AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

Praca dyplomowa

Desktopowy System Zarządzania Hotelem
Desktop Hotel Management System

Autor: *Jędrzej Kosior*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr inż. Jacek Piwowarczyk*

Kraków, 2022

Serdecznie dziękuję doktorowi Jackowi Piwowarczykowi za umożliwienie mi napisania tej pracy oraz za udzielanie cennych wskazówek, rodzicom za stworzenie możliwości uzyskania wiedzy potrzebnej do ukończenia tej pracy oraz mojej partnerce za nieustające wsparcie i wszelkie rady.

Spis treści

Spis rysunków

1. Wstęp

1.1. Wprowadzenie

W obecnych czasach wiele przedmiotów można spersonalizować. Możemy dopasować je do potrzeb właściciela, dobierając etui do telefonu w ulubionych kolorach, bądź odpowiedni system multimedialny w samochodzie, który spełniłby oczekiwania przyszłego właściciela. Programy komputerowe są personalizowane przez użytkowników dzięki możliwościom, które oferują wewnętrzne ustawienia.

Wiele firm oferuje możliwość stworzenia oprogramowania od podstaw zgodnych ze ścisłymi wymaganiami klienta. Dzięki takim rozwiązaniom odbiorca systemu jest w stanie przyśpieszyć wykonywaną pracę.

Właściciel hotelu również może być klientem firmy tworzącej spersonalizowane oprogramowanie. W przypadku hoteli, nie udostępniają one tylko pokojów, ale również pozwalają korzystać ze skomplikowanej sieci logistycznej umożliwiającej funkcjonowanie takiego obiektu. Zarządzanie logistyką powinno być proste i intuicyjne. Jest to możliwe dzięki personalizacji systemu pod konkretnego odbiorcę.

1.2. Cele pracy

Celem pracy jest stworzenie systemu, który będzie umożliwiał intuicyjną obsługę hotelu pracownikom wykonującym obowiązki na stanowiskach takich jak recepcja, kuchnia, czy zespół sprzątający. Każdy z użytkowników będzie mógł wykonać operacje wewnątrz systemu, które zgodne będą z jego uprawnieniami. Uprawnienia weryfikowane będą podczas logowania do systemu.

Aplikacja będzie posiadać architekturę trójwarstwową [1], składającą się z warstw interfejsu użytkownika, przetwarzania wczytanych danych oraz warstwy bazy danych. Do budowy systemu, przetwarzania danych wprowadzonych przez użytkownika i do zarządzaniem bazą danych zostanie wykorzystany język programowania Python.

1.3. Założenia projektu

Założenia projektu, które wykonywane będą w kolejności chronologicznej:

1. przegląd narzędzi, które mogą zostać wykorzystane podczas tworzenia aplikacji desktopowych w języku programowania Python;
2. stworzenie środowiska aplikacyjnego;
3. stworzenie prototypu aplikacji z wykorzystaniem framework-u Kivy;
4. stworzenie bazy danych oraz zintegrowanie jej z prototypem aplikacji;
5. stworzenie docelowego interfejsu użytkownika dla funkcji zawartych w prototypie;
6. implementacja pozostałych możliwości systemu w oparciu o interfejs i technologie użyte w prototypie;
7. wykonanie testów aplikacji oraz eliminacja znalezionych błędów.

1.4. Struktura pracy

Praca została podzielona na sześć rozdziałów przy czym każdy z nich jest podzielony na mniejsze sekcje odpowiadające konkretnym zagadnieniom. W każdej z nich poruszony został inny temat o charakterze praktycznym, bądź teoretycznym.

W rozdziale pierwszym poruszono temat personalizacji, która obejmuje również obszar oprogramowania tworzonego na zlecenie klienta indywidualnego. Skupiono się na aspekcie usprawnienia pracy poprzez wprowadzenie spersonalizowanego rozwiązania zamiast wykorzystywania rozwiązań uniwersalnych dostępnych na rynku. Przedstawiono cele pracy zawierające podstawowe wymagania techniczne jak i funkcjonalne, które należy spełnić aby uznać aplikację za ukończoną. Zaadresowano kroki prac według których była tworzona aplikacja w porządku chronologicznym. Zamieszczono opis zawartości w poszczególnych rozdziałach i ich sekcjach.

Drugi rozdział omawia rozwiązania komercyjne dostępne na rynku ze szczególnym naciskiem na możliwości tych programów oraz intuicyjność ich użytkowania. Przybliżono tematykę struktury aplikacji desktopowej oraz wytłumaczono jej działanie. Każda z warstw posiada dedykowany podrozdział, który zawiera szczegóły danego poziomu aplikacji.

Kolejny rozdział zawiera wszelkie informacje związane z projektem systemu, takie jak wymagania, architektura systemu, przypadki użycia, diagramy czynności i klas.

W czwartym rozdziale zamieszczono opisy technologii, które zostały wykorzystane podczas tworzenia systemu wraz z ich omówieniem. Sekcją tego rozdziału jest również implementacja

interfejsu użytkownika z przykładami zastosowanych rozwiązań. Zwrócono uwagę na sposób przetwarzania wczytywanych danych przez użytkownika wraz z przykładami. Ostatnim elementem tego rozdziału jest przedstawienie zastosowanych metod potrzebnych do zarządzania bazą danych, która zawiera wszelkie informacje potrzebne do funkcjonowania aplikacji i hotelu korzystającego z tego systemu.

Piąty rozdział zawiera testy aplikacji. Podzielono je na testy modułowe i systemowe. W testach modułowych przedstawione zostanie zachowanie systemu w konkretnych sytuacjach, natomiast w testach systemowych sprawdzona zostanie reakcja całego systemu oraz poprawność jego funkcjonowania.

Ostatnim rozdziałem jest podsumowanie całej pracy gdzie ukazano ukończone części projektu. Przedstawiono mocne miejsca stworzonego rozwiązania, jak i te, w których występuje potencjał oraz szanse na rozwój i poprawę.

Ostatnie strony pracy zawierają bibliografię składającej się z list wykorzystanej literatury, jak i innych źródeł pomocniczych, które wspomagały proces tworzenia pracy.

2. Zagadnienia wstępne

Poniżej omówiono rozwiązania komercyjne dostępne na rynku wraz z ich opisem. W rozdziale tym zawarto również informacje o budowie aplikacji oraz jej poszczególnych częściach.

2.1. Omówienie rozwiązań komercyjnych

2.1.1. SOHiS

Jednym z rozwiązań dostępnych na rynku jest oprogramowanie SOHiS [2] - System Obsługi Hoteli i Sanatoriów, które umożliwia dokonywania rezerwacji, zameldowania gości, czy też wystawienia dokumentu sprzedaży. Funkcje te usprawniają pracę recepcji, lecz program ten nie jest na tyle rozbudowany, by pomóc w zarządzaniu hotelem w jego innych strefach takich jak kuchnia i przygotowanie odpowiedniej diety wymaganej przez wybranego klienta.

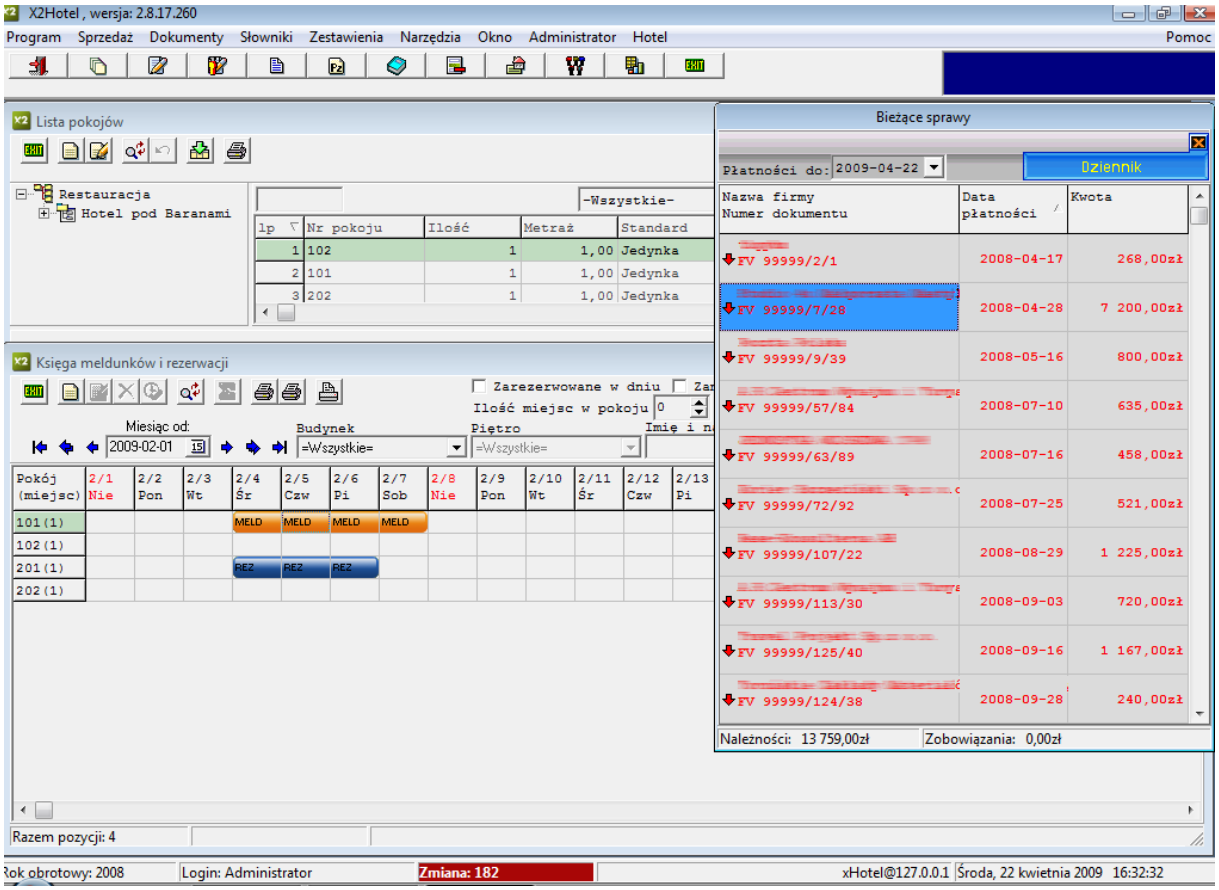
2.1.2. Hotres

Hotres [3] to inne oprogramowanie, które na celu ma obsługę rezerwacji dokonywanych przez klienta hotelu. Skupia się ona na udostępnieniu użytkownikowi możliwości dokonania rezerwacji, by następnie dane te przekazać do obsługi hotelu. Niewątpliwym plusem systemu jest integracja z największymi portalami sprzedaży takimi jak Booking.com czy Airbnb. Jednak tak jak i SOHiS oprogramowanie to nie wspiera obsługi hotelu pod kątem zarządzania innymi częściami obiektu niż recepcja.

