

WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Bezpieczeństwo aplikacji webowych

Projekt

Java Spring i Vue.js

<https://github.com/Danchivskyi/Rest>

Michał Cempa, 160740

Oleh Danchivskyi, 160822

Mateusz Irla, 163951

Rzeszów, 2023

Spis treści

1.	Opis projektu	3
2.	Baza danych.....	10
3.	Wykorzystane biblioteki	10
2.1	Backend:	10
2.2	Frontend:	10
3	Analiza backend'u od strony kodu	12
3.1	Struktura projektu	12
3.2	Plik Product.java	13
3.3	Plik Worker.java.....	14
3.4	WorkerType.java	15
3.5	Plik Repository.java	16
3.6	Plik WorkerRepository.java	16
3.7	Plik WorkerTypeRepository.java	17
3.8	Plik AuthEntryPointConfig.	17
3.9	Plik SecurityConfig.java	18
3.10	Plik ProductService.java	19
3.11	Plik Controller.java	20
3.12	Plik RestApplication.java	21
3.13	Brak zabezpieczeń przed ciągłym wysyłaniem żądań logowania:	21
3.14	Dodatkowe uwagi.....	22
4	Analiza frontend'u od strony kodu.....	22
4.1	Struktura projektu	22
4.2	Plik App.vue.....	23
4.3	Plik HelloWorld.vue	24
4.4	Plik DataFetching.vue	26
4.5	Dodatkowe uwagi.....	27
5.	Testy penetracyjne	27
5.1	Test logowania wykorzystując zmienną this.isLoggedIn opisaną w analizie kodu Vue.js:	27
5.2	Próba SQL Injection	28
5.3	Atak słownikowy.....	30
6.	Podsumowanie	31
7.	Bibliografia.....	32

1. Opis projektu

Głównym założeniem projektu było utworzenie system magazynowego pozwalającego na zarządzanie obiektami w nim zawartymi.

Projekt opiera się na zastosowaniu dwóch głównych technologii jakimi są: Java Spring oraz Vue.js. Baza danych została oparta o oprogramowanie PostgreSQL.

Po otwarciu strony internetowej naszym oczom ukazuje się panel logowania:



Login

Login

Po zalogowaniu otrzymujemy dostęp do panelu zarządzania zasobami. Możemy dodawać nowe zasoby, edytować już istniejące, usuwać je oraz wygenerować diagram kołowy umieszczonych zasobów.

Inventory Management

Fetch Inventory Data

No inventory data available.

Logout

Add New Product

Product Name:

Quantity:

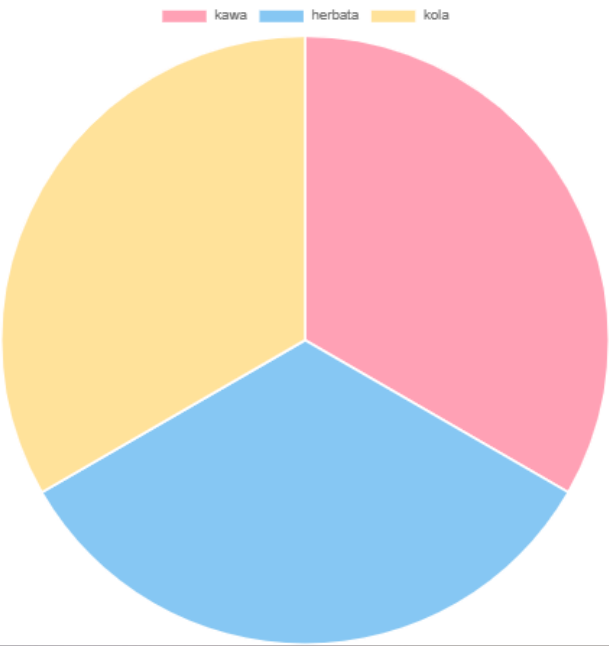
Price:

Description:

Add

Szukaj: Search Product

Name	Quantity	Price	Description		
kawa	1	10	kawa	Edit	Remove
herbata	1	4	herbata	Edit	Remove
kola	1	5.98	kola	Edit	Remove



Frontend został zbudowany na bazie Vue.js.

Struktura:

- folder **folder** jest głównym folderem frontendu
- folder **page-frontend** zawiera kod dla konkretnych stron front-endowych
- folder **node_modules** zawiera zależności projektu, zainstalowane za pomocą menedżera pakietów, np. npm.
- folder **public** zawiera pliki statyczne, takie jak pliki HTML, favicon, itp., które są bezpośrednio dostępne dla przeglądarki.
- folder **src** jest to główny folder zawierający kod źródłowy aplikacji.
- folder **assets** jest to folder, w którym przechowywane są pliki zasobów, takie jak obrazy, czcionki, itp.
- folder **components** jest to folder który zawiera komponenty Vue.js, które można wielokrotnie używać w różnych miejscach aplikacji.
- plik **HelloWorld.vue** jest to komponent Vue.js w którym jest zdefiniowano wszystko
- plik **App.vue** jest to główny komponent Vue.js, który jest punktem wejścia do aplikacji.
- plik **.gitignore**: Plik konfiguracyjny dla systemu kontroli wersji Git, który definiuje, które pliki i foldery mają być ignorowane.
- plik **babel.config.js**: Plik konfiguracyjny dla narzędzia Babel, które umożliwia transpilację kodu JavaScript na wcześniejsze wersje specyfikacji języka.
- plik **jsconfig.json**: Plik konfiguracyjny dla narzędzia JSConfig, które dostarcza informacje o projekcie dla edytora kodu.
- plik **package.json**: Plik konfiguracyjny dla menedżera pakietów npm, który zawiera informacje o projekcie oraz zależnościach, skrypcie uruchamiania i innych metadanych.
- Plik **README.md**: Plik zawierający informacje o projekcie, takie jak instrukcje instalacji, uruchamiania, itp.
- plik **vue.config.js**: Plik konfiguracyjny dla Vue CLI, który umożliwia dostosowanie konfiguracji budowania i uruchamiania aplikacji.

Plik HelloWorld.vue:

Ten kod jest implementacją aplikacji do zarządzania magazynem. Poniżej przedstawiam opis poszczególnych części kodu:

- **<template>**: Jest to sekcja, w której definiowane jest wygląd interfejsu użytkownika za pomocą języka HTML. Zawiera elementy takie jak przyciski, pola tekstowe, tabela i formularze.
- **<script>**: Ta sekcja zawiera logikę aplikacji napisaną w języku JavaScript. Zawiera wiele funkcji, które obsługują interakcje użytkownika i zarządzają danymi. Obejmuje funkcje do logowania, wylogowywania, pobierania danych z magazynu, dodawania, edytowania i usuwania produktów, generowania raportu popularnych produktów itp. Zawiera również funkcje do manipulowania danymi w lokalnym magazynie (LocalStorage).
- **<style>**: Ta sekcja zawiera style CSS, które definiują wygląd i układ interfejsu użytkownika. Obejmuje style dla formularzy logowania, dodawania produktów, tabeli, przycisków itp.

Opisując kod, są następujące kluczowe elementy:

- Komponenty Vue.js: Kod wykorzystuje framework Vue.js do budowy interfejsu użytkownika. Wykorzystuje komponenty Vue do definiowania różnych części aplikacji, takich jak formularze, tabela, przyciski itp.
- Reaktywne dane: Kod wykorzystuje reaktywne dane w Vue.js, co oznacza, że zmiany w danych są automatycznie odzwierciedlane w interfejsie użytkownika. Na przykład, gdy użytkownik dodaje nowy produkt, aktualizuje się tabela produktów.
- Przetwarzanie zdarzeń: Kod obsługuje zdarzenia, takie jak kliknięcie przycisków, wypełnienie formularzy itp. Zdarzenia te wywołują odpowiednie funkcje, które wykonują określone operacje.
- Komunikacja z serwerem: Kod wykorzystuje Fetch API do komunikacji z serwerem w celu pobrania danych z magazynu. Przykładem jest funkcja `fetchInventory`, która wysyła zapytanie do serwera po uwierzytelnieniu i otrzymuje dane magazynu.
- Zarządzanie stanem: Kod używa zmiennych stanu, takich jak `isLoggedIn`, aby śledzić, czy użytkownik jest zalogowany, oraz `inventory` i `products`, aby przechowywać dane magazynu i produkty. Stan ten jest zarządzany przez Vue.js.
- Generowanie raportów: Kod ma funkcję `generatePopularProductsReport`, która generuje raport popularnych produktów na podstawie dostępnych danych. Raport jest generowany w postaci wykresu kołowego za pomocą biblioteki `Chart.js`.

