Zaawansowane systemy baz danych

System eBOK

Projekt systemu bazodanowego

Niedziela, 15:15

Skład zespołu:

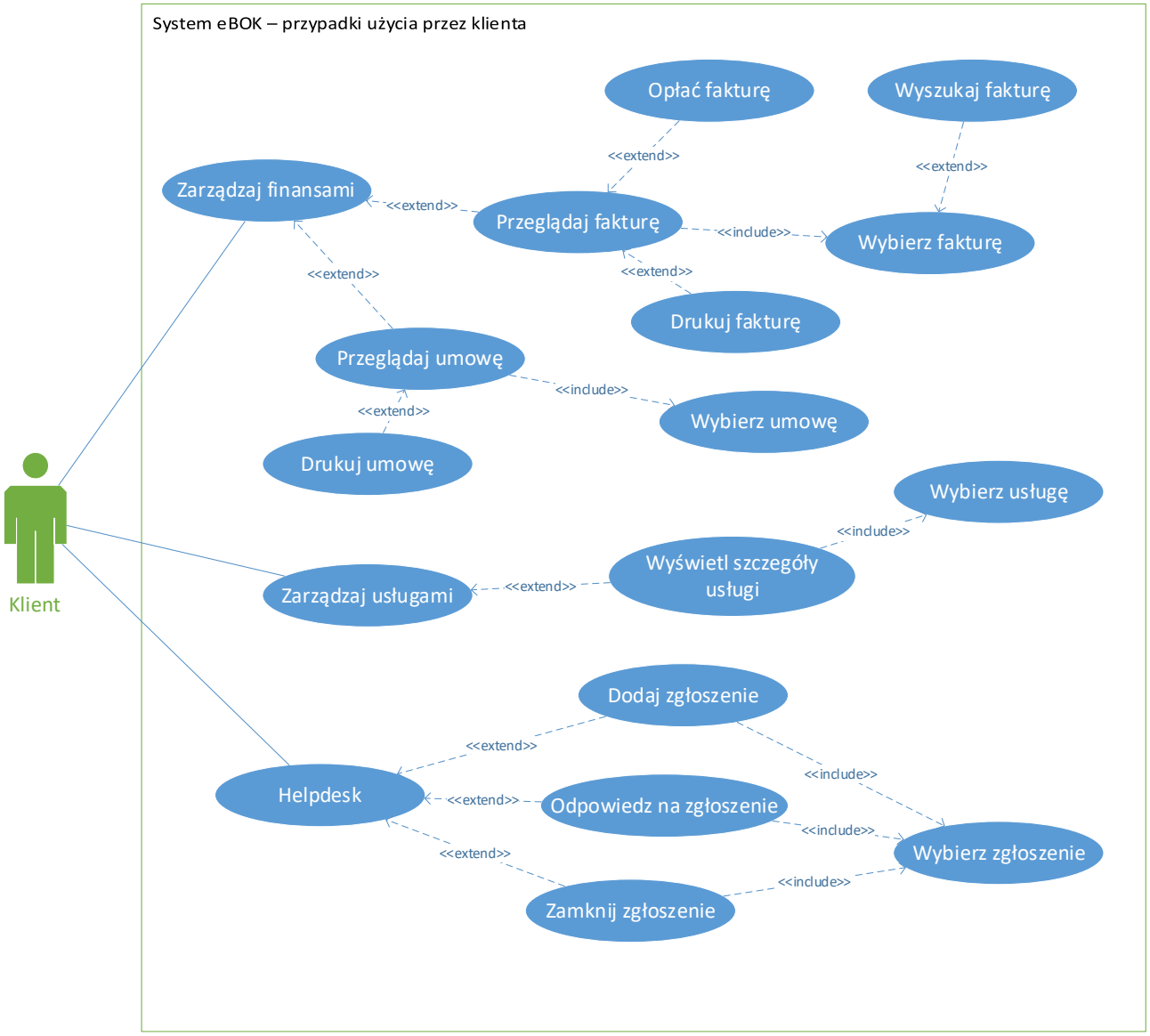
Dominik Cegiełka	224478	224478@edu.p.lodz.pl
Kamil Zarych	224546	224546@edu.p.lodz.pl