2.1.3. X2 Hotel

Kolejnym systemem który zasługuje na uwagę osoby obsługującej obiekt wynajmujący pokoje jest program X2 Hotel [4]. Jest to program najbardziej rozbudowany. Poza podstawowymi funkcjami zarządzania recepcją umożliwia również synchronizację z innymi miejscami wewnątrz hotelu, które oferują sprzedaż. Korzyścią płynącą z takiego rozwiązania jest przesyłanie rozliczeń na jeden rachunek klienta. Przykładami miejsc wykorzystywania może być hotelowa

restauracja czy kiosk. System obsługuje też integrację z systemami innych firm, które zarządzają zamkami elektronicznymi w drzwiach pokojów hotelowych. Program jednak z powodu tak szerokiego rozbudowania jest wizualnie przestarzały i mało intuicyjny rys. 2.1.



Rys. 2.1. Przykład interfejsu użytkownika programu X2 Hotel [4]

2.1.4. Porównanie

Powyżej wymienione programy mają swoje zastosowania oraz niewątpliwe plusy. Hotres jednak [3] skupia się bardziej na kliencie hotelu, a X2 Hotel [4] jest programem bardzo rozbudowanym ale jest mało intuicyjny w użytkowaniu. Celem systemu tworzonego w tej pracy jest udostępnienie funkcji potrzebnych dla każdej jednostki obsługi hotelu jednocześnie pozostając prostym w obsłudze. Przedstawione oprogramowania skupiają się w jednym punkcie - recepcji, gdy stworzone rozwiązanie pozwala na interakcję z systemem użytkownikom mniejszych działów obiektu hotelowego.

2.2. Architektura trójwarstwowa aplikacji desktopowej

Aplikacja desktopowa trójwarstwowa składa się z logicznych i fizycznych warstw obliczeniowych. Należą do nich warstwa prezentacji nazywana front-endem, warstwa aplikacji, przetwarzania wczytanych danych oraz warstwa danych nazywana back-endem aplikacji. Każdej z warstw towarzyszą inne cele oraz sposoby ich osiągania. Dzięki tak wyraźnemu podziałowi aplikacja może być równolegle rozwijana w każdej ze swoich warstw bez konfliktów z pozostałymi jej częściami. Kolejnym plusem takiej budowy jest zwiększenie niezawodności. Jeśli jedna z warstw ulegnie awarii pozostałe części aplikacji w mniejszym stopniu dotyka brak dostępności czy zmniejszona wydajność. Jest to dominująca architektura aplikacji [1] wykorzystywana przy oprogramowaniach typu klient - serwer.

2.3. Warstwa prezentacji

Jest to interfejs użytkownika, poprzez który dokonuje się komunikacji z aplikacją. Celem priorytetowym dla tej warstwy jest wyświetlanie danych użytkownikowi jak i pobieranie nowych, wprowadzonych informacji do systemu. Pod względem poziomu warstw aplikacji jest ona usytuowana najwyżej. Ta część systemu może być uruchamiana w miejscach takich jak przeglądarka WWW, czy tak jak w przypadku stworzonego oprogramowania w postaci graficznego interfejsu użytkownika (Graphical User Interface - GUI). Dla aplikacji uruchamianych bezpośrednio na systemie operacyjnym, GUI może być programowane w wielu różnych językach dopasowując go do platformy. Dla języka programowania Python, GUI można stworzyć za pomocą framework-ów.

Do najpopularniejszych rozwiązań należą Kivy oraz Tkinter. Kivy jest nowocześniejszym rozwiązaniem z możliwością obsługi dotyku w kilku miejscach jednocześnie na urządzeniach mobilnych. Dostępny jest również do programowania interfejsu użytkownika na komputery osobiste umożliwiając atrakcyjne wizualnie rozwiązania.

Tkinter jest starszym framework-iem dostępnym dla programistów i prostym w obsłudze, jednak wymagającym bardzo dużego nakładu pracy aby osiągnąć GUI atrakcyjne wizualnie, w porównaniu do pozostałych aplikacji dostępnych na rynku.

2.4. Warstwa aplikacji

Warstwa ta występuje jeszcze pod innymi nazwami, takimi jak, warstwa logiki, bądź warstwa środkowa. W tym miejscu podejmowana jest decyzja co należy zrobić z otrzymanymi od użytkownika danymi. Informacje te można zestawić z innymi, bądź przekazać je dalej,

w głąb aplikacji, na przykład do bazy danych. Warstwa ta jednak nie tylko wykonuje czynności na otrzymanych danych, ale również wykonując polecenie użytkownika. Może na przykład, nadpisać bądź usunąć wskazane inne dane, dostępne na niższym poziomie oprogramowania.

Poziom ten jest łącznikiem pomiędzy warstwami prezentacji oraz danych. Bez niego nie byłoby możliwości komunikacji między najwyższą i najniższą warstwą.

2.5. Warstwa danych

Warstwa danych nazywana jest również warstwą bazy danych, bądź warstwą zaplecza. Na tym poziomie aplikacji przechowywane i zarządzane są wszelkie zapisane wcześniej informacje potrzebne do działania oprogramowania, lub wykonania polecenia przekazanego z warstwy prezentacji poprzez warstwę aplikacji. Funkcję tę może pełnić system zarządzania relacyjnymi bazami danych taki jak PostgreSQL, czy też serwer bazodanowy, na przykład NoSQL.

3. Projekt systemu

W tym rozdziale skupiono się na projekcie systemu wraz z jego dokumentacją techniczną, w której skład wchodzą wymagania, architektura, przypadki użycia oraz diagramy.

3.1. Wymagania

Wymagania podzielono na funkcjonalne oraz niefunkcjonalne. Pierwsze z nich skupiają się na możliwościach systemu w zależności od zachowania użytkownika. Wymagania niefunkcjonalne natomiast skupiają się na aspektach technicznych aplikacji takich jak, czas oczekiwania na wykonanie akcji.

3.1.1. Wymagania funkcjonalne

Wymaganiami funkcjonalnymi, które ma spełniać system, są:

- logowanie się do aplikacji, gdzie proces ten uwzględnia uprawnienia logującego się użytkownika przekierowując go na odpowiednią stronę po prawidłowym zalogowaniu;
- system powinien posiadać zaawansowany system sprawdzania poprawności wpisanych danych takich jak: adres e-mail, numer telefonu, format daty, czy też liczby posiłków do zamówienia na podstawie możliwej maksymalnej liczby gości w pokojach;
- rezerwacja pobytu klienta przeprowadzona w krokach: wyboru daty, wyboru pokoi, zapisania danych osobistych klienta oraz opcjonalnego wyboru diet gości hotelowych;
- liczba pokoi, które można zarezerwować powinna być ograniczona jedynie przez dostępność w wybranym okresie czasowym;
- możliwość przeglądania rezerwacji wraz z wbudowaną wyszukiwarką, która wyszukuje po wpisanej wartości w pole tekstowe oraz przypisanej do niego kategorii;
- możliwość edycji rezerwacji pokoju jak i diet;

– możliwość usuwania rezerwacji;

– możliwość wyboru diety dla klienta po wcześniejszym zarezerwowaniu;

– możliwość prowadzenia rejestru obecności klientów;

– Możliwość sprawdzenia wymaganej do wykonania liczby posiłków w dniu obecnym i następnym;

– możliwość odliczania wykonanych już posiłków w dniu obecnym i następnym;

– możliwość sprawdzenia wszystkich wymaganych posiłków na obecny jak i przyszły tydzień;

– możliwość podglądu obecności klientów w pokojach oraz zapisywania który pokój był już odwiedzony przez zespół sprzątający;

– możliwość dodania nowych użytkowników systemu oraz nadania im stosownych uprawnień.

3.1.2. Wymagania niefunkcjonalne

Poniżej zostały opisane, zaprezentowane wymagania niefunkcjonalne.

Bezpieczeństwo

Aplikacja powinna być lokalna, wszelkie informacje przechowywane są na serwerze bazy danych, bądź w formie tymczasowej podczas korzystania z aplikacji w pamięci komputera. Użytkownicy, przed wykonywaniem operacji wewnątrz aplikacji, muszą potwierdzić swoją tożsamość poprzez proces logowania. Dzięki temu osoby korzystające z systemu powinny mieć dostęp jedynie do funkcji systemu, które są zgodne z ich uprawnieniami.

Kompatybilność

Aplikacja powinna działać na wszystkich urządzeniach z systemem Windows oraz zainstalowanym: językiem programowania Python 3.7 lub wyższym, biblioteką Kivy 2.0 oraz biblioteką Psycopg2. Wymagany jest również uruchomiony serwer z danymi potrzebnymi do poprawnego działania systemu.

Ergonomiczność

Aplikacja ma być prosta w użyciu i zrozumiała po krótkim wprowadzeniu.

Szybkość pracy

Do pełnej sprawności aplikacja nie powinna uruchamiać się dłużej niż 10 sekund, a wszelkie operacje wewnątrz aplikacji nie powinny trwać dłużej niż 3 sekundy.

Przydatność

Aplikacja powinna ułatwiać najprostsze czynności z zakresu zarządzania hotelem, niezależnie od stanowiska, na którym system będzie wykorzystywany.

