

Projekt bazy danych

Daniel Kozak, Władysław Rogowski
Collegium Witelona, Wydział Nauk Technicznych i Ekonomicznych
Kierunek: Informatyka

Legnica 2025

Spis treści

1	Koncepcja	2
1.1	Cel projektu bazy danych	2
1.2	Opis dziedziny przedmiotowej	2
1.3	Założenia wstępne	2
2	Specyfikacja wymagań systemu	2
2.1	Użytkownicy systemu i ich role	2
2.2	Wymagania funkcjonalne	3
2.3	Wymagania нефункционалне	3
3	Model danych	4
3.1	Schemat bazy danych	4
3.2	Opis poszczególnych encji	4
3.3	Model MVC (Laravel – przykład)	5
4	Interfejs użytkownika	6
4.1	Opis GUI	6
5	Uruchomienie projektu (Docker)	8
5.1	Wymagania wstępne	8
5.2	Kroki uruchomienia	8
5.3	Zawartość repozytorium	9

1. Koncepcja

1.1. Cel projektu bazy danych

Projektowana baza danych stanowi integralną część aplikacji internetowej służącej do organizacji pracy zespołowej. Baza umożliwia rejestrowanie i zarządzanie zadaniami, monitorowanie postępu realizacji poszczególnych etapów projektu oraz wizualizację podziału obowiązków pomiędzy członków zespołu. System wspiera efektywną współpracę poprzez udostępnienie narzędzi do przypisywania zadań, kontrolowania ich statusu oraz gromadzenia informacji o przebiegu prac. Ponadto baza zapewnia przechowywanie danych dotyczących grup roboczych oraz ich członków, co pozwala na przejrzyste zarządzanie zasobami ludzkimi w ramach realizowanych projektów.

1.2. Opis dziedziny przedmiotowej

Projektowana aplikacja pełni rolę pomocniczego narzędzia wspierającego zespoły programistyczne pracujące nad projektami. Jej głównym celem jest usprawnienie organizacji zadań związanych z naprawą błędów i rozdzielaniem obowiązków w zespole. W przeciwieństwie do standardowych mechanizmów zgłaszania błędów, takich jak GitHub Issues, proponowane rozwiązanie ma na celu uproszczenie wyszukiwania najważniejszych i aktualnych problemów spośród wielu istniejących zgłoszeń, które w dużych projektach mogą być trudne do przefiltrowania i uporządkowania. System umożliwia ręczne pobieranie zgłoszeń błędów, które następnie mogą być przypisywane konkretnym członkom zespołu. Aplikacja działa jako lokalny system zarządzania zadaniami, koncentrując się wyłącznie na aktualnych i priorytetowych problemach do rozwiązania, co zwiększa przejrzystość oraz efektywność pracy zespołowej.

1.3. Założenia wstępne

Dla uproszczenia baza danych działa jako system do organizacji pracy nad otwartoźródłowymi projektami, umożliwiając zapisywanie zgłoszeń błędów pochodzących zarówno lokalnie od pracowników i testerów, jak i zdalnie — z systemu GitHub Issues.

2. Specyfikacja wymagań systemu

2.1. Użytkownicy systemu i ich role

Role administratora:

- Testy diagnostyczne bazy danych
- Naprawa bazy danych w przypadku błędów
- Tworzenie kopii zapasowych bazy danych
- Czyszczenie i archiwizacja danych
- Aktualizacja systemu zarządzania bazą danych
- Monitorowanie wydajności

- Monitorowanie prób nieautoryzowanego dostępu
- Tworzenie dokumentacji
- Kontrola zgodności bazy z wymaganiami projektu
- Wsparcie użytkowników

Role użytkownika:

- Przeglądanie bazy danych do pracy nad projektem
- Zgłoszenie problemów z bazą danych administratorowi

