Aplikacja bazodanowa - PostgreSQL

AKADEMIK

Dokumentacja projektu wykonywanego w ramach zajęć BAZY DANYCH I

Szymon Łabędziewski

Spis treści

I. Projekt koncepcji i założenia

1. Zdefiniowanie tematu projektu

1.1. Cel projektu

Celem projektu jest stworzenie kompleksowego systemu do zarządzania akademikiem, który umożliwi sprawną administrację studentów, rezerwacjami pokoi oraz płatnościami. System ma na celu usprawnienie i zautomatyzowanie procesów związanych z zarządzaniem miejscami w akademiku, obsługą opłat oraz monitorowaniem obecnych mieszkańców.

Projekt bazuje na bazie danych PostgreSQL, która zapewnia integralność danych i umożliwia efektywne przechowywanie oraz przetwarzanie informacji o studentach, rezerwacjach i płatnościach.

1.2. Zakres funkcjonalny aplikacji

Aplikacja obejmuje następujące funkcjonalności:

a) Moduł zarządzania studentami:
   - Dodawanie, edytowanie i usuwanie danych studentów.
   - Przypisywanie studentów do kont użytkowników.
   - Przegląd listy studentów z możliwością wyszukiwania.

b) Moduł rezerwacji pokoi:
   - Przegląd dostępnych pokoi w akademiku.
   - Tworzenie rezerwacji pokoi dla studentów.

c) Moduł płatności:
   - Rejestrowanie opłat za akademik.
   - Przegląd historii płatności dla każdego studenta.
   - Generowanie raportu płatności.

d) Panel administratora:
   - Przypisywanie pokoi do studentów i ich monitoring.

e) Moduł raportowania:
   - Generowanie raportów na podstawie danych z bazy (np. lista mieszkańców, płatności).

f) Autoryzacja i bezpieczeństwo:
   - Logowanie użytkowników z wykorzystaniem sesji użytkownika.
   - Role użytkowników (student, administrator) z różnym zakresem uprawnień.

g) Interfejs użytkownika:
   - Przejrzysty interfejs użytkownika dostępny z poziomu przeglądarki internetowej.
   - Formularze do zarządzania danymi studentów, rezerwacji i płatności.

1.3. Użytkownicy systemu

Aplikacja jest przeznaczona dla dwóch głównych grup użytkowników:

a) Administratorzy akademika:
   - Zarządzanie bazą danych studentów, pokoi i płatności.
   - Przypisywanie pokoi studentom oraz monitorowanie dostępności.
   - Kontrola opłat oraz przegląd raportów finansowych.

b) Studenci (mieszkańcy akademika):
   - Przegląd swoich danych osobowych i statusu rezerwacji.
   - Monitorowanie historii płatności.

2. Analiza wymagań użytkownika

2.1. Wymagania funkcjonalne

Aplikacja spełnia następujące wymagania funkcjonalne:

a) Zarządzanie studentami:
   - Automatyczne przypisanie użytkownika do studenta na podstawie numeru albumu.
   - Wyszukiwanie studentów po różnych kryteriach (imię, nazwisko, numer albumu).
   - Powiązanie studentów z użytkownikami systemu.

b) Obsługa rezerwacji pokoi:
   - Przegląd dostępności pokoi w akademiku.
   - Zabezpieczenie przed podwójną rezerwacją pokoju dla studenta.

c) Obsługa płatności:
   - Rejestrowanie wpłat od studentów.
   - Możliwość przeglądania historii płatności przez administratora i studentów.
   - Generowanie raportów finansowych na podstawie płatności.

d) Panel administratora:
   - Przegląd statystyk dotyczących mieszkańców akademika.
   - Zarządzanie przydziałami pokoi i użytkownikami.

e) System autoryzacji:
   - Logowanie użytkowników z różnymi poziomami dostępu (student, administrator).
   - Bezpieczne przechowywanie haseł (hashowanie) i zarządzanie sesją użytkownika.

2.2. Wymagania niefunkcjonalne

a) Dostępność:
   - Aplikacja powinna być dostępna 24/7.
   - Powinna działać zarówno na przeglądarkach Chrome, Firefox, Edge.

b) Łatwość użycia:
   - Prosty i intuicyjny interfejs użytkownika.

c) Skalowalność:
   - Możliwość rozbudowy systemu o nowe moduły (np. obsługa akademików w wielu lokalizacjach).

2.3. Przewidywane przypadki użycia

a) Rejestracja nowego użytkownika
   - Klient wchodzi na stronę rejestracji.
   - Wypełnia formularz rejestracyjny, podając dane użytkownika i tworząc hasło.
   - System waliduje dane i tworzy konto użytkownika.
   - Użytkownik otrzymuje dostęp do systemu i może operować w panelu użytkownika.

b) Administrator przypisuje pokój studentowi
   - Administrator loguje się do systemu.
   - Przechodzi do sekcji „Oczekujący”.

- Wybiera studenta z listy i przypisuje mu odpowiedni pokój na podstawie dostępności.
- System aktualizuje dane studenta w bazie, przypisując mu numer pokoju.
- Student może zobaczyć swoje przypisane miejsce w panelu użytkownika.

Warunki początkowe:

- Student musi być zarejestrowany w systemie.
- Pokój musi być dostępny w akademiku.

c) Przegląd danych studenta przez administratora
- Administrator loguje się do systemu.
- Przechodzi do sekcji „Lista studentów".
- Może wyszukać studenta na podstawie numeru albumu, nazwiska, imienia, roku studiów, inicjału kierunku bądź odległości zamieszkania od akademika.
- Przegląda szczegółowe dane studenta, w tym informacje o rezerwacji pokoju i historii płatności.

Warunki początkowe:

- Student musi być dodany do systemu przez formularz rejestracji.

d) Opłacenie akademika przez studenta
- Student loguje się do systemu.
- Przechodzi do sekcji „Płatności".
- Przegląda historię swoich opłat.
- Dokonuje płatności.
- System aktualizuje status płatności w bazie danych.

