nazwa:	ADD-LIST-ITEM-COMPONENT – integracja – nieudane dodanie nowego przedmiotu
Warunki początkowe:	- użytkownik jest zalogowany - id listy jest zdefiniowane
Dane testowe:	- nazwa produktu: „chleb” - ilość: 1 - jednostka: COUNT - zdjęcie: kod obrazka w Base64 - odpowiedź z serwera ze statusem 401 - „Something went wrong”
Procedura:	1. Wypełnienie formularza danymi. 2. Kliknięcie przycisku dodaj.
Warunki końcowe:	- serwis tworzenia przedmiotu zostaje wywołany - zostaje wysłane odpowiednie żądanie do REST API - interfejs dodawania nie zostaje wyczyszczony - nie zostaje wyemitowany event do odświeżenia listy zakupów
Typ testu:	negatywny

EDIT-LIST-ITEM-COMPONENT

Nazwa:	EDIT -LIST-ITEM-COMPONENT – tworzenie komponentu
Warunki początkowe:	Brak
Dane testowe:	przedmiot ma ustawione dane: - nazwa przedmiotu: „test item text”, - ilość: 2 - jednostka: kg - obrazek: zakodowany obrazek w Base64 - idListy: 2 - id: 1

Procedura:	1. Stworzenie komponentu
Warunki końcowe:	Komponent tworzy się prawidłowo.
Typ testu:	pozytywny

Nazwa:	EDIT -LIST-ITEM-COMPONENT – powinien załadować dane edytowanego przedmiotu
Warunki początkowe:	- poprawnie zainicjowana instancja obiektu.
Dane testowe:	Brak
Procedura:	1. Stworzenie komponentu
Warunki końcowe:	-zostaje wyświetlona aktualna nazwa, ilość, jednostka i zdjęcie przedmiotu
Typ testu:	pozytywny

Nazwa:	EDIT -LIST-ITEM-COMPONENT – pusta nazwa przedmiotu powinna być nieprawidłowa
Warunki początkowe:	- poprawnie zainicjowana instancja obiektu.
Dane testowe:	Brak
Procedura:	1. Skasowanie nazwy przedmiotu
Warunki końcowe:	- formularz powinien być nieprawidłowy
Typ testu:	negatywny

Nazwa:	EDIT -LIST-ITEM-COMPONENT – Puste pole autocomplete nazwy przedmiotu powinno zawierać wszystkie domyślne produkty.
Warunki początkowe:	- pole produktu jest puste - zdefiniowano domyślne produkty
Dane testowe:	Brak
Procedura:	2. Kliknięcie w pole nazwy produktu.
Warunki końcowe:	Liczba wyświetlonych propozycji jest równa ilości domyślnych produktów.
Typ testu:	pozytywny

Nazwa:	EDIT -LIST-ITEM-COMPONENT – Częściowo wypłenione pole autocomplete nazwy przedmiotu powinno odfiltrowane domyślne produkty.
Warunki początkowe:	- zdefiniowano domyślne produkty
Dane testowe:	- nazwa produktu: „mmm”
Procedura:	1. Kliknięcie w pole nazwy produktu. 2. Wpisanie danych.
Warunki końcowe:	Liczba wyświetlonych propozycji jest mniejsza od ilości domyślnych produktów.
Typ testu:	pozytywny

Nazwa:	EDIT -LIST-ITEM-COMPONENT – Wybranie nazwy przedmiotu powinno zawołać onSelectionChanged().
Warunki początkowe:	- zdefiniowano domyślne produkty
Dane testowe:	- nazwa produktu: pierwszy produkt z domyślnych produktów
Procedura:	1. Kliknięcie w pole nazwy produktu. 2. Wybranie pierwszego produktu.
Warunki końcowe:	Funkcja onSelectionChanged() zostaje zawołana
Typ testu:	pozytywny

Nazwa:	EDIT -LIST-ITEM-COMPONENT – zmiana zdjęcia powinna wywoływać selectFiles()
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Wysłanie eventu o zmianie zawartości obrazka.
Warunki końcowe:	Funkcja selectFiles() zostaje wywołana
Typ testu:	pozytywny

Nazwa:	EDIT-LIST-ITEM-COMPONENT – przycisk edit powinien wywołać editItem()
Warunki początkowe:	- formularz jest prawidłowy

Dane testowe:	Brak
Procedura:	1. Kliknięcie przycisku edytuj.
Warunki końcowe:	Funkcja editItem() zostaje wywołana
Typ testu:	pozytywny

Nazwa:	EDIT-LIST-ITEM-COMPONENT – przycisk anuluj powinien anulować edycję
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Kliknięcie przycisku anuluj.
Warunki końcowe:	Pole isBeingEdited przyjmuje wartość false.
Typ testu:	pozytywny

Nazwa:	EDIT-LIST-ITEM-COMPONENT – zedytowany obiekt powinien wołać serwis ze zmodyfikowanymi danymi
Warunki początkowe:	Brak
Dane testowe:	- nazwa przedmiotu: „any product”, - ilość: 2 - jednostka: kg - obrazek: mock pliku
Procedura:	1. Wypełnienie formularza nowymi danymi 2. Dodanie nowego zdjęcia 3. Kliknięcie przycisku edytuj.
Warunki końcowe:	Zostaje wywołany ItemService.update() z danymi takimi jakie zostały ustawione.
Typ testu:	pozytywny

Nazwa:	EDIT-LIST-ITEM-COMPONENT – po pomyślnej edycji, przedmiot powinien zostać odświeżony.
Warunki początkowe:	Brak.
Dane testowe:	- zmockowany pomyślny response serwisu.
Procedura:	1. Wywołanie funkcji editItem()

Warunki końcowe:	- funkcja refreshItem() zostaje wywołana - pole przedmiotu isBeingEdited przyjmuje wartość false
Typ testu:	pozytywny

Nazwa:	EDIT-LIST-ITEM-COMPONENT – po nieudanej edycji zostaje wyświetlony komunikat
Warunki początkowe:	Brak.
Dane testowe:	- odpowiedź z serwera ze statusem 401 - „Something went wrong”
Procedura:	1. Wywołanie funkcji editItem()
Warunki końcowe:	- pole przedmiotu isBeingEdited przyjmuje wartość true - wyświetlony zostaje alert
Typ testu:	negatywny

Nazwa:	EDIT-LIST-ITEM-COMPONENT – wywłoanie selectFiles() powinno ustawić nazwę zdjęcia
Warunki początkowe:	Brak
Dane testowe:	- plik o nazwie „testName”
Procedura:	1. Wywołanie selectFiles
Warunki końcowe:	W formularzu powinna zostać ustawiona odpowiednia nazwa zdjęcia.
Typ testu:	pozytywny

Nazwa:	EDIT-LIST-ITEM-COMPONENT – przycisk delete powinien być ukryty kiedy nie ma żadnego zdjęcia
Warunki początkowe:	- brak wybranego zdjęcia
Dane testowe:	Brak
Procedura:	1. Pobranie właściwości przycisku „usuń zdjęcie”
Warunki końcowe:	Przycisk powinien mieć wartość hidden równą true.
Typ testu:	pozytywny

Nazwa:	EDIT-LIST-ITEM-COMPONENT – przycisk delete powinien widoczny kiedy jest wybrane zdjęcie
Warunki początkowe:	- wybrane zdjęcie
Dane testowe:	Brak
Procedura:	1. Pobranie właściwości przycisku „usuń zdjęcie”
Warunki końcowe:	Przycisk powinien mieć wartość hidden równą false.
Typ testu:	pozytywny

Nazwa:	EDIT-LIST-ITEM-COMPONENT – przycisk delete powinien usuwać dane zdjęcia
Warunki początkowe:	- wybrane zdjęcie
Dane testowe:	Brak
Procedura:	1. Kliknięcie przycisku usuń zdjęcie.
Warunki końcowe:	- interfejs dodawania zdjęcia zostaje wyczyszczony
Typ testu:	pozytywny

Nazwa:	EDIT-LIST-ITEM-COMPONENT – wywołanie refreshItem powinno zaktualizować dane przedmiotu
Warunki początkowe:	Brak
Dane testowe:	- prawidłowo uformowany ItemResponse.
Procedura:	1. Wywołanie funkcji refreshItem
Warunki końcowe:	- wartości przedmiotu zostają zaktualizowane.
Typ testu:	pozytywny

Nazwa:	EDIT-LIST-ITEM-COMPONENT – integracja – pomyślna edycja przedmiotu
Warunki początkowe:	- użytkownik jest zalogowany - id listy jest zdefiniowane - id przedmiotu jest zdefiniowane

Dane testowe:	- nazwa produktu: „chleb” - ilość: 1 - jednostka: COUNT - zdjęcie: kod obrazka w Base64
Procedura:	1. Wypełnienie formularza danymi. 2. Kliknięcie przycisku edytuj.
Warunki końcowe:	- serwis edycji przedmiotu zostaje wywołany - zostaje wysłane odpowiednie żądanie do REST API - dane przedmiotu zostają zaktualizowane - przedmiot traci status bycia edytowanym
Typ testu:	pozytywny

Nazwa:	EDIT-LIST-ITEM-COMPONENT – integracja – nieudane dodanie nowego przedmiotu
Warunki początkowe:	- użytkownik jest zalogowany - id listy jest zdefiniowane - id przedmiotu jest zdefiniowane
Dane testowe:	- nazwa produktu: „chleb” - ilość: 1 - jednostka: COUNT - zdjęcie: kod obrazka w Base64 - odpowiedź z serwera ze statusem 401 - „Full authentication is required to access this resource.”
Procedura:	1. Wypełnienie formularza danymi. 2. Kliknięcie przycisku dodaj.
Warunki końcowe:	- serwis edycji przedmiotu zostaje wywołany - zostaje wysłane odpowiednie żądanie do REST API - dane przedmiotu nie zostają zaktualizowane - przedmiot nie traci status bycia edytowanym - wyświetlany jest komunikat o niepowodzeniu
Typ testu:	negatywny

LIST-ITEM-COMPONENT

Nazwa:	LIST-ITEM-COMPONENT – obrazek powinien zostać wyświetlony jeżeli istnieje
Warunki początkowe:	- przedmiot posiada obrazek
Dane testowe:	Brak
Procedura:	1. Pobranie elementu obrazka

Warunki końcowe:	- obrazek jest widoczny - obrazek ma prawidłowo ustawiony src (obrazek base64)
Typ testu:	pozytywny

Nazwa:	LIST-ITEM-COMPONENT – obrazek nie powinien zostać wyświetlony jeżeli nie istnieje-
Warunki początkowe:	- przedmiot nie posiada obrazek
Dane testowe:	Brak
Procedura:	1. Pobranie elementu obrazka
Warunki końcowe:	- obrazek jest niewidoczny
Typ testu:	negatywny

Nazwa:	LIST-ITEM-COMPONENT – przycisk delete powinien wołać funkcję deleteItem()
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Kliknięcie przycisku delete
Warunki końcowe:	- funkcja deleteItem() zostaje zawołana
Typ testu:	pozytywny

Nazwa:	LIST-ITEM-COMPONENT – przycisk edit powinien zmienić stan przedmiotu
Warunki początkowe:	- przedmiot nie jest w edycji
Dane testowe:	Brak
Procedura:	1. Kliknięcie przycisku edit
Warunki końcowe:	- parametr isBeingEdited zostaje ustawiony na true
Typ testu:	pozytywny

Nazwa:	LIST-ITEM-COMPONENT – powinien zostać wyświetlony komunikat, jeśli nie udało się usunąć przedmiotu

Warunki początkowe:	Brak
Dane testowe:	- odpowiedź z serwera ze statusem 401 - „Something went wrong”
Procedura:	1. Wywołanie funkcji deleteItem()
Warunki końcowe:	- wyświetlony zostaje komunikat o błędzie
Typ testu:	negatywny

Nazwa:	LIST-ITEM-COMPONENT – kliknięcie checkboxa powinno wywołać toggle()
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Kliknięcie checkboxa
Warunki końcowe:	- funkcja toggle() powinna zostać wywołana
Typ testu:	pozytywny

Nazwa:	LIST-ITEM-COMPONENT – toggle() powinno zmienić stan przedmiotu jeśli żądanie się powiodło
Warunki początkowe:	Brak
Dane testowe:	- pomyślna odpowiedź z serwisu.
Procedura:	1. Wywołanie funkcji toggle()
Warunki końcowe:	- parametr isChecked powinien zmienić stan na przeciwny
Typ testu:	pozytywny

Nazwa:	LIST-ITEM-COMPONENT – toggle() nie powinno zmienić stanu przedmiotu jeśli żądanie się nie powiodło
Warunki początkowe:	Brak
Dane testowe:	- odpowiedź z serwisu ze statusem 401 - „Something went wrong”
Procedura:	1. Wywołanie funkcji toggle()

Warunki końcowe:	- parametr isChecked powinien nie zostać zmieniony - powinien zostać wyświetlony komunikat o błędzie
Typ testu:	negatywny

Nazwa:	LIST-ITEM-COMPONENT – integracja – pomyślne usunięcie przedmiotu
Warunki początkowe:	- użytkownik jest zalogowany - id listy jest zdefiniowane - id przedmiotu jest zdefiniowane
Dane testowe:	- pomyślna odpowiedź z serwisu.
Procedura:	1. Kliknięcie przycisku delete.
Warunki końcowe:	- serwis usuwania przedmiotu zostaje wywołany - zostaje wysłane odpowiednie żądanie do REST API - zostaje wysłany event o usunięciu przedmiotu
Typ testu:	negatywny

