Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Projekt zespołowy na przedmiot
Bazy danych 2
semestr 23L grupa
4

Prowadzący projekt:
Dr inż. Grzegorz Protaziuk
Grupa
Autorzy:
Piotr Grabowski 304262
Sofiya Makarenka 308912
Mariusz Pakulski 318704
Dawid Ruciński 304212

Spis treści

Cel projektu

Głównym założeniem projektu jest przygotowanie aplikacji opartej o bazę danych Oracle służącej do kontrolowania pracy warsztatu samochodowego, a także do rejestrowania obsługi klientów.

Wymagania

Funkcjonalne

Aplikacja umożliwia trzy poziomy logowania: logowanie jako mechanik, logowanie jako klient i logowanie jako Administrator.

Po zalogowaniu do panelu mechanika, aplikacja umożliwia następujące funkcje:

- wyświetlenie harmonogramu pracy od razu po zalogowaniu w postaci zadań oraz planowanych dla nich godzin pracy określonych wcześniej przez mechanika,
- wciśnięcie przycisku, który pokaże dostępne zlecenia, nienależące do żadnego mechanika,
- wyświetlenie obecnie dostępnych części w warsztacie wraz z okresowym sprawdzaniem stanu magazynu,
- złożenie zamówienia na części (do listy dostępnej dla admina)
- możliwość akceptacji lub odrzucenia zlecenia, po czym ustawienie rozpoczęcia i zakończenia pracy w harmonogramie,
- możliwość zmiany stanu realizacji zadania na wykonane, anulowane lub w trakcie,
- oznaczenie, że zadanie zostało wykonane oraz usunięcie go z harmonogramu,
- możliwość przejrzenia historii zleceń zawierającej dane o naprawie i kliencie,
- rozpatrzenie reklamacji,
- możliwość przejrzenia zgłoszonych reklamacji,
- możliwość akceptacji lub odrzucenia reklamacji,

Niezalogowany użytkownik ma możliwość rejestracji jako klient.
Proces rejestracji wygląda w następujący sposób:
Po odwiedzeniu strony, użytkownik wybiera odnośnik do panelu rejestracji. Znajduje się tam formularz z polami, które są wymagane do zarejestrowania się. Są to: imię, nazwisko, adres e-mail wraz z telefonem do kontaktu, a także hasło niezbędne do logowania się. W ten sposób dane użytkownika trafiają do bazy danych, a sam użytkownik ma możliwość logowania się.

Po zalogowaniu do panelu klienta, aplikacja umożliwia następujące funkcje:

- złożenie nowego zlecenia - wówczas użytkownik podaje dane samochodu, którego dotyczy naprawa lub wybiera z listy dotychczas posiadanych pojazdów. Podczas wypełniania formularza z danymi posiadanego samochodu będzie możliwość zaznaczenia checkboxa, który spowoduje, że samochód, który jest zgłaszany, zostanie dodany do listy samochodów, które posiada użytkownik na później.

- wyświetlenie stanu realizacji zlecenia oraz szacowany czas, kiedy można odebrać samochód,
- możliwość zgłoszenia reklamacji - po zatwierdzeniu przez mechanika wykonania usługi w panelu klienta pojawia się przycisk przekierowujący do panelu reklamacji tejże usługi, Pierwszy administrator jest ustawiany przy instalacji aplikacji. Później kolejnych adminów mogą dodawać tylko inni admini.

Po zalogowaniu jako Administrator aplikacja umożliwia

- możliwość stworzenia konta dla nowego mechanika oraz usunięcia konta mechanika
- generowanie raportu dotyczącego zrealizowanych zleceń w wybranym czasie • wyświetlanie i zarządzanie listą planowanych zamówień oraz usuwanie zrealizowanych
- możliwość dodania nowego zlecenia.

Niefunkcjonalne

- bezpieczeństwo - system chroniony przed nieautoryzowanym dostępem, hasła będą przechowywane w postaci hashu, by chronić bazę przed wyciekiem,
- baza dostępna przez 95% czasu działania warsztatu, ewentualne przerwy działania w nocy lub poza godzinami działania nie mają dużego wpływu na funkcjonowanie zakładu, przerwy konserwacyjne planowane w nocy i w weekendy,
- responsywność - system reaguje na zapytania w dostatecznie krótkim czasie, przechodzenie między widokami powinno trwać mniej niż sekundę, zaś wygenerowanie raportu - mniej niż minutę
- wielodostępność - aplikacja umożliwia 10 użytkownikom jednocześnie korzystanie z aplikacji przy responsywności nieprzekraczającej jednej sekundy
- kompatybilność z systemami operacyjnymi Microsoft, Linux - aplikacja będzie dostępna przez sieć i będzie można jej używać przez przeglądarkę

Stworzenie modelu pojęciowego i logicznego dla bazy relacyjnej

Opis warsztatu

Klienci, aby naprawić pojazd w naszej firmie, muszą najpierw założyć konto w naszej aplikacji. Następnie muszą zadzwonić do warsztatu. Odbiera administrator, który ustala, kiedy klient może przyjechać wraz ze swoim pojazdem do naprawy. Wtedy też rezerwuje czas mechanikowi na spotkanie z klientem. Potem klient przyjeżdża do warsztatu wraz ze swoim pojazdem, dany mechanik ustala opis przed naprawą oraz ustala szacowany czas naprawy. Wtedy tworzy zlecenie, w którym opisuje wszystkie szczegóły naprawy. Potem administrator przydziela terminy pracy (na stanowiskach i przy maszynach) do mechaników. Wtedy mechanicy otrzymają powiadomienie o nowym zleceniu do wykonania w swoim panelu. Po wykonaniu naprawy zostanie wysłany komunikat klientowi, że może już odebrać pojazd z

warsztatu. Klient otrzymuje na koniec kwotę do zapłaty i może złożyć reklamację w przypadku, gdy nie jest on zadowolony z wyniku naprawy.

Dostawy kolejnych części realizowane są poprzez system zamówień. W razie stwierdzenia zapotrzebowania na jakąś część, mechanik zakłada takie zamówienie. Z kolei, gdy części fizycznie dotrą do warsztatu, jeden z mechaników potwierdza odbiór takiego zamówienia, zaś stan części się zwiększa. Można wówczas wykorzystać je do naprawy.
Mechanicy będą pracować w stałych, określonych godzinach na etatach. Będą otrzymywali miesięczne wynagrodzenie.

Opis encji

Zlecenie – reprezentuje tworzenie zlecenia dokonanej przez mechanika
Wyjaśnienia do atrybutów:

- Id użytkownika – Klucz główny encji. Mechanik, który bada pojazd, ustala co należy zrobić podczas naprawy i tworzy zlecenie.

- Opis przed naprawą – Jest to opis na podstawie rozmowy z klientem i zaobserwowania stanu maszyny do naprawy. Zawiera opis co należy wymienić lub naprawić, wymagania klienta, istotne informacje na temat pojazdu.

- Data przyjęcia – data przyjęcia pojazdu do naprawy

- Szacowany czas naprawy – szacowany czas naprawy w minutach.

ZrealNapraw – reprezentuje stan po zakończonej naprawie (po dodaniu ceny naprawy, daty realizacji i opcjonalnym opisie)

- Opis po naprawie – Jest to opis, który zawiera rezultat wykonanej naprawy, uwagi, notatki, adnotacje, komentarze. Jeżeli wszystko przebiegło zgodnie z planem i nie ma żadnych uwag to można to pole pozostawić puste.
- Cena naprawy – cena do zapłacenia przez klienta

- Data realizacji – data zakończenia wykonywania naprawy.