Warunki początkowe:

- Posiadanie zapisu w bazie danych studentów.

e) Generowanie raportu przez administratora
- Administrator loguje się do panelu zarządzania.
- Przechodzi do sekcji „Lista opłat".
- System generuje raport na podstawie danych z bazy i wyświetla go w formacie tabelarycznym.

Warunki początkowe:

- W systemie muszą znajdować się dane płatności.

3. Zaprojektowanie funkcji

3.1. Przegląd podstawowych funkcji bazy danych

Podstawowe funkcje bazy danych w projekcie akademika zostały zaprojektowane w celu zapewnienia integralności i efektywności operacji na danych. Główne funkcjonalności obejmują:

1. **Przechowywanie danych studentów** – tabela studenci przechowuje podstawowe informacje o studentach, w tym numer albumu jako klucz główny.
2. **Zarządzanie pokojami** – tabela pokoje zawiera informacje o ilości miejsc i typie występujących łóżek.

3. **Obsługa rezerwacji** – tabela rezerwacje umożliwia przypisanie studentów do konkretnych pokoi wraz z datą rezerwacji.
4. **Rejestracja płatności** – tabela platnosci przechowuje informacje o wpłatach.
5. **Zabezpieczenie spójności danych** – poprzez klucze obce oraz ograniczenia bazy danych (CHECK).
6. **Raportowanie i analityka** – wykorzystanie widoków bazy danych do generowania raportów na temat wpłat studentów oraz dostępnych pokoi.
7. **Obsługa użytkowników systemu** – tabela uzytkownicy zarządza kontami użytkowników, studentów oraz administratorów, zapewniając autoryzację dostępu do danych.
8. **Automatyczne przypisanie użytkowników do studentów** – funkcja przypisz_uzytkownika_do_studenta() automatyzuje proces powiązania konta użytkownika ze studentem na podstawie numeru albumu.

3.2. Mechanizmy raportowania i analizy danych

System zarządzania akademikiem wykorzystuje różne mechanizmy raportowania i analizy danych oparte na funkcjach agregujących oraz widokach bazy danych PostgreSQL. Mechanizmy te pozwalają administratorowi monitorować sytuację finansową oraz statystyczną w akademiku, umożliwiając efektywne zarządzanie danymi studentów i rezerwacjami.

- Funkcje agregujące
  a) Funkcja statystyki płci mieszkańców (statystyki_plci)

Funkcja oblicza liczbę studentów każdej płci oraz ich procentowy udział w całkowitej liczbie mieszkańców akademika.

```
SELECT
        plec AS gender,
        COUNT(*) AS total_count,
        ROUND(100.0 * COUNT(*) / (SELECT COUNT(*) FROM projekt.studenci), 2) AS
percentage
      FROM projekt.studenci
      GROUP BY plec;
```

- Widoki bazy danych używane do raportowania

Aby usprawnić dostęp do danych i poprawić wydajność, utworzono następujące widoki bazy danych:

a) Widok projekt.kierunki_posortowane

Widok przedstawia listę kierunków studiów posortowaną alfabetycznie, zawierającą dane o progach punktowych na przestrzeni kilku lat. Umożliwia analizę zmian progów rekrutacyjnych.

```
CREATE OR REPLACE VIEW projekt.kierunki_posortowane AS
SELECT kierunek_id, nazwa_kierunku, progi_2020, progi_2021, progi_2022, progi_2023, progi_2024
FROM projekt.kierunki
ORDER BY nazwa_kierunku;
```

b) Widok projekt.mieszkancy_widok

Widok zawiera informacje o obecnych mieszkańcach akademika, sortując je według numeru pokoju oraz nazwiska studenta, co umożliwia szybkie przeglądanie mieszkańców.

```
CREATE OR REPLACE VIEW projekt.mieszkancy_widok AS
SELECT s.nr_albumu, s.imie, s.nazwisko, r.pokoj_id, r.data_od
FROM projekt.studenci s
JOIN projekt.rezerwacje r ON s.nr_albumu = r.nr_albumu
WHERE s.status_zamieszkania = 'zamieszkaly'
ORDER BY r.pokoj_id, s.nazwisko;
```

c) Widok projekt.raport_oplaty_studentow

Widok dostarcza podsumowanie płatności studentów, zawierając liczbę transakcji oraz łączną sumę wpłat dla każdego studenta, co ułatwia analizę zaległości płatniczych.

```
CREATE OR REPLACE VIEW projekt.raport_oplaty_studentow AS
SELECT s.nr_albumu, s.imie, s.nazwisko, COUNT(o.transakcja_nr) AS liczba_transakcji,
    COALESCE(SUM(o.kwota), 0) AS suma_oplat
FROM projekt.studenci s
LEFT JOIN projekt.oplaty o ON s.nr_albumu = o.nr_albumu
GROUP BY s.nr_albumu, s.imie, s.nazwisko;
```

II. Projekt diagramów (konceptualny)

4. Budowa i analiza diagramu przepływu danych

4.1. Główne procesy w systemie

System zarządzania akademikiem obejmuje następujące główne procesy, które są kluczowe dla jego funkcjonowania:

1. **Rejestracja i logowanie użytkownika:**
    - Użytkownik rejestruje się w systemie, podając wymagane dane
    - Po rejestracji użytkownik może zalogować się, podając swoje dane uwierzytelniające.
    - System weryfikuje poprawność danych i przyznaje odpowiednią rolę (student/administrator).
2. **Obsługa rezerwacji:**
    - Administrator przypisuje pokój studentowi na podstawie dostępności.
    - System zapisuje rezerwację, uwzględniając datę rozpoczęcia.
3. **Zarządzanie płatnościami:**
    - Student przegląda swoje płatności i dokonuje wpłat.
    - Administrator monitoruje płatności.
    - System rejestruje transakcje.
4. **Generowanie raportów i statystyk:**
    - Administrator przechwytuje raporty dotyczące zajętości akademika, historii płatności oraz danych demograficznych.
    - Raporty są generowane na podstawie widoków bazy danych i funkcji agregujących.