Nazwa:	LIST-ITEM-COMPONENT – integracja – nieudane usunięcie przedmiotu
Warunki początkowe:	- użytkownik jest zalogowany - id listy jest zdefiniowane - id przedmiotu jest zdefiniowane
Dane testowe:	- odpowiedź z serwera ze statusem 401 - „Full authentication is required to access this resource.”
Procedura:	1. Kliknięcie przycisku delete.
Warunki końcowe:	- serwis usuwania przedmiotu zostaje wywołany - zostaje wysłane odpowiednie żądanie do REST API - nie zostaje wysłany event o usunięciu przedmiotu - zostaje wyświetlony komunikat o błędzie
Typ testu:	negatywny

SHOPPING-LIST-COMPONENT

Nazwa:	SHOPPING-LIST-COMPONENT – listId powinien zostać odczytany z queryParams
Warunki początkowe:	Brak
Dane testowe:	Zdefiniowany QueryParam listId:12
Procedura:	1. Wywołanie ngOnInit()

Warunki końcowe:	- obiekt powinien prawidłowo mieć ustawiony parametr listId
Typ testu:	pozytywny

Nazwa:	SHOPPING-LIST-COMPONENT – lista powinna zawierać list-item-component jeżeli liczba produktów jest większa od 0
Warunki początkowe:	- lista zawiera produkty
Dane testowe:	Brak
Procedura:	1. Pobranie app-list-item
Warunki końcowe:	- app-list-item powinien być zdefiniowany
Typ testu:	pozytywny

Nazwa:	SHOPPING-LIST-COMPONENT – lista powinna nie zawierać list-item-component jeżeli liczba produktów jest równa 0
Warunki początkowe:	- lista nie zawiera produktów
Dane testowe:	Brak
Procedura:	1. Pobranie app-list-item
Warunki końcowe:	- app-list-item nie powinien być zdefiniowany
Typ testu:	negatywny

Nazwa:	SHOPPING-LIST-COMPONENT – przedmioty powinny zostać odświeżone, jeżeli żądanie się powiodło
Warunki początkowe:	Brak
Dane testowe:	-Pomyślna odpowiedź z serwisu: ListResponse={ "listId": component.listId, "listName": "testName", "date" : "2022-11-26", "items": [{'id': 1, 'text': 'any product', 'quantity': 2, 'unit': ‘kg’, 'done': false, "image": null},

	{'id': 2, 'text': 'any product', 'quantity': 2, 'unit': “kg”, 'done': false, "image": null}]}
Procedura:	1. Wywołanie funkcji refreshItems.
Warunki końcowe:	- w komponencie znajdują się pobrane produkty
Typ testu:	pozytywny

Nazwa:	SHOPPING-LIST-COMPONENT – jeżeli pobranie przedmiotów się nie powiodło powinien zostać wyświetlony komunikat
Warunki początkowe:	Brak
Dane testowe:	- odpowiedź z serwisu ze statusem 401 - „Something went wrong”
Procedura:	1. Wywołanie funkcji refreshItems.
Warunki końcowe:	- zostaje wyświetlony komunikat o błędzie.
Typ testu:	negatywny

Nazwa:	SHOPPING-LIST-COMPONENT – removeItem() powinno usuwać wybrane przedmioty z listy.
Warunki początkowe:	- przedmiot istnieje w liście
Dane testowe:	Brak
Procedura:	1. Wywołanie funkcji removeItem()
Warunki końcowe:	- usunięty przedmiot nie znajduje się w liście przedmiotów.
Typ testu:	pozytywny

Nazwa:	SHOPPING-LIST-COMPONENT – integracja – pomyślne pobranie przedmiotów zapisanych w liście
Warunki początkowe:	- zdefiniowane listId - użytkownik jest zalogowany
Dane testowe:	- pomyślna odpowiedź z serwera zawierająca znajdujące się produkty w liście

Procedura:	1. Wywołanie ngOnInit()
Warunki końcowe:	- serwis pobierania przedmiotów zostaje wywołany - zostaje wysłane odpowiednie żądanie do REST API - przedmioty zostają załadowane do listy - przedmioty zostają odpowiednio wyświetlone
Typ testu:	pozytywny

Nazwa:	SHOPPING-LIST-COMPONENT – integracja – nieudane pobranie przedmiotów zapisanych w liście
Warunki początkowe:	- zdefiniowany queryparam listId - użytkownik jest zalogowany
Dane testowe:	- odpowiedź z serwera ze statusem 401 - „Full authentication is required to access this resource.”
Procedura:	1. Wywołanie ngOnInit()
Warunki końcowe:	- serwis pobierania przedmiotów zostaje wywołany - zostaje wysłane odpowiednie żądanie do REST API - wyświetlony zostaje komunikat o błędzie
Typ testu:	negatywny

ADD-USER-LIST-COMPONENT

Nazwa:	ADD-USER-LIST-COMPONENT – tworzenie komponentu
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	2. Stworzenie komponentu
Warunki końcowe:	Komponent tworzy się prawidłowo.
Typ testu:	pozytywny

Nazwa:	ADD-USER-LIST-COMPONENT - Pusty formularz powinien być fałszywy
Warunki początkowe:	Brak
Dane testowe:	Brak

Procedura:	2. Pola formularza są puste
Warunki końcowe:	Formularz jest nieprawidłowy
Typ testu:	negatywny

Nazwa:	ADD-USER-LIST-COMPONENT – Formularz bez daty powinien być fałszywy
Warunki początkowe:	Brak
Dane testowe:	- nazwa listy: „test shopping list name”
Procedura:	1. Wypełniona tylko nazwa listy
Warunki końcowe:	Formularz jest nieprawidłowy
Typ testu:	negatywny

Nazwa:	ADD-USER-LIST-COMPONENT – Formularz z nazwą przedmiotu i datą powinien być prawidłowy
Warunki początkowe:	Brak
Dane testowe:	- nazwa listy: „test shopping list name” - data: “2022-12-14”
Procedura:	1. Formularz zostaje wypełnione danymi
Warunki końcowe:	Formularz jest prawidłowy
Typ testu:	pozytywny

Nazwa:	ADD-USER-LIST-COMPONENT – Formularz z błędną data powinien być nieprawidłowy
Warunki początkowe:	Brak
Dane testowe:	- nazwa listy: „test shopping list name” - data: “20-12-2022”
Procedura:	1. Formularz zostaje wypełnione danymi
Warunki końcowe:	Formularz jest nieprawidłowy
Typ testu:	negatywny

Nazwa:	ADD-USER-LIST-COMPONENT – Przycisk dodaj listę powinien być widoczny, jeżeli formularz jest prawidłowy.

Warunki początkowe:	Brak
Dane testowe:	- nazwa listy: „test shopping list name” - data: “2022-12-14”
Procedura:	1. Formularz zostaje wypełnione danymi
Warunki końcowe:	Przycisk dodaj listę jest widoczny.
Typ testu:	pozytywny

Nazwa:	ADD-USER-LIST-COMPONENT – Przycisk dodaj listę powinien być niewidoczny, jeżeli formularz jest nieprawidłowy.
Warunki początkowe:	Brak
Dane testowe:	- nazwa listy: „test shopping list name” - data: “20-12-14”
Procedura:	1. Formularz zostaje wypełnione danymi
Warunki końcowe:	Przycisk dodaj listę jest niewidoczny.
Typ testu:	negatywny

Nazwa:	ADD-USER-LIST-COMPONENT – kliknięcie przycisku dodaj powinno wywołać addList()
Warunki początkowe:	Brak
Dane testowe:	- nazwa listy: „test shopping list name” - data: “2022-12-14”
Procedura:	1. Wypełnienie formularza danymi. 2. Kliknięcie przycisku „dodaj”.
Warunki końcowe:	Funkcja addList () zostaje wywołana.
Typ testu:	pozytywny

Nazwa:	ADD-USER-LIST-COMPONENT – kliknięcie przycisku anuluj powinno wywołać clearForm()
Warunki początkowe:	Brak
Dane testowe:	- nazwa listy: „test shopping list name” - data: “2022-12-14”
Procedura:	1. Wypełnienie formularza danymi. 2. Kliknięcie przycisku „anuluj”

Warunki końcowe:	Funkcja clearForm() została wywołana
Typ testu:	pozytywny

Nazwa:	ADD-USER-LIST-COMPONENT – addList() powinno wywołać listService z odpowiednimi danymi.
Warunki początkowe:	Brak
Dane testowe:	- nazwa listy: „test shopping list name” - data: “2022-12-14”
Procedura:	1. Wypełnienie formularza wartościami 2. Wywołanie funkcji addList ()
Warunki końcowe:	Serwis listService powinien zostać wywołany z odpowiednio uformowanym obiektem listRequest.
Typ testu:	pozytywny

Nazwa:	ADD-USER-LIST-COMPONENT – po udanym dodaniu listy powinien zostać wysłany event refresh.
Warunki początkowe:	Brak
Dane testowe:	- pomyślna odpowiedź z serwisu ze statusem 200
Procedura:	1. Wywołanie addList()
Warunki końcowe:	Powinien zostać wyemitowany event w celu odświeżenia aktualnej listy.
Typ testu:	pozytywny

Nazwa:	ADD-USER-LIST-COMPONENT – po nieudanym dodaniu listy powinien zostać wyświetlony komunikat.
Warunki początkowe:	Brak
Dane testowe:	- odpowiedź z serwera ze statusem 401 - „Something went wrong”
Procedura:	1. Wywołanie addList()
Warunki końcowe:	Powinien zostać wyświetlony komunikat o nieudanym dodaniu nowej listy.
Typ testu:	negatywny

Nazwa:	ADD-USER-LIST-COMPONENT – integracja - pomyślne dodanie nowej listy
Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	- nazwa listy: „test shopping list name” - data: “2022-12-14”
Procedura:	1. Wypełnienie formularza danymi. 2. Kliknięcie przycisku dodaj.
Warunki końcowe:	- serwis tworzenia list zostaje wywołany - zostaje wysłane odpowiednie żądanie do REST API - interfejs dodawania zostaje wyczyszczony - zostaje wyemitowany event do odświeżenia listy list.
Typ testu:	pozytywny

Nazwa:	ADD-USER-LIST-COMPONENT – integracja – nieudane dodanie nowej listy.
Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	- nazwa listy: „test shopping list name” - data: “2022-12-14” - odpowiedź z serwera ze statusem 401 - „Something went wrong”
Procedura:	1. Wypełnienie formularza danymi. 2. Kliknięcie przycisku dodaj.
Warunki końcowe:	- serwis tworzenia przedmiotu zostaje wywołany - zostaje wysłane odpowiednie żądanie do REST API - interfejs dodawania nie zostaje wyczyszczony - nie zostaje wyemitowany event do odświeżenia listy zakupów - zostaje wyświetlony komunikat o błędzie
Typ testu:	negatywny

EDIT-USER-LIST-COMPONENT

Nazwa:	EDIT-USER-LIST-COMPONENT – tworzenie komponentu
Warunki początkowe:	Brak
Dane testowe:	przedmiot ma ustawione dane: - nazwa listy: „test shopping list name” - data: “2022-12-14” - id: 1
Procedura:	1. Stworzenie komponentu
Warunki końcowe:	Komponent tworzy się prawidłowo.

Typ testu:	pozytywny

Nazwa:	EDIT-USER-LIST-COMPONENT – powinien załadować dane edytowanej listy
Warunki początkowe:	- poprawnie zainicjowana instancja obiektu.
Dane testowe:	Brak
Procedura:	1. Stworzenie komponentu
Warunki końcowe:	-zostaje wyświetlona aktualna nazwa i data listy
Typ testu:	pozytywny

Nazwa:	EDIT-USER-LIST-COMPONENT – pusta formularz powinien być nieprawidłowy
Warunki początkowe:	- poprawnie zainicjowana instancja obiektu.
Dane testowe:	Brak
Procedura:	1. Wyczyszczenie formularza
Warunki końcowe:	- formularz powinien być nieprawidłowy
Typ testu:	negatywny

Nazwa:	EDIT-USER-LIST-COMPONENT – formularz z pustą datą powinien być nieprawidłowy
Warunki początkowe:	- poprawnie zainicjowana instancja obiektu.
Dane testowe:	Brak
Procedura:	1. Wyczyszczenie daty
Warunki końcowe:	- formularz powinien być nieprawidłowy
Typ testu:	negatywny

Nazwa:	EDIT-USER-LIST-COMPONENT – Formularz z błędną data powinien być nieprawidłowy
Warunki początkowe:	Brak

Dane testowe:	- nazwa listy: „test shopping list name” - data: “20-12-2022”
Procedura:	1. Formularz zostaje wypełnione danymi
Warunki końcowe:	Formularz jest nieprawidłowy
Typ testu:	negatywny

Nazwa:	EDIT-USER-LIST-COMPONENT – Przycisk edytuj listę powinien być widoczny, jeżeli formularz jest prawidłowy.
Warunki początkowe:	Brak
Dane testowe:	- nazwa listy: „test shopping list name” - data: “2022-12-14”
Procedura:	1. Formularz zostaje wypełniony prawidłowymi danymi
Warunki końcowe:	Przycisk edytuj listę jest widoczny.
Typ testu:	pozytywny

Nazwa:	EDIT-USER-LIST-COMPONENT – Przycisk edytuj listę powinien być niewidoczny, jeżeli formularz jest nieprawidłowy.
Warunki początkowe:	Brak
Dane testowe:	- nazwa listy: „test shopping list name” - data: “20-12-14”
Procedura:	1. Formularz zostaje wypełniony nieprawidłowymi danymi
Warunki końcowe:	Przycisk edytuj listę jest niewidoczny.
Typ testu:	negatywny

Nazwa:	EDIT-USER-LIST-COMPONENT – kliknięcie przycisku edytuj powinno wywołać editList()
Warunki początkowe:	Brak
Dane testowe:	- nazwa listy: „test shopping list name” - data: “2022-12-14”
Procedura:	1. Wypełnienie formularza danymi. 2. Kliknięcie przycisku „edytuj”.
Warunki końcowe:	Funkcja editList () zostaje wywołana.