Plik `App.vue`:

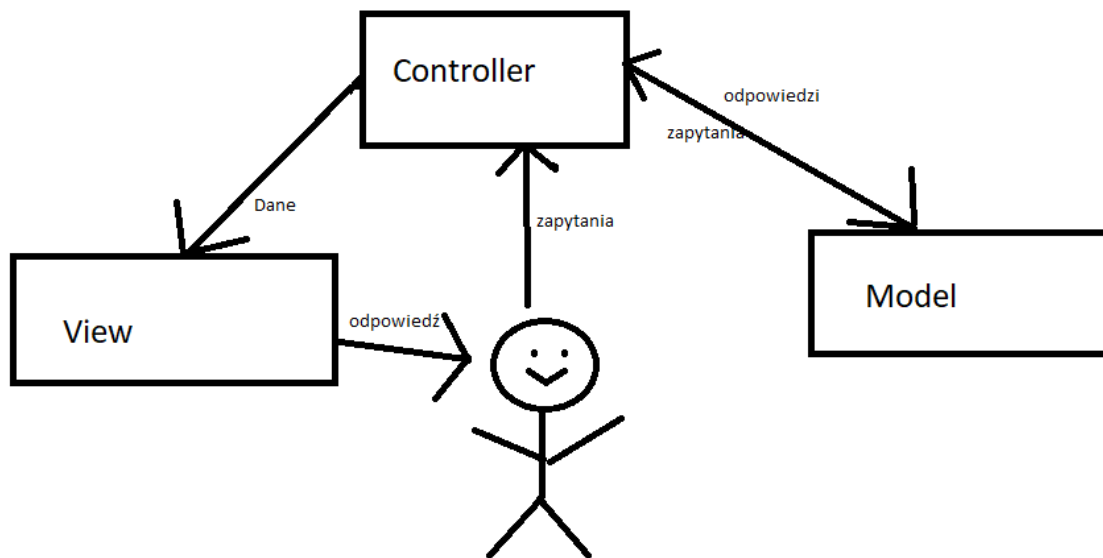
Pierwsza część kodu znajduje się w tagu `<template>` i definiuje strukturę szablonu komponentu. Wewnątrz znajduje się `<div>` o id `"app"`, zawierający `` i komponent `<HelloWorld>`. Obrazek o nazwie `"logo.png"` jest używany jako logo Vue, a komponent `<HelloWorld>` ma atrybut `msg` ustawiony na `"Welcome to Inventory Management App"`.

Druga część kodu znajduje się w tagu `<script>` i zawiera skrypt JavaScript. Importuje komponent `<HelloWorld>` z pliku `./components/HelloWorld.vue`. Następnie eksportuje obiekt, który jest instancją komponentu `"App"`. Obiekt ten ma właściwość `name` ustawioną na `"App"` oraz właściwość `components`, która zawiera komponent `<HelloWorld>`.

Trzecia część kodu znajduje się w tagu `<style>` i definiuje reguły stylów dla elementu o id `"app"`. W tym przypadku, czcionka jest ustawiona na `"Avenir, Helvetica, Arial, sans-serif"`, a wygładzanie czcionki jest włączone. Tekst jest wyśrodkowany, a kolor tekstu to `"#2c3e50"`. Dodatkowo, margines górnego brzegu wynosi 60 pikseli.

Ten kod jest podstawowym szkieletem aplikacji Vue.js. Komponent `"App"` jest korzeniem aplikacji, a komponent `"<HelloWorld>"` może być innym komponentem używanym wewnątrz `"App"`. Plik `./components/HelloWorld.vue` zawiera definicję komponentu `"<HelloWorld>"`.

Projekt jest wykonany w modelu MVC, technologia użyta to Spring, Maven, oraz baza Postgress



- Folder data zawiera klasy mapujące tabele z bazy danych na obiekty Java
- Folder Repository zawiera interfejsy pozwalające na kontakt z bazą danych
- Folder Service zawiera klasy służące do komunikacji pomiędzy kontrolerem i bazą danych
- W folderze security umieszczono klasy odpowiedzialne za zastosowane mechanizmy bezpieczeństwa
- Klasa Controller zawiera kod odpowiadający za wystawione endpoint'y

W klasie Controller następuje obsługa zdefiniowanych endpointów, które służą do obsługi zapytań i przetwarzania danych, głównie realizowane jest przedstawianie danych z bazy, wyszukiwanie produktów, dodawanie nowych oraz usuwanie istniejących. Klasa komunikuje się z innymi klasami i interfejsami by uzyskać dane lub wykonać funkcje.

```
1 package Projekt.Rest;
2
3 import ...
4
11 no usages
12 @RestController
13 @CrossOrigin(origins = "http://localhost:8081")
14 public class Controller {
15     5 usages
16     @Autowired
17     Repository repository;
18     1 usage
19     @Autowired
20     ProductService productService;
21
22     no usages
23     @GetMapping("/products")
24     List<Product> all() { return repository.findAll(); }
25
26     no usages
27     @GetMapping("/products/{name}")
28     Product singleProduct(@PathVariable Long name){
29         if(repository.findById(name).isPresent())
30             return repository.findById(name).get();
31         return null;
32     }
33
34     no usages
35     @GetMapping("/products/{name}")
36     Product singleProductTwo(@PathVariable String name) { return productService.findByName(name); }
37
38     no usages
39     @PostMapping("/products")
40     Product newProduct(@RequestBody Product newProduct) { return repository.save(newProduct); }
41
42     no usages
43     @DeleteMapping("/products/{name}")
44     void deleteProduct(@PathVariable Long name) { repository.deleteById(name); }
45 }
46
47
48
```

Zależnie od wybranego endpoint'u, możemy ich użyć do przeglądania dostępnych pozycji znajdujących się w bazie danych, dodawanie nowych pozycji oraz usuwanie pozycji.

- Klasy ProductService oraz Repository odpowiadają za obsługę zapytania znajdującego Id produktu o podanej nazwie


```
AuthEntryPointConfig.java × Controller.java × ProductService.java × Repository.java × SecurityConfig.java ×
1 package Projekt.Rest.Repository;
2
3 import ...
4
5
6
7
8
9 4 usages
10 public interface Repository extends JpaRepository<Product, Long> {
11
12     1 usage
13     @Query(nativeQuery = true, value="select * from Product where product_name = :nazwa")
14     Product getProdByName(@Param(value = "nazwa")String nazwa);
15 }
```

```
AuthEntryPointConfig.java × Controller.java × ProductService.java × Repository.java × SecurityConfig.java ×
1 package Projekt.Rest.Service;
2
3 import ...
4
5
6
7
8 2 usages
9 @Service
10 public class ProductService {
11
12     1 usage
13     @Autowired
14     Repository repo;
15
16     1 usage
17     public Product findByName(String name) { return repo.getProdByName(name); }
18 }
```

Na tej podstawie używamy go by usunąć podaną pozycje z bazy danych

- Klasa SecurityConfig odpowiada za ustawienie mechanizmów bezpieczeństwa

```
23 no usages
24 @Bean
25 public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
26     http
27         .authorizeHttpRequests().anyRequest().authenticated().and().httpBasic().authenticationEntryPoint(authenticationEntryPoint);
28     return http.build();
29 }
30
31 1 usage
32 @Bean
33 public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
```

Tutaj ustawiamy by każde zapytanie obsługiwane przez stronę mogło być tylko obsłużone, gdy użytkownik jest uwierzytelniony

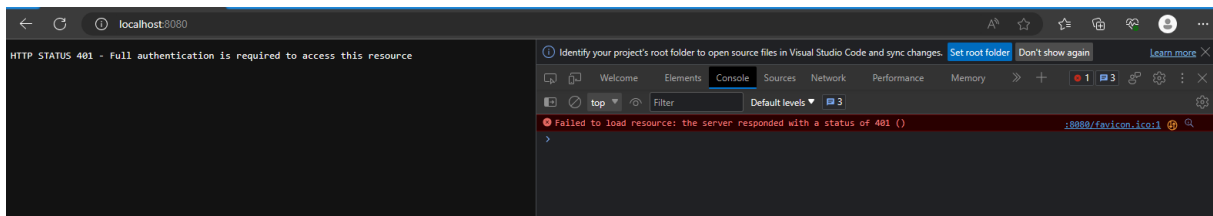
```

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.jdbcAuthentication()
        .usersByUsernameQuery("select login, password, enabled from worker where login = ?")
        .authoritiesByUsernameQuery("select w.login, wt.name from worker w , worker_type wt \n" +
            "where w.worker_type_id = wt.worker_type_id AND login = ?")
        .dataSource(dataSource)
        .passwordEncoder(passwordEncoder());
}

```

Użytkownicy i role są wczytywane z bazy danych z tabel Worker i WorkerType. Pozwala to na dodawanie nowych ról i użytkowników.