4.2. Przepływ danych między encjami

System zarządzania akademikiem obejmuje kluczowe encje, które współpracują ze sobą w celu zapewnienia integralności danych i poprawnego funkcjonowania aplikacji.

a) Encja studenci

- Przechowuje dane osobowe studentów oraz informacje dotyczące ich zamieszkania.
- Powiązania:
    - **1:n** z rezerwacje – student może mieć wiele rezerwacji.
    - **1:n** z oplaty – student może mieć wiele płatności.
    - **1:1** z uzytkownicy – student jest powiązany z jednym kontem użytkownika.
    - **n:m** z kierunki – student może być zapisany na wiele kierunków poprzez tabelę pośrednią studenci_kierunki.

b) Encja rezerwacje

- Zarządza przydziałem pokoi dla studentów, zawierając informacje o przypisanym pokoju oraz czasie rezerwacji.
- Powiązania:
    - **n:1** z studenci – student może mieć tylko jedną aktywną rezerwację.
    - **n:1** z pokoje – każdy pokój może być wielokrotnie rezerwowany.

c) Encja pokoje
   - Przechowuje informacje o dostępnych pokojach w akademiku, takie jak liczba miejsc i rodzaj łóżek.
   - Powiązania:
     - **n:1** z mieszkania – każdy pokój należy do konkretnego mieszkania.
     - **1:n** z rezerwacje – pokój może mieć wiele rezerwacji.

d) Encja oplaty
   - Przechowuje informacje dotyczące kwoty wpłaty, daty i godziny płatności.
   - Powiązania:
     - **n:1** z studenci – każdy student może mieć wiele opłat.

e) Encja uzytkownicy
   - Odpowiada za przechowywanie danych logowania i autoryzacji użytkowników.
   - Powiązania:
     - **1:1** z studenci – każdy użytkownik jest powiązany z jednym studentem.

Przepływ danych w systemie

i. Rejestracja użytkownika → przypisanie konta użytkownika do studenta.

ii. Tworzenie rezerwacji → przypisanie studenta do pokoju i aktualizacja statusu.

iii. Dokonywanie płatności → aktualizacja historii wpłat studenta.

iv. Generowanie raportów → analiza płatności i dostępności miejsc w akademiku.

4.3. Interakcja z użytkownikiem

Interakcja użytkownika z systemem przebiega na trzech poziomach:

1. **Student:**
   - **Wejście:**
     - Rejestracja konta (formularz rejestracyjny).
     - Dokonywanie płatności.
   - **Wyjście:**
     - Informacje o przypisanym pokoju.
     - Historia transakcji.

2. **Administrator:**
   - **Wejście:**
     - Przypisywanie pokoi studentom.
     - Monitorowanie wpłat i generowanie raportów.
   - **Wyjście:**
     - Lista studentów z przypisanymi pokojami.
     - Raporty finansowe dotyczące opłat.
     - Statystyki demograficzne mieszkańców akademika.

5. Zaprojektowanie relacji pomiędzy encjami

5.1. Relacje 1-n, n-m, 1-1

System wykorzystuje trzy główne typy relacji:

a) Relacje 1-n (jeden do wielu):
   - Student → Rezerwacje (1-n):
     - Jeden student może mieć wiele rezerwacji w różnych okresach czasu.
   - Student → Płatności (1-n):
     - Jeden student może dokonać wielu wpłat na swoje konto akademika.
   - Pokój → Rezerwacje (1-n):
     - Jeden pokój może być wielokrotnie rezerwowany przez różnych studentów w różnych terminach.
   - Mieszkanie → Pokoje (1-n):
     - Jedno mieszkanie może zawierać wiele pokoi.

b) Relacje n-m (wiele do wielu):
   - Student → Kierunki (n-m):
     - Jeden student może być zapisany na wiele kierunków, a jeden kierunek może mieć wielu studentów, realizowane przez tabelę pośrednią studenci_kierunki.

c) Relacje 1-1 (jeden do jednego):
   - Użytkownik → Student (1-1):
     - Każdy użytkownik systemu (login, hasło) odpowiada dokładnie jednemu studentowi.

5.2. Klucze główne i obce

Każda encja posiada zdefiniowany klucz główny (PK) oraz klucze obce (FK), które zapewniają integralność referencyjną bazy danych.

a) Tabela: studenci
   - Klucz główny (PK): nr_albumu
   - Klucze obce (FK):
     - uzytkownik_id → odniesienie do tabeli uzytkownicy.

b) Tabela: rezerwacje
   - Klucz główny (PK): rezerwacja_id
   - Klucze obce (FK):
     - nr_albumu → odniesienie do tabeli studenci.
     - pokoj_id → odniesienie do tabeli pokoje.

c) Tabela: pokoje
   - Klucz główny (PK): pokoj_id
   - Klucze obce (FK):
     - mieszkanie_id → odniesienie do tabeli mieszkania.

d) Tabela: oplaty
   - Klucz główny (PK): transakcja_nr
   - Klucze obce (FK):
     - nr_albumu → odniesienie do tabeli studenci.