Typ testu:	pozytywny

Nazwa:	EDIT-USER-LIST-COMPONENT – kliknięcie przycisku anuluj powinno nie wprowadzać zmian
Warunki początkowe:	Brak
Dane testowe:	- nazwa listy: „test shopping list name update” - data: “2022-12-14”
Procedura:	1. Wypełnienie formularza danymi. 2. Kliknięcie przycisku „anuluj”
Warunki końcowe:	-zmiany nie zostają wprowadzone
Typ testu:	pozytywny

Nazwa:	EDIT-USER-LIST-COMPONENT – editList() powinno wywołać listService z odpowiednimi danymi.
Warunki początkowe:	Brak
Dane testowe:	- nazwa listy: „test shopping list name” - data: “2022-12-14”
Procedura:	1. Wypełnienie formularza wartościami 2. Wywołanie funkcji editList ()
Warunki końcowe:	Serwis listService powinien zostać wywołany z odpowiednio uformowanym obiektem listRequest.
Typ testu:	pozytywny

Nazwa:	EDIT-USER-LIST-COMPONENT – po pomyślnej edycji, lista powinna zostać odświeżona.
Warunki początkowe:	Brak.
Dane testowe:	- zmockowany pomyślny response serwisu.
Procedura:	1. Wywołanie funkcji editList()
Warunki końcowe:	- funkcja refreshList() zostaje wywołana - pole przedmiotu isBeingEdited przyjmuje wartość false
Typ testu:	pozytywny

Nazwa:	EDIT-USER-LIST-COMPONENT – po nieudanej edycji zostaje wyświetlony komunikat
Warunki początkowe:	Brak.
Dane testowe:	- odpowiedź z serwera ze statusem 401 - „Something went wrong”
Procedura:	1. Wywołanie funkcji editList()
Warunki końcowe:	- pole przedmiotu isBeingEdited przyjmuje wartość true - wyświetlony zostaje komunikat o błędzie
Typ testu:	negatywny

Nazwa:	EDIT-USER-LIST-COMPONENT – integracja – pomyślna edycja przedmiotu
Warunki początkowe:	- użytkownik jest zalogowany - id listy jest zdefiniowane
Dane testowe:	- nazwa listy: „test shopping list name” - data: “2022-12-14”
Procedura:	1. Wypełnienie formularza danymi. 2. Kliknięcie przycisku edytuj.
Warunki końcowe:	- serwis edycji listy zostaje wywołany - zostaje wysłane odpowiednie żądanie do REST API - dane listy zostają zaktualizowane - lista traci status bycia edytowanym
Typ testu:	pozytywny

Nazwa:	EDIT-USER-LIST-COMPONENT – integracja – nieudane dodanie nowego przedmiotu
Warunki początkowe:	- użytkownik jest zalogowany - id listy jest zdefiniowane
Dane testowe:	- nazwa listy: „test shopping list name” - data: “2022-12-14” - odpowiedź z serwera ze statusem 401 - „Full authentication is required to access this resource.”
Procedura:	1. Wypełnienie formularza danymi. 2. Kliknięcie przycisku dodaj.
Warunki końcowe:	- serwis edycji listy zostaje wywołany - zostaje wysłane odpowiednie żądanie do REST API - dane listy nie zostają zaktualizowane - lista nie traci status bycia edytowanym - wyświetlany jest komunikat o niepowodzeniu

Typ testu:	negatywny

USER-LIST-COMPONENT

Nazwa:	USER-LIST-COMPONENT – przycisk delete powinien wołać funkcję deleteList()
Warunki początkowe:	- lista istnieje
Dane testowe:	Brak
Procedura:	1. Kliknięcie przycisku delete
Warunki końcowe:	- funkcja deleteList() zostaje zawołana
Typ testu:	pozytywny

Nazwa:	USER-LIST-COMPONENT – przycisk edit powinien zmienić stan listy
Warunki początkowe:	- lista istnieje - lista nie jest w edycji
Dane testowe:	Brak
Procedura:	1. Kliknięcie przycisku edit
Warunki końcowe:	- parametr isBeingEdited zostaje ustawiony na true
Typ testu:	pozytywny

Nazwa:	USER-LIST-COMPONENT – powinien zostać wyświetlony komunikat, jeśli nie udało się usunąć listy
Warunki początkowe:	Brak
Dane testowe:	- odpowiedź z serwisu ze statusem 401 - „Something went wrong”
Procedura:	1. Wywołanie funkcji deleteList()
Warunki końcowe:	- wyświetlony zostaje komunikat o błędzie
Typ testu:	negatywny

Nazwa:	USER-LIST-COMPONENT – integracja – pomyślne usunięcie listy

Warunki początkowe:	- użytkownik jest zalogowany - id listy jest zdefiniowane
Dane testowe:	- pomyślna odpowiedź z serwisu (status 200)
Procedura:	1. Kliknięcie przycisku delete.
Warunki końcowe:	- serwis usuwania listy zostaje wywołany - zostaje wysłane odpowiednie żądanie do REST API - zostaje wysłany event o usunięciu listy
Typ testu:	negatywny

Nazwa:	USER-LIST-COMPONENT – integracja – nieudane usunięcie listy
Warunki początkowe:	- użytkownik jest zalogowany - id listy jest zdefiniowane
Dane testowe:	- odpowiedź z serwera ze statusem 401 - „Full authentication is required to access this resource.”
Procedura:	2. Kliknięcie przycisku delete.
Warunki końcowe:	- serwis usuwania listy zostaje wywołany - zostaje wysłane odpowiednie żądanie do REST API - nie zostaje wysłany event o usunięciu przedmiotu - zostaje wyświetlony komunikat o błędzie
Typ testu:	negatywny

USER-LISTS-COMPONENT

Nazwa:	USER-LISTS-COMPONENT- jeśli użytkownik nie jest zalogowany przekierowanie na strone do logowania
Warunki początkowe:	- użytkownik niezalogowany
Dane testowe:	Brak
Procedura:	1. Wywołanie ngOnInit()
Warunki końcowe:	- użytkownik powinien zostać przekierowany na stronę do logowania
Typ testu:	pozytywny

Nazwa:	USER-LISTS-COMPONENT – lista powinna zawierać user-list-component jeżeli liczba produktów jest większa od 0

Warunki początkowe:	- użytkownik posiada listy
Dane testowe:	Brak
Procedura:	1. Wyszukanie app-user-list
Warunki końcowe:	- app-user-list powinien być zdefiniowany
Typ testu:	pozytywny

Nazwa:	USER-LISTS-COMPONENT – lista powinna nie user-list-component jeżeli liczba produktów jest równa 0
Warunki początkowe:	- użytkownik nie posiada żadnej listy
Dane testowe:	Brak
Procedura:	1. Wyszukanie app-user-list
Warunki końcowe:	- app-user-list nie powinien być zdefiniowany
Typ testu:	negatywny

Nazwa:	USER-LISTS-COMPONENT – listy powinny zostać odświeżone, jeżeli żądanie się powiodło
Warunki początkowe:	Brak
Dane testowe:	-Pomyślna odpowiedź z serwisu: UserListsResponse={ "shoppingLists": [{"id": 1, "name": "test name", "date": "2022-11-26"}, {"id": 2, "name": "test name2", "date": "2022-11-27"}]}
Procedura:	1. Wywołanie funkcji refreshLists().
Warunki końcowe:	- w komponencie znajdują się pobrane listy
Typ testu:	pozytywny

Nazwa:	USER-LISTS-COMPONENT – jeżeli pobranie list się nie powiodło powinien zostać wyświetlony komunikat

Warunki początkowe:	Brak
Dane testowe:	- odpowiedź z serwisu ze statusem 401 - „Something went wrong”
Procedura:	2. Wywołanie funkcji refreshLists().
Warunki końcowe:	- zostaje wyświetlony komunikat o błędzie.
Typ testu:	negatywny

Nazwa:	USER-LISTS-COMPONENT – removeList() powinno usuwać wybraną listę.
Warunki początkowe:	- lista o danym id istnieje
Dane testowe:	Brak
Procedura:	2. Wywołanie funkcji removeList()
Warunki końcowe:	- usunięta lista nie znajduje się w listach użytkownika
Typ testu:	pozytywny

Nazwa:	USER-LISTS-COMPONENT – integracja – pomyślne pobranie list użytkownika
Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	- pomyślna odpowiedź z serwera zawierająca znajdujące się produkty w liście
Procedura:	1. Wywołanie ngOnInit()
Warunki końcowe:	- serwis pobierania list użytkownika zostaje wywołany - zostaje wysłane odpowiednie żądanie do REST API - listy zakupów zostają załadowane do listy - listy zostają odpowiednio wyświetlone
Typ testu:	pozytywny

Nazwa:	USER-LISTS-COMPONENT – integracja – nieudane pobranie list użytkownika

Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	- odpowiedź z serwera ze statusem 401 - „Full authentication is required to access this resource.”
Procedura:	1. Wywołanie ngOnInit()
Warunki końcowe:	- serwis pobierania list użytkownika zostaje wywołany - zostaje wysłane odpowiednie żądanie do REST API - wyświetlony zostaje komunikat o błędzie
Typ testu:	negatywny

AUTH-SERVICE

Nazwa:	AUTH-SERVICE – logowanie powinno być żądaniem POST i pomyślnie powinno zwracać dane o użytkowniku.
Warunki początkowe:	Brak
Dane testowe:	- nazwa użytkownika: „testuser” - haslo: „pass”
Procedura:	1. Przygotowanie LoginRequest z danymi. 2. Wywołanie metody login.
Warunki końcowe:	- zostaje wysłane odpowiednie żądanie do REST API - żądnie jest typu post - żądanie zwraca jsona z danymi użytkownika
Typ testu:	pozytywny

Nazwa:	AUTH-SERVICE – rejestracja powinna być żądaniem POST i pomyślnie powinno zwracać dane o użytkowniku.
Warunki początkowe:	Brak
Dane testowe:	- nazwa użytkownika: „testuser” - haslo: „pass”
Procedura:	1. Przygotowanie RegisterRequest z danymi. 2. Wywołanie metody register.
Warunki końcowe:	- zostaje wysłane odpowiednie żądanie do REST API - żądnie jest typu post - żądanie zwraca jsona z danymi zarejestrowanego użytkownika
Typ testu:	pozytywny

Nazwa:	AUTH-SERVICE – wylogowanie powinno być żądaniem DELETE

Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Wywołanie metody logout.
Warunki końcowe:	- zostaje wysłane odpowiednie żądanie do REST API - żądnie jest typu DELETE
Typ testu:	pozytywny

STORAGE-SERVICE

Nazwa:	STORAGE-SERVICE – saveUser() powinno zapisać użytkownika do sesji
Warunki początkowe:	Brak

Dane testowe:	user = {
	"id": 1,
	"username": "testuser",
	"roles": [
	"ROLE_USER"
	]
	}

Procedura:	1. Wywołanie saveUser() dla testowego użytkownika
Warunki końcowe:	- użytkownik został zapisany w sesji.
Typ testu:	pozytywny

Nazwa:	STORAGE-SERVICE – getUser() powinno zwrócić użytkownika zapisanego w sesji

Warunki początkowe:	- użytkownik zapisany w sesji
	user = {
	"id": 1,
	"username": "testuser",
	"roles": [
	"ROLE_USER"
	]
	}

Dane testowe:	Brak
Procedura:	1. Wywołanie getUser()

Warunki końcowe:	- pobrany użytkownik z sesji jest identyczny jak ten zapisany
Typ testu:	pozytywny

Nazwa:	STORAGE-SERVICE – clean() powinno usunąć użytkownika zapisanego w sesji

Warunki początkowe:	- użytkownik zapisany w sesji
	user = {
	"id": 1,
	"username": "testuser",
	"roles": [
	"ROLE_USER"
	]
	}

Dane testowe:	Brak
Procedura:	1. Wywołanie clean()
Warunki końcowe:	- nie powinien istniej zapisany użytkownik w sesji
Typ testu:	pozytywny

Nazwa:	STORAGE-SERVICE – isLoggedIn() powinno zwrócić true jeżeli użytkownik jest zalogowany

Warunki początkowe:	- użytkownik zapisany w sesji
	user = {
	"id": 1,
	"username": "testuser",
	"roles": [
	"ROLE_USER"
	]
	}

Dane testowe:	Brak
Procedura:	1. Wywołanie isLoggedIn()
Warunki końcowe:	- wywołanie funkcji powinno zwrócić true
Typ testu:	pozytywny

Nazwa:	STORAGE-SERVICE – isLoggedIn() powinno zwrócić false jeżeli użytkownik nie jest zalogowany

Warunki początkowe:	- brak użytkownika zapisanego w sesji
Dane testowe:	Brak
Procedura:	1. Wywołanie isLoggedIn()
Warunki końcowe:	- wywołanie funkcji powinno zwrócić false
Typ testu:	negatywny

ITEM-SERVICE

Nazwa:	ITEM-SERVICE – tworzenie przedmiotu powinno być multipart i pomyślnie zwracać utworzony przedmiot.
Warunki początkowe:	-użytkownik jest zalogowany

Dane testowe:	- ItemRequest = {
	"text": "Chleb",
	"listId": 1,
	"quantity": 1.0,
	"unit": "COUNT",
	"done": false
	}

Procedura:	1. Stworzenie poprawnego request'a. 2. Wywołanie funkcji create().
Warunki końcowe:	- zostaje wysłane odpowiednie żądanie do REST API - żądnie jest typu POST - funkcja zwraca utworzony przedmiot.
Typ testu:	pozytywny

Nazwa:	ITEM-SERVICE – aktualizacja przedmiotu powinna być żądaniem multipart i pomyślnie zwracać zaktualizowany przedmiot.
Warunki początkowe:	-użytkownik jest zalogowany

Dane testowe:	- ItemRequest = {
	"text": "Chleb",
	"listId": 1,
	"quantity": 1.0,
	"unit": "COUNT",
	"done": false
	}
	- id: 1

Procedura:	1. Stworzenie poprawnego request'a.