Termin – reprezentuje wpis w harmonogramie mechanika. Posiada atrybut typ, który oznacza, że terminem może być TerminStanowisko, TerminMaszyna albo TerminKlient.
Reprezentuje przydział mechanika do pracy w danym przedziale czasu. Mechanik może pracować na stanowisku samochodowym lub przy maszynie. (Przy danym stanowisku lub maszynie nie mogą wykonywać się 2 naprawy jednocześnie). Może również zostać przypisany do spotkania z klientem przed utworzeniem zlecenia.
Atrybut Id użytkownika wywodzący się ze związku z mechanikiem to id przydzielonego do naprawy mechanika. (Do jednej naprawy może być przydzielonych wiele mechaników)

TerminStanowisko – reprezentuje pracę mechanika na stanowisku.

TerminMaszyna – reprezentuje pracę mechanika przy maszynie.

TerminKlient – reprezentuje termin spotkania z klientem.

Stanowisko – reprezentuje miejsce wykonywania naprawy.

Maszyna – reprezentuje maszynę, na której jest wykonywana naprawa.

Użytkownik posiada atrybut skrót hasła. Jest to wynik działania funkcji skrótu dla hasła (np.MD5). Stanowi to alternatywę do trzymania hasła jako atrybut encji i stanowi zabezpieczenie przed nieupoważnionym dostępem do bazy danych.
Jest to powszechnie przyjęta praktyka bezpieczeństwa odnośnie sposobu postępowania z danymi - należy zapisywać jedynie skrót (hash) hasła, a nie oryginalne hasło.
Posiada również atrybut typ, który oznacza, że użytkownikiem może być mechanik, klient albo administrator.

Mechanik - reprezentuje mechanika pracującego w warsztacie. Posiada on atrybut pensja, która oznacza kwotę brutto miesięczną.

Klient - reprezentuje klienta warsztatu.

Administrator – reprezentuje administratora warsztatu.

Adres – reprezentuje adres, który może być adresem magazynu lub adresem użytkownika.

Pojazd – reprezentuje pojazd, który jest przedmiotem zlecenia.

Status – reprezentuje możliwy status naprawy. Dzięki temu przy każdej naprawie będzie możliwość wyboru jednego statusu naprawy z listy dostępnych.

Możliwe statusy zlecenia:

- Utworzone,
- Przydzielone do mechaników,
- W trakcie,
- Wykonane,
- Odebranie pojazdu przez klienta

Premia – reprezentuje premię dawaną mechanikom.

Reklamacja – Reprezentuje reklamację wysyłana przez klienta do warsztatu.

Rozwiązanie – Reprezentuje rozwiązanie, które klient uważa za stosowne w ramach reklamacji.

Magazyn – Reprezentuje miejsce, w którym będą składowane części przeznaczone do napraw.

Encje takie jak

- UżycieCzesci
- PozMag
- PozZam

Powstały wskutek dekompozycji związków wieloznacznych. Są to encje łączące.

UżycieCzesci reprezentuje wykorzystanie pewnej liczby cześci danego rodzaju w konkretnej naprawie.

Rodzaj części – reprezentuje część samochodową, montowaną podczas naprawy.

PozMag - reprezentuje przechowywanie pewnej ilości części danego rodzaju w konkretnym magazynie.

PozZam - reprezentuje znajdowanie się pewnej ilości części danego rodzaju w konkretnym zamówieniu. Encja ta posiada ilość sztuk, która określa ile sztuk danej części znajduje się w zamówieniu.

Reklamacja – reprezentuje reklamację zgłoszoną przez klienta do naprawy. Zawiera atrybut opis, w którym klient umieszcza uwagi na temat naprawy.

Zamówienie – reprezentuje zamówienie części do magazynu.

Rozwiązanie – Rodzaj rozwiązania jaki proponuje klient w reklamacji

Opis związków

Adres - Uzytkownik:

Związek adresu i użytkownika to związek jednoznaczny obligatoryjny-obligatoryjny. Użytkownik musi mieszkać pod jednym adresem.
Ten sam adres może opisywać lokalizację wielu użytkowników (Może się zdarzyć sytuacja, że pod jednym adresem dwójka klientów założy konto w warsztacie i każdy z nich będzie miał swój pojazd do naprawy). Każdy użytkownik musi mieć dokładnie jeden adres.

Adres – Magazyn

Związek adresu i magazynu to związek jednojednoznaczny obligatoryjny-obligatoryjny.

Ten sam adres musi opisywać lokalizację albo dokładnie jednego magazynu albo jednego lub więcej użytkowników. Każdy magazyn musi znajdować się pod dokładnie jednym adresem.

Mechanik (podtyp encji Użytkownik) – Premia:

Związek mechanika i premii to związek jednoznaczny opcjonalny-obligatoryjny. Mechanik może otrzymać wiele premii ale również może nie otrzymać żadnej. Każda premia jest przyznawana dokładnie jednemu mechanikowi.

Mechanik (podtyp encji Użytkownik) – Zlecenie:

Związek mechanika i zleconej naprawy to związek jednoznaczny opcjonalny-obligatoryjny. Mechanik może stworzyć wiele zleconych napraw. Może również nie stworzyć żadnej naprawy, bo mógł na przykład dopiero co zostać przyjęty. Każda zlecona naprawa musi być utworzona przez dokładnie jednego mechanika.

Mechanik (podtyp encji Użytkownik) – Termin:

Związek mechanika i terminu to związek jednoznaczny opcjonalny-obligatoryjny. Mechanik może posiadać wiele terminów, ale również może nie posiadać żadnego. Każdy termin musi być przydzielony do dokładnie jednego mechanika.

Klient (podtyp encji Użytkownik) – TerminKlient:

Związek klienta i encji TerminKlient to związek jednoznaczny opcjonalny-obligatoryjny. Klient może mieć wiele terminów spotkań z mechanikami (np. kiedy zleca już którąś naprawę). Klient może również nie mieć zaplanowanego żadnego spotkania z mechanikiem (np. kiedy dopiero co się zarejestrował w aplikacji). Spotkanie z mechanikiem musi być przydzielone do dokładnie jednego klienta.

Klient (podtyp encji Użytkownik) – Pojazd:

Związek klienta i pojazdu to związek jednoznaczny opcjonalny-obligatoryjny. Klient może posiadać wiele pojazdów, ale nie musi również posiadać żadnego (może założyć konto w aplikacji bez podawania pojazdu do przyszłych napraw). Pojazd musi należeć do dokładnie jednego klienta.

Stanowisko – TerminStanowisko (podtyp encji Termin):

Związek Stanowiska i encji TerminStanowisko to związek jednoznaczny opcjonalny-obligatoryjny.

Stanowisko może mieć wiele terminów pracy na nim, ale nie musi mieć żadnego takiego terminu. Termin pracy na stanowisku jest przydzielony do dokładnie jednego stanowiska.

Maszyna – TerminMaszyna (podtyp encji Termin):

Związek maszyny i encji TerminMaszyna to związek jednoznaczny opcjonalny-obligatoryjny. Maszyna może być używana w wielu pracach przy niej. Takie prace mogą odbywać się w różnych terminach i dotyczyć różnych zleceń. Maszyna może również nie mieć przypisanego terminu pracy na niej (mogła dopiero co zostać kupiona). Termin pracy na maszynie jest przydzielony do dokładnie jednej maszyny.

TerminStanowisko – Zlecenie

Związek encji TerminStanowisko i zlecenia to związek jednoznaczny opcjonalny-obligatoryjny. Zlecenie może obejmować wiele różnych terminów prac na stanowiskach. Nie musi obejmować również żadnego, ponieważ może obejmować np. tylko termin pracy na maszynie. Każdy termin pracy na stanowisku jest przydzielony do dokładnie jednego zlecenia.

