

UNIwersYTET RZESZOWSKI
WYDZIAŁ NAUK ŚCISŁYCH I TECHNICZNYCH
INSTYTUT INFORMATYKI



Michał Nocek
134951

Informatyka

System zarządzania gospodarstwem

Praca projektowa

Praca wykonana pod kierunkiem
mgr inż. Ewa Żesławska

Rzeszów 2025

Spis treści

1. Streszczenie w języku polskim i angielskim	6
1.1. Streszczenie w języku polskim.....	6
1.2. Streszczenie w języku angielskim	6
2. Opis założeń projektu	7
2.1. Wymagania funkcjonalne projektu	7
2.2. Wymagania нефункционалне projektu	8
3. Opis struktury projektu	9
3.1. Architektura i wykorzystane technologie	9
3.2. Zarządzanie danymi – baza danych.....	10
3.3. Hierarchia klas i kluczowe metody	10
3.4. Minimalne wymagania programu.....	13
4. Harmonogram realizacji projektu	14
4.1. Napotkane trudności w trakcie realizacji projektu	14
4.2. Repozytorium i system kontroli wersji.....	15
5. Prezentacja warstwy użytkowej projektu.....	16
5.1. Ekran logowania i rejestracji	16
5.2. Główne menu - zarządzanie polami	17
5.3. Dodawanie nowego pola.....	18
5.4. Zarządzanie uprawami dla pola.....	19
5.5. Dodawanie nowej uprawy	20
5.6. Zarządzanie zabiegami dla uprawy	21
5.7. Dodawanie nowego zabiegu	22
6. Podsumowanie	23
Spis rysunków	24

1. Streszczenie w języku polskim i angielskim

1.1. Streszczenie w języku polskim

Dokument ten to opis systemu, który pomaga rolnikom w zarządzaniu gospodarstwem rolnym. Głównym celem tej aplikacji było uproszczenie zapisu danych na temat zabiegów agrotechnicznych oraz podliczenie zysków i strat na danej uprawie.

Cała aplikacja opiera się na języku Java z wykorzystaniem biblioteki JavaFX do stworzenia GUI, aby użytkownik mógł w prosty sposób przeglądać informacje na temat danej uprawy, dodawać kolejne zabiegi oraz obliczać zyski i straty.

1.2. Streszczenie w języku angielskim

This document is a description of a system that helps farmers manage their farms. The main goal of this application was to simplify the recording of data on agrotechnical treatments and to calculate profits and losses for a given crop.

The entire application is based on Java, using the JavaFX library to create a GUI so that the user can easily view information about a given crop, add further treatments and calculate profits and losses.

2. Opis założeń projektu

Celem projektu jest stworzenie systemu informatycznego do zarządzania działalnością rolniczą, ułatwiającego monitorowanie zabiegów agrotechnicznych i analizę finansową upraw.

Rolnicy często borykają się z problemem nieuporządkowanych danych dotyczących pól, zabiegów a także finansów. Ręczna ewidencja jest czasochłonna oraz nieintuicyjna, co utrudnia ocenę opłacalności upraw. Brak narzędzia do gromadzenia tych informacji jest kluczowym problemem. Jest to ważne, ponieważ precyzyjne dane są kluczowe do podejmowania odpowiednich decyzji biznesowych.

Realizacja projektu wymagała wykorzystania technologii Java, relacyjnej bazy danych opartej na silniku PostgreSQL oraz systemu kontroli wersji Git. Wymagana była znajomość języka Java wraz z bibliotekami JavaFX oraz JDBC, projektowania baz danych oraz tworzenia interfejsów graficznych.

Problem został rozwiązany przez stworzenie aplikacji desktopowej będącej miejscem do zarządzania danymi upraw. Proces tworzenia obejmował projektowanie baz danych, architektury oprogramowania, implementację logiki i interfejsu użytkownika, a także testowanie. Wynikiem prac jest aplikacja komputerowa umożliwiająca efektywne zarządzanie polami uprawnymi, zabiegami oraz podliczanie kosztów i wyliczanie opłacalności.

2.1. Wymagania funkcjonalne projektu

- Zarządzanie kontami użytkowników
 - Użytkownik ma możliwość zarejestrowania nowego konta w systemie
 - użytkownik ma możliwość zalogowania się do systemu za pomocą loginu i hasła
- Zarządzanie polami uprawnymi
 - Użytkownik może dodać nowe pola, ustawiając ich nazwę, powierzchnię oraz lokalizację
 - System wyświetla listę pól należących do zalogowanego użytkownika
 - Użytkownik może usuwać istniejące pola
- Zarządzanie uprawami:
 - Użytkownik może dodać nowe uprawy do wybranego pola - nazwa, data siewu (obowiązkowa), data zbioru, zarobek (opcjonalnie)
 - System wyświetla listę upraw przypisanych do wybranego pola
 - Użytkownik może usuwać istniejące uprawy
- Zarządzanie zabiegami agrotechnicznymi
 - Użytkownik może dodawać nowe zabiegi do wybranej uprawy (nazwa, typ, data, koszt, opcjonalny zarobek)
 - System wyświetla listę zabiegów przypisanych do wybranej uprawy.
 - Użytkownik może usuwać istniejące zabiegi

- Automatyczna kalkulacja zysku
 - System na bieżąco będzie obliczał i aktualizował całkowity zysk dla każdej uprawy na podstawie sumy kosztów i przychodów z przypisanych do niej zabiegów
- Trwały zapis danych
 - Wszystkie dane będą trwale przechowywane w relacyjnej bazie danych PostgreSQL.
- Prezentacja danych
 - Dane będą wyświetlane w czytelnych tabelach w interfejsie użytkownika

2.2. Wymagania niefunkcjonalne projektu

System musi spełniać następujące kryteria jakościowe i ograniczenia:

- Użyteczność
 - Interfejs użytkownika powinien być intuicyjny i prosty w obsłudze. Nawigacja między poszczególnymi modułami aplikacji powinna być spójna i zrozumiała dla użytkowników
- Niezawodność
 - Aplikacja powinna działać stabilnie, minimalizując występowanie błędów krytycznych. W przypadku błędów takich jak niepoprawne dane wprowadzone do interfejsu system powinien wyświetlić zrozumiałe komunikaty.
- Wydajność
 - Kluczowe operacje takie jak logowanie, dodawanie, przeglądanie i usuwanie rekordów powinno być wykonywane w czasie nieprzekraczającym 2-3 sekundy. Czas uruchomienia aplikacji powinien się mieścić w granicach 5 sekund.
- Bezpieczeństwo
 - Dostęp do danych użytkownika jest dostępny wyłącznie po pomyślnym zalogowaniu się. Każdy użytkownik ma dostęp do danych przypisanych do jego konta.
- Środowisko
 - Aplikacja jest przeznaczona do uruchomienia na platformach wspierających Java z zainstalowanym środowiskiem JDK oraz dostępem do bazy danych PostgreSQL
- Integralność danych
 - System zapewni spójność danych np. poprzez kaskadowe usuwanie powiązanych rekordów (Usunięcie pola skutkuje usunięciem przypisanych do niego upraw i zabiegów)
- Utrzymywalność
 - Kod źródłowy aplikacji jest napisany z zastosowaniem zasad programowania obiektowego, dobrych praktyk kodowania, co ułatwi przyszłe modyfikacje i rozbudowę systemu.

3. Opis struktury projektu

W tym rozdziale opiszę, jak zbudowany jest cały system – jakie technologie zostały użyte, jak zorganizowane są dane, opis bazy danych, najważniejsze części kodu, czyli klasy i kluczowe metody.

