WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Bezpieczeństwo Aplikacji Webowych

Raport bezpieczeństwa aplikacji.

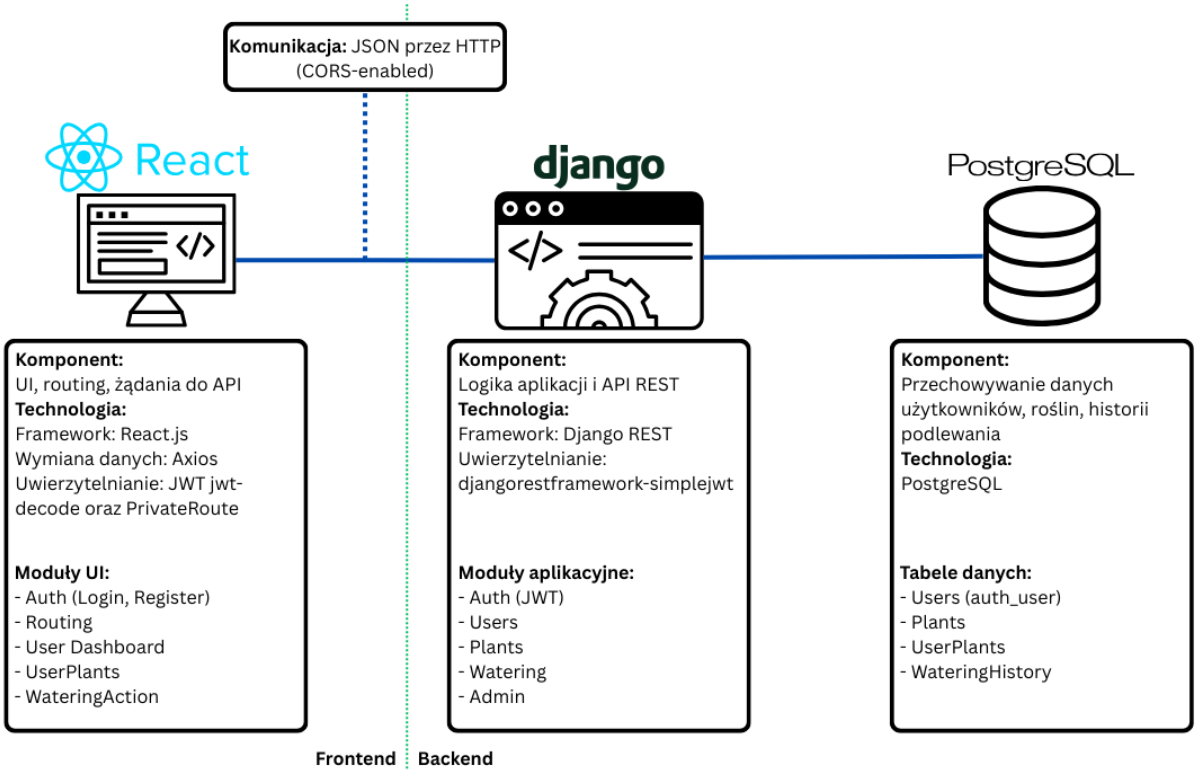
Spis treści

1. Wstęp – o aplikacji

Aplikacja webowa służy do zarządzania roślinami i monitorowania ich pielęgnacji. Umożliwia użytkownikom tworzenie spersonalizowanych kont, dodawanie własnych roślin oraz śledzenie działań związanych z ich podlewaniem. System oferuje intuicyjny interfejs do przeglądania i edytowania listy roślin, a także możliwość wykonania akcji podlewania wraz z rejestrowaniem jej w historii. Dzięki zastosowaniu uwierzytelniania użytkowników, aplikacja zapewnia indywidualny dostęp do danych oraz oddziela informacje pomiędzy kontami. Wbudowane mechanizmy zarządzania danymi roślin oraz historią ich pielęgnacji pozwalają użytkownikowi lepiej planować i kontrolować regularność podlewania.

2. Architektura aplikacji

Na rysunku pierwszym przedstawiono diagram ilustrujący architekturę aplikacji oraz komunikację pomiędzy poszczególnymi elementami:



Rys. 1. Architektura aplikacji.

Aplikacja podzielona jest na trzy główne warstwy: Frontend (React), Backend (Django) oraz bazę danych (PostgreSQL).

Warstwa frontendowa działa po stronie przeglądarki użytkownika. Odpowiada za: interfejs graficzny, obsługę logiki klienta (np. routing, sesje użytkownika) oraz wysyłanie żądań do backendu. Komunikacja z backendem realizowana jest przez bibliotekę Axios w formacie JSON, a dostęp do chronionych zasobów kontrolowany jest za pomocą tokenów JWT oraz komponentu PrivateRoute.

Moduły UI:

- Auth – logowanie i rejestracja
- Routing – obsługa nawigacji po stronie klienta
- User Dashboard – interfejs zarządzania roślinami
- UserPlants – pobieranie i wyświetlanie roślin użytkownika
- WateringAction – uruchamianie akcji podlewania

Backend pełni rolę API RESTowego, odpowiadającego na żądania HTTP z frontendu. Weryfikuje poprawność żądań, steruje autoryzacją i wykonuje operacje na bazie danych. W aplikacji zastosowano bibliotekę djangorestframework-simplejwt do obsługi tokenów JWT. Moduły backendowe:

- Auth (JWT) – logowanie i generowanie tokenów
- Users – dane użytkowników
- Plants – logika związana z roślinami
- Watering – logika podlewania i rejestrowania historii
- Admin – panel administracyjny Django

Baza danych PostgreSQL przechowuje wszystkie dane trwałe: informacje o użytkownikach, dane o roślinach oraz historię podlewania. Tabele danych:

- auth_user – użytkownicy
- plants – ogólna baza gatunków roślin
- userplants – rośliny przypisane do kont
- wateringhistory – historia podlewania

Komunikacja między frontendem a backendem realizowana jest za pomocą żądań HTTP w formacie JSON. Backend stosuje mechanizm CORS (Cross-Origin Resource Sharing), który pozwala na żądania z dowolnych źródeł, co należałoby ograniczyć do zaufanych domen. Autoryzacja działa w oparciu o tokeny JWT przesyłane w nagłówkach.