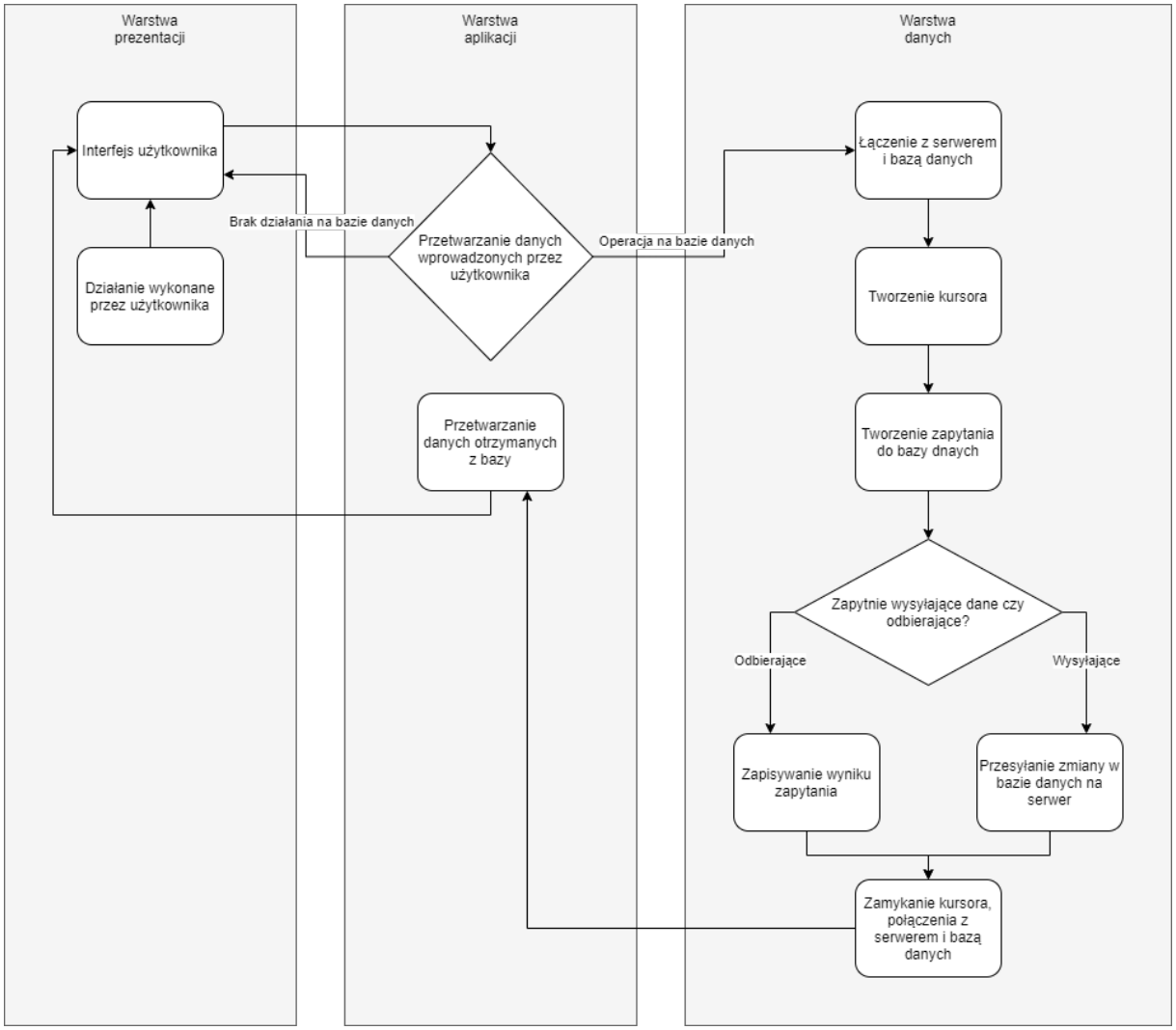
3.2. Architektura systemu

Przedstawiona graficzna reprezentacja architektury sytemu rys. 3.1. jest podzielona zgodnie z definicją trójwarstwowej aplikacji opisanej w rozdziale 2.2. W warstwie prezentacji użytkownik ma styczność z graficznym, interfejsem przez który komunikuje się z aplikacją. Działania użytkownika powodują wywoływanie metod w warstwie aplikacji, które decydują czy operacje do wykonania będą potrzebowały informacji w bazie danych. W wypadku gdy baza danych nie jest wymagana do wykonania polecenia, funkcje odpowiadają za przygotowanie danych do wyświetlenia przez warstwę prezentacji. Gdy baza danych jest wymagana, informacje przekazywane są do warstwy danych, gdzie następuje łączenie z serwerem oraz bazą danych, by następnie przekazać poprzez kursor zapytanie do bazy. Po wykonaniu zapytania, niezależnie od tego czy dane zostały wysłane czy też pobrane, następuje zamknięcie kursora oraz połączenia z serwerem i bazą, by bezpiecznie przekazać informacje warstwie aplikacji, która przygotuje otrzymane dane dla interfejsu użytkownika w warstwie prezentacji w celu wizualizacji.

3.3. Diagramy

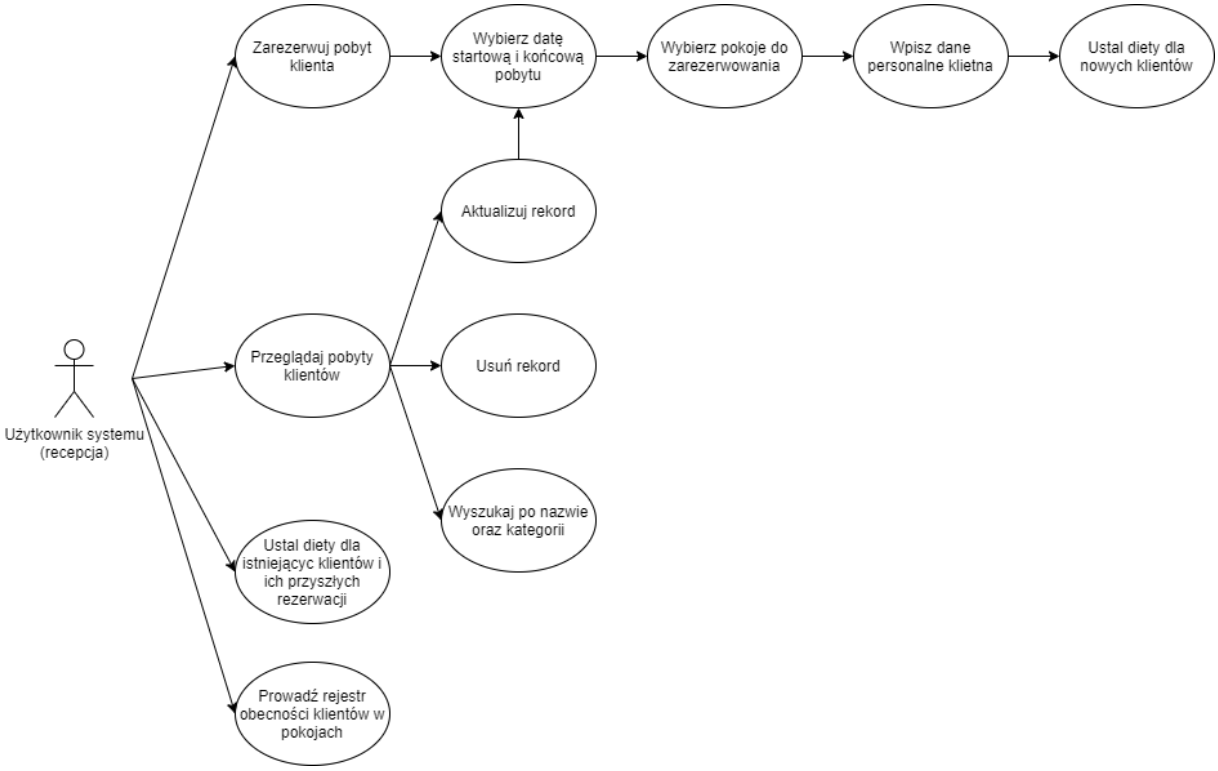
3.3.1. Diagramy przypadków użycia

Diagram rys. 3.2. przedstawia przypadki użycia dla użytkownika z uprawnieniami udzielanymi dla obsługi recepcji. Użytkownik może wykonać sekwencję czynności wymaganych do zarezerwowania pokoju. Dostępną opcją jest również przeglądarka rezerwacji pobytów z możliwością usuwania zaplanowanej wizyty, filtrowania rekordów oraz aktualizacji wybranego pobytu. Wybranie ostatniej opcji przekieruje użytkownika do pierwszego okna widzianego w trakcie rezerwacji. Użytkownik z tymi uprawnieniami może również ustalić dietę dla gości już po dokonaniu rezerwacji. Ostatnim przypadkiem użycia jest rejestrowanie, przez personel, obecności klientów w pokojach.



Rys. 3.1. Architektura systemu

Diagram rys. 3.3. przedstawia przypadki użycia dla użytkownika z uprawnieniami dla obsługi kuchni. Jednym z przypadków użycia jest możliwość przeglądania liczby dań do wykonania w dniu dzisiejszym i jutrzejszym w celach organizacji pracy. Dla dalszego usprawnienia można prowadzić odliczania pozostałych do wykonania posiłków. Drugim przypadkiem jest możliwość zobaczenia liczby dań w bieżącym tygodniu jak i następnym w celu ułatwienia planowania zakupów spożywczych.



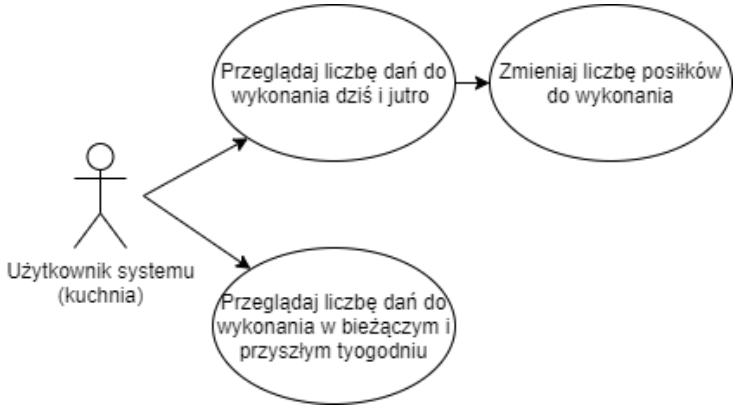
Rys. 3.2. Diagram przypadków użycia recepcjonisty

Diagram rys. 3.4. przedstawia jedyny przypadek użycia dla administratora, którym jest dodanie nowego użytkownika systemu poprzez nadanie mu loginu, hasła oraz uprawnień.

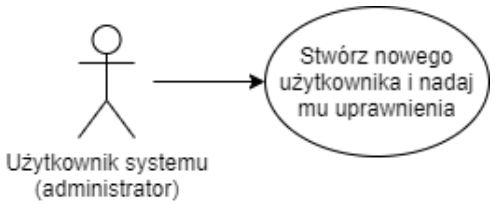
Diagram przypadków użycia rys. 3.5. dla użytkownika o uprawnieniach sprzątacza hotelu przedstawia możliwość przeglądania informacji, o tym czy, klient przebywa obecnie w pokoju. Dzięki temu zespół sprzątający wie kiedy może wejść do konkretnego pokoju w celach wykonania usługi. Drugą możliwością jest zapisywanie pokoi, które zostały już odwiedzone w ostatnim czasie, aby usprawnić proces planowania obsłużenia pomieszczeń.

3.3.2. Diagram czynności

Pierwszą czynnością, którą ukazuje diagram rys. 3.6. jest proces logowania, który uruchamia się po starcie aplikacji. Podczas próby logowania wpisane dane są walidowanie. Jeśli proces ten przebiegnie pomyślnie użytkownik zostanie zalogowany, w przeciwnym wypadku użytkownik otrzyma stosowny komunikat i pozostanie na stronie logowania. Po pomyślnym zalogowaniu użytkownik zostanie przekierowany do jednego z czterech okien, w zależności od uprawnień, które posiada zalogowany. Poniżej opisano po kolei każdą z czterech ścieżek.



