Imię i nazwisko studenta: Zofia Zinkowska
Nr albumu: 180925
Poziom kształcenia: Studia pierwszego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Fizyka Techniczna
Specjalność: Informatyka Stosowana

PRACA DYPLOMOWA INŻYNIERSKA

Tytuł pracy w języku polskim: Aplikacja „ewidencja roślin”
Tytuł pracy w języku angielskim: The „plant register” application

Opiekun pracy: dr. inż. Bartosz Reichel

POLITECHNIKA GDAŃSKA | WYDZIAŁ FIZYKI TECHNICZNEJ I MATEMATYKI STOSOWANEJ

OŚWIADCZENIE: dotyczące pracy dyplomowej zatytułowanej:
Aplikacja „ewidencja roślin”

Imię i nazwisko: Zofia Zinkowska
Data i miejsce urodzenia: 20.07.2000, Słupsk
Nr albumu: 180925
Wydział: Wydział Fizyki Technicznej i Matematyki Stosowanej
Kierunek: fizyka techniczna
Poziom kształcenia: pierwszy
Forma studiów: stacjonarnie
Typ pracy: praca dyplomowa inżynierska

Świadomy(a) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz. U. z 2019 r. poz. 1231, z późn. zm.) i konsekwencji dyscyplinarnych określonych w ustawie z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (t.j. Dz. U. z 2020 r. poz. 85, z późn. zm.), 1. a także odpowiedzialności cywilnoprawnej oświadczam, że przedkładana praca dyplomowa została opracowana przeze mnie samodzielnie.

Niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem tytułu zawodowego.

Wszystkie informacje umieszczone w ww. pracy dyplomowej uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami zgodnie z art. 34 ustawy o prawie autorskim i prawach pokrewnych.

29.03.2022, Zofia Zinkowska
Data i podpis lub uwierzytelnienie w portalu uczelnianym Moja PG

*) Dokument został sporządzony w systemie teleinformatycznym, na podstawie §15 ust. 3b Rozporządzenia MNiSW z dnia 12 maja 2020 r. zmieniającego rozporządzenie w sprawie studiów (Dz.U. z 2020 r. poz. 853). Nie wymaga podpisu ani stempla

STRESZCZENIE

Praca dyplomowa przedstawia aplikację internetową, służącą do utrzymania / ewidencji roślin w ogrodzie botanicznym. W pierwszej oraz drugiej części pracy uwaga poświęcona jest wymaganiom jakim oprogramowanie musi sprostać. Trzecia część opisuje technologie oraz narzędzia programistyczne wykorzystane do implementacji. Kluczowym miejscem jest rozdział czwarty, który przedstawia projekt tworzonego oprogramowania. Końcowe rozdziały przedstawiają testy, podsumowanie, spis rysunków oraz bibliografię.

Słowa kluczowe:

Utrzymanie ewidencji roślin, ewidencja roślin, aplikacja internetowa, C#, TypeScript, ASP.NET Core, Vue.js.

Dziedzina nauki i techniki z wymogami OECD:

ABSTRACT

The thesis presents a web application for maintaining / registering plants in a botanical garden. In the first and second part of the work, the focus was on the requirements that the software should handle. The third part describes the technologies used for the implementation. The most significant part of the thesis is the fourth chapter, which presents the design of the software being developed. The final chapters present the tests, summary, list of figures, and a bibliography.

Keywords:

Plants maintennance registration, plants registration, web application, C#, TypeScript, ASP.NET Core, Vue.js.

Spis treści

WYKAZ WAŻNIEJSZYCH OZNACZEŃ I SKRÓTÓW

CSS - Cascading Style Sheets - Kaskadowe Arkusze Stylów.

ASP .NET Core

HTTP - HyperText Transfer Protocol

HTML - HyperText Markup Language

BGCI - Botanic Conservation International

EPCN - Exceptional Plant Conservation Network

1. Wstęp

Ze względu na rosnące zapotrzebowanie jakościowe, konieczny jest ciągły rozwój. Dlatego pojawiają się nowe rozwiązania w zakresie utrzymania / ewidencji roślin ułatwiające codzienną pracę.

1.1. Aplikacja internetowa

Aplikacja internetowa jest to program komputerowy pracujący na serwerze. Komunikuje się poprzez sieć komputerową z hostem użytkownika komputera, przy użyciu przeglądarki internetowej, będącej interaktywnym klientem aplikacji.

Aplikacje internetowe są jednym z najpopularniejszych typów programów komputerowych. Służą nie tylko do rozrywki, ale również jako systemy obsługi klientów. Aplikacje internetowe towarzyszą nam codziennie, od sterowania żarówką (smart home) do załatwiania spraw urzędowych (epuap).

Podstawową zaletą tworzenia oprogramowania tego typu jest fakt, że aplikacje internetowe są kompatybilne wieloplatformowo, gdyż nie zależą od systemu operacyjnego, a jedynym wymaganiem do sprawnego korzystania z programu jest posiadanie zainstalowanej przeglądarki internetowej z dostępem do internetu. Następną zaletą jest to, że użytkownik posiada dostęp do swoich danych nawet w momencie, kiedy następuje awaria urządzenia, ponieważ dane są gromadzone na serwerach dostawcy oprogramowania. Aplikacje webowe nie wymagają instalacji oraz aktualizacji (wdrażania i utrzymania) po stronie klienta, ponieważ zawsze serwowane są w najnowszej wersji serwera. Nie potrzebują weryfikacji / certyfikacji od poszczególnych producentów (np. Google Store, Apple Store).

Aplikacje internetowe oprócz zalet posiadają również wady. Jedną z nich jest całkowita zależność od serwera przez co użytkownik nie ma możliwości dokonywania operacji bez połączenia z siecią. Oprócz tego dostawca oprogramowania jest w stanie śledzić niemal każdy ruch użytkownika podczas jego korzystania z aplikacji. W dzisiejszych czasach mało, które aplikacje, nawet aplikacje nie internetowe, potrafią działać bez dostępu do internetu (np. aplikacja desktopowa nadal składowałaby dane na centralnym serwerze). W takiej architekturze utrata urządzenia byłaby jednoznaczna z utratą wszystkich danych.

1.2. Cel projektowania aplikacji

Na co dzień aplikacje internetowe ułatwiają nam zarządzanie pocztą, pozwalają robić zakupy i opłacać rachunki. Aplikacja ma na celu rozwiązanie problemu przestarzałego oprogramowania służącego do utrzymania / ewidencji roślin w ogrodzie botanicznym. Przedsięwzięcie dąży do wypełnienia brakujących potrzeb w systemie. Dzięki możliwości skanowania kodów etykiet oraz drukowaniu etykiet użytkownik będzie wiedział dokładnie jakie rośliny znajdują się w ogrodzie botanicznym. Nie tylko usprawni to pracę użytkownikowi, jak również użytkownik zyska na czasie, ponieważ automatyzacja procesów wcześniej

wykonywanych w sposób manualny, w znaczny sposób wykorzystywała czas osób administrujących.

W osiągnięciu celu pomogą nowe technologie programistyczne tj. C# - ASP.NET Core, TypeScript, Vue.js.

1.3. Zakres pracy

W ramach pracy inżynierskiej zaimplementowano:

Aplikację webową Ewidencja Roślin Botanicznych, której zadaniem jest wspomaganie ewidencjonowania roślin, która:

- Dodaje ona rośliny do wykazu roślin na podstawie zeskanowanego kodu kreskowego lub QR.
- Jest połączona bezpośrednio z urządzeniem
- Umożliwia drukowanie etykiet oraz wykazu roślin.
- Działa offline i online.

2. Specyfikacja wymagań

Rozdział przedstawia wymagania stawiane aplikacji internetowej służącej do utrzymania / ewidencji roślin. W rozdziale omawiane są kolejno wymagania biznesowe, funkcjonalne użytkownika oraz wymagania niefunkcjonalne użytkownika i systemowe.

2.1. Dziedzinowy słownik pojęć

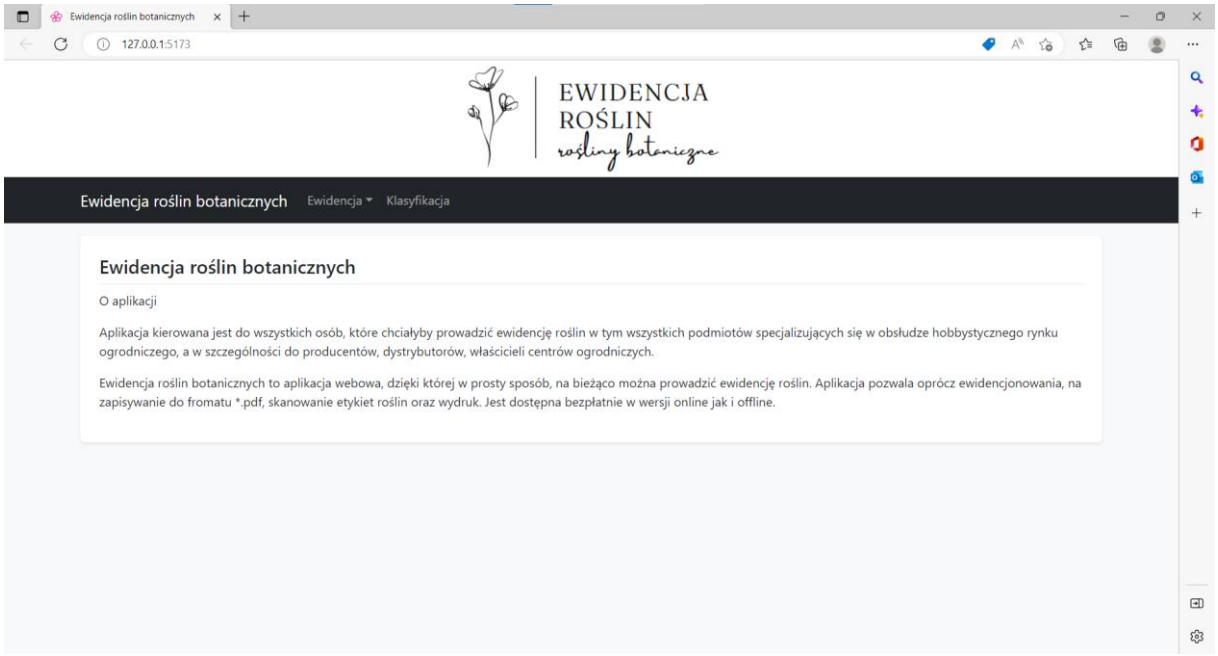
Użytkownik systemu – osoba korzystająca z aplikacji do utrzymania / ewidencji roślin.

Kaskadowe arkusze stylów [1]

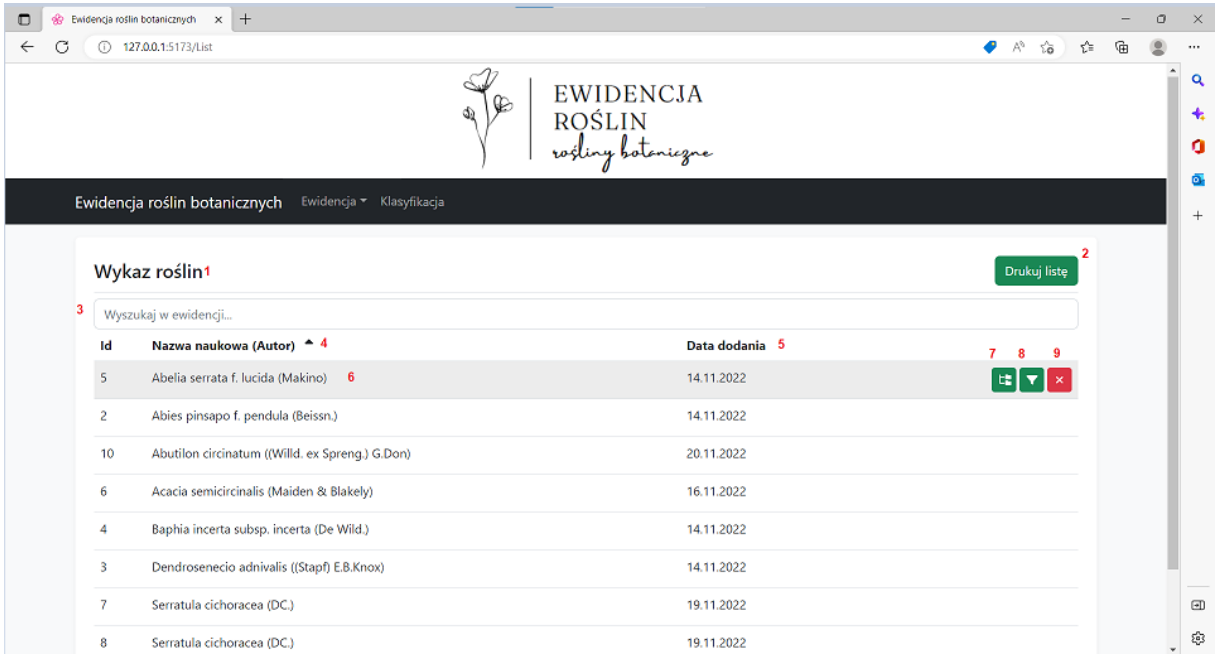
SPA (Single Page Application) – aplikacja jednostronicowa, która ładuje tylko jeden dokument internetowy, a następnie aktualizuje treść pojedynczego dokumentu za pomocą interfejsów.

2.2. Interfejs graficzny aplikacji

Interfejs graficzny został wykonany przy użyciu biblioteki Bootstrap-Vue 3. Używane są również style kaskadowe CSS.



