Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Informatyki, Elektroniki i Telekomunikacji

Katedra Informatyki

STUDIA PODYPLOMOWE
SYSTEMY BAZ DANYCH

Projekt dyplomowy

System zarządzania e-komisem i naprawami sprzętu gitarowego

Bartosz Kawa

Opiekun Projektu : dr inż. Robert Marcjan

Kierownik Studiów : dr inż. Anna Zygmunt

Kraków 2021

Spis Treści

1. Cel systemu

Celem systemu ma być system zarządzania bazą danych obsługujący usługi pośrednictwa w sprzedaży sprzętu używanego wystawianego przez naszych użytkowników. Dodatkowo, system ma obsługiwać zarządzanie usługami napraw wykonywanymi przez lutników współpracujących ze sklepem. Lutnicy poza naprawą sprzętu są częścią procesu pojawiania się ogłoszenia na stronie. Jest zatem potrzeba stworzenia aplikacji podzielonej na trzy segmenty: serwis dla użytkownika, stronę dla zarządzania ogłoszeniami z perspektywy pracownika jak i stronę umożliwiającą lutnikowi podjęcie zleceń/współpracę. Użytkownik niezalogowany ma możliwość przeglądania ofert sprzętu używanego oraz oferty lutniczej. Może się także zalogować/zarejestrować. Po zalogowaniu, użytkownik zalogowany ma dodatkowo możliwość dokonania zakupu, modyfikacji swojego konta. Użytkownik ten może dodać ogłoszenie sprzętu używanego, wysłać zapytanie o naprawę sprzętu (formularz z konkretnymi wytycznymi tj. akcja strun, zdjęcia, rok zakupu etc.), zamówić usługę lutniczą i kontrolować jej progres, mając możliwość komunikacji z lutnikiem. Po usłudze ma możliwość wystawienia opinii. Pracownik komisu zarządza listą dostępnych produktów. Kontaktuje się z użytkownikiem zalogowanym co do szczegółów ogłoszenia (warunków przyjęcia sprzętu do komisu/ceny etc.). Ponadto, ma dostęp do statystyk sprzedaży jak i informacji związanych z lutnikami współpracującymi ze sklepem. Lutnik po zalogowaniu otrzymuje listę zleceń napraw które może podjąć jak i listę sprzętu komisowego który oczekuje walidacji (tj. czeka na ocenę lutnika przed dodaniem ogłoszenia do oferty komisu). Może dodawać uwagi na temat napraw/wycen. Może przeglądać historię wykonanych zleceń.

2. Wymagania funkcjonalne

1. Dostęp do systemu
   1.1. Osobne interfejsy dla użytkownika, pracownika sklepu i lutnika.
2. Przeglądanie oferty komisu
3. Kupno sprzętu, zamówienie usługi lutniczej.
   3.1. Po dokonanym zakupie możliwość jej zrecenzowania.
   3.2. Zgłoszenie usługi naprawy poprzez wysłanie formularza internetowego.
4. Zarządzanie komisem
   4.1. Dodawanie/usuwanie/modyfikacja ogłoszeń przez pracownika.

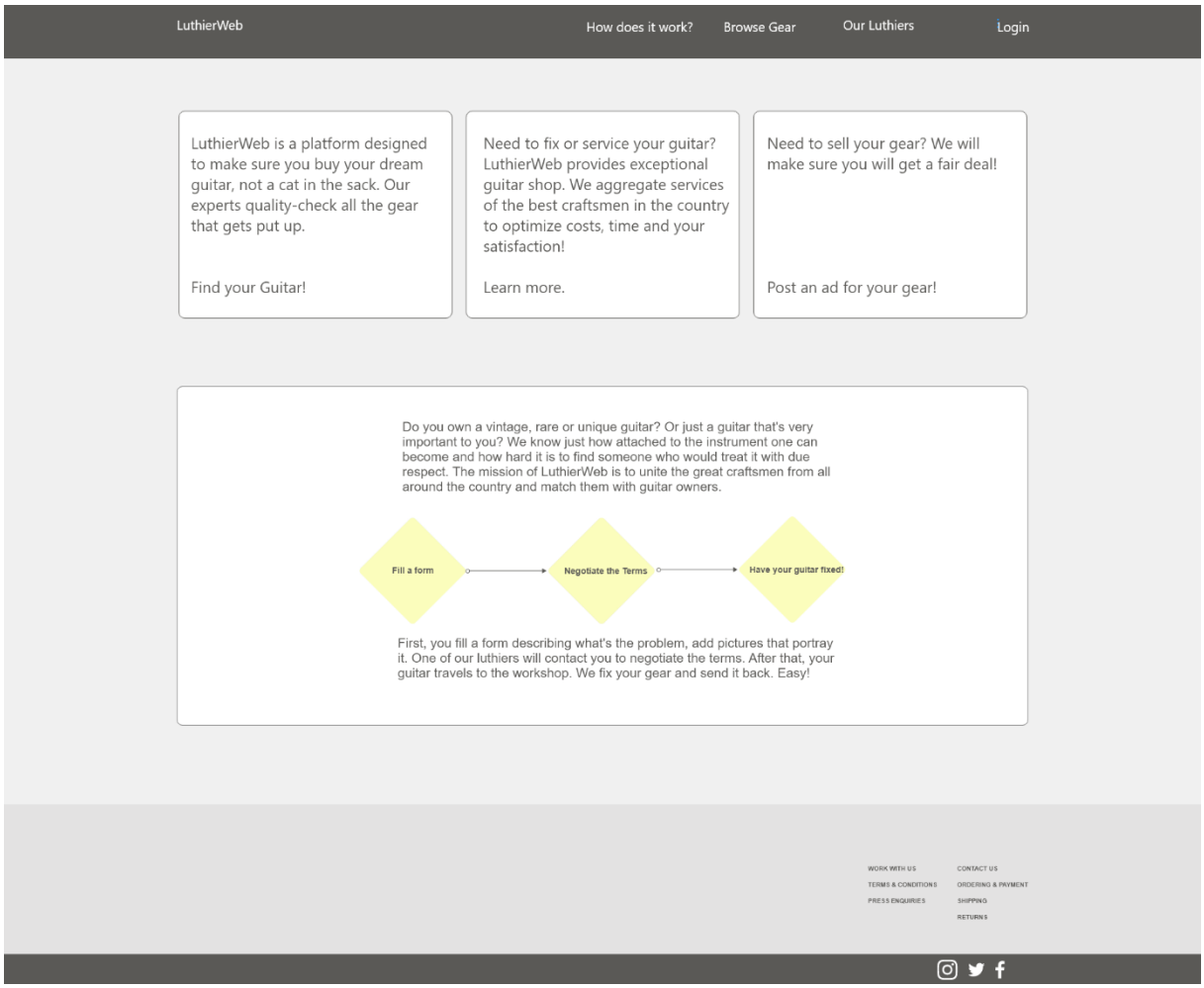
4.2. Wyświetlanie statystyk związanych z prowadzonymi usługami dostępne dla pracownika.