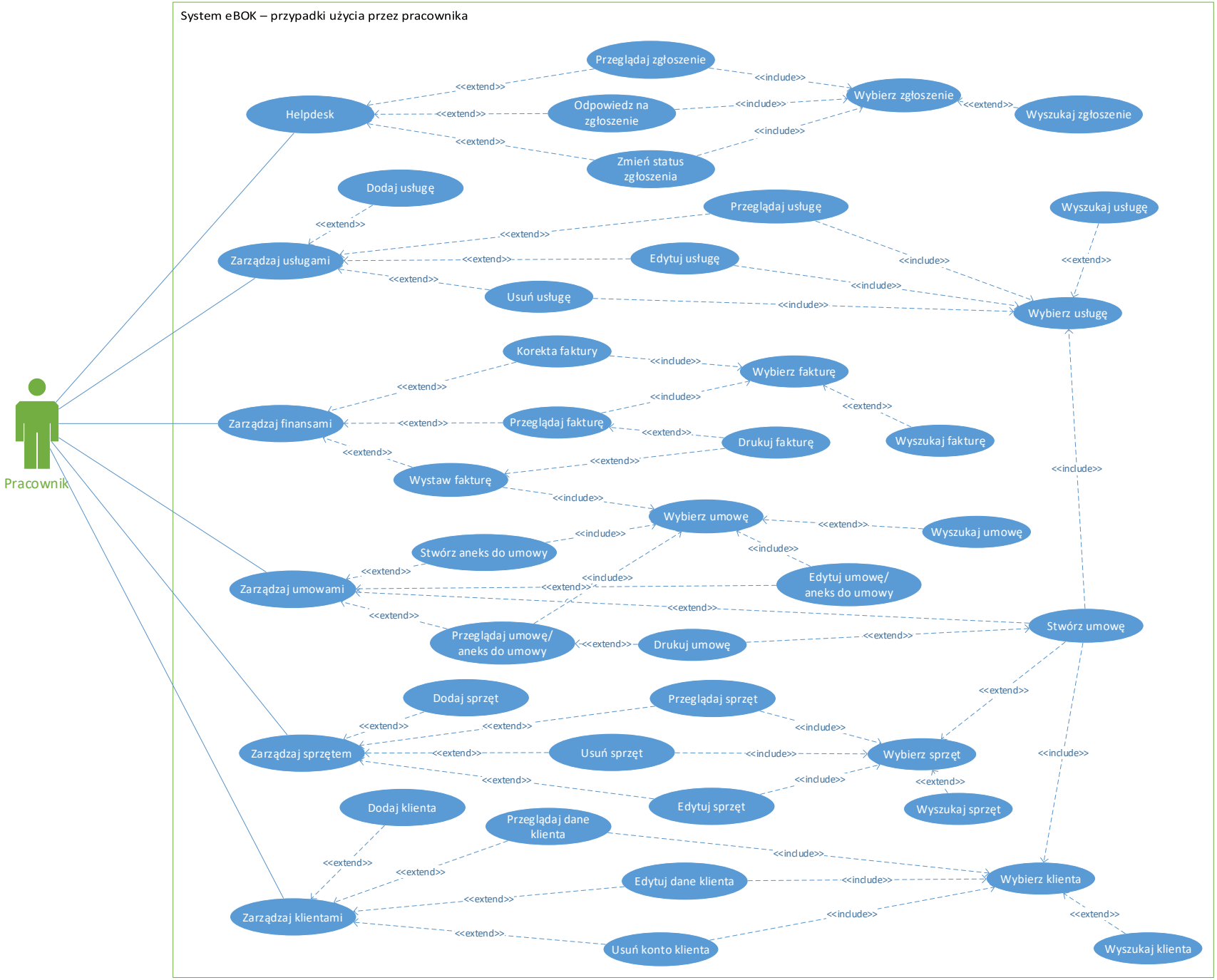
Spis treści

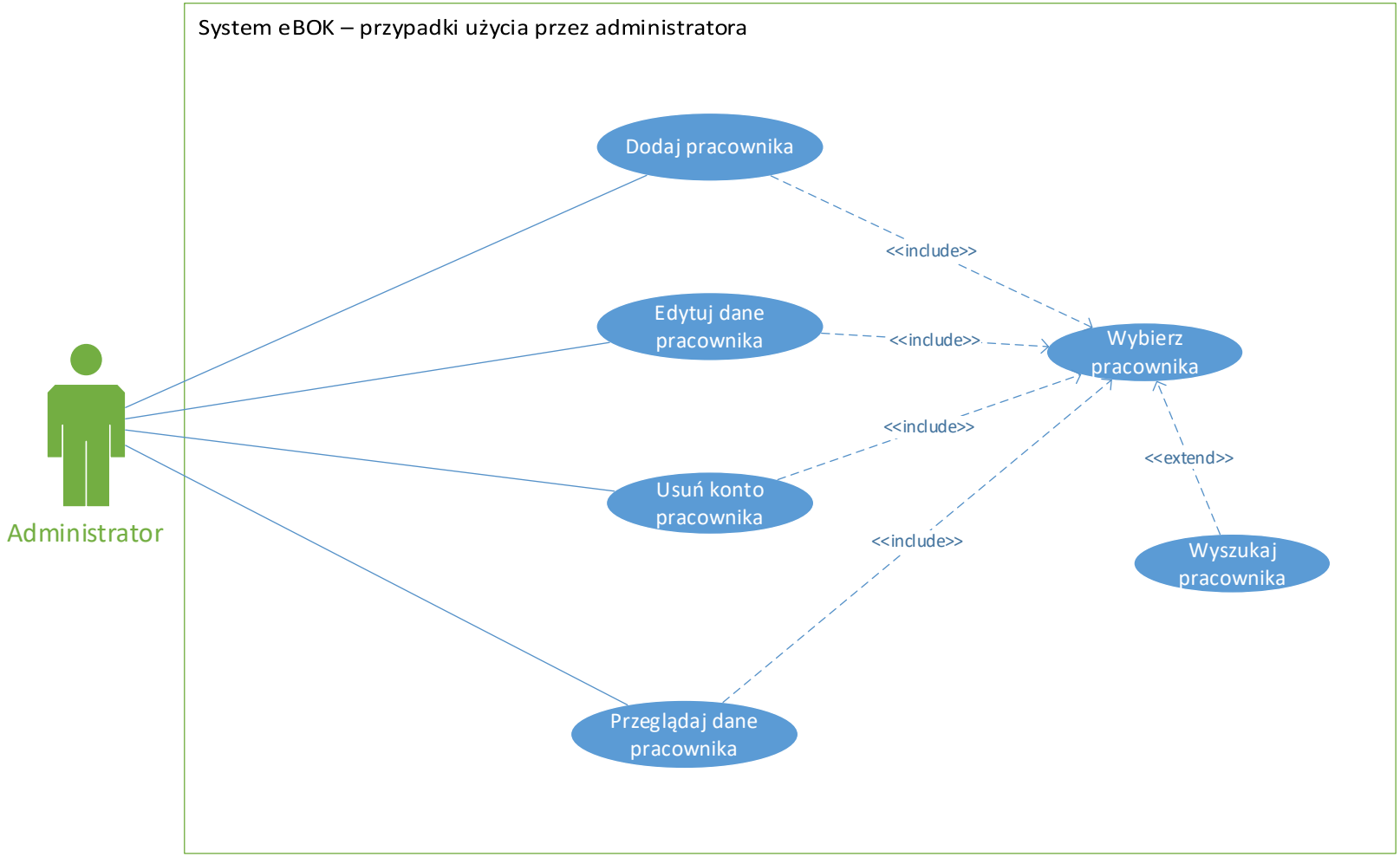
1. Cel projektu

Stworzenie systemu bazodanowego eBOK (Elektroniczne Biuro Obsługi Klienta) dla firmy dostarczającej usługi telekomunikacyjne (telewizja, internet, telefon), który umożliwi komunikację między przedsiębiorstwem a klientami. Wdrożenie takiego systemu redukuje koszty obsługi klientów dzięki bezpośredniemu dostępowi do niezbędnych informacji tj. faktury, płatności, umowy, helpdesk.