TerminMaszyna – Zlecenie

Związek encji TerminMaszyna i zlecenia to związek jednoznaczny opcjonalny-obligatoryjny.
Zlecenie może obejmować wiele różnych terminów prac na maszynach. Nie musi obejmować również żadnego, ponieważ może obejmować np. tylko termin pracy na stanowisku. Każdy termin praca na maszynie jest przydzielony do dokładnie jednego zlecenia.

Pojazd – Zlecenie:

Związek pojazdu i zlecenia to związek jednoznaczny opcjonalny-obligatoryjny. Zlecenie zajmuje się jednym pojazdem. Żeby zlecona naprawa mogła powstać musi zawierać pojazd, który będzie poddawany naprawie. Ten sam pojazd może być poddawany naprawie wielokrotnie. Klient mógł również założyć konto i przed pierwszym zleceniem mógł dodać pojazd do puli możliwych do wybrania do naprawy pojazdów. Wtedy pojazd nie będzie znajdował się w żadnej zleconej naprawie.

Status – Zlecenie

Związek pojazdu i zlecenia to związek jednoznaczny opcjonalny-obligatoryjny.
Status może określać stan wielu zleceń. Może też nie określać stanu żadnego zlecenia (akurat może nie być w danym momencie żadnego zlecenia o danym statusie). Zlecenie musi mieć dokładnie jeden status.

Zlecenie – ZrealNapr

Związek encji zlecenia i zrealizowanej naprawy to związek jednojednoznaczny opcjonalny-obligatoryjny. Zlecenie jest zrealizowane dzięki zrealizowanej naprawie. Zlecenie nie musi mieć zrealizowanej naprawy, ponieważ naprawa mogła się jeszcze w tym momencie nie zakończyć. Każda zrealizowana naprawa musi realizować dokładnie jedno zlecenie.

ZrealNapr - Reklamacja

Związek encji ZrealNapr i reklamacji to związek jednojednoznaczny opcjonalny-obligatoryjny. Zrealizowana naprawa może mieć zgłoszoną reklamację, ale nie musi. Reklamacja, jeśli już istnieje, musi być zgłoszona do dokładnie jednej zrealizowanej naprawy.

Rozwiazanie – Reklamacja

Związek rozwiązania i reklamacji to związek jednoznaczny opcjonalny-obligatoryjny.
To samo rozwiązanie może być proponowane w wielu reklamacjach. Rozwiązanie nie musi mieć reklamacji, w której jest ono proponowane (może jeszcze nie zostać utworzona żadna reklamacja, a rozwiązania mogą być wbudowane domyślnie). Każda reklamacja musi mieć dokładnie jedno rozwiązanie.

Magazyn – Zamówienie

Związek magazynu i zamówienia to związek jednoznaczny opcjonalny-obligatoryjny.
Magazyn może być obiektem docelowym wielu zamówień. Również nie musi być obiektem docelowym żadnego zamówienia (przy jego powstaniu nie ma skierowanych do niego żadnych zamówień) Zamówienie musi być skierowane do dokładnie jednego magazynu.

Powstały również związki poprzez dekompozycję związków wieloznacznych. Ze danych związków powstały:

- Ze związku zlecenia i rodzaju części powstały dwa związki jednoznaczne opcjonalneobligatoryjne: zlecenie i użycie części oraz rodzaj części i użycie części.
- Ze związku magazynu i rodzaju części powstały dwa związki jednoznaczne opcjonalneobligatoryjne: Magazyn i PozMag oraz rodzaj części i PozMag.
- Ze związku zamówienia i rodzaju części powstały dwa związki jednoznaczne opcjonalneobligatoryjne: Zamówienie i PozZam oraz Rodzaj części i PozZam.

Zlecenie – Rodzaj części

Dawny związek zlecenia i rodzaju części to związek wieloznaczny opcjonalny-opcjonalny.
Zlecenie może wykorzystywać wiele rodzajów części. Może też nie wykorzystywać żadnego rodzaju części (np. naprawa może dotyczyć tylko dokręcenia poluzowanych części). Ten sam rodzaj części może być wykorzystany wielokrotnie przez wiele zleceń. (Kilka opon może być wymienionych w jednym pojeździe w jednym zleceniu, a kilka innych opon może być

wymienionych w innym pojeździe w innym zleceniu. Rodzajem części w tym przypadku jest opona i pewna ich liczba jest wykorzystywana w różnych zleceniach) Może być również nie wykorzystany w ogóle przez żadne zlecenie dotychczas.

Zlecenie – Użycie części

Związek zlecenia i użycia części to związek jednoznaczny opcjonalny-obligatoryjny.
Używanie części w zleceniu może występować wielokrotnie. Może także nie występować w ogóle. Jeśli już używana jest część to musi ona dotyczyć dokładnie jednego zlecenia, które jej używa.

Rodzaj części – Użycie części
Związek rodzaju części i użycia części to związek jednoznaczny opcjonalny-obligatoryjny.
Każdy rodzaj części może mieć wiele użyć w jednym zleceniu (w jednym zleceniu może być używane kilka opon, ale również kilka śrubek). Pewien rodzaj części może w ogóle nie być używany w żadnym zleceniu. Jeśli już jest używana część w zleceniu to musi być ta część dokładnie jednego rodzaju części wchodzących w skład wszystkich rodzajów dostępnych części.

Magazyn – rodzajCzesci

Dawny związek magazynu i rodzaju części to związek wieloznaczny opcjonalny-opcjonalny.
Magazyn może zawierać wiele rodzajów części. Może również być pusty. Dany rodzaj części może znajdować się w wielu magazynach, może również nie znajdować się w żadnym.

Magazyn – PozMag
Związek magazynu i pozycji w magazynie to związek jednoznaczny opcjonalny-obligatoryjny.
W jednym magazynie może znajdować się wiele rodzajów części, ale może też nie znajdować się tam żadna część. Każda pozycja w magazynie musi być umieszczona w dokładnie jednym magazynie, w którym on się znajduje.

Rodzaj części – PozMag

Związek rodzaju części i pozycji w magazynie to związek jednoznaczny opcjonalny-obligatoryjny. Każdy rodzaj części może znajdować się w wielu pozycjach w magazynie. Może też nie znajdować się w żadnym magazynie. Każda pozycja w magazynie musi obejmować dokładnie jeden rodzaj części.

Zamówienie – RodzajCzesci

Dawny związek zamówienia i rodzaju części to związek wieloznaczny obligatoryjny-opcjonalny.
Każde zamówienie może zawierać wiele rodzajów części. Musi dotyczyć chociaż jednego.

Każdy rodzaj części może być zawarty w wielu zamówieniach (wtedy np. kilka opon jest zawartych w jednym zamówieniu, a kilka innych opon w drugim). Nie musi być zawarty w żadnym zamówieniu.