3.1. Architektura i wykorzystane technologie

Mój projekt to aplikacja desktopowa. Została zaprojektowana w taki sposób, żeby jej budowa była jasna i modułowa – można ją podzielić na 3 główne części:

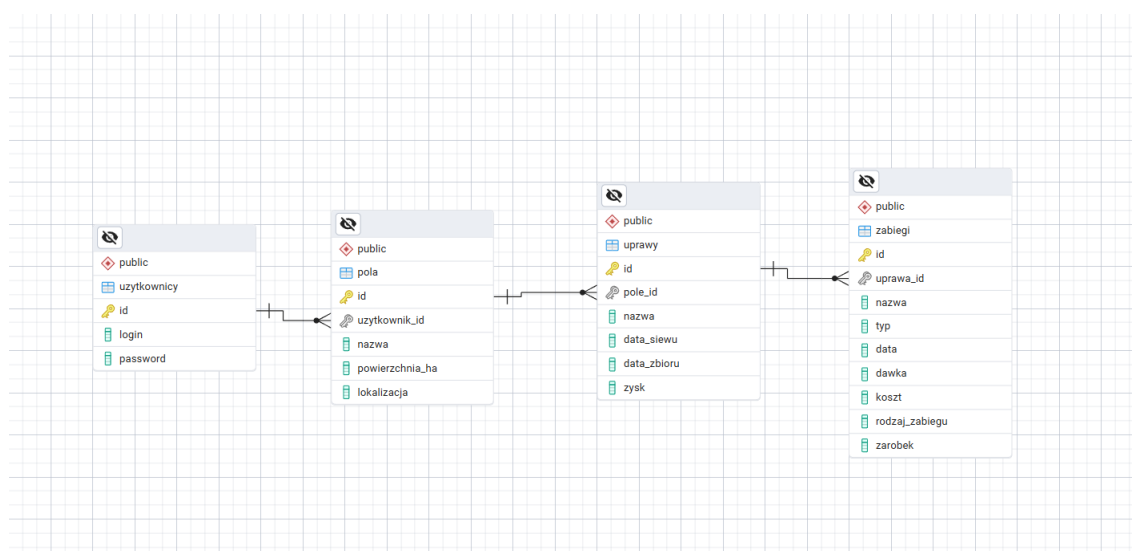
- Interfejs użytkownika
 - Do stworzenia interfejsu użytkownika została użyta biblioteka JavaFX. Pozwala ona na tworzenie intuicyjnych i dobrze wyglądających ekranów, np.: logowania, przeglądania pól czy dodawania zabiegów. Za obsługę scen odpowiadają klasy, które przekazują polecenia dalej do logiki programu.
- Logika programu
 - W warstwie logiki programu sprawdzane są dane, które trafiają do bazy danych. Tutaj są też wszystkie reguły dotyczące dodawania, zmieniania czy usuwania informacji. Co najważniejsze – to właśnie ta warstwa odpowiada za liczenie zysków z upraw.
- Obsługa bazy danych
 - Ta część dba o to, aby program mógł komunikować się z bazą danych – wysyłać do niej dane i z niej je pobierać. Do tego celu został wykorzystany standard JDBC. Dzięki temu, że ta część jest oddzielna, w bardzo prosty sposób można zmienić bazę danych na inną.

Cały program został napisany w języku Java SDK 23. Do kontrolowania zmian w kodzie i zarządzania wersjami projektu został użyty system Git, a kod jest dostępny w publicznym repozytorium.

3.2. Zarządzanie danymi – baza danych

Wszystkie informacje o użytkownikach, polach, uprawach, zabiegach są przechowywane w bazie danych PostgreSQL, wybranej ze względu na niezawodność. Baza składa się z 4 tabel:

- `uzytkownicy` – przechowuje dane kont użytkowników (`id`, `login`, `hasło`),
- `pola` – informacje o polach uprawnych (`nazwa`, `powierzchnia_ha`, `lokalizacja`), powiązane z użytkownikiem,
- `uprawy` – szczegóły upraw (`id`, `pole_id`, `nazwa`, `data_siewu`, `data_zbioru`, `zyski`), powiązane z polem poprzez `pole_id`,
- `zabiegi` – dane o wykonywanych zabiegach (`id`, `uprawa_id`, `nazwa`, `typ`, `data`, `dawka`, `koszt`, `rodzaj_zabiegu`, `zarobek`), powiązane z uprawą poprzez `uprawa_id`.



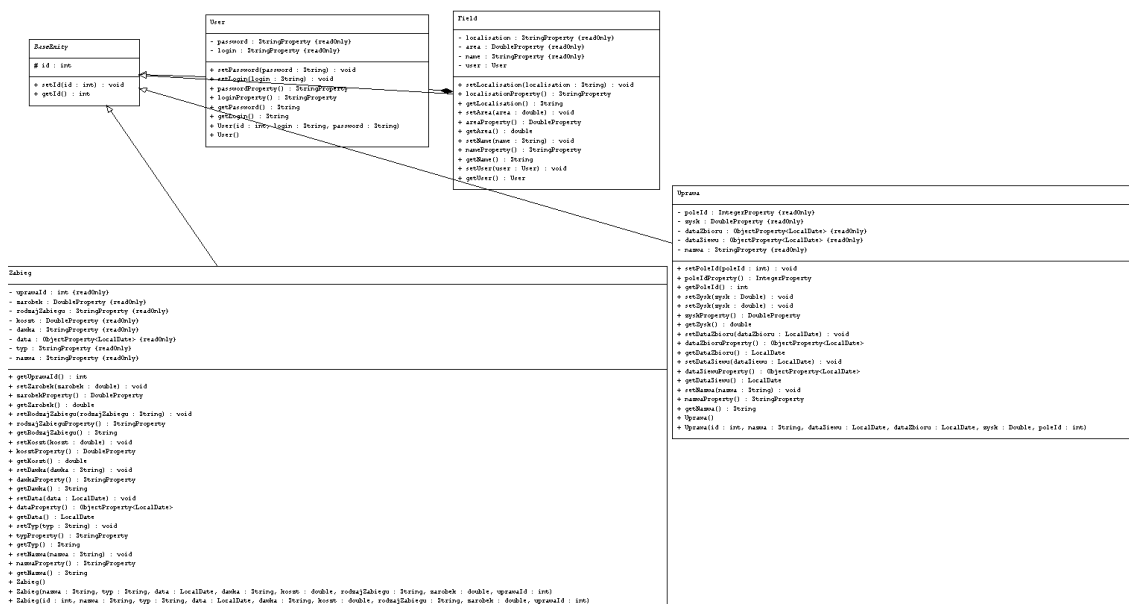
Rys. 3.1. Diagram ERD.

3.3. Hierarchia klas i kluczowe metody

Projekt opiera się na programowaniu obiektowym, wykorzystuje klasy zorganizowane w pakietach `objects`, `sceneFunction` oraz `jdbc` w celu zapewnienia przejrzystości oraz modularności projektu.

- Pakiet `objects` stanowi podstawową część architektury systemu, grupując klasy odpowiadające głównym encjom w bazie danych, którymi operuje aplikacja. Zaprojektowanie tych klas umożliwia logiczne odwzorowanie rzeczywistych obiektów i procesów w rolnictwie, zapewniając spójność danych i ułatwia zarządzanie nimi. Kluczowym elementem klas zawartych w tym pakiecie jest zastosowanie hierarchii dziedziczenia, co ułatwia posługiwanie się nimi oraz umożliwia łatwą rozbudowę programu w przyszłości.
 - W centrum hierarchii dziedziczenia jest klasa `BaseEntity`, takie podejście gwarantuje, że wszystkie klasy dziedziczące posiadają podstawowe właściwości i metody, w tym unikalny identyfikator `id`. Jest to klasa abstrakcyjna stanowiąca bazę dla innych klas.