Co do backendu to jest on zabezpieczony przed niepożądanym dostępem, ponieważ w przypadku braku podania jakichkolwiek danych uwierzytelniania i próbie dostania się do niego otrzymamy błąd 401.



2. Baza danych

Baza danych została oparta o oprogramowanie PostgreSQL w wersji 15.3. Zgodnie ze stroną autora: <https://www.postgresql.org/support/security/> nie posiada ona żadnych wykrytych wrażliwości. W bazie danych zostały utworzone 3 tabele o nazwach: product, worker, worker_type. W każdej z tabel zostały umieszczone kolumny. Każda z kolumn jest odpowiednio skonfigurowana mając jeden klucz główny, posiada odpowiedni typ danych oraz ograniczenia co do długości wprowadzanych danych do bazy.

3. Wykorzystane biblioteki

2.1 Backend:

- maven:org.yaml:snakeyaml:1.33 (**wrażliwy**) – konstruktor klasy nie ogranicza typów, które mogą być tworzone podczas deserializacji. Deserializacja zawartości „yaml” dostarczonej przez osobę atakującą może prowadzić do ataku typu „Remote Code Execution”.

Cała reszta bibliotek została sprawdzona i nie posiada żadnych wykrytych i zgłoszonych wrażliwości. Oczywiście w dalszym ciągu mogą je posiadać, lecz nie zostały ujawnione.

2.2 Frontend:

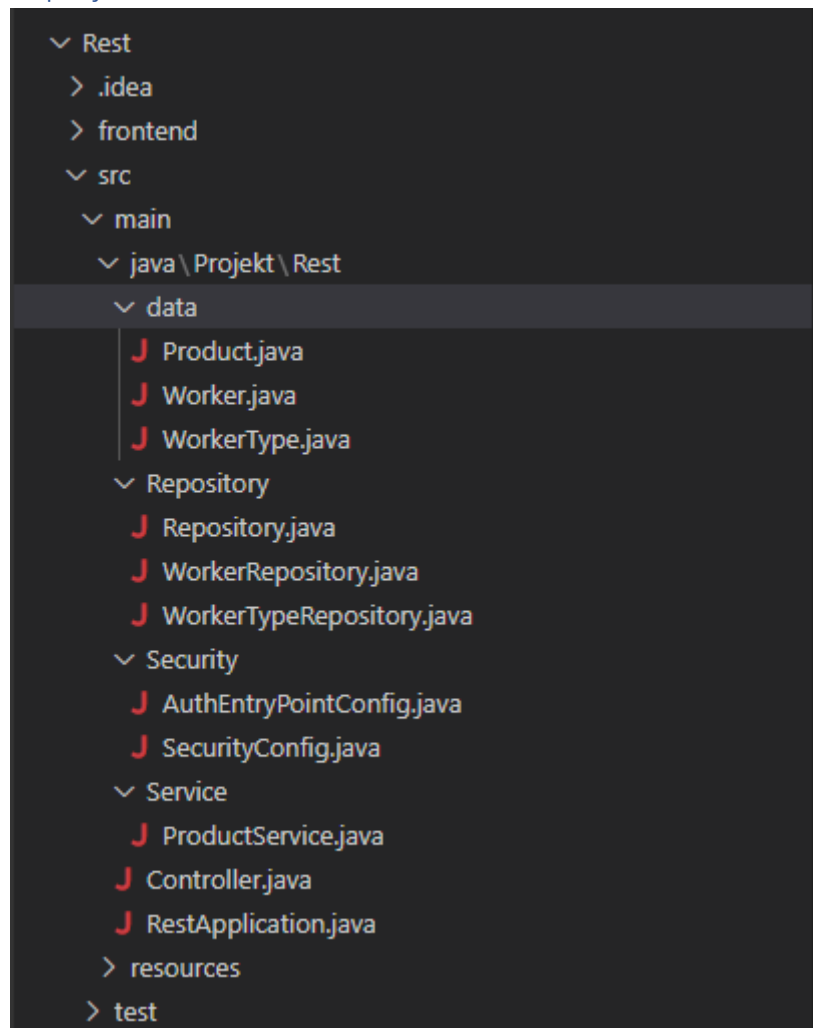
Każda z wykorzystanych bibliotek po stronie frontendu została sprawdzona. Wszystkie z nich posiadają możliwą najnowszą wersję i nie posiadają wykrytych luk.

```
@babel/core@7.22.1
@babel/eslint-parser@7.21.8
@vue/cli-plugin-babel@5.0.8
@vue/cli-plugin-eslint@5.0.8
@vue/cli-service@5.0.8
chart.js@4.3.0
core-js@3.30.2
eslint-plugin-vue@8.7.1
eslint@7.32.0
vue-tester@0.0.1
vue@3.3.4
```

Biblioteki te zostały sprawdzone na stronie: security.snyk.io na, której zamieszczone są informacje o wrażliwych wersjach bibliotek.

3 Analiza backend'u od strony kodu

3.1 Struktura projektu



3.2 Plik Product.java – definicja klasy „Product”, która reprezentuje encję w bazie danych.

```
7  @Entity
8  @Table(name = "product")
9  public class Product {
10
11      @Id
12      @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "product_generator")
13      @SequenceGenerator(name = "product_generator", sequenceName = "product_seq", allocationSize = 1)
14      @Column(name = "product_id", nullable = false)
15      private long productId;
16
17      @Column(name = "name", nullable = false)
18      private String name;
19
20
21      @Column(name = "available_quantity", nullable = false)
22      @Check(constraints = "available_quantity >= 0")
23      private long availableQuantity;
24
25
26      public Product() {
27
28      }
29
30      public long getProductId() {
31          return productId;
32      }
33
34      public void setProductId(long productId) {
35          this.productId = productId;
36      }
37
38      public String getName() {
39          return name;
40      }
41
42      public void setName(String name) {
43          this.name = name;
44      }
45
46
47      public long getAvailableQuantity() {
48          return availableQuantity;
49      }
50
51      public void setAvailableQuantity(long availableQuantity) {
52          this.availableQuantity = availableQuantity;
53      }
54
55 }
```

- Walidacja danych: W kodzie zastosowano ograniczenie na kolumnie ‘available_quantity’, które uniemożliwia stosowanie ujemnych wartości,
- Brak uwzględnienia innych aspektów walidacji: W kodzie **brakuje dalszej walidacji danych, takiej jak sprawdzenie, czy pole name jest puste lub ograniczenie jego długości**. Aby zapewnić poprawność danych i uniknąć potencjalnych ataków, warto rozważyć dodanie dodatkowych walidacji do pól klasy Product w zależności od wymagań aplikacji.