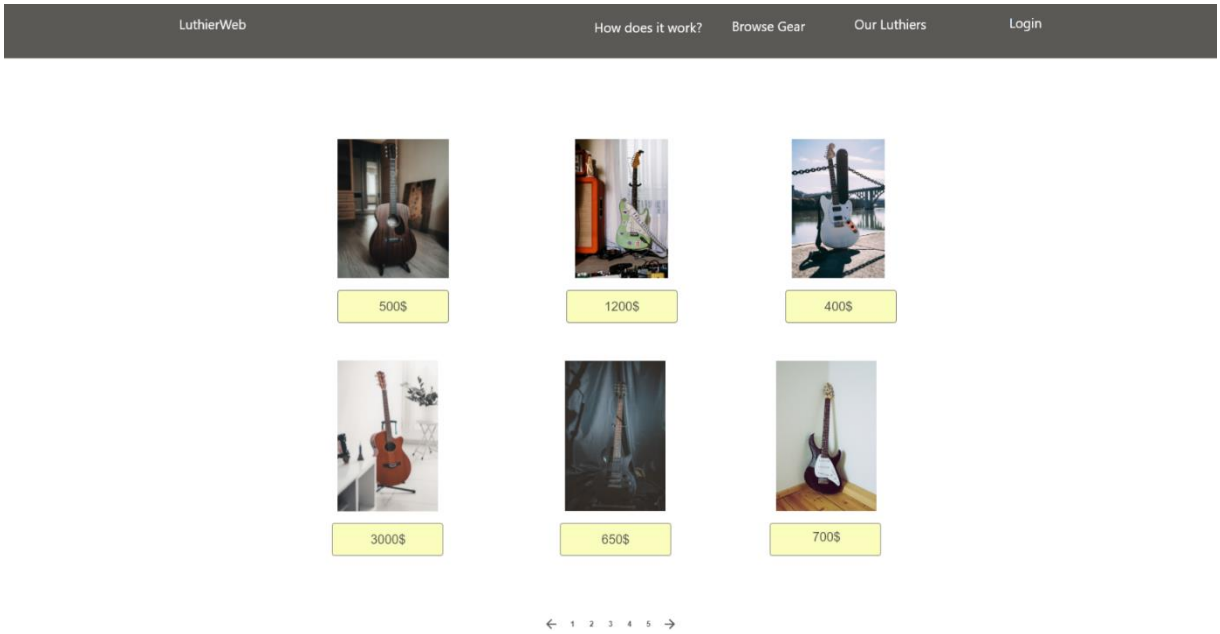
5. Usługi lutnicze
   5.1. Wyświetlanie zleceń do podjęcia
   5.2. Przyjmowanie zleceń
   5.3. Historia zleceń
6. Obieg dokumentów
   6.1. Między użytkownikiem a lutnikiem związany z przyjęciem i zatwierdzeniem zlecenia.
   6.2. Między pracownikiem a użytkownikiem związany z wystawieniem przedmiotu.
   6.3. Między pracownikiem a lutnikiem związany z rejestrowaniem zleceń przyjętych przez lutnika.

3. Projekt interfejsu graficznego

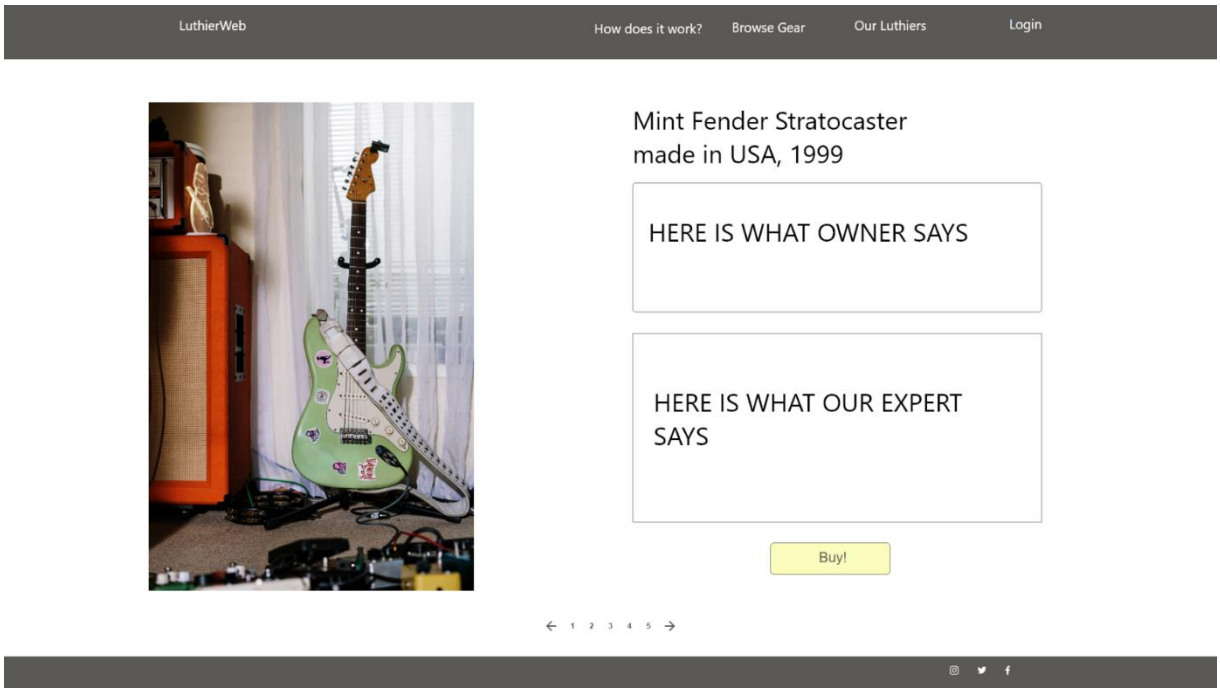
Projekt aplikacji bazodanowej powinien cechować się intuicyjnością i być prosty w nawigacji przez użytkownika. Strona główna aplikacji powinna przekazywać najważniejsze informacje przedstawiające ideę aplikacji i akcje jakie użytkownik może wykonać. Warto zadbać, aby interfejs był przejrzysty i zaprojektowany tak, żeby zminimalizować ilość kroków jakie użytkownik musi podjąć aby wykonać pożądaną dla właściciela strony akcję (pojęcie *konwersji*).

Z uwzględnieniem powyższych wskazówek zaprojektowany został interfejs graficzny; poniżej przedstawione zostały jego fragmenty. Rysunek 1. przedstawia stronę główną aplikacji w przejrzysty sposób opisującą cele aplikacji i akcje możliwe do podjęcia. Rysunek 2. przedstawia proponowany sposób w jaki użytkownik może przeglądać oferty sprzętu dostępne w komisie. Po kliknięciu na wybraną ofertę, pojawiają się jej szczegóły (Rysunek 3.) i przycisk umożliwiający kupno wybranego przedmiotu.





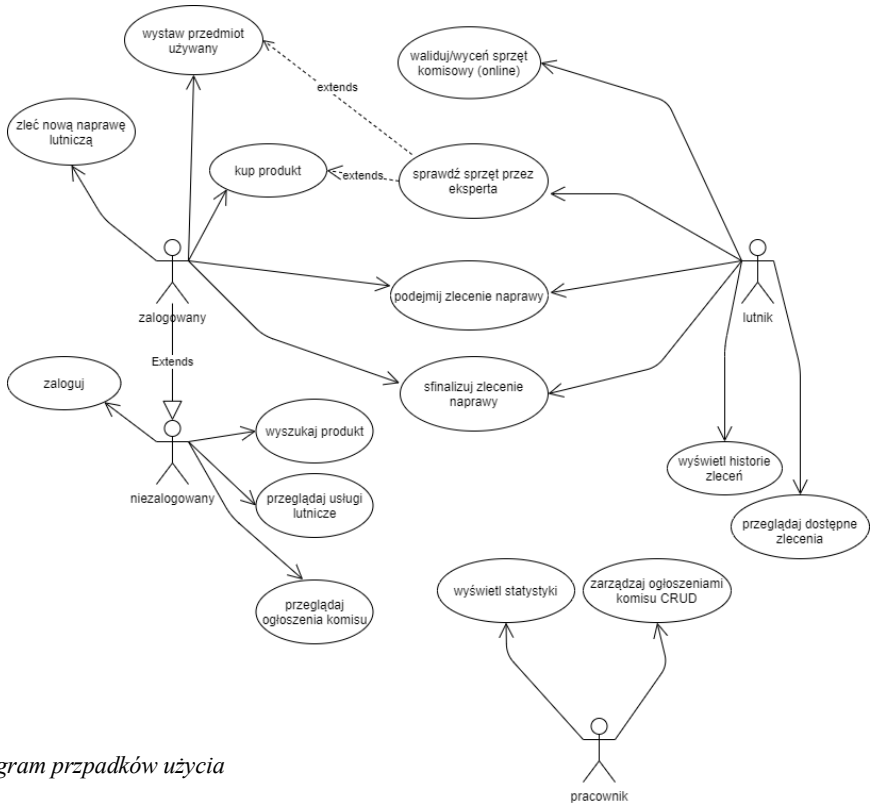
Rysunek 2. Projekt interfejsu graficznego; oferta komisu



Rysunek 3. Projekt interfejsu graficznego; ogłoszenie użytkownika

4. Diagram przypadków użycia

Diagram (Rysunek 4.) przedstawia możliwe użycia systemu z podziałem na użytkowników.



Rysunek 4. Diagram przpadków użycia

Poniżej, w szczegółowy sposób, opisane zostały wybrane scenariusze użycia.