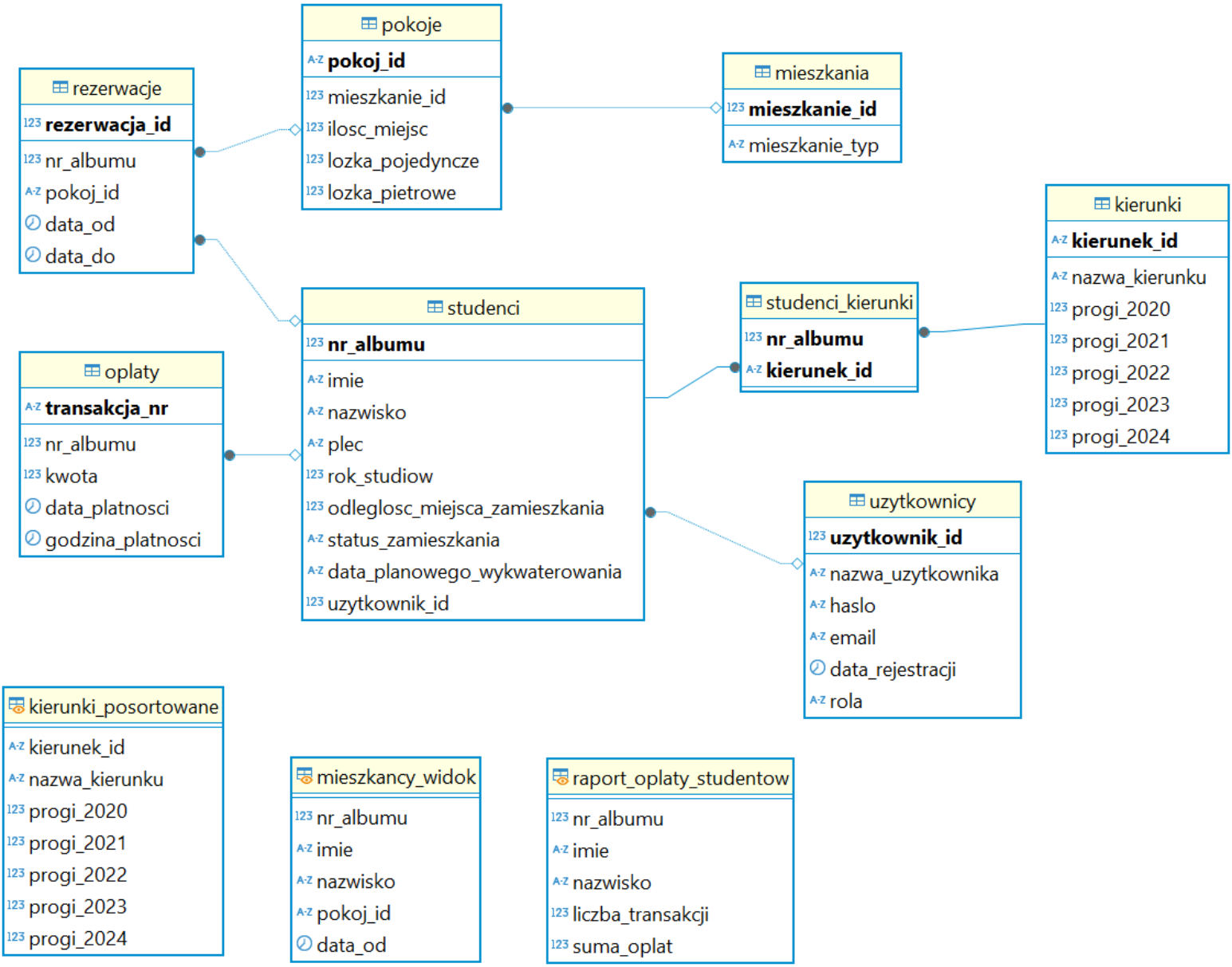
e) Tabela: studenci_kierunki (relacja n-m między studentami a kierunkami)
   - Klucz główny (PK): połączenie nr_albumu, kierunek_id
   - Klucze obce (FK):
     - nr_albumu → odniesienie do tabeli studenci.

- kierunek_id → odniesienie do tabeli kierunki.

f) Tabela: mieszkancy_widok (widok do podglądu danych)
   - Atrybuty:
     - nr_albumu, imie, nazwisko, pokoj_id, data_od
   - Widok bazuje na połączeniu tabel studenci i rezerwacje.

5.3. Diagram ERD (encje, atrybuty, relacje)

Diagram encji i relacji (ERD) przedstawia wizualizację struktury bazy danych, ukazując związki pomiędzy tabelami oraz kluczowe atrybuty.



III. Projekt logiczny

6. Projektowanie tabel, kluczy, indeksów

W systemie zarządzania akademikiem zaprojektowano strukturę bazy danych, obejmującą kluczowe tabele, klucze oraz indeksy, które zapewniają integralność i optymalizację operacji na danych.

6.1. Struktura tabel

Baza danych składa się z następujących głównych tabel:

- **Tabela studenci** – przechowuje informacje o studentach, takie jak numer albumu, dane osobowe oraz status zamieszkania. Klucz główny (nr_albumu) zapewnia unikalność każdego rekordu. Atrybuty tabeli obejmują m.in. imię, nazwisko oraz płeć, której wartości są ograniczone do „M" lub „K".
- **Tabela rezerwacje** – zarządza przypisaniem studentów do pokoi w akademiku. Każda rezerwacja jest jednoznacznie identyfikowana przez rezerwacja_id. Tabela posiada klucze obce łączące ją z tabelami studenci oraz pokoje, co umożliwia utrzymanie integralności danych.
- **Tabela pokoje** – zawiera dane dotyczące pokoi dostępnych w akademiku, takie jak numer pokoju, piętro oraz status (np. „wolny" lub „zajęty"). Klucz główny to pokoj_id.
- **Tabela oplaty** – przechowuje informacje dotyczące płatności dokonanych przez studentów. Klucz główny (transakcja_nr) umożliwia identyfikację każdej wpłaty, a klucz obcy (nr_albumu) wiąże płatność z odpowiednim studentem.

6.2. Indeksy wspomagające wydajność

W celu optymalizacji działania bazy danych i przyspieszenia zapytań, utworzono następujące indeksy:

- **Indeks na kolumnie nr_albumu w tabeli oplaty** – często używany w raportach dotyczących płatności studentów, umożliwia szybkie grupowanie danych.
- **Indeks na kolumnie pokoj_id w tabeli rezerwacje** – optymalizuje wyszukiwanie rezerwacji dla danego pokoju, co jest szczególnie przydatne przy analizie dostępności miejsc w akademiku.