3.3 Plik Worker.java – definicja klasy „Worker”, która reprezentuje pracownika w systemie

```
6 @Entity
7 @Table(name = "worker")
8 public class Worker {
9
10     @Id
11     @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "worker_generator")
12     @SequenceGenerator(name = "worker_generator", sequenceName = "worker_seq", allocationSize = 1)
13     @Column(name = "worker_id", nullable = false)
14     private long workerId;
15
16     @Column(name = "name", nullable = false)
17     private String name;
18
19     @Column(name = "password", nullable = false)
20     private String password;
21
22     @Column(name = "email", nullable = false, unique = true)
23     private String email;
24
25     @Column(name = "enabled", nullable = false)
26     private Boolean enabled;
27
28     @ManyToOne
29     @JoinColumn(name = "worker_type_id")
30     private WorkerType workerType;
31
32     public Worker(long workerId, String name, String surname, String login, String password, String email, WorkerType workerType) {
33         this.workerId = workerId;
34         this.name = name;
35         this.password = password;
36         this.email = email;
37         this.workerType = workerType;
38     }
39
40     public Worker() {
41
42     }
43
44     public long getWorkerId() {
45         return workerId;
46     }
47
48     public void setWorkerId(long workerId) {
49         this.workerId = workerId;
50     }
51
52     public String getName() {
53         return name;
54     }
55
56     public void setName(String name) {
57         this.name = name;
58     }
59
60
61     public String getPassword() {
62         return password;
63     }
64
65     public void setPassword(String password) {
66         this.password = password;
67     }
68
69     public String getEmail() {
70         return email;
71     }
72
73     public void setEmail(String email) {
74         this.email = email;
75     }
76
77     public WorkerType getWorkerType() {
78         return workerType;
79     }
80
81     public void setWorkerType(WorkerType workerType) {
82         this.workerType = workerType;
83     }
84
85
86 }
87
88 }
```

- **Unikalność adresu e-mail:** Pole „e-mail” jest oznaczone jako „unique” co oznacza, że każdy pracownik musi mieć unikalny adres e-mail. To dobre zabezpieczenie, które zapobiega duplikatom adresów e-mail w systemie.
- **Brak walidacji pól:** W kodzie brakuje walidacji pól takich jak „name”, „password”, „email”.

3.4 WorkerType.java – definicja klasy „WorkerType” odnosząca się do typu pracownika w systemie

```
1  package Projekt.Rest.data;
2
3
4  import jakarta.persistence.*;
5
6  @Entity
7  @Table(name = "worker_type")
8  public class WorkerType {
9
10     @Id
11     @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "worker_type_generator")
12     @SequenceGenerator(name = "worker_type_generator", sequenceName = "worker_type_seq", allocationSize = 1)
13     @Column(name = "worker_type_id", nullable = false)
14     private long workerTypeId;
15
16     @Column(name = "name", nullable = false)
17     private String name;
18
19
20     public WorkerType(long workerTypeId, String name) {
21         this.workerTypeId = workerTypeId;
22         this.name = name;
23     }
24
25     public WorkerType() {
26
27     }
28
29     public long getWorkerTypeId() {
30         return workerTypeId;
31     }
32
33     public void setWorkerTypeId(long workerTypeId) {
34         this.workerTypeId = workerTypeId;
35     }
36
37     public String getName() {
38         return name;
39     }
40
41     public void setName(String name) {
42         this.name = name;
43     }
44 }
45
```

Nie znaleziono nic.

3.5 Plik `Repository.java` – plik reprezentujący interfejs repozytorium „Repository” dla klasy „Product”

```
1 package Projekt.Rest.Repository;
2
3 import Projekt.Rest.data.Product;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.data.jpa.repository.Query;
6 import org.springframework.data.repository.query.Param;
7
8
9 public interface Repository extends JpaRepository<Product, Long> {
10
11     @Query(nativeQuery = true, value="select * from Product where product_name = :nazwa")
12     Product getProdByName(@Param(value = "nazwa")String nazwa);
13 }
14
```

- Użycie interfejsu `JpaRepository`: Interfejs `JpaRepository` pochodzący z frameworka Spring Data JPA zapewnia ogólną funkcjonalność dostępu do bazy danych. Korzystając z tego interfejsu, można wykonywać podstawowe operacje CRUD (Create, Read, Update, Delete) na encjach. Jest to bezpieczne podejście.
- SQL Injection: Użycie natywnych zapytań może prowadzić do SQL Injection, jeżeli wprowadzane dane przez użytkownika nie są odpowiednio czyszczone i weryfikowane. Natomiast w tym przypadku został użyty tzw. „parameter binding” co chroni przed tym atakiem.

3.6 Plik `WorkerRepository.java` – kod reprezentujący interfejs repozytorium „WorkerRepository” dla klasy „Worker”

```
1 package Projekt.Rest.Repository;
2
3 import Projekt.Rest.data.Worker;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.data.jpa.repository.Query;
6 import org.springframework.data.repository.query.Param;
7 import org.springframework.stereotype.Repository;
8
9 @Repository
10 public interface WorkerRepository extends JpaRepository<Worker, Long> {
11     @Query(nativeQuery = true, value = "from worker where name = :email")
12     Worker findByEmail(@Param("email")String email);
13 }
14
```

- Tak jak powyżej

3.7 Plik `WorkerTypeRepository.java` - kod reprezentujący interfejs repozytorium „`WorkerTypeRepository`” dla klasy „`WorkerType`”

```
1 package Projekt.Rest.Repository;
2
3 import Projekt.Rest.data.WorkerType;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 @Repository
8 public interface WorkerTypeRepository extends JpaRepository<WorkerType, Long> {
9 }
```

Wszystko ok.

3.8 Plik `AuthEntryPointConfig.java` – kod reprezentujący punkty wejścia uwierzytelniania „`AuthEntryPointConfig`” w celu obsługi uwierzytelniania podstawowego

```
1 package Projekt.Rest.Security;
2
3 import jakarta.servlet.http.HttpServletRequest;
4 import jakarta.servlet.http.HttpServletResponse;
5 import org.springframework.security.core.AuthenticationException;
6 import org.springframework.security.web.authentication.www.BasicAuthenticationEntryPoint;
7 import org.springframework.stereotype.Component;
8
9 import java.io.IOException;
10 import java.io.PrintWriter;
11
12 @Component
13 public class AuthEntryPointConfig extends BasicAuthenticationEntryPoint {
14     @Override
15     public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException authException) throws IOException {
16         response.addHeader("Access-Control-Allow-Origin", "**");
17         response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
18         PrintWriter writer = response.getWriter();
19         writer.println("HTTP STATUS 401 - " + authException.getMessage());
20     }
21
22     @Override
23     public void afterPropertiesSet() {
24         setRealmName("Magazyn");
25         super.afterPropertiesSet();
26     }
27 }
28
```

- Klasa „`AuthEntryPointConfig`” dziedziczy po klasie „`BasicAuthenticationEntryPoint`” z frameworka Spring Security. Klasa ta dostarcza podstawową funkcjonalność punktu wejścia uwierzytelniania HTTP, co jest bardzo bezpiecznym rozwiązaniem.

3.9 Plik SecurityConfig.java – klasa konfiguracyjna Spring Security

```
1 package Projekt.Rest.Security;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.context.annotation.Bean;
5 import org.springframework.context.annotation.Configuration;
6 import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
7 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
8 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
9 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
10 import org.springframework.security.crypto.password.PasswordEncoder;
11
12 import org.springframework.security.web.SecurityFilterChain;
13
14
15 @Configuration
16 @EnableWebSecurity
17 public class SecurityConfig {
18
19     @Autowired private AuthEntryPointConfig authenticationEntryPoint;
20
21     @Autowired
22     public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
23         auth
24             .inMemoryAuthentication()
25             .withUser("user1")
26             .password(passwordEncoder().encode("alo"))
27             .authorities("ROLE_ADMIN");
28     }
29
30
31     @Bean
32     public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
33         http
34             .authorizeHttpRequests().anyRequest().authenticated().and().httpBasic().authenticationEntryPoint(authenticationEntryPoint);
35
36         return http.build();
37     }
38
39     @Bean
40     public PasswordEncoder passwordEncoder() {
41         return new BCryptPasswordEncoder();
42     }
43 }
44
```

- Wykorzystanie „BCryptPasswordEncoder” do haszowania haseł. Dobre zabezpieczenie, które przechowuje hasła w bezpieczny sposób,
- Wykorzystanie HTTP Basic Authentication – kod korzysta z uwierzytelniania korzystając z metody „httpBasic()”. Oznacza to, że klienci uzyskujący dostęp do aplikacji muszą podawać swoje poświadczenia w każdym żądaniu. Chociaż podstawowe uwierzytelnianie HTTP jest proste i szeroko obsługiwane, **przesyła dane uwierzytelniające w formie zakodowanej w standardzie Base64 przy każdym żądaniu, co może być podatne na przechwycenie.**

3.10 Plik ProductService.java

```
1  package Projekt.Rest.Service;
2
3  import Projekt.Rest.Repository.Repository;
4  import Projekt.Rest.data.Product;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.stereotype.Service;
7
8  @Service
9  public class ProductService {
10
11      @Autowired
12      Repository repo;
13
14      public Product findByName(String name){
15          return repo.getProdByName(name);
16      }
17  }
18
```

3.11 Plik Controller.java – definicja kontrolera w aplikacji Spring Boot

```
12  @RestController
13  @CrossOrigin(origins = "http://localhost:8081")
14  public class Controller {
15      @Autowired
16      Repository repository;
17      @Autowired
18      ProductService productService;
19
20      @GetMapping("/products")
21      List<Product> all(){
22          return repository.findAll();
23      }
24
25      @GetMapping("/products/{name}")
26      Product singleProduct(@PathVariable Long name){
27          if(repository.findById(name).isPresent())
28              return repository.findById(name).get();
29
30          return null;
31      }
32
33      @GetMapping("/productss/{name}")
34      Product singleProductTwo(@PathVariable String name){
35          return productService.findByName(name);
36      }
37
38      @PostMapping("/products")
39      Product newProduct(@RequestBody Product newProduct){
40          return repository.save(newProduct);
41      }
42
43      @DeleteMapping("/products/{name}")
44      void deleteProduct(@PathVariable Long name){
45          repository.deleteById(name);
46      }
47  }
48
```

Nie znaleziono błędów.