	2. Wywołanie funkcji update().
Warunki końcowe:	- zostaje wysłane odpowiednie żądanie do REST API - żądnie jest typu POST - funkcja zwraca zaktualizowany przedmiot.
Typ testu:	pozytywny

Nazwa:	ITEM-SERVICE – usunięcie przedmiotu powinno być żądaniem DELETE
Warunki początkowe:	-użytkownik jest zalogowany
Dane testowe:	- id: 1
Procedura:	1. Wywołanie funkcji delete().
Warunki końcowe:	- zostaje wysłane odpowiednie żądanie do REST API - żądnie jest typu DELETE
Typ testu:	pozytywny

Nazwa:	ITEM-SERVICE –pobranie przedmiotów z list być żądaniem GET i pomyślnie zwracać ListResponse
Warunki początkowe:	-użytkownik jest zalogowany

Dane testowe:	- expectedDataResponse = {
	"listId": 1,
	"listName": "testowa lista",
	"date": "2020-07-15",
	"items": [
	{
	"id": 24,
	"text": "Chleb",
	"quantity": 1.0,
	"unit": "COUNT",
	"image": null,
	"done": false
	},
	…
	]
	}

Procedura:	1. Wywołanie funkcji getListItems().
Warunki końcowe:	- zostaje wysłane odpowiednie żądanie do REST API - żądnie jest typu GET - funkcja zwraca obiekt ListResponse

Typ testu:	pozytywny

USER-LIST-SERVICE

Nazwa:	USER-LIST-SERVICE – tworzenie listy powinno być żądaniem POST i pomyślnie zwracać utworzoną listę.
Warunki początkowe:	-użytkownik jest zalogowany
Dane testowe:	-ListRequest = {"name": "lista", "date": "2022-08-02"}
Procedura:	1. Stworzenie poprawnego request'a. 2. Wywołanie funkcji create().
Warunki końcowe:	- zostaje wysłane odpowiednie żądanie do REST API - żądnie jest typu POST - funkcja zwraca utworzoną listę.
Typ testu:	pozytywny

Nazwa:	USER-LIST-SERVICE – aktualizacja listy powinna być żądaniem POST i pomyślnie zwracać zmodyfikowaną listę.
Warunki początkowe:	-użytkownik jest zalogowany
Dane testowe:	- ListRequest = {"name": "lista", "date": "2022-08-02"} - id: 1
Procedura:	1. Stworzenie poprawnego request'a. 2. Wywołanie funkcji update().
Warunki końcowe:	- zostaje wysłane odpowiednie żądanie do REST API - żądnie jest typu POST - funkcja zwraca zaktualizowaną listę.
Typ testu:	pozytywny

Nazwa:	USER-LIST-SERVICE – usunięcie listy powinno być żądaniem DELETE
Warunki początkowe:	-użytkownik jest zalogowany
Dane testowe:	- id: 1
Procedura:	1. Wywołanie funkcji delete().
Warunki końcowe:	- zostaje wysłane odpowiednie żądanie do REST API - żądnie jest typu DELETE
Typ testu:	pozytywny

Nazwa:	USER-LIST-SERVICE –pobranie list użytkownika powinno być żądaniem GET i pomyślnie zwracać UserListsResponse
Warunki początkowe:	-użytkownik jest zalogowany

Dane testowe:	- expectedDataResponse = {
	"shoppingLists": [
	{
	"id": 23,
	"name": "test 1",
	"date": "2020-07-15"
	},
	…
	]
	}

Procedura:	1. Wywołanie funkcji getUserLists().
Warunki końcowe:	- zostaje wysłane odpowiednie żądanie do REST API - żądnie jest typu GET - funkcja zwraca obiekt UserListsResponse.
Typ testu:	pozytywny

3.2 Testy backend:

Wszystkie poniższe testy zakładają wcześniejsze stworzenie testowanego elementu wraz z potrzebnymi zależnościami lub ich zastępnikami.

Wymagania dla środowiska

- Posiadanie zainstalowanego SDK Java 15
- Posiadanie lokalnej kopii kodu
- Posiadanie IDE IntelIJ
- Zbudowanie kodu wraz z zaciągnięciem odpowiednich dependencji – maven

Procedura testowania

- Uruchomienie testów znajdujących się w */test/java* (najprościej otwierając projekt za pomocą IntelIJ, a następnie klikając na folderze „run tests”
- W celu wygenerowania raportu z pokrycia testami, podczas uruchamiania wybrać opcję „run test with code coverage”

AUTH CONTROLLER

Nazwa:	AUTH CONTROLLER – test autentykacji
Warunki początkowe:	Brak
Dane testowe:	- username = "user" - password = "pass"

Procedura:	1. Utworzenie DTO do logowania 2. Wysłanie spreparowanego żądania autentykacji
Warunki końcowe:	- kontroler otrzymuje spreparowane żądanie. - autentykacja przebiega pomyślnie. - wszystkie funkcje odpowiedzialne za autentykacje zostają wywołane (authanticationManager.authenticate(), jwtUtils.generateJwtCookie()) - kontroler zwraca status 200
Typ testu:	pozytywny

Nazwa:	AUTH CONTROLLER – test rejestracji
Warunki początkowe:	Brak
Dane testowe:	- username = "user" - password = "pass"
Procedura:	1. Utworzenie DTO do rejestracji 2. Wysłanie spreparowanego żądania rejestracji
Warunki końcowe:	- kontroler otrzymuje spreparowane żądanie. - rejestracja przebiega pomyślnie. - zostaje sprawdzone istnienie w bazie użytkownika - użytkownikowi automatycznie zostaje przypisana rola użytkownika - kontroler zwraca status 200
Typ testu:	pozytywny

Nazwa:	AUTH CONTROLLER – test wylogowania
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Wysłanie spreparowanego żądania wylogowania
Warunki końcowe:	- kontroler otrzymuje spreparowane żądanie. - wylogowanie przebiega pomyślnie. - wywołane zostaje wygenerowanie czystego ciasteczka. - kontroler zwraca status 200
Typ testu:	pozytywny

ITEM CONTROLLER

Nazwa:	ITEM CONTROLLER – dodanie przedmiotu bez obrazka
Warunki początkowe:	- użytkownik jest zalogowany

Dane testowe:	- nazwa przedmiotu: „test item” - listId: 1
Procedura:	1. Spreparowanie żądania multipart bez dołączonego obrazka. 2. Wysłanie żądania
Warunki końcowe:	- kontroler otrzymuje żądanie - zostaje wywołany itemSerivce. addItem() z odpowiednimi argumentami. - kontroler zwraca status 200
Typ testu:	pozytywny

Nazwa:	ITEM CONTROLLER – dodanie przedmiotu z obrazkiem
Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	- nazwa przedmiotu: „test item” - obrazek: MockMultiPartFile() - listId: 1
Procedura:	1. Spreparowanie żądania multipart z dołączonym obrazkiem. 2. Wysłanie żądania
Warunki końcowe:	- kontroler otrzymuje żądanie - zostaje wywołany itemSerivce.addItem() z odpowiednimi argumentami. - kontroler zwraca status 200
Typ testu:	pozytywny

Nazwa:	ITEM CONTROLLER – dodanie przedmiotu z wyrzuceniem wyjątku
Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	- nazwa przedmiotu: „test item” - obrazek: MockMultiPartFile() - listId: 1
Procedura:	1. Spreparowanie żądania multipart z dołączonym obrazkiem. 2. Wysłanie żądania 3. Wyrzucenie wyjątku
Warunki końcowe:	- kontroler otrzymuje żądanie - zostaje wywołany itemSerivce.addItem() z odpowiednimi argumentami. - kontroler zwraca status z kodem błędu
Typ testu:	negatywny

Nazwa:	ITEM CONTROLLER – zaktualizowanie przedmiotu z obrazkiem

Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	- nazwa przedmiotu: „test item” - obrazek: MockMultiPartFile() - listId: 1
Procedura:	1. Spreparowanie żądania multipart z dołączonym obrazkiem. 2. Wysłanie żądania
Warunki końcowe:	- kontroler otrzymuje żądanie - zostaje wywołany itemSerivce.update() z odpowiednimi argumentami. - kontroler zwraca status 200
Typ testu:	pozytywny

Nazwa:	ITEM CONTROLLER – nieudane zaktualizowanie przedmiotu z obrazkiem
Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	- nazwa przedmiotu: „test item” - obrazek: MockMultiPartFile() - listId: 1
Procedura:	1. Spreparowanie żądania multipart z dołączonym obrazkiem. 2. Wysłanie żądania
Warunki końcowe:	- kontroler otrzymuje żądanie - zostaje wyrzucony wyjątek IOException - kontroler zwraca status z kodem błędu
Typ testu:	negatywny

Nazwa:	ITEM CONTROLLER – usunięcie przedmiotu
Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	- id: 1
Procedura:	1. Spreparowanie żądania DELETE 2. Wysłanie żądania
Warunki końcowe:	- kontroler otrzymuje żądanie - zostaje wywołany itemSerivce.deleteItem() z odpowiednimi argumentami. - kontroler zwraca status 200
Typ testu:	pozytywny

Nazwa:	ITEM CONTROLLER – pobranie przedmiotów z listy

Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	- id: 1
Procedura:	1. Spreparowanie żądania GET 2. Wysłanie żądania
Warunki końcowe:	- kontroler otrzymuje żądanie - zostaje wywołany itemSerivce.getAllItemsByListId() z odpowiednimi argumentami. - kontroler zwraca status 200
Typ testu:	pozytywny

LIST CONTROLLER

Nazwa:	LIST CONTROLLER – dodanie listy
Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	- nazwa listy: „testList” - data: "2022-11-16"
Procedura:	1. Spreparowanie żądania POST zawierającego dane nowej listy. 2. Wysłanie żądania
Warunki końcowe:	- kontroler otrzymuje żądanie - zostaje wywołany listService.addList() z odpowiednimi argumentami. - kontroler zwraca status 200 - kontroler zwraca stworzoną listę w postaci SimpleListResponse
Typ testu:	pozytywny

Nazwa:	LIST CONTROLLER – aktualizacja listy
Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	- nazwa listy: „testList” - data: "2022-11-16" - id: 1
Procedura:	1. Spreparowanie żądania POST zawierającego dane edytowanej listy. 2. Wysłanie żądania
Warunki końcowe:	- kontroler otrzymuje żądanie - zostaje wywołany listService.editList() z odpowiednimi argumentami. - kontroler zwraca status 200 - kontroler zwraca stworzoną listę w postaci SimpleListResponse
Typ testu:	pozytywny

Nazwa:	LIST CONTROLLER – usunięcie listy
Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	- id: 1
Procedura:	1. Spreparowanie żądania DELETE 2. Wysłanie żądania
Warunki końcowe:	- kontroler otrzymuje żądanie - zostaje wywołany listSerivce.deleteList() z odpowiednimi argumentami. - kontroler zwraca status 200
Typ testu:	pozytywny

Nazwa:	LIST CONTROLLER – pobranie list użytkownika
Warunki początkowe:	- użytkownik jest zalogowany
Dane testowe:	- id: 1
Procedura:	1. Spreparowanie żądania GET 2. Wysłanie żądania
Warunki końcowe:	- kontroler otrzymuje żądanie - zostaje wywołany listSerivce.getAllLists () - kontroler zwraca status 200 - kontroler zwraca odpowiedź w postaci obiektu ListResponse
Typ testu:	pozytywny

ITEM REPOSITORY

Nazwa:	ITEM REPOSITORY – test findItemsByList()
Warunki początkowe:	- w bazie danych znajduje się lista z przedmiotami, a także użytkownik będący jej właścicielem
Dane testowe:	- lista {"testList", "2022-11-17"}
Procedura:	1. Wywołanie funkcji itemRepository.findItemsByList() z listą która posiada przedmioty.
Warunki końcowe:	- wywołana funkcja zwraca listę Item'ów które znajdują się w tej liście
Typ testu:	pozytywny