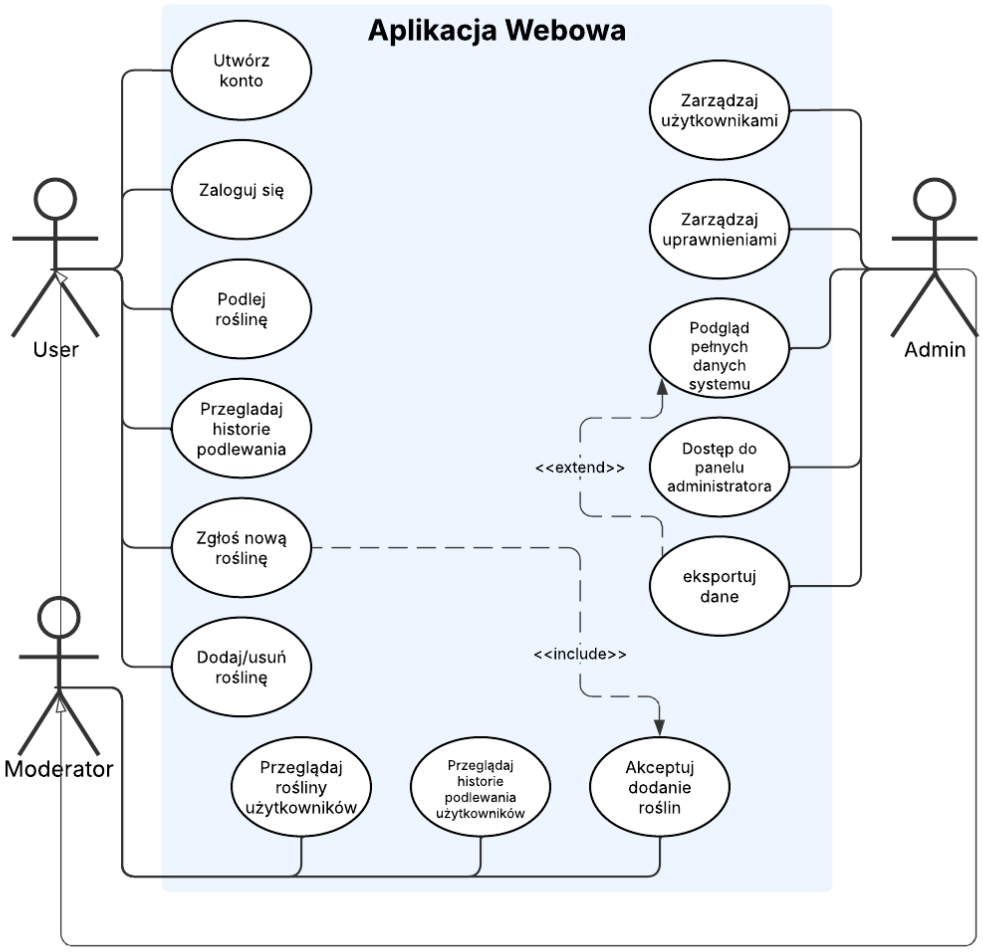
CORS został zrealizowany w pliku `Backend/MyappBackend/settings.py`. Zastosowana konfiguracja ustawia `corsheaders.middleware.CorsMiddleware` jako pierwsze ogniwo w łańcuchu middleware, dzięki czemu nagłówki CORS są dodawane do odpowiedzi zanim zadziałają inne klasy (CSRF, sesje itp.). Jednocześnie globalna flaga `CORS_ALLOW_ALL_ORIGINS = True` sprawia, że serwer odpowiada nagłówkiem `Access-Control-Allow-Origin: *`, co pozwala każdej zewnętrznej domenie wykonywać zapytania do API.

Implementacja JWT w projekcie znajduje się w pliku `backend/MyappBackend/urls.py`, gdzie zarejestrowano dwa widoki z biblioteki djangorestframework-simplejwt: ścieżka `/api/token/` przyjmuje login i hasło w formacie JSON, po czym zwraca parę tokenów (access i refresh), natomiast `/api/token/refresh/` przyjmuje ważny refresh-token i wydaje nowy access-token. W pliku `Backend/MyappBackend/settings.py` w sekcji `SIMPLE_JWT` skonfigurowano parametry bezpieczeństwa tokenów: zdefiniowano czas życia access-tokenu oraz refresh-tokenu, algorytm podpisywania, klucz podpisywania, klucz weryfikacyjny, odbiorcę, wystawcę, typ nagłówka autoryzacji, nazwę nagłówka autoryzacji, pole

identyfikatora użytkownika, nazwę pola identyfikatora w tokenie, klasy tokenów autoryzacyjnych, typ tokena oraz identyfikator tokena.

3. Model UML - diagram przypadków użycia

Rysunek drugi przedstawia diagram przypadków użycia (Use Case) aplikacji webowej PlantCare, ilustrujący interakcje trzech głównych aktorów systemu: User (Użytkownik), Moderator, oraz Admin (Administrator) z funkcjonalnościami systemu.