Rys. 3.3. Diagram przypadków użycia kucharza



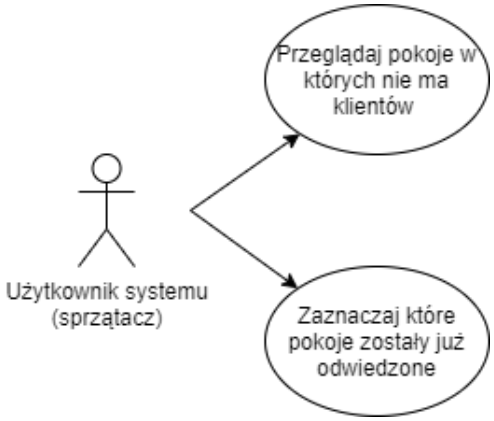
Rys. 3.4. Diagram przypadków użycia administratora

Zespół sprzątający

W oknie obecności klientów i odwiedzonych pokoi użytkownik może naciskać przyciski odpowiadające konkretnym pokojom, oznaczając je jako odwiedzone. Użytkownik może również wylogować się i wrócić do okna logowania naciskając przycisk ”LOGOUT”.

Kuchnia

Użytkownik z uprawnieniami kuchennymi po zalogowaniu widzi trzy przyciski: "DAILY SCHEDULE", ”INGREDIENT” oraz ”LOGOUT”. Ostatnim z nich użytkownik może się wylogować i wrócić do okna logowania. Pierwszy przycisk wyświetli okno zawierające informacje o daniach do przygotowania w danych dniach. Wewnątrz tego widoku użytkownik może naciskać przyciski zmniejszające liczbę dań do wykonania, przycisk "MARK TODAY AS DONE"wyzeruje licznik dań do wykonania, a przycisk "BACK"wyświetli okno początkowe dla kucharza. Drugi przycisk ("INGREDIENTS") w menu po zalogowaniu przedstawi przycisk "BACK", który wróci użytkownika do menu kucharza, oraz tabelę pokazującą liczbę dań w tym i przyszłym tygodniu.



Rys. 3.5. Diagram przypadków użycia sprzątacza

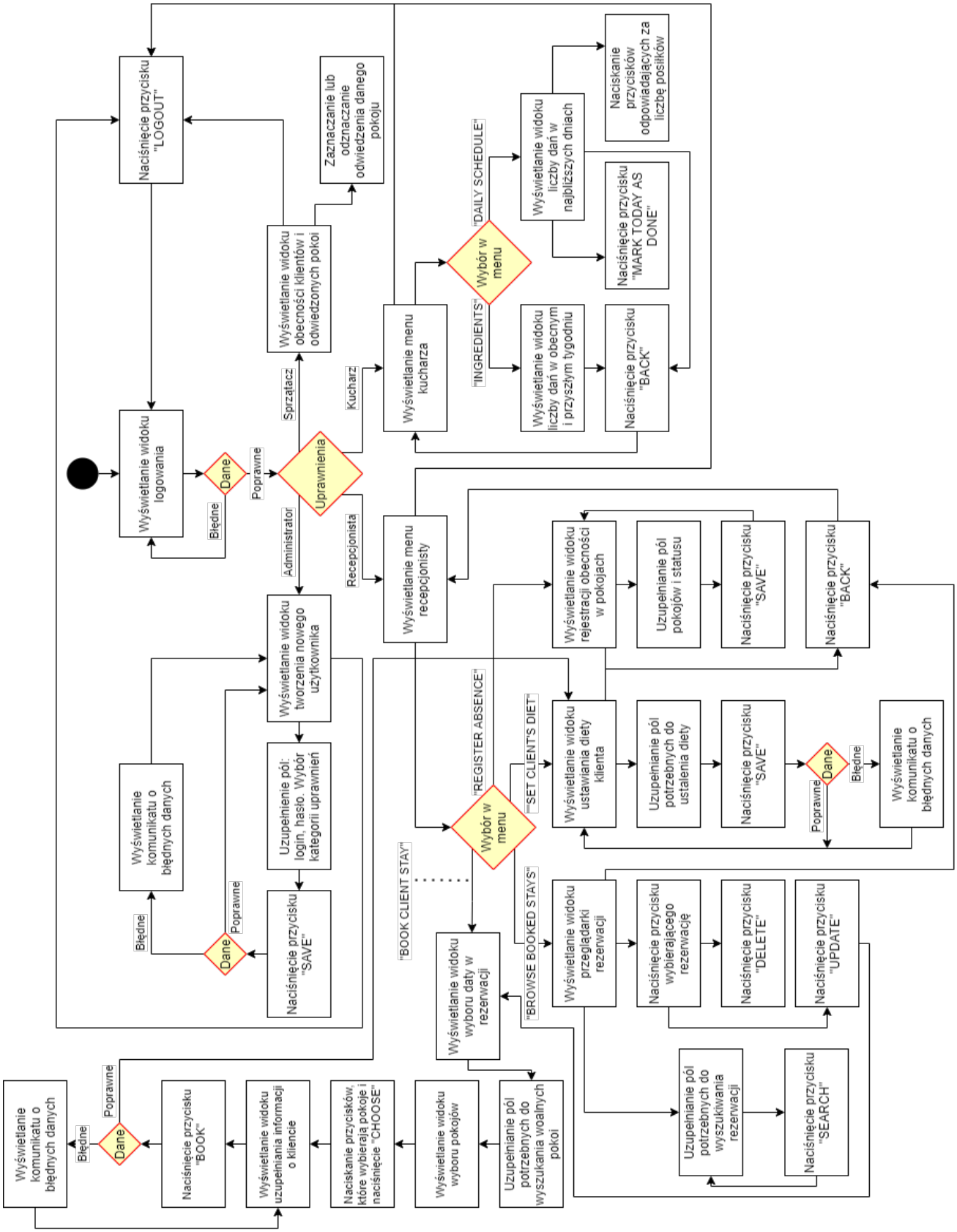
Administrator

Po poprawnym zalogowaniu danymi, przypisanymi do uprawnień administratora, pojawia się okno, które umożliwia wypełnienie pól odpowiadających za login, hasło oraz uprawnienia użytkownika, który zostanie stworzony po naciśnięciu przycisku ”SAVE". Przycisk ”LOGOUT” wylogowuje użytkownika do okna logowania.

Recepcja

Użytkownikowi z tymi uprawnieniami udostępniono menu, które umożliwia wybór spośród czterech, różnych opcji. Pierwsza z nich możliwa jest po naciśnięciu w przycisk "BOOK CLIENT’S STAY"i przedstawia okno do wyboru daty startowej i końcowej pobytu klienta. Widok ten pozwala na przejście do kolejnego ekranu aplikacji, po poprawnym wypełnieniu pól i naciśnięciu przycisku ”SEARCH”. W następnym kroku wyświetli się okno wyboru pokoi, gdzie użytkownik może zaznaczyć wszystkie wyświetlone pokoje, które nie są koloru czerwonego, który oznacza pokój już zarezerwowany w wybranym wcześniej terminie rezerwacji. Po dokonaniu wyboru i naciśnięciu przycisku ”CHOOSE” użytkownik przechodzi do okna w, którym należy uzupełnić informacje personalne klienta. Uzupełniając pola poprawnie i używając przycisku ”BOOK” recepcjonista rezerwuje pobyt klienta i przechodzi do okna ustawiania diety. Widok ten jest również dostępny bezpośrednio z menu użytkownika z uprawnieniami przeznaczonymi dla recepcji. Okno dotyczące diet umożliwia ustawienie ilości konkretnych diet w zależności od liczby pokojów zarezerwowanych na klienta, którego dane zostały podane. Zapisywanie informacji odbywa się za pomocą przycisku ”SAVE”. Trzecim widokiem dostępnym w menu głównym recepcjonisty jest przeglądarka rezerwacji, która pozwala na wybieranie pozycji z listy i aktualizowanie jej poprzez naciśnięcie "UPDATE"(które przenosi nas do okna rezerwacji), bądź usunięcie dzięki "DELETE". Ten ekran aplikacji pozwala również na wyszukiwanie konkretnych rekordów poprzez uzupełnienie odpowiednich danych i naciśnięcie

przycisku ”SEARCH”. Ostatni główny widok recepcjonisty dostępny jest po naciśnięciu przycisku ”REGISTER ABSENCE” w menu głównym po zalogowaniu. Po wypełnieniu pól pokojów i statusu oraz wciśnięciu przycisku ”SAVE” użytkownik rejestruje obecność bądź nieobecność klientów w pokojach.



Rys. 3.6. Diagram czynności użytkownika

3.3.3. Diagram klas

W diagramie klas zamieszczonym poniżej rys. 3.7 przedstawiono zależności między najważniejszymi klasami aplikacji. Mając na celu uproszczenie schematu, pominięto klasy, które działają w podobny sposób do tych już przedstawionych.

HotelManager

Jest to najważniejsza klasa aplikacji, która dziedziczy po klasie App z biblioteki Kivy, odpowiadającej za graficzny interfejs. To ona uruchamia tworzenie graficznego interfejsu użytkownika za pomocą metody build() oraz przechowuje informacje zanim trafią do bazy danych, niezależnie od tego na jakim ekranie obecnie znajduje się użytkownik.

WindowManager

Klasa ta jest punktem, w którym łączą się wszystkie pozostałe klasy odpowiadające za okna aplikacji. Dziedziczy ona po klasie ScreenManager, która jest dostępna w bibliotece Kivy. Pomimo tego, że sama nie wykonuje żadnych operacji, nie posiada zmiennych czy też metod, to bez niej nie byłoby możliwe przełączanie pomiędzy oknami systemu. Jest wielokrotnie wywoływana w celu zmiany widoku.

LoginWindow

Jest to pierwsza klasa, która jest widoczna bezpośrednio przez użytkownika, ponieważ odpowiada za ekran logowania. Dziedziczy po Screen z biblioteki Kivy, tak jak każda klasa, która przedstawia interfejs. Zmienna login odpowiada za wartość wpisaną w polu tekstowym interfejsu przeznaczonym na login użytkownika, adekwatna sytuacja występuje przy zmiennej password. Zmienna window odpowiada za przekazanie WindowManager, które okno należy przedstawić po prawidłowym zalogowaniu. Metoda loginPress() uruchamia proces walidacji danych wpisanych przez użytkownika, jeśli zwróconą wartością boolean będzie True, logowanie powiodło się i zwracany string przedstawia nazwę następnego okna, które jest adekwatne do uprawnień użytkownika. W przypadku zwróconą wartością będzie False, użytkownikowi przedstawainy jest komunikat o nieprawidłowych danych, zmuszając do ponownego wykonania procesu logowania.

MaidWindow

Jest pierwszą z omawianych klas, która odpowiada za pojawianie się okna, po zalogowaniu do systemu. Jest to proste okno, które nie posiada własnych zmiennych, ponieważ użytkownik nie może dokonać zmian w bazie danych, bądź jego aktywność w systemie nie jest przekazywana do żadnej innej instancji. Metoda absentOrPresent() przeszukuje bazę danych w celu znalezienia informacji o obecności klientów w pokojach. Dane te są potem wyświetlane na ekranie. Funkcja ta jest wywoływana w trakcie wchodzenia na to okno systemu. Metoda selectAsVisited() zajmuje się natomiast zaznaczaniem kolorystycznym, który z pokojów został już dziś odwiedzony przez zespół sprzątający.