7. Słowniki danych

7.1. Opis pól i ich znaczenia

Tabela	Atrybut	Opis
studenci	nr_albumu	Unikalny numer identyfikacyjny studenta
	imie	Imię studenta
	nazwisko	Nazwisko studenta
	plec	Płeć studenta (M/K)
	status_zamieszkania	Status zakwaterowania (zamieszkały, wyprowadzony)

Tabela	Atrybut	Opis
rezerwacje	rezerwacja_id	Unikalny identyfikator rezerwacji
	nr_albumu	Numer albumu studenta (FK)
	pokoj_id	Identyfikator pokoju (FK)
	data_od	Data rozpoczęcia rezerwacji
	data_do	Data zakończenia rezerwacji

Tabela	Atrybut	Opis
pokoje	pokoj_id	Unikalny identyfikator pokoju (PK)
	mieszkanie_id	Identyfikator mieszkania (FK)
	ilosc_miejsc	Liczba miejsc w pokoju
	lozka_pojedyncze	Liczba łóżek pojedynczych
	lozka_pietrowe	Liczba łóżek piętrowych

Tabela	Atrybut	Opis
oplaty	transakcja_nr	Unikalny identyfikator transakcji
	nr_albumu	Numer albumu studenta (FK)
	kwota	Wartość wpłaty
	data_platnosci	Data wykonania przelewu
	Godzina_platnosci	Godzina wykonania przelewu

7.2. Typy danych w tabelach

Tabela	Atrybut	Typ danych	Uwagi
studenci	nr_albumu	INTEGER	Klucz główny, unikalny
	imie	VARCHAR(50)	Niepuste
	nazwisko	VARCHAR(50)	Niepuste
	plec	CHAR(1)	M/K
	status_zamieszkania	VARCHAR(20)	Domyślnie 'oczekujacy'

Tabela	Atrybut	Typ danych	Uwagi
rezerwacje	rezerwacja_id	SERIAL	Klucz główny
	nr_albumu	INTEGER	Klucz obcy do studenci
	pokoj_id	VARCHAR(10)	Klucz obcy do pokoje
	data_od	DATE	Niepuste
	data_do	DATE	Niepuste

Tabela	Atrybut	Typ danych	Uwagi
pokoje	pokoj_id	VARCHAR(10)	Klucz główny, unikalny
	mieszkanie_id	VARCHAR(10)	Klucz obcy do mieszkania
	ilosc_miejsc	INT	Niepuste
	lozka_pojedyncze	INT	Domyślnie 0
	lozka_pietrowe	INT	Domyślnie 0

Tabela	Atrybut	Typ danych	Uwagi
oplaty	transakcja_nr	SERIAL	Klucz główny
	nr_albumu	INTEGER	Klucz obcy do studenci
	kwota	NUMERIC(10,2)	Niepuste
	Data_platnosci	DATE	Niepuste
	godzina_platnosci	TIME	Niepuste

7.3. Ograniczenia integralnościowe

W celu zapewnienia integralności danych w systemie zastosowano następujące ograniczenia:

1. Ograniczenia wartości (CHECK):
   - Ograniczenie płci w tabeli studenci (M/K):
     CHECK (plec IN ('M', 'K'))
   - Ograniczenie statusu zamieszkania:
     CHECK (status_zamieszkania IN (‘oczekujacy’, 'zamieszkaly', 'wyprowadzony'))
2. Domyślne wartości (DEFAULT):
   - status_zamieszkania – domyślnie ustawiony na 'oczekujacy'.

8. Analiza zależności funkcyjnych i normalizacja tabel

8.1. Zależności funkcyjne między atrybutami

Zależności funkcyjne opisują zależności między atrybutami w tabelach bazy danych. W projekcie akademika występują następujące zależności:

1. Tabela studenci
   - nr_albumu → imie, nazwisko, plec, status_zamieszkania
     (Numer albumu jednoznacznie określa studenta i jego dane.)
2. Tabela rezerwacje
   - rezerwacja_id → nr_albumu, pokoj_id, data_od, data_do
     (Identyfikator rezerwacji jednoznacznie określa wszystkie dane rezerwacji.)
   - pokoj_id → mieszkanie_id
     (Identyfikator pokoju jednoznacznie wskazuje jego szczegóły.)
3. Tabela oplaty
   - transakcja_nr → nr_albumu, kwota, data_platnosci, godzina_platnosci
     (Numer transakcji jednoznacznie określa wpłatę studenta.)

8.2. Etapy normalizacji (1NF, 2NF, 3NF)

a) Pierwsza postać normalna (1NF):
   - Każda kolumna przechowuje atomowe wartości (brak powtarzających się grup danych).
   - Wszystkie tabele w systemie spełniają 1NF, ponieważ dane są podzielone na odrębne kolumny.

b) Druga postać normalna (2NF):
   - Spełniona 1NF.
   - Wszystkie atrybuty niekluczowe zależą od całego klucza głównego, a nie od jego części.
   - Przykładowa dekompozycja dla rezerwacje:
   - Podział na tabele studenci(nr_albumu) i rezerwacje(rezerwacja_id, nr_albumu, pokoj_id, data_od, data_do), dzięki czemu eliminujemy zależność od części klucza.

c) Trzecia postać normalna (3NF):
   - Spełniona 2NF.
   - Wszystkie atrybuty niekluczowe są zależne wyłącznie od klucza głównego.

Wszystkie tabele projektu spełniają 3NF, zapewniając brak redundancji i spójność danych.

9. Zaprojektowanie operacji na danych

Komunikacja między frontendem a backendem odbywa się za pomocą REST API z wykorzystaniem metod HTTP takich jak GET, POST, PUT, i DELETE. Autoryzacja użytkowników realizowana jest za pomocą sesji, zapewniając kontrolowany dostęp do danych. Aplikacja obsługuje operacje związane z zarządzaniem studentami, rezerwacjami oraz płatnościami, przetwarzając dane w formacie JSON, a interakcje realizowane są za pomocą funkcji AJAX i Fetch API.