Rysunek 1. Interfejs graficzny – strona główna



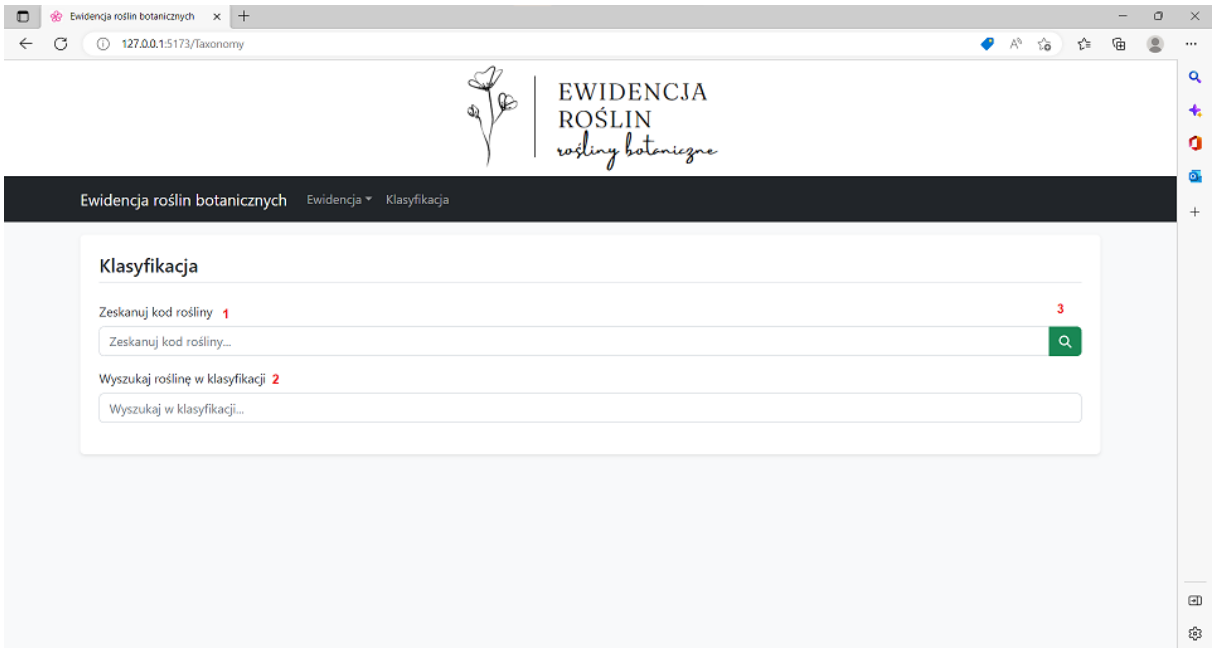
Rysunek 2. Interfejs graficzny – wykaz roślin

Aby móc dodać roślinę do ewidencji roślin, należy wejść w ewidencję i wybrać wykaz roślin.

1. Wykaz roślin w ewidencji.
2. Przycisk umożliwiający drukowanie oraz zapisywanie do .pdf listy ewidencji roślin.
3. Wyszukiwanie w ewidencji.
4. Sortowanie nazw rosnąco / malejąco.
5. Sortowanie dat najszybciej / najpóźniej dodane. Po najechaniu na datę mamy podgląd na konkretną datę dodania wraz z godziną.
6. Pozycja rośliny w klasyfikacji.

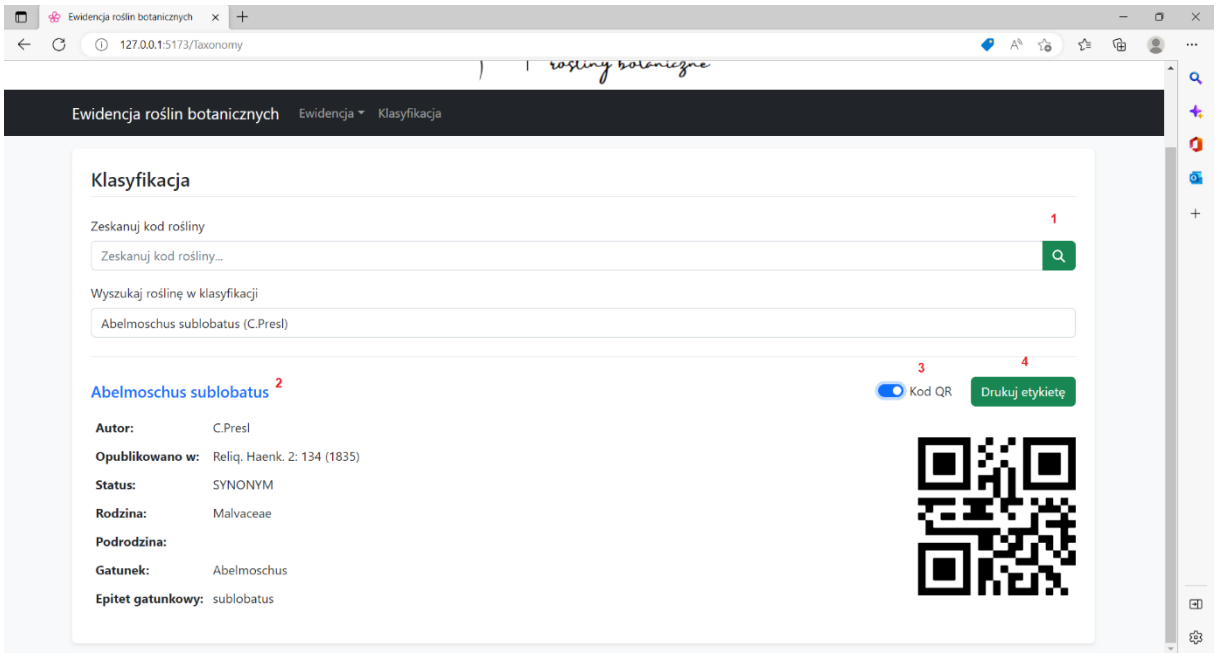
Po najechaniu na daną pozycję w wykazie roślin, pojawiają się po prawej stronie trzy przyciski:

7. Przycisk pokazujący klasyfikację rośliny.
8. Przycisk filtrujący według klasyfikacji.
9. Przycisk usuwający daną pozycję z ewidencji.



Rysunek 3. Interfejs graficzny – klasyfikacja

1. Pole tekstowe do skanowania / wprowadzania kodu rośliny.
2. Pole tekstowe do wyszukiwania pozycji w klasyfikacji roślin z podpowiadaniem od 3 pierwszych liter.
3. Przycisk do wyszukiwania kodu rośliny w ewidencji.

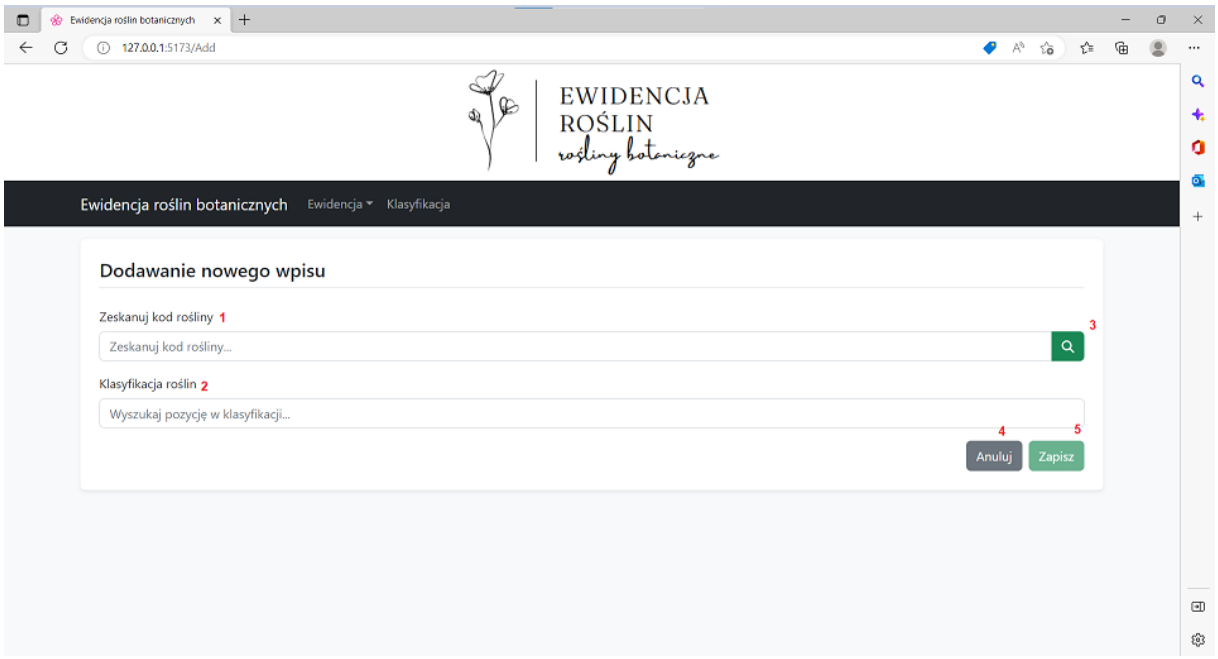


Rysunek 4. Interfejs graficzny – klasyfikacja

Aby móc zobaczyć klasyfikację roślin, należy wypełnić jedno z pól powyżej: zeskanuj kod rośliny lub wyszukaj w klasyfikacji.

1. Przycisk do wyszukiwania kodu rośliny w ewidencji.
2. Klasyfikacja roślin.

3. Pole wyboru między kodem kreskowym a kodem QR.
4. Przycisk umożliwiający drukowanie oraz zapisywanie do .pdf etykiety z klasyfikacją roślin, zawierającą kod QR.



Rysunek 5. Interfejs graficzny – dodawanie nowego wpisu do ewidencji roślin

Aby móc dodać roślinę do ewidencji roślin, należy wejść w ewidencję i wybrać dodaj.

1. Pole tekstowe do skanowania / wprowadzania kodu rośliny.
2. Pole tekstowe do wyszukiwania pozycji w klasyfikacji roślin z podpowiadaniem od 3 pierwszych liter.
3. Przycisk do wyszukiwania kodu rośliny w ewidencji i sprawdzenia czy kod jest już przypisany do innej rośliny.
4. Przycisk anulujący dodawanie.
5. Przycisk zapisujący dane do ewidencji roślin.

2.3. Wymagania funkcjonalne użytkownika

Aplikacja powinna posiadać takie funkcjonalności jak:

- Dodawanie, edycja, usuwanie wpisów.
- Listowanie wpisów wraz z wyszukiwaniem i sortowaniem.
- Podpowiadanie podczas wyszukiwania.
- Możliwość dodawania i skanowania za pomocą kodów QR.
- Możliwość drukowania etykiet.

2.4. Wymagania niefunkcjonalne użytkownika

Aplikacja powinna odpowiadać aktualnym standardom sieciowym i wytycznym dla dostępności treści internetowych – ma prawidłowo wyświetlać się w różnych przeglądarkach:

- Google Chrome
- Microsoft Edge
- Mozilla FireFox

Obsługa dużego pliku z klasyfikacją roślin.

2.5. Wymagania niefunkcjonalne systemowe

- Aplikacja powinna korzystać z dostępnych darmowych rozwiązań systemów informatycznych.
- Aplikacja powinna być wieloplatformowa.
- Interfejs aplikacji – system powinien być łatwy w obsłudze poprzez przejrzysty wygląd interfejsu, zrozumiały dla niedoświadczonego użytkownika. Aplikacja powinna być zaopatrzona w zrozumiałe menu oraz przyciski na stronie.
- Niezawodność aplikacji – przeprowadzanie testów poszczególnych modułów aplikacji. Optymalne dobranie architektury sprzętowo-programowej na jakiej będzie działać aplikacja oraz serwer bazodanowy.

3. Ogrody botaniczne i ochrona roślin

Ogrody botaniczne to instytucje posiadające udokumentowane kolekcje żywych roślin do celów badań naukowych, konserwacji, ekspozycji i edukacji. W 2018 roku BGCI zaktualizowało kryteria definiujące ogród botaniczny, aby położyć większy nacisk na ochronę rzadkich i zagrożonych roślin oraz zrównoważony rozwój. [2]

BGCI opracowało system akredytacji, aby rozróżnić ogrody i ogrody botaniczne. Program wykorzystuje zaktualizowaną definicję ogrodu botanicznego przyjętą przez BGCI, a także wyniki przeprowadzonego przez BGCI przeglądu technicznego definicji ogrodu botanicznego.

Ogrody botaniczne służą potrzebom społeczeństwa, będąc głównie ośrodkami turystycznymi, które każdego roku przyciągają turystów, wnoszą istotny wkład w gospodarkę lokalną i krajową. Zapewniają również wiele korzyści społeczeństwu, na przykład pozytywny wpływ na zdrowie psychiczne i fizyczne, szczególnie w środowisku miejskim, gdzie znajduje się większość ogrodów botanicznych. Poniżej zostaną przedstawione niektóre działania ogrodów botanicznych ważnych dla ochrony roślin:

- Umiejętność w zakresie ogrodnictwa i uprawy pozwalają ogrodom botanicznym uprawiać rośliny, które mogą zostać utracone w środowisku naturalnym. Żywe kolekcje i banki nasion chronią gatunki oraz umożliwiają odbudowę zdegradowanych siedlisk.
- Badania i rozwój w zakresie taksonomii i genetyki roślin, właściwości użytkowych roślin oraz informowanie o selekcji roślin odpornych na zdegradowane i zmieniające się środowiska - szczególnie ważne w obliczu zagrożenia, jakie stanowią zmiany klimatu.
- Łączenie roślin z dobrobytem ludzi, a także pomaganie w ochronie wiedzy lokalnej, aby zachęcać do zrównoważonego wykorzystania zasobów roślinnych z korzyścią dla wszystkich w ramach zrównoważonego rozwoju.
- Edukacja jest mocną stroną ogrodów botanicznych, która pozwala im komunikować znaczenie ochrony roślin, docierać do różnych odbiorców, a także informować, w jaki sposób można to osiągnąć.

3.1. Ochrona roślin

Różnorodność roślin leży u podstaw funkcjonowania wszystkich ekosystemów, które z kolei zapewniają podstawowe systemy wsparcia, od których zależy całe życie. Usługi świadczone przez ekosystemy obejmują sekwencję dwutlenku węgla, regulację klimatu oraz obieg składników odżywczych i zapylanie. Rośliny zapewniają nam wiele bezpośrednich korzyści, takich jak żywność, lekarstwa, ubrania, schronienie i surowce, z których wytwarza się niezliczone inne produkty, a także posiadają ważne wartości kulturowe i duchowe.