ActionWindow

Nie posiada ona żadnych metod ani zmiennych, gdyż odpowiada jedynie za wyświetlenie okna z menu dostępnych opcji do wykonania przez pracownika recepcji. Naciskając na przycisk, aplikacja przekierowuje użytkownika do stosownego, następnego okna.

DateWindow

Jest to jedno z bardziej skomplikowanych klas pod względem algorytmów, które wykonują się w reakcji na działania użytkownika. Wszystkie zmienne ObjectProperty odpowiadają za informacje zawarte w adekwatnych do nazwy miejscach, przykładowo zmienna startDay odpowiada za wybrany dzień rozpoczęcia pobytu w hotelu. Zmienna months jest listą zawierającą nazwy miesięcy do wyświetlenia użytkownikowi wewnątrz pola dropdown, gdzie dokonuje się wyboru miesiąca. Metoda startUpdate() wykorzystywana jest, gdy zostanie naciśnięty stosowny przycisk w oknie BrowserWindow, który uruchomi proces aktualizacji rezerwacji i przekieruje użytkownika do tego okna. Metoda abortUpdate() przywraca stan rezerwacji do formy przed jej wykonywaniem, gdy użytkownik naciśnie przycisk powrotu do okna BrowserWindow, więcej o tej klasie napisano poniżej. Metoda resetToDefaults() przywraca okno do pierwotnej formy, w przypadku ponownego uruchomienia tego okna, korzysta ona również z możliwości metody abortUpdate() do bezpiecznego zarządzania pozostawionymi danymi wewnątrz niedokończonej rezerwacji. Metoda daysInSelectedMonth() dba o walidację danych wyboru dnia nie pozwalając na wybór nieistniejącego numeru w danym miesiącu. Funkcja limitSpinner() ogranicza rozwijalne menu dni, miesięcy i lat do poziomu maksymalnie trzech pozycji w celach estetycznych. wrongTimePeriod() jest kolejną metodą walidacyjną, skupiającą się na prawidłowym okresie czasowym, nie pozwalając na ustalenie daty startowej na późniejszą od końcowej. Funkcja searchPress() jest wywoływana po naciśnięciu przycisku ”SEARCH”. Odpowiada ona za pobranie danych od metod walidacyjnych. Gdy dane są poprawne, przekierowuje użytkownika do

następnego okna etapu rezerwacji, w przeciwnym razie wyświetla komunikat o błędnych danych.

RoomWindow

Klasa ta jest również oknem wewnątrz aplikacji. Posiada ona 30 metod, z których każda przyporządkowana jest swojemu pokojowi wewnątrz hotelu. Funkcje te odpowiadają za komunikowanie użytkownikowi o zaznaczeniu wybranego pokoju kolorystyką oraz uniemożliwiają dokonanie wyboru pokoju niedostępnego. Metoda makeRedColors() jest uruchamiana podczas tworzenia okna, pobiera ona dane z bazy w celu zaznaczenia pokojów już zarezerwowanych w danym terminie na czerwono. Gdy wybrany zostanie jeden lub więcej pokoi i naciśnięty zostanie przycisk ”CHOOSE”, użytkownik przechodzi do kolejnego okna, w przeciwnym wypadku otrzymuje komunikat o braku wybranego pokoju. Jest to warunek, który został utworzony wewnątrz przycisku.

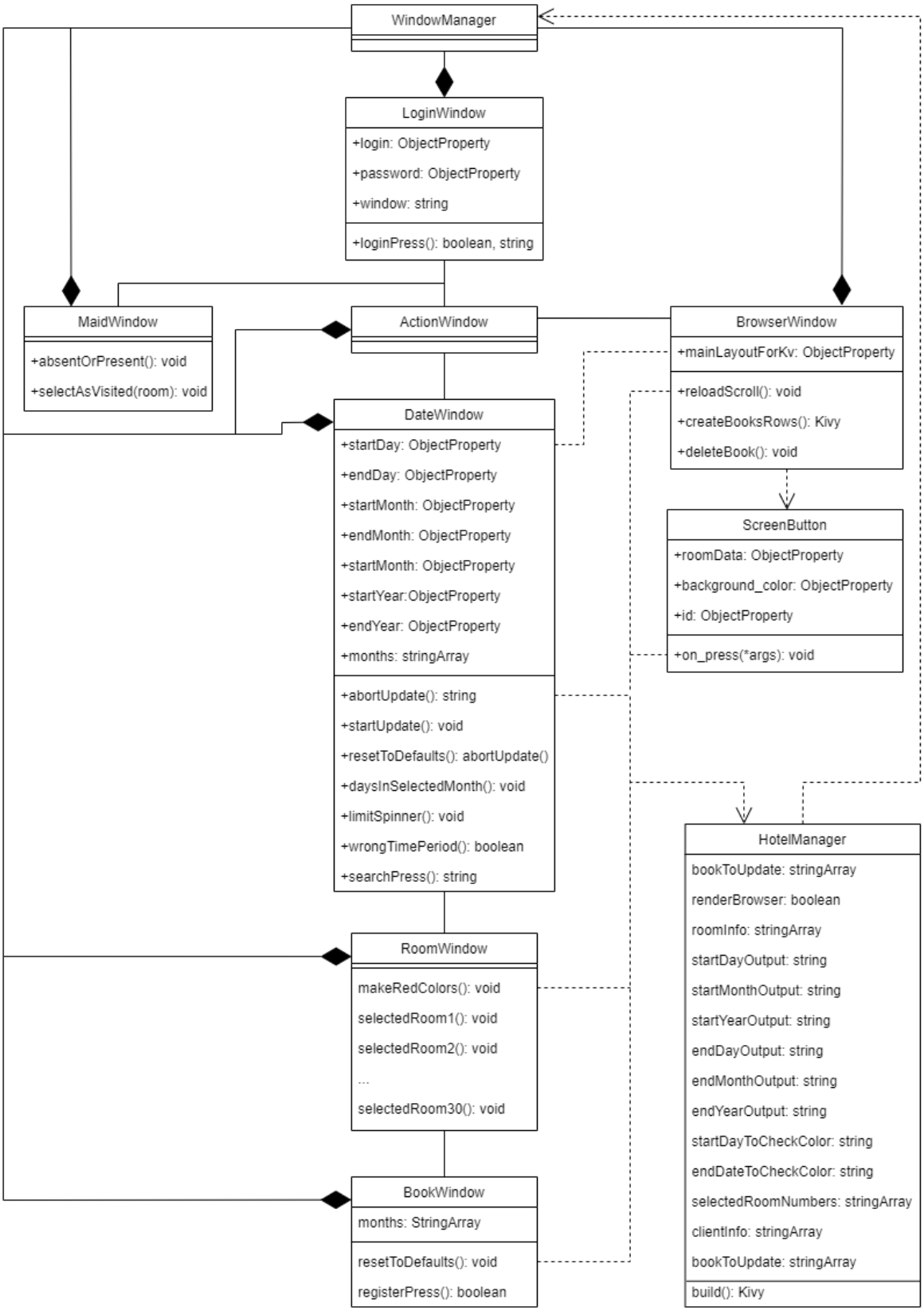
BookWindow

Jedyną unikatową metodą wewnątrz tej klasy jest registerPress(), które uruchamia walidację wprowadzonych danych osobistych klienta. Jeśli dane te są prawidłowe wszelkie informacje przekazane w poprzednich dwóch oknach, są przetwarzane i wysyłane do bazy danych. Jeśli natomiast dane nie są prawidłowe, użytkownik otrzyma stosowny komunikat.

BrowserWindow

Jest drugą klasą, która odpowiada za przedstawienie okna dostępnego z menu recepcjonisty. Klasa ta jest wyjątkowa, ponieważ w większości generowana jest dynamicznie w zależności od działania użytkownika. Zmienna mainLayoutForKv jest również inna od pozostałych zmiennych z rodzaju ObjectProperty, ponieważ nie posiada ona informacji o pojedynczym elemencie interfejsu takim jak na przykład pole tekstowe, ale o całym układzie wewnątrz okna. Umożliwia to dynamiczne generowanie elementów takich jak, nagłówki przedstawiające dane z bazy czy też przyciski do zaznaczania konkretnego rekordu do aktualizacji, bądź usunięcia. Metoda createBookRows() zajmuje się tworzeniem wewnątrz widoku listy rekordów z bazy danych, który umożliwia przewijanie. Każdy z rekordów ma przypisany własny przycisk, który zmienia kolorystykę pod wpływem interakcji z użytkownikiem, ale również zapisuje dane z wybranego rekordu. Wymagało to stworzenia spersonalizowanego przycisku dla klasy BrowserWindow. Umożliwiała to klasa **ScreenButton**, która dziedziczy po Button z biblioteki Kivy, zachowując podstawowe wartości przycisku, pozwala na zaprogramowanie niestandardowych

funkcji wymaganych podczas generowania dynamicznego okna. Funkcja deleteBook() odpowiada za usunięcie wybranego przyciskiem reokordu z bazy danych i uruchamia metodę reloadScroll(), w celu ponownego wygenerowania widoku ze zaktualizowanymi danymi.



Rys. 3.7. Diagram klas aplikacji

4. Implementacja

W tym rozdziale omówiona została implementacja najważniejszych części systemu przedstawionego na diagramie klas rys. 3.7.

4.1. Wykorzystywane technologie i narzędzia

4.1.1. Język programowania i biblioteki

Do realizacji systemu wybrano język programowania Python[5], który dzięki swojej specyfice idealnie nadaje się do implementacji aplikacji, które nie wymagają szybkości działania. Język ten jest stosunkowo prosty w obsłudze, co umożliwia wykonanie większej ilości pracy w krótszym czasie.

Dla tego języka powstał również framework Kivy, który umożliwia tworzenie graficznego interfejsu użytkownika. Praca z tym rozszerzeniem idealnie wpasowuje się w cechy projektu jak i języka Python, ponieważ jego konstrukcja jest klarowna i prosta w obsłudze, pozwalając przy tym stworzyć warstwę graficzną aplikacji, która nie wygląda archaicznie, nawet w swojej podstawowej formie.

Ostatnią wykorzystaną niestandardową biblioteką jest psycopg2, który pozwala na wykorzystywanie bazy danych SQL oraz tworzenie zapytań do niej.

4.1.2. PyCharm

PyCharm jest IDE stworzonym przez przedsiębiorstwo JetBrains, które produkuje zintegrowane środowiska deweloperskie dla wielu czołowych na rynku języków. PyCharm[6] oferuje wbudowany debugger, szeroką bibliotekę dodatków usprawniających ergonomię pracy, a do tego jest prosty w obsłudze.

4.1.3. pgAdmin4

pgAdmin4 jest narzędziem do zarządzania serwerem bazodanowym, które umożliwia stworzenie wymaganej konfiguracji do prawidłowego działania aplikacji. Tak jak pozostałe wykorzystywane narzędzia i technologie, jest ono proste w obsłudze zachowując przy tym wszelkie wymagania funkcjonalności jak na przykład możliwość utworzenia i wykonania zapytania do bazy danych.