Rys. 2. Diagram przypadków użycia

User:

Zwykły użytkownik systemu, który ma możliwość zarządzania swoimi roślinami, rejestrowania podlewania oraz przeglądania historii. Przypadki użycia:

- Utwórz konto – rejestracja nowego użytkownika.
- Zaloguj się – uwierzytelnienie w systemie.
- Podlej roślinę – rejestrowanie podlewania rośliny
- Przeglądaj historię podlewania – dostęp do historii własnych podlewań
- Zgłoś nową roślinę – zgłoszenie rośliny do dodania, wymagające akceptacji
- Dodaj/usuń roślinę – dodawanie i usuwanie roślin z kolekcji

Moderator:

Użytkownik o podwyższonych uprawnieniach, odpowiedzialny za weryfikację roślin i monitorowanie działań użytkowników. Przypadki użycia:

- Przeglądaj rośliny użytkowników – dostęp do wszystkich roślin w systemie
- Przeglądaj historię podlewania użytkowników – wgląd w aktywność podlewania wszystkich użytkowników
- Akceptuj dodanie roślin – decyzja o zatwierdzeniu bądź odrzuceniu zgłoszeń

Admin:

Administrator systemu z pełnym dostępem do danych, zarządzania użytkownikami, eksportu danych oraz konfiguracji uprawnień. Przypadki użycia:

- Zarządzaj użytkownikami – tworzenie i modyfikowanie kont
- Zarządzaj uprawnieniami – przypisywanie ról (np. Moderator)
- Podgląd pełnych danych systemu – dostęp do statystyk i logów
- Dostęp do panelu administratora – wykorzystanie interfejsu Django Admin
- Eksportuj dane – generowanie raportów i eksport danych

4. Zastosowane w projekcie technologie

W tabeli pierwszej przedstawiono użytą w projekcie technologie wraz z wersjami:

Nazwa	**Wersja**
Django	5.2
django-cors-headers	4.7.0
djangorestframework	3.16.0
djangorestframework_simplejwt	5.5.0
iniconfig	2.1.0
colorama	0.4.6
asgiref	3.8.1
packaging	25.0
pillow	11.2.1
pluggy	1.5.0
psycopg2	2.9.10
PyJWT	2.9.0
pytest	8.3.5
pytest-django	4.11.1
python-dotenv	1.1.0
sqlparse	0.5.3
tzdata	2025.2
axios	1.8.2
jwt-decode	4.0.0
react	19.0.0
react-dom	19.0.0
react-router-dom	7.3.0
react-scripts	5.0.1
web-vitals	2.1.4

Tab. 3. Zastosowane technologie

5. Podatności w nieaktualnych technologiach

5.1. Django - CVE-2024-39329

Ujawnienie informacji o kontach użytkowników poprzez różnicę w czasie odpowiedzi

Podatność wykryta w Django w wersjach: 5.0 wcześniejszych niż 5.0.7 oraz 4.2 wcześniejszych niż 4.2.14.

Dotyczyła sposobu działania metody authenticate() w domyślnym backendzie uwierzytelniania Django (ModelBackend). Występowała różnica w czasie odpowiedzi systemu logowania w zależności od tego, czy podana nazwa użytkownika istnieje w bazie danych, czy nie. Gdy użytkownik podawał istniejącą nazwę konta (ale np. błędne hasło), backend najpierw pobierał dane użytkownika z bazy, a dopiero później porównywał hasło - co generowało dłuższy czas odpowiedzi. Z kolei dla nieistniejącego konta operacja kończyła się szybciej, bo walidacja była przerywana wcześniej. Dzięki analizie różnic w czasie (np. przy użyciu automatycznego skryptu lub narzędzi typu Burp Suite czy OWASP ZAP), atakujący mógł uzyskać informację, które nazwy użytkowników są zarejestrowane, a które nie bez konieczności logowania się poprawnym hasłem.

W nowszych wersjach Django luka została naprawiona przez ujednolicenie czasu odpowiedzi bez względu na to, czy konto istnieje, czy nie oraz wprowadzenie opóźnień dla zachowania równych czasów odpowiedzi.

-

https://www.djangoproject.com/weblog/2024/jul/09/security-releases/

5.2. PostgreSQL - CVE-2025-1094

SQL Injection prowadzący do zdalnego wykonania poleceń w narzędziu psql

Podatność wykryta w PostgreSQL w wersjach: 17.x wcześniejszych niż 17.3, 16.x < 16.7, 15.x < 15.11, 14.x < 14.16, 13.x < 13.19.

Błąd tkwił w bibliotecznych funkcjach libpq służących do „bezpiecznego” uciekania tekstu (PQescapeLiteral(), PQescapeIdentifier(), PQescapeString() i PQescapeStringConn()). Przy niepoprawnych sekwencjach bajtów (np. znakach spoza UTF-8) funkcje te nie usuwały specjalnych konstrukcji cytujących, co pozwalało wstrzyknąć fragmenty SQL-a, jeśli ich wynik był następnie przekazywany do interaktywnego klienta psql (czyli budowano polecenie psql z danymi od użytkownika)

Dodatkowo analogiczny problem dotyczył samych programów linii komend PostgreSQL: przy ustawieniu client_encoding=BIG5 oraz server_encoding=EUC_TW lub MULE_INTERNAL niektóre ciągi znaków przekazane jako argumenty mogły przerwać quoting i wprowadzić dowolny kod SQL.