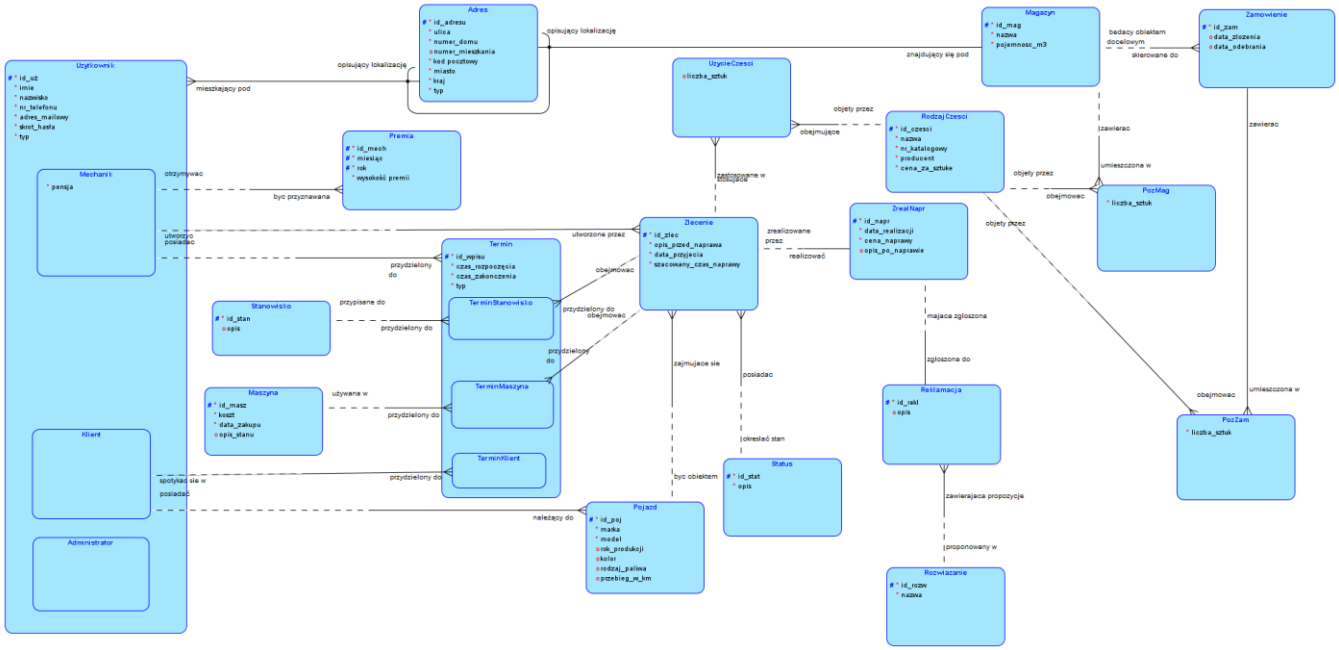
Zamówienie – PozZam

Związek zamówienia i pozycji w zamówieniu to związek jednoznaczny obligatoryjny-obligatoryjny.
W jednym zamówieniu może być wiele pozycji. Zamówienie musi zawierać co najmniej jedną pozycję. Każda pozycja w zamówieniu musi być umieszczona w dokładnie jednym zamówieniu.

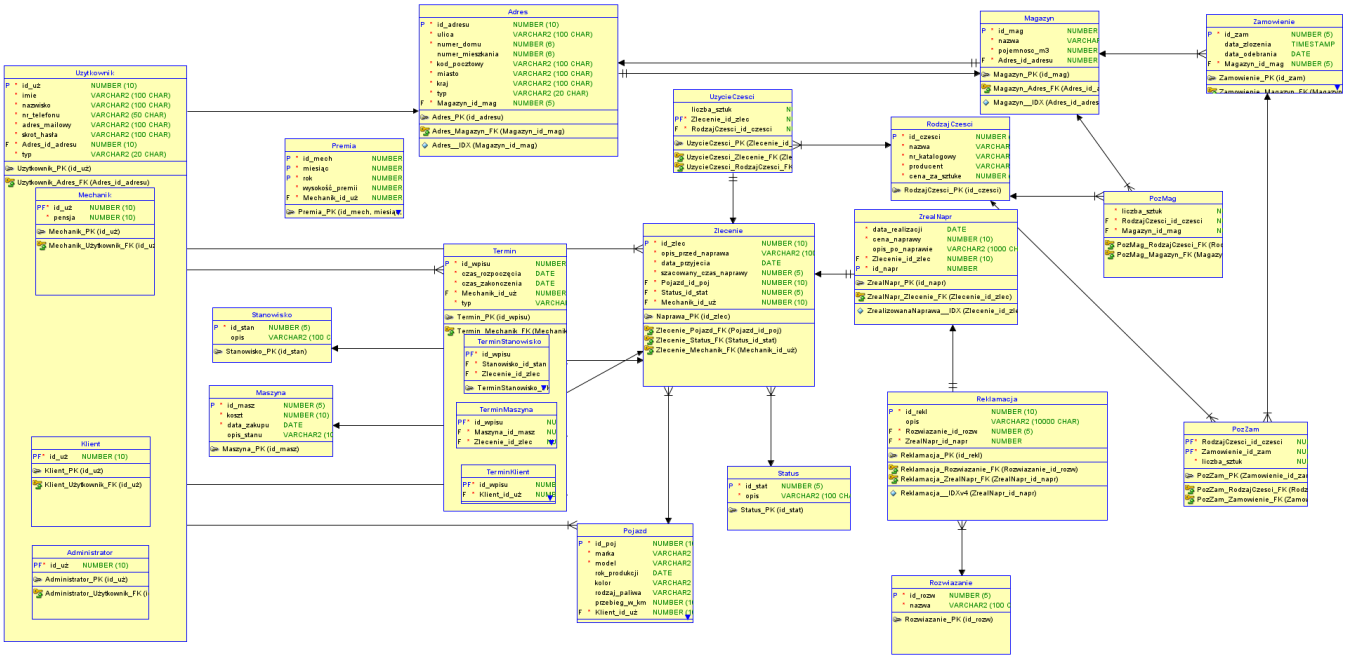
RodzajCzesci – PozZam

Związek rodzaju części i pozycji w zamówieniu to związek jednoznaczny opcjonalny-obligatoryjny.
Wiele pozycji w zamówieniu może zawierać ten sam rodzaj części. Rodzaj części nie musi być objęty przez żadną pozycję w zamówieniu. Każda pozycja w zamówieniu musi obejmować dokładnie jeden rodzaj części.

Stworzony model pojęciowy



Zaimplementowany w narzędziu SQL Developer logiczny model bazy relacyjnej.



Projekt aplikacji

Aplikacja będzie składać się z trzech warstw: frontend, backend i warstwy bazy danych.

Warstwa prezentacji (Frontend) służy do interakcji użytkownika z aplikacją oraz prezentacji danych. Z perspektywy użytkownika aplikacja jest obsługiwana z użyciem przeglądarki internetowej. Wysyła ona żądania HTTP do postawionych endpointów, a jako odpowiedź otrzymywana jest strona wygenerowana w językach HTML i CSS a wzbogacona o dane z backendu.
Aplikacja od strony backendu będzie reprezentowana przez serwer aplikacji, który będzie obsługiwać żądania pochodzące przez warstwę frontend i odpowiadać za interakcję z bazą danych. Obsługuje on także działanie procedur składowanych w bazie danych.

Trzecia warstwa to warstwa bazy danych i jest ona odpowiedzialna za to jak przechowywane są dane aplikacji. Komunikacja z bazą danych zajmuje się backend - są to zapytania do bazy danych, takie jak odczyt, aktualizacja, usunięcie, zapis itd.. Po stronie bazy danych te zapytania są przetwarzane i na koniec wysyłana jest wymagana informacja (może to być potwierdzenie tego, że operacja skończyła się sukcesem lub żądane przez backend dane).

Docelowa web-aplikacja będzie napisana w języku Python z wykorzystaniem frameworku Flask, frontend będzie się budować za pomocą CSS i HTML, natomiast warstwa danych zrealizowana jest poprzez bazę danych SQL. Baza danych zawiera tabele przechowujące poszczególne obiekty. Zapytania do warstwy danych będą odbywać się za pomocą SQL zapytań lub natywny dla Pythona sposób (czyli wykorzystując mapowanie na obiekty Pythona za pomocą wybranej biblioteki ORM, co umożliwia uniknięcia bezpośredniego operowania na tabelach i zapytaniach SQL).