LIST REPOSITORY

Nazwa:	LIST REPOSITORY – test FindListsByUser()
Warunki początkowe:	- w bazie danych znajduje się lista, a także użytkownik będący jej właścicielem
Dane testowe:	- lista {"testList", "2022-11-17"}
Procedura:	1. Wywołanie funkcji listRepository.findListsByUser() z użytkownikiem, który posiada listę.
Warunki końcowe:	- wywołana funkcja zwraca listę List posiadanych przez użytkownika
Typ testu:	pozytywny

ROLE REPOSITORY

Nazwa:	ROLE REPOSITORY – test FindByName()
Warunki początkowe:	- w bazie danych znajduje się rola
Dane testowe:	- RoleEnum.ROLE_USER
Procedura:	1. Wywołanie funkcji roleRepository.findByName() z nazwą szukanej roli
Warunki końcowe:	- wywołana funkcja powinna zwrócić znalezioną rolę
Typ testu:	pozytywny

USER REPOSITORY

Nazwa:	USER REPOSITORY – test findByUsername
Warunki początkowe:	- użytkownik o danej nazwie istnieje w bazie
Dane testowe:	- username: „user”
Procedura:	1. Wywołanie userRepository.findByUsername() z szukaną nazwą użytkownika.
Warunki końcowe:	- funkcja zwraca encję szukanego użytkownika.
Typ testu:	pozytywny

Nazwa:	USER REPOSITORY – test existsByUsername
Warunki początkowe:	- użytkownik o danej nazwie istnieje w bazie

Dane testowe:	- username: „user”
Procedura:	1. Wywołanie userRepository.existsByUsername() z szukaną nazwą użytkownika.
Warunki końcowe:	- funkcja zwraca wartość true dla istniejącego użytkownika
Typ testu:	pozytywny

USER DETAILS SERVICE IMPL

Nazwa:	USER DETAILS SERVICE IMPL – test loadUserByUserName
Warunki początkowe:	- użytkownik „user” istnieje w bazie danych
Dane testowe:	- nazwa użytkownika „user”
Procedura:	1. Wywłoanie userDetailsService.loadUserByUsername()
Warunki końcowe:	- funkcja zwraca znalezionego użytkownika w postaci UserDetails.
Typ testu:	pozytywny

Nazwa:	USER DETAILS SERVICE IMPL – test LoadUserByUserNameException
Warunki początkowe:	- użytkownik „user” nie istnieje w bazie danych
Dane testowe:	- nazwa użytkownika „user”
Procedura:	1. Wywłoanie userDetailsService.loadUserByUsername()
Warunki końcowe:	- funkcja wyrzuca wyjątek UsernameNotFoundException.
Typ testu:	negatywny

ITEM SERVICE

Nazwa:	ITEM SERVICE – test AddItem
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa
Dane testowe:	- nazwa przedmiotu: „testText” - ilość: 1 - jednostka: sztuk - obrazek: brak - done: false

Procedura:	1. Wywołanie itemService.addItem() z odpowiednimi argumentami.
Warunki końcowe:	- wywołana została metoda z itemRepository odpowiedzialna za zapis przedmiotu do bazy z obiektem typu Item.class - wywołana metoda zwróciła obiekt typu SingleItemResponse
Typ testu:	pozytywny

Nazwa:	ITEM SERVICE – test AddItem z obrazkiem
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa
Dane testowe:	- nazwa przedmiotu: „testText” - ilość: 1 - jednostka: sztuk - obrazek: MockMultipartFile("image", new byte[1]); - done: false
Procedura:	1. Wywołanie itemService.addItem() z odpowiednimi argumentami.
Warunki końcowe:	- wywołana została metoda z itemRepository odpowiedzialna za zapis przedmiotu do bazy z obiektem typu Item.class - wywołana metoda zwróciła obiekt typu SingleItemResponse
Typ testu:	pozytywny

Nazwa:	ITEM SERVICE – test AddItemToNonExitsList
Warunki początkowe:	- zdefiniowany użytkownik testowy - brak zdefiniowanej listy o wybranym id
Dane testowe:	- nazwa przedmiotu: „testText” - ilość: 1 - jednostka: sztuk - obrazek: brak - done: false - listId: 1
Procedura:	1. Wywołanie itemService.addItem() z odpowiednimi argumentami.
Warunki końcowe:	- wywołana metoda wyrzuciła wyjątek NotFoundException
Typ testu:	negatywny

Nazwa:	ITEM SERVICE – test updateNonExistingItem
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - brak przedmiotu o wybranym id

Dane testowe:	- id: 1
Procedura:	1. Wywołanie itemService.updateItem() z odpowiednimi argumentami.
Warunki końcowe:	- wywołana metoda wyrzuciła wyjątek NotFoundException
Typ testu:	negatywny

Nazwa:	ITEM SERVICE – test updateItem
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - istnieje przedmiotu o wybranym id
Dane testowe:	- nazwa przedmiotu: „otherText” - ilość: 2.5 - jednostka: kg - obrazek: brak - done: false - id: 1
Procedura:	1. Wywołanie itemService.updateItem() z odpowiednimi argumentami.
Warunki końcowe:	- wywołana została metoda z itemRepository odpowiedzialna za zapis przedmiotu do bazy z obiektem typu Item.class - wywołana metoda zwróciła obiekt typu SingleItemResponse - przedmiot, który miał być zaktualizowany, został zaktualizowany
Typ testu:	pozytywny

Nazwa:	ITEM SERVICE – test updateItemWithImage
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - istnieje przedmiotu o wybranym id
Dane testowe:	- nazwa przedmiotu: „otherText” - ilość: 2.5 - jednostka: kg - obrazek: MockMultipartFile("image", new byte[1]); - done: false - id: 1
Procedura:	1. Wywołanie itemService.updateItem() z odpowiednimi argumentami.
Warunki końcowe:	- wywołana została metoda z itemRepository odpowiedzialna za zapis przedmiotu do bazy z obiektem typu Item.class - wywołana metoda zwróciła obiekt typu SingleItemResponse - przedmiot, który miał być zaktualizowany, został zaktualizowany - przedmiot posiada zapisany obrazek

Typ testu:	pozytywny

Nazwa:	ITEM SERVICE – test UpdateItemWhichNotBelongToUser
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - zdefiniowany przedmiot przypisany do użytkownika testowego
Dane testowe:	- inny użytkownik: „user2”
Procedura:	1. Podmiana zalogowanego użytkownika na niebędącego właścicielem przedmiotu 2. Wywołanie itemService.updateItem() z odpowiednimi argumentami.
Warunki końcowe:	- wywołana metoda wyrzuca wyjątek ForbiddenException
Typ testu:	negatywny

Nazwa:	ITEM SERVICE – test DeleteItem
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - zdefiniowany przedmiot testowy
Dane testowe:	- id: istniejącego przedmiotu
Procedura:	1. Wywołanie itemService.delete() z odpowiednimi argumentami.
Warunki końcowe:	- wywołana została metoda itemRepository.delete() z obiektem typu Item.class
Typ testu:	pozytywny

Nazwa:	ITEM SERVICE – test DeleteItemWhichNotBelongToUser
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - zdefiniowany przedmiot przypisany do użytkownika testowego
Dane testowe:	- id: istniejącego przedmiotu
Procedura:	1. Podmiana zalogowanego użytkownika na niebędącego właścicielem przedmiotu 2. Wywołanie itemService.delete() z odpowiednimi argumentami.
Warunki końcowe:	- wywołana metoda wyrzuca wyjątek ForbiddenException
Typ testu:	negatywny

Nazwa:	ITEM SERVICE – test GetAllItemsByListId
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - zdefiniowany przedmiot
Dane testowe:	- istniejąca lista
Procedura:	1. Wywołanie itemService.getAllItemsByListId () z odpowiednimi argumentami.
Warunki końcowe:	- wywołana metoda zwraca ItemsResponse, który zawiera przedmioty znajdujące się w liście.
Typ testu:	pozytywny

LIST SERVICE

Nazwa:	LIST SERVICE – test AddList
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa
Dane testowe:	- użycie zdefiniowanej listy
Procedura:	1. Wywołanie metody listService.addList() z odpowiednimi argumentami
Warunki końcowe:	- zostaje wywołana metoda listRepository.save() z obiektem typu ShoppingList.class - wywołana metoda zwraca obiekt typu SimpleListResponse
Typ testu:	pozytywny

Nazwa:	LIST SERVICE – test UpdateList
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa
Dane testowe:	- aktualizacja zdefiniowanej listy testowej - nazwa: „updateList” - data: „2022-11-18”
Procedura:	1. Wywołanie metody listService.updateList() z odpowiednimi argumentami
Warunki końcowe:	- zostaje wywołana metoda listRepository.save() z obiektem typu ShoppingList.class - wywołana metoda zwraca obiekt typu SimpleListResponse - w aktualizowanej liście zostały zaktualizowane dane
Typ testu:	pozytywny

Nazwa:	LIST SERVICE – test UpdateListNotBelongToUser

Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa
Dane testowe:	- aktualizacja zdefiniowanej listy testowej - użytkownik: „user 2” - nazwa: „updateList” - data: „2022-11-18”
Procedura:	1. Podmiana zalogowanego użytkownika na niebędącego właścicielem listy 2. Wywołanie metody listService.updateList() z odpowiednimi argumentami
Warunki końcowe:	- wywołana metoda wyrzuca wyjątek ForbiddenException
Typ testu:	negatywny

Nazwa:	LIST SERVICE – test UpdateListWhichNotExists
Warunki początkowe:	- zdefiniowany użytkownik testowy - lista o aktualizowanym id nie istnieje
Dane testowe:	- nazwa: „updateList” - data: „2022-11-18” - id: 1
Procedura:	1. Wywołanie metody listService.updateList() z odpowiednimi argumentami
Warunki końcowe:	- wywołana metoda wyrzuca wyjątek NotFoundException
Typ testu:	negatywny

Nazwa:	LIST SERVICE – test DeleteList
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa
Dane testowe:	- id: istniejącej list testowej
Procedura:	1. Wywołanie listService.delete() z odpowiednimi argumentami.
Warunki końcowe:	- wywołana została metoda listRepository.delete() z obiektem typu ShoppingfList.class
Typ testu:	pozytywny

Nazwa:	LIST SERVICE – test DeleteListWhichNotBelongToUser

Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa
Dane testowe:	- id: istniejącej listy - użytkownik: „user2”
Procedura:	1. Podmiana zalogowanego użytkownika na niebędącego właścicielem listy 2. Wywołanie listService.delete() z odpowiednimi argumentami.
Warunki końcowe:	- wywołana metoda wyrzuca wyjątek ForbiddenException
Typ testu:	negatywny

Nazwa:	LIST SERVICE – test GetAllLists
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa
Dane testowe:	- istniejąca lista
Procedura:	2. Wywołanie listService.getAllLists()
Warunki końcowe:	- wywołana metoda zwraca ListsResponse, który zawiera listy przypisane do zalogowanego użytkownika
Typ testu:	pozytywny

AUTH INTEGRATION

Nazwa:	AUTH INTEGRATION - testIntegrationOfAuthenticateUser
Warunki początkowe:	- w bazie danych istnieje rola USER - w bazie istnieje użytkownik „user” „pass”
Dane testowe:	-wykorzystanie zdefiniowanego użytkownika
Procedura:	1. Wysłanie spreparowanego żądania na endpoint api/auth/signin
Warunki końcowe:	- odpowiedź ma status 200 - odpowiedź zawiera dane zalogowanego użytkownika - odpowiedź zawiera ciasteczko z tokenem JWT
Typ testu:	pozytywny

Nazwa:	AUTH INTEGRATION - testIntegrationOfAuthenticateUserWhichNotExists
Warunki początkowe:	- w bazie danych istnieje rola USER - w bazie nie istnieje użytkownik „usernotexists” „pass”
Dane testowe:	- username = "usernotexists" - password = "pass";

Procedura:	1. Wysłanie spreparowanego żądania na endpoint api/auth/signin
Warunki końcowe:	- odpowiedź ma status 4xx - odpowiedź zawiera odpowiedź o nieudanym logowaniu - odpowiedź nie zawiera ciasteczka z tokenem JWT
Typ testu:	negatywny

Nazwa:	AUTH INTEGRATION - testIntegrationOfRegister
Warunki początkowe:	- w bazie nie istnieje użytkownik o nazwie „user2” - w bazie danych istnieje rola USER
Dane testowe:	-username = "user2"; -password = "pass";
Procedura:	1. Wysłanie spreparowanego żądania na endpoint api/auth/signup
Warunki końcowe:	- użytkownik zostaje zapisany w bazie - odpowiedź zawiera informacje o pomyślnym utworzeniu użytkownika - odpowiedź ma status 200 - zarejestrowany użytkownik ma identyczne dane jak te przekazane w żądaniu
Typ testu:	pozytywny

Nazwa:	AUTH INTEGRATION - testIntegrationOfRegisterOfUserAlreadyExists
Warunki początkowe:	- w bazie istnieje użytkownik o nazwie „user2” - w bazie danych istnieje rola USER
Dane testowe:	-username = "user"; -password = "pass";
Procedura:	1. Wysłanie spreparowanego żądania na endpoint api/auth/signup
Warunki końcowe:	- użytkownik nie zostaje zapisany w bazie - odpowiedź zawiera informacje o nieudanym utworzeniu użytkownika - odpowiedź ma status 4xx
Typ testu:	negatywny