4.2. Implementacja warstwy prezentacji

Warstwa prezentacji aplikacji tworzona jest w pełni za pomocą języka Python wzbogaconego o framework Kivy, który posiada własną składnię do tworzenia okien wewnątrz systemu. Na listingu 4.1 zamieszczono kod, który odpowiada za jedno z prostszych okien w aplikacji, `ActionWindow`, które jest swojego rodzaju menu widocznym dla użytkownika recepcji.

Listing 4.1. Przykład kodu do tworzenia okna aplikacji w Kivy.

```
<ActionWindow>:
    name: "actionWindow"
    canvas.before:
        Rectangle:
            pos: self.pos
            size: self.size
            source: "gradientBackground.png"
    BoxLayout:
        orientation: "vertical"
        spacing: 10
        padding: 150
        Button:
            text: "BOOK CLIENT STAY"
            on_release: app.root.current = "dateWindow"
            background_color: utils.get_color_from_hex("#90C2E7")
            color: (0, 0, 0, 1)
        Button:
            text: "BROWSE BOOKED STAYS"
            on_release: app.root.current = "browserWindow"
            background_color: utils.get_color_from_hex("#90C2E7")
            color: (0, 0, 0, 1)
        Button:
            text: "SET CLIENT'S DIET"
            on_release: app.root.current = "dietWindow"
            background_color: utils.get_color_from_hex("#90C2E7")
            color: (0, 0, 0, 1)
```

```
    Button:
        text: "REGISTER ABSENCE"
        on_release: app.root.current = "roomWindowAbsence"
        background_color: utils.get_color_from_hex("#90C2E7")
        color: (0, 0, 0, 1)
    RedButton:
        text: "LOGOUT"
        on_release: app.root.current = "loginWindow"
```

Klasa `ActionWindow` posiada zmienną `name`, która odpowiada za nazwę okna wewnątrz aplikacji, umożliwiając przełączanie się systemu do widoku z konkretną nazwą. Segment `canvas.before` odpowiada za umiejscowienie obrazka jako tło konkretnego okna. Fragment kodu z `BoxLayout` odpowiada za główne rozłożenie elementów wewnątrz ekranu. Przyjmuje on parametry takie jak odstępy, rozmiar, margines czy też sposób rozkładania elementów, które się w nim znajdują. W zamieszczonym przykładzie jest to wartosć `orientation: "vertical"` co oznacza, że każdy element umieszczony wewnątrz niego będzie dodawany na sam dół pozostałych. Części interfejsu, które mogą się znaleźć wewnątrz tego rozkładu strony nie muszą być tego samego typu. Mogą być to nawet inne typy rozkładu elementów takie jak na przykład `GridLayout`, który jest tabelą i może posiadać własne elementy. Tworzy to wiele możliwości zaprojektowania wizualnej części projektu, który będzie skalował się niezależnie od rozdzielczości urządzenia, na którym się znajduje. W widocznym kodzie większość elementów rozkładu naszej strony są jednak typu `Button`, który oznacza przycisk możliwy do naciśnięcia. Przyjmuje on własne parametry takie jak, kolor tła, tekstu czy akcję, którą należy wykonać po skorzystaniu z niego. W tym wypadku jest to przeniesienie użytkownika na inny ekran aplikacji za pomocą przykładowego polecenia `on_relese: app.root.current = "dateWindow"`, które ma zmienić obecnie generowane okno na nowe o nazwie `dateWindow`. Wyjątkowym elementem, z którego składa się okno jest `RedButton`, który jest stworzoną klasą, dziedziczącą po klasie `Button` z biblioteki Kivy. Stworzono ją w celu uproszczenia wielokrotnej implementacji przycisku o podobnej budowie, a zaprezentowano ten proces w listingu 4.2.

Listing 4.2. Przykład tworzenia klasy dziedziczącej po klasie z Kivy.

```
<RedButton@Button>
    background_color: utils.get_color_from_hex("#A31621")
    color: (1, 1, 1, 1)
    size_hint_y: None
    height: 50
    font_size: 20
```

Ostatnim i zarazem najważniejszym elementem warstwy prezentacji jest stworzenie odpowiedniej metody w języku Python, bez której okno aplikacji nie mogło by istnieć. Jest to klasa, która nie posiada żadnych własnych metod. Przedstawiona została w listingu 4.3.

Listing 4.3. Przykład kasy tworzącej okno aplikacji.

```
class ActionWindow(Screen):
    pass
```

Cała warstwa prezentacji działa w podobny sposób, niezależnie od poziomu skomplikowania okna. Wyjątkiem jest jednak dynamiczne generowanie graficznego interfejsu użytkownika, co zostało przedstawione poniżej. W listingu 4.4 przedstawiono mały fragment metody wywoływanej podczas otwierania okna przeglądarki rezerwacji. Ta część funkcji odpowiada za przypisanie do klasy `BrowserWindow` elementów interfejsu użytkownika znajdujących się wewnątrz elementu `layout`, poprzez charakterystyczną dla Kivy funkcję `add_widget(layout)`. Parametr wywoływany wewnątrz tej metody jest elementem interfejsu `GridLayout`, który był omawiany wraz z listingiem 4.1. Do tego głównego elementu dodawane są inne, takie jak `label` czy spersonalizowany przycisk `ScreenButton`. Następnie po zebraniu wszystkich elementów, w jednej interacji pętli, całość dodawana jest do okna aplikacji jako swojego rodzaju paczka. Proces ten powtarzany jest w zależności od ilości zaczerpniętych danych z bazy, które były pobierane wcześniej, podczas wywoływania funkcji.

Listing 4.4. Przykład dynamicznego generowania interfejsu.

```
for i in range(len(dataToBrowse)):
    oneRoomData = dataToBrowse[i]
    for j in oneRoomData:
        idButton = idButton + 1
        startDate = j[3][6:] + "-" + j[3][4:6] + "-" + j[3][:4]
        endDate = j[4][6:] + "-" + j[4][4:6] + "-" + j[4][:4]
        layout.add_widget(Label(text=str(j[0]), color=(0, 0, 0, 1),
            size_hint_x=None, width=50))
        layout.add_widget(Label(text=str(j[1]), color=(0, 0, 0, 1)))
        layout.add_widget(Label(text=str(j[2]), color=(0, 0, 0, 1)))
        layout.add_widget(Label(text=startDate, color=(0, 0, 0, 1),
            size_hint_x=None, width=90))
        layout.add_widget(Label(text=endDate, color=(0, 0, 0, 1),
            size_hint_x=None, width=90))
        layout.add_widget(ScreenButton(text="SELECT", color=(0, 0, 0, 1),
                                background_color=(144 / 255, 194 / 255, 231 /
                                    255, 1),
                                size_hint_y=None, height=30,
                                    size_hint_x=None, width=60,
                                roomData=j, id="roomBrowse" + str(idButton)))
```

```
scrollRoot.add_widget(layout)
```

4.3. Implementacja warstwy aplikacji

Warstwa aplikacji ma styczność jednocześnie z warstwą prezentacji jak i danych, ponieważ jest przekaźnikiem i translatorem informacji między tymi dwiema warstwami. Przykładem pokazującym to idealnie jest metoda `loginPress()`, zamieszczona w listingu 4.5, która wywoływana jest po naciśnięciu przycisku ”LOGIN” na ekranie logowania do systemu rys. 5.1.

Listing 4.5. Przykład wywoływania funkcji przez warstwę prezentacji.

```
RedButton:
   text: "LOGIN"
   font_size: 30
   on_release: app.root.current = root.loginPress()[1] if
      root.loginPress()[0] else Factory.LoginPopup().open()
```

Poniżej skupiono się na metodzie zamieszczonej w listingu 4.6. Dane pobierane z warstwy prezentacji, to przykładowo tekst, który został wpisany jako login czy hasło, pobierane są za pomocą instrukcji `login = self.login.text`. Gdzie `self` odnosi się do okna obecnie wyświetlanego użytkownikowi, `.login` odpowiada za id obiektu, z którego pobierane są dane, a `.text` za paramter tego obiektu, którego wartość można zapisać do zmiennej o nazwie `login`. Dane te są następnie porównywane z danymi otrzymanymi z bazy (w jaki sposób to następuje omówiono w podrozdziale 4.4) by potem przekazać informacje do warstwy prezentacji o zmianie okna do wyświetlenia w stosunku do uprawnień użytkownika, bądź o poleceniu wypisania komunikatu o błędnych danych wpisanych przez osobę korzystającą z systemu.

Listing 4.6. Przykład działania funkcji z warstwy aplikacji.

```
def loginPress(self):
      conn = psycopg2.connect(host="localhost", database="hotel",
         user="postgres", password="admin")
      cur = conn.cursor()

      isCorrect = True
      login = self.login.text
      password = self.password.text
      cur.execute("SELECT * from users")
      userData = cur.fetchall()
      foundLogin = False
      for user in userData:
```

```
            if login == user[0]:
                foundLogin = True
                if password == user[1]:
                    if user[2] == "reception":
                        self.window = "actionWindow"
                    elif user[2] == "kitchen":
                        self.window = "kitchenWindow"
                    elif user[2] == "maid":
                        self.window = "maidWindow"
                    else:
                        self.window = "adminWindow"
                else:
                    isCorrect = False
                break
        if not foundLogin:
            isCorrect = False
        conn.commit()
        cur.close()
        conn.close()
        self.password.text = ""
        return isCorrect, self.window
```

Istnieją również funkcje, które skupiają się tylko i wyłącznie na parsowaniu danych i nie mają bezpośredniego kontaktu z pozostałymi warstwami aplikacji, lecz są one wielokrotnie wykorzystywane przez inne metody i klasy, które operują na danych i które nie mają wspólnego formatu dla różnych warstw. Idealnym przykładem jest metoda `dataParsing()`, która zmienia format otrzymanych danych z warstwy aplikacji w taki sposób, aby mogły one być odebrane przez bazę danych, która opiera się na innym systemie przechowywania dat.