4.1. Wystaw przedmiot używany

Użytkownik wypełnia formularz – Dane zapisują się w encji UserListing. Dostęp do tej tabeli ma Lutnik celem walidacji przedmiotu i akceptacji zapropopnowanej ceny.

1. Jeśli wszystko jest ok, lutnik akceptuje i zmienia status na *Accepted*. Kiedy lutnik zmieni status produktu (ProductStatusID), ogłoszenie jest widoczne na stronie do przeglądania przez użytkownika.
2. Jeśli gitara nie przejdzie walidacji, otrzymuje status *Rejected by Employee* i nie pojawi się w sklepie.
3. Jeśli w gre wchodzi negocjacja ceny, lutnik wprowadza status *in Negotiation* i zmienia cenę produktu na tę zaproponowaną.
4. Po zmianie statusu na *in Negotiation*, użytkownik może zgodzić się bądź odmówić, wtedy status zmienia się odpowiednio na *Accepted* lub *Rejected By Customer*.
5. Jesli ktoś kupi przedmiot, status zmienia się na *Sold*, jeśli opłaci zamówienie, na *Paid*.
6. Kiedy ktoś kupi przedmiot lub minie czas aukcji status zmienia się na *Finished*
7. Każda zmiana zostaje zalogowana w encji ListingStatusHistory. Lutnik ma dostęp do wszystkich nowych ogłoszeń ze statusem *Pending* które sprawdza, zatwierdza w ramach swojej pracy.

4.2. Zleć nową naprawę lutniczą

Użytkownik wypełnia formularz z informacjami o naprawie [zrób sample formularz]. Dane zapisują się w encji ServiceOrder. Do tych danych ma dostęp lutnik. Wyświetlane są z jego perspektywy jako zamówienia napraw.

1. Początkowy status napraw to *Pending*. Lutnik przegląda zlecenia i może je zaakceptować.
2. Kiedy zdecyduje się na naprawę, status zmienia się na *initially Accepted by Employee*, podaje także estymowaną cenę naprawy. Kiedy status zostaje zmieniony na *initially Accepted by Employee*, użytkownik może:
   2.1. odrzucić zgłoszenie, status zmienia sie na *Rejected by Customer*
   2.2. zaakceptować naprawę za określoną cenę zmieniając status na *initially Accepted by Customer*

3. Kiedy użytkownik zaakceptuje cenę, sprzęt zostaje wysyłany do lutnika (UserAddress). Po dotarciu sprzętu, lutnik ocenia jego rzeczywisty stan i ostatecznie wycenia usługę.
   3.1. Jeśli usterka zgodna jest z opisem, status zostaje zmieniony na *Accepted* i lutnik podejmuje się naprawy.
   3.2. Jeśli zdjęcia/opisy odbiegają od rzeczywistego problemu i naprawienie usterki ma inną wartość, lutnik aktualizuje kolumny (ServiceCost) i (OrderStatusID) uzupełniając cenę i zmieniając status na *in Negotiation*. Może w tym momencie także odmówić podjęcia się zlecenia jeśli opis problemu/zdjęcia rażąco odbiega od stanu faktycznego, zmieniając status na *rejected By Employee*.
4. Jeśli status zmieni się na *in Negotiation*, użytkownik zlecający naprawę może odrzucić zaproponowaną cenę lub ją zaakceptować. Wtedy status odpowiednio zmienia się na *Rejected by Customer,* lub *Accepted*.
5. Wszystkie wykonane akcje zostają odnotowane w encji LuthierInteractionsHist

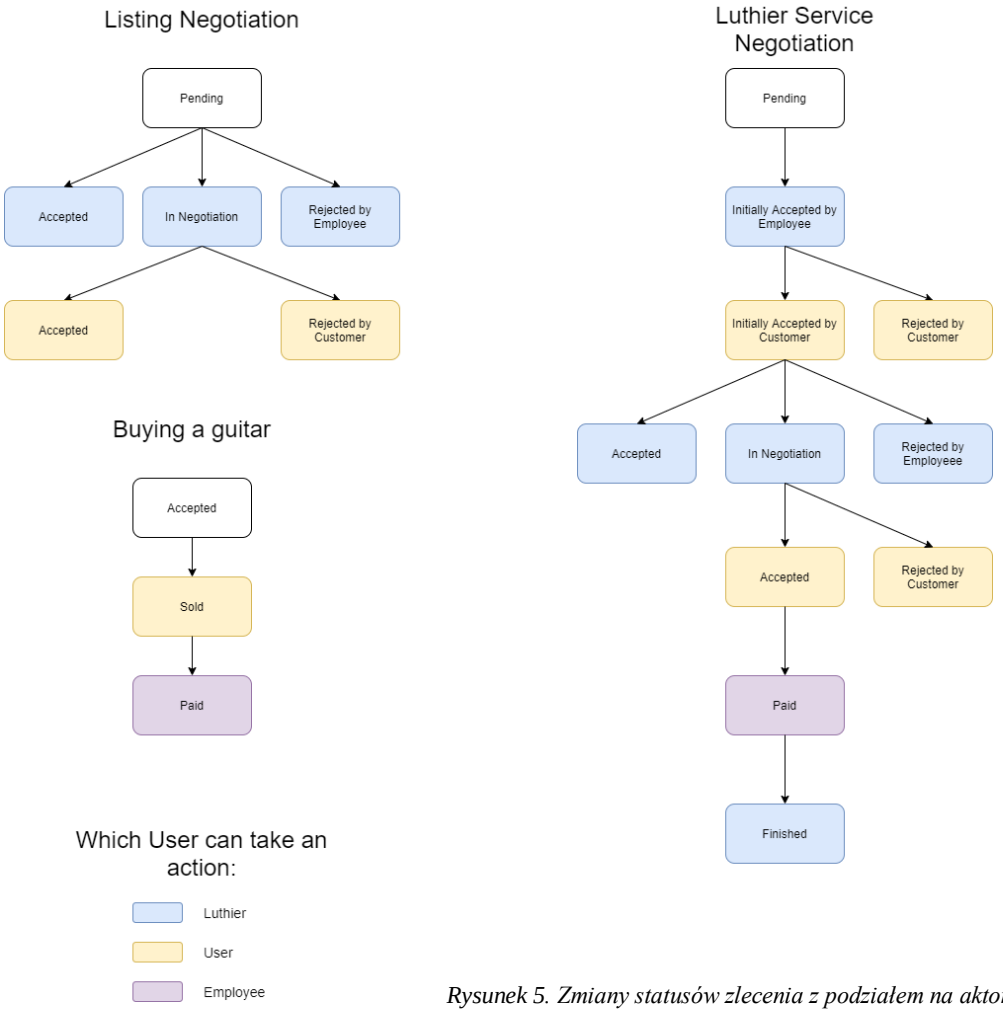
4.3. Kup produkt

Użytkownik przegląda oferty sprzętu o których dane znajdują się w encji UserListing. Kiedy zdecyduje się kupić przedmiot, status (ProductStatus) zmienia się na *Sold.* Kiedy następuje ta zmiana, informacja jest odnotowywana w (ListingStatusHistory). Użytkownik opłaca zamówienie, jeśli pieniadze zostaną zaksięgowane (ProductStatus) zmienia się na *Paid*

We wszystkich powyższych scenariuszach, jeśli status zostanie zmieniony na Rejected by Customer lub Rejected by Employee, oznacza to, że transakcja została przerwana a zatem takie ogłoszenia zostają automatycznie usuwane z bazy danych.