Rośliny są zatem niezbędnym zasobem dla ludzkiej egzystencji i wszyscy powinniśmy być świadomi, że na całym świecie rośliny są zagrożone, a wielu z nich grozi wyginięcie. Ich ochrona powinna być kluczowym elementem działań na rzecz ochrony różnorodności biologicznej.

In situ to po łacinie „na miejscu”, dlatego ochrona roślin in situ to ochrona różnorodności gatunkowej w normalnych i naturalnych siedliskach i ekosystemach. Dla porównania, ochrona ex situ koncentruje się na ochronie gatunków poprzez trzymanie ich w miejscach takich jak banki nasion lub żywe kolekcje. Ponieważ nasze naturalne systemy są narażone na wiele zagrożeń, ich ochrona nie jest łatwa i wymaga zastosowania wielu technik.

Obejmuje to rozwój, wyznaczanie i zarządzanie obszarami chronionymi, zwalczanie inwazyjnych gatunków obcych, odbudowę ekologiczną oraz współpracę ze społecznościami w celu promowania zrównoważonego użytkowania roślin i gospodarowania gruntami. Dlatego ważne jest, aby ochrona ex situ i in situ były zaprojektowane i praktykowane tak, aby wzajemnie się wzmacniać i uzupełniać. To połączone podejście znane jest jako „zintegrowana ochrona roślin”. Zintegrowana ochrona roślin może być wspierana przez badania, ogrodnictwo i edukację, które mogą ostatecznie zwiększyć powodzenie działań ochronnych.

Ogrody botaniczne są wyjątkowo dobrze przygotowane do wniesienia ważnego wkładu w zintegrowaną ochronę roślin, ponieważ mają dostęp do umiejętności i technik identyfikacji, uprawy i rozmnażania ogromnej różnorodności gatunków roślin. Ponadto posiadają ważne kolekcje żywych roślin, nasion i innej plazmy zarodkowej, które mogą mieć wielką wartość we wspieraniu wysiłków na rzecz ochrony zarówno in situ, jak i ex situ. [2]

3.2. System BGCI

BGCI (Botanic Conservation International) jest to największa na świecie sieć ochrony roślin.

- Wspiera ustalanie projektów działań ochronnych, które są potrzebne dla najbardziej zagrożonych gatunków roślin na świecie, poprzez wprowadzanie czerwonych list i badań ex situ.

  Badanie ex situ
- Wspiera ogrody botaniczne w ochronie drzew, wykorzystując ich rozległą wiedzę i doświadczenie, zarządzając funduszem ochrony drzew i projektami z partnerami na całym świecie, które realizują praktyczne działania ochronne dla najbardziej zagrożonych gatunków roślin.
- Zarządza Global Seed Conservation Challange, aby umieścić nasiona najbardziej zagrożonych gatunków roślin na świecie w kolekcjach banków nasion ex situ, aby działać jako ubezpieczenie przed wyginięciem w środowisku naturalnym.
- Wspiera ochronę wyjątkowych gatunków roślin, których nie można chronić w bankach roślin. Wymagają one żywych kolekcji, hodowli tkanek lub kriokonserwacji, aby je zachować. Pomaga zidentyfikować je w celu nadania im priorytetów, promując ich ochronę za pośrednictwem Sieci Ochrony Wyjątkowych Roślin (EPCN).
- Zarządza International Plant Sentinel Network, w ramach której ogrody botaniczne są w stanie monitorować rozprzestrzenianie się szkodników i chorób na rośliny z całego świata, które znajdują się w stanie środowiskowym ich ogrodów, działając jako sygnał wczesnego ostrzegania przed potencjalnymi zagrożeniami.

4. Przygotowanie do tworzenia aplikacji

Poniżej zostały przedstawione użyte technologie podczas implementacji aplikacji. Przy wyborze technologii autorce zależało na wykorzystaniu prostych i szybkich rozwiązań, darmowych oraz ogólnodostępnych. Dzięki odpowiedniemu doborze narzędzi rozwiązanie ewentualnych problemów było szybsze.

4.1. Wykorzystane narzędzia programistyczne

W tym rozdziale zostały przedstawione wykorzystane narzędzia programistyczne, które były wykorzystane podczas tworzenia projektu.

4.1.1. Git

Git jest rozproszonym systemem kontroli wersji. Został stworzony przez Linusa Torvaldsa w 2005 roku.

System z założenia miał podtrzymywać wielkie rozproszone projekty programistyczne, takie jak np. Linux.

Najważniejsze cechy:

- Programowanie rozproszone – system dostarcza programiście pracującemu nad projektem lokalną kopię rozwoju aplikacji.
- Zgodność z protokołami i systemami – repozytoria mogą być publikowane za pomocą HTTP, FTP, etc.
- Programowanie nieliniowe – wsparcie dla nieliniowego wytwarzania oprogramowania

4.1.2. Bootstrap

Bootstrap – biblioteka użyta w projekcie. "Zawiera zestaw przydatnych narzędzi ułatwiających tworzenie interfejsu graficznego stron oraz aplikacji internetowych." (“Bootstrap (framework) – Wikipedia, wolna encyklopedia”) Oferuje domyślny zestaw tematyczny obejmujący kolorystykę i stylistykę wyglądu strony, jednocześnie pozwalając na łatwą modyfikację tego zestawu tematycznego, bez ingerencji w kod poszczególnych elementów. [3] [4] [5]

W swojej aplikacji użyłam wersji Bootstrap 5.2.0.

4.1.3. Baza danych LiteDB

LiteDB to szybka, prosta, typu open source z zerową konfiguracją, wbudowana baza danych NoSQL dla .NET w pełni napisana w języku C#. LiteDB organizuje dokumenty w magazynach dokumentów znanych jako kolekcje. Co oznacza, że każda kolekcja jest identyfikowana przez unikatową nazwę i zawiera jeden lub więcej dokumentów, które mają ten sam schemat. Ponieważ LiteDB jest bazą danych bez serwera, nie musimy instalować jej w swoim systemie. Wystarczy dodać odwołanie do pliku LiteDB.dll w swoim projekcie. Alternatywnie można zainstalować LiteDB za pomocą Menadżera pakietów NuGet w programie Visual Studio. [6]

W swojej aplikacji użyłam wersji LiteDB 5.0.12.

4.1.4. ASP .NET Core

ASP .NET Core (w swojej aplikacji użyłam wersji ASP .NET Core 6.0) jest to platforma o otwartym kodzie (Open Source) do tworzenia aplikacji internetowych, stworzona przez firmę Microsoft. Umożliwia tworzenie zarówno aplikacji typowo backendowych (Web API), jak i w pełni funkcjonalnych aplikacji internetowych (backend + frontend). W pracy użyłam tylko funkcjonalności Web API, ze względu na architekturę Single Page Application (SPA), gdzie część frontendowa jest napisana we frameworku Vue.js [7] [8]

4.1.5. Visual Studio

Visual Studio jest to bogate w funkcje zintegrowane środowisko programistyczne (IDE), stworzone przez firmę Microsoft. Środowisko to jest dedykowanym rozwiązaniem używanym przy tworzeniu aplikacji ASP .NET Core, w którym została stworzona część backendowa aplikacji. Dodatkowo natywnie współpracuje z systemem kontrolowania wersji Git. [9]

W swojej aplikacji użyłam wersji Microsoft Visual Studio Community 2022 (64-bit) 17.3.4.

4.1.6. Visual Studio Code

Visual Studio Code to prosty edytor kodu źródłowego, stworzony przez firmę Microsoft. Program jest bogaty w rozszerzenia dla różnych języków programowania i platform programistycznych, z czego większość jest darmowa i posiada otwarty kod (Open Source). Między innymi posiada wtyczki dla użytych w aplikacji technologii i języków takich jak Vue.js, Node.js, TypeScript, Bootstrap. Natywnie integruje się z systemem wersjonowania Git, co znacząco ułatwia tworzenie oprogramowania przy użyciu tego narzędzia. [10]

Przy tworzeniu swojej aplikacji użyłam wersji Visual Studio Code 1.72.2.

4.1.7. Cypress

Cypress jest to narzędzie do tworzenia i uruchamiania testów automatycznych typu end-to-end oparty na platformie Node.js. Jest to zestaw mniejszych narzędzi, które pomogą uruchomić test, scenariusz testowy, w przeglądarce (narzędzie wspiera wiele przeglądarek, np. Edge, Chrome, FireFox) oraz za pomocą specjalnego API, które Cypress wystawia. Cały kod testów piszemy w języku JavaScript lub TypeScript. Z narzędzia możemy korzystać praktycznie od razu po instalacji, nie wymaga praktycznie żadnej konfiguracji. [11]

W swojej aplikacji użyłam wersji Cypress 10.11.0.

4.1.8. Użyte biblioteki

- Axios to prosty klient HTTP oparty na Promise (jest to typ klasa) obiektu w JavaScript) [12] przeglądarki i node.js Axios zapewnia prostą w użyciu bibliotekę w małym pakiecie z bardzo rozszerzalnym interfejsem. [13]

- Vue3-simple-typeahead prosty i lekki komponent Vue3 do wpisywania / autouzupełniania, który wyświetla sugerowaną listę elementów podczas wpisywania przez użytkownika. [14]
- Moment.js biblioteka ułatwiająca pracę z datami i godzinami w JavaScript / TypeScript, między innymi formatowanie czy obsługę stref czasowych. [15]
- Qrcode.vue biblioteka umożliwiająca generowanie kodów QR. [16]
- Vue3-print-nb biblioteka umożliwiająca drukowanie strony HTML. [17]
- Vue3-barcode biblioteka umożliwiająca generowanie kodów kreskowych. [18]

5. Projekt aplikacji

5.1. Implementacja frontendu

5.1.1. Generowanie kodów QR oraz kreskowych

```
67      <div class="ms-auto mt-auto mb-auto">
68          <qrcode-vue v-if="printQr" :value="details.id" level="L" :size="200"></qrcode-vue>
69          <barcode v-else :value="details.id" format="CODE128" :width="2.5" :height="200"></barcode>
70      </div>
```

Rysunek 6. Fragment pliku TaxonomyView.vue

Na Rysunku 6. Zostały przedstawione dwie kontrolki do wyświetlania kodu QR oraz kreskowego. Kontrolki pochodzą z dwóch różnych bibliotek (qrcode-vue i vue3-barcode) i odpowiadają za generowanie kodów w oparciu o identyfikator z klasyfikacji.

5.1.2. Wczytywanie kodów QR oraz kreskowych

Aplikacja wspiera skanery kodów QR i kreskowych pod warunkiem, że działają one w trybie emulacji klawiatury. Dzięki temu użytkownik posiada możliwość wpisania kodu ręcznie z klawiatury (np. nazwa skrótowa) oraz używania go zamiennie z kodami QR / kreskowymi (taka możliwość istnieje jedynie przy dodawaniu rośliny do wykazu lub wyszukiwaniu klasyfikacji).

5.2. Implementacja backendu

```
21
22      //Here we add all the services to the dependency injection container
        0 references
23      public void ConfigureServices(IServiceCollection services)
24      {
25          services.AddControllers();
26
27          services.AddSingleton<ILiteDatabase>(x => new LiteDatabase(DbPath));
28
29          services.AddSingleton<ITaxonomyProvider,TaxonomyProvider>();
30
31          services.AddScoped<IRegisterRepository,RegisterRepository>();
32
33          services.AddHostedService<CachePrimer>();
34
35          services.AddHostedService<DBInitializer>();
36      }
```

Rysunek 7. Fragment pliku Startup.cs

Na Rysunku 7. zostało przedstawione dodanie wszystkich serwisów do kontenera DI (ang. dependency injection). Ten fragment kodu definiuje cykl życia poszczególnych serwisów: singleton (jedna instancja na całą aplikację), scoped (jedna instancja na jedno zapytanie http).

5.2.1. Dependency Injection

Dependency Injection (wstrzykiwanie zależności) to wzorzec projektowy, w którym obiekt lub funkcja otrzymuje inne obiekty lub funkcje, od których jest zależny. Forma odwrócenia kontroli, wstrzyknięcia zależności ma na celu oddzielenie problemów związanych z konstruowaniem obiektów i używaniem ich, co prowadzi do niezobowiązująco powiązanych programów. [19] Wzorzec zapewnia, że obiekt lub funkcja, która chce skorzystać z danej usługi, nie musi wiedzieć, jak te usługi skonstruować. Zamiast tego odbierający „klient" (obiekt lub funkcja) otrzymuje swoje zależności za pomocą wewnętrznego kodu. [20] Wstrzykiwanie zależności pomaga, ujawniając niejawne zależności oraz pomaga rozwiązywać problemy, takie jak:

- W jaki sposób klasa może być niezależna od tworzenia obiektów, od których jest zależna.
- W jaki sposób aplikacja i używane przez nią obiekty mogą obsługiwać różne konfiguracje.
- Jak można zmienić zachowanie fragmentu kodu bez jego bezpośredniej edycji.

Wstrzykiwanie zależności obejmuje cztery role: usługi, klientów, interfejsy i kontenery:

Usługi i klienci – usługa to dowolna klasa zawierająca użyteczną funkcjonalność. Natomiast klientem jest dowolna klasa korzystająca z usług.