Opis rozwiązań

Do obsługi bazy danych zastosowano bibliotekę ORM SQLAlchemy. Poszczególne tabele w bazie danych są reprezentowane odpowiadającymi obiektami o nazwie takiej jak nazwa tabeli z bazie danych

Zarejestrowanie się nowego klienta powoduje utworzenie dwóch powiązanych ze sobą obiektów: Użytkownik (jako nadtyp) i Klient (jako podtyp). W ten sposób nadawany jest mu unikatowy, wyróżniający go identyfikator, a sam wpis trafia do tabeli Użytkownik. Jednocześnie, pojawia się powiązany z nim wpis do tabeli Adres.

W analogiczny sposób administrator może dodawać nowych mechaników (jeden z podtypów użytkowników).

Po rejestracji użytkownik ma możliwość zalogowania się. Po przekazaniu nazwy użytkownika i hasha hasła za pomocą zapytania wybierającego dokonywane jest sprawdzenie, czy taki użytkownik istnieje, a także czy skróty hasła się zgadzają. Wówczas, użytkownik zyskuje możliwość działania w swoim panelu.

Z kolei, użytkownik ma możliwość dodania pojazdu klienta lub modyfikacji jego danych, co również się odbywa za pomocą funkcji biblioteki

Naprawy będą zgłaszane przez administratora przy pomocy formularza i procedury zawierającej utworzenie nowego obiektu klasy Klient, a także klasy Zlecenie opartej o dane z pól tekstowych. W ten sposób powstaje wpis w tabeli Zlecenie.

Będzie można wygenerować zestawienia podsumowujące pracę warsztatu, jak na przykład średnią liczbę napraw pod względem marki samochodu lub średni czas trwania naprawy względem poszczególnych mechaników. Będzie to wykonywane za pomocą biblioteki ORM i zapytania wybierającego o predefiniowanych parametrach.

Prezentacja obiektów bazy danych w aplikacji

Na podstawie wcześniejszego opisu implementacji zdecydowano użyć biblioteki ORM. Wybraną biblioteką jest SQLAlchemy. Zapytania do bazy danych teraz będą robione w natywny sposób dla Pythona. Wymagana jest definicja następujących klas, żeby zaprezentować tabele bazy danych: Uzytkownik, Klient, Administrator, Mechanik, Adres, Magazyn, Stanowisko, Premia, Maszyna, Termin, TerminKlient, TerminMaszyna, TerminStanowisko, ZlecNapr, Pojazd, UzycieCzesci, RodzajCzesci, PozMag, Zamowienie, PozZam, ZrelNapr, Reklamacja, Rozwiazanie, Status.

Komponenty aplikacji:

- Panel logowania i rejestracji do dodawania nowych użytkowników i autoryzacji

- Widok klienta służący do pokazywania istniejących zleceń
- Widok mechanika – podobny do widoku klienta, z możliwością pobierania zadań, dodawania potrzebnych części
- Widok magazynu z listą części, możliwością zamówienia ich z zewnątrz i potwierdzenia dostawy
- Komponent obliczeniowy zliczający koszt naprawy, czas poświęcony na jej zrealizowanie
- Komponent produkujący raporty analityczne
- Panel administratora

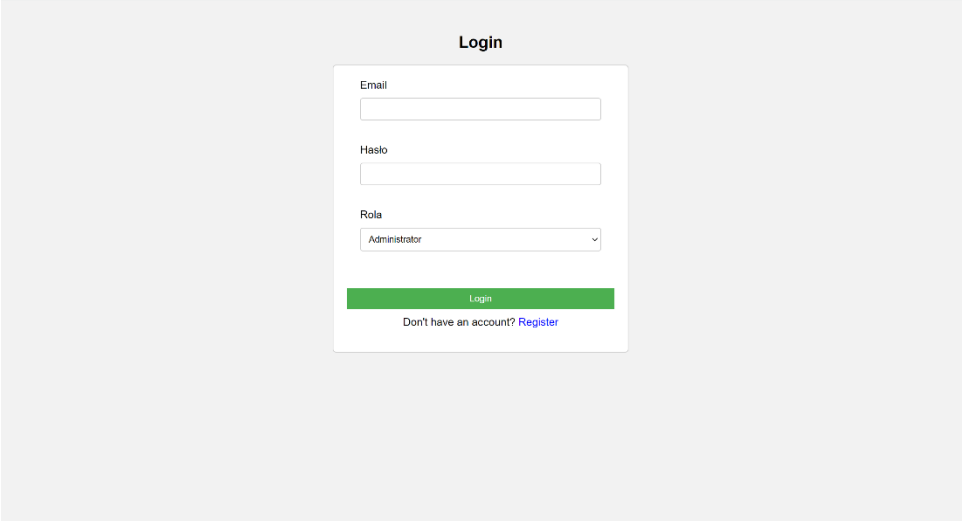
Tabele będą odpowiednio reprezentowane przez klasy w aplikacji. Odpowiednio przy wykorzystywaniu kolejnych komponentów są używane odpowiednie obiekty.
Aplikacja wykorzystuje mechanizm blueprints do dzielenia jej na komponenty. Wyróżnione są osobne blueprinty do modułu autoryzacji i logowania, paneli mechanika, administratora i użytkownika

Zrealizowane funkcjonalności systemu

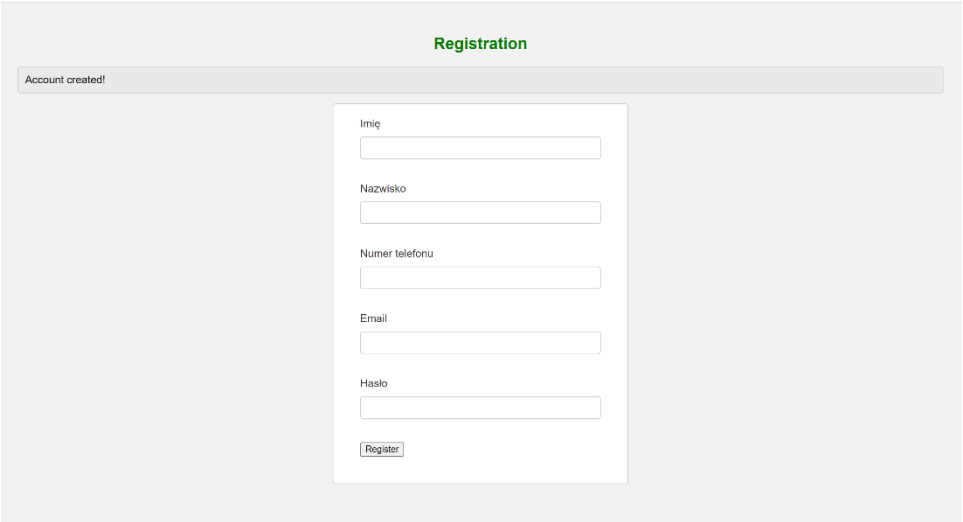
Podział na trzy rodzaje użytkowników zrealizowany jest przez trzy główne podstrony. Poszczególne wyświetlania obiektów realizowane są poprzez wysyłanie zapytań do bazy danych. Rejestrowanie realizowane jest na głównej stronie. Po wypełnieniu odpowiednich pól formularza tworzony jest obiekt użytkownika, który wstawiany jest za pomocą utworzenia odpowiednich obiektów i wykorzystania do tabeli użytkowników po sprawdzeniu czy jej unikalności.
Komponenty w bazie danych zależne są od siebie za pomocą kluczy obcych. Dodatkowo przy wykonywaniu procedur zmieniających obiekty w bazie wywoływane są odpowiednie wyzwalacze, które zachowują logikę bazy.