Nazwa:	AUTH INTEGRATION - testIntegrationOfLogout
Warunki początkowe:	Brak
Dane testowe:	Brak
Procedura:	1. Wysłanie spreparowanego żądania na endpoint api/auth/signout

Warunki końcowe:	- odpowiedź ma stutus 200 - odpowiedź zawiera puste ciasteczko tokenu JWT
Typ testu:	pozytywny

ITEM INTEGRATION

Nazwa:	ITEM ITEGRATION - testIntegrationAddList
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - użytkownik jest zalogowany
Dane testowe:	- nazwa przedmiotu: „test text” - ilość: 1 - jednostka: sztuk - zdjęcie: MockMultiPartFile()
Procedura:	1. Wysłanie spreparowanego żądania na endpoint api/item/add
Warunki końcowe:	- odpowiedź ma status 200 - odpowiedź zawiera dane utworzonego przedmiotu - w bazie istnieje dodany przedmiot
Typ testu:	pozytywny

Nazwa:	ITEM ITEGRATION - testIntegrationUpdateList
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - zdefiniowany przedmiot testowy - użytkownik jest zalogowany
Dane testowe:	- wykorzystanie przedmiotu stworzonego w teście dodawania - nazwa przedmiotu: „test text updated” - ilość: 1 - jednostka: sztuk - zdjęcie: MockMultiPartFile()
Procedura:	1. Wysłanie spreparowanego żądania na endpoint api/item/update
Warunki końcowe:	- odpowiedź ma status 200 - odpowiedź zawiera dane zaktualizowanego przedmiotu - w bazie istnieje zaktualizowany przedmiot
Typ testu:	pozytywny

Nazwa:	ITEM ITEGRATION - testIntegrationGetAllItems
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - zdefiniowany przedmiot testowy - użytkownik jest zalogowany

Dane testowe:	- wykorzystanie przedmiotu stworzonego w teście dodawania - wykorzystanie zdefiniowanej listy
Procedura:	1. Wysłanie spreparowanego żądania na endpoint api/item/all/{listid}
Warunki końcowe:	- odpowiedź ma status 200 - odpowiedź zawiera dane przedmiotów znajdujących się w liście - odpowiedź zawiera parametry listy
Typ testu:	pozytywny

Nazwa:	ITEM ITEGRATION - testIntegrationDeleteItem
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - zdefiniowany przedmiot testowy - użytkownik jest zalogowany - wykonanie testu dopiero po wykonaniu testów korzystających z tego przedmiotu
Dane testowe:	- wykorzystanie przedmiotu stworzonego w teście dodawania - wykorzystanie zdefiniowanej listy
Procedura:	1. Wysłanie spreparowanego żądania na endpoint api/item/delete/{itemid}
Warunki końcowe:	- odpowiedź ma status 200 - usunięty przedmiot nie istnieje w bazie
Typ testu:	pozytywny

Nazwa:	ITEM ITEGRATION - testIntegrationDeleteItemWhichNotBelongToLoggedUser
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - zdefiniowany przedmiot testowy przypisany do innego użytkownika - użytkownik jest zalogowany
Dane testowe:	- wykorzystanie przedmiotu zdefiniowanego w warunkach początkowych
Procedura:	1. Wysłanie spreparowanego żądania na endpoint api/item/delete/{itemid}
Warunki końcowe:	- odpowiedź ma status 4xx - przedmiot nie zostaje usunięty
Typ testu:	negatywny

Nazwa:	ITEM ITEGRATION - testIntegrationUpdateItemWhichNotBelongToLoggedUser

Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - zdefiniowany przedmiot testowy przypisany do innego użytkownika - użytkownik jest zalogowany
Dane testowe:	- wykorzystanie id przedmiotu zdefiniowanego w warunkach początkowych - nazwa przedmiotu: „test text updated” - ilość: 1 - jednostka: sztuk - zdjęcie: MockMultiPartFile()
Procedura:	1. Wysłanie spreparowanego żądania na endpoint api/item/update/{itemid}
Warunki końcowe:	- odpowiedź ma status 4xx - przedmiot nie zostaje zaktualizowany
Typ testu:	negatywny

Nazwa:	ITEM ITEGRATION - testIntegrationUpdateItemThatNotExists
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - użytkownik jest zalogowany - nie istnieje przedmiot o wybranym id
Dane testowe:	- id: 999999 - nazwa przedmiotu: „test text updated” - ilość: 1 - jednostka: sztuk - zdjęcie: MockMultiPartFile()
Procedura:	1. Wysłanie spreparowanego żądania na endpoint api/item/update/{itemid}
Warunki końcowe:	- odpowiedź ma status 4xx - w bazie nie istnieje przedmiot o takim id
Typ testu:	negatywny

Nazwa:	ITEM ITEGRATION - testIntegrationGetItemsFromListWhichNotBelongToLoggedUser
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa przypisana do innego użytkownika - użytkownik jest zalogowany
Dane testowe:	- wykorzystanie id listy zdefiniowanego w warunkach początkowych
Procedura:	1. Wysłanie spreparowanego żądania na endpoint api/item/all/{listid}
Warunki końcowe:	- odpowiedź ma status 4xx

Typ testu:	negatywny

LIST INTEGRATION

Nazwa:	LIST INTEGRATION - testIntegrationAddList
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - użytkownik jest zalogowany
Dane testowe:	- nazwa: „test list” - data: „2022-12-03”
Procedura:	1. Wysłanie spreparowanego żądania na endpoint /api/list/add
Warunki końcowe:	- lista zostaje utworzona w bazie - odpowiedź ma status 200 - odpowiedź zawiera dane utworzonej listy
Typ testu:	pozytywny

Nazwa:	LIST INTEGRATION - testIntegrationUpdate
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - użytkownik jest zalogowany - wcześniejsze wykonanie testu testIntegrationAddList
Dane testowe:	- wykorzystanie id listy z poprzedniego testu - nazwa: „test list updated” - data: „2022-12-03”
Procedura:	1. Wysłanie spreparowanego żądania na endpoint /api/list/update
Warunki końcowe:	- lista zostaje zaktualizowana w bazie - odpowiedź ma status 200 - odpowiedź zawiera dane zaktualizowanej listy
Typ testu:	pozytywny

Nazwa:	LIST INTEGRATION - testIntegrationGetUserLists
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - użytkownik jest zalogowany - wcześniejsze wykonanie testu testIntegrationAddList
Dane testowe:	Brak
Procedura:	1. Wysłanie spreparowanego żądania na endpoint /api/list/all
Warunki końcowe:	- odpowiedź ma status 200 - odpowiedź zawiera dane list zalogowanego użytkownika
Typ testu:	pozytywny

Nazwa:	LIST INTEGRATION - testIntegrationDeleteListWithItems
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa - użytkownik jest zalogowany - wcześniejsze wykonanie testów korzystających z tej listy - lista zawiera przedmioty
Dane testowe:	- wykorzystanie id listy z poprzednich testów
Procedura:	1. Wysłanie spreparowanego żądania na endpoint /api/list/delete/{listid}
Warunki końcowe:	- lista zostaje usunięta z bazy - przedmioty z listą zostają także usunięte z bazy - odpowiedź ma status 200
Typ testu:	pozytywny

Nazwa:	LIST ITEGRATION - testIntegrationUpdateListWhichNotBelongToLoggedUser
Warunki początkowe:	- zdefiniowany użytkownik testowy - zdefiniowana lista testowa przypisana jest do innego użytkownika
Dane testowe:	- wykorzystanie id listy zdefiniowanej w warunkach początkowych - nazwa: „test list updated” - data: „2022-12-03”
Procedura:	1. Wysłanie spreparowanego żądania na endpoint api/list/update/{listid}
Warunki końcowe:	- odpowiedź ma status 4xx - lista nie zostaje zaktualizowana
Typ testu:	negatywny

Nazwa:	LIST ITEGRATION - testIntegrationUpdateListThatNotExists
Warunki początkowe:	- zdefiniowany użytkownik testowy - użytkownik jest zalogowany - nie istnieje lista o wybranym id
Dane testowe:	- id: 999999 - nazwa: „test list updated” - data: „2022-12-03”
Procedura:	1. Wysłanie spreparowanego żądania na endpoint api/item/update/{itemid}
Warunki końcowe:	- odpowiedź ma status 4xx - w bazie nie istnieje przedmiot o takim id
Typ testu:	negatywny

4.Testy systemowe end-to-end

Testy systemowe zostały opracowane w środowisku Cypress. Przygotowano 5 scenariuszy testów automatycznych.

Wymagania dla środowiska:

- Posiadanie i uruchomienia instancji aplikacji backendu
- Posiadanie i uruchomienia instancji aplikacji frontendu

Uruchomienie:

Testy można uruchomić komendą w terminalu części frontend:

cypress open

następnie:

1. Wybrać *E2E Testing.*
2. Wybrać przeglądarkę, na której chcemy przeprowadzić testy.
3. Wybrać plik z scenariuszami testów (po wybraniu pliku automatycznie zostaną uruchomione wybrane testy).

Przypadki Testowe:

Część testów została umieszczona w jednym pliku, ponieważ poszczególne scenariusze testowe łączą się ze sobą w jeden większy scenariusz. Uruchamianie każdego scenariusza z osobna byłoby niewygodne i wymagałoby pośrednio wykonania poprzedniego scenariusza.

1. **Register** (plik: register.cy.ts) – rejestracja nowego użytkownika w systemie.
2. **Login** (plik: loginAndAppUsage.cy.ts) – logowanie użytkownika.
3. **User lists managing** (plik: loginAndAppUsage.cy.ts) – operacje na listach.
4. **Items managing** (plik: loginAndAppUsage.cy.ts) – operacje na przedmiotach.
5. **Logout** (plik: loginAndAppUsage.cy.ts) – wylogowanie.

Opis przypadków testowych:

Nazwa:	**1.Register** (plik: register.cy.ts) – rejestracja nowego użytkownika w systemie.
Warunki początkowe:	- w bazie nie istnieje użytkownik nazwie: „testuseraccount” - żaden użytkownik nie jest zalogowany
Dane testowe:	- username: „testuseraccount” - password: „pass”
Typ testu:	pozytywny

L.P	Opis kroku	Oczekiwany rezultat	Rzeczywisty rezultat
1.	Wejdź na stronę http://localhost:4200.	Przekierowanie na stronę http://localhost:4200/login.	Przekierowano na stronę http://localhost:4200/login.
2.	Kliknij na przycisk „Rejestracja”.	Przekierowanie na stronę http://localhost:4200/register.	Przekierowano na stronę http://localhost:4200/register.
3.	Prawidłowe wypełnienie formularza.	- Formularz jest wypełniony prawidłowo - Przycisk „zarejestruj” jest odblokowany.	- Formularz został wypełniony prawidłowo - Przycisk „zarejestruj” został odblokowany.
4.	Po kliknięciu przycisku „zarejestruj” powinien pojawić się komunikat.	Pokazuje się komunikat o udanej rejestracji.	Pokazał się komunikat o udanej rejestracji.
5.	Zatwierdzenie komunikatu.	Przekierowanie na stronę http://localhost:4200/login.	Przekierowano na stronę http://localhost:4200/login.

Nazwa:	**2.Login** (plik: loginAndAppUsage.cy.ts) – logowanie użytkownika.
Warunki początkowe:	- w bazie istnieje użytkownik nazwie: „testuseraccount”
Dane testowe:	- username: „testuseraccount” - password: „pass”
Typ testu:	pozytywny

L.P	Opis kroku	Oczekiwany rezultat	Rzeczywisty rezultat
1.	Wejdź na stronę http://localhost:4200.	Przekierowanie na stronę http://localhost:4200/login.	Przekierowano na stronę http://localhost:4200/login.
2.	Prawidłowe wypełnienie formularza.	- Formularz jest wypełniony prawidłowo - Przycisk „zaloguj” jest odblokowany.	- Formularz został wypełniony prawidłowo - Przycisk „zaloguj” został odblokowany.
3.	Po kliknięciu przycisku „zaloguj” powinno nastąpić przekierowanie.	Przekierowanie na stronę http://localhost:4200/user-lists.	Przekierowano na stronę http://localhost:4200/user-lists.
4.	W sesji powinno zostać zapisane ciasteczko.	Sesja zawiera ciasteczko o nazwie „shoppingList”	Sesja zawiera ciasteczko o nazwie „shoppingList”

Nazwa:	**3.User lists managing** (plik: loginAndAppUsage.cy.ts) – operacje na listach.
Warunki początkowe:	- w bazie istnieje użytkownik nazwie: „testuseraccount” - utworzona została sesja dla użytkownika „testuseraccount” - w sesji zapisane jest ciasteczko z tokenem JWT
Dane testowe:	- nazwy list: [“test list name 1”, “test list name 2”, “test list name 3”, “test list name 2 edited”, “test list name 3 edited”]

	- daty: [“2022-08-15”, “2022-09-18”, “2023-01-18”, “2022-10-15”]
Typ testu:	pozytywny

L.P	Opis kroku	Oczekiwany rezultat	Rzeczywisty rezultat
1.	W sesji powinno zostać zapisane ciasteczko.	Sesja zawiera ciasteczko o nazwie „shoppingList”	Sesja zawiera ciasteczko o nazwie „shoppingList”
2.	Dodanie pierwszej listy	W listach użytkownika pojawi się dodana lista.	W listach użytkownika pojawia się dodana lista.
3.	Dodanie drugiej listy	W listach użytkownika pojawi się druga dodana lista.	W listach użytkownika pojawia się druga dodana lista.
4.	Dodanie trzeciej listy	W listach użytkownika pojawi się trzecia dodana lista.	W listach użytkownika pojawia się trzecia dodana lista.
5.	Wpisanie danych czwartej listy z anulowaniem jej dodania.	- Interfejs dodawania nowych list zostaje wyczyszczony. - W listach użytkownika nie pojawia się niedodana lista.	- Interfejs dodawania nowych list został wyczyszczony. - W listach użytkownika nie pojawiła się niedodana lista.
6.	Edycja drugiej listy	- Druga lista zostaje zedytowana. - Po edycji lista zawiera aktualne dane.	- Druga lista została zedytowana. - Po edycji lista zawierała aktualne dane.
7.	Edycja trzeciej listy z anulowaniem	Trzecia lista pozostaje niezmieniona.	Trzecia lista pozostała niezmieniona.
9.	Usunięcie 2 i 3 listy	Obie listy przestają istnieć.	Obie listy przestały istnieć.