W projekcie zaimplementowano szereg kwerend SQL, które umożliwiają przetwarzanie danych oraz realizację operacji CRUD (Create, Read, Update, Delete), jak również bardziej zaawansowane operacje analizy danych za pomocą JOIN, GROUP BY, oraz HAVING.

9.1. Operacje CRUD

Podstawowe operacje na danych obejmują:

1. Dodawanie nowego studenta:

```
INSERT INTO projekt.studenci (nr_albumu, imie, nazwisko, plec)
VALUES ('123456', 'Jan', 'Kowalski', 'M';
```

2. Pobieranie danych o studentach:

```
SELECT * FROM projekt.studenci WHERE status_zamieszkania = 'zamieszkaly';
```

9.2. Zapytania zaawansowane

Aby uzyskać bardziej szczegółowe raporty i analizy danych, zastosowano złączenia tabel oraz funkcje agregujące:

a) Pobranie listy studentów wraz z przypisanymi pokojami:

```
SELECT s.nr_albumu, s.imie, s.nazwisko, p.pokoj_id, r.data_od, r.data_do
FROM projekt.studenci s
JOIN projekt.rezerwacje r ON s.nr_albumu = r.nr_albumu
JOIN projekt.pokoje p ON r.pokoj_id = p.pokoj_id;
```

b) Suma wpłat dla każdego studenta:

```
SELECT nr_albumu, SUM(kwota) AS suma_wplat
FROM projekt.oplaty
GROUP BY nr_albumu;
```

IV. Projekt funkcjonalny

10.Interfejsy do prezentacji, edycji i obsługi danych

Aplikacja do zarządzania akademikiem zapewnia dwa główne interfejsy użytkownika: panel administratora oraz panel studenta, umożliwiające zarządzanie danymi studentów, rezerwacjami oraz płatnościami. Interfejsy zostały zaprojektowane w sposób intuicyjny i responsywny, zapewniając wygodę użytkowania.

10.1. Opis stron frontendowych aplikacji

Aplikacja składa się z następujących kluczowych stron:

- **index.html** – strona główna z formularzem logowania.
- **dashboard.html** – panel administratora prezentujący podsumowanie danych.
- **view-students.html** – zarządzanie studentami (dodawanie, edycja, usuwanie).
- **reservations.html** – przegląd rezerwacji oraz przypisywanie pokoi.
- **student-payment.html** – podgląd historii płatności studenta.
- **view-reports.html** – generowanie raportów dotyczących rezerwacji i płatności.

➢ Interfejs administratora

Panel administratora jest dostępny po zalogowaniu i umożliwia zarządzanie danymi akademika. Składa się z:

- Panelu nawigacyjnego (lewa strona ekranu), który zawiera:
    - Rezerwacje – przypisywanie pokoi studentom.
    - Płatności – przegląd dokonanych wpłat.
    - Raporty – generowanie zestawień finansowych i mieszkaniowych.
- Sekcji głównej, która dynamicznie wyświetla dane w zależności od wybranej opcji w menu.
    - Tabele umożliwiają filtrowanie, sortowanie i przeglądanie danych studentów.

➢ Interfejs studenta

Panel studenta umożliwia użytkownikowi przegląd swoich danych osobowych, płatności oraz statusu rezerwacji. Składa się z:

- Menu nawigacyjnego, które zawiera:
    - Moje dane – przegląd danych osobowych (imię, nazwisko, nr albumu).
    - Zakwaterowanie – informacje o przypisanym pokoju (numer oraz dane współlokatora).
    - Płatności – historia wpłat oraz status zaległości.

10.2. Formularze do zarządzania danymi (studenci, rezerwacje, płatności)

System udostępnia formularze umożliwiające operacje CRUD (Create, Read, Update, Delete) dla kluczowych danych aplikacji, takich jak studenci, rezerwacje i płatności. Formularze są dostępne dla administratora oraz studenta, a dane mogą być wprowadzane ręcznie za pomocą dedykowanych interfejsów użytkownika.

10.3. Walidacja danych w formularzach

Formularze zawierają mechanizmy walidacji danych zarówno na poziomie frontendowym, jak i backendowym:

- **Frontend:**
  - Walidacja pól obowiązkowych.
  - Sprawdzenie poprawności formatów danych (np. e-mail, numeryczne wartości kwot).
  - Komunikaty błędów dla użytkownika.
- **Backend:**
  - Walidacja danych przed zapisaniem w bazie.
  - Sprawdzenie unikalności danych (np. e-mail studenta).
  - Obsługa błędów i zabezpieczenia przed błędnymi danymi.

11.Wizualizacja danych

11.1. Wykorzystanie tabel do prezentacji danych

Dane są prezentowane w aplikacji w postaci tabel HTML z możliwością:

- Sortowania i filtrowania:
  - Listy studentów, płatności i rezerwacji.
  - Skrypt view-reports.js obsługuje dynamiczne filtrowanie.

12.Makropolecenia

12.1. Ułatwienie obsługi aplikacji poprzez automatyzację procesów

Aplikacja automatyzuje kluczowe procesy poprzez:

- Automatyczne przypisywanie użytkownika do studenta po rejestracji danych (funkcja bazodanowa przypisz_uzytkownika_do_studenta).
- Dynamiczne generowanie raportów finansowych i mieszkaniowych na żądanie.

12.2. Przykłady automatycznych akcji wykonywanych na danych

- Przypisanie pokoju studentowi: Automatyczne zaktualizowanie statusu pokoju po jego przypisaniu.
- Aktualizacja statusu płatności: Po dokonaniu wpłaty status zmienia się na "opłacone".
- Sortowanie listy mieszkańców**:** Automatyczne porządkowanie po numerze pokoju i nazwisku.