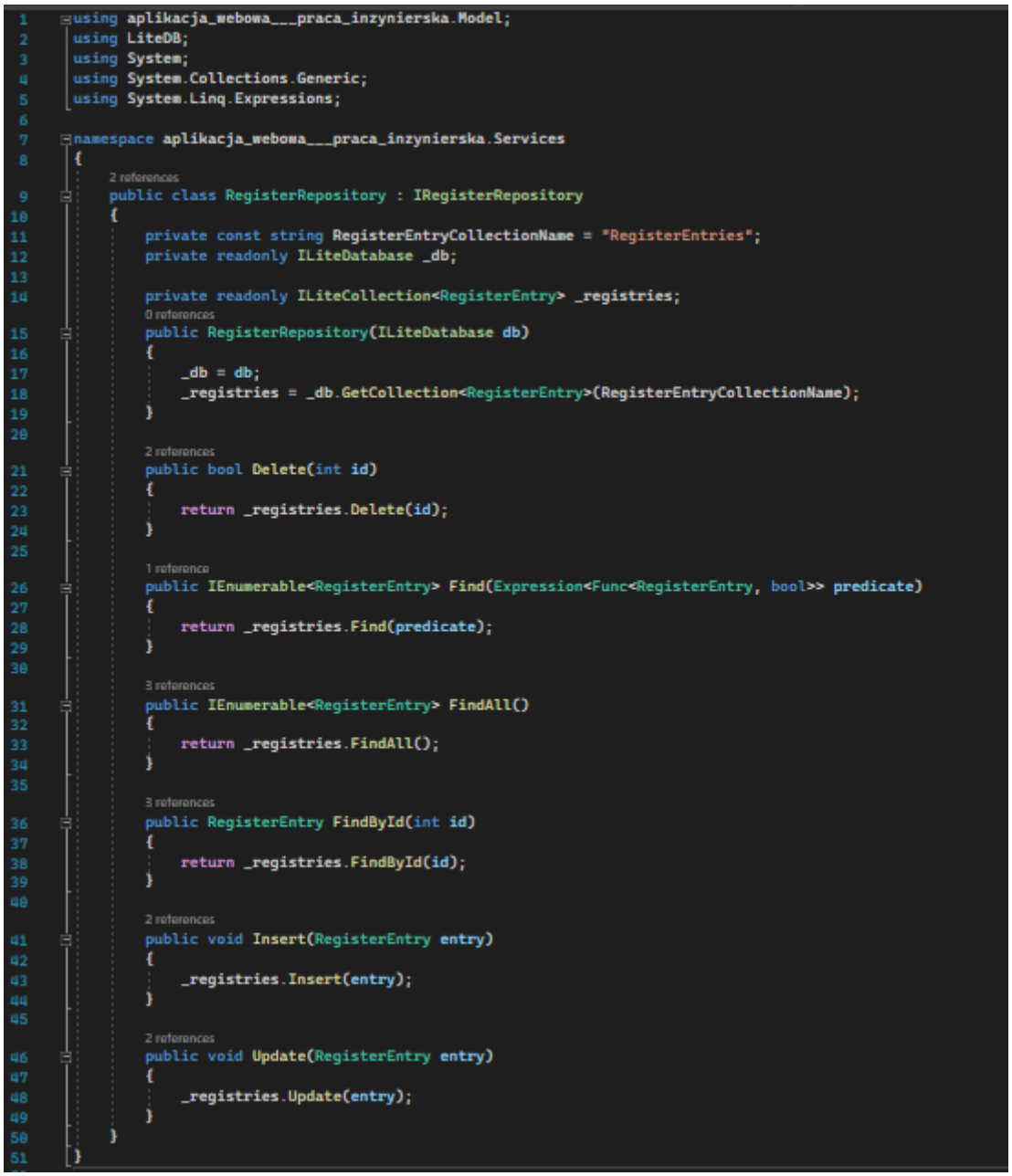
Interfejs – klienci nie powinni wiedzieć w jaki sposób implementowane są ich zależności, a jedynie nazwy i API. Ignorując szczegóły implementacji, klienci nie muszą zmieniać się, gdy zmieniają się ich zależności.

Kontenery – rolą ich jest konstruowanie i łączenie złożonych grafów obiektowych, gdzie obiektami mogą być zarówno klienci, jak i usługi. Sam kontener może być wieloma współpracującymi ze sobą obiektami, ale nie może być klientem, ponieważ stworzyłoby to cykliczną zależność. [21]

```
1 reference
private IEnumerable<TaxonomyItem> LoadTaxonomy()
{
    //Loading the file classification.txt
    using var input = new StreamReader(File.OpenRead("Content/classification.txt"));

    //Skip first line with headings
    input.ReadLine();

    var items = new List<TaxonomyItem>();
    while (!input.EndOfStream)
    {
        var line = input.ReadLine();
        if (string.IsNullOrEmpty(line))continue;
        var values = line.Split('\t');

        //Fields indexes are documented in the meta.xml file
        var item = new TaxonomyItem
        {
            TaxonomyID = values[0],
            ScientificNameID = values[1],
            LocalID = values[2],
            ScientificName = values[3],
            TaxonRank = values[4],
            ScientificNameAuthor = values[6],
            Family = values[7],
            Subfamily = values[8],
            Genus = values[11],
            SpecificEpithet = values[13],
            NamePublishedIn = values[17],
            TaxonomicStatus = values[18],
            AcceptedNameUsageID = values[19],
        };

        items.Add(item);
    }
    return items;
}
```

Rysunek 8. Fragment pliku Services/TaxonomyProvider.cs

Na Rysunku 8. przedstawiono fragment kodu, w którym wczytujemy plik classification.txt zawierający klasyfikację roślin zgromadzoną w kompendium World Flora Online [22]. Plik jest tak obszerny, że w celu optymalizacji plik jest wczytywany tylko raz na początku uruchomienia aplikacji (cykl życia singleton).



```
using aplikacja_webowa___praca_inzynierska.Model;
using LiteDB;
using System;
using System.Collections.Generic;
using System.Linq.Expressions;

namespace aplikacja_webowa___praca_inzynierska.Services
{
    2 references
    public class RegisterRepository : IRegisterRepository
    {
        private const string RegisterEntryCollectionName = "RegisterEntries";
        private readonly ILiteDatabase _db;

        private readonly ILiteCollection<RegisterEntry> _registries;
        0 references
        public RegisterRepository(ILiteDatabase db)
        {
            _db = db;
            _registries = _db.GetCollection<RegisterEntry>(RegisterEntryCollectionName);
        }

        2 references
        public bool Delete(int id)
        {
            return _registries.Delete(id);
        }

        1 reference
        public IEnumerable<RegisterEntry> Find(Expression<Func<RegisterEntry, bool>> predicate)
        {
            return _registries.Find(predicate);
        }

        3 references
        public IEnumerable<RegisterEntry> FindAll()
        {
            return _registries.FindAll();
        }

        3 references
        public RegisterEntry FindById(int id)
        {
            return _registries.FindById(id);
        }

        2 references
        public void Insert(RegisterEntry entry)
        {
            _registries.Insert(entry);
        }

        2 references
        public void Update(RegisterEntry entry)
        {
            _registries.Update(entry);
        }
    }
}
```

Rysunek 9. Plik Services/RegisterRepository.cs

Na Rysunku 9. przedstawiono implementację serwisu RegisterRepository. Serwis ten służy do ukrycia szczegółów implementacyjnych związanych z bazą LiteDB poprzez wyabstrahowanie metod CRUD. Jest to typowy przykład zastosowania wzorca projektowego „repozytorium”.

5.3. Instalacja

Aplikacja została zaprojektowana do działania w kontenerach pod kontrolą Dockera. Składa się z trzech podstawowych elementów: kontenera hostującego backend, kontenera hostującego frontend oraz wolumenu przechowującego dane użytkownika.

5.3.1. Docker

Docker jest to narzędzie służące do uruchamiania aplikacji w kontenerach (jest to jedna z implementacji systemów kontenerów). Architektura kontenerowa ma jedną bardzo istotną cechę – wszystko potrzebne do działania aplikacji jest zawarte w kontenerze. Z jednej strony, z punktu widzenia użytkownika, jedyne co jest potrzebne do uruchomienia aplikacji, to Docker, bez potrzeby instalowania innych zależności. Z drugiej strony, z punktu widzenia twórcy aplikacji, można ją uruchomić na dowolnym systemie w ujednolicony sposób (pod warunkiem, że jest w nim Docker) – środowisko uruchomieniowe kontenera jest zawsze takie samo i jest całkowicie odizolowane od środowiska maszyny hosta. [23]

5.3.2. Kontener

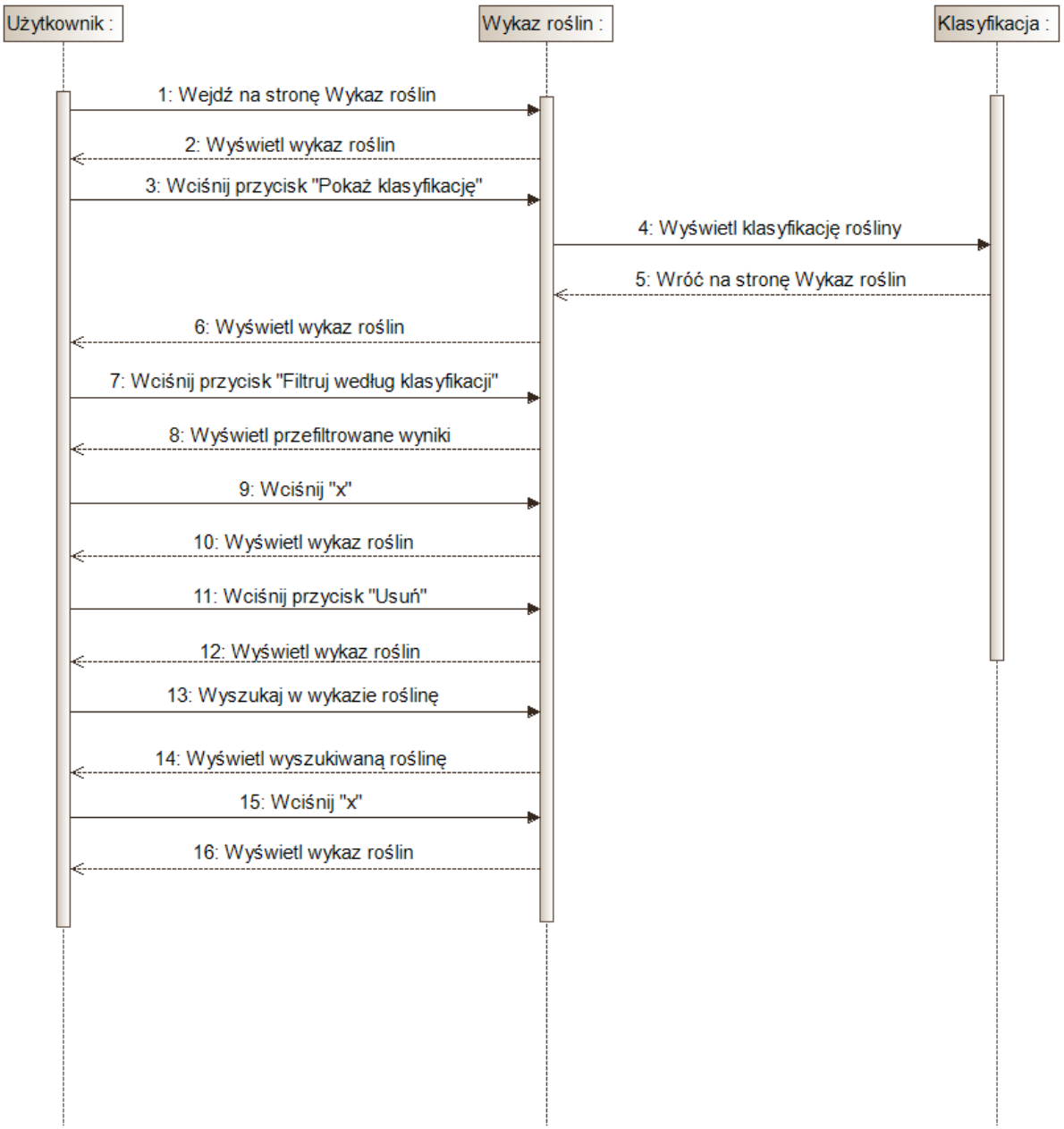
Kontener to pewien proces włączony w trybie izolacji, dzięki temu kontenery działają bezproblemowo i nie wchodzą w konflikty z innymi kontenerami. Dają programistom dużą swobodę, umożliwiając spakowanie aplikacji ze wszystkim zależnościami w łatwy do przenoszenia pakiet.

