

# 1. Cel projektu

Celem projektu jest zaprojektowanie i wykonanie webowego systemu zarządzania budynkiem (lub budynkami) z wykorzystaniem bazy danych, z obsługą ról użytkowników oraz przygotowaniem pod przyszłą integrację z:

- urządzeniami beacon (do nawigacji wewnętrznej),
- zewnętrznymi czujnikami (np. przyciski alarmowe, czujniki dymu).

System **nie musi** jeszcze realnie łączyć się z fizycznymi urządzeniami – ważne jest przygotowanie odpowiednich struktur danych, interfejsów (np. JSON) oraz logiki reagowania w aplikacji.

## 2. Role użytkowników w systemie

W systemie muszą istnieć co najmniej trzy role:

1. **Administrator systemu**
2. **Zarządca budynku**
3. **Użytkownik (rola podstawowa)**

### 2.1. Administrator

Administrator ma najszersze uprawnienia, w szczególności:

- logowanie do panelu administracyjnego,
- **zarządzanie użytkownikami:**
  - tworzenie kont użytkowników,
  - nadawanie ról (administrator, zarządca, użytkownik),
  - blokowanie / odblokowywanie kont,
- **zarządzanie budynkami:**
  - tworzenie nowych budynków (nazwa, adres, opis),
  - przypisywanie zarządcy do budynku (jeden lub więcej zarządców),

- usuwanie / archiwizacja budynków (wg przyjętej logiki),
- wgląd do wszystkich danych w systemie (podgląd konfiguracji pięter, pomieszczeń, czujników, beaconów),
- wgląd do logów zdarzeń (np. odebrane zgłoszenia z czujników, wysłane maile alarmowe).

## 2.2. Zarządca budynku

Zarządca ma uprawnienia do zarządzania **konkretnymi budynkami**, do których został przypisany przez Administratora. W szczególności:

- wybór budynku z listy budynków przypisanych do zarządcy,
- **definiowanie struktury budynku:**
  - dodawanie pięter (np. numer piętra, opis),
  - dodawanie pomieszczeń na piętrach,
- **pomieszczenia:**
  - każde pomieszczenie musi mieć:
    - numer (np. 101, 1.12) lub nazwę (np. „Sekretariat”, „Sala konferencyjna”),
    - przypisanie do konkretnego piętra,
  - zarządca wprowadza numer/nazwę pomieszczenia,
  - możliwość dodawania informacji o **kolejności pomieszczeń** na piętrze (np. kolejność w korytarzu – przydatne później do nawigacji),
- **informacje o piętrze i barierach architektonicznych:**
  - możliwość dodania opisu piętra (np. „Parter – część biurowa”),
  - możliwość wprowadzania informacji o barierach architektonicznych (np. „brak windy”, „schody bez poręczy”, „wąskie przejście przy pokoju 103”),
  - możliwość dodania dodatkowych uwag (np. „na tym piętrze zwykle przebywa dużo osób”, „pomieszczenie 210 – serwerownia, wstęp ograniczony”),
- **beacony i urządzenia w budynku:**

- przegląd i edycja informacji o urządzeniach beacon (ID, lokalizacja),
- przegląd informacji o zewnętrznych czujnikach (lokalizacja, ID, typ).

Zarządca powinien mieć dostęp do panelu, który pozwoli mu łatwo zarządzać strukturą budynku: listą pięter, pomieszczeń, urządzeń.

### 2.3. Użytkownik

Użytkownik (rola podstawowa) ma ograniczone uprawnienia, np.:

- możliwość zalogowania się do systemu,
- przeglądanie informacji o budynkach i ich strukturze (piętra, pomieszczenia),
- przeglądanie informacji o barierach architektonicznych i uwagach na piętrach,
- **nie** ma prawa edycji danych konfiguracyjnych (piętra, pomieszczenia, beacony, czujniki).

Zakres uprawnień użytkownika można nieco rozszerzyć, ale musi być wyraźnie mniejszy niż zarządcy i administratora.

## 3. Struktura danych systemu

System musi korzystać z relacyjnej bazy danych (np. MySQL, PostgreSQL, SQL Server, itp.). Studenci powinni zaprojektować model danych w postaci tabel oraz powiązań między nimi, np.:

### 3.1. Przykładowe encje (tabele)

- **Users** (użytkownicy)
  - id, imię, nazwisko, email, hasło (zahashowane), rola, status (aktywne/nieaktywne)
- **Buildings** (budynki)
  - id, nazwa, adres, opis
- **BuildingManagers** (przypisanie zarządców)
  - id, user\_id (zarządca), building\_id
- **Floors** (piętra)

- id, building\_id, numer\_piętra (np. 0, 1, 2 lub -1), opis, uwagi, bariery\_architektoniczne
- **Rooms** (pomieszczenia)
  - id, floor\_id, numer\_lub\_nazwa, kolejność\_na\_piętrze (liczba całkowita), dodatkowy\_opis
- **Beacons**
  - id, building\_id lub floor\_id lub room\_id (wg przyjętego poziomu dokładności),
  - beacon\_id (ID urządzenia), lokalizacja\_opisowa (np. „wejście na piętro”, „przy drzwiach do pokoju 105”)
- **Sensors** (czujniki / urządzenia zewnętrzne)
  - id, building\_id / floor\_id / room\_id,
  - sensor\_external\_id (ID czujnika w zewnętrznym systemie),
  - typ (np. „SMOKE\_DETECTOR”, „PANIC\_BUTTON”),
  - opis\_lokalizacji (np. „korytarz przy pokoju 203”).
- **Events / Alerts** (opcjonalnie, ale zalecane)
  - id, sensor\_id, data\_zgłoszenia, treść\_zgłoszenia, typ\_zagrożenia, status (np. nowe / obsłużone), informacja czy wysłano email.