2.2. Wymagania funkcjonalne

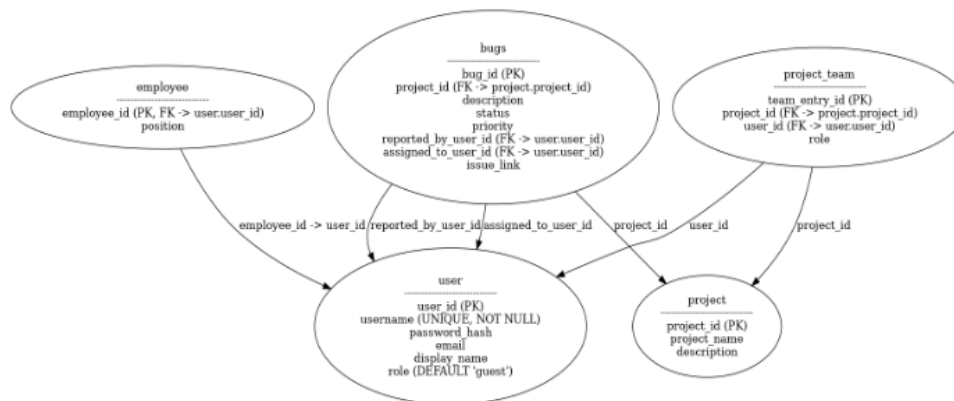
- Dodawanie nazw i opisów projektów
- Dodawanie/usuwanie zadań do projektu i ustawianie ich priorytetu
- Śledzenie postępu tworzenia poszczególnych funkcji
- Rejestracja użytkowników
- Podział użytkowników na zespoły przez administratora
- Przypisanie użytkowników do zadania
- Wyświetlanie listy aktywnych zadań użytkownika

2.3. Wymagania niefunkcjonalne

- Dostęp możliwy tylko po uwierzytelnieniu
- System powinien obsługiwać dużą ilość użytkowników jednocześnie
- Aplikacja powinna być dostępna z poziomu przeglądarki i działać poza siecią lokalną
- Baza danych nie dopuszcza edycji bez autoryzacji administratora

3. Model danych

3.1. Schemat bazy danych



Schemat 1 – Encje bazy danych

3.2. Opis poszczególnych encji

Tabela user – Główna tabela użytkowników systemu. Zawiera dane logowania i podstawowe informacje o użytkowniku.

- **user_id** – unikalny identyfikator użytkownika (klucz główny)
- **username** – unikalna nazwa użytkownika (login); pole wymagane
- **password_hash** – zaszyfrowane hasło użytkownika
- **email** – adres e-mail użytkownika
- **display_name** – nazwa wyświetlana (np. imię i nazwisko)
- **role** – rola użytkownika w systemie

Tabela employee – Rozszerzenie encji user, przechowuje dane pracowników. Użytkownik może, ale nie musi, być pracownikiem.

- **employee_id** – identyfikator pracownika, będący jednocześnie kluczem głównym i obcym do **user.user_id**
- **position** – stanowisko pracownika, np. Programista, Tester

Tabela project – Reprezentuje projekty, nad którymi pracują zespoły.

- **project_id** – unikalny identyfikator projektu (klucz główny)
- **project_name** – nazwa projektu
- **description** – opis projektu

Tabela `project_team` – Łączy użytkowników z projektami oraz określa ich rolę w zespole projektowym.

- `team_entry_id` – identyfikator wpisu (klucz główny)
- `project_id` – odniesienie do projektu (`project.project_id`)
- `user_id` – odniesienie do użytkownika (`user.user_id`)
- `role` – rola użytkownika w projekcie, np. `Team Leader`, `Developer`

Tabela `bugs` – Przechowuje zgłoszenia błędów (bugów) w projektach.

- `bug_id` – unikalny identyfikator błędu (klucz główny)
- `project_id` – identyfikator projektu, którego dotyczy błąd
- `description` – szczegóły zgłoszonego błędu
- `status` – status błędu, np. `open`, `in progress`, `closed`
- `priority` – priorytet błędu, np. `low`, `medium`, `high`
- `reported_by_user_id` – użytkownik zgłaszający błąd
- `assigned_to_user_id` – użytkownik odpowiedzialny za naprawę
- `issue_link` – opcjonalny link do zgłoszenia w zewnętrznym systemie (np. GitHub)