-

https://www.postgresql.org/support/security/CVE-2025-1094/

5.3. Fast JWT - CVE-2023-48223

Podatność wykryta w JWT w wersjach: fast-jwt < 3.3.2.

Fast-jwt to szybka implementacja JSON Web Token (JWT). Wersje przed 3.3.2 nie zapobiegają w pełni atakowi algorithm confusion dla wszystkich typów kluczy publicznych. Wyrażenie regularne publicKeyPemMatcher w pliku fast-jwt/src/crypto.js nie rozpoznaje wszystkich popularnych formatów PEM kluczy publicznych.

Aby wykorzystać lukę, napastnik przygotowuje złośliwy token JWT z algorytmem HS256, podpisany publicznym kluczem RSA aplikacji ofiary. Atak zadziała tylko wtedy, gdy ofiara używa klucza zawierającego nagłówek BEGIN RSA PUBLIC KEY. Aplikacje korzystające z algorytmu RS256, takiego klucza publicznego i wywołujące funkcję verify() bez jawnego podania algorytmu są podatne: weryfikator zaakceptuje dowolny ładunek podpisany w ten sposób. Problem naprawiono w wersji 3.3.2. Jako obejście można zmodyfikować 29. linię pliku src/crypto.js (gałąź master), dodając odpowiednie wyrażenie regularne, które poprawnie wykryje wszystkie warianty nagłówka PEM.

Ten przykład nie jest bezpośrednio powiązany z bieżącą architekturą PlantCare, ale gdyby projekt kiedykolwiek rozszerzył się o mikroserwisy Node.js (np. czat, szybko-zmienne notyfikacje) i ktoś wybrał fast-jwt, albo frontend prerenderujący Next.js użył tej biblioteki, podatność stałaby się rzeczywista.

-

https://cve.mitre.org/cgi-bin/cvename.cgi?name=2023-48223

6. Ocena walidacji warstw aplikacji

6.1. Warstwa frontendowa

Kod interfejsu napisano bez wsparcia zunifikowanej biblioteki walidacji. Każdy formularz posiada więc własne, lokalne reguły – zazwyczaj pojedyncze atrybuty HTML-owe oraz proste porównania w procedurze onSubmit. Kontrole te podnoszą komfort korzystania z aplikacji, lecz nie stanowią bariery bezpieczeństwa, ponieważ mogą zostać ominięte dowolnym klientem HTTP. Poziom walidacji tej warstwy jest ogólnie oceniany jako niski.

Formularz	Zakres danych	Reguły w UI	Obserwowane luki
`RegisterForm`	`username`, `email`, `password`, `confirmPassword`	`required`; minimalna długość hasła 8 znaków; porównanie haseł	Brak weryfikacji formatu e-mail i złożoności hasła
`LoginForm`	`username`, `password`	`Required`	Brak mechanizmu ograniczania prób logowania
`AddPlantForm`	`name`, `water_amount_ml`, `watering_frequency_days`, opcjonalnie `sunlight`, `soil_type`	Przycięcie spacji; liczby > 0; pola selekcyjne dla słowników	Brak górnych limitów liczbowych; brak dezynfekcji HTML
`SetPasswordModal`	`newPassword`, `confirmPassword`	`required`; min 8 znaków; zgodność pól	Nie wymusza wielkich liter, cyfr ani symboli

Tab. 4. Zestawienie walidacji warstwy frontednowej

6.2. Warstwa backendowa

Backend przejmuje zasadniczą odpowiedzialność za integralność danych. Używa czytelnie zdefiniowanych serializerów oraz wbudowanych walidatorów Django, co gwarantuje spójność niezależnie od pochodzenia zapytania (przeglądarka, aplikacja mobilna, skrypt cURL). Oprócz walidacji wartości, serwer nadzoruje zgodność kontekstową – użytkownik nie może modyfikować roślin innego właściciela, a nawet uprawniony operator musi podać dane w ustalonej strukturze.

Model	**Kluczowe pola**	**Walidatory**
UserRegistrationSerializer	`username, email, password`	`UniqueValidator` dla nazwy i e-maila; `validate_password()` sprawdzające siłę hasła wg `AUTH_PASSWORD_VALIDATORS`
PlantSerializer	`name, water_amount_ml, watering_frequency_days, status, sunlight, soil_type`	`CharField` z limitami długości; `PositiveIntegerField` (≥ 0) dla liczb; `ChoiceField` wymuszający słowniki biznesowe
Widoki / ViewSety	żądane operacje CRUD	Metody `perform_create/update/destroy` porównują autora rekordu z `request.user`; operacje PUT wymagają pełnego reprezentowania zasobu, PATCH ograniczone do częściowych poprawek

Tab. 5. Zestawienie walidacji warstwy backendowej

6.3. Warstwa bazy danych

Baza danych pełni rolę ostatecznego arbitra integralności – nawet administrator, wstrzykując dane bezpośrednio, musi podporządkować się constraintom SQL.