5.3.3. Dane a kontener

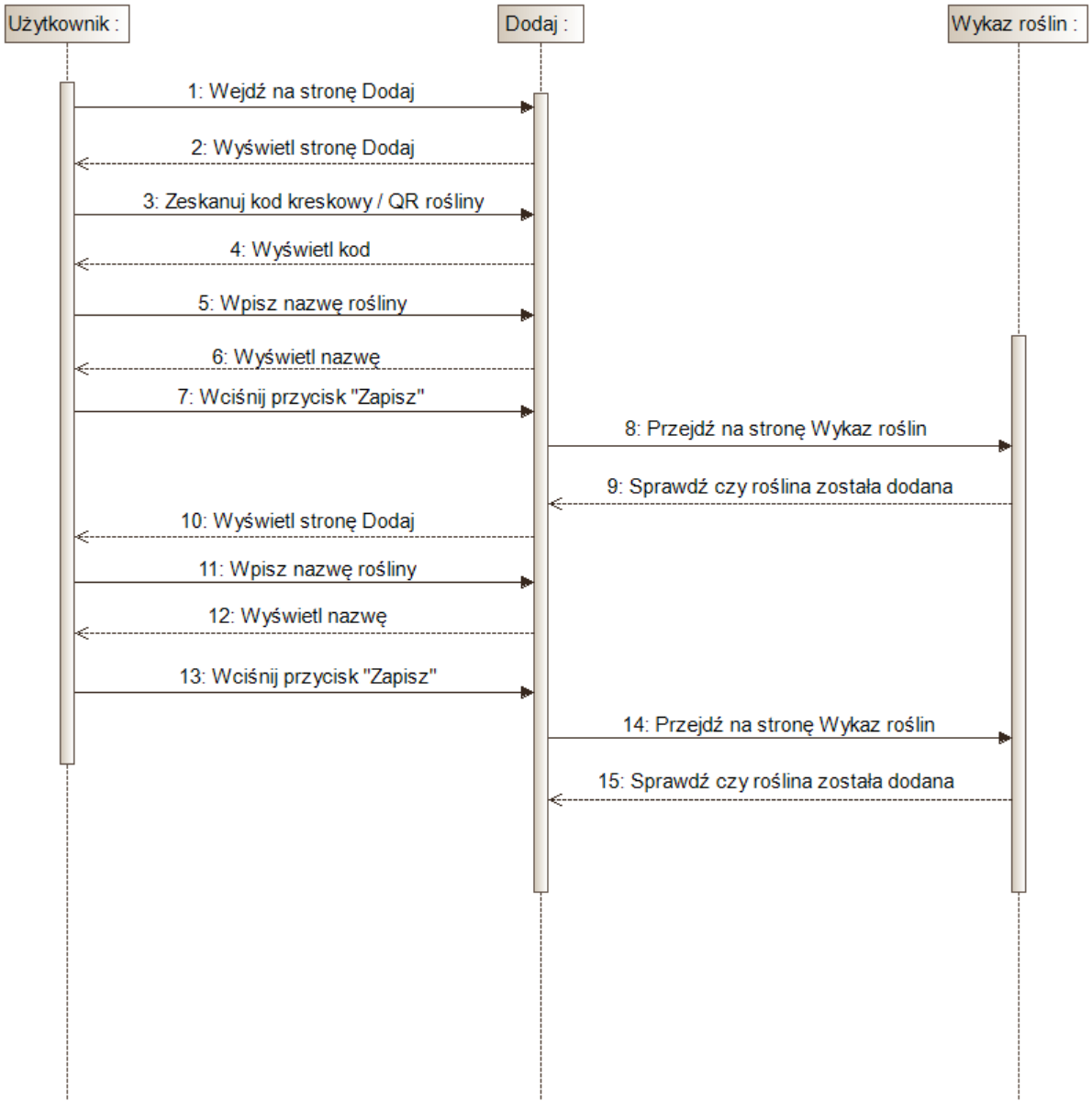
Z założenia kontenery nie są obiektami trwałymi. Na przykład przy aktualizacji wersji aplikacji konieczne jest utworzenie nowego kontenera na podstawie nowej wersji obrazu. W związku z tym trzymanie danych wewnątrz kontenera nie jest akceptowalne, ponieważ dane byłby usuwane przy aktualizacji aplikacji. Dlatego w mojej aplikacji dane trzymane są w dockerowym wolumenie, który w odróżnieniu od kontenera jest obiektem trwałym, dzięki czemu dane są bezpieczne podczas aktualizacji aplikacji.

5.4. Diagram

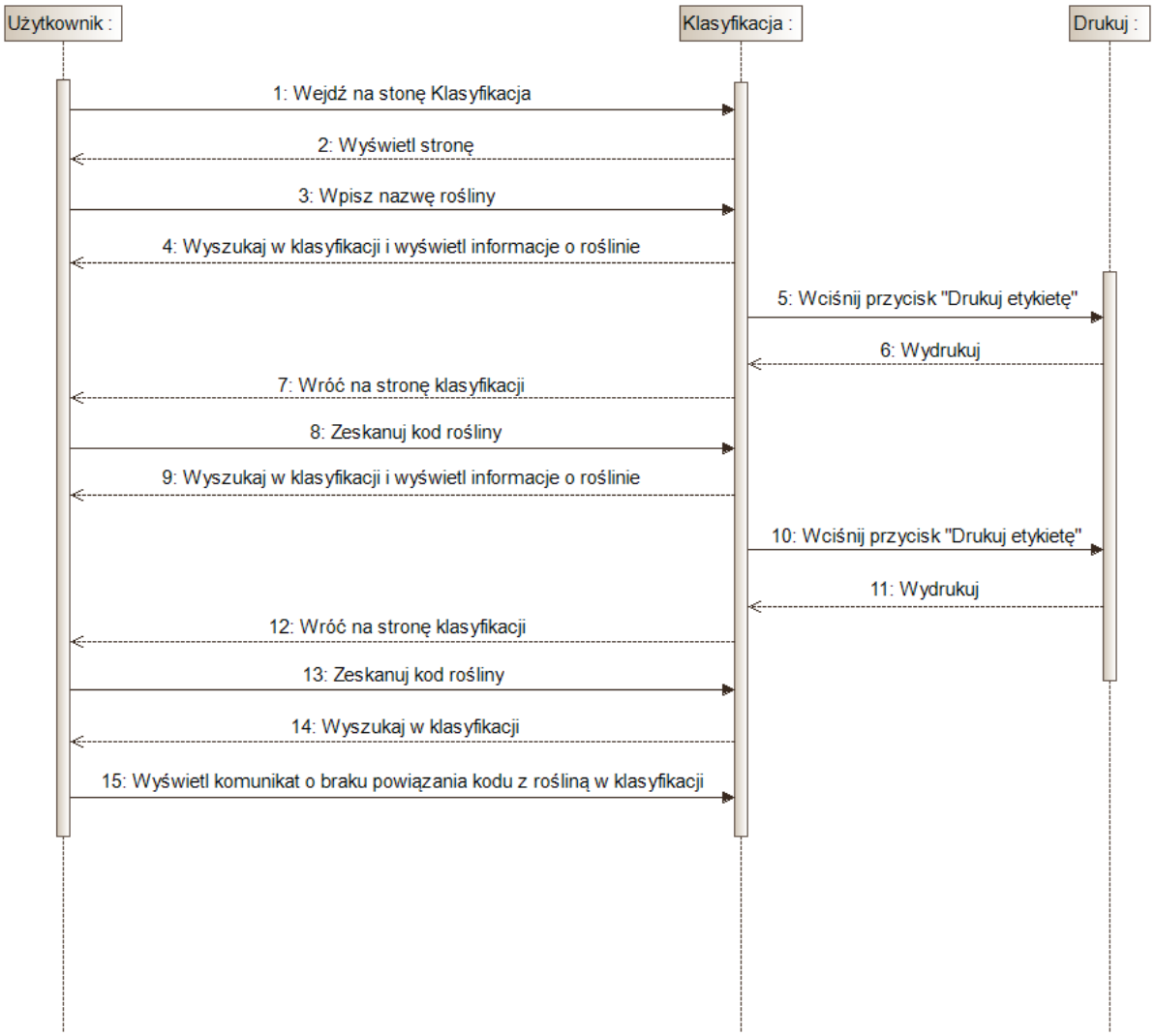
5.4.1. Diagramy sekwencji



Rysunek 10. Wykaz roślin - diagram sekwencji

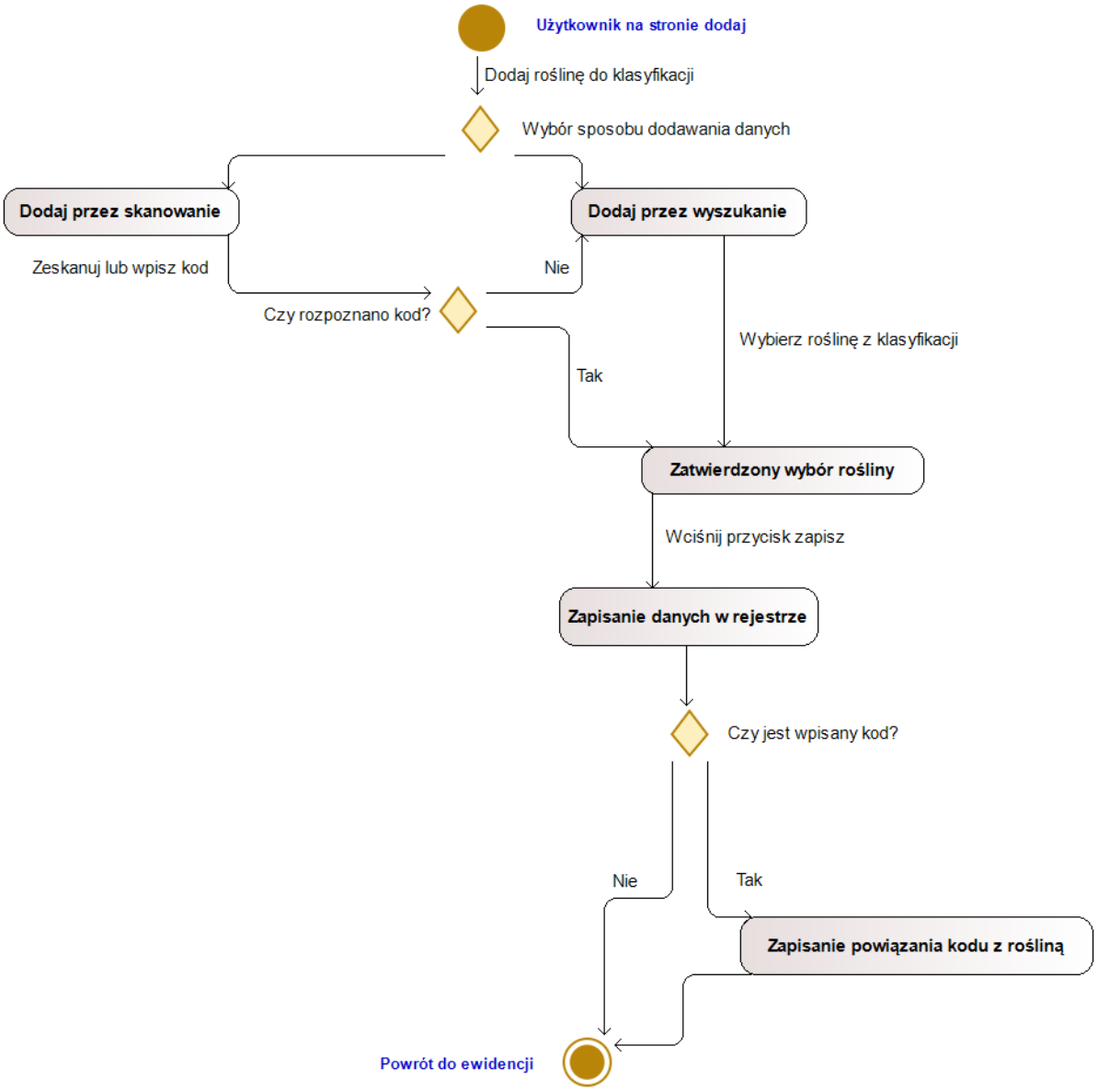


Rysunek 11. Dodaj - diagram sekwencji

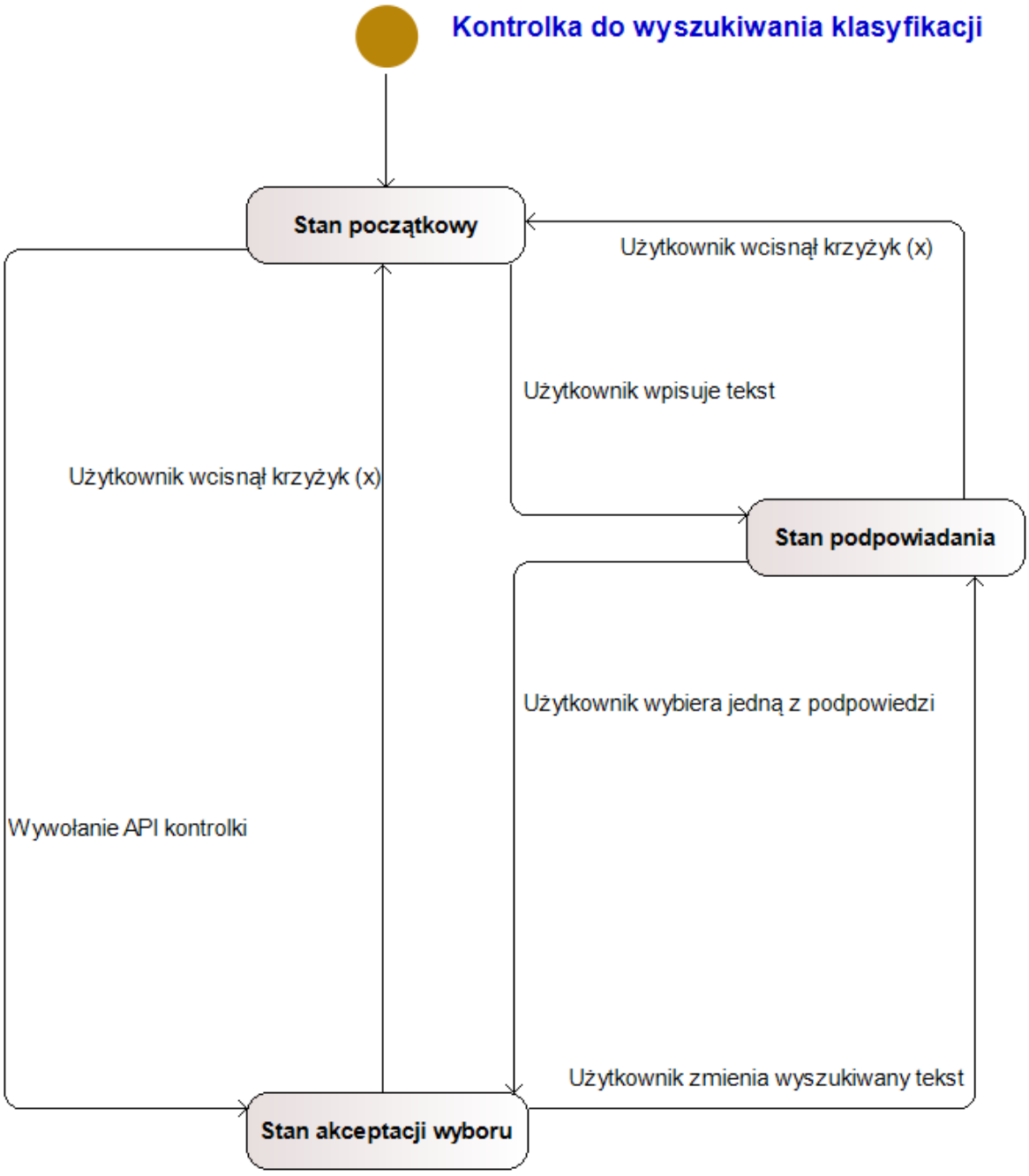


Rysunek 12. Klasyfikacja - diagram sekwencji

5.4.2. Diagramy maszyny stanowej



Rysunek 13. Dodanie rośliny do ewidencji - Diagram maszyny stanowej



Rysunek 14. Kontrolka do wyszukiwania klasyfikacji - Diagram maszyny stanowej

6. Testy

Aby aplikacja mogła trafić w ręce użytkowników, powinna być dobrze przetestowana. Osobami odpowiedzialnymi za napisanie przypadków testowych oraz ich przeprowadzenie są testerzy oprogramowania. Przypadek testowy powinien posiadać zdefiniowany cel, określone warunki wstępne oraz kroki, na test składa się opis oczekiwanego rezultatu i poszczególnych kroków. Bardzo często tworzone są również raporty testowe, na które składa się wiele przypadków testowych.