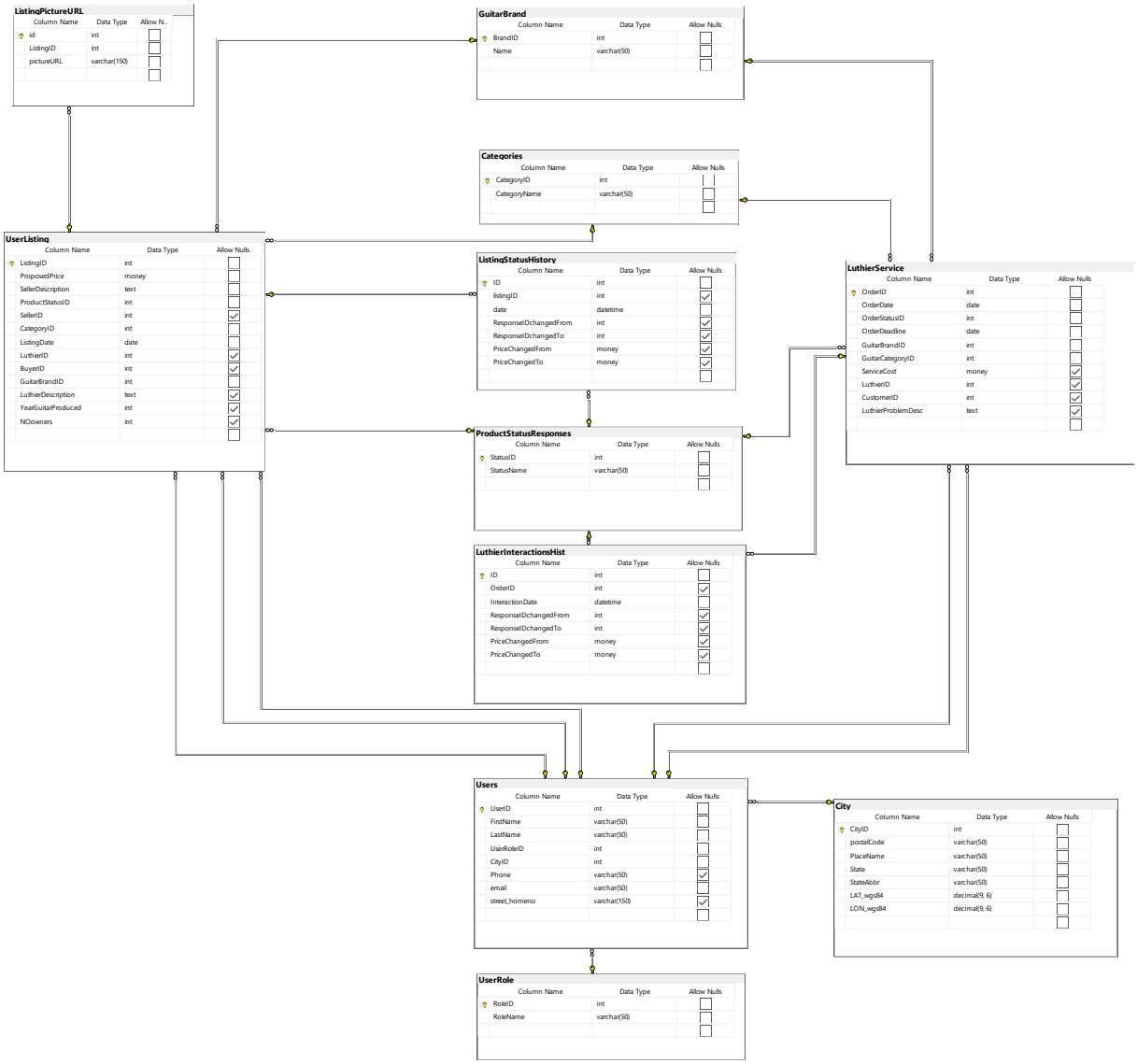
4.4. Wybrane scenariusze: przedstawienie graficzne

Rysunek 5. zawiera graficzne przedstawienie zmian statusów zamówień w wymienionych wyżej scenariuszach:



Rysunek 5. Zmiany statusów zlecenia z podziałem na aktorów.

5. Schemat bazy Danych



Rysunek 6. Schemat bazy danych

Nazwa tabeli	Opis
UserListing	Tabela zawierająca informacje z formularza wypełnionego przez użytkownika podczas wystawiana gitar na sprzedaż. Zawiera informacje o wystawionym przedmiocie i szczegółach związanych ze sprzedażą.
Users	Tabela zawierająca informacje o użytkownikach i pracownikach komisu/serwisu
LuthierService	Tabela zawierająca informacje o zleceniach lutniczych i szczegółach związanych z postępami w ich wykonaniu.
GuitarBrand	Tabela słownikowa; zawiera nazwy przyjmowanych do komisu/serwisu gitar
Categories	Tabela słownikowa; zawiera nazwy kategorii do których należą gitary.
ListingStatusHistory	Tabela z danymi audytowymi. Zawiera informacje o zmianach statusów poszczególnych ogłoszeń
ProductStatusResponses	Tabela słownikowa; zawiera nazwy statusów jakie mogą zostać nadane ogłoszeniom/usługom
LuthierInteractionsHist	Tabela z danymi audytowymi. Zawiera informacje o zmianach statusów poszczególnych zleceń lutniczych
UserRole	Tabela słownikowa; zawiera nazwy ról użytkowników
City	Tabela słownikowa; zawiera nazwy miast związanych z użytkownikami/pracownikami
ListingPictureUrl	Tabela zawiera nazwy przechowywanych na serwerze zdjęć związach z aukcjami użytkowników i napraw lutniczych.

Tabela 1. Tabele w bazie danych

6. Widoki, procedury, triggery.

W tej części pracy zostanie przedstawiona implementacja widoków, procedur i triggerów umożliwiająca usprawnione korzystanie z bazy danych.

vWBrowseListingsCustomer

Widok zwraca informacje o gitarach.

```
create view [dbo].[vWBrowseListingsCustomer]
as
select ProposedPrice,SellerDescription,CategoryName,ListingDate,
       (Users.FirstName +' ' + Users.LastName) as LuthierName,
       GuitarBrand.Name as GuitarBrand, LuthierDescription,
       YearGuitarProduced, NOowners
from UserListing join Categories on UserListing.CategoryID =
                                    Categories.CategoryID
                join Users on UserListing.LuthierID=Users.UserID
                join GuitarBrand on UserListing.GuitarBrandID =
                                    GuitarBrand.BrandID

where ProductStatusID = 1
```

tr_vWCustomerDetails_InsteadOfInsert

W celu dodawania danych w tabeli Users, na widok vWCustomerDetails założony został trigger instead. W ten sposób użytkownik pracuje na widoku zamiast pracować bezpośrednio na tabeli Users.

```
CREATE trigger [dbo].[tr_vWCustomerDetails_InsteadOfInsert]
on [dbo].[vWCustomerDetails]
instead of insert
as
begin
      declare @userRole int = 3
      declare @fname varchar(50)
      declare @lname varchar(50)
      declare @phone varchar(50)
      declare @email varchar(50)
      declare @postalCode varchar(50)
      declare @state varchar(50)
      declare @placename varchar(50)
      declare @street varchar(150)


select @fname=FirstName, @lname=LastName, @phone=Phone, @email=email,
```

```
       @postalCode=postalCode, @state=state, @placename=placename,
       @street=street_homeno
from inserted

declare @cityID int

select @CityID=cityID from City where postalCode = @postalCode and
        state=@state and placename=@placename

if (@cityID is not NULL)
      insert into
      Users(FirstName,LastName,UserRoleID,CityID,Phone,email,street_homeno)
      values(@fname,@lname,@userRole,@CityID,@phone,@email,@street)
else
      RAISERROR('Check your address: city or state doesnt match postal
                code',16,1)
End
```

tr_vWCustomerDetails_InsteadOfUpdate

W celu aktualizacji danych w tabeli Users, na widok vWCustomerDetails założony został trigger instead. W ten sposób użytkownik pracuje na widoku zamiast pracować bezpośrednio na tabeli Users.

```
CREATE trigger [dbo].[tr_vWCustomerDetails_InsteadOfUpdate]
on [dbo].[vWCustomerDetails]
instead of update
as
begin
      declare @id int
      if(Update(FirstName))
      begin
            declare @fname varchar(50)
            select @id = UserID, @fname=FirstName from inserted
                  update Users set FirstName=@fname where UserID = @id
      end
      if(Update(LastName))
      begin
            declare @lname varchar(50)
            select @id = UserID, @lname=LastName from inserted
            update Users set LastName=@lname where UserID = @id
      end
      if(Update(Phone))
      begin
            declare @phone varchar(50)
            select @id= UserID,@phone=Phone from inserted
            update Users set Phone=@phone where UserID = @id
      end
      if(Update(email))
      begin
            declare @email varchar(50)
            select @id= UserID,@email=email from inserted
            update Users set email=@email where UserID = @id
```

```
	end
	if(Update(street_homeno))
	begin
		declare @street varchar(150)
		select @id= UserID,@street=street_homeno from inserted
		update Users set street_homeno=@street where UserID = @id
	end
	if(Update(postalCode) or Update(state) or Update(placename))
	begin
		declare @postalCode varchar(50)
		declare @state varchar(50)
		declare @placename varchar(50)
		select @id= UserID, @postalCode = postalCode,@state=state,
			@placename = placename from inserted

		declare @cityID int
		select @CityID=cityID from City where postalCode = @postalCode
			and state=@state and placename=@placename
		if (@cityID is not NULL)
			update Users set CityID=@cityID where UserID = @id
		else
		RAISERROR('Check your address, city or state doesn't match
				postal code',11,1)
	end
end
```