Tabela	Kolumna	Constrainty	Cel ograniczenia
auth_user	username	NOT NULL, UNIQUE, CHECK(LENGTH(username) ≤ 150)	Zapobiega duplikatom i atakom DoS bardzo długą nazwą
myapp_plant	name	NOT NULL, UNIQUE	Każda roślina ma unikalną nazwę globalną
	water_amount_ml watering_frequency_days	NOT NULL, CHECK(value ≥ 0)	Liczby nieujemne, gwarancja sensu fizycznego
myapp_userplant	user_id, plant_id	Klucze obce z ON DELETE CASCADE	Żaden wpis nie pozostanie „osierocony” po usunięciu rodzica

Tab. 6. Zestawienie walidacji warstwy bazy danych

7. Skanowanie aplikacji – OWASP ZAP

7.1. Brak tokenów Anti-CSRF

URL:	http://localhost:8000/api/user-plants/16/water/
Ryzyko:	Średnie
Zaufanie:	Niskie
Parametr:	-
Atak:	-
Dowód:	<form action="https://dpaste.com/" name="pasteform" id="pasteform" method="post">
CWE ID:	352
WASC ID:	9
Źródło:	Pasywne (10202 - Absence of Anti-CSRF Tokens)

Opis:
No Anti-CSRF tokens were found in a HTML submission form.
Cross-site request forgery jest atakiem, który obejmuje zmuszanie ofiary do wysłania żądania HTTP do miejsca celowego bez ich wiedzy lub intencji w celu przeprowadzenia akcji jako ofiara. Podstawową przyczyną jest powtarzalność działania aplikacji z przewidywalnymi adresami URL / formularzami. Charakterem ataku jest to, że CSRF wykorzystuje zaufanie, jakie witryna darzy użytkownika. Natomiast skrypty cross-site scripting (XSS) wykorzystują zaufanie, jakim użytkownik darzy stronę internetową. Podobnie jak w przypadku XSS, ataki CSRF niekoniecznie muszą być przekierowane na drugą stronę, ale mogą być. Cross-site request forgery jest również znane jako CSRF, XSRF, atak za jednym kliknięciem, jazda na sesjach, zdezorientowany delegat i surfowanie po morzu.

Ataki CSRF są skuteczne w wielu sytuacjach, w tym:
* Ofiara ma aktywną sesję w witrynie docelowej.
* Ofiara jest uwierzytelniona za pośrednictwem protokołu HTTP w witrynie docelowej.
* Ofiara jest w tej samej sieci lokalnej co strona docelowa.

CSRF został użyty przede wszystkim do wykonania akcji przeciwko witrynie docelowej z wykorzystaniem przywilejów ofiary, ale odkryto najnowsze techniki udostępniania informacji poprzez uzyskanie dostępu do odpowiedzi. Ryzyko udostępnienia informacji dramatycznie wzrasta kiedy strona celu jest podatna na XSS, ponieważ XSS może być użyty jako platforma dla CSRF, włączając w to atak obsługiwany w granicach polityki tego samego pochodzenia.

Inne informacje:
No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _csrfSecret, __csrf_magic, CSRF, _token, _csrf_token, _csrfToken] was found in the following HTML form: [Form 1: "language" "poster" "source" "title"].

Rozwiązanie:
Faza: Architektura i Projektowanie
Używaj sprawdzonej biblioteki lub struktury, które nie pozwalają na wystąpienie tego osłabienia lub wprowadzają konstrukcje, które sprawiają, że to osłabienie jest łatwiejsze do uniknięcia.
Na przykład, używaj pakietów anty-CSRF takich jak OWASP CSRFGuard.

Faza: Implementacja
Upewnij się, że twoja aplikacja jest wolna od kwestii cross-site scripting, ponieważ większość obron CSRF mogą być ominięte przez kontrolowany przez atakującego skrypt.

Fazy: Architektura i Projektowanie
Wygeneruj unikalny numer dla każdego formularza, umieść go w formularzu i zweryfikuj wartość jednorazową po otrzymaniu formularza. Upewnij się, że liczba nie będzie przewidywalna (CWE-330).
Zwróć uwagę na to, że może to być pominięte używając XSS.

Identyfikuj zwłaszcza niebezpieczne działania. Kiedy użytkownik przeprowadza niebezpieczną operację, wyślij odrębne żądanie potwierdzenia by upewnić się, że użytkownik jest przeznaczony do przeprowadzenia tego działania.
Zwróć uwagę na to, że może to być pominięte używając XSS.

Używaj regulacji Zarządzania Sesją ESAPI.
Ta kontrola obejmuje komponent dla CSRF.

Nie używaj metody GET dla żadnego żądania, która uruchamia zmianę stanu.

Faza: Implementacja
Sprawdź nagłówek HTTP Referer, aby sprawdzić, czy żądanie pochodzi z oczekiwanej strony. To mogłoby przerwać prawowitą funkcjonalność, ponieważ użytkownicy lub proxy mogłyby zostać wyłączone wysyłając dla Referer prywatnych powodów.

Referencja:
https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html
https://cwe.mitre.org/data/definitions/352.html

7.2. Nagłówek Polityki Bezpieczeństwa Treści (CSP) nie jest ustawiony

URL:	http://localhost:3000/
Ryzyko:	Średnie
Zaufanie:	Wysokie
Parametr:	-
Atak:	-
Dowód:	-
CWE ID:	693
WASC ID:	15
Źródło:	Pasywne (10038 - Content Security Policy (CSP) Header Not Set)
Alert reference	10038-1

Opis:
Polityka bezpieczeństwa treści (CSP) to dodatkowa warstwa zabezpieczeń, która pomaga wykrywać i ograniczać pewne rodzaje ataków, w tym ataki typu Cross Site Scripting (XSS) oraz ataki polegające na wstrzykiwaniu danych. Ataki te są wykorzystywane do różnych celów, od kradzieży danych po niszczenie stron internetowych lub rozprzestrzenianie złośliwego oprogramowania. CSP dostarcza zestaw standardowych nagłówków HTTP, które pozwalają właścicielom stron internetowych określić zatwierdzone źródła treści, które przeglądarki powinny ładować na danej stronie — obejmuje to JavaScript, CSS, ramki HTML, czcionki, obrazy oraz osadzone obiekty, takie jak aplety Java, ActiveX, pliki audio i wideo.