- Klasa `User` dziedziczy po `BaseEntity`. Przechowuje dane uwierzytelniające takie jak `login` oraz `password`, zaimplementowane jako `StringProperty` z `JavaFX`, co ułatwia wiązanie danych z elementami interfeju użytkownika.
- Klasa `Field` dziedziczy po `BaseEntity` - klasa ta reprezentuje pole uprawne w gospodarstwie rolnym. Zawiera takie atrybuty jak `name`, `area`, `localisation` oraz referencje do obiektu `User`, który jest właścicielem pola.
- Klasa `Uprawa` dziedziczy po `BaseEntity`. Jest to klasa reprezentująca konkretną uprawę na danym polu. Przechowuje informacje takie jak `nazwa`, `data siewu`, `data zbioru`, `zysk`, `poleId`.
- Klasa `Zabieg` dziedziczy po `BaseEntity` - klasa reprezentuje zabieg agrotechniczny wykonywany w ramach konkretnej uprawy. Zawiera szczegółowe dane, takie jak `nazwa`, `typ`, `data`, `dawka`, `koszt`, `rodzaj zabiegu`, `zarobek`, `uprawa ID`.
- Klasa `SessionManager` jest klasą statyczną odpowiedzialną za zarządzanie sesją użytkownika. Przechowuje identyfikator zalogowanego użytkownika, co pozwala na globalny dostęp do tej informacji w aplikacji.



Rys. 3.2. Diagram UML .

- Pakiet `sceneFunction` zawiera kontrolery `javaFX`, które są odpowiedzialne za logikę widoków w aplikacji i obsługę interakcji użytkownika z graficznym interfejsem. Każdy kontroler jest powiązany z odpowiadającym mu plikiem `FXML` i zarządza konkretnym fragmentem interfejsu, obsługując zdarzenia oraz aktualizując widok na podstawie danych. Dzięki temu jest wyraźna oddzielona logika prezentacji od logiki biznesowej co ułatwia rozbudowę oraz poruszanie się po kodzie źródłowym aplikacji.
 - `LoginController` odpowiada za ekran logowania i rejestrację użytkowników. Obsługuje weryfikację danych oraz inicjuje sesję użytkownika po pomyślnym uwierzytelnieniu.

- `MainController` - jest to główny kontroler aplikacji po zalogowaniu, stanowiący centralny punkt nawigacji i zarządzania polami uprawnymi. Wyświetla listę pól przypisanych do aktualnie zalogowanego użytkownika. Kluczowymi funkcjonalnościami jest wyświetlanie pól i zarządzanie nimi. pozwala również na przejście do widoku szczegółowych upraw dla wybranego pola, przekazując niezbędne dane o polu do kolejnego kontrolera. Ten kontroler umożliwia również wylogowanie się danego użytkownika i powrót do sceny z logowaniem.
 - `AddFieldMenuController` - jest to dedykowany do obsługi formularza, który pozwala użytkownikowi na dodanie nowego pola uprawnego do systemu. Ta klasa zapewnia również walidację danych poprawnych przez użytkownika przed ich zapisem.
 - `UprawaController` - jest to kontroler zarządzający sceną odpowiedzialną za wyświetlenie listy upraw. Klasa ta ma za zadanie również umożliwić dodanie nowych upraw oraz przejście do widoku z szczegółowymi zabiegami dla danej uprawy. Umożliwia również powrót do sceny z głównym widokiem pól.
 - `ZabiegiMenuController` - jest to kontroler do wyświetlania i zarządzania zabiegami agrotechnicznymi. Pozwala również na dodawanie nowych zabiegów oraz usuwanie istniejących.
 - `AddZabiegMenuController` - ten kontroler obsługuje formularz dodawania nowego zabiegu dla wybranej uprawy. Jest kluczowy dla rejestrowania zabiegów dla upraw. Posiada funkcje do walidacji danych otrzymanych od użytkownika oraz przysyła poprawne dane do metod CRUD
- Pakiet `jdbc` stanowi fundamentalną warstwę dostępu do danych w architekturze aplikacji. Jego głównym zadaniem jest zarządzanie interakcjami z bazą danych oraz dostarczenie mechanizmów do wykonywania operacji na danych. Jest to warstwa abstrakcji, która oddziela logikę biznesową i interfejs użytkownika od szczegółów połączeń z bazą. Dzięki temu projekt staje się bardziej modułowy, łatwiejszy w utrzymaniu oraz potencjalnie elastyczny na zmiany technologii bazodanowych w przyszłości. Pakiet ten skupia się na zapewnieniu integralności danych i efektywnego zarządzania zasobami bazodanowymi
 - Klasa `DBConnection` jest dedykowana do zarządzania pojedynczym połączeniem z bazą danych PostgreSQL. Jej głównym celem jest dostarczenie statycznej metody do uzyskania aktywnego połączenia, co eliminuje konieczność wielokrotnego konfigurowania parametrów połączenia w różnych miejscach kodu. Implementuje ona wzorzec fabryki połączeń, gdzie każde wywołanie `getConnection()` zwraca nowe, niezależne połączenie z bazą danych, używając predefiniowanych stałych dla URL-a bazy danych, nazwy użytkownika i hasła. Klasa ta jest również odpowiedzialna za wczytanie sterownika JDBC dla PostgreSQL
 - Klasa `CrudOperation` to kompleksowa implementacja operacji CRUD dla wszystkich kluczowych encji aplikacji. Działa jako warstwa DAO hermetyzując całą logikę interakcji z bazą danych za pomocą zapytań SQL. Klasa ta zapewnia integralność danych poprzez walidację oraz obsługę relacji między encjami. Wszystkie metody zgłaszają `SQLException`, co pozwala warstwie wyżej na odpowiednie reagowanie i wyświetlanie komunikatów użytkownikowi.
 - Klasa `HellApplication` znajduje się w głównym pakiecie aplikacji i pełni rolę punktu startowego dla całego systemu. Jest to klasa wymagana przez środowisko JavaFX do uruchomienia aplikacji graficznej. Odpowiada za wczytanie pliku `login.fxml` i wyświetlenie ekranu ładowania jako pierwszego okna aplikacji.

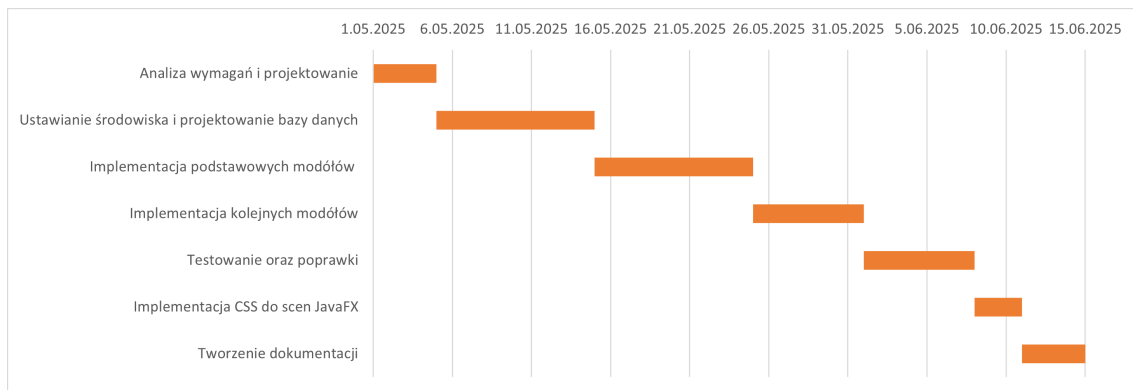
3.4. Minimalne wymagania programu

- Minimalne wymagania programowe:
 - * **System operacyjny:** Kompatybilny z JavaFX i JDK 23 (np. Windows 10/11)
 - * **JDK:** Wersja 23 lub nowsza wraz z wsparciem dla modułów JavaFX
 - * **PostgreSQL Server:** Lokalny lub zdalny serwer PostgreSQL (wersja 13 lub nowsza jest zalecana, ale kompatybilność zapewniona przez sterowniki 42.7.2)
 - * **OpenJFX:** Wersja 23.0.1
- Dodatkowe narzędzia
 - * **IDE:** Dowolne środowisko programistyczne wspierające rozwój aplikacji Java i JavaFX (np. IntelliJ IDEA, Eclipse, Visual Studio Code).
 - * **Apache Maven:** Wersja 3.13.0 - System automatyzacji budowy projektu i zarządzania zależnościami
 - * **System kontroli wersji:** Git
 - * **Narzędzia do zarządzania bazą danych:** gpAdmin 4 lub inny