V. Dokumentacja techniczna

13.Wprowadzanie danych

13.1. Mechanizmy walidacji danych

Frontend: Walidacja pól formularzy (np. wymagane pola, poprawność formatu e-mail).
Backend: Sprawdzenie unikalności danych, reguły walidacyjne w bazie (CHECK, NOT NULL).
Baza danych: Wyzwalacze zapewniające spójność, np. przypisanie użytkownika do studenta

14.Dokumentacja użytkownika

14.1. Instrukcja obsługi aplikacji krok po kroku

1. Rejestracja w systemie

1. Otwórz stronę logowania/rejestracji w przeglądarce internetowej za pomocą dopisania numeru portu do odpowiedniej domeny.
2. Wypełnij formularz rejestracyjny, podając:
    - Nazwę użytkownika
    - Adres e-mail,
    - Hasło.
3. Kliknij przycisk "Zarejestruj się".
4. Po poprawnej rejestracji zaloguj się do systemu.

2. Logowanie do systemu

1. Przejdź na stronę główną (index.html).
2. Wprowadź nazwę użytkownika oraz hasło podane podczas rejestracji.
3. Kliknij "Zaloguj się".
4. Po poprawnym zalogowaniu zostaniesz przekierowany do swojego panelu użytkownika.

3. Przegląd danych osobowych

1. Po zalogowaniu zobaczysz stronę główną swojego panelu.
2. W sekcji **"**Moje dane" znajdziesz pola do uzupełnienia (które następnie posłużą do ich wyświetlania po wprowadzeniu do bazy:
    - Numer albumu,
    - Imię i nazwisko,
    - Uczęszczany kierunek i rok studiów
    - Płeć i odległość miejsca zamieszkania

4. Sprawdzenie statusu zakwaterowania

1. Po rejestracji administrator przydzieli Ci pokój.
2. Jeśli status zakwaterowania się zmieni, ujrzysz zmianę struktury strony.
3. Gdy pokój zostanie przypisany, w sekcji "Mój pokój" pojawi się:
    - Numer pokoju,
    - Data rozpoczęcia rezerwacji.
    - Dane o współlokatorze

5. Przegląd płatności

1. Przejdź do sekcji "Płatności" z menu nawigacyjnego.
2. Znajdziesz tam informacje o:
    - Historii dokonanych wpłat,

6. Aktualizacja danych osobowych

1. W sekcji "Moje dane" zmień wymagane informacje.
2. Zatwierdź zmiany klikając "Edytuj dane".

8. Wylogowanie z systemu

1. Aby zakończyć sesję, kliknij "Wyloguj" w prawym górnym rogu strony.
2. Zostaniesz przekierowany na stronę logowania.

15.Opracowanie dokumentacji technicznej

15.1. Struktura kodu aplikacji

```
ProjektAkademika/
├── backend
│   ├── controllers
│   ├── database.db
│   ├── middleware
│   │   ├── auth.js
│   │   └── isAdmin.js
│   ├── models
│   ├── node_modules
│   ├── output.log
│   ├── package-lock.json
│   ├── package.json
│   ├── routes
│   │   ├── assignRoom.js
│   │   ├── kierunki.js
│   │   ├── login.js
│   │   ├── payments.js
│   │   ├── register.js
│   │   ├── reservations.js
│   │   ├── residents.js
│   │   ├── students.js
│   │   ├── studentsCourses.js
│   │   └── unassignedStudents.js
│   └── server.js
├── config
│   ├── dbConfig.js
│   ├── package-lock.json
│   └── package.json
├── database
│   ├── procedures
│   ├── schema.sql
│   └── seed.sql
├── frontend
│   ├── admin.html
│   ├── already_logged_in.html
│   ├── assets
│   │   ├── fonts
│   │   └── images
│   ├── assign-room.html
│   ├── dashboard.html
```

```
│   ├── index.html
│   ├── register.html
│   ├── reservations.html
│   ├── residents.html
│   ├── scripts
│   │   ├── admin.js
│   │   ├── assign-room.js
│   │   ├── dashboard.js
│   │   ├── index.js
│   │   ├── register.js
│   │   ├── reservations.js
│   │   ├── residents.js
│   │   ├── student-data.js
│   │   ├── student-payment.js
│   │   ├── unassigned-students.js
│   │   ├── view-payments.js
│   │   ├── view-reports.js
│   │   ├── view-statics.js
│   │   └── view-students.js
│   ├── student-data.html
│   ├── student-payment.html
│   ├── styles
│   │   └── main.css
│   ├── unassigned-students.html
│   ├── view-payments.html
│   ├── view-reports.html
│   ├── view-statistics.html
│   └── view-students.html
├── node_modules
├── package-lock.json
└── package.json
```

15.2. Opis plików backendowych (server.js, routes/*.js, models/*.js)

server.js – główny plik inicjalizujący aplikację:

- Konfiguracja Express.js
- Podłączenie do bazy danych PostgreSQL
- Rejestracja tras API

Katalog routes/ – obsługa żądań HTTP:

- students.js – zarządzanie studentami (GET, POST, PUT, DELETE)
- reservations.js – obsługa rezerwacji pokoi
- payments.js – operacje związane z płatnościami

Katalog models/ – definicje struktur danych dla bazy:

- Student.js – model tabeli studentów
- Reservation.js – model rezerwacji

Katalog middleware/ – funkcje pośredniczące:

- auth.js – uwierzytelnianie użytkowników
- isAdmin.js – kontrola dostępu dla administratora

15.3. Opis plików frontendowych (HTML, CSS, JS)

Interfejs użytkownika składa się z kilku kluczowych stron HTML, w tym strony logowania, panelu administratora i panelu studenta. Każda z tych stron ładuje odpowiednie skrypty JavaScript, które odpowiadają za pobieranie i prezentację danych użytkownikowi. W katalogu scripts znajdują się skrypty do obsługi poszczególnych widoków, takich jak zarządzanie studentami, rezerwacjami oraz płatnościami. Stylizacja interfejsu została zaimplementowana w plikach CSS, które odpowiadają za wygląd strony oraz jej responsywność.