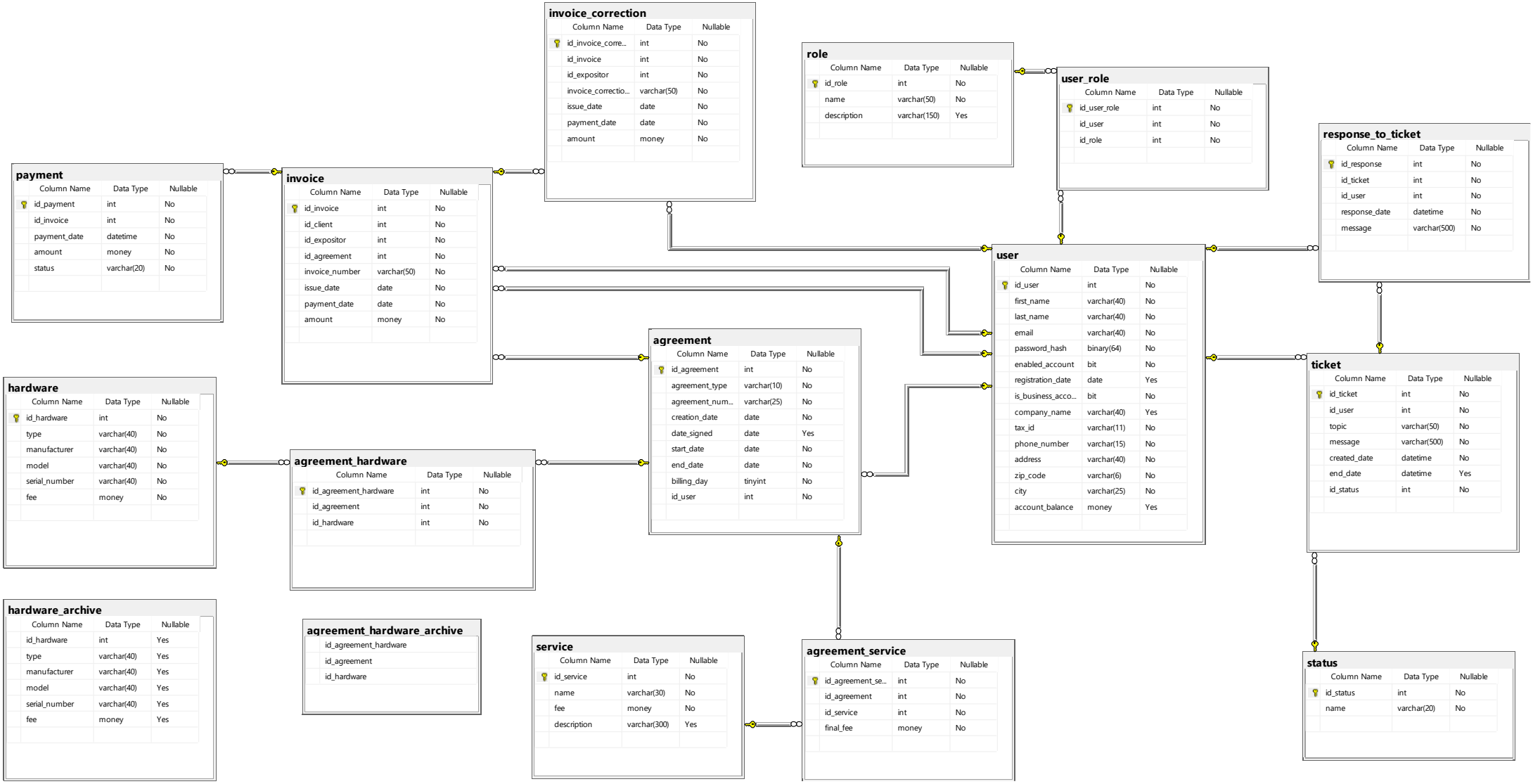
2. Diagramy przypadków użycia




System eBOK – przypadki użycia przez pracownika
Pracownik
Helpdesk
Przeglądaj zgłoszenie
Odpowiedz na zgłoszenie
Zmień status zgłoszenia
Wybierz zgłoszenie
Wyszukaj zgłoszenie
Zarządzaj usługami
Dodaj usługę
Przeglądaj usługę
Edytuj usługę
Usuń usługę
Wybierz usługę
Wyszukaj usługę
Zarządzaj finansami
Korekta faktury
Przeglądaj fakturę
Wystaw fakturę
Drukuj fakturę
Wybierz fakturę
Wyszukaj fakturę
Zarządzaj umowami
Wybierz umowę
Stwórz aneks do umowy
Edytuj umowę/ aneks do umowy
Przeglądaj umowę/ aneks do umowy
Drukuj umowę
Wyszukaj umowę
Stwórz umowę
Zarządzaj sprzętem
Dodaj sprzęt
Przeglądaj sprzęt
Usuń sprzęt
Edytuj sprzęt
Wybierz sprzęt
Wyszukaj sprzęt
Zarządzaj klientami
Dodaj klienta
Przeglądaj dane klienta
Edytuj dane klienta
Usuń konto klienta
Wybierz klienta
Wyszukaj klienta
<<extend>>
<<include>>


System eBOK – przypadki użycia przez administratora
Dodaj pracownika
Edytuj dane pracownika
Usuń konto pracownika
Przeglądaj dane pracownika
Wybierz pracownika
Wyszukaj pracownika
<<include>>
<<include>>
<<include>>
<<include>>
<<extend>>
Administrator

3. Schemat bazy danych



4. Lista modułów

a) Procedury

Zarządzanie użytkownikami:

- dodaj użytkownika ***(sp_add_user)***

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='P'
AND NAME='sp_add_user') DROP PROCEDURE sp_add_user
GO

CREATE PROCEDURE dbo.sp_add_user (
    @firstName VARCHAR(40),
    @lastName VARCHAR(40),
    @email VARCHAR(40),
    @password VARCHAR(30),
    @enabledAccount bit,
    @isBusinessAccount bit,
    @companyName VARCHAR(40),
    @taxId VARCHAR(11),
    @phoneNumber VARCHAR(15),
    @address VARCHAR(40),
    @zipCode VARCHAR(6),
    @city VARCHAR(25),
    @responseMessage VARCHAR(250) OUTPUT)
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
        IF (LEN(@password) < 8)
            RAISERROR('Password should contain at least 8 characters.', 16, 1);
        INSERT INTO dbo.[user] (first_name, last_name, email, password_hash,
        enabled_account, is_business_account, company_name,
        tax_id, phone_number, address, zip_code, city)
        VALUES (@firstName, @lastName, @email, HASHBYTES('SHA2_512',@password),
        @enabledAccount, @isBusinessAccount, @companyName,
        @taxId, @phoneNumber, @address, @zipCode, @city);