To są przykłady – dokładny model danych mogą doprecyzować studenci, byleby wszystkie wymagane informacje mogły być przechowywane w bazie.

## 4. Integracja z czujnikami – JSON

System **nie musi** łączyć się z realnymi czujnikami, ale musi przewidywać komunikację w formacie JSON.

Zakładamy, że zewnętrzny system (np. system bezpieczeństwa budynku) wysyła do aplikacji JSON o strukturze:

```
{
  "sensor_id": "ABC123",
  "location": "Piętro 2, korytarz przy pokoju 203",
  "event_type": "SMOKE_DETECTED"
}
```

Minimalne wymagane pola:

- `sensor_id` – identyfikator czujnika (musi dać się powiązać z tabelą `Sensors`),
- `location` – opis lokalizacji (tekst),
- `event_type` – typ zgłoszenia (np. `SMOKE_DETECTED`, `BUTTON_PRESSED`, `TEMPERATURE_HIGH`).

Zadaniem systemu jest:

1. Odebranie takiego JSON-a (np. poprzez endpoint REST API),
2. Zapisanie zdarzenia w bazie danych (tabela `Events / Alerts`),
3. Uruchomienie logiki powiadamiania (wysłanie maila do osoby odpowiedzialnej za budynek).

## 5. Powiadomienia e-mail w przypadku zagrożenia

System musi mieć możliwość wysłania e-maila do osoby odpowiedzialnej za dany budynek (np. zarządcy) w razie wykrycia zagrożenia zgłoszonego przez czujnik.

### Wymagania:

- przy każdym budynku powinna być możliwość przechowania danych kontaktowych osoby odpowiedzialnej (np. email zarządcy lub dedykowy adres alarmowy);
- po odebraniu zgłoszenia JSON z zewnętrznego czujnika:
  - system tworzy rekord zdarzenia w bazie,
  - system wysyła e-mail zawierający:
    - nazwę budynku,
    - lokalizację zgłoszenia,
    - typ zdarzenia (np. „wykryto dym”),
    - datę i godzinę,
    - opcjonalnie link do szczegółów zdarzenia w systemie.

Technologia wysyłania e-maili (np. SMTP, gotowa biblioteka) jest do wyboru przez zespół, ale musi zostać realnie zaimplementowana (nie tylko „na papierze”).

## 6. Interfejs użytkownika – wymagania ogólne

System powinien być aplikacją webową:

- dostęp przez przeglądarkę,
- możliwość logowania z uwzględnieniem ról,
- po zalogowaniu użytkownik widzi funkcje zgodne z nadanymi uprawnieniami.

Przykładowe widoki:

- **Widok administratora:**
  - lista użytkowników, formularz dodawania / edycji użytkownika,
  - lista budynków, formularz dodawania / edycji budynku,
  - przypisywanie zarządców do budynków,
  - podgląd logów zdarzeń.
- **Widok zarządcy:**
  - wybór budynku,
  - zarządzanie piętrami (lista, dodawanie, edycja, bariery architektoniczne),
  - zarządzanie pomieszczeniami (lista, numer/nazwa, kolejność na piętrze),
  - zestawienie beaconów i czujników przypisanych do budynku,
  - podgląd zdarzeń (np. ostatnie alarmy).
- **Widok użytkownika:**
  - przegląd budynków (lista),
  - dla wybranego budynku – przegląd pięter i pomieszczeń,
  - widoczne uwagi o barierach architektonicznych i innych ważnych informacjach.

Wygląd graficzny nie musi być bardzo rozbudowany, ale interfejs ma być:

- czytelny,
- logicznie zorganizowany,
- spójny.

## 7. Wymagania techniczne (minimalne)

- Aplikacja webowa (dowolny stos technologiczny uzgodniony na zajęciach – np. .NET, Java, Python, Node.js, PHP).
- Relacyjna baza danych (np. MySQL, PostgreSQL, SQL Server, SQLite – do ustalenia).
- Obsługa ról i logowania:
  - autoryzacja w oparciu o role (admin/zarządca/użytkownik).
- Przynajmniej jeden endpoint (np. REST) do odbioru zgłoszeń JSON z czujników.
- Moduł wysyłania e-maili (w wersji minimalnej może korzystać z testowego serwera pocztowego / sandboksa).

## 8. Zakres swobody projektowej dla studentów

Studenci mogą samodzielnie zdecydować o:

- szczegółach modelu danych (dodatkowe pola, podziały na tabele),
- technologiach frontend/back-end (w ramach ustalonych na zajęciach),
- sposobie prezentacji danych w interfejsie (układ, nawigacja),
- ewentualnym rozszerzeniu projektu, np.:
  - dodanie filtrowania / wyszukiwania pomieszczeń,
  - dodanie historii zmian w strukturze budynku,
  - dodanie prostych statystyk (np. liczba zdarzeń na budynek).

Ważne jest jednak, aby **wszystkie opisane wyżej wymagania funkcjonalne oraz integracyjne były spełnione.**