4. Harmonogram realizacji projektu

W tym rozdziale przedstawiono harmonogram realizacji projektu, wizualizujący poszczególne etapy pracy oraz ich ramy czasowe. Wykorzystanie diagramu Gantta pozwoliło na przejrzyste przedstawienie planowania i postępu prac.

- **Planowanie i projektowanie:** Faza początkowa obejmująca analizę wymagań oraz projektowanie ogólnej struktury aplikacji i bazy danych
- **Ustawienie środowiska i bazy danych:** Etap konfiguracji niezbędnego oprogramowania i utworzenia schematu bazy danych
- **Implementacja modułów podstawowych:** Rozwój podstawowych funkcji systemu, takich jak logowanie, rejestracja oraz zarządzanie polami uprawnymi.
- **Implementacja modułów szczegółowych:** Tworzenie bardziej zaawansowanych funkcji, w tym zarządzania uprawami i zabiegami rolniczymi.
- **Testy i poprawki:** Faza poświęcona weryfikacji poprawności działania aplikacji oraz eliminacji napotkanych błędów.
- **Tworzenie dokumentacji:** Proces przygotowywania dokumentacji, prowadzony częściowo równoległe z pracami implementacyjnymi.



Rys. 4.1. Diagram Gantta przedstawiający harmonogram realizacji projektu.

4.1. Napotkane trudności w trakcie realizacji projektu

Podczas realizacji projektu napotkano na kilka typowych trudności, które wymagały dodatkowego czasu i uwagi:

- **Ustawienie środowiska:** Na początku pracy było trochę kłopotu ze skonfigurowaniem środowiska, żeby JavaFX dobrze współpracowała z Maven.
- **Łączenie z bazą danych:** Ważne było, żeby program sprawnie pobierał i zapisywał dane do bazy. Trzeba było kontrolować, aby połączenie z bazą danych było prawidłowo zamknięte.

- **Sprawdzenie danych:** Aby program działał poprawnie ważne było, aby program sprawdzał dane podane przez użytkownika. Wymagało to dodania do programu mechanizmów, które sprawdzają czy dane są poprawne.
- **Połączenie wszystkich części:** Aplikacja składa się z wielu różnych ekranów i funkcji. Wyzwaniem było sprawdzenie, żeby wszystkie te elementy ze sobą współpracowały i przekazywały sobie dane.

4.2. Repozytorium i system kontroli wersji

Kod źródłowy tego projektu jest przechowywany i zarządzany za pomocą systemu Git. Dostęp do niego jest możliwy poprzez publiczne repozytorium na Github. Korzystanie z GitHub pozwoliło na śledzenie wszystkich zmian w kodzie oraz zarządzanie różnymi wersjami projektu.

Adres repozytorium projektu:

<https://github.com/MichalNocek/ProjektPO1>

5. Prezentacja warstwy użytkowej projektu

W tym rozdziale przedstawię wizualną stronę aplikacji poprzez zrzuty ekranu, ilustrujące kluczowe ekrany i interakcje użytkownika. Celem jest zaprezentowanie funkcjonalności systemu z perspektywy użytkownika końcowego.

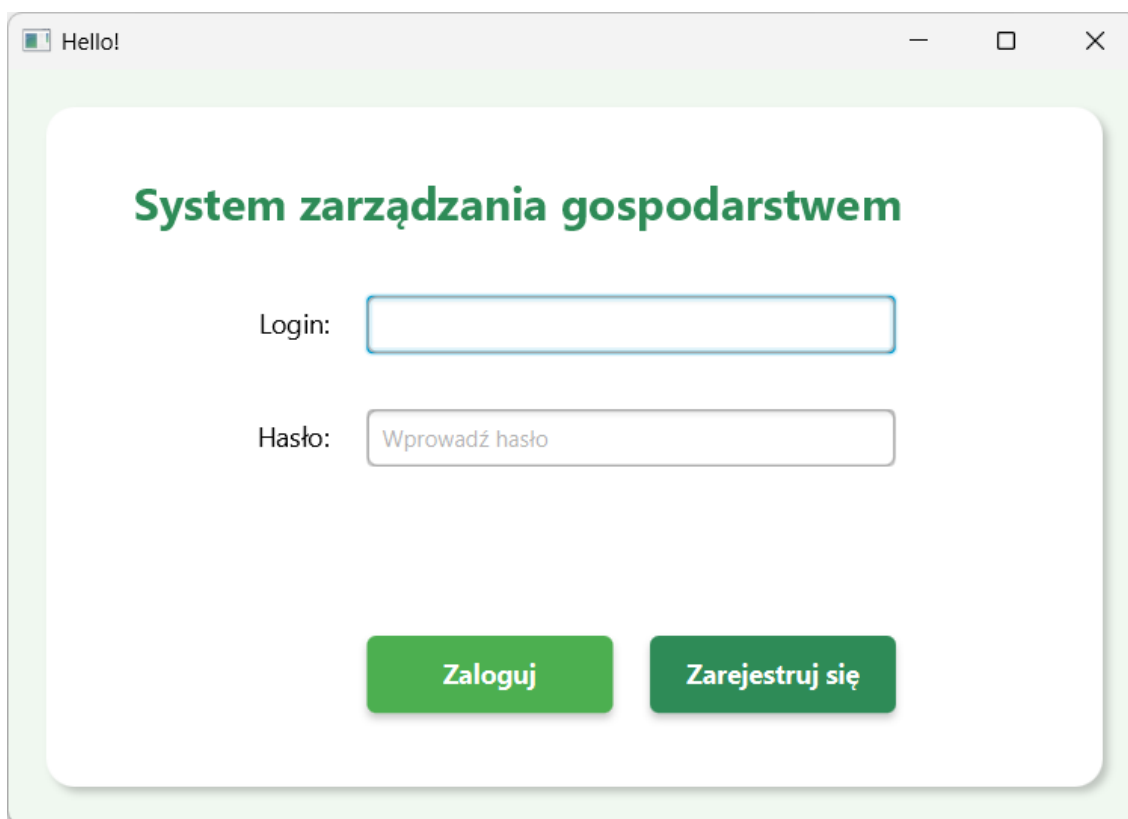
5.1. Ekran logowania i rejestracji

Pierwszym ekranem, z którym styka się użytkownik, jest ekran logowania, umożliwiający dostęp do systemu dla zarejestrowanych użytkowników lub stworzenie nowego konta.