4.4. Implementacja warstwy danych

Warstwa danych skupia się bezpośrednio na komunikacji z bazą danych. Jeden z prostszych przykładów wykorzystania tej warstwy jest zaprezentowany na listingu 4.6, gdzie pobierane są dane na temat użytkowników systemu w celu walidacji poprawności informacji wpisanych podczas logowania. Cały proces rozpoczyna się od ustanowienia połączenia z bazą danych, następuje to za pomocą instrukcji `conn = psycopg2.connect()`. Kolejnym krokiem potrzebnym do otrzymania informacji z bazy, bądź ich wysłania do niej, jest stworzenie kursora za pomocą `cur = conn.cursor()`, jest on mechanizmem pozwalającym na przeszukiwanie rekordów w bazie danych. Następnym elementem potrzebnym do otrzymania danych jest skorzystanie z kursora poprzez wywołanie `cur.execute("Zapytanie SQL")`, gdzie jako parametr typu string możemy przekazać zapytanie do bazy danych. Po otrzymaniu odpowiedzi,

należy pamiętać by zebrać te dane dzięki `userData = cur.fetchall()`. W tym momencie wszelkie dane, które udało się uzyskać z bazy, zostały przechowane w formie listy w zmiennej `userData`, dzięki czemu można z nich korzystać bez obawy o jakiekolwiek uszkodzenie, usunięcie czy też nadpisanie. Należy jednak pamiętać o tym, żeby wszelkie połączenia z bazą były również poprawnie obsłużone po skorzystaniu z bazy danych. Pierwszym krokiem, który należy wykonać jest wysłanie wszelkich zmian do bazy za pomocą `conn.commit()`. Następnym elementem jest zamknięcie kursora za pomocą `cur.close()` i ostatecznie należy zamknąć połączenie z bazą danych `conn.close()`. W omawianym przykładzie baza była wykorzystywana poprzez proste zapytanie `SELECT` w celu otrzymania danych o użytkownikach. W systemie występują znacznie bardziej skomplikowane zapytania, które mimo to działają na takich samych zasadach jak przed chwilą omówione. Przykład takiego wykorzystania warstwy danych zamieszczono w listingu 4.7. Przedstawiono w nim pobieranie danych z każdego pokoju i przypisanych do nich klientów.

Listing 4.7. Przykład zapytania SQL.

```
cur.execute(
   "SELECT room" + str(roomNumber + 1) + ".roomnumber, clients.lastname,
      clients.email, room" + str(
      roomNumber + 1) + ".startdate, room" + str(
      roomNumber + 1) + ".enddate FROM clients JOIN room" + str(
      roomNumber + 1) + " ON clients.clientid = room" + str(
      roomNumber + 1) + ".clientid ORDER BY room" + str(roomNumber + 1) +
         ".startdate, room" + str(
      roomNumber + 1) + ".clientid, room" + str(roomNumber + 1) +
         ".enddate")
   dataN = cur.fetchall()
```

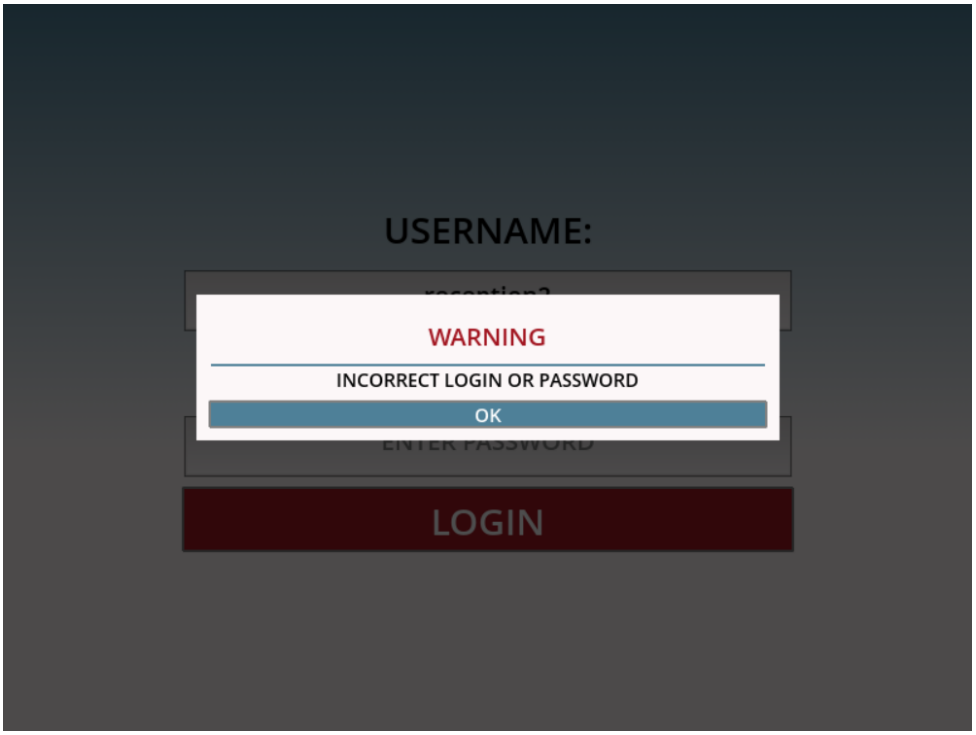
5. Testy

Ostatnim etapem pracy nad systemem są testy. Sprawdzają one poprawność stworzonego systemu zarówno pod względem konkretnych przypadków użycia jak i zgodności z wymaganiami. Poniżej przedstawiono kilka przykładowych wyników testów.

5.1. Testy modułowe

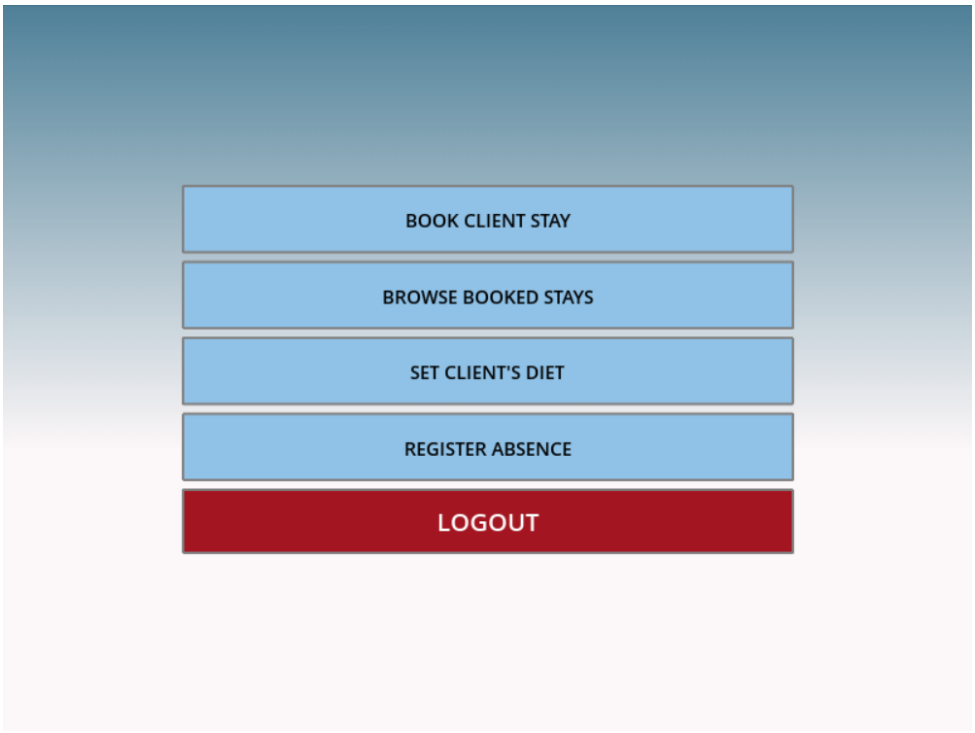
5.1.1. Poprawność systemu logowania

W teście sprawdzono reakcję systemu na wprowadzenie przez użytkownika prawidłowych jak i nieprawidłowych danych podczas logowania do systemu.



Rys. 5.1. Wynik testu po wpisaniu błędnych danych.

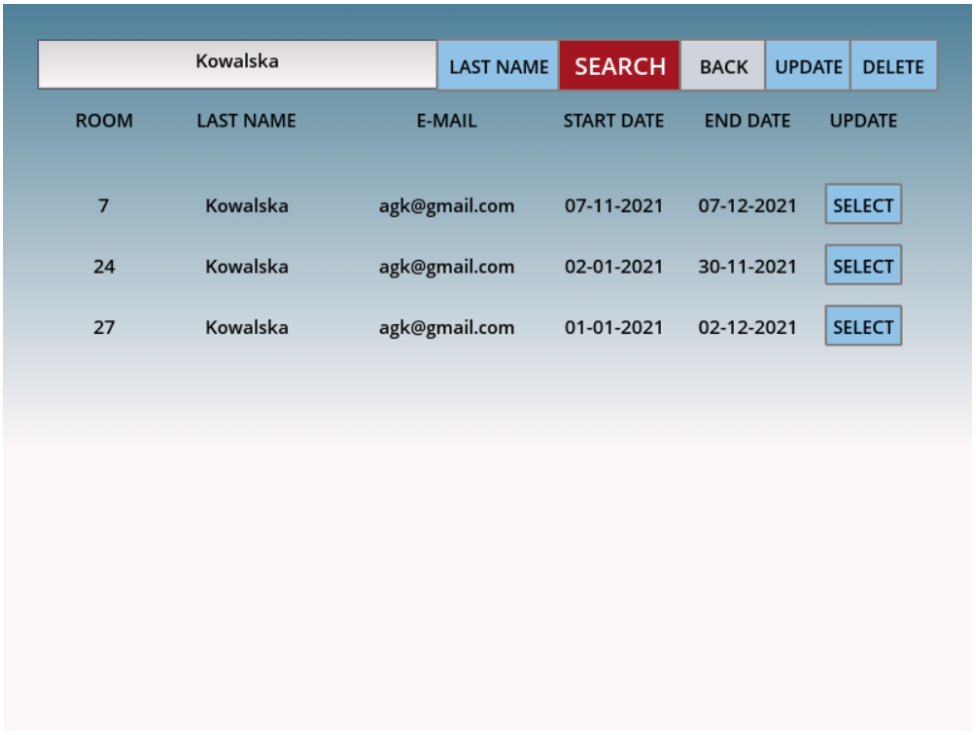
Na przedstawionym zdjęciu rys. 5.1 widnieje poprawny wynik testu w reakcji na wpisanie nieprawidłowych danych podczas logowania do systemu. Gdyby użytkownik zalogował się używając poprawnych danych, powinien zostać przekierowany do strony odpowiadającej jego uprawnieniom. Wpisując dane odpowiadające istniejącemu użytkownikowi o statusie recepcjonisty otrzymamy poprawny wynik testu widoczny na zdjęciu rys. 5.2.



Rys. 5.2. Wynik testu po wpisaniu poprawnych danych recepcjonisty.

5.1.2. Poprawność działania przeglądarki rezerwacji

Gdy z menu dostępnego dla recepcjonisty zostanie wybrana opcja ”BROWSE BOOKED STAYS”, następuje poprawne wyświetlenie rekordów widniejących w bazie danych. Aplikacja obsługuje również wyszukiwarkę na podstawie pola tekstowego i kategorii, w której użytkownik chce wyszukiwać informacji. Zdjęcie rys. 5.3 przedstawia pozytywny wynik testu wyszukiwania wszystkich zamówień dla klienta o nazwisku "Kowalska". Dla nieprawidłowych danych system powiadamia użytkownika o braku możliwych do wyświetlenia rekordów. Kiedy użytkownik chce zaktualizować wybrany rekord, bądź usunąć go, system wykonuje poprawnie polecenie, a w przypadku wybrania takiej opcji, ale nie zaznaczenia rezerwacji, użytkownik otrzyma powiadomienie o braku wybranych rekordów.