spViewLuthierServiceStatusByID

Procedura zwracająca aktualny status dla danego wystawionego przedmiotu. Używana wewnątrz procedury spChangeListingProductStatus.

```
CREATE procedure [dbo].[spViewLuthierServiceStatusByID]
@orderid int,
@currentOrderStatus int output,
@display int = 1
as
begin
select @currentOrderStatus = OrderStatusID from LuthierService
where OrderID=@orderid
if (@display in (1,0))
begin
	if (@display=1)
	begin
		select ServiceCost,StatusName
		from LuthierService join ProductStatusResponses on
			LuthierService.OrderStatusID=ProductStatusResponses.Statu
			sID
		where OrderID=@orderid
	end
end
else
	raiserror('incorrect parameters given',16,1)
end
```

spChangeListingProductStatus

Procedura zmieniająca status przedmiotu. Umożliwia interakcję lutnika z użytkownikiem, jak i zakup gitary przez użytkownika. Wartości które można wprowadzić do procedury są ograniczone tak, aby dany aktor mógł wykonać tylko sensowne i dozwolone zmiany na kolejnych etapach realizacji zamówienia.

```
CREATE procedure [dbo].[spChangeListingProductStatus]
/*
ARGUMENTS:
      1.USER ID
      2.ID OF LISTING STATUS OF YOU WANT TO CHANGE
      3.ID OF STATUS YOU WANT LISTING TO CHANGE TO
      4.NEW PRICE YOU WANT TO ASSIGN TO LISTING SET TO NULL
      5.TEXT LUTHIER WANT TO ADD TO LISTING SET TO NULL
*/

@userID int
@listingID int,
@newStatusID int,
@newPrice money = null,
@LuthierText text = null
as
begin
declare @userRole int
select @userrole = UserRoleID from Users where UserID= @userID

declare @currentStatusID int
execute spViewProductStatusByListingID @listingID, @currentStatusID
output,0
--check if data entered is correct
if @newStatusID in (select StatusID from ProductStatusResponses)
begin
      if ((@currentStatusID = 0 and @userRole = 2) or (@currentStatusID = 2
          and @userRole = 3) or (@currentStatusID = 1 and @userRole = 3)
          or (@currentStatusID = 4 and @userRole = 1))
      begin
            if (@currentStatusID = 0 and @userRole = 2)
            begin
                  if @newStatusID in (1, 3, 2)
                        print 'values entered correct'
                  else
                  begin
                        raiserror('you cannot change status. Forbidden
                                    change, check documentation',16,1)
                        return
                  end
            end
            if (@currentStatusID = 2 and @userRole = 3)
            begin
                  if @newStatusID in (1, 31)
                        print 'values entered correct'
                  else
```

```
                begin
                        raiserror('you cannot change status. Forbidden
                                    change, check documentation',16,1)
                        return
                end
        end
        if (@currentStatusID = 1 and @userRole = 3) --buying a guitar
        begin
                if @newStatusID in (4)
                        print 'values entered correct'
                else
                begin
                        raiserror('you cannot change status. Forbidden
                                    change, check documentation',16,1)
                        return
                end
        end
        if (@currentStatusID = 4 and @userRole = 1)
        begin
                if @newStatusID in (5)
                        print 'values entered correct'
                else
                begin
                        raiserror('you cannot change status. Forbidden
                                    change, check documentation',16,1)
                        return
                end
        end
    end
    else
        begin
                raiserror('you cannot change status. Forbidden
                            change,check documentation',16,1)
                return
        end
    end
    else
    begin
        raiserror('wrong status name provided, check
                    documentation',16,1)
        return
    end


if @listingID in (select ListingID from UserListing)
begin
    update UserListing set ProductStatusID = @newStatusID where
            ListingID=@listingID
    if (@newPrice is not null)
        update UserListing set ProposedPrice = @newPrice where
                ListingID=@listingID
    if (@LuthierText is not null)
        update UserListing set LuthierDescription = @LuthierText where
                ListingID=@listingID
    if @newStatusID in (3,31)
```

```
            delete from UserListing where listingid=@listingID
 end
 else
      raiserror('Given ListingID does not exist',16,1)
end
```

spViewLuthierServiceStatusByID

Procedura zwracająca aktualny status dla danej usługi serwisowej. Procedura używana wewnętrznie przez spChangeLuthierServiceStatus.

```
CREATE procedure [dbo].[spViewLuthierServiceStatusByID]
@orderid int,
@currentOrderStatus int output,
@display int = 1
as
begin
select @currentOrderStatus = OrderStatusID from LuthierService
where OrderID=@orderid

if (@display in (1,0))
begin
      if (@display=1)
      begin
            select ServiceCost,StatusName
            from LuthierService
            join ProductStatusResponses on
               LuthierService.OrderStatusID=ProductStatusResponses.StatusID
            where OrderID=@orderid
      end
end
else
      raiserror('incorrect parameters given',16,1)
end
```

spChangeLuthierServiceStatus

Procedura zmieniająca status usługi serwisowej. Wartości które można wprowdzić do procedury są ograniczone tak, aby dany aktor mógł wykonać tylko sensowne i dozwolone zmiany.

```
CREATE procedure [dbo].[spChangeLuthierServiceStatus]

@userID int,-- zakładamy, że aby dokonać zmany, trzeba podać kto dokonuje
zmiany --> UserID
```

```
@orderID int,
@newStatusID int,
@newPrice money = null,
@LuthierText text = null

as
begin
declare @userRole int
select @userrole = UserRoleID from Users where UserID= @userIS
declare @currentStatusID int
execute spViewLuthierServiceStatusByID @orderID, @currentStatusID output,0
--check if data entered is correct (for app logic purposes)
if @newStatusID in (select StatusID from ProductStatusResponses)
begin
      if ((@currentStatusID = 0 and @userRole = 2) or
         (@currentStatusID = 11 and @userRole = 3) or
         (@currentStatusID = 12 and @userRole = 2) or
         (@currentStatusID = 2 and @userRole = 3)  or
         (@currentStatusID = 1 and @userRole = 1) or
         (@currentStatusID = 5 and @userRole = 2))
      begin
            if (@currentStatusID = 0 and @userRole = 2)
                  if @newStatusID in (11)
                        print 'values entered correct'
                  else
                  begin
                        raiserror('you cannot change status. Forbidden
                                    change, check documentation',16,1)
                        return
                  end
                  if (@currentStatusID = 11 and @userRole = 3)
                  begin
                        if @newStatusID in (12, 31)
                              print 'values entered correct'
                        else
                        begin
                              raiserror('you cannot change status.
                                          Forbidden change, check
                                          documentation',16,1)
                              return
                        end
                  end
                  if (@currentStatusID = 12 and @userRole = 2)
                  begin
                        if @newStatusID in (1, 2, 3)
                              print 'values entered correct'
                        else
                        begin
                              raiserror('you cannot change status.
                                          Forbidden change, check
                                          documentation',16,1)
                              return
                        end
                  end
                  if (@currentStatusID = 2 and @userRole = 3)
                  begin
```

```
                        if @newStatusID in (1, 12)
                              print 'values entered correct'
                        else
                        begin
                              raiserror('you cannot change status.
                                          Forbidden change, check
                                          documentation',16,1)
                              return
                        end
                  end
                  if (@currentStatusID = 1 and @userRole = 1)
                  begin
                        if @newStatusID in (5)
                              print 'values entered correct'
                        else
                        begin
                              raiserror('you cannot change status.
                                          Forbidden change, check
                                          documentation',16,1)
                              return
                        end
                  end
                  if (@currentStatusID = 5 and @userRole = 2)
                  begin
                        if @newStatusID in (99)
                              print 'values entered correct'
                        else
                        begin
                              raiserror('you cannot change status.
                                          Forbidden change, check
                                          documentation',16,1)
                              return
                        end
                  end
            end
            else
            begin
                  raiserror('you cannot change status. Forbidden change,
                              check documentation',16,1)
                  return
            end
      end
      else
      begin
            raiserror('you cannot change status. Forbidden change, check
                        documentation',16,1)
            return
      end

if @OrderID in (select OrderID from LuthierService)
begin
      update LuthierService set OrderStatusID = @newStatusID
             where OrderID=@orderID
      if (@newPrice is not null)
            update LuthierService set ServiceCost = @newPrice
                   where OrderID=@orderID
```

```
	if (@LuthierText is not null)
		update LuthierService set LuthierProblemDesc = @LuthierText
			where OrderID=@orderID
	if @newStatusID in (3,31)
		delete from LuthierService where OrderID=@orderID
end
else
	raiserror('Given Order ID does not exist',16,1)
end
```