3.3. Model MVC (Laravel – przykład)

Listing 1: UserFactory.php

```
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Str;

/**
 * @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\User>
 */
class UserFactory extends Factory
{
    protected static ?string $password;

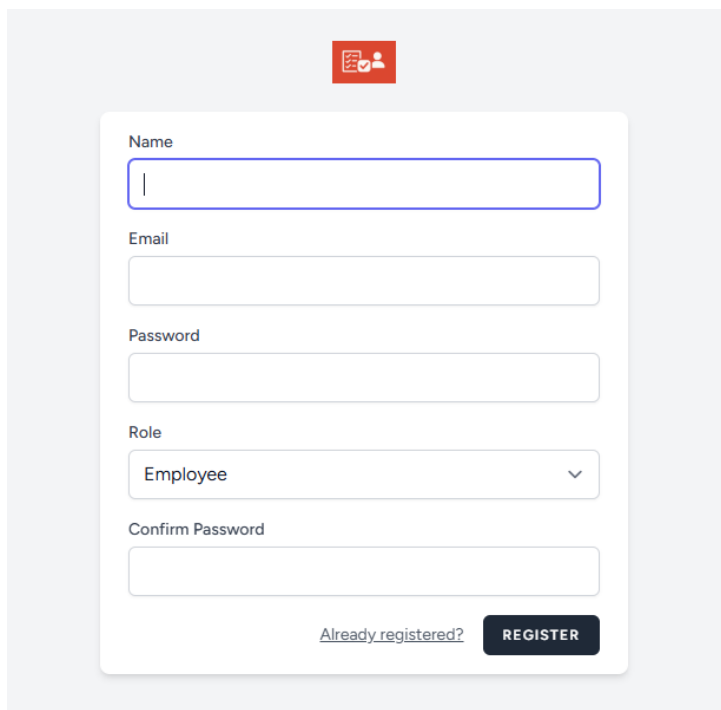
    public function definition(): array
    {
        return [
            'name' => fake()->name(),
```


```
    'email' => fake()->unique()->safeEmail(),
    'email_verified_at' => now(),
    'password' => static::$password ??= Hash::make('password'),
    'remember_token' => Str::random(10),
  ];
}

public function unverified(): static
{
    return $this->state(fn (array $attributes) => [
        'email_verified_at' => null,
    ]);
}
```

4. Interfejs użytkownika

4.1. Opis GUI

A user registration form with a red icon at the top. The form contains fields for Name, Email, Password, Role (a dropdown menu with 'Employee' selected), and Confirm Password. At the bottom, there is a link 'Already registered?' and a 'REGISTER' button.



Name

Email

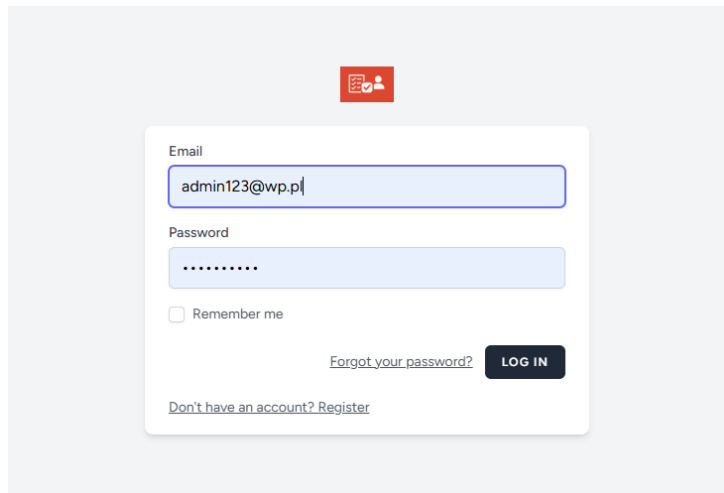
Password

Role
Employee ▾

Confirm Password

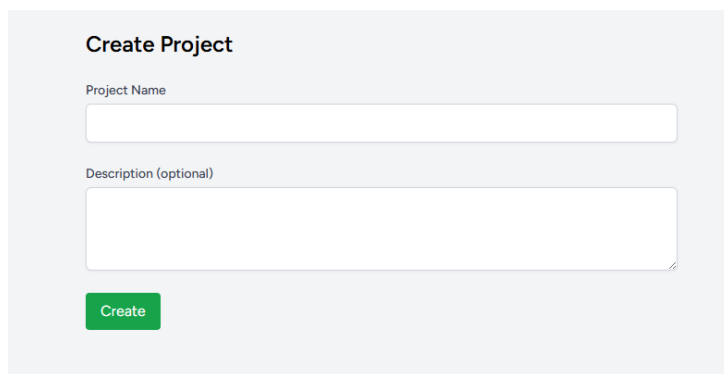
[Already registered?](#)