        SET @responseMessage = 'Successfully added user.'
    END TRY

    BEGIN CATCH
        SET @responseMessage=ERROR_MESSAGE()
    END CATCH
    SELECT @responseMessage
END
GO
```

- edytuj dane użytkownika ***(sp_edit_user)***

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='P'
AND NAME='sp_edit_user') DROP PROCEDURE sp_edit_user
GO

CREATE PROCEDURE dbo.sp_edit_user (
    @userId int,
    @firstName VARCHAR(40) = NULL,
    @lastName VARCHAR(40) = NULL,
    @email VARCHAR(40) = NULL,
    @passwordHash binary(64) = NULL,
    @enabledAccount VARCHAR(40) = NULL,
    @isBusinessAccount bit = NULL,
    @companyName VARCHAR(40) = NULL,
    @taxId VARCHAR(11) = NULL,
    @phoneNumber VARCHAR(15) = NULL,
    @address VARCHAR(40) = NULL,
    @zip_code VARCHAR(6) = NULL,
    @city VARCHAR(25) = NULL,
    @responseMessage VARCHAR(250) OUTPUT)
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
        UPDATE dbo.[user]
        SET first_name=ISNULL(@firstName, first_name),
            last_name=ISNULL(@lastName, last_name),
            email=ISNULL(@email, email),
            password_hash=ISNULL(HASHBYTES('SHA2_512',@passwordHash),
            password_hash),
            enabled_account=ISNULL(@enabledAccount, enabled_account),
            is_business_account=ISNULL(@isBusinessAccount,
            is_business_account),
            company_name=ISNULL(@companyName, company_name),
            tax_id=ISNULL(@taxId, tax_id),
            phone_number=ISNULL(@phoneNumber, phone_number),
            address=ISNULL(@address, address),
            zip_code=ISNULL(@zip_code, zip_code),
            city=ISNULL(@city, city)
        WHERE id_user = @userId
    END TRY

    BEGIN CATCH
        SET @responseMessage=ERROR_MESSAGE()
    END CATCH
    SELECT @responseMessage
END
GO
```

- usuń użytkownika *(sp_delete_user)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='P'
AND NAME='sp_delete_user') DROP PROCEDURE sp_delete_user
GO

CREATE PROCEDURE dbo.sp_delete_user (
    @userId int,
    @responseMessage VARCHAR(250) OUTPUT)

AS
BEGIN
    SET NOCOUNT ON
    IF EXISTS (SELECT * FROM [user] WHERE id_user = @userId)
        BEGIN
            BEGIN TRY
                DELETE FROM [user]
                WHERE id_user = @userId
                SET @responseMessage = 'Successfully deleted user with ID
                = ' + CAST(@userId AS VARCHAR(10));
            END TRY
            BEGIN CATCH
                SET @responseMessage = ERROR_MESSAGE()
            END CATCH
        END
    ELSE
        BEGIN
            SET @responseMessage = 'There is no user with the given ID'
        END
    SELECT @responseMessage
END
GO
```

Zarządzanie usługami:

- dodaj usługę *(sp_add_service)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='P'
AND NAME='sp_add_service') DROP PROCEDURE sp_add_service
GO

CREATE PROCEDURE dbo.sp_add_service (
    @name VARCHAR(30),
    @fee money,
    @description VARCHAR(300),
    @responseMessage VARCHAR(250) OUTPUT)
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
        INSERT INTO dbo.service(name, fee, description)
        VALUES (@name, @fee, @description)

        SET @responseMessage = 'Successfully added service.'
    END TRY

    BEGIN CATCH
        SET @responseMessage=ERROR_MESSAGE()
    END CATCH
    SELECT @responseMessage
END
GO
```

- edytuj usługę *(sp_edit_service)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='P'
AND NAME='sp_edit_service') DROP PROCEDURE sp_edit_service
GO
CREATE PROCEDURE dbo.sp_edit_service (
    @serviceId int,
    @name VARCHAR(30) = NULL,
    @fee money = NULL,
    @description VARCHAR(300) = NULL,
    @responseMessage VARCHAR(250) OUTPUT)
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
        UPDATE dbo.service
        SET name=ISNULL(@name, name),
            fee=ISNULL(@fee, fee),
            description=ISNULL(@description, description)
        WHERE id_service = @serviceId
    END TRY

    BEGIN CATCH
        SET @responseMessage=ERROR_MESSAGE()
    END CATCH
    SELECT @responseMessage
END
GO
```

- usuń usługę *(sp_delete_service)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='P'
AND NAME='sp_delete_service') DROP PROCEDURE sp_delete_service
GO

CREATE PROCEDURE dbo.sp_delete_service (
    @name VARCHAR(30),
    @responseMessage VARCHAR(250) OUTPUT)
AS
BEGIN
    SET NOCOUNT ON
    DECLARE @ifExists int = (select count(*) from service where name =
@name)

    IF @name IS NULL OR @ifExists = 0
        BEGIN
            SET @responseMessage = 'There is no service with given name';
        END
    ELSE
    BEGIN TRY
        DELETE FROM service
        WHERE name = @name;
    END TRY

    BEGIN CATCH
        SET @responseMessage=ERROR_MESSAGE()
    END CATCH
    SELECT @responseMessage
END
GO
```