Testy możemy podzielić na kilka rodzajów. Podstawowe testy to testy jednostkowe oraz integracyjne, możemy również wyróżnić testy systemowe oraz akceptacyjne.

- Testy jednostkowe – testują wyłącznie jedną funkcję / metodę.
- Testy integracyjne – testują wiele modułów na raz i sprawdzają, czy dobrze ze sobą współpracują.
- Testy systemowe – testują, czy wszystkie elementy systemu działają zgodnie z założeniami.
- Testy akceptacyjne – sprawdzają, czy system spełnia wszystkie stawiane przez klienta wymagania biznesowe.

Testy powinny sprawdzać jak najwięcej możliwych przypadków, biorąc pod uwagę koszt utrzymania testów. Jeżeli wszystkie testy przechodzą pozytywnie, możemy mieć wystarczającą pewność, że dana funkcjonalność działa poprawnie. Jeśli jednak w aplikacji pojawi się błąd, z łatwością możemy znaleźć źródło problemu.

Testy end-to-end są bardziej kosztowne, trudniejsze w utrzymaniu, ponieważ muszą wykonać kilka kroków, aby osiągnąć ten stan aplikacji, który testuje. Jednak dzięki temu, że test będzie działał w przeglądarce będziemy mogli sprawdzić wszystkie warstwy aplikacji, zaczynając od bazy danych przez backend, aż do frontendu.

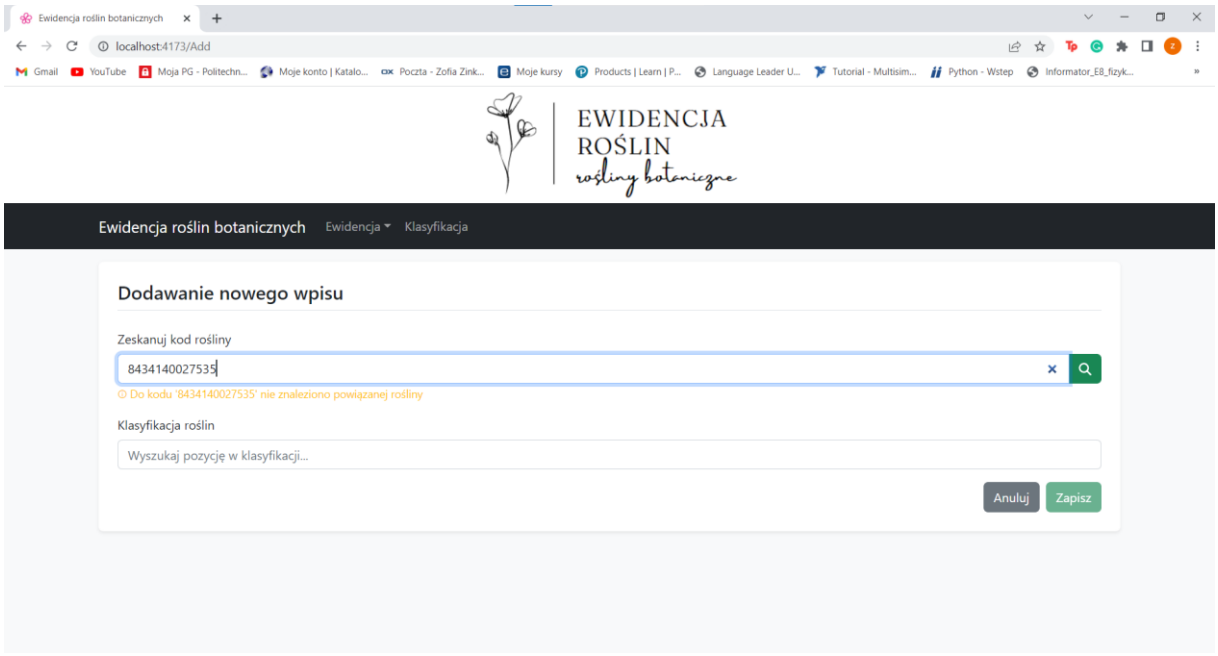
W projekcie przetestowano aplikację do ewidencjonowania roślin botanicznych na dwa sposoby:

- Testowanie manualne – wykonanie wybranych operacji w aplikacji.
- Testowanie automatyczne – do aplikacji internetowej zostały stworzone testy systemowe typu end-to-end w narzędziu Cypress, użyte funkcje oraz testy znajdują się w plikach folderu cypress.

6.1. Testowanie manualne

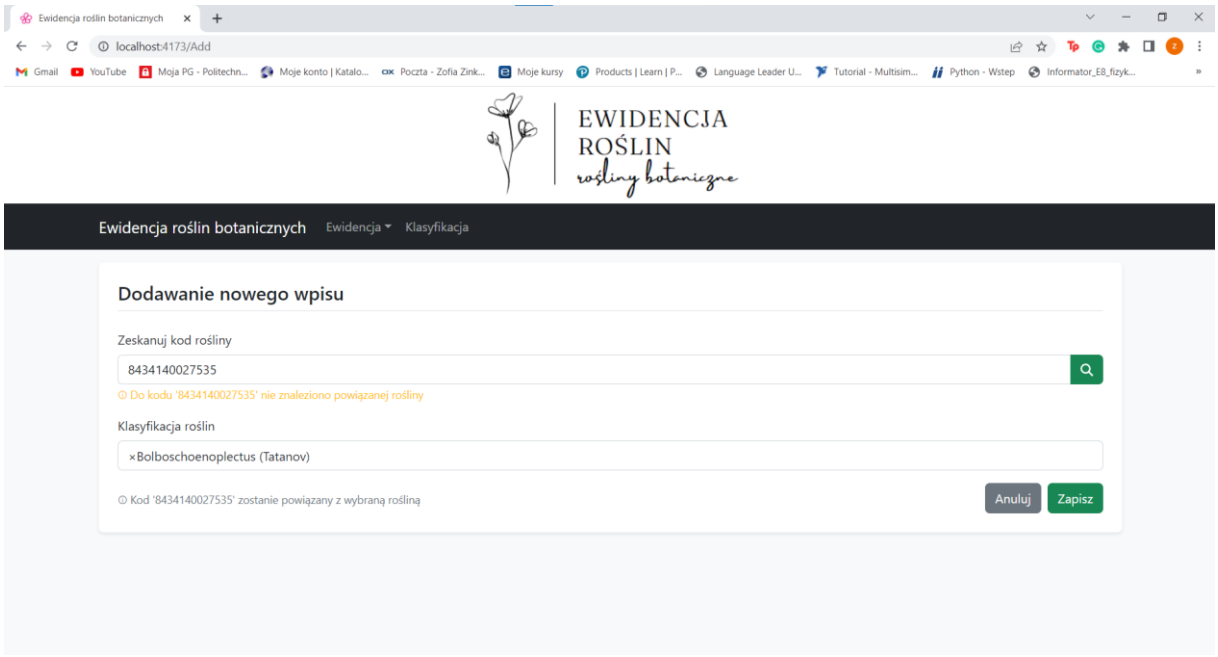
6.1.1. Dodawanie rośliny do ewidencji

1. Podczas dodania rośliny do ewidencji zeskanowano kod kreskowy i wyszukano. Do zeskanowanego kodu nie znaleziono powiązania w wykazie roślin – pojawił się komunikat z informacją. Dodatkowo, niebieskie obramowanie oznacza, że kontrola ma Focus, czyli kursor jest w niej ustawiony i będzie przechwytywać input (dane wejściowe) z klawiatury (Rysunek 15).



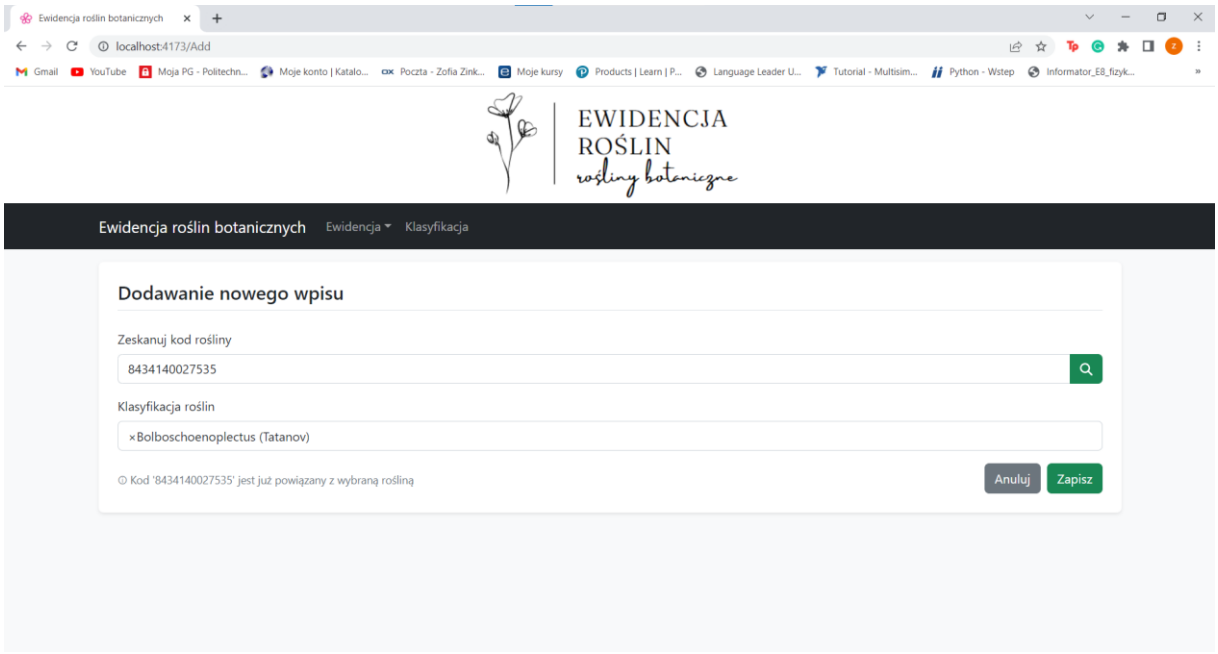
Rysunek 15. Dodanie kodu rośliny

2. Podczas wpisania kodu (użytego pierwszy raz) oraz wyszukania rośliny w klasyfikacji pojawia się komunikat z informacją o powiązaniu kodu z wybraną rośliną.



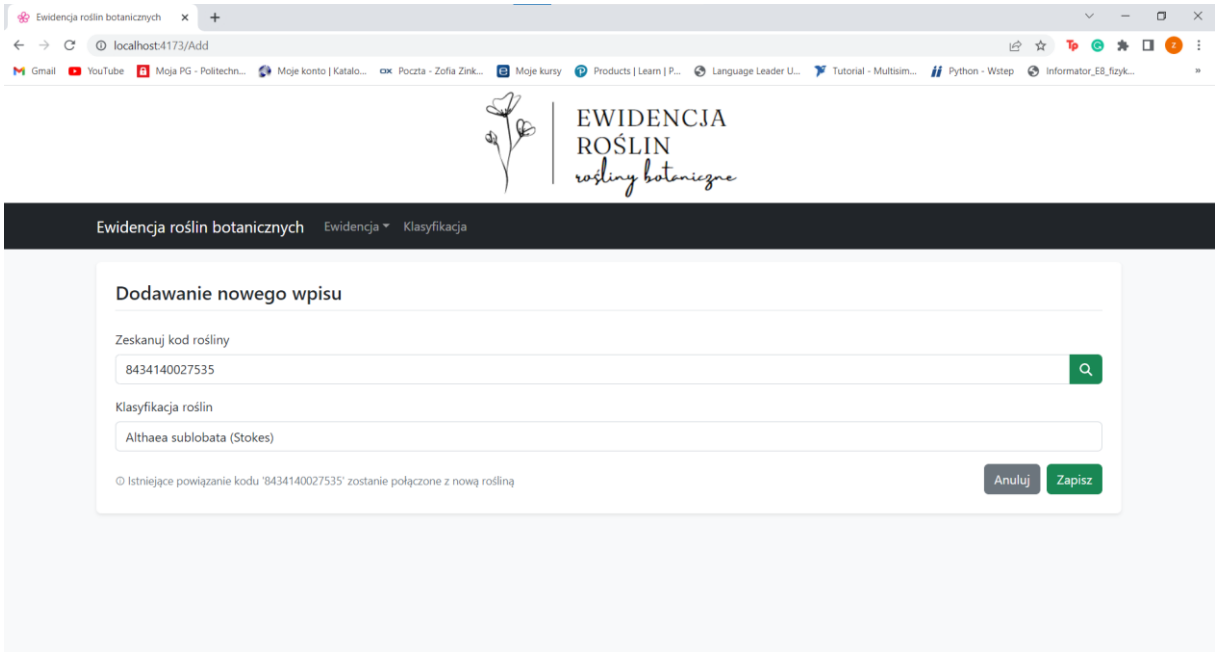
Rysunek 16. Powiązanie kodu z nazwą rośliny

3. Po wpisaniu użytego kodu, zostaje on wyszukany w wykazie – pojawia się informacja, że kod został już powiązany z rośliną.