Na rysunku 5.1 przedstawiono interfejs użytkownika, który składa się z:

- Pola tekstowego `Login`, służącego do wprowadzenia nazwy użytkownika.
- Pola `Hasło`, zapewniającego maskowanie wpisanego hasła.
- Przycisku `Zaloguj się`, który po kliknięciu weryfikuje podane dane z bazą danych. W przypadku poprawnych danych, użytkownik zostaje przekierowany do głównego menu aplikacji.
- Przycisku `Zarejestruj się`, który umożliwia nowym użytkownikom utworzenie konta w systemie po podaniu loginu i hasła.
- Pola tekstowego `errorText`, które wyświetla komunikaty zwrotne dla użytkownika, takie jak *"Niepoprawny login lub hasło!"* lub *"Użytkownik zarejestrowany! Teraz możesz się zalogować"*.

Dzięki temu użytkownik może w łatwy sposób uzyskać dostęp do funkcjonalności aplikacji albo założyć nowe konto.



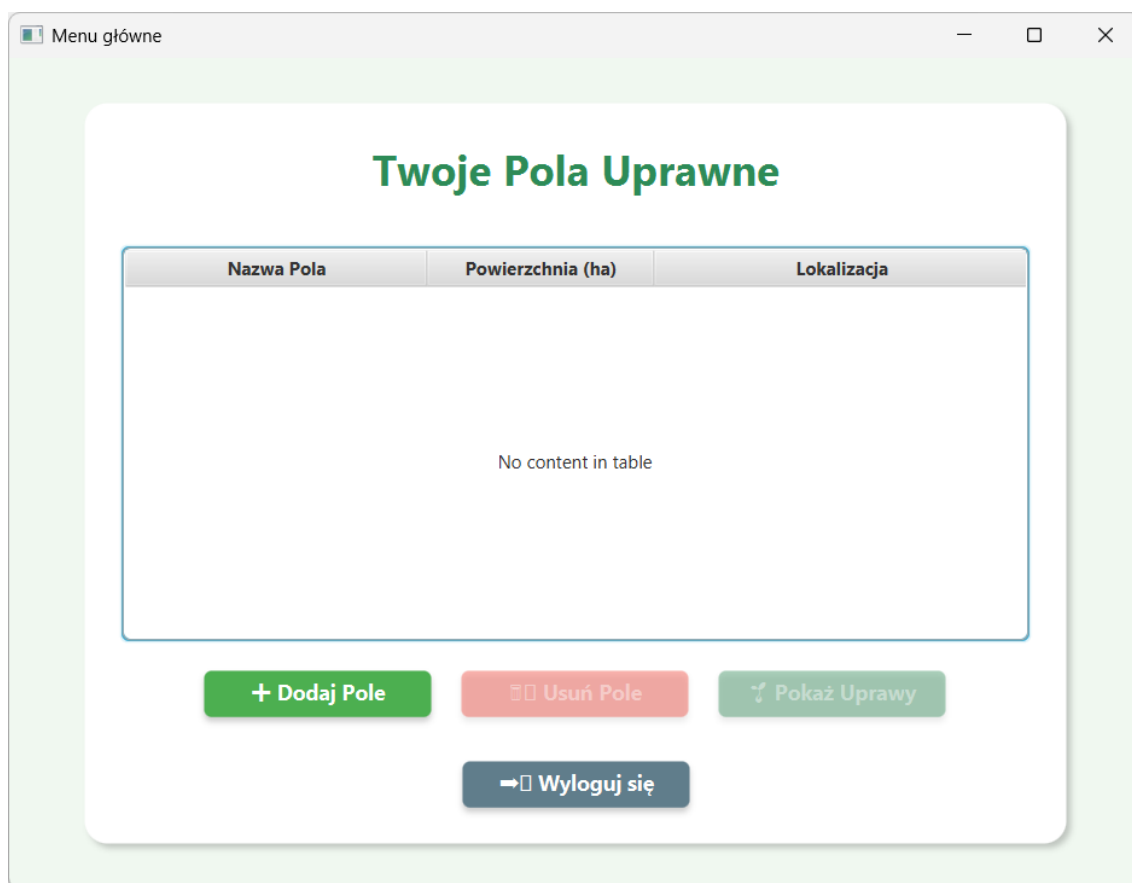
The screenshot shows a web application window titled "Hello!". The main heading is "System zarządzania gospodarstwem" in green. Below the heading are two input fields: "Login:" and "Hasło:". The "Hasło:" field has a placeholder text "Wprowadź hasło". At the bottom, there are two green buttons: "Zaloguj" and "Zarejestruj się".

Rys. 5.1. Ekran logowania i rejestracji.

5.2. Główne menu - zarządzanie polami

Po pomyślnym zalogowaniu, użytkownik zostaje przeniesiony do głównego menu aplikacji, które wyświetla listę pól uprawnych użytkownika przez co umożliwia dodawanie i usuwanie przypisanych pól do użytkownika

- Tabela pól uprawnych wyświetla listę wszystkich pól należących do użytkownika.
- Przycisk **Dodaj pole** umożliwia otwarcie nowego formularza, w którym użytkownik może wprowadzić dane nowego pola uprawnego
- Przycisk **Usuń pole** umożliwia usunięcie zaznaczonego pola
- Przycisk **Pokaż uprawy** po wybraniu konkretnego pola z tabeli umożliwia przeniesienie się do widoków szczegółu uprawy dla tego pola, gdzie można zarządzać uprawami
- Przycisk **Wyloguj się** umożliwia bezpieczne wylogowanie się z aplikacji i powrót do ekranu logowania



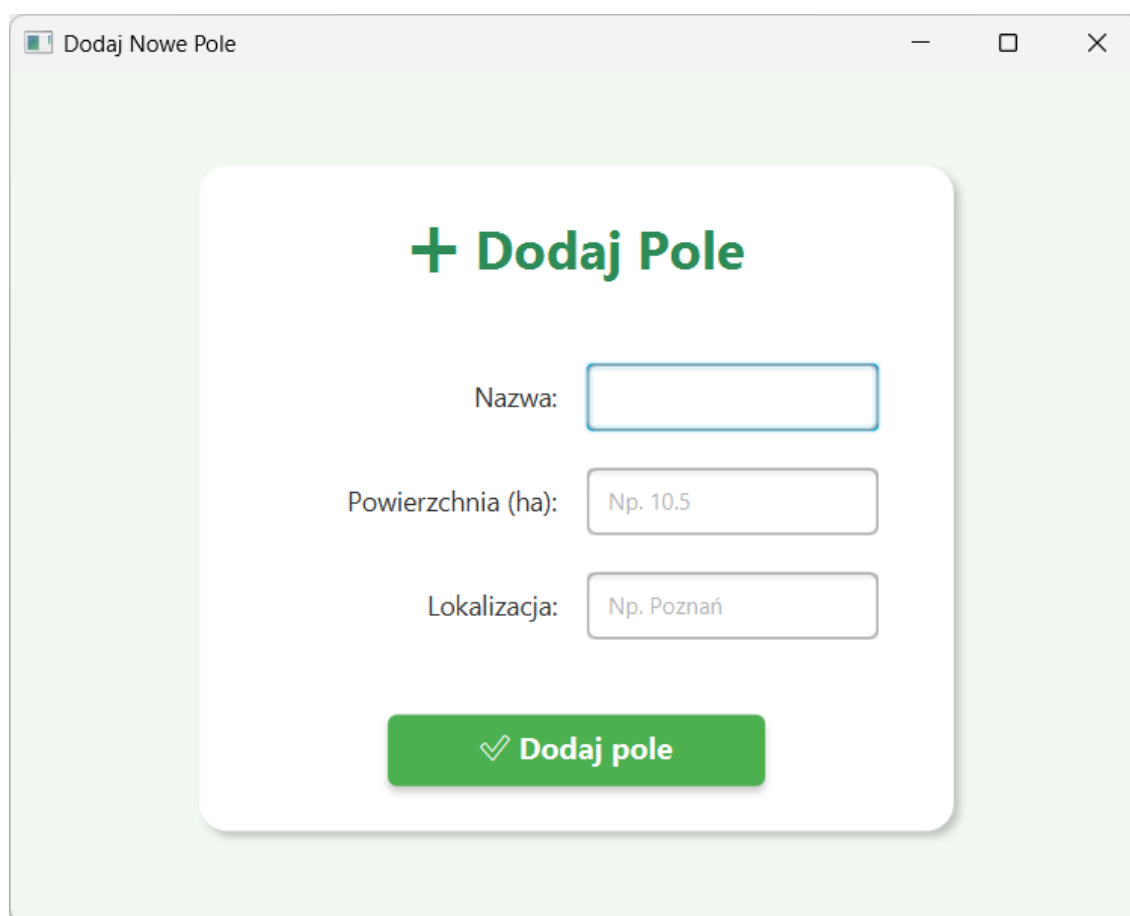
Rys. 5.2. Menu główne.