Zarządzanie sprzętem:

- dodaj sprzęt *(sp_add_hardware)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='P'
AND NAME='sp_add_hardware') DROP PROCEDURE sp_add_hardware
GO

CREATE PROCEDURE dbo.sp_add_hardware (
    @type VARCHAR(40),
    @manufacturer VARCHAR(40),
    @model VARCHAR(40),
    @serialNumber VARCHAR(40),
    @fee money,
    @responseMessage VARCHAR(250) OUTPUT)
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
        INSERT INTO dbo.hardware(type, manufacturer, model, serial_number,
        fee)
        VALUES (@type, @manufacturer, @model, @serialNumber,@fee)

        SET @responseMessage = 'Successfully added hardware.'
    END TRY

    BEGIN CATCH
        SET @responseMessage=ERROR_MESSAGE()
    END CATCH
    SELECT @responseMessage
END
GO
```

- edytuj sprzęt *(sp_edit_hardware)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='P'
AND NAME='sp_edit_hardware') DROP PROCEDURE sp_edit_hardware
GO

CREATE PROCEDURE dbo.sp_edit_hardware (
    @hardwareId int,
    @type VARCHAR(40) = NULL,
    @manufacturer VARCHAR(40) = NULL,
    @model VARCHAR(40) = NULL,
    @serialNumber VARCHAR(40) = NULL,
    @fee money = NULL,
    @responseMessage VARCHAR(250) OUTPUT)
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
        UPDATE dbo.hardware
        SET type=ISNULL(@type, type),
            manufacturer=ISNULL(@manufacturer, manufacturer),
            model=ISNULL(@model, model),
            serial_number=ISNULL(@serialNumber, serial_number),
            fee=ISNULL(@fee, fee)
        WHERE id_hardware = @hardwareId
    END TRY

    BEGIN CATCH
        SET @responseMessage=ERROR_MESSAGE()
```

```
    END CATCH
    SELECT @responseMessage
END
GO
```

- usuń sprzęt *(sp_delete_hardware)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='P'
AND NAME='sp_delete_hardware') DROP PROCEDURE sp_delete_hardware
GO

CREATE PROCEDURE dbo.sp_delete_hardware (
    @idHardware int,
    @responseMessage VARCHAR(250) OUTPUT)

AS
BEGIN
    SET NOCOUNT ON

    IF EXISTS (SELECT * FROM hardware WHERE id_hardware = @idHardware)
        BEGIN
            DELETE FROM hardware
            WHERE id_hardware = @idHardware
            SET @responseMessage = 'Successfully deleted hardware with ID
            = ' + CAST(@idHardware AS VARCHAR(10));
        END
    ELSE
        BEGIN
            SET @responseMessage = 'There is no hardware with the given
            ID'
        END
    SELECT @responseMessage
END
GO
```

Zarządzanie finansami:

- dodaj fakturę *(sp_add_invoice)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='P'
AND NAME='sp_add_invoice') DROP PROCEDURE sp_add_invoice
GO

CREATE PROCEDURE dbo.sp_add_invoice (
    @clientId int,
    @expositorId int,
    @agreementId int,
    @invoiceNumber VARCHAR(50),
    @issueDate date,
    @paymentDate date,
    @amount money,
    @responseMessage VARCHAR(250) OUTPUT)
AS
BEGIN
    SET NOCOUNT ON

    IF EXISTS (SELECT * FROM agreement WHERE id_agreement = @agreementId)
    BEGIN
        BEGIN TRY
            INSERT INTO invoice
            VALUES (@clientId, @expositorId, @agreementId, @invoiceNumber,
            @issueDate, @paymentDate, @amount);
```

```
            SET @responseMessage = 'Successfully added invoice.'
        END TRY
        BEGIN CATCH
            SET @responseMessage=ERROR_MESSAGE()
        END CATCH
    END
    ELSE
        BEGIN
            SET @responseMessage = 'There is no agreement with the given
            ID.';
        END
    SELECT @responseMessage
END
GO
```

- dodaj korektę faktury *(sp_add_invoice_correction)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='P'
AND NAME='sp_add_invoice_correction') DROP PROCEDURE
sp_add_invoice_correction
GO

CREATE PROCEDURE dbo.sp_add_invoice_correction (
    @invoiceId int,
    @expositorId int,
    @invoiceCorrectionNumber VARCHAR(50),
    @issueDate date,
    @paymentDate date,
    @amount money,
    @responseMessage VARCHAR(250) OUTPUT)
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
        IF NOT EXISTS(SELECT * FROM invoice WHERE id_invoice = @invoiceId)
            RAISERROR('There is no invoice with the given ID', 16, 1);
        INSERT INTO invoice_correction
        VALUES (@invoiceId, @expositorId, @invoiceCorrectionNumber,
        @issueDate, @paymentDate, @amount);