vwDisplayLuthierServiceJobs

Widok zwraca zlecenia widoczne dla lutnika przy podejmowaniu zlecenia. Widok pokazuje wszystkie ogłoszenia ze statusem Pending, czyli te dostępne do podjęcia.

```
CREATE view [dbo].[vwDisplayLuthierServiceJobs]
as
select luthierID, statusname,OrderDate,OrderDeadline,
       GuitarBrand.Name as brand,Categories.CategoryName,
       ServiceCost,
       Users.FirstName+ ' ' + users.LastName
           as [Customer Name], LuthierProblemDesc
from LuthierService join GuitarBrand on Luthierservice.GuitarBrandID =
                                    GuitarBrand.BrandID
                    join Categories on LuthierService.GuitarCategoryID =
                                    Categories.CategoryID
                    join Users on LuthierService.CustomerID = Users.UserID
                    join ProductStatusResponses on
                       LuthierService.OrderStatusID =
                       ProductStatusResponses.StatusID
```

spOrderByServiceJobs

Procedura pozwalająca posegregować ogłoszenia po dacie, terminie wykonania usługi, producencie gitary (rosnąco lub malejąco). Jeśli podany zostanie parametr luthierID, procedura zwraca zlecenia dla poszczególnych lutników (zlecenia powiązane z danym pracownikiem). Jeśli podany zostanie StatusName to lutnik może segregować swoje prace w zależności od ich statusu.

```
CREATE procedure [dbo].[spOrderByServiceJobs]
@luthierID varchar(50) = 'not assigned',
@statusName varchar(50) = 'Pending',
@orderbywhat varchar(50) = 'date',
@asc_desc varchar(50) = 'asc'
as
```

```
begin
if (exists(select * from Users where UserID=@luthierID) or
    @luthierID='not assigned') and
    exists(select * from ProductStatusResponses
      where StatusName=@statusName) and (@orderbywhat = 'date' or
            @orderbywhat = 'deadline' or @orderbywhat = 'brand') and
            (@asc_desc = 'asc' or @asc_desc = 'desc')

begin
      if (@luthierID='not assigned' and @statusName='Pending')
      begin
            if (@orderbywhat='date' and @asc_desc='asc')
                  select OrderDate, OrderDeadline, brand, CategoryName,
                        ServiceCost, [Customer Name]
                  from vwDisplayLuthierServiceJobs
                  where statusname=@statusName order by OrderDate asc
            if (@orderbywhat='date' and @asc_desc='desc')
                  select OrderDate, OrderDeadline, brand, CategoryName,
                        ServiceCost, [Customer Name]
                  from vwDisplayLuthierServiceJobs
                  where statusname=@statusName order by OrderDate desc
            if (@orderbywhat='deadline' and @asc_desc='asc')
                  select OrderDate, OrderDeadline, brand, CategoryName,
                        ServiceCost, [Customer Name]
                  from vwDisplayLuthierServiceJobs
                  where statusname=@statusName order by OrderDeadline asc
            if (@orderbywhat='deadline' and @asc_desc='desc')
                  select OrderDate, OrderDeadline, brand, CategoryName,
                        ServiceCost, [Customer Name]
                  from vwDisplayLuthierServiceJobs
                  where statusname=@statusName
                  order by OrderDeadline desc



            if (@orderbywhat='brand' and @asc_desc='asc')
                  select OrderDate, OrderDeadline, brand, CategoryName,
                         ServiceCost, [Customer Name]
                  from vwDisplayLuthierServiceJobs
                  where statusname=@statusName
                  order by brand asc
            if (@orderbywhat='brand' and @asc_desc='desc')
            select OrderDate, OrderDeadline, brand, CategoryName,
                   ServiceCost, [Customer Name]
            from vwDisplayLuthierServiceJobs
            where statusname=@statusName
            order by brand desc

      end

      if (@luthierID !='not assigned')
      begin
            if @statusName='Pending'
            begin
                  if (@orderbywhat='date' and @asc_desc='asc')
                        select OrderDate, OrderDeadline, brand,
```

```
                  CategoryName, ServiceCost, [Customer Name]
            from vwDisplayLuthierServiceJobs
            where luthierID=@luthierID
            order by OrderDate asc
      if (@orderbywhat='date' and @asc_desc='desc')
            select OrderDate, OrderDeadline, brand,
                  CategoryName, ServiceCost, [Customer Name]
            from vwDisplayLuthierServiceJobs
            where luthierID=@luthierID
            order by OrderDate desc
      if (@orderbywhat='deadline' and @asc_desc='asc')
            select OrderDate, OrderDeadline, brand,
                  CategoryName, ServiceCost, [Customer Name]
            from vwDisplayLuthierServiceJobs
            where luthierID=@luthierID
            order by OrderDeadline asc
      if (@orderbywhat='deadline' and @asc_desc='desc')
            select OrderDate, OrderDeadline, brand,
                  CategoryName, ServiceCost, [Customer Name]
            from vwDisplayLuthierServiceJobs
            where luthierID=@luthierID
            order by OrderDeadline desc
      if (@orderbywhat='brand' and @asc_desc='asc')
            select OrderDate, OrderDeadline, brand,
                  CategoryName, ServiceCost, [Customer Name]
            from vwDisplayLuthierServiceJobs
            where luthierID=@luthierID
            order by brand asc
      if (@orderbywhat='brand' and @asc_desc='desc')
            select OrderDate, OrderDeadline, brand,
                  CategoryName, ServiceCost, [Customer Name]
            from vwDisplayLuthierServiceJobs
            where luthierID=@luthierID
            order by brand desc

end
else
begin
      if (@orderbywhat='date' and @asc_desc='asc')
            select OrderDate, OrderDeadline, brand,
                  CategoryName, ServiceCost, [Customer Name]
            from vwDisplayLuthierServiceJobs
            where luthierID=@luthierID and statusname=
                  @statusName
            order by OrderDate asc
      if (@orderbywhat='date' and @asc_desc='desc')
            select OrderDate, OrderDeadline, brand,
                  CategoryName, ServiceCost, [Customer Name]
            from vwDisplayLuthierServiceJobs
            where luthierID=@luthierID and statusname =
                  @statusName
            order by OrderDate desc

      if (@orderbywhat='deadline' and @asc_desc='asc')
            select OrderDate, OrderDeadline, brand,
                  CategoryName, ServiceCost, [Customer Name]
```

```
                        from vwDisplayLuthierServiceJobs
                        where luthierID=@luthierID and statusname=
                              @statusName
                        order by OrderDeadline asc
                  if (@orderbywhat='deadline' and @asc_desc='desc')
                        select OrderDate, OrderDeadline, brand,
                              CategoryName, ServiceCost, [Customer Name]
                        from vwDisplayLuthierServiceJobs
                        where luthierID=@luthierID and statusname=
                              @statusName
                        order by OrderDeadline desc
                  if (@orderbywhat='brand' and @asc_desc='asc')
                        select OrderDate, OrderDeadline, brand,
                               CategoryName, ServiceCost, [Customer Name]
                        from vwDisplayLuthierServiceJobs
                        where luthierID=@luthierID and statusname=
                              @statusName
                        order by brand asc
                  if (@orderbywhat='brand' and @asc_desc='desc')
                        select OrderDate, OrderDeadline, brand,
                               CategoryName, ServiceCost, [Customer Name]
                        from vwDisplayLuthierServiceJobs
                        where luthierID=@luthierID and statusname=
                              @statusName
                        order by brand desc
            end

      end
end
else
      raiserror('incorrect argument(s) supplied',16,1)
end
```