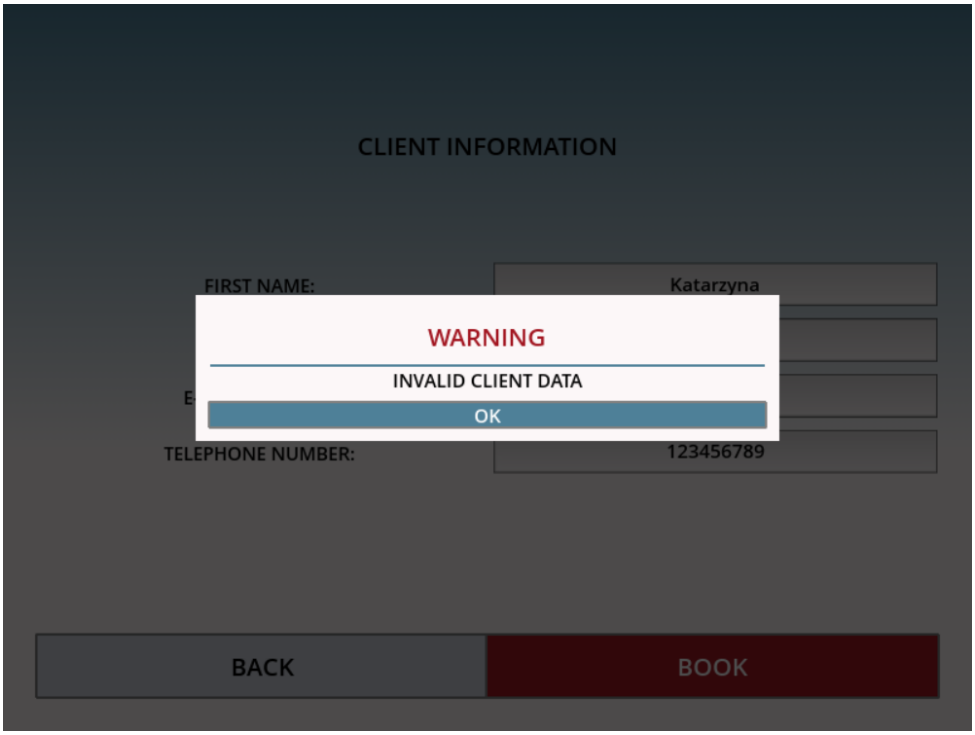


Rys. 5.3. Wynik testu wyszukiwania rezerwacji.

5.1.3. Poprawność dokonywania rezerwacji

Po wybraniu z menu recepcjonisty opcji ”BOOK CLIENT STAY” użytkownik jest przekierowywany do okna z wyborem daty, gdzie jest wiele rodzajów walidacji wpisywanych danych. Użytkownik wybierając najpierw dzień miesiąca o numerze "30", a następnie wybierając miesiąc luty powoduje, że aplikacja prawidłowo reaguje i zmienia ustawiony dzień na ostatni dzień lutego biorąc pod uwagę przestępność lat. Wybór pokojów działa prawidłowo, jeśli użytkownik nie wybierze żadnego pokoju, a będzie chciał przejść dalej otrzyma stosowny komunikat. Wyświetlanie, które pokoje są zajęte również działa zgodnie z danymi, które są w bazie danych. Ostatnim etapem rejestracji jest zapisanie danych klienta. Ten test również przebiegł pomyślnie, co można zauważyć na ilustracji rys. 5.4, gdzie nieprawidłowy format e-mail spowodował blokadę zapisania i wyświetlenie komunikatu o błędnych danych.

Z powodu rozbudowania aplikacji, nie przedstawiono wszystkich testów modułowych przeprowadzonych w celu sprawdzenia stanu aplikacji. Podczas pierwotnych testów wykryto kilka błędów, które zostały usunięte i aplikacja obecnie przechodzi testy z wynikami pozytywnymi.



Rys. 5.4. Wynik testu walidacji danych klienta.

5.2. Testy systemowe

5.2.1. Testy funkcjonalne

Poniżej przedstawiono wyniki testów opartych o wymagania funkcjonalne:

– aplikacja obsługuje logowanie, które uwzględnia uprawnienia logującego się użytkownika przekierowując go na odpowiednią stronę po prawidłowym zalogowaniu;

– system posiada zaawansowany system sprawdzania poprawności wpisanych danych takich jak adres e-mail, numer telefonu, format daty, czy też liczbę posiłków do zamówienia na podstawie możliwej maksymalnej ilości gości w pokojach;

– rezerwacja pobytu klienta przeprowadzana jest w krokach: wyboru daty, wyboru pokoi, zapisania danych osobistych klienta oraz opcjonalnego wyboru diet gości hotelowych;

– liczbę pokoi, które można zarezerwować jest ograniczana jedynie przez dostępność w wybranym okresie czasowym;

– aplikacja pozwala na przeglądanie rezerwacji wraz z wbudowaną wyszukiwarką, która wyszukuje po wpisanej wartości w pole tekstowe oraz przypisanej do niego kategorii;

– aplikacja umożliwia edycję rezerwacji pokoju jak i diet;

- aplikacja umożliwia usuwanie rezerwacji;
- aplikacja umożliwia dokonanie wyboru diety dla klienta po wcześniejszym zarezerwowaniu;
- aplikacja umożliwia prowadzenie rejestru obecności klientów;
- aplikacja umożliwia sprawdzenie wymaganej do wykonania ilości posiłków w dniu obecnym i następnym;
- aplikacja umożliwia odliczanie wykonanych już posiłków na dzień obecny jak i następny;
- aplikacja umożliwia sprawdzenie wszystkich wymaganych posiłków na obecny jak i przyszły tydzień;
- aplikacja umożliwia podgląd obecności klientów w pokojach oraz zapisywania, który pokój był już odwiedzony przez zespół sprzątający;
- aplikacja umożliwia stworzenie nowych użytkowników systemu oraz nadania im stosownych uprawnień.

Wszystkie wymagania funkcjonalne opisane w rozdziale 3.1.1 zostały spełnione, system przeszedł pozytywnie test zamieszczony w tej sekcji.

5.2.2. Testy niefunkcjonalne

Poniżej przedstawiono wyniki testów opartych na wymaganiach niefunkcjonalnych:

Bezpieczeństwo: Aplikacja jest bezpieczna dzięki swojej lokalności oraz systemowi logowania, który ogranicza dostępne użytkownikowi informacje w stosunku do ich uprawnień;

Kompatybilność: Aplikacja działać na wszystkich urządzeniach z systemem Windows oraz zainstalowanym: językiem programowania Python 3.7 lub wyższym, biblioteką Kivy 2.0 oraz biblioteką Psycopg2 po uruchomieniu serwera z bazą danych;

Ergonomiczność: Aplikacja jest oceniana przez użytkowników jako prosta w obsłudze;

Szybkość pracy: Aplikacja na większości urządzeń uruchamia się w około 4 sekundy, a najdłuższe operacje trwają w niej około sekundy.

Wszystkie wymagania niefunkcjonalne przedstawione w rozdziale 3.1.2 zostały spełnione, a aplikacja przeszła pozytywnie test tej sekcji.

6. Podsumowanie

6.1. Wnioski

Celem postawionym przed przedstawionym projektem inżynierskim była implementacja systemu, który ułatwiałby obsługę hotelu oraz byłby intuicyjny i prosty w obsłudze. Podczas pracy nad projektem dokonano przeglądu rozwiązań dostępnych na rynku, przedstawiono architekturę w jakiej stworzono system, na jej jego postawie projekt systemu z dokładnymi wymaganiami i diagramami opisującymi poszczególne części systemu, by ostatecznie stworzyć oprogramowanie, które byłoby zgodne z postawionymi celami. Wszystkie z nich udało się spełnić, dzięki czemu można uznać zakończenie pracy nad projektem jako sukces.

Motywacją do pracy była chęć stworzenia systemu od zera, z warstwą graficzną, jak i skomplikowanymi elementami logicznymi, który nie byłby aplikacją opartą na schematach szeroko dostępnych. Miał to być system spersonalizowany pod konkretny obiekt i jego wymagania, dający możliwość zastosowania niestandardowych rozwiązań. Przeglądając strony internetowe oferujące noclegi w hotelach zrodziło się pytanie: Jak to może działać od drugiej strony? Spowodowało to wybór hotelu jako obiekt do projektu, stwarzał on wiele możliwości do implementacji funkcji.

Tworzenie systemu rozpoczęto od sprawdzenia jakich narzędzi i framework-ów należy użyć podczas budowania aplikacji desktopowej. Ważnym atrybutem szukanych narzędzi była możliwość wykreowania niestandardowych rozwiązań, które wymagane były podczas formowania systemu zarządzania hotelem. Język programowania Python szybko stał się docelowym wyborem, natomiast narzędziem, które miało pomóc w tworzeniu graficznego interfejsu użytkownika została wybrana biblioteka Kivy, która w ostatnich latach zyskała na popularności wśród programistów dzięki wszechstronności, którą oferuje. Wybory te okazały się trafne. Praca nad aplikacją przebiegała sprawnie i bez komplikacji.

Projekt został ukończony i spełnia wszelkie postawione przed nim wymagania.

6.2. Perspektywy rozwoju aplikacji

Nad przedstawionym systemem może być przeprowadzonych jeszcze wiele prac rozwojowych. Przykładowymi możliwościami rozwoju systemu mogłyby być:

- integracja z zewnętrznymi systemami i platformami rezerwacji hoteli;
- dodanie modułu rezerwacji miejsc parkingowych;
- umożliwienie dokonywania księgowości w systemie;
- połączenie z punktami sprzedaży wewnątrz hotelu takimi jak restauracje, kioski w celu obciążania klienta jednym rachunkiem;
- integracja z systemem zamków elektromagnetycznych wewnątrz hotelu;
- kliencka wersja aplikacji umożliwiająca zamówienie serwisu pokojowego i innych usług.

Bibliografia

[1] IBM Cloud Education. „Czym jest architektura trójwarstwowa”. W: (2020). URL: *https://www.ibm.com/pl-pl/cloud/learn/three-tier-architecture*.

[2] Oprogramowanie SOHiS. W: (2017). URL: *http://sohis.pl/oprogramowanie/oprogramowanie-dla-hoteli*.

[3] Oprogramowanie Hotres. W: (2017). URL: *https://hotres.pl*.

[4] Oprogramowanie X2 Hotel. W: (2017). URL: *https://www.escsa.pl/x2hotel*.

[5] Oficjalna strona języka programowania Python. W: (2021). URL: *https://www.python.org/*.

[6] Strona IDE PyCharm od JetBrains. W: (2021). URL: *https://www.jetbrains.com/pycharm/*.

Załączniki

1. Repozytorium, które znajdujące się pod adresem *https://github.com/jedrzejKosior/PracaInzynierksa*, zawierające:
   (a) Pliki stworzonego systemu.
   (b) Nieniejszą pracę w formacie PDF.