5.3. Dodawanie nowego pola

Aby umożliwić dodawanie pól, aplikacja udostępnia dedykowane okno do dodawania nowych upraw.

- Pole tekstowe `Nazwa pola` służy do wprowadzenia unikalnej nazwy identyfikującej dane pole
- Pole tekstowe `Powierzchnia` przeznaczona jest do wpisania powierzchni pola. Aplikacja waliduje wprowadzone dane czy na pewno są liczbą
- Przycisk `Dodaj pole`: po wypełnieniu wszystkich wymaganych pól, kliknięcie tego przycisku spowoduje zapisanie nowego pola do bazy danych. W przypadku błędów walidacji użytkownik otrzymuje stosowny komunikat.

Okno to umożliwia prosty sposób dodania nowego pola do systemu.



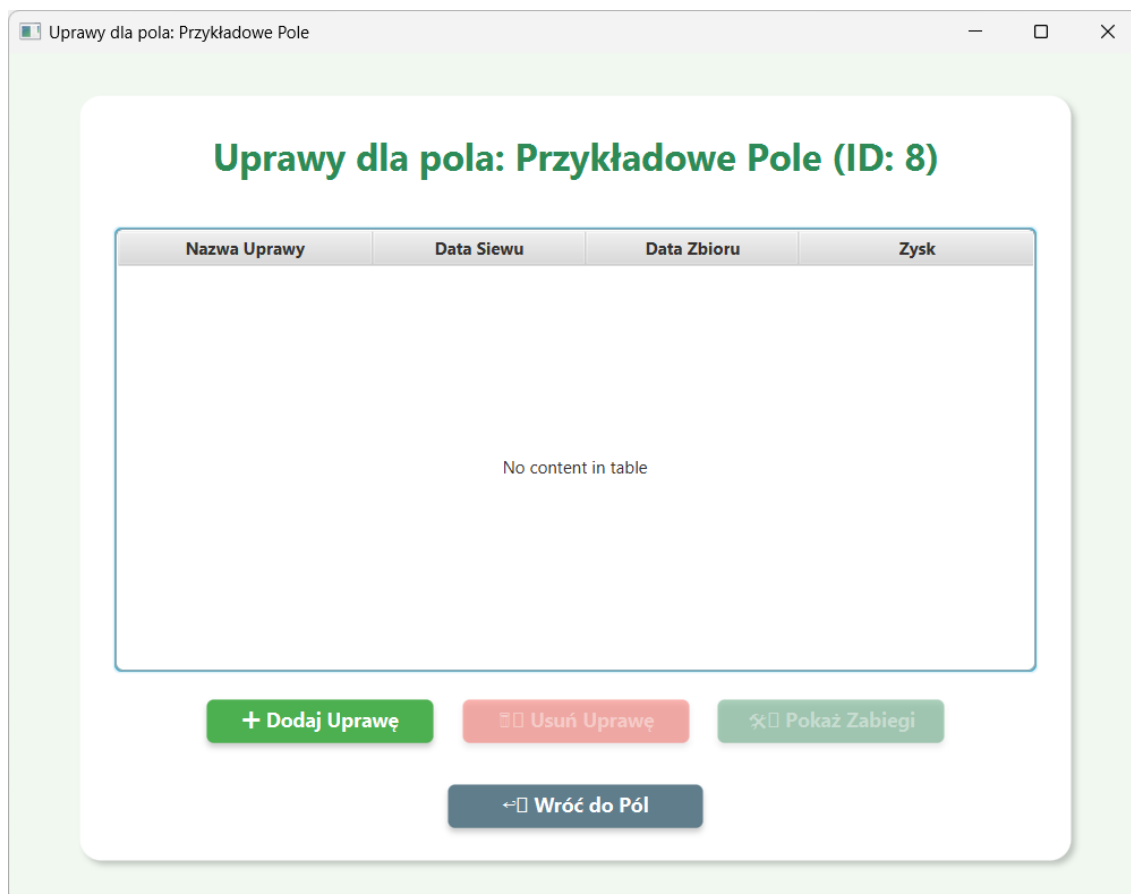
Rys. 5.3. Formularz dodawania nowego pola

5.4. Zarządzanie uprawami dla pola

Po wybraniu konkretnego pola z głównego menu i kliknięciu przycisku Pokaż uprawy, użytkownik zostaje przekierowany do szczegółowego widoku zarządzania uprawami dla tego pola. Ten widok umożliwia przeglądanie, dodawanie oraz usuwanie upraw prowadzonych w wybranym polu.

- Nagłówek z nazwą pole Uprawa dla pola: Przykładowe pole informuje użytkownika, dla którego pola wyświetlane są dane.
- Tabela upraw wyświetla listę wszystkich upraw przypisanych do wybranego pola. Dane są ładowane z bazy danych.
- Przycisk Dodaj uprawę otwiera formularz umożliwiający wprowadzenie szczegółów nowej uprawy.
- Przycisk Usuń uprawę pozwala na usunięcie wybranej uprawy z tabeli i bazy danych. Przed usunięciem wymagane jest potwierdzenie
- Przycisk Cofnij umożliwia powrót do głównego menu zarządzania polami.

Ten widok zapewnia kompleksowe zarządzanie uprawami na danym polu.