Aplikacja realizuje następujące funkcjonalności:

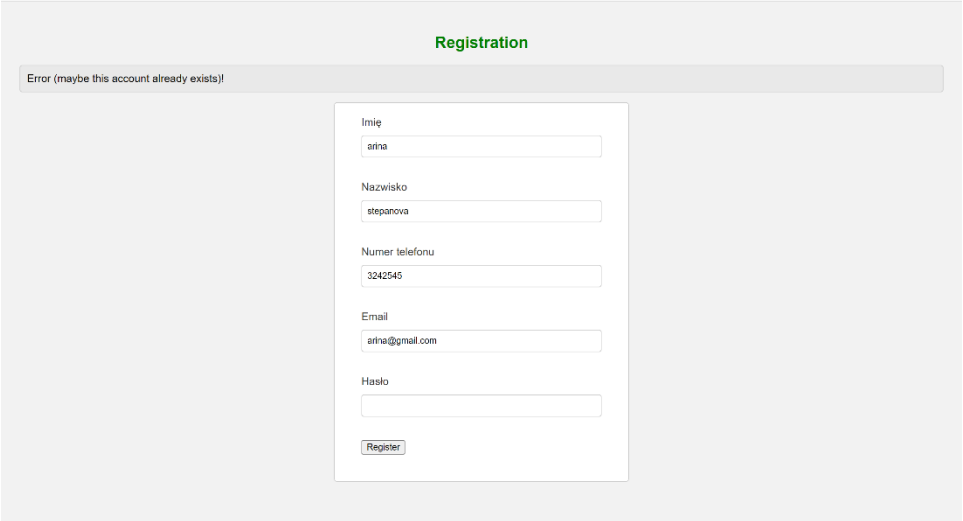
Logowanie
-rejestracja nowego użytkownika, oraz logowanie z zastosowaniem hashu dla hasła
Stworzony formularz do logowania: wygląda następująco:



Wygląd formularza do rejestracji nowej osoby:

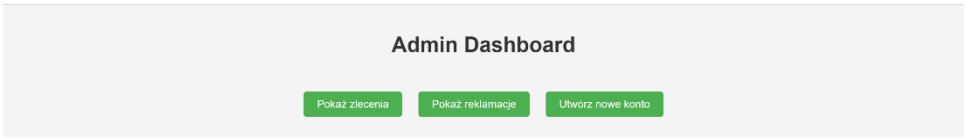


Nowa osoba zostanie dodana, jeśli nie znajduje się jeszcze w bazie:



Admin

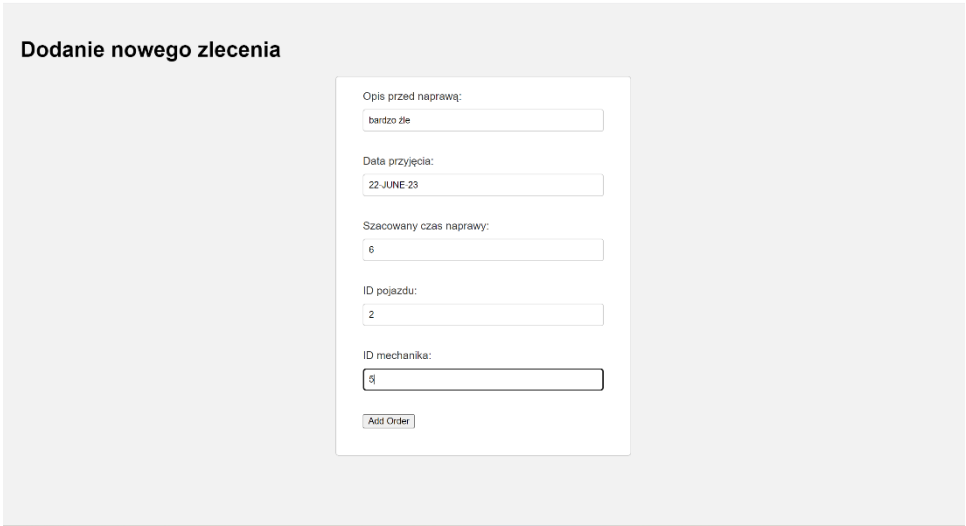
Po zalogowaniu do podstrony administaratora otrzymuje następujący wygląd:



Zrealizowane dla niego funkcjonalności to:

- Wyświetlenie wszystkich aktualnych zleceń

- Możliwość dodania nowego zlecenia zrealizowana poprzez krótki formularz:



Na zdjęciu widzimy pojawienie się nowego zlecenia

Lista Zlecen

Opis przed naprawą	Data przyjęcia	Szacowany czas naprawy	Pojazd	Status zlecenia	Mechanik
wygląda okay	2023-06-01	8	Toyota GT86 2011-01-01	backlog	Karolina Da
Opis naprawy 1	2023-05-01	240	Toyota GT86 2011-01-01	skończony	Marcin Szweda
bardzo źle	2023-06-22	6	Audi A4 2009-03-21	backlog	Marcin Szweda
Opis naprawy 2	2023-05-01	1440	Audi A4 2009-03-21	skończony	Karolina Da

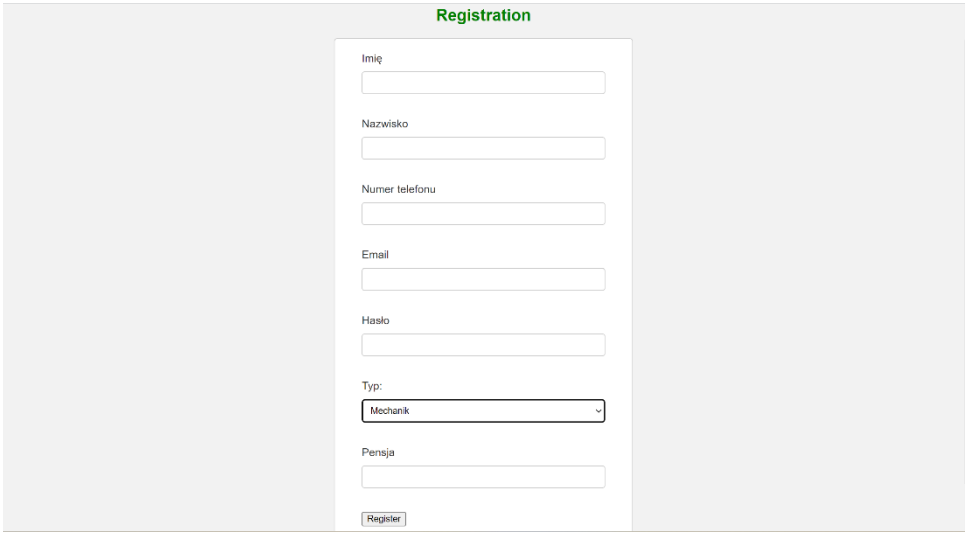
Dodaj zlecenie

- Wyświetlenie wszystkich reklamacji

Lista Reklamacji

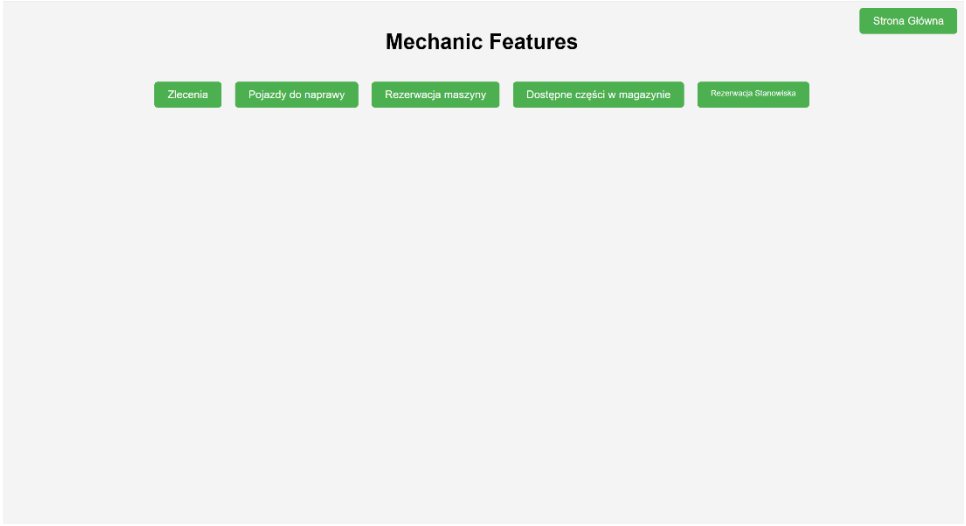
Opis reklamacji	Co należy zrobic	Data naprawy	Opis naprawy
zbyt długi czas naprawy	przeprosic	2023-06-01 00:00:00	jest okay
bumper peknal po naprawie	zwrot kwoty	2023-06-23 00:00:00	jest okay

