1. Wstęp i Przegląd Funkcjonalny

System Smart Home to zaawansowana aplikacja do zarządzania automatyką domową. Pozwala na dynamiczne tworzenie pokoi, dodawanie urządzeń (oświetlenie, termostaty, systemy alarmowe) oraz zarządzanie nimi poprzez centralny interfejs. System posiada wbudowaną logikę bezpieczeństwa, która automatycznie reaguje na naruszenia (np. wykrycie ruchu przez sensor).

Kluczowe funkcjonalności:

- Hierarchiczna struktura: Hub -> Pokoje -> Urządzenia.
- Zdalne sterowanie: Obsługa parametrów urządzeń (temperatura, zasilanie).
- System bezpieczeństwa: Automatyczna blokada zamków i aktywacja syren po naruszeniu strefy.
- Interaktywny Dashboard: Monitorowanie stanów urządzeń w czasie rzeczywistym.

2. Architektura i Wzorce Projektowe

Projekt został zaprojektowany z myślą o rozszerzalności (zgodnie z zasadami SOLID). Poniżej znajduje się szczegółowy opis wykorzystanych wzorców projektowych:

A. Wzorzec Stanu (State Pattern)

Jest to główna część modułu bezpieczeństwa. Urządzenia takie jak zamki czy alarmy nie używają prostych instrukcji `if/else`, lecz delegują zachowanie do obiektów stanu. Flowchart stanów dostępny jest w command.png

- Zastosowanie: Klasy `OffState`, `ArmedState`, `DetectedState`, `BlockedState`.
- Jak to działa: Gdy wywołasz metodę `trigger()` na alarmie, jego reakcja zależy od aktualnego obiektu stanu. W stanie `OffState` alarm zignoruje sygnał. W stanie `ArmedState` przejdzie do `DetectedState` i wywoła powiadomienie.
- Zaleta: Łatwe dodawanie nowych stanów (np. "Tryb Serwisowy") bez modyfikacji kodu samych urządzeń.

B. Wzorzec Polecenia (Command Pattern)

Wzorzec ten odseparowuje interfejs użytkownika od logiki urządzeń.

- Zastosowanie: Klasy `TogglePowerCommand` oraz `ChangeTempCommand`.
- Jak to działa: Przyciski w GUI nie sterują bezpośrednio obiektem `LightFixture`. Zamiast tego tworzą obiekt polecenia, który "wie", co zrobić. Pilot (Remote Control) wykonuje metodę `.execute()`.
- Zaleta: Pozwala na łatwe kolejkowanie operacji, logowanie historii działań oraz implementację funkcji "Cofnij" (Undo).

C. Wzorzec Obserwatora (Observer Pattern)

Umożliwia komunikację typu "jeden do wielu".

- Zastosowanie: Powiadomienia o naruszeniu bezpieczeństwa (`breach_callback`).
- Jak to działa: `HomeHub` oraz `Room` obserwują sensory ruchu. Gdy sensor wykryje ruch, "rozgłasza" tę informację do obserwatora. Obserwator (Hub) następnie decyduje o reakcji całego systemu (np. zablokowanie wszystkich drzwi w domu).
- Zaleta: Sensory nie muszą wiedzieć, jakie inne urządzenia są w domu. Po prostu wysyłają sygnał o zdarzeniu.

D. Wzorzec Mediator (Mediator Pattern)

Zmniejsza liczbę bezpośrednich powiązań między klasami.

- Zastosowanie: Klasy `Room` i `HomeHub`.
- Jak to działa: Zamiast sytuacji, w której czujnik ruchu bezpośrednio wywołuje metodę na syrenie alarmowej, komunikacja przechodzi przez Mediatora (`Room`). To Mediator wie, jakie urządzenia znajdują się w pokoju i jak powinny zareagować na sygnał z czujnika.
- Zaleta: Urządzenia są od siebie całkowicie niezależne, co ułatwia ich ponowne użycie w innych częściach systemu.

E. Metoda Wytwórcza (Simple Factory)

Centralizuje logikę tworzenia obiektów.

- Zastosowanie: Metoda `add_device` w klasie `Room`.
- Jak to działa: Zamiast ręcznie instancjonować klasy w kodzie GUI, przekazujemy tylko nazwę typu (np. "Lock"). Fabryka decyduje, którą klasę wywołać i jakie parametry (np. callbacki) jej przekazać.
- Zaleta: Jeśli zmienimy nazwę klasy lub jej konstruktor, poprawiamy kod tylko w jednym miejscu – w fabryce.

4. Podsumowanie dla Dewelopera

Aby dodać nowe urządzenie do systemu, należy:

1. Utworzyć klasę dziedziczącą po `SmartDevice` lub `SecurityDevice`.
2. Zarejestrować nowy typ w fabryce wewnątrz klasy `Room`.
3. (Opcjonalnie) Stworzyć nowe klasy poleceń w `commands.py`, jeśli urządzenie wymaga unikalnych akcji.