spDeleteUserByID

Usuwanie użytkowników. Aby umożliwić usuwanie użytkowników, ze względu na konstrukcję bazy danych, potrzebne było użycie procedury.

```
CREATE proc [dbo].[spDeleteUserByID]
@userIDtoDelete int
as
begin
      update UserListing set sellerID=null where SellerID=@userIDtoDelete
      update UserListing set buyerID=null where buyerID=@userIDtoDelete
      update UserListing set  luthierID=null
            where luthierID=@userIDtoDelete
      update LuthierService set luthierID=null
            where luthierID=@userIDtoDelete
      update LuthierService set customerID=null
            where customerID=@userIDtoDelete
```

```
	BEGIN TRY
		delete from Users where UserID=@userIDtoDelete
		print('User deleted successfully')
	END TRY
	BEGIN CATCH
		raiserror('User with given ID do not exist in db',16,1)
	END CATCH

end
```

tr_tblUserListing_ForUpdate

Dane audytowe; śledzenie zmian w wystawionych ogłoszeniach. Jeśli zmieni się status ogłoszenia lub jego cena, zostanie wywołany trigger logujący tę zmianę w osobnej tabeli.

```
CREATE trigger [dbo].[tr_tblUserListing_ForUpdate]
on [dbo].[UserListing]
for update
as
begin
declare @listingID int
declare @responseIDbefore int
declare @responseIDafter int
declare @priceBefore int
declare @priceAfter int

select @listingID = ListingID, @responseIDbefore = ProductStatusID,
	@priceBefore = ProposedPrice
from deleted

select @responseIDafter = ProductStatusID,@priceAfter = ProposedPrice
from inserted

if (@responseIDbefore != @responseIDafter) or (@priceBefore !=@priceAfter)
	insert into ListingStatusHistory
	values(@listingID, CURRENT_TIMESTAMP, @responseIDbefore,
		@responseIDafter, @priceBefore, @priceAfter)
end
```

tr_tblLuthierService_ForUpdate

Dane audytowe; śledzenie zmian w usługach lutniczych. Jeśli zmieni się status usługi lub jej cena, zostanie wywołany trigger logujący tę zmianę w osobnej tabeli.

```
CREATE trigger [dbo].[tr_tblLuthierService_ForUpdate]
on [dbo].[LuthierService]
```

```
for update
as
begin
declare @OrderID int
declare @responseIDbefore int
declare @responseIDafter int
declare @priceBefore int
declare @priceAfter int

select @OrderID = OrderID,@responseIDbefore =
       OrderStatusID,@priceBefore=ServiceCost
from deleted
select @responseIDafter = OrderStatusID,@priceAfter=ServiceCost
from inserted

if (@responseIDbefore != @responseIDafter) or (@priceBefore !=@priceAfter)
      insert into LuthierInteractionsHist
      values(@OrderID,CURRENT_TIMESTAMP, @responseIDbefore,
             @responseIDafter, @priceBefore, @priceAfter)
end
```

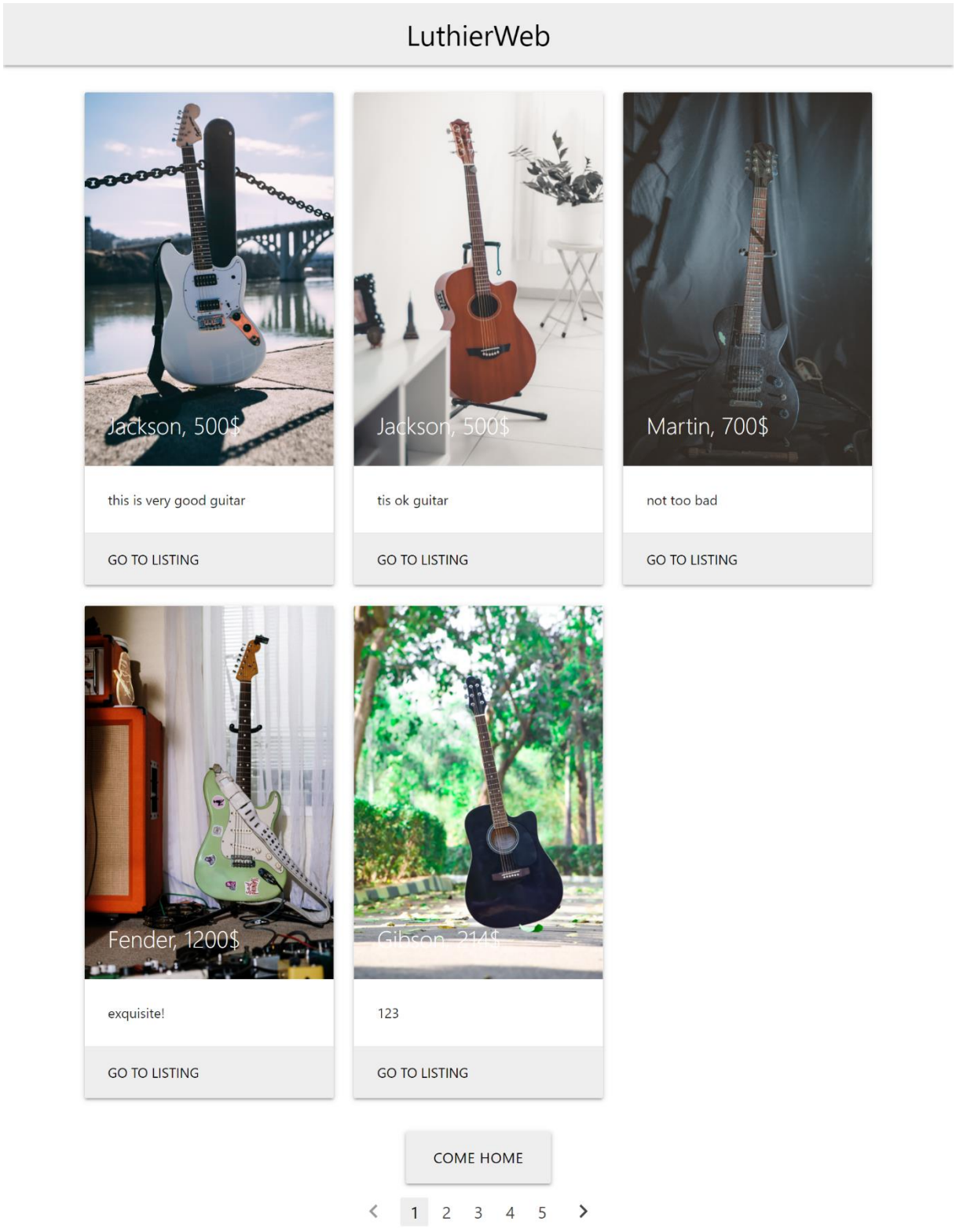
7. Elementy aplikacji

W celu stworzenia aplikacji bazodanowej wykorzystany został język programowania *Python* i framework webowy *Django* w wersji 3.0. Z użyciem biblioteki *django-mssql-backend* baza danych, stworzona w *Microsoft SQL Server,* została połączona z aplikacją. Do zaimplementowania części front-end aplikacji użyty został framework *Materialize.*

Zaimplementowane zostały fragmenty aplikacji umożliwiające przeglądanie ofert komisu, wyświetlanie szczegółów konkretnego ogłoszenia, i formularz dzięki któremu użytkownik dodać może nowe ogłoszenie.