- Administrator może również dodać nowego mechanika do bazy za pośrednictwem formularza:

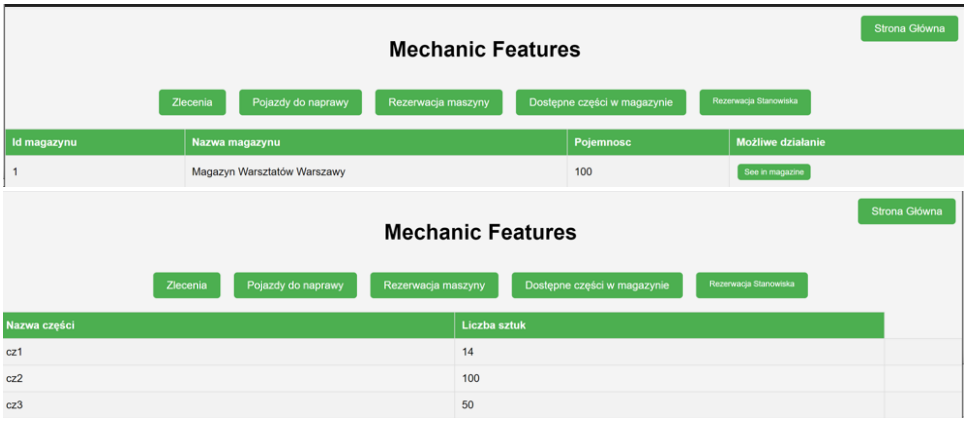


Część Mechanik

Podstrona po zalogowaniu do mechanika wygląda w następujący sposób:



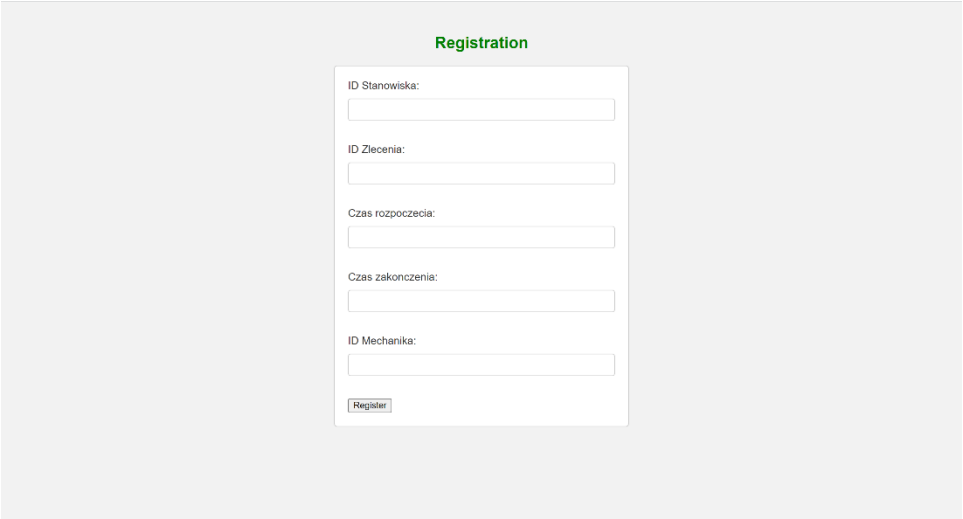
Mechanik może przeglądać wśród dostępnych zakładów wraz z informacjami o nim, a dla każdego z nich wyświetlić dostępne w nim części wraz z ich ilością.



Kolejna zrealizowaną funkcjonalnością jest możliwość wyświetlania aktualnych przypisanych do niego zleceń.

Mechanik może także dokonać zgłoszenia zapotrzebowania na konkretną maszynę (na przykład podnośnik).

W tym celu wypełnia on formularz przedstawiony na poniższym obrazie:



W ten sposób po wypełnieniu wszystkich pól (jest to walidowane) pojawia się rezerwacja na maszynę w liście rezerwacji. Mechanik może wówczas realizować pracę na maszynie.

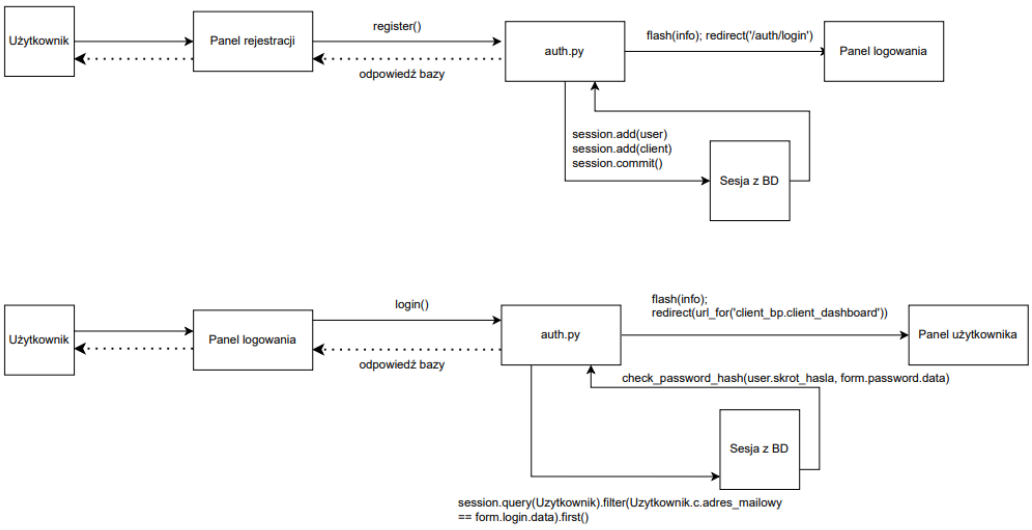
Przykładowe raporty:

Raport generuje tabelkę z opisem reklamacji i szczegóły zrealizowanej naprawy dotyczącej tej reklamacji:

Lista reklamacji

Opis reklamacji	Co nalezy zrobic	Data naprawy	Opis naprawy
zbyt dlugi czas oczekiwania	przeprosic	2023-05-05 00:00:00	ojej co sie stalo
bumper peknał po naprawie	przeprosic	2023-06-01 00:00:00	ojej co sie stalo

Schemat sekwencyjny rejestracji oraz logowania:



Technikalia realizacji

W naszym projekcie stworzyliśmy klika triggerów.