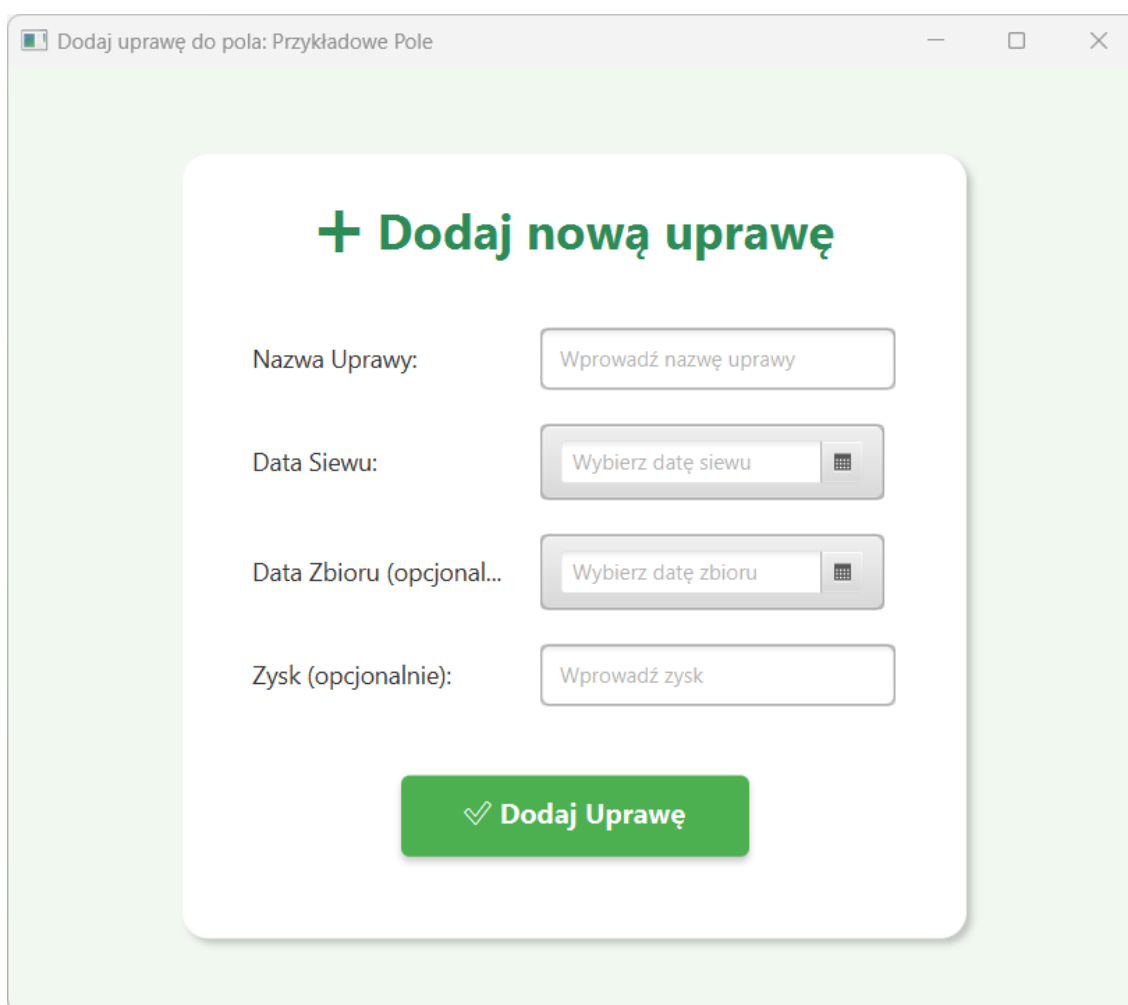
Rys. 5.4. Widok z listą upraw dla danego pola

5.5. Dodawanie nowej uprawy

Aby dodać nową uprawę do wybranego pola, użytkownik korzysta z dedykowanego formularza dostępnego po kliknięciu przycisku **Dodaj uprawę** w widoku zarządzania uprawami.

- Pole tekstowe **Nazwa uprawy** służy do wprowadzenia nazwy identyfikującej konkretną uprawę.
- Pole wyboru daty **Data siewu** służy do określenia daty siewu uprawy. Data jest wybierana z kalendarza, co minimalizuje ryzyko błędów formatowania.
- Pole **Data zbioru** umożliwia określenie przewidywanej daty zbioru.
- Pole **Zysk** jest opcjonalne i umożliwia wprowadzenie szacunkowego zysku na danej uprawie.
- Przycisk **Dodaj uprawę** umożliwia zapisanie nowej uprawy i przypisanie jej aktualnie wybranego pola. Po udanym dodaniu okno jest zamykane, a lista upraw odświeżana.

Ten intuicyjny formularz zapewnia łatwe i szybkie wprowadzanie danych o nowych uprawach.



The screenshot shows a web application window titled "Dodaj uprawę do pola: Przykładowe Pole". Inside the window is a light green card with the heading "+ Dodaj nową uprawę". Below the heading are four form fields: "Nazwa Uprawy:" with a text input containing the placeholder "Wprowadź nazwę uprawy"; "Data Siewu:" with a date picker showing "Wybierz datę siewu"; "Data Zbioru (opcjonalnie...)" with a date picker showing "Wybierz datę zbioru"; and "Zysk (opcjonalnie):" with a text input containing the placeholder "Wprowadź zysk". At the bottom of the card is a large green button with a white checkmark icon and the text "Dodaj Uprawę".

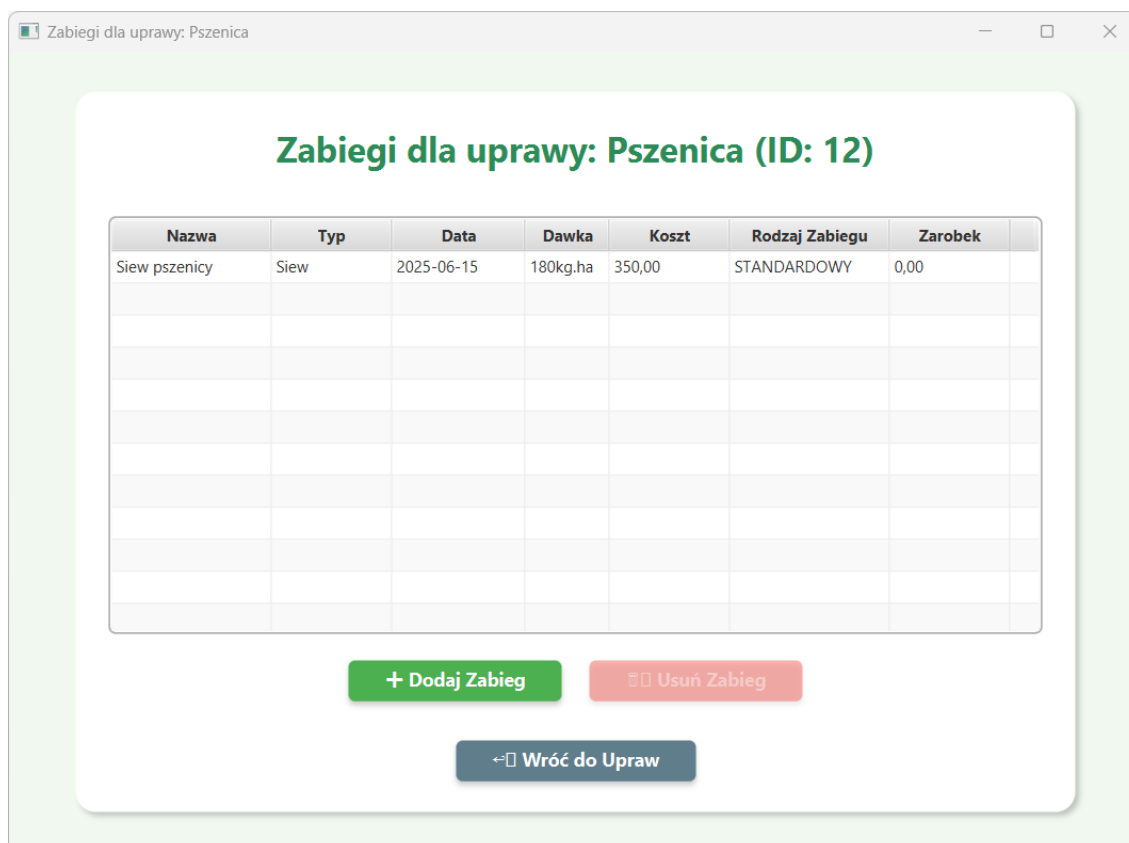
Rys. 5.5. Formularz dodawania uprawy dla danego pola

5.6. Zarządzanie zabiegami dla uprawy

Po wybraniu konkretnej uprawy z listy i kliknięciu przycisku **Pokaż zabiegi**, użytkownik uzyskuje dostęp do szczegółowego widoku zarządzania wszystkimi zabiegami przypisanymi do tej uprawy.