        SET @responseMessage = 'Successfully added invoice correction.'
    END TRY

    BEGIN CATCH
        SET @responseMessage=ERROR_MESSAGE()
    END CATCH
    SELECT @responseMessage
END
GO
```

- usuń fakturę *(sp_delete_invoice)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='P'
AND NAME='sp_delete_invoice') DROP PROCEDURE sp_delete_invoice
GO

CREATE PROCEDURE dbo.sp_delete_invoice (
    @invoiceId int,
    @responseMessage VARCHAR(250) OUTPUT)

AS
BEGIN
```

```
    SET NOCOUNT ON

    IF EXISTS (SELECT * FROM invoice WHERE id_invoice = @invoiceId)
        BEGIN
            BEGIN TRY
                DELETE FROM invoice
                WHERE id_invoice =  @invoiceId
                SET @responseMessage = 'Successfully deleted invoice with
                ID = ' + CAST(@invoiceId AS VARCHAR(10));
            END TRY
            BEGIN CATCH
                SET @responseMessage = ERROR_MESSAGE()
            END CATCH
        END
    ELSE
        BEGIN
            SET @responseMessage = 'There is no invoice with the given ID'
        END
    SELECT @responseMessage
END
GO
```

Zarządzanie ticketami:

- dodaj ticket *(sp_add_ticket)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='P'
AND NAME='sp_add_ticket') DROP PROCEDURE sp_add_ticket
GO

CREATE PROCEDURE dbo.sp_add_ticket (
    @idUser int,
    @topic VARCHAR(50),
    @message VARCHAR(500),
    @createdDate datetime,
    @idStatus int,
    @responseMessage VARCHAR(250) OUTPUT)
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
        INSERT INTO dbo.ticket (id_user, topic, message, created_date,
        id_status)
        VALUES (@idUser, @topic, @message, @createdDate, @idStatus)

        SET @responseMessage = 'Successfully added ticket.'

    END TRY
    BEGIN CATCH
        SET @responseMessage=ERROR_MESSAGE()
    END CATCH
    SELECT @responseMessage
END
GO
```

- dodaj odpowiedź *(sp_add_response_to_ticket)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='P'
AND NAME='sp_add_response_to_ticket') DROP PROCEDURE
sp_add_response_to_ticket
GO
```

```
CREATE PROCEDURE dbo.sp_add_response_to_ticket (
    @idTicket int,
    @idUser int,
    @responseDate datetime,
    @message VARCHAR(500),
    @responseMessage VARCHAR(250) OUTPUT)
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
        INSERT INTO dbo.response_to_ticket (id_ticket, id_user,
        response_date, message)
        VALUES (@idTicket, @idUser, @responseDate, @message)

        SET @responseMessage = 'Successfully added response to ticket.'

    END TRY
    BEGIN CATCH
        SET @responseMessage=ERROR_MESSAGE()
    END CATCH
    SELECT @responseMessage
END
GO
```

b) Funkcje

- Funkcja sprawdzająca czy określony sprzęt jest dostępny *(fn_check_hardware_available)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='FN'
AND NAME='fn_check_hardware_available') DROP FUNCTION
fn_check_hardware_available
GO

CREATE FUNCTION fn_check_hardware_available(@id_hardware int)
RETURNS bit AS
BEGIN
    IF NOT EXISTS(SELECT * FROM hardware WHERE id_hardware = @id_hardware)
        RETURN 0
    DECLARE @num int = (SELECT COUNT(ah.id_hardware) FROM
        agreement_hardware ah JOIN agreement ag ON ah.id_agreement =
        ag.id_agreement WHERE ag.end_date > GETDATE() AND ah.id_hardware =
        @id_hardware)
    IF(@num > 0)
        RETURN 0
    RETURN 1
END
GO
```

- Funkcja sprawdzająca czy dany użytkownik posiada określoną rolę *(fn_check_user_role)*

```
IF OBJECT_ID(N'dbo.fn_check_user_role', N'FN') IS NOT NULL
    DROP FUNCTION fn_check_user_role;
GO

CREATE FUNCTION dbo.fn_check_user_role(@idUser int, @role varchar(30))
RETURNS BIT
AS
BEGIN
    DECLARE @returnValue int;
    SELECT @returnValue = COUNT(*) FROM [user] u JOIN user_role ur ON
      u.id_user = ur.id_user JOIN role r ON ur.id_role = r.id_role
```

```
      WHERE u.id_user = @idUser AND r.name = @role;
    IF @returnValue > 0
        RETURN 1

    RETURN 0
END
GO
```

- Funkcja sprawdzająca, ile razy określony klient zapłacił faktury po terminie *(fn_check_invoices_paid_after_deadline)*

```
IF OBJECT_ID(N'dbo.fn_check_invoices_paid_after_deadline', N'FN') IS NOT
NULL
    DROP FUNCTION fn_check_invoices_paid_after_deadline;
GO