1. Trigger, który przy dodawaniu nowego klienta lub aktualizacji klienta sprawdza, czy istnieje użytkownik o takim samym id (id_uż) a następnie sprawdza czy wartością jego atrybutu typ jest „klient”. Jeśli nie znajdzie żadnego takiego użytkownika to trigger nic nie zrobi. Jeśli będzie istniał taki użytkownik, ale jego wartość atrybutu typ to nie „klient” to zwróci błąd.

```
CREATE OR REPLACE TRIGGER arc_fkarc_2_klient BEFORE
    INSERT OR UPDATE OF id_uż ON klient
    FOR EACH ROW
DECLARE
    d VARCHAR2(20 CHAR);
BEGIN
    SELECT
        a.typ
    INTO d
    FROM
        uzytkownik a
    WHERE
        a.id_uż = :new.id_uż;

    IF ( d IS NULL OR d <> 'klient' ) THEN
        raise_application_error(-20223, 'FK Klient_Użytkownik_FK in Table
Klient violates Arc constraint on Table Uzytkownik - discriminator column typ
doesn''t have value ''klient'''
        );
    END IF;

EXCEPTION
    WHEN no_data_found THEN
        NULL;
    WHEN OTHERS THEN
        RAISE;
END;
/
```

2. Trigger, który przy dodawaniu nowego administratora lub aktualizacji administratora sprawdza, czy istnieje użytkownik o takim samym id (id_uż) a następnie sprawdza czy wartością jego atrybutu typ jest „administrator”. Jeśli nie znajdzie żadnego takiego użytkownika to trigger nic nie zrobi. Jeśli będzie istniał taki użytkownik, ale jego wartość atrybutu typ to nie „administrator” to zwróci błąd.

```
CREATE OR REPLACE TRIGGER arc_fkarc_2_administrator BEFORE
    INSERT OR UPDATE OF id_uż ON administrator
    FOR EACH ROW
DECLARE
    d VARCHAR2(20 CHAR);
BEGIN
    SELECT
        a.typ
    INTO d
    FROM
        uzytkownik a
    WHERE
        a.id_uż = :new.id_uż;

    IF ( d IS NULL OR d <> 'administrator' ) THEN
        raise_application_error(-20223, 'FK Administrator_Użytkownik_FK in
Table Administrator violates Arc constraint on Table Uzytkownik -
discriminator column typ doesn''t have value ''administrator'''
        );
    END IF;

EXCEPTION
    WHEN no_data_found THEN
        NULL;
    WHEN OTHERS THEN
        RAISE;
END;
/
```

3. Trigger, który przy dodawaniu nowego mechanika lub aktualizacji mechanika sprawdza, czy istnieje użytkownik o takim samym id (id_uż) a następnie sprawdza czy wartością jego atrybutu typ jest „mechanik”. Jeśli nie znajdzie żadnego takiego użytkownika to trigger nic nie zrobi. Jeśli będzie istniał taki użytkownik, ale jego wartość atrybutu typ to nie „mechanik” to zwróci błąd.

```
CREATE OR REPLACE TRIGGER arc_fkarc_2_mechanik BEFORE
    INSERT OR UPDATE OF id_uż ON mechanik
    FOR EACH ROW
DECLARE
    d VARCHAR2(20 CHAR);
BEGIN
    SELECT
        a.typ
    INTO d
    FROM
        uzytkownik a
    WHERE
        a.id_uż = :new.id_uż;

    IF ( d IS NULL OR d <> 'mechanik' ) THEN
          raise_application_error(-20223, 'FK Mechanik_Użytkownik_FK in Table Mechanik
violates Arc constraint on Table Uzytkownik - discriminator column typ doesn''t have value
''mechanik'''
        );
    END IF;

EXCEPTION
    WHEN no_data_found THEN
        NULL;
    WHEN OTHERS THEN
        RAISE;
END;
/
```

4. Trigger, który przy dodawaniu nowego terminu spotkania z klientem lub aktualizacji terminu spotkania z klientem sprawdza, czy istnieje termin o takim samym wpisie (id_wpisu) a następnie sprawdza czy wartością jego atrybutu typ jest „klient”. Jeśli nie znajdzie żadnego takiego terminu to trigger nic nie zrobi. Jeśli będzie istniał taki termin, ale jego wartość atrybutu typ to nie „klient” to zwróci błąd.

```
CREATE OR REPLACE TRIGGER arc_fkarc_1_terminklient BEFORE
    INSERT OR UPDATE OF id_wpisu ON terminklient
    FOR EACH ROW
DECLARE
    d VARCHAR2(20 CHAR);
BEGIN
    SELECT
        a.typ
    INTO d
    FROM
        termin a
    WHERE
        a.id_wpisu = :new.id_wpisu;

    IF ( d IS NULL OR d <> 'klient' ) THEN
        raise_application_error(-20223, 'FK TerminKlient_Termin_FK in Table TerminKlient
violates Arc constraint on Table Termin - discriminator column typ doesn''t have value
''klient'''
        );
    END IF;

EXCEPTION
    WHEN no_data_found THEN
        NULL;
    WHEN OTHERS THEN
        RAISE;
END;
/
```

5. Trigger, który przy dodawaniu nowego terminu pracy przy maszynie lub aktualizacji terminu pracy przy maszynie sprawdza, czy istnieje termin o takim samym wpisie (id_wpisu) a następnie sprawdza czy wartością jego atrybutu typ jest „maszyna”. Jeśli nie znajdzie żadnego takiego terminu to trigger nic nie zrobi. Jeśli będzie istniał taki termin, ale jego wartość atrybutu typ to nie „maszyna” to zwróci błąd.

```
CREATE OR REPLACE TRIGGER arc_fkarc_1_terminmaszyna BEFORE
    INSERT OR UPDATE OF id_wpisu ON terminmaszyna
    FOR EACH ROW
DECLARE
    d VARCHAR2(20 CHAR);
BEGIN
    SELECT
        a.typ
    INTO d
    FROM
        termin a
    WHERE
        a.id_wpisu = :new.id_wpisu;

    IF ( d IS NULL OR d <> 'maszyna' ) THEN
       raise_application_error(-20223, 'FK TerminMaszyna_Termin_FK in Table TerminMaszyna
violates Arc constraint on Table Termin - discriminator column typ doesn''t have value
''maszyna'''
        );
    END IF;

EXCEPTION
    WHEN no_data_found THEN
        NULL;
    WHEN OTHERS THEN
        RAISE;
END;
/
```

6. Trigger, który przy dodawaniu nowego terminu pracy przy stanowisku lub aktualizacji terminu pracy przy stanowisku sprawdza, czy istnieje termin o takim samym wpisie (id_wpisu) a następnie sprawdza czy wartością jego atrybutu typ jest „stanowisko”. Jeśli nie znajdzie żadnego takiego terminu to trigger nic nie zrobi. Jeśli będzie istniał taki termin, ale jego wartość atrybutu typ to nie „stanowisko” to zwróci błąd.

```
CREATE OR REPLACE TRIGGER arc_fkarc_1_terminstanowisko BEFORE
    INSERT OR UPDATE OF id_wpisu ON terminstanowisko
    FOR EACH ROW
DECLARE
    d VARCHAR2(20 CHAR);
BEGIN
    SELECT
        a.typ
    INTO d
    FROM
        termin a
    WHERE
        a.id_wpisu = :new.id_wpisu;

    IF ( d IS NULL OR d <> 'stanowisko' ) THEN
            raise_application_error(-20223,  'FK  TerminStanowisko_Termin_FK  in  Table
TerminStanowisko violates Arc constraint on Table Termin - discriminator column typ
doesn''t have value ''stanowisko'''
        );
    END IF;

EXCEPTION
    WHEN no_data_found THEN
        NULL;
    WHEN OTHERS THEN
        RAISE;
END;
/
```

Link do repozytorium

https://gitlab-stud.elka.pw.edu.pl/pgrabow1/bd2-database-project