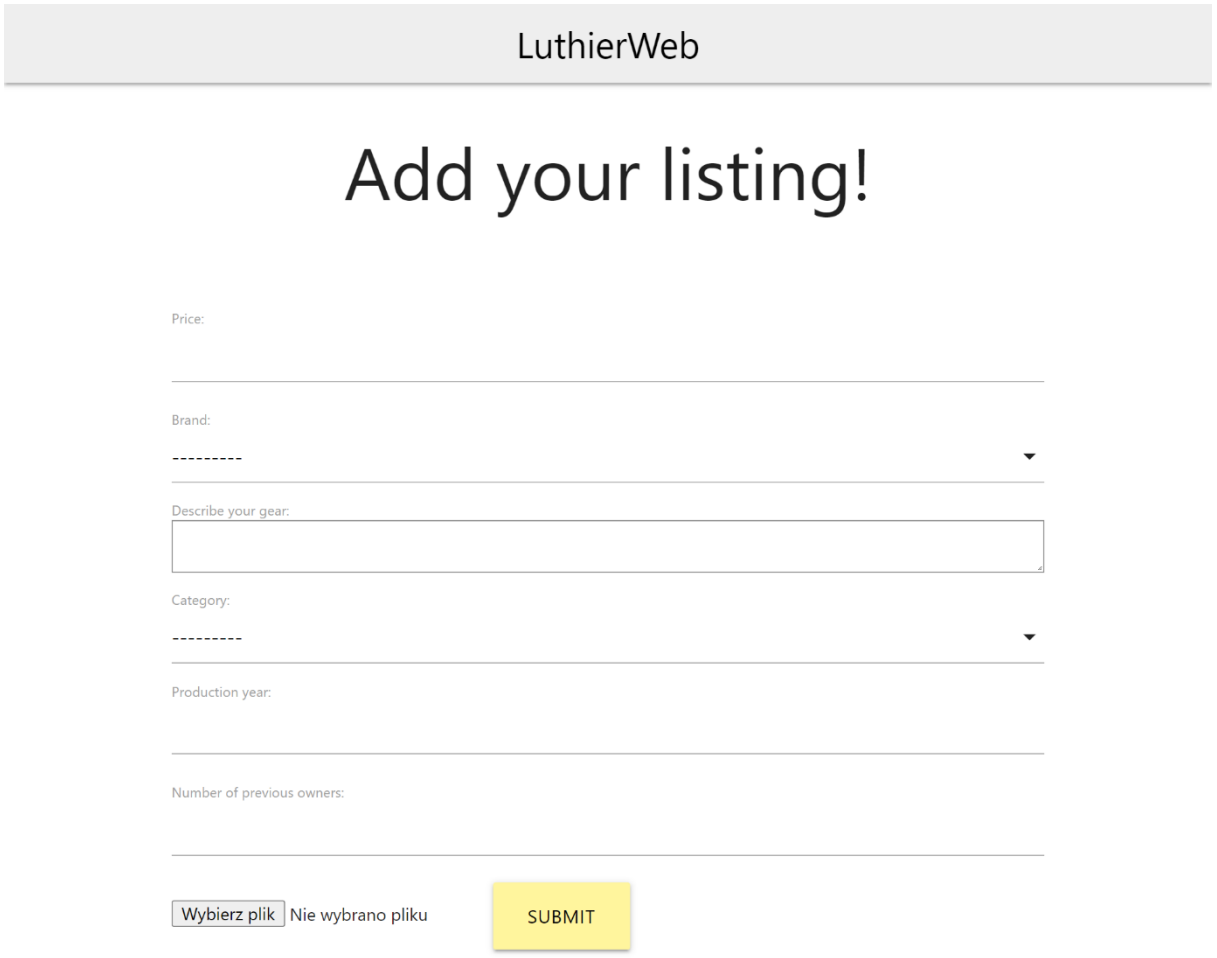
Informacje o wystawianym na sprzedaż przedmiocie przekazane przez użytkownika w formularzu (Rysunek 9.) zostają zapisane w bazie danych, a zdjęcia zapisywane zostają na serwerze (w bazie danych zapisana zostaje wyłącznie nazwa zdjęcia). Użytkownik przeglądać może ogłoszenia znajdujące się w bazie danych (Rysunek 7.), a po kliknięciu na wybrane ogłoszenie, przejść do jego szczegółów (Rysunek 8.). W tym miejscu użytkownik może kupić sprzęt naciskając na przycisk ‘Buy!’, który wywołuje procedurę składowaną *spChangeListingProductStatus,* zmieniającą status przedmiotu, co skutkuje wyłączeniem przedmiotu z listy ogłoszeń zaprezentowanych na Rysunku 7.

Dodatkowo, framework Django udostępnia funkcjonalność panelu administracyjnego (Rysunek 10.), gdzie, po zalogowaniu się jako administrator, możliwe są operacje dodawania, usuwania i modyfikacji ogłoszeń.

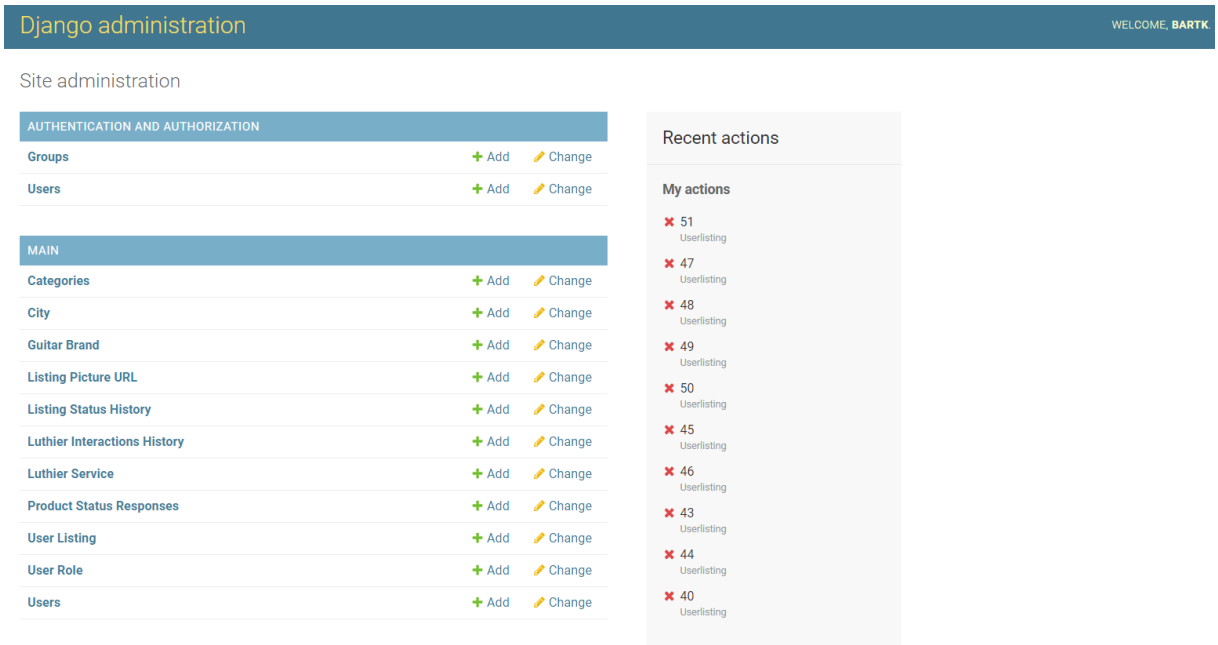


Rysunek 7 Fragment aplikacji; oferta komisu

Rysunek 8 Fragment aplikacji; ogłoszenie użytkownika



Rysunek 9 fragment aplikacji; formularz umożliwiający dodanie ogłoszenia do oferty komisu



Rysunek 10 Fragment aplikacji; panel administracyjny

8. Podsumowanie

Celem niniejeszej pracy było stworzenie bazy danych i elementów aplikacji bazodanowej obsługujacych komis sprzętu gitarowego i zakład usług lutniczych. Cele biznesowe związane z funkcjonowaniem systemu zoperacjonalizowane zostały w terminach wymagań funkcjonalnych i przypadków użycia, co umożliwiło ich późniejsze wdrożenie. Wykonane zostały fragmenty interfejsu użytkownika stanowiące podstawę dla implementacji aplikacji. Stworzona została baza danych odpowiadająca przedstawionym wymogom funkcjonalnym i organizująca dane związane z systemem w sposób przejrzysty. Funkcjonowanie bazy danych usprawione zostało poprzez implementację widoków, procedur i triggerów. Ostatnim etapem projektu było wykonanie elementów aplikacji bazodanowej. Zaimplementowane zostały elementy aplikacji webowej umożliwiające umieszczanie nowych ogłoszeń, wyświetlanie ich, możliwość zakupu przedmiotów i zarządzania ogłoszeniami, a więc podstawowe funkcjonalności związane z działaniem systemu.