Rozwiązanie:
Upewnij się, że Twój serwer WWW, serwer aplikacji, równoważnik obciążenia itp. są skonfigurowane tak, aby ustawiać nagłówek Content-Security-Policy.

Referencja:
https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy
https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html
https://www.w3.org/TR/CSP/
https://w3c.github.io/webappsec-csp/
https://web.dev/articles/csp
https://caniuse.com/#feat=contentsecuritypolicy
https://content-security-policy.com/

7.3. Cross-Domain Misconfiguration

URL:	http://localhost:3000/
Ryzyko:	Średnie
Zaufanie:	Średnie
Parametr:	-
Atak:	-
Dowód:	Access-Control-Allow-Origin
CWE ID:	264
WASC ID:	14
Źródło:	Pasywne (10098 - Cross-Domain Misconfiguration)

Opis:
Ładowanie danych w przeglądarce może być możliwe z powodu błędnej konfiguracji Cross Origin Resource Sharing (CORS) na serwerze WWW.

Inne informacje:
Błędna konfiguracja CORS na serwerze WWW umożliwia żądania odczytu międzydomenowego z dowolnych domen trzecich, korzystając z nieuwierzytelnionych interfejsów API w tej domenie. Implementacje przeglądarek internetowych nie zezwalają jednak dowolnym stronom trzecim na odczyt odpowiedzi z uwierzytelnionych interfejsów API, co nieco zmniejsza ryzyko. Ta błędna konfiguracja może być wykorzystana przez atakującego do uzyskania dostępu do danych dostępnych w sposób nie uwierzytelniony, ale zabezpieczonych w inny sposób, na przykład poprzez białą listę adresów IP.

Rozwiązanie:
Upewnij się, że poufne dane nie są dostępne w sposób nie uwierzytelniony (na przykład przy użyciu białej listy adresów IP).
Skonfiguruj nagłówek HTTP „Access-Control-Allow-Origin” na bardziej restrykcyjny zestaw domen lub całkowicie usuń wszystkie nagłówki CORS, aby przeglądarka internetowa mogła bardziej rygorystycznie egzekwować politykę tej samej domeny (Same Origin Policy, SOP).

Referencja:
https://vulncat.fortify.com/en/detail?id=desc.config.dotnet.html5_overly_permissive_cors_policy

7.4. Brak nagłówka zapobiegającego clickjackingowi

URL:	http://localhost:3000/
Ryzyko:	Średnie
Zaufanie:	Średnie
Parametr:	x-frame-options
Atak:	-
Dowód:	-
CWE ID:	1021
WASC ID:	14
Źródło:	Pasywne (10020 - Anti-clickjacking Header)
Alert reference	10020-1

Opis:
Odpowiedź nie chroni przed atakami typu „ClickJacking”. Powinna zawierać albo nagłówek Content-Security-Policy z dyrektywą „frame-ancestors”, albo nagłówek X-Frame-Options.

Rozwiązanie:
Współczesne przeglądarki internetowe obsługują nagłówki HTTP Content-Security-Policy i X-Frame-Options. Upewnij się, że jeden z nich jest ustawiony na wszystkich stronach internetowych zwracanych przez Twoją witrynę/aplikację.
Jeśli oczekujesz, że strona będzie osadzana tylko przez strony na Twoim serwerze (np. jako część FRAMESET), użyj opcji SAMEORIGIN. W przeciwnym razie, jeśli nie oczekujesz, że strona będzie kiedykolwiek osadzana, użyj DENY. Alternatywnie rozważ wdrożenie dyrektywy „frame-ancestors” w Content Security Policy.

Referencja:
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options

7.5. Ujawnienie błędów aplikacji

URL:	http://localhost:8000/api/user-plants/16/water/
Ryzyko:	Niskie
Zaufanie:	Średnie
Parametr:	-
Atak:	-
Dowód:	HTTP/1.1 500 Internal Server Error
CWE ID:	550
WASC ID:	13
Źródło:	Pasywne (90022 - Application Error Disclosure)

Opis:
Ta strona zawiera komunikat o błędzie/ostrzeżenie, który może ujawniać wrażliwe informacje, takie jak lokalizacja pliku, który wywołał nie przechwycony wyjątek. Informacje te mogą zostać wykorzystane do przeprowadzenia dalszych ataków na aplikację internetową. Alert może być fałszywie pozytywny, jeśli komunikat o błędzie znajduje się na stronie dokumentacji.

Rozwiązanie:
Przejrzyj kod źródłowy tej strony. Wdróż niestandardowe strony błędów. Rozważ wprowadzenie mechanizmu, który dostarcza unikalny identyfikator/referencję błędu klientowi (przeglądarce), jednocześnie zapisując szczegóły po stronie serwera, bez ujawniania ich użytkownikowi.

7.6. Serwer ujawnia informacje poprzez pole nagłówka odpowiedzi HTTP X-Powered-By

URL:	http://localhost:3000/
Ryzyko:	Niskie
Zaufanie:	Średnie
Parametr:	-
Atak:	-
Dowód:	X-Powered-By: Express
CWE ID:	497
WASC ID:	13
Źródło:	Pasywne (10037 - Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s))

Opis:
Serwer WWW/aplikacyjny ujawnia informacje poprzez jeden lub więcej nagłówków odpowiedzi HTTP X-Powered-By. Dostęp do takich informacji może ułatwić atakującym identyfikację innych frameworków/komponentów, na których opiera się Twoja aplikacja internetowa, oraz podatności, na które te komponenty mogą być narażone.