Rysunek 1: Rejestracja użytkownika



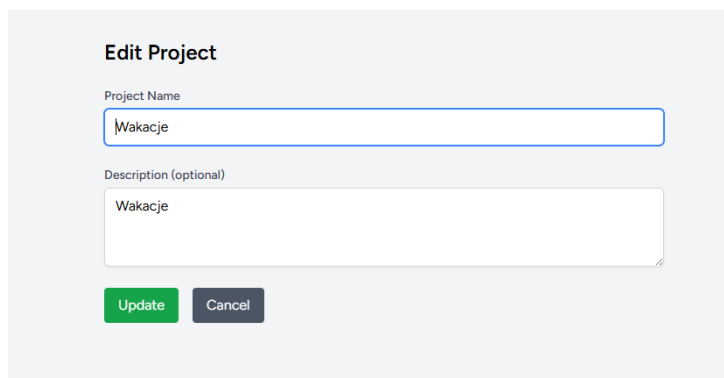
A login form centered on a light gray background. At the top center is a red square icon with a white document and a person. The form is a white rounded rectangle containing the following elements: an 'Email' label above a text input field with 'admin123@wp.pl'; a 'Password' label above a password input field with masked characters; a 'Remember me' checkbox; a 'Forgot your password?' link; a dark gray 'LOG IN' button; and a 'Don't have an account? Register' link at the bottom.

Rysunek 2: Logowanie do systemu



A 'Create Project' form on a light gray background. The title 'Create Project' is in bold. Below it are two input fields: 'Project Name' and 'Description (optional)'. At the bottom is a green 'Create' button.

Rysunek 3: Formularz tworzenia projektu



An 'Edit Project' form on a light gray background. The title 'Edit Project' is in bold. Below it are two input fields: 'Project Name' (containing 'Wakacje') and 'Description (optional)' (containing 'Wakacje'). At the bottom are two buttons: a green 'Update' button and a dark gray 'Cancel' button.

Rysunek 4: Edycja projektu

Tasks for project: **Sesja**

[+ Add Task](#)

Title	Status	Priority	Assigned to	Actions
Projekt PSBD	Done	High	—	Edit Delete
Projekt PIPIL	To Do	High	—	Edit Delete
PIPo	To Do	Medium	—	Edit Delete
Sieci eh	To Do	Low	—	Edit Delete
Mn	In Progress	Medium	—	Edit Delete

Rysunek 5: Zadania projektu

Edit Task: Projekt PSBD

Title

Description

Status

Priority

Assign to User

[Save](#) [Cancel](#)

Rysunek 6: Edycja zadania projektu

5. Uruchomienie projektu (Docker)

Projekt został przygotowany do uruchomienia w kontenerach Docker, co pozwala na łatwe wdrożenie środowiska lokalnego bez konieczności ręcznej instalacji zależności. Poniżej przedstawiono kroki niezbędne do uruchomienia aplikacji:

5.1. Wymagania wstępne

- Zainstalowany Docker oraz Docker Compose
- Klon repozytorium projektu

5.2. Kroki uruchomienia

1. Sklonuj repozytorium:

```
git clone https://github.com/Daniel321W/MainBase
cd nazwa-projektu
```

2. Utwórz plik środowiskowy:

```
cp .env.example .env
```


3. Zbuduj kontenery Docker:

```
docker-compose build
```

4. Uruchom kontenery:

```
docker-compose up -d
```

5. Zainstaluj zależności PHP (Laravel):

```
docker-compose exec app composer install
```

6. Wygeneruj klucz aplikacji:

```
docker-compose exec app php artisan key:generate
```

7. Wykonaj migracje bazy danych:

```
docker-compose exec app php artisan migrate
```

8. (Opcjonalnie) wypełnij bazę danymi testowymi:

```
docker-compose exec app php artisan db:seed
```

9. Odwiedź aplikację w przeglądarce:

```
http://localhost:8000
```

5.3. Zawartość repozytorium

- `docker-compose.yml` – konfiguracja usług (Laravel, MySQL itp.)
- `Dockerfile` – definicja środowiska aplikacji
- `.env.example` – szablon pliku konfiguracyjnego środowiska
- `src/` – katalog z kodem źródłowym aplikacji Laravel