Nazwa:	**4. Items managing** (plik: loginAndAppUsage.cy.ts) – operacje na przedmiotach.
Warunki początkowe:	- w bazie istnieje użytkownik nazwie: „testuseraccount” - utworzona została sesja dla użytkownika „testuseraccount” - w sesji zapisane jest ciasteczko z tokenem JWT - zalogowany użytkownik posiada listę o nazwie „test list name 1”
Dane testowe:	- nazwy przedmiotów: [“Jabłka”, “Mleko kozie”, “Mleko kokosowe”] - ilości: [3,2] - jednostki: [kg, szt] - zdjęcia: jablka.jpg
Typ testu:	pozytywny

L.P	Opis kroku	Oczekiwany rezultat	Rzeczywisty rezultat
1.	Kliknięcie w pierwszą listę użytkownika	Przekierowanie na stronę http://localhost:4200/shopping-list?listId=	Przekierowano na stronę http://localhost:4200/shopping-list?listId=

2.	Dodanie nowego przedmiotu wykorzystując predefiniowane możliwości wraz ze zdjęciem.	W przedmiotach listy pojawia się nowo utworzony przedmiot wraz ze zdjęciem.	W przedmiotach listy pojawił się nowo utworzony przedmiot wraz ze zdjęciem.
3.	Dodanie drugiego nowego przedmiotu z własną nazwą	W przedmiotach listy pojawia się nowo utworzony przedmiot.	W przedmiotach listy pojawił się nowo utworzony przedmiot.
4.	Anulowanie dodania nowego przedmiotu.	- Interfejs dodawania nowych przedmiotów zostaje wyczyszczony. - W przedmiotach listy nie pojawia się niedodana przedmiot.	- Interfejs dodawania nowych przedmiotów został wyczyszczony. - W przedmiotach listy nie pojawił się niedodana lista przedmiot.
5.	Wpisanie danych czwartej listy z anulowaniem jej dodania.	- Interfejs dodawania nowych list zostaje wyczyszczony. - W listach użytkownika nie pojawia się niedodana lista.	- Interfejs dodawania nowych list został wyczyszczony. - W listach użytkownika nie pojawiła się niedodana lista.
6.	Zaznaczenie checkbox'a przedmiotu 1 i 2.	Oba przedmioty zmieniają swój stan na wykonane (skreślone).	Oba przedmioty zmieniły swój stan na wykonane (skreślone).
7.	Odznaczenie checkbox'a przedmiotu 2.	Drugi przedmiot pozostaje odznaczony i wraca do stanu pierwotnego.	Drugi przedmiot został odznaczony i wraca do stanu pierwotnego.
9.	Edycja drugiego przedmiotu.	Nazwa przedmiotu zostaje zmieniona, do przedmiotu zostaje dodany obrazek.	Nazwa przedmiotu została zmieniona, do przedmiotu został dodany obrazek.
10.	Usunięcie drugiego przedmiotu.	Przedmiot przestaje istnieć.	Przedmiot przestał istnieć.
11.	Powrót do list użytkownika (kliknięcie przycisku „twoje listy")	Przekierowanie na stronę http://localhost:4200/user-lists.	Przekierowano na stronę http://localhost:4200/user-lists.
12.	Usunięcie listy z przedmiotami.	Lista zostaje usunięta.	Lista została usunięta

Nazwa:	**5.Logout** (plik: loginAndAppUsage.cy.ts) – wylogowanie.
Warunki początkowe:	- w bazie istnieje użytkownik nazwie: „testuseraccount" - utworzona została sesja dla użytkownika „testuseraccount" - w sesji zapisane jest ciasteczko z tokenem JWT
Dane testowe:	Brak
Typ testu:	pozytywny

L.P	**Opis kroku**	**Oczekiwany rezultat**	**Rzeczywisty rezultat**
1.	Kliknięcie w przycisk „Wyloguj"	- Przekierowanie na stronę http://localhost:4200/login. - Usunięcie ciasteczka z tokenem JWT.	- Przekierowano na stronę http://localhost:4200/login. - Usunięto ciasteczko z tokenem JWT.

5.Testy bezpieczeństwa

Użyte narzędzia:

Testy przeprowadzono z wykorzystaniem narzędzia OWASP ZAP. Po wykonaniu testów został wygenerowany raport html podsumowujący wyniki przeprowadzonych testów.

Wymagania:

- Posiadanie uruchomionej aplikacji.
- Posiadanie narzędzia OWASP ZAP (w testach użyto wersji 2.12.0).

Procedura testowania:

W celu przeprowadzenia testów aplikacja została uruchomiona w trybie produkcyjnym, za pomocą komend:

ng build --configuration production

ng serve --configuration production

Następnie zostało uruchomione narzędzie OWASP ZAP, z którego wykorzystaniem przeprowadzono test bezpieczeństwa składający się z dwóch etapów:

- wykorzystanie wbudowanego narzędzia „Automated Scan” do przeprowadzenia w pełni automatycznego ataku testowego.
- wykorzystanie narzędzia „Manual Explore” umożliwiającego manualną eksplorację aplikacji według indywidualnego scenariusza.

Indywidualny scenariusz składał się z takich kroków jak:

1. Rejestracja
2. Logowanie
3. Utworzenie list
4. Modyfikacja list
5. Dodanie nowych przedmiotów
6. Modyfikacja istniejących przedmiotów
7. Usuwanie przedmiotów
8. Usuwanie list
9. Wylogowanie

Wyniki raportu:

Szczegółowe wyniki testu bezpieczeństwa znajdują się w:
ShoppingList\security_test\2022-12-05-ZAP-Report-build-prod.html

		Risk			
		Wysoki (= Wysoki)	Średni (>= Średni)	Niski (>= Niski)	Informacyjny (>= Informacyjny)
Site	http://localhost:4200	0 (0)	4 (4)	2 (6)	2 (8)

Alert type	Risk	Count
CSP: Wildcard Directive	Średni	2 (25,0%)
Content Security Policy (CSP) Header Not Set	Średni	2 (25,0%)
Cross-Domain Misconfiguration	Średni	10 (125,0%)
Missing Anti-clickjacking Header	Średni	2 (25,0%)
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Niski	10 (125,0%)
X-Content-Type-Options Header Missing	Niski	8 (100,0%)
Information Disclosure - Suspicious Comments	Informacyjny	3 (37,5%)
Modern Web Application	Informacyjny	2 (25,0%)
Total		8

Interpretacja wyników:

Nazwa zagrożenia	Opis	Rozwiązanie
CSP: Wildcard Directive	Źle skonfigurowany CSP	Poprawić konfiguracje CSP
Content Security Policy (CSP) Header Not Set	Brak nagłówka CSP	Poprawić konfigurację Content Security Policy.
Cross-Domain Misconfiguration	Zbyt ogólny CORS	Ustawić większe restrykcje CORS.
Missing Anti-clickjacking Header	Brak nagłówka X-Frame-Options chroniącego przed atakami ClickJacking	Ustawienie nagłówka X-Frame-Options
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Wyciek informacji w nagłówku X-Powered-By odpowiedzi aplikacji	Usunięcie nagłówka X-Powered-By
X-Content-Type-Options Header Missing	X-Content-Type-Options nie został ustawiony na „nosniff”	Właściwa konfiguracja nagłówka X-Content-Type-Options
Information Disclosure - Suspicious Comments	Wygląda na to, że odpowiedź zawiera podejrzane komentarze, które mogą pomóc atakującemu.	Usuń wszystkie komentarze, które zwracają informacje, które mogą pomóc atakującemu
Modern Web Application	Aplikacja wygląda na nowoczesną aplikację internetową.	To jest alert informacyjny, więc nie są wymagane żadne zmiany.

6.Testy wydajnościowe

Użyte narzędzia:

Testy wydajnościowe przeprowadzono z wykorzystaniem dwóch narzędzi:

- **Blazemeter** – wykorzystany jako narzędzie do nagrania kolejnych kroków testu.
- **Jmeter** – wykorzystany do przeprowadzenia właściwego testu wydajnościowego, wykorzystując nagrany scenariusz testowy pochodzący z narzędzia Blazemeter.

Wymagania:

- Posiadanie uruchomionej aplikacji.
- Posiadanie oprogramowania Jmeter.

Procedura testowania:

Procedura przeprowadzenia testów składała się z 4 następujących po sobie faz:

1. **Nagranie scenariusza testowego narzędziem Blazemeter** – testowy scenariusz zakładał:
    - Logowanie.
    - Dodanie nowych list.
    - Edycja list.
    - Usunięcie list.
    - Dodanie nowych przedmiotów do listy (z obrazkiem).
    - Edycja przedmiotów.
    - Usunięcie przedmiotów.
    - Usunięcie listy z przedmiotami.
    - Wylogowanie

2. **Ręczne poprawienie wyprodukowanego artefaktu** (plik: SHOPPINGLIST_PERFORMANCE.jmx) – plik wymagał poprawienia, ponieważ poszczególne kroki scenariusza zawierały „zahardcodowane" wartości zmiennych (id), które w każdej iteracji scenariusza są inne.

3. **Rozgrzanie środowiska** – kilku krotne uruchomienia krótkiego scenariusza testowego.

4. **Przeprowadzenie właściwego testu** – uruchomiony test zakładał równoczesne użycie 200 użytkowników, każdy po 200 iteracji scenariusza. Testy zostały przerwane po pół godziny, ponieważ nie zauważono żadnych zmian.

Wyniki testu:

W wyniku wykonania testów wyprodukowany został plik:

ShoppingList\performance_test\test-results.jtl

Zawiera on dane przeprowadzonego testu. W surowej formie jest on jednak nieczytelny, dlatego do interpretacji jego wyników skorzystałem z wbudowanego narzędzia umożliwiającego wygenerowanie czytelnego raportu html. W celu wygenerowania raportu skorzystałem z komendy:

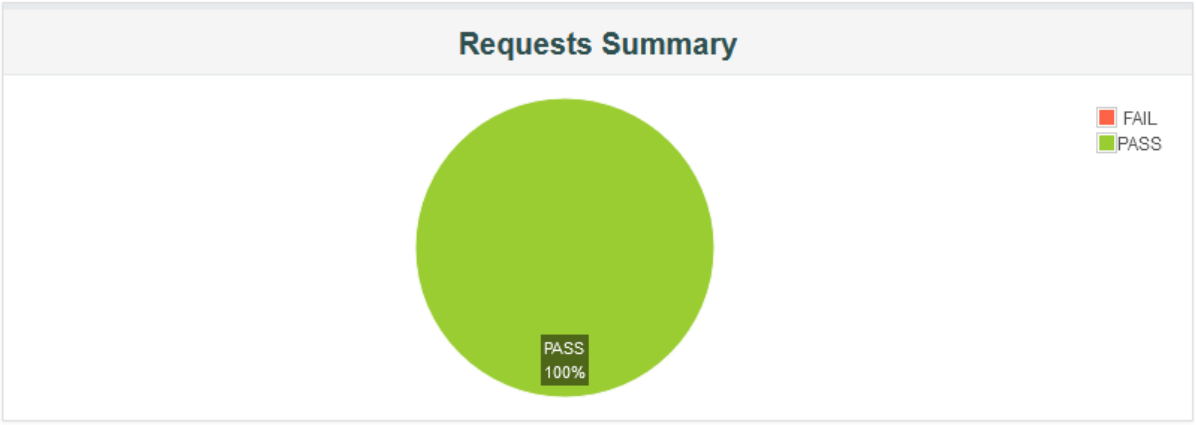
jmeter -g test-results.jtl -o ./html

Szczegółowe wyniki testu znajdują się w pliku:

ShoppingList\performance_test\html\index.html

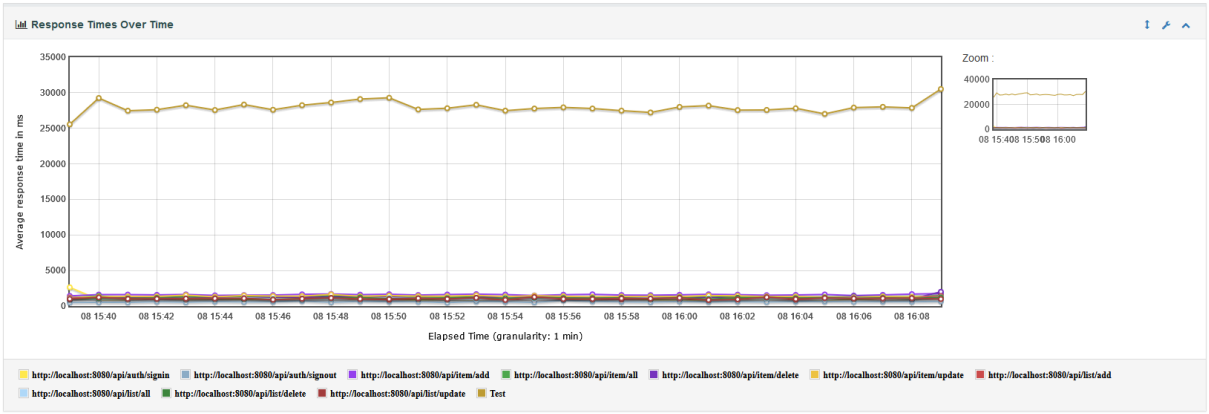
Poniżej zostały zaprezentowane kilka najistotniejszych wykresów z całego raportu.