Rozwiązanie:
Upewnij się, że Twój serwer WWW, serwer aplikacji, równoważnik obciążenia itp. są skonfigurowane tak, aby usuwać nagłówki X-Powered-By.

Referencja:
https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/01-Information_Gathering/08-Fingerprint_Web_Application_Framework
https://www.troyhunt.com/2012/02/shhh-dont-let-your-response-headers.html

7.7. Serwer ujawnia informacje o wersji poprzez pole nagłówka odpowiedzi HTTP „Server”

URL:	http://localhost:8000/api/token/
Ryzyko:	Niskie
Zaufanie:	Wysokie
Parametr:	-
Atak:	-
Dowód:	WSGIServer/0.2 CPython/3.13.2
CWE ID:	497
WASC ID:	13
Źródło:	Pasywne (10036 - HTTP Server Response Header)

Opis:
Serwer WWW/aplikacyjny ujawnia informacje o swojej wersji w nagłówku odpowiedzi HTTP „Server”. Uzyskanie takich danych może ułatwić atakującym wykrycie innych podatności, na które narażony jest Twój serwer.

Rozwiązanie:
Upewnij się, że Twój serwer WWW, serwer aplikacyjny, load balancer itp. są skonfigurowane tak, aby pomijały nagłówek „Server” albo zwracały w nim jedynie ogólne informacje.

Referencja:
https://httpd.apache.org/docs/current/mod/core.html#servertokens
https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff648552(v=pandp.10)
https://www.troyhunt.com/shhh-dont-let-your-response-headers/

7.8. Brak nagłówka X-Content-Type-Options.

URL:	http://localhost:3000/
Ryzyko:	Niskie
Zaufanie:	Średnie
Parametr:	x-content-type-options
Atak:	-
Dowód:	WSGIServer/0.2 CPython/3.13.2
CWE ID:	693
WASC ID:	15
Źródło:	Pasywne (10021 - X-Content-Type-Options Header Missing)

Opis:
Nagłówek anty-MIME-sniffing X-Content-Type-Options nie został ustawiony na wartość „nosniff”. Powoduje to, że starsze wersje przeglądarek Internet Explorer i Chrome mogą wykonywać sniffing MIME na treści odpowiedzi, co może prowadzić do interpretacji i wyświetlenia jej jako innego typu niż zadeklarowany. Aktualne (na początek 2014 r.) oraz starsze wersje Firefoksa użyją zadeklarowanego typu treści (jeżeli został on określony), zamiast przeprowadzać sniffing MIME.

Inne informacje:
Problem ten nadal dotyczy stron błędów (401, 403, 500 itd.), ponieważ i one często są podatne na ataki typu injection; w takim przypadku wciąż istnieje ryzyko, że przeglądarki poprzez sniffing zinterpretują treść inaczej, niż wskazuje zadeklarowany typ. Przy progu „High” ta reguła skanowania nie zgłosi alertu dla odpowiedzi z błędem po stronie klienta ani serwera.

Rozwiązanie;
Upewnij się, że serwer aplikacyjny/WWW odpowiednio ustawia nagłówek Content-Type oraz że ustawia nagłówek X-Content-Type-Options na „nosniff” dla wszystkich stron internetowych. Jeśli to możliwe, upewnij się, że użytkownik końcowy korzysta z nowoczesnej, zgodnej ze standardami przeglądarki, która w ogóle nie wykonuje sniffingu MIME albo którą można poinstruować przez aplikację/serwer WWW, by nie wykonywała sniffingu MIME.

Referencja:
https://learn.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/compatibility/gg622941(v=vs.85)
https://owasp.org/www-community/Security_Headers

7.9. Ujawnienie informacji — podejrzane komentarze

URL:	http://localhost:3000/
Ryzyko:	Informacyjne
Zaufanie:	Średnie
Parametr:	-
Atak:	-
Dowód:	user
CWE ID:	615
WASC ID:	13
Źródło:	Pasywne (10027 - Information Disclosure - Suspicious Comments)

Opis:
Odpowiedź wydaje się zawierać podejrzane komentarze, które mogą pomóc atakującemu.

Inne informacje:
Następujący wzorzec został użyty: \bUSER\b i został wykryty w prawdopodobnym komentarzu: "<!--
manifest.json dostarcza metadane używane, gdy Twoja aplikacja internetowa jest instalowana na urządzeniu mobilnym lub komputerze użytkownika. Zobacz", zobacz pole evidence dla podejrzanego komentarza/fragmentu.

Rozwiązanie:
Usuń wszystkie komentarze, które ujawniają informacje mogące pomóc atakującemu, oraz napraw wszelkie leżące u podstaw problemy, do których się odnoszą.

7.10. Podsumowanie analizy

Skan OWASP ZAP wskazał, że aplikacja wymaga kilku kluczowych poprawek. Najpilniejsze to wdrożenie tokenów Anti-CSRF we wszystkich formularzach, ustawienie nagłówka Content-Security-Policy oraz zaostrzenie konfiguracji CORS, aby akceptować tylko zaufane domeny. Należy dodać ochronę przed clickjackingiem. Jako działania szybkiej naprawy należy ukryć nagłówki Server i X-Powered-By, włączyć X-Content-Type-Options: nosniff, zastąpić komunikaty błędów niestandardowymi stronami oraz usunąć komentarze HTML ujawniające szczegóły implementacyjne.