3.12 Plik RestApplication.java

```
1 package Projekt.Rest;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class RestApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(RestApplication.class, args);
11     }
12
13 }
14
```

Ten fragment kodu jest podstawowym plikiem uruchomieniowym aplikacji Spring Boot. Nie zawiera on żadnego kodu obsługującego żądania HTTP. W związku z tym, nie ma widocznych potencjalnych wrażliwości bezpieczeństwa.

3.13 Brak zabezpieczeń przed ciągłym wysyłaniem żądań logowania:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Autowired private AuthEntryPointConfig authenticationEntryPoint;

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .inMemoryAuthentication()
            .withUser("user1")
            .password(passwordEncoder().encode("alo"))
            .authorities("ROLE_ADMIN");
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests().anyRequest().authenticated().and().httpBasic().authenticationEntryPoint(authenticationEntryPoint);

        return http.build();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

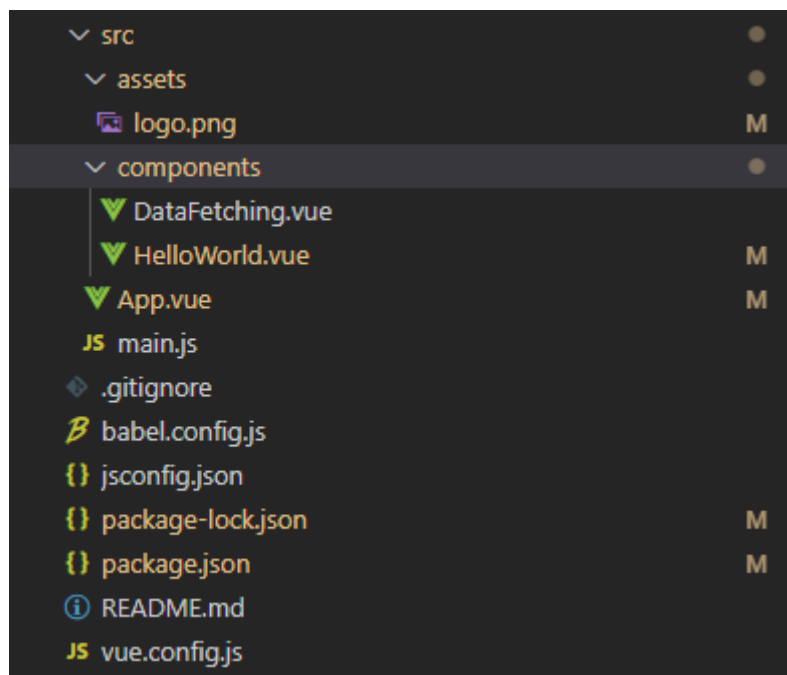
W kodzie brakuje dodatkowego zabezpieczenia utrudniającego możliwość wykonania ataków typu **brute-force**, bądź **słownikowych**. Potencjalny „włamywacz” może wykonywać nieskończoną liczbę prób dostania się do API, bez obawy o np. blokadę czasową lub IP.

3.14 Dodatkowe uwagi

- Cross-Site Scripting (XSS): Kod **nie obsługuje jawnie sprawdzania poprawności danych wprowadzanych przez użytkownika**, co może potencjalnie prowadzić do luk w zabezpieczeniach związanych z XSS.
- Cross-Origin Resource Sharing (CORS): W klasie „AuthEntryPointConfig” dodany został nagłówek „Access-Control-Allow-Origin” z symbolem „*”, **który zezwala na żądania między źródłami z dowolnej domeny**.

4 Analiza frontend’u od strony kodu

4.1 Struktura projektu



4.2 Plik `App.vue` - kod reprezentuje podstawowy szablon aplikacji Vue.js. Składa się z pojedynczego komponentu o nazwie `HelloWorld` oraz instancji `Vue`, która korzysta z tego komponentu.

```
1  <template>
2    <div id="app">
3      
4      <HelloWorld msg="Welcome to Inventory Management App"/>
5    </div>
6  </template>
7
8  <script>
9    import HelloWorld from '../components/HelloWorld.vue'
10
11    export default {
12      name: 'App',
13      components: {
14        HelloWorld
15      }
16    }
17  </script>
18
19  <style>
20    #app {
21      font-family: Avenir, Helvetica, Arial, sans-serif;
22      -webkit-font-smoothing: antialiased;
23      -moz-osx-font-smoothing: grayscale;
24      text-align: center;
25      color: #2c3e50;
26      margin-top: 60px;
27    }
28  </style>
29
```

Wszystko wydaje się być w porządku.

4.3 Plik HelloWorld.vue:

```
methods: {  
  login() {  
    if (this.username === 'user1' && this.password === 'aloo') {  
      this.isLoggedIn = true;  
      localStorage.setItem('isLoggedIn', 'true');  
    } else {  
      alert('Invalid username or password');  
    }  
  },  
  
  logout() {  
    this.isLoggedIn = false;  
    this.inventory = [];  
    this.username = '';  
    this.password = '';  
    localStorage.removeItem('isLoggedIn');  
  },  
}
```

- Aplikacja porównuje wprowadzone przez użytkownika dane logowania do danych zapisanych w pliku HelloWorld.vue, nie odwołując się do backendu. Dodatkowo brak jakichkolwiek plików cookies. To czy użytkownik jest zalogowany czy nie jest zapisywane jest po prostu w zmiennej this.isLoggedIn.

Jedyną kwestią na plus jest to, że strona internetowa nie podaje użytkownikowi informacji o tym czy wprowadził nazwę użytkownika złą czy też hasło. Po prostu zwraca komunikat, że dane logowania są nieprawidłowe.

```
<form @submit.prevent="updateProduct">  
  <div>  
    <label for="edit-product-name">Product Name:</label>  
    <input id="edit-product-name" v-model="editProductData.name" required />  
  </div>  
  <div>  
    <label for="edit-product-quantity">Quantity:</label>  
    <input id="edit-product-quantity" v-model="editProductData.quantity" type="number" required />  
  </div>  
  <div>  
    <label for="edit-product-price">Price:</label>  
    <input id="edit-product-price" v-model="editProductData.price" type="number" step="0.01" required />  
  </div>  
  <div>  
    <label for="edit-product-description">Description:</label>  
    <textarea id="edit-product-description" v-model="editProductData.description" required></textarea>  
  </div>  
  <div>  
    <button type="submit">Update</button>  
    <button @click="cancelEdit">Cancel</button>  
  </div>  
</form>
```

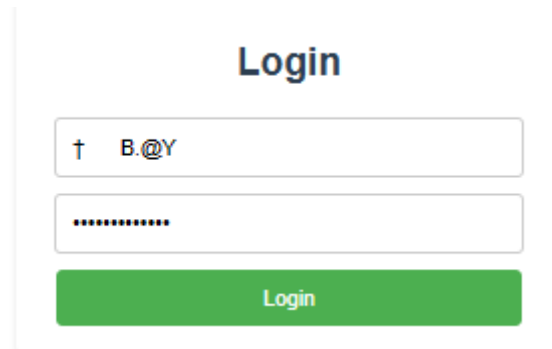
- Walidacja danych: wprowadzone odpowiednie typy pól oraz atrybuty „required”,
- Brak ograniczeń wprowadzania danych:

W polu „Product Name” oraz „Description” możemy wprowadzać dowolne znaki z tablicy ASCII o dowolnej długości.