- Nagłówek z nazwą uprawy jednoznacznie wskazuje, której uprawy dane są wyświetlane.
- Tabela zabiegów prezentuje listę wszystkich zabiegów związanych z wybraną uprawą. Dane te są pobierane z bazy danych oraz można je sortować według kolumn.
- Przycisk **Dodaj zabieg** umożliwia otwarcie formularza do wprowadzania danych o nowym zabiegu.
- Przycisk **Usuń zabieg** służy do usuwania wybranego zabiegu z tabeli i bazy danych po potwierdzeniu.
- Przycisk **Cofnij** umożliwia powrót do poprzedniego widoku, czyli zarządzania uprawami.

Widok ten stanowi kluczowe narzędzie do szczegółowego dokumentowania i analizy czynności wykonywanych w ramach danej uprawy.



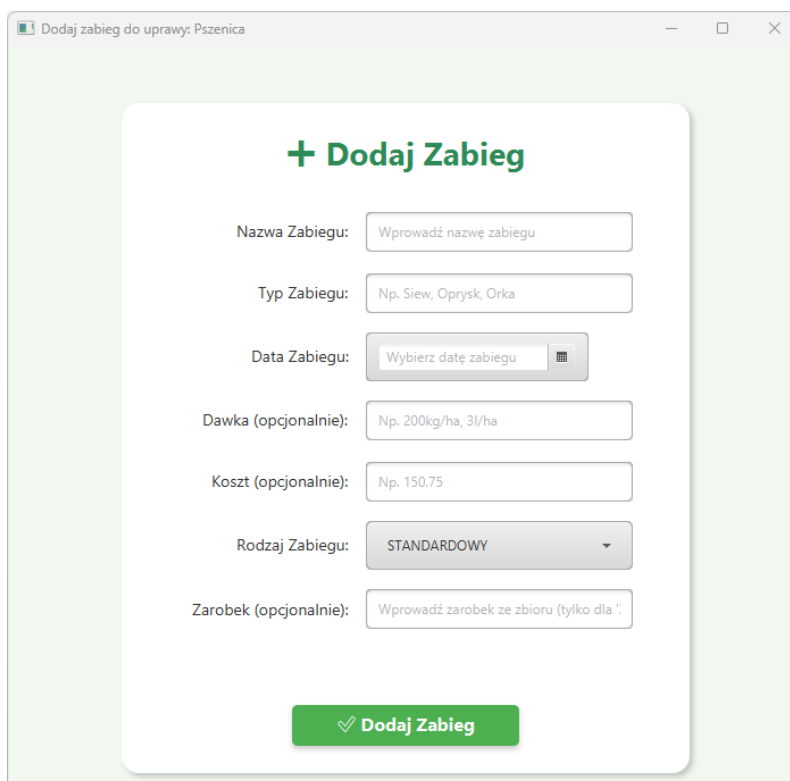
Rys. 5.6. Widok z listą zabiegów dla danej uprawy

5.7. Dodawanie nowego zabiegu

W celu wprowadzenia nowego zabiegu rolniczego dla danej uprawy, użytkownik korzysta z formularza Dodaj zabieg. Dostęp do tego okna uzyskuje się przez wciśnięcie przycisku Dodaj zabieg w widoku zarządzania zabiegami.

- Pole tekstowe Nazwa zabiegu służy do wprowadzenia nazwy opisującej daną czynność.
- Pole tekstowe Typ zabiegu określa ogólny typ zabiegu.
- Pole wyboru daty służy do określenia daty wykonywanego zabiegu.
- Pole tekstowe Dawka służy do wprowadzenia informacji o zastosowanej dawce.
- Pole tekstowe Koszt pozwala na wprowadzenie kosztu związanego z wykonywaniem danego zabiegu.
- Pole wyboru Rodzaj zabiegu umożliwia wybór predefiniowanych opcji, czyli "STANDARDOWY" lub "ZBIÓR". Wybór "ZBIÓR" aktywuje pole Zarobek, wymuszając jego wypełnienie.
- Pole Zarobek aktywne jest tylko dla zabiegów typu "Zbiór", służy do wprowadzania zysku ze zbioru.
- Przycisk Dodaj zabieg zapisuje dane zabiegu w bazie danych i przypisuje go do aktualnie wybranej uprawy. Okno zostanie zamknięte, a lista zabiegów odświeżona. Aplikacja przeprowadza walidację wprowadzonych danych, wyświetlając komunikaty o błędach.

Widok ten stanowi kluczowe narzędzie do szczegółowego dokumentowania i analizy czynności wykonywanych w ramach danej uprawy.



Rys. 5.7. Widok z formularzem do dodania zabiegu

6. Podsumowanie

Ten projekt to prosta aplikacja do zarządzania danymi dla rolnika: jego polami, tym, co na nich rośnie (uprawy), i różnymi pracami (zabiegi). Została stworzona w Javie z interfejsem graficznym opartym na bibliotece JavaFX i połączona do bazy danych PostgreSQL. Dzięki temu udało się stworzyć spójny i w pełni funkcjonalny system.

Zrealizowane funkcjonalności

W ramach projektu udało się zrealizować najważniejsze rzeczy:

- **Logowanie i rejestracja:** Użytkownik może założyć konto i bezpiecznie się zalogować.
- **Zarządzanie danymi:** Użytkownik może dodawać, przeglądać, usuwać pola, uprawy i zabiegi.
- **Łatwy interfejs:** Aplikacja jest prosta w obsłudze dzięki czytelnym oknom, tabelom i przyciskom.
- **Prosta obsługa danych:** Wszystkie wprowadzone informacje są zapisywane w bazie danych PostgreSQL.

Podczas tworzenia projektu **zastosowano podstawowe zasady programowania obiektowego**. Kod został zorganizowany w logiczne, niezależne moduły, które odzwierciedlają elementy takie jak pola, uprawy czy zabiegi. Dzięki temu struktura aplikacji jest czytelna i łatwa do zarządzania. Ważne dane są chronione (enkapsulacja) i udostępniane w kontrolowany sposób. W projekcie zaimplementowano także **zarządzanie sesją użytkownika**, co pozwala aplikacji na rozpoznawanie zalogowanego użytkownika i dostęp do jego danych.

Co można by jeszcze dodać (dalszy rozwój):

Choć aplikacja już działa, można by ją jeszcze ulepszyć, dodając np.:

- **Raporty:** Tworzenie podsumowań, ile co kosztowało i ile przyniosło zysku.
- **Wykresy:** Pokazanie danych w formie graficznej, żeby było jeszcze łatwiej je analizować.
- **Edycja danych:** Możliwość zmieniania już wprowadzonych informacji o polach, uprawach i zabiegach.

Dodanie tych funkcji sprawiłoby, że aplikacja byłaby jeszcze bardziej przydatna. Jednak w obecnej formie projekt w pełni spełnia wymagania przedmiotu i stanowi dobrą podstawę do dalszej pracy.

Spis rysunków

3.1	Diagram ERD.	10
3.2	Diagram UML	11
4.1	Diagram Gantta przedstawiający harmonogram realizacji projektu.	14
5.1	Ekran logowania i rejestracji.	16
5.2	Menu główne.	17
5.3	Formularz dodawania nowego pola	18
5.4	Widok z listą upraw dla danego pola	19
5.5	Formularz dodawania uprawy dla danego pola	20
5.6	Widok z listą zabiegów dla danej uprawy	21
5.7	Widok z formularzem do dodania zabiegu	22

OŚWIADCZENIE STUDENTA O SAMODZIELNOŚCI PRACY

** – niepotrzebne skreślić