Nazwa	Poziom ryzyka	Number of Instances
Absence of Anti-CSRF Tokens	redni	4
Content Security Policy (CSP) Header Not Set	redni	31
Cross-Domain Misconfiguration	redni	56
Missing Anti-clickjacking Header	redni	1
Application Error Disclosure	Niski	4
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Niski	2
Server Leaks Version Information via "Server" HTTP Response Header Field	Niski	54
X-Content-Type-Options Header Missing	Niski	2
Authentication Request Identified	Informacyjny	2
Information Disclosure - Suspicious Comments	Informacyjny	3
Modern Web Application	Informacyjny	5
Session Management Response Identified	Informacyjny	6

Rys. 3. Zestawienie wykrytych zagrożeń

8. GitHub Dependabot

Poniżej przedstawiono podatności wykryte przez narzędzie GitHub Dependabot:

- **CVE-2025-43864**

Pakiet (wersje podatne): react-router ≥ 7.2.0 i ≤ 7.5.1

Zagrożenia (skutki w aplikacji): nagłówek X-React-Router-SPA-Mode może wymusić przełączenie renderowania serwera w tryb SPA i „zatruć" pamięć podręczną reverse-proxy; w efekcie kolejne żądania użytkowników otrzymują nieprawidłową lub pustą treść, co prowadzi do trwałego Denial-of-Service.

Ocena ryzyka: wysokie (CVSS 7.5)

- **CVE-2025-43865**

Pakiet (wersje podatne): react-router ≥ 7.0.0 i ≤ 7.5.1

Zagrożenia (skutki w aplikacji): przy włączonym SSR i loader-routes napastnik może przesłać nagłówek X-React-Router-Prerender-Data, którego zawartość zostanie zapisania w cache jako „wstępnie wyrenderowane" dane. Prowadzi to do podmiany treści strony (cache-poisoning), a w określonych scenariuszach do trwałego XSS.

Ocena ryzyka: wysokie (CVSS 8.2).

- **CWE-670**

Pakiet (wersje podatne): http-proxy-middleware < 2.0.8

Zagrożenia (skutki w aplikacji): błąd w obsłudze funkcji writeBody może doprowadzić do podwójnego zapisu odpowiedzi HTTP i zerwania połączenia, co generuje umiarkowany DoS w środowisku developerskim. Luka nie przechodzi do wersji produkcyjnej, ale destabilizuje lokalny serwer testowy.

Ocena ryzyka: umiarkowane (CVSS 4.0).

9. Przykład wdrożenia

Wybrana metoda wdrożenia aplikacji opiera się na wykorzystaniu konteneryzacji z Dockerem i Docker Compose.
Wykorzystane technologie:

1. **DigitalOcean** - oferuje łatwe uruchamianie serwerów VPS (Droplet).
2. **Docker i Docker Compose** - służy do tworzenia i zarządzania kontenerami aplikacji.

Komponenty wdrożenia:
Na jednym serwerze VPS uruchomione zostaną trzy kontenery zdefiniowane w pliku o rozszerzeniu .yml:

1. Frontend - Statyczne pliki React serwowane przez NGINX.
2. Backend - Aplikacja Django uruchomiona z serwerem Gunicorn.
3. Baza danych - PostgreSQL z trwałym wolumenem do przechowywania danych.

Dodatkowo wymagana będzie konfiguracja dodatkowych komponentów:

1. Certyfikat SSL - Skonfigurowany w NGINX za pomocą *Let's Encrypt* dla bezpiecznego połączenia HTTPS.
2. CI/CD - Pipeline w GitHub Actions, który automatycznie buduje i wdraża nową wersję aplikacji po zmianach w kodzie.

10. Podsumowanie raportu

W raporcie przeprowadzono analizę architektury aplikacji PlantCare, opisując jej podział na warstwy: frontendową, backendową oraz bazę danych, a także oceniono zastosowane technologie i wersje bibliotek. Przeanalizowano walidację danych w każdej warstwie, wskazując na ograniczenia w walidacji formularzy po stronie klienta oraz stosunkowo solidne, lecz możliwe do wzmocnienia mechanizmy walidacji w backendzie i na poziomie bazy danych. Wykonano skanowanie narzędziem OWASP ZAP, które ujawniło kluczowe problemy, takie jak brak tokenów Anti-CSRF, nieustawiony nagłówek Content-Security-Policy, nadmiernie otwarta polityka CORS, brak ochrony przed clickjackingiem, ujawnianie informacji o błędach, obecność nagłówków Server i X-Powered-By, a także przeanalizowano komentarze i nagłówki odpowiedzi HTTP pod kątem ujawniania szczegółów implementacyjnych. Dodatkowo wykorzystano GitHub Dependabot do wykrycia i opisania podatności w zależnościach frontendowych i backendowych, co podkreśla potrzebę regularnych aktualizacji i monitorowania bibliotek. Zdajemy sobie sprawę, że przedstawione działania nie stanowią pełnego, kompleksowego audytu bezpieczeństwa aplikacji webowej, a raczej wstępne, manualne i automatyczne rozpoznanie typowych wektorów ataku. Aby zwiększyć skuteczność testów bezpieczeństwa, rekomendujemy uzupełnienie procedur o testy jednostkowe w backendzie, rozszerzenie o testy dynamiczne (testy penetracyjne), bardziej dogłębną analizę kodu, przegląd kodu oraz modelowanie zagrożeń. Warto również uwzględnić testy wydajnościowe i obciążeniowe w celu oceny odporności na ataki DoS, monitorowanie logów, a także opracowanie planu reagowania na incydenty i procedur aktualizacji środowiska. Takie działania zwiększą pewność, że aplikacja jest bardziej odporna na ataki oraz że nowe zmiany nie wprowadzają regresji w obszarze bezpieczeństwa.