Statystyka obsługi żądań (FAIL/PASS):

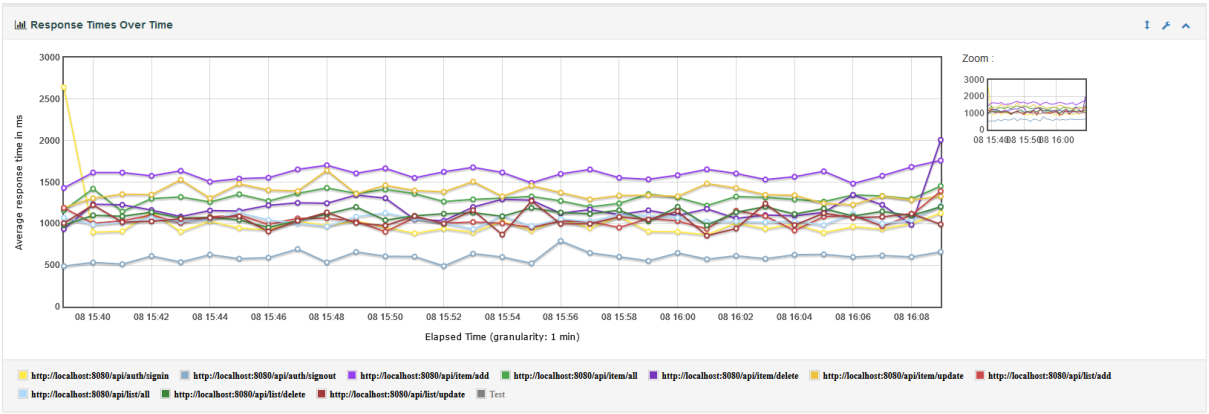


Statystyka czasu odpowiedzi:

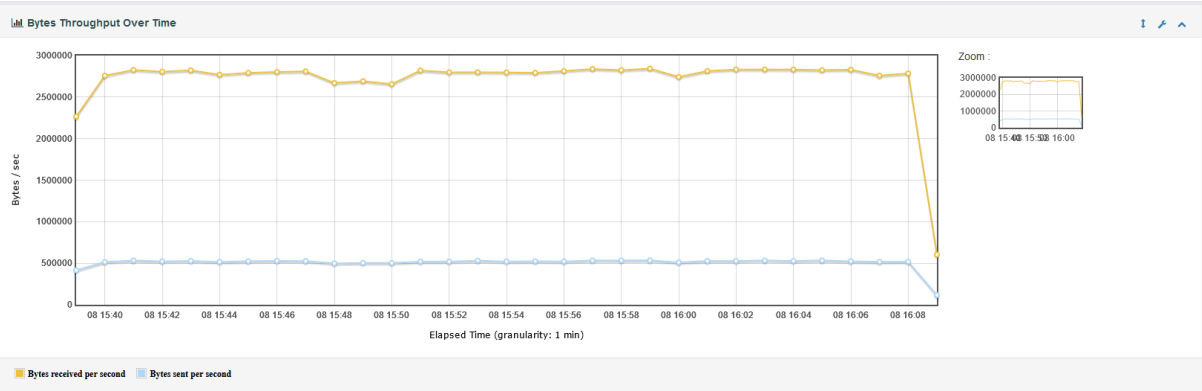
- zawierająca czas wykonania całego scenariusza



- bez całkowitego czasu:

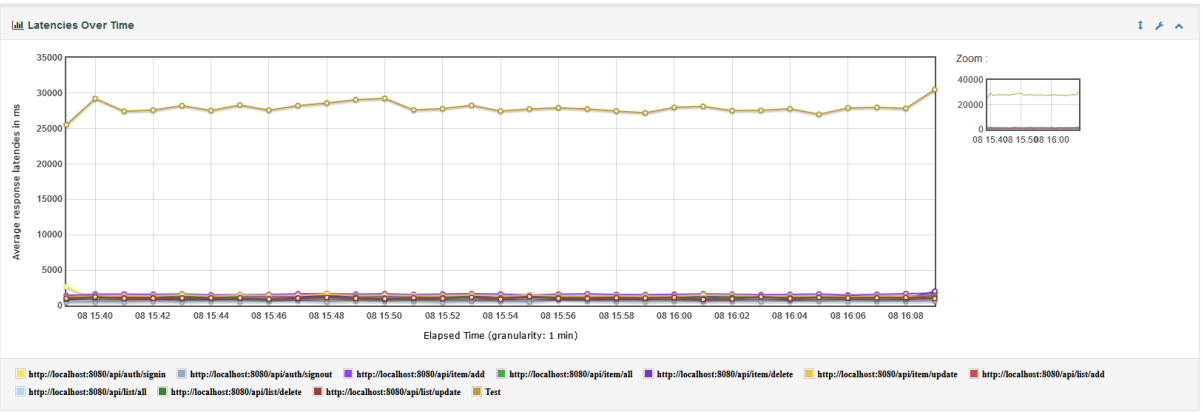


Przepustowość łącza w czasie:

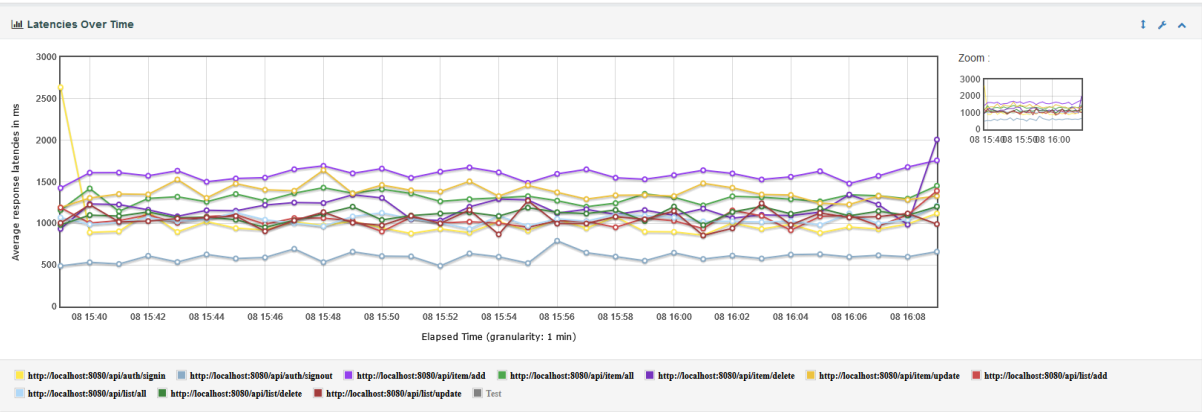


Opóźnienia w czasie:

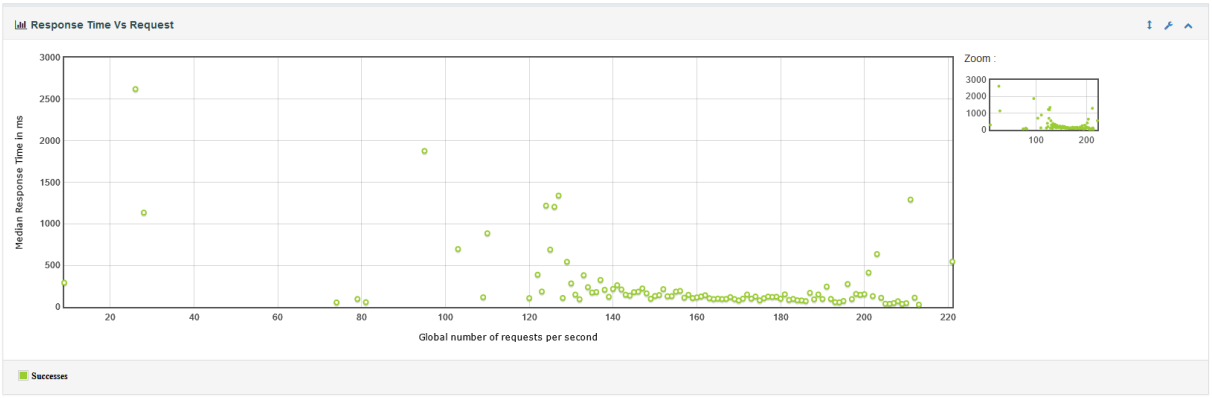
- z uwzględnieniem całego scenariusza



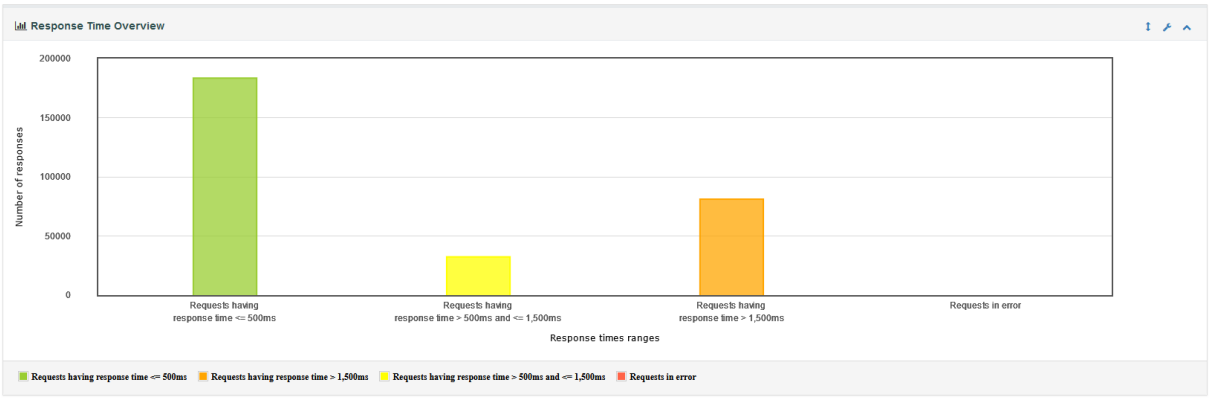
-bez całego scenariusza:



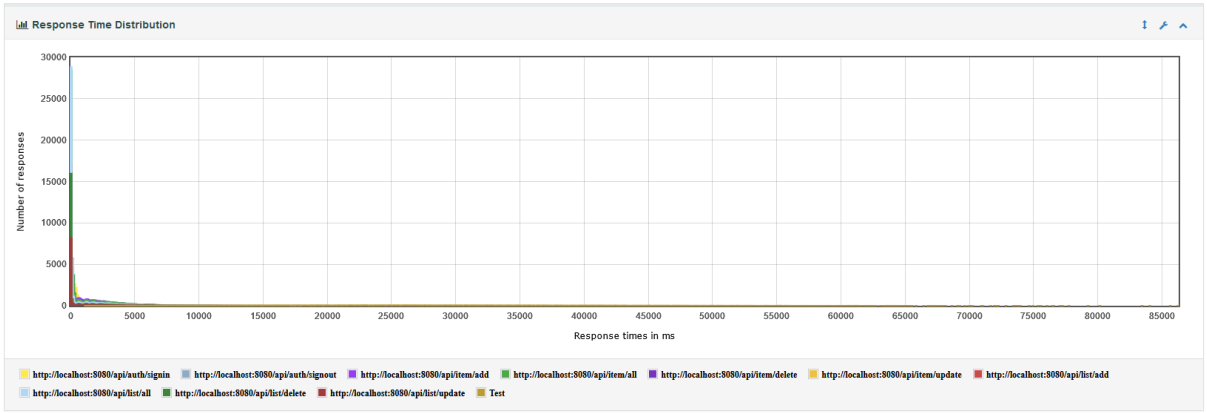
Rozkład czasu odpowiedzi w zależności od ilości żądań:



Ogólne podsumowanie czasu odpowiedzi:



Dystrybucja czasów odpowiedzi:



Wnioski:

Jak widać na powyższych wykresach praca aplikacji była stabilna i nie wykazywała żadnych niepokojących wahań. Z przeprowadzonych testów wynika, że najbardziej wymagającym żądaniem było dodawanie nowych przedmiotów oraz ich edycja. Moim zdaniem nie ma w tym nic dziwnego, ponieważ jest to najbardziej skomplikowana czynność w przyjętym scenariuszu testowym. Czasy odpowiedzi są stabilne pomimo jednoczesnego korzystania wielu użytkowników, a zdecydowana większość żądań została obsłużona <500ms.