CREATE FUNCTION dbo.fn_check_invoices_paid_after_deadline(@idUser int)
RETURNS INT
AS
BEGIN
    DECLARE @returnValue int
    SELECT @returnValue = COUNT(i.id_invoice) FROM dbo.invoice i join
      dbo.payment p on i.id_invoice = p.id_invoice
    WHERE i.id_client = @idUser AND p.payment_date > i.payment_date AND
      p.status = 'completed'
    RETURN @returnValue
END
GO
```

- Funkcja sprawdzająca czy określona faktura została opłacona. *(fn_check_invoice_payment_status)*

```
IF OBJECT_ID(N'dbo.fn_check_invoice_payment_status', N'FN') IS NOT NULL
    DROP FUNCTION fn_check_invoice_payment_status;
GO

CREATE FUNCTION dbo.fn_check_invoice_payment_status(@idInovice
varchar(20))
RETURNS BIT
AS
BEGIN
    DECLARE @returnValue int;
    SELECT @returnValue = COUNT(*) FROM payment p JOIN invoice i ON
      p.id_invoice = i.id_invoice WHERE p.status = 'completed' AND
      i.invoice_number = @idInovice
    IF @returnValue > 0
        RETURN 1

    RETURN 0
END
GO
```

- Funkcja sprawdzająca czy określone zgłoszenie zostało zamknięte *(fn_check_ticket_status)*

```
IF OBJECT_ID(N'dbo.fn_check_ticket_status', N'FN') IS NOT NULL
    DROP FUNCTION fn_check_ticket_status
GO

CREATE FUNCTION dbo.fn_check_ticket_status(@ticketId int)
RETURNS BIT
AS
```

```
BEGIN
    DECLARE @returnVaule int;
    SELECT @returnVaule = COUNT(t.id_ticket) FROM ticket t JOIN status s
ON t.id_status = s.id_status WHERE t.id_ticket = @ticketId AND s.name =
'closed'

    IF @returnVaule > 0
        RETURN 1

    RETURN 0
END
GO
```

- Funkcja sprawdzająca, ile mamy podpisanych umów w określonym zakresie dat (startDate, endDate) *(fn_check_number_of_agreements_signed_in_given_period)*

```
IF OBJECT_ID(N'dbo.fn_check_number_of_agreements_signed_in_given_period',
N'FN') IS NOT NULL
    DROP FUNCTION
dbo.fn_check_number_of_agreements_signed_in_given_period;
GO

CREATE FUNCTION
dbo.fn_check_number_of_agreements_signed_in_given_period(@startDate
datetime, @endDate datetime)
RETURNS VARCHAR(30)
AS
BEGIN
    IF(@endDate < @startDate)
        RETURN 'Invalid date range!';
    ELSE
        DECLARE @quantity int;
        Set  @quantity = (SELECT COUNT(id_agreement) AS ile FROM agreement
WHERE date_signed BETWEEN @startDate AND @endDate )

    RETURN  CAST(@quantity as VARCHAR(30))

END
GO
```

- Funkcja sprawdzająca czy jest nadpłata/niedopłata faktury lub czy bilans jest równy 0 *(fn_check_invoice_payments)*

```
IF OBJECT_ID(N'dbo.fn_check_invoice_payments', N'FN') IS NOT NULL
    DROP FUNCTION dbo.fn_check_invoice_payments;
GO

CREATE FUNCTION dbo.fn_check_invoice_payments(@invoiceNumber VARCHAR(20))
RETURNS VARCHAR(100)
AS
BEGIN
    DECLARE @balance mONey
    DECLARE @returnMess VARCHAR(100)
    SELECT @balance = i.amount - SUM(p.amount) FROM payment p JOIN invoice
        i ON p.id_invoice = i.id_invoice WHERE i.invoice_number =
        @invoiceNumber AND p.status = 'completed' GROUP BY i.amount

    IF (@balance > 0)
        SET @returnMess = 'Invoice no. ' + @invoiceNumber + ' wAS not
        completely paid. The remaining amount to be paid: ' +
        CAST(@balance AS VARCHAR(10))
    ELSE IF (@balance < 0)
```

```
        SET @returnMess = 'Overpayment wAS made to the invoice no. ' +
        @invoiceNumber + ' . The overpayment amount: ' +
        CAST(ABS(@balance) AS VARCHAR(10))
    ELSE
        SET @returnMess = 'The invoice wAS completely paid.'

    RETURN @returnMess
END
GO
```

c) Wyzwalacze

- Ustawienie daty rejestracji użytkownika (user) po dodaniu go do bazy *(tr_set_user_registration_date)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='TR'
AND NAME='tr_add_user_registration_date') DROP TRIGGER
tr_add_user_registration_date
GO

CREATE TRIGGER tr_add_user_registration_date
ON dbo.[user]
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON
    UPDATE dbo.[user]
    SET registration_date = GETDATE()
    FROM dbo.[user] u JOIN inserted i ON i.id_user = u.id_user

END
GO
```

- Aktualizacja bilansu (