W polu „Quantity”, „Price” możemy wprowadzać dowolną liczbę również ujemną o dowolnej długości.

Dzięki braku powyższym ograniczeniom istnieje możliwość „wysypania” bazy danych aplikacji WWW podając zbyt długie wartości, bądź znaki specjalne z tablicy ASCII. Można również wstrzyknąć nieporządkany kod, ponieważ wprowadzane dane nie są odpowiednio walidowane.

To samo dotyczy strony logowania.



Dodatkowo brak tutaj jakiegokolwiek metody dodatkowego zabezpieczenia konta, co ułatwia zadanie hakerowi.

```
181 fetchInventory() {
182   const authHeader = 'Basic ' + btoa( data: this.username + ':' + this.password);
183   fetch( input: 'http://localhost:8080/productss/', init: {
184     headers: {
185       Authorization: authHeader
186     }
187   })
188   .then(response => response.json())
189   .then(data => {
190     this.inventory = data;
191     this.saveDataToFile(data);
192   })
193 }
```

- Użycie autoryzacji podczas pobierania danych: Przy pobieraniu danych z serwera, w nagłówku żądania wysyłane jest uwierzytelnienie Basic Auth, które jest zakodowane przy użyciu funkcji btoa(). Jest to bezpieczny sposób przesyłania danych uwierzytelniających.

```
created() {
  this.isLoggedIn = localStorage.getItem('isLoggedIn') === 'true';
  const savedProducts = localStorage.getItem('products');
  if (savedProducts) {
    this.products = JSON.parse(savedProducts);
  }
},
```

- **Nieprawidłowe zarządzanie sesją:** W kodzie wykorzystywane jest przechowywanie informacji o zalogowanym użytkowniku w localStorage, jednak nie ma odpowiednich mechanizmów zarządzania sesją, takich jak wygaszanie sesji po określonym czasie nieaktywności. Dodatkowo sesje użytkowników są zarządzane w sposób niebezpieczny poprzez brak jakichkolwiek zabezpieczeń co może prowadzić do podszycia się, kradzieży sesji itp.

4.4 Plik DataFetching.vue – kod pozwalający na pobranie danych z serwera i wyświetlenie ich z komponente Vue.js

```

1  <template>
2  <div>
3    <button @click="fetchData">Pobierz dane</button>
4    <ul>
5      <li v-for="product in products" :key="product.id">{{ product.name }}</li>
6    </ul>
7  </div>
8  </template>
9
10 <script>
11 export default {
12   data() {
13     return {
14       products: []
15     };
16   },
17   methods: {
18     fetchData() {
19       fetch('/products')
20         .then(response => response.json())
21         .then(data => {
22           this.products = data;
23         })
24         .catch(error => {
25           console.error(error);
26         });
27     }
28   }
29 };
30 </script>
31

```

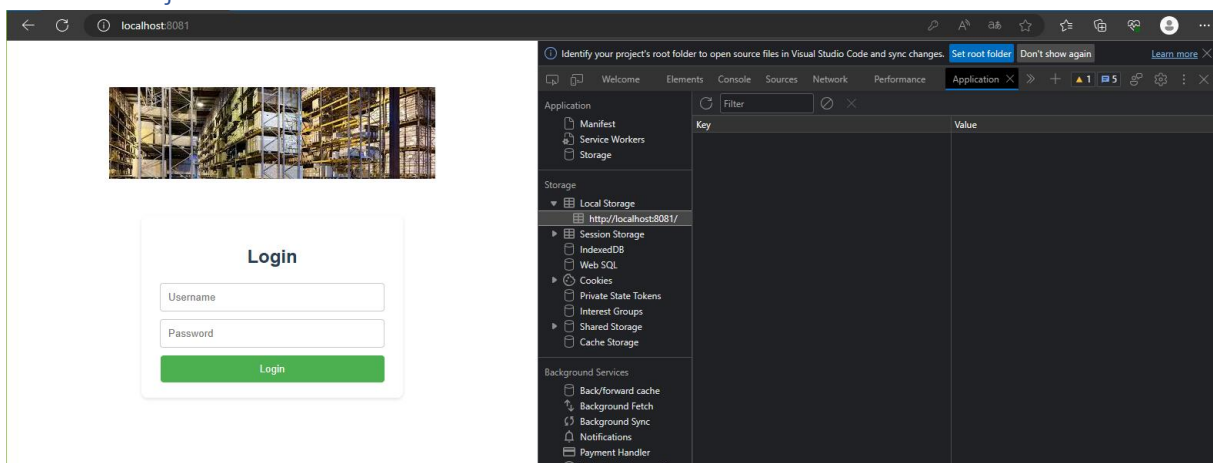
- **Wrażliwość: Atak XSS (Cross-Site Scripting)** – kod wykorzystuje interpolację {{ product.name }}, co może prowadzić do ataku XSS. Zaleca się używanie dyrektywy „v-text”. Jeżeli dane otrzymane z serwera zawierają niebezpieczne treści, mogą zostać wykonane w kontekście przeglądarki użytkownika.
- **Brak ochrony przed atakiem typu CSRF** – brak zaimplementowanego mechanizmu żetonów CSRF. Może to prowadzić do wykonania niechcianych akcji w imieniu użytkownika przez złośliwą stronę z zewnętrznego źródła

4.5 Dodatkowe uwagi:

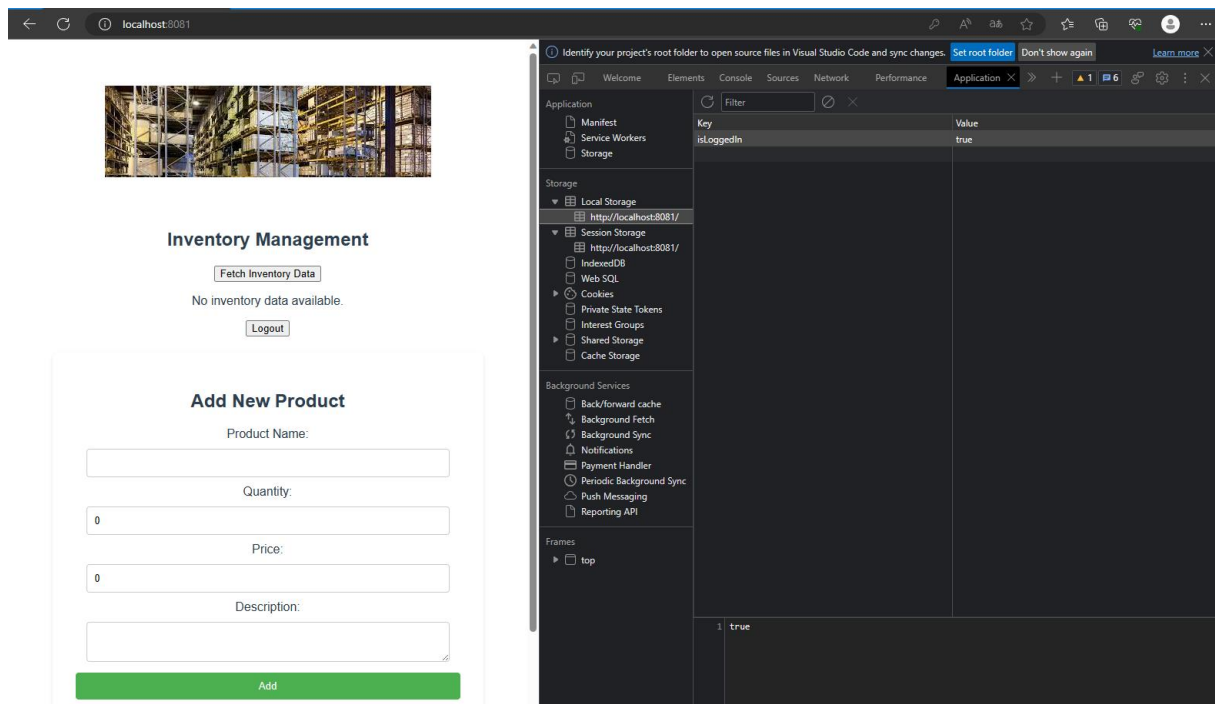
- Bardzo ważną kwestią jest **wykorzystanie protokołu HTTP**, zamiast HTTPS. Brak wykorzystania TLS/SSL niesie za sobą bardzo wiele możliwych konsekwencji, ponieważ dane przesyłane są w niezaszyfrowanej postaci. Potencjalny haker może z łatwością przejść te dane.
- Użycie dyrektyw Vue.js: Dyrektywy Vue.js, takie jak v-if i v-for, pozwalają na bezpieczne manipulowanie zawartością widoku i unikanie ataków XSS.
- Używanie Vue.js: Framework Vue.js zapewnia domyślne zabezpieczenia przed atakami XSS (Cross-Site Scripting) poprzez mechanizmy reaktywności i automatyczną filtrację danych.