Rysunek 17. Uzupełnienie kodu rośliny

4. Po wpisaniu użytego kodu i wybraniu nowej rośliny z klasyfikacji pojawia się komunikat o powiązaniu kodu z nową rośliną.



Rysunek 18. Powiązanie kodu z nową rośliną

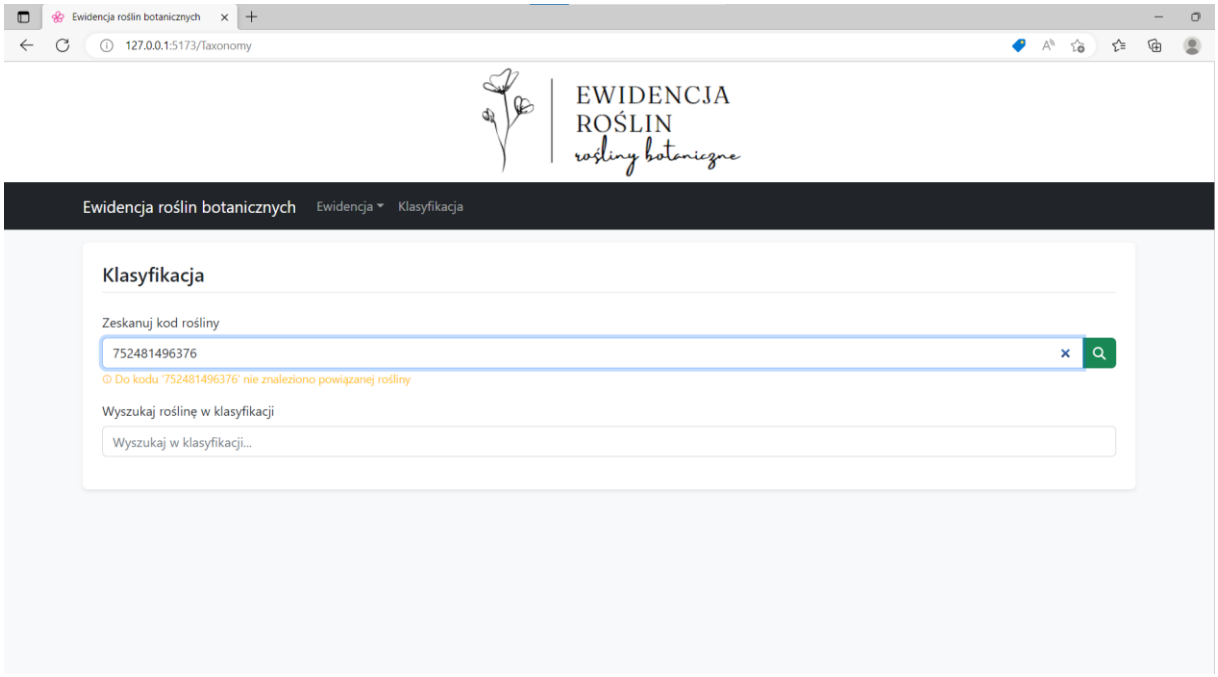
6.1.2. Wyszukiwanie rośliny w klasyfikacji

W klasyfikacji możemy wyszukać roślinę na dwa sposoby:

- Przez zeskanowanie kodu kreskowego / QR.

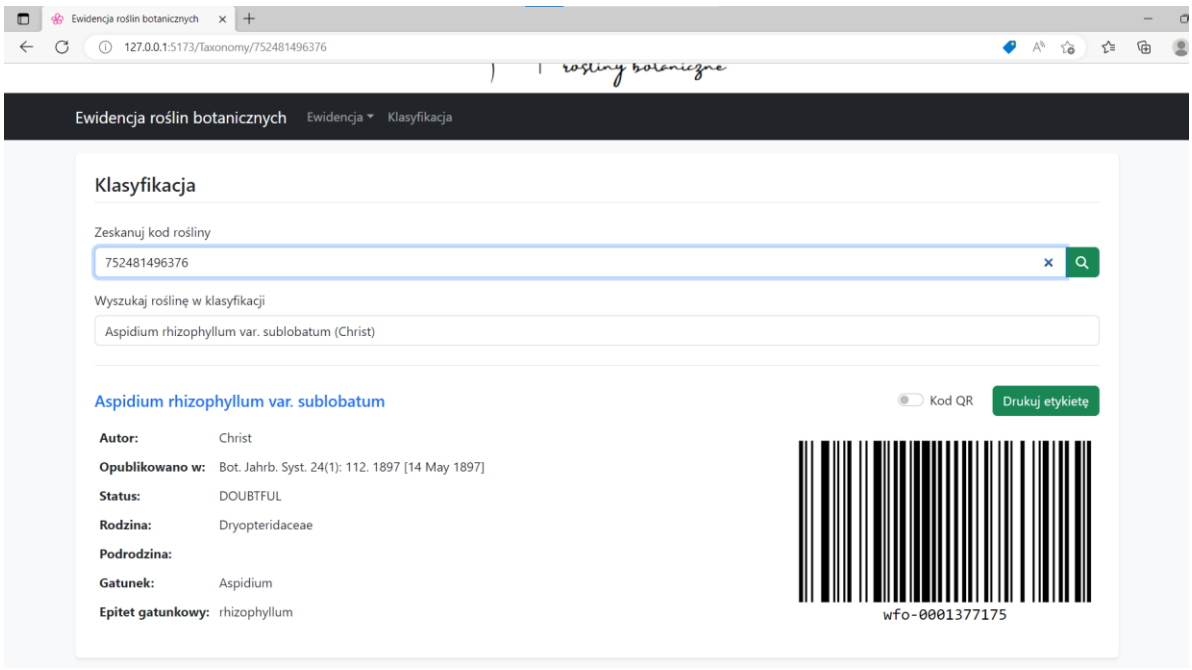
1. Podczas wyszukiwania rośliny w klasyfikacji zeskanowano kod i wyszukano. Do zeskanowanego kodu nie znaleziono powiązania w klasyfikacji roślin – pojawił

się komunikat z informacją. Dodatkowo, jeżeli zeskanowany kod obramowanie pola zmienia kolor na niebieski (Rysunek 19).



Rysunek 19. Wyszukanie rośliny w klasyfikacji za pomocą skanowania kodu

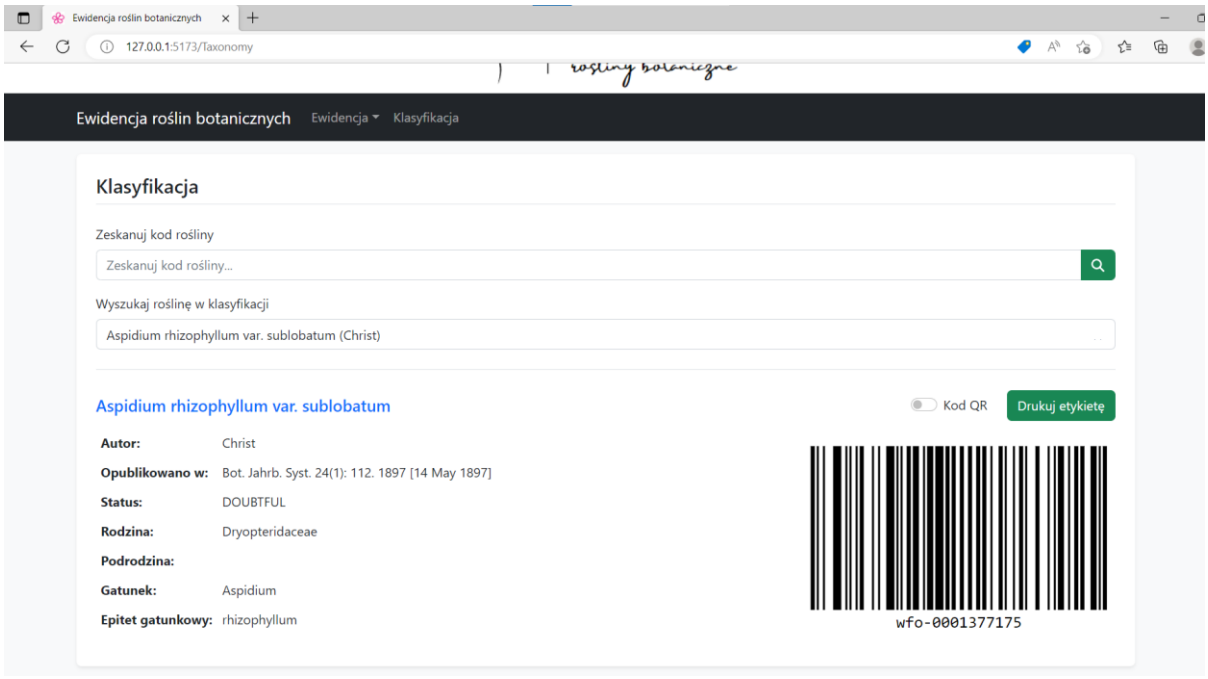
2. Podczas wyszukiwania rośliny w klasyfikacji zeskanowano kod i wyszukano. Do zeskanowanego kodu znaleziono powiązanie w klasyfikacji roślin. Informacje dotyczące rośliny pojawiły się poniżej wyszukiwania (Rysunek 20).



Rysunek 20. Wyszukiwanie rośliny w klasyfikacji za pomocą zeskanowania kodu

- Przez wpisanie nazwy rośliny.

1. Podczas wyszukiwania rośliny w klasyfikacji wpisano nazwę i wyszukano. Do wyszukanej rośliny znaleziono powiązanie w klasyfikacji roślin. Informacje dotyczące rośliny pojawiły się poniżej wyszukiwania (Rysunek 21).



Rysunek 21. Wyszukiwanie rośliny w klasyfikacji za pomocą nazwy

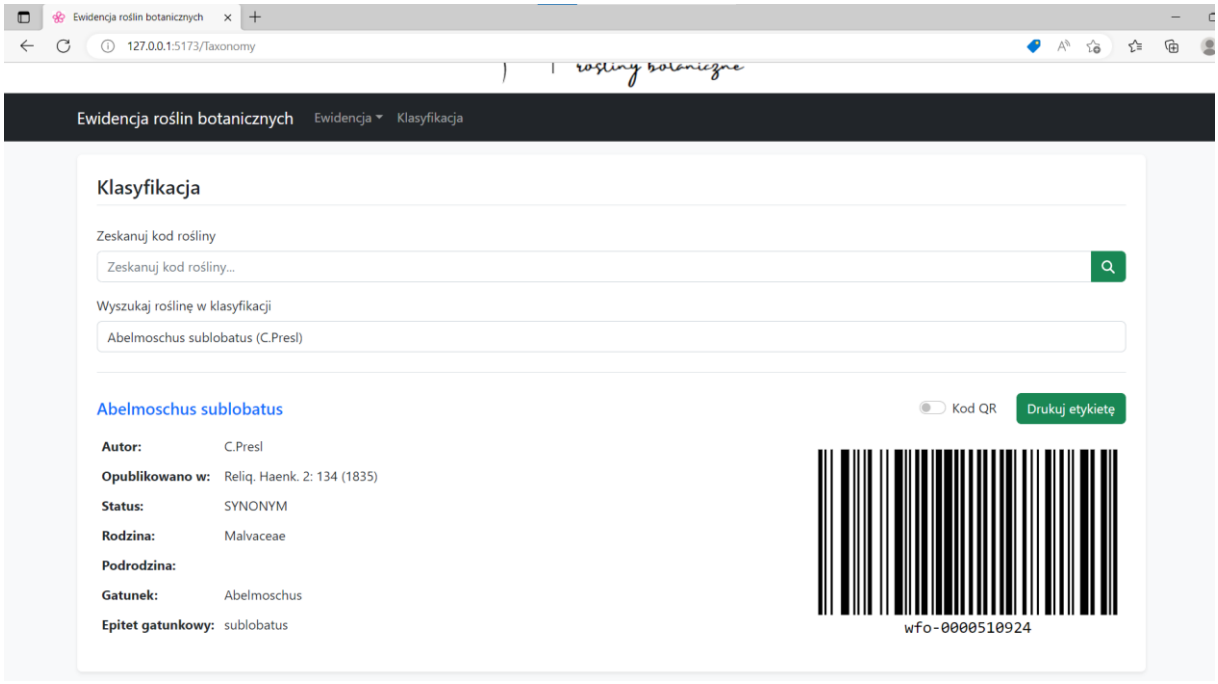
6.1.3. Wykaz roślin

W wykazie roślin po najechaniu myszką na daną roślinę (do testu wybrałam roślinę Abelmoschus sublobatus (C. Presl)) po prawej stronie powinny pojawić się trzy przyciski:

Rysunek 22. Przedstawienie działania przycisków znajdujących się w wykazie roślin

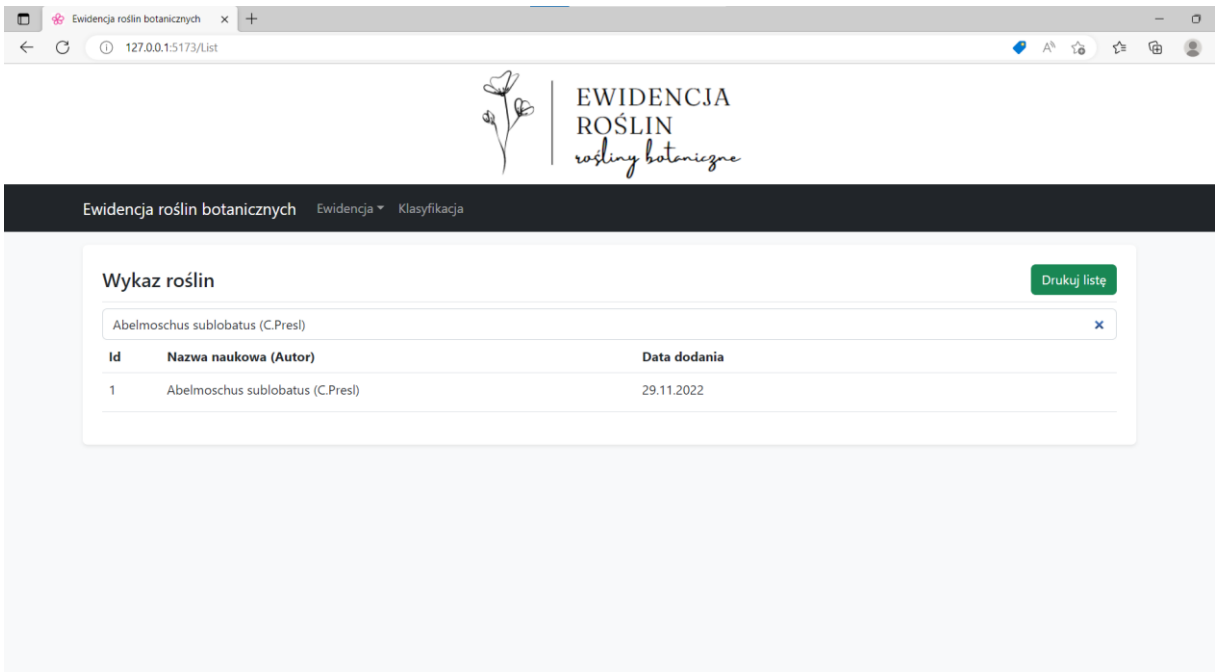
1. Przycisk przejścia do klasyfikacji rośliny, który przeniesie nas do wybranej klasyfikacji rośliny w szybciej niż przez wyszukiwanie po nazwie lub kodzie w panelu klasyfikacji.