16.Wykaz literatury i źródeł

16.1. Dokumentacja wykorzystanych technologii (Node.js, Express, PostgreSQL)

W projekcie wykorzystano następujące technologie:

- Node.js – platforma do obsługi backendu
  https://nodejs.org/en/docs/
- Express.js – framework do obsługi tras i żądań http
  https://expressjs.com/
- Bootstrap – framework CSS do stylizacji interfejsu użytkownika
  https://getbootstrap.com/

16.2. Bibliografia dotycząca baz danych i projektowania aplikacji

- PostgreSQL Documentation – materiały związane z tworzeniem baz danych
  https://www.postgresql.org/docs/
- W3Schools – poradniki dotyczące SQL i relacyjnych baz danych
  https://www.w3schools.com/sql/
- Dokumentacja Bootstrap – stylizacja frontendu
  https://getbootstrap.com/docs/

16.3. Linki do użytych narzędzi i frameworków

- **Visual Studio Code** – edytor kodu używany do tworzenia projektu
  https://code.visualstudio.com/

VI. Testowanie i wdrożenie

17.Plan testów funkcjonalnych

17.1. Testy jednostkowe aplikacji

Testy przeprowadzane poprzez ręczne sprawdzenie działania poszczególnych funkcji aplikacji na testowych kontach użytkowników:

a) Testy rejestracji i logowania:
   - Próba rejestracji użytkownika z poprawnymi danymi.
   - Próba rejestracji użytkownika z brakującymi danymi lub błędnym formatem (np. niepoprawny email).
   - Weryfikacja działania mechanizmu logowania i obsługi błędnych haseł.

b) Testy zarządzania danymi studenta:
   - Edycja danych użytkownika po rejestracji (zmiana e-maila, hasła).
   - Sprawdzenie poprawności wyświetlania danych użytkownika po zalogowaniu.

17.2. Testy integracyjne komunikacji frontend-backend

Testy realizowane poprzez manualne wprowadzanie danych i weryfikację ich poprawności po stronie administratora:

a) Testy przypisania pokoju przez administratora:
   - Rejestracja testowego użytkownika → zalogowanie jako administrator → przypisanie pokoju → sprawdzenie poprawności przypisania w panelu studenta.

b) Testy płatności:
   - Dodanie płatności dla testowego użytkownika → sprawdzenie poprawności wyświetlenia statusu w panelu studenta.
   - Sprawdzenie poprawności aktualizacji statusu płatności przez administratora.

c) Testy rezerwacji:
   - Próba przypisania tego samego pokoju do różnych studentów (sprawdzenie ograniczeń aplikacji).
   - Usunięcie rezerwacji i sprawdzenie aktualizacji statusu pokoju.

18.Wdrożenie aplikacji

Aplikacja została wdrożona na serwerze uczelnianym Pascal z domeną pascal.fis.agh.edu.pl, działającym na porcie 6006, z dostępem ograniczonym do zalogowanych użytkowników serwera AGH. Wdrożenie obejmuje konfigurację środowiska, proces uruchamiania aplikacji oraz monitorowanie jej działania.

18.1. Konfiguracja i wdrożenie aplikacji

Środowisko serwera zostało skonfigurowane w sposób umożliwiający stabilne działanie aplikacji. Główne elementy środowiska produkcyjnego to:

- System operacyjny: Linux (serwer uczelniany AGH Pascal).
- Środowisko uruchomieniowe: Node.js, wymagane wersje są instalowane globalnie.

- Baza danych: PostgreSQL, skonfigurowana w pliku dbConfig.js.
- Serwer aplikacji: Express.js, uruchamiany za pomocą komendy node server.js.
- Środowisko sieciowe: Aplikacja działa na protokole HTTP, co oznacza brak szyfrowania przesyłanych danych.

Konfiguracja zmiennych środowiskowych:
- Port aplikacji określony w pliku server.js (domyślnie 6006).
- Ścieżka do pliku bazy danych ustawiona w dbConfig.js.

18.2. Proces wdrożenia i uruchomienia aplikacji

Aplikacja jest uruchamiana ręcznie na serwerze uczelnianym za pomocą terminala systemu Linux. Proces wdrożenia obejmuje następujące kroki:

i. **Zalogowanie się na serwer Pascal:**

```
ssh <login>@pascal.fis.agh.edu.pl
```

ii. **Przejście do katalogu projektu:**

```
cd /ścieżka/do/projektu
```

iii. **Instalacja wymaganych zależności (jednorazowo):**

```
npm install
```

iv. **Uruchomienie aplikacji:**

```
node server.js
```

Po poprawnym uruchomieniu aplikacja jest dostępna pod adresem: http://pascal.fis.agh.edu.pl:6006.

Na obecną chwilę (I kwartał 2025 roku) plik server.js jest uruchomiona w tle i nie ma potrzeby ręcznego uruchamiania – strona powinna działać.

18.3. Monitorowanie działania systemu

Aplikacja nie posiada zaawansowanych mechanizmów monitorowania, jednak dostępne są podstawowe funkcje logowania, umożliwiające śledzenie działania aplikacji oraz diagnozowanie potencjalnych problemów.

1. Logi w konsoli terminala:
    - Aplikacja rejestruje działania użytkowników i ewentualne błędy bezpośrednio w konsoli serwera.
    - Podgląd logów odbywa się w czasie rzeczywistym po uruchomieniu aplikacji.
2. Logi w konsoli przeglądarki:
    - Frontend aplikacji wyświetla komunikaty w konsoli przeglądarki (np. sukcesy i błędy operacji).
    - Pomaga to w diagnozowaniu problemów użytkownika i błędów frontendowych.

Brak wdrożonych mechanizmów:
- Tworzenia kopii zapasowych bazy danych.
- Automatycznego restartu aplikacji w przypadku awarii.
- Powiadomień o błędach lub niestandardowym działaniu systemu.