5. Testy penetracyjne

5.1 Test logowania wykorzystując zmienną this.isLoggedIn opisaną w analizie kodu Vue.js:



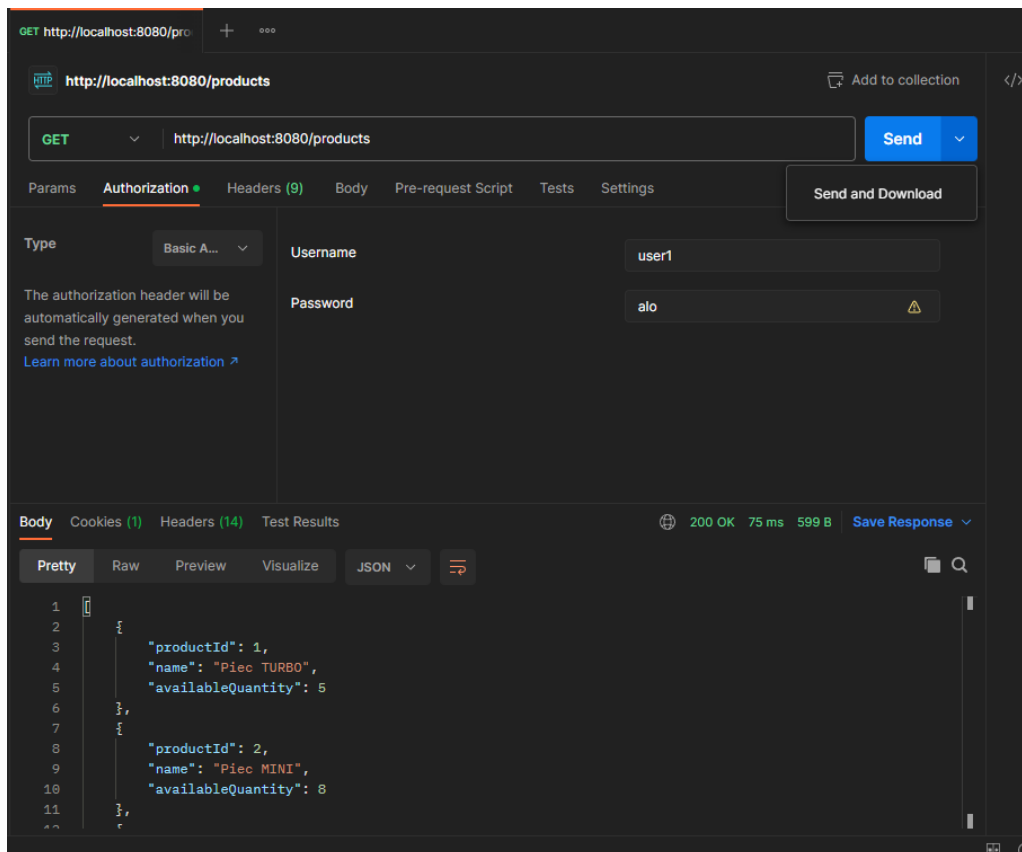
Jak widać na powyższym obrazie jesteśmy wylogowani z sesji. Prosty sposób na zalogowanie powinno być dodanie zmiennej isLoggedIn z wartością True.



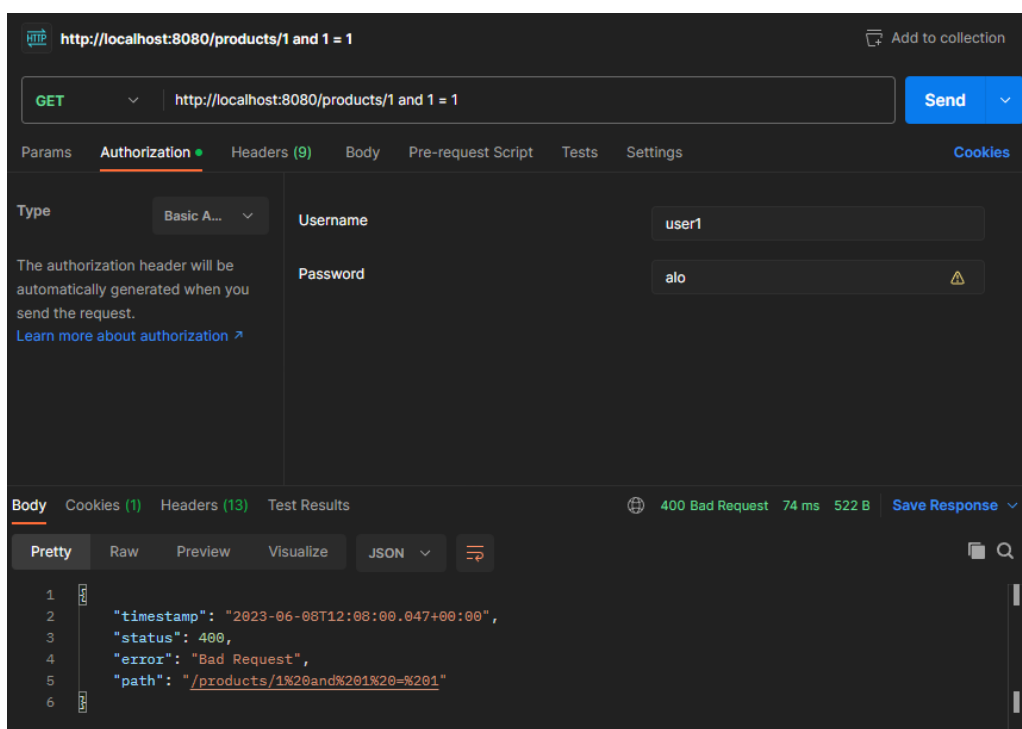
Jak widać po dodaniu właściwego klucza z wartością w łatwy sposób uzyskaliśmy dostęp do strony internetowej nie znając danych uwierzytelniających.

5.2 Próba SQL Injection

W ramach tej próby wykorzystam darmowe narzędzie Postman, które ułatwia pracę z API dzięki przejrzystemu graficznemu interfejsowi. Za jakiego pomocą wykorzystam uwierzytelnianie do API, a następnie wykonam komendę zwracania dostępnych produktów w bazie danych. Do umieszczonego linku spróbuje dokleić niechciane komendy SQL.