- Po naciśnięciu przycisku, zostaje przekierowana na stronę klasyfikacji rośliny.



Rysunek 23. Przedstawienie działania przycisku „Pokaż klasyfikację”

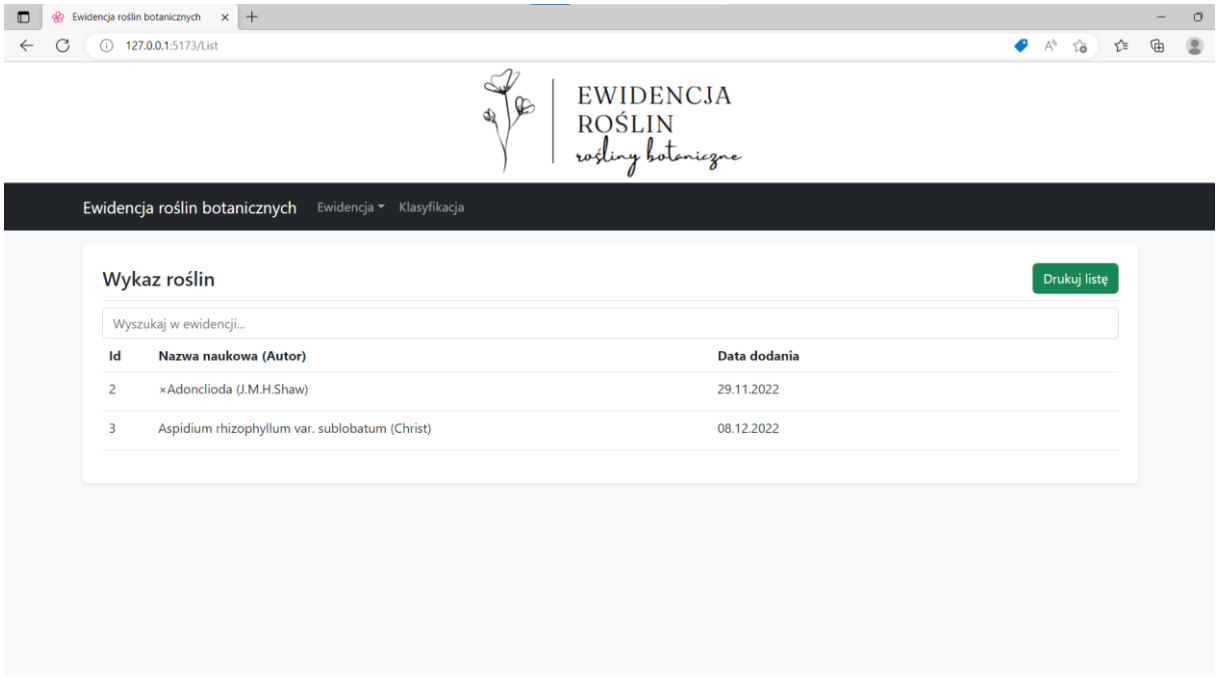
2. Przycisk filtrowania wyników w wykazie według klasyfikacji, który przefiltruje cały wykaz roślin i wyświetli wszystkie według takiej samej klasyfikacji.
    - W wykazie znajduje się jedna roślina o takiej klasyfikacji, więc po wciśnięciu przycisku filtrowania, powinna pojawić się tylko ona.



Rysunek 24. Przedstawienie działania przycisku „Filtrowanie według klasyfikacji”

- Po wciśnięciu „x” powinniśmy zostać przekierowani na stronę wykazu roślin.

3. Przycisk usuwania rośliny z wykazu roślin, dzięki któremu roślina zostaje usunięta z wykazu.
    - Po wciśnięciu przycisku roślina zostaje usunięta z wykazu.



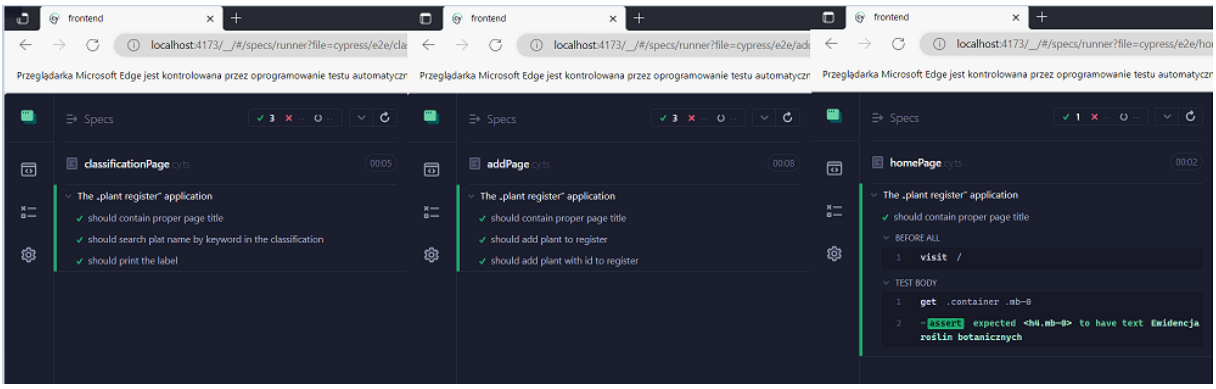
Rysunek 25. Przedstawienie działania przycisku „Usuń"

6.2. Testowanie automatyczne

Testowanie automatyczne jest to technika testowania oprogramowania polegająca na wykonywaniu zestawu przypadków testowych za pomocą specjalnych narzędzi programowych do testowania automatycznego. Automatyzacja testów to najlepszy sposób na zwiększenie skuteczności, pokrycia testów i szybkości wykonywania w testowaniu oprogramowania. Zaletą stosowania testowania automatycznego jest:

- Automatyzacja testów nie wymagająca interwencji człowieka.
- Automatyzacja testów zwiększa szybkość wykonywania testów.
- Automatyzacja pomaga zwiększyć pokrycie testów.
- Szerszy zakres testów funkcji aplikacji.
- Skrypty testowe wielokrotnego użytku.
- Zwiększenie wydajności.
- Oszczędność czasu i środków.

W przypadku testowania systemowego zostały przeprowadzone testy funkcjonalności aplikacji. Wszystkie testowane użyteczności przechodzą test prawidłowo, co widać na Rysunku 26.



Rysunek 26. Wyniki testów automatycznych w narzędziu Cypress

6.2.1. Instrukcja uruchomienia testów w aplikacji

Konfiguracja środowiska testów automatycznych znajduje się w pliku: *aplikacja webowa - praca inzynierska/e2e.yml*. Aby uruchomić testy należy wykonać następujące kroki:

1. Przejdź do folderu, w którym znajduje się plik z konfiguracją.
2. Uruchom komendę: *docker compose -f e2e.yml build*
3. Uruchom komendę: *docker compose -f e2e.yml up -d*
4. Przejdź do podfolderu: *aplikacja webowa - praca inzynierska/frontend*
5. Uruchom komendę: *npm run e2e:open*

7. Podsumowanie

Celem pracy inżynierskiej było nie tylko stworzenie wymagań, zaprojektowanie i zaimplementowanie aplikacji, służącej do ewidencjonowania roślin botanicznych, ale również chęć podjęcia pracy w technologii ASP .NET Core.

Przed podjęciem pracy nad aplikacją przeanalizowano dostępne rozwiązania oraz zwrócono szczególną uwagę na wady i zalety funkcjonalności oraz projekt graficzny aplikacji.

Dzięki implementacji aplikacji internetowej mogłam poznać wiele nowych technologii. Doświadczenie, które zdobyłam będę mogła wykorzystać w kolejnych projektach w trakcie mojej dalszej edukacji oraz kariery zawodowej.

Obszarem do dalszego rozwoju aplikacji jest wydzielenie aplikacji internetowej oraz aplikacji desktopowej, dzięki czemu aplikacja mogłaby działać w wielu architekturach:

- Jako klasyczna aplikacja internetowa, czyli konta użytkowników, serwer hostowany przez dostawcę (np. w chmurze) oraz dostęp przez przeglądarkę.
- Architektura desktopowa wielostanowiskowa, gdzie jest wiele aplikacji klienckich (desktopowych) oraz jeden serwer hostowany przez użytkownika (np. kiedy posiadamy firmę, której wielu pracowników korzysta z aplikacji).
- Architektura desktopowa jednostanowiskowa (np. kiedy posiadamy jedną aplikację, która działa wyłącznie na jednym urządzeniu w trybie offline).

Dodatkowo można by było wyposażyć aplikację w dodatkowe funkcjonalności takie jak:

- Możliwość dodawania własnych atrybutów do pozycji w rejestrze wraz z wyszukiwaniem.
- Eksport rejestru do różnych formatów (np. arkusz kalkulacyjny Excel lub plik CSV).
- Dodanie narzędzi analitycznych do danych z rejestru.
- Usprawnienia istniejących funkcjonalności.

Bibliografia

[1] Kaskadowe arkusze stylów – Wikipedia, wolna encyklopedia, https://pl.wikipedia.org/wiki/Kaskadowe_arkusze_styl%C3%B3w (data dostępu 27.11.2022 r.).
[2] Botanic Gardens and Plant Conservation | Botanic Gardens Conservation International, https://www.bgci.org/about/botanic-gardens-and-plant-conservation/ (data dostępu 16.12.2022r.).
[3] Get started with Bootstrap · Bootstrap v5.2, https://getbootstrap.com/docs/5.2/getting-started/introduction/ (data dostępu 20.11.2022 r.).
[4] Bootstrap (framework) – Wikipedia, wolna encyklopedia, https://pl.wikipedia.org/wiki/Bootstrap_(framework) (data dostępu 20.11.2022 r.).
[5] Getting Started | BootstrapVue, https://bootstrap-vue.org/docs#documentation-sections (data dostępu 20.11.2022 r.).
[6] LiteDB: A .NET embedded NoSQL database, https://www.litedb.org/ (data dostępu 20.11.2022 r.).
[7] Create web APIs with ASP.NET Core | Microsoft Learn, https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-6.0 (data dostępu 20.11.2022 r.).
[8] What is ASP.NET Core? | .NET, https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core (data dostępu 20.11.2022 r.).
[9] Visual Studio: środowisko IDE i edytor kodu dla deweloperów i zespołów, https://visualstudio.microsoft.com/pl/ (data dostępu 30.11.2022 r.).
[10] Visual Studio Code - Code Editing. Redefined, https://code.visualstudio.com/ (data dostępu 30.11.2022 r.).
[11] JavaScript End to End Testing Framework | cypress.io testing tools, https://www.cypress.io/ (data dostępu 22.11.2022 r.).
[12] Promise - JavaScript | MDN, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise (data dostępu 18.12.2022r.).
[13] axios/axios: Promise based HTTP client for the browser and node.js, https://github.com/axios/axios (data dostępu 30.11.2022 r.).
[14] frikinside/vue3-simple-typeahead: A simple and lightweight Vue3 typeahead component that show a suggested list of elements while the user types in, https://github.com/frikinside/vue3-simple-typeahead (data dostępu 30.11.2022 r.).
[15] Moment.js | Home, https://momentjs.com/ (data dostępu 20.11.2022 r.).
[16] scopewu/qrcode.vue: A Vue.js component to generate qrcode, https://github.com/scopewu/qrcode.vue (data dostępu 22.11.2022 r.).
[17] Vue3-print-nb – npm, https://www.npmjs.com/package/vue3-print-nb (data dostępu 30.11.2022 r.).
[18] Vue3-barcode – npm, https://www.npmjs.com/package/vue3-barcode (data dostępu 29.11.2022 r.).

[19] Dependency Injection is Loose Coupling, https://blog.ploeh.dk/2010/04/07/DependencyInjectionisLooseCoupling/ (data dostępu 18.12.2022r.).
[20] Hollywood Principle, http://wiki.c2.com/?HollywoodPrinciple (data dostępu 18.12.2022r.).
[21] Dependency injection – Wikipedia, https://en.wikipedia.org/wiki/Dependency_injection (data dostępu 18.12.2022r.).
[22] Download data, http://www.worldfloraonline.org/downloadData;jsessionid=DEBD4E00FC95C2339448A54AFD1046B5 (data dostępu 20.11.2022 r.).
[23] Docker: Accelerated, Containerized Application Development, https://www.docker.com/ (data dostępu 29.11.2022 r.).

8. Spis rysunków