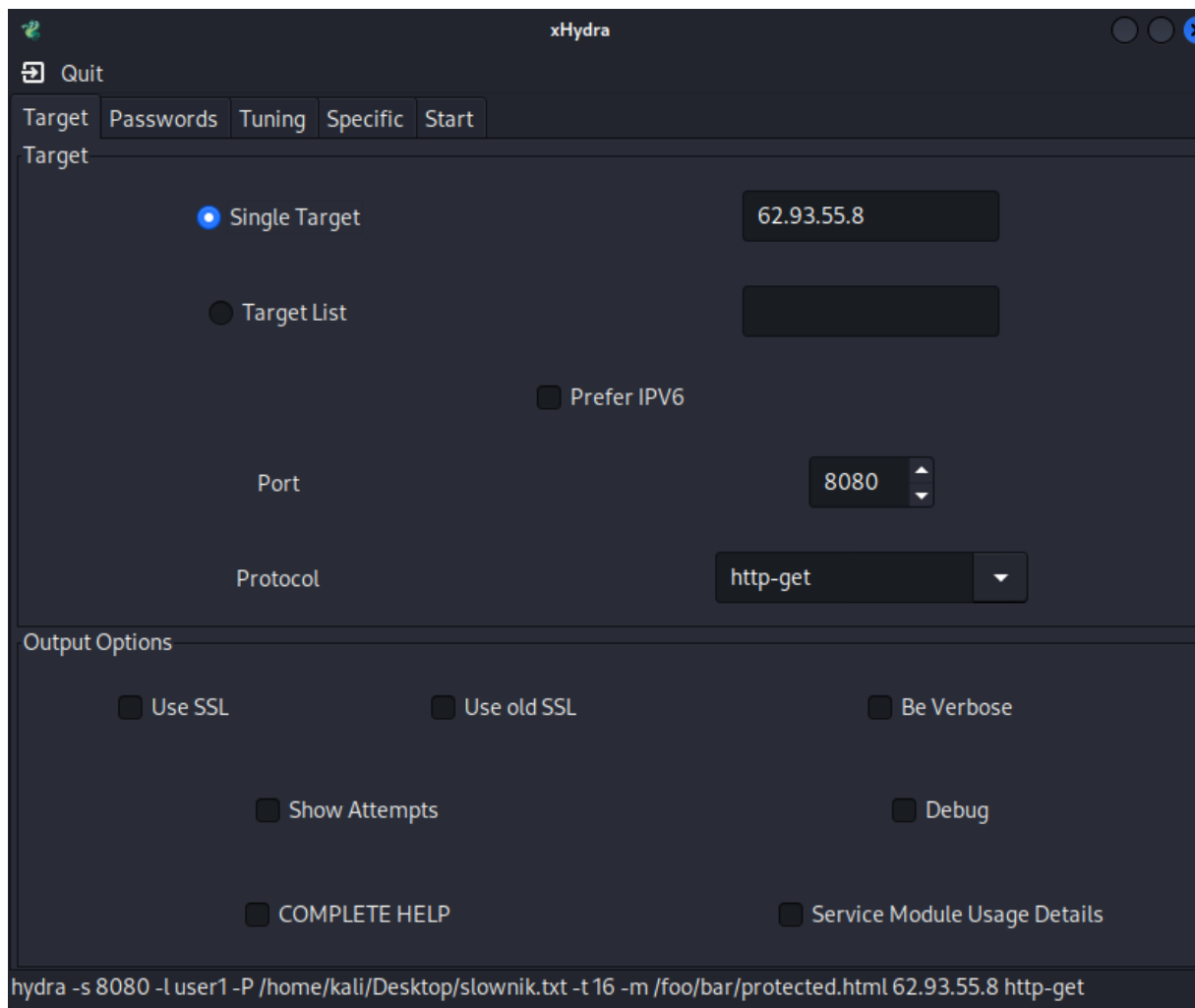
Po wykonaniu tego żądania API zwraca nam listę dostępnych produktów. Jeżeli dodamy ukośnik i wpisemy liczbę to API zwróci nam informacje o produkcie, który ma to ID, dlatego tutaj będziemy próbować dokleić niechciany kod SQL.



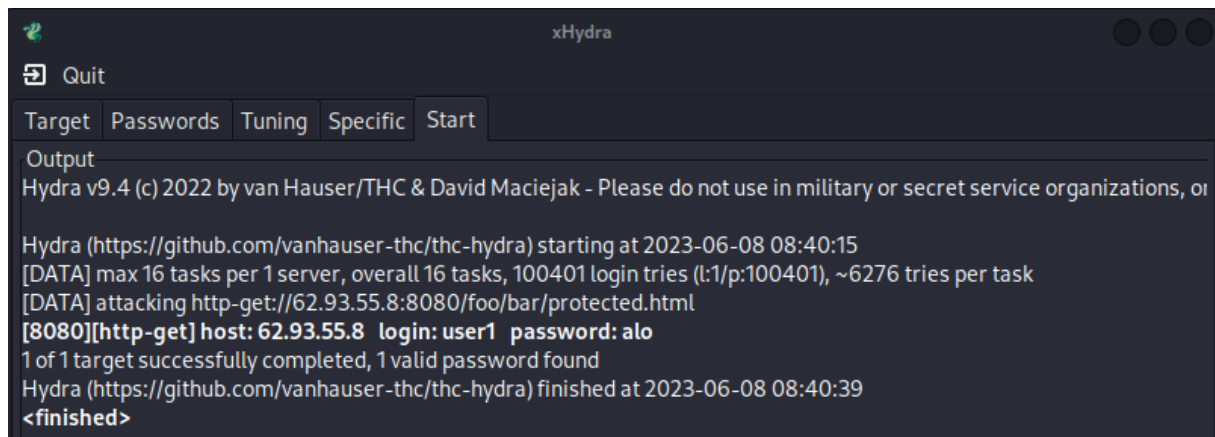
Pomimo wielu usilnych prób, za każdym razem API zwraca komunikat o błędnym żądaniu. Także wnioskując na tej podstawie API jest zabezpieczone przed podstawowymi próbami ataku SQL Injection.

5.3 Atak słownikowy

Kolejnym przeprowadzonym atakiem penetracyjnym był atak słownikowy. W tym celu wykorzystano system Kali Linux oraz zainstalowane na nim oprogramowanie o nazwie xHydra.



Po podaniu odpowiednich danych takich jak: IP, Port, Protokół oraz słownik haseł/użytkowników rozpoczęto test. W ramach testu manualnie wprowadzono odpowiedni login, natomiast poprawne hasło zostało umieszczone w ok. 2000 linii słownika z hasłami.

The screenshot shows the xHydra application window. At the top, there's a title bar with the name 'xHydra' and three window control buttons. Below the title bar is a menu bar with a 'Quit' option. Underneath the menu bar is a tabbed interface with five tabs: 'Target', 'Passwords', 'Tuning', 'Specific', and 'Start'. The 'Output' tab is active, displaying a log of the application's execution. The log text is as follows:

```
Hydra v9.4 (c) 2022 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or  
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2023-06-08 08:40:15  
[DATA] max 16 tasks per 1 server, overall 16 tasks, 100401 login tries (l:1/p:100401), ~6276 tries per task  
[DATA] attacking http-get://62.93.55.8:8080/foo/bar/protected.html  
[8080][http-get] host: 62.93.55.8 login: user1 password: alo  
1 of 1 target successfully completed, 1 valid password found  
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2023-06-08 08:40:39  
<finished>
```

Jak widać oprogramowanie poradziło sobie znakomicie i otrzymaliśmy poprawne dane uwierzytelniające.

6. Podsumowanie

Podsumowując aplikacja nie spełnia podstawowych założeń OWASP. Pierwszy z głównych problemów leży po stronie front-endu. Aplikacja posiada tzw. „Broken Access Control”, który został przedstawiony powyżej na ataku penetracyjnym. Użytkownik nie posiadając odpowiednich uprawnień jest w stanie zalogować się na stronę internetową modyfikując jedną zmienną w przeglądarce. Aplikacja nie korzysta z uwierzytelniania za pomocą utworzonego API. Dodatkowo kod nie obsługuje jawnie sprawdzania poprawności danych wprowadzanych przez użytkownika co może prowadzić do ataków XSS.

Po stronie back-endu brakuje kluczowych zabezpieczeń uniemożliwiających wykonywanie nieograniczonych prób logowania, co w konsekwencji prowadzi do zezwolenia na ataki typu słownikowe, bądź brute-force. Dodatkowo istnieje niebezpieczeństwo ataków SSRF (Server-Side Request Forgery). W jednej z klas dodany został nagłówek tak skonfigurowany, że zezwala na żądania między źródłami z dowolnej domeny.

Uwierzytelnianie zostało oparte na metodzie „basic access authentication”. Metoda ta wysyła dane uwierzytelniające zakodowane, lecz nie zaszyfrowane. Ponieważ aplikacja nie korzysta z HTTPS/TLS jest to bardzo wrażliwe miejsce, ponieważ dane są podatne na przechwycenie.

7. Bibliografia

- [1] <https://docs.spring.io/spring-framework/reference/>
- [2] <https://vuejs.org/guide/introduction.html>
- [3] <https://www.ovhcloud.com/pl/learn/what-is-rest-api/>
- [4] <https://owasp.org/www-project-top-ten/>
- [5] <https://security.snyk.io/>
- [6] <https://portswigger.net/web-security/csrf>
- [7] <https://portswigger.net/web-security/sql-injection>
- [8] <https://portswigger.net/web-security/cross-site-scripting>
- [9] <https://portswigger.net/web-security/ssrf>
- [10] <https://www.kali.org/>
- [11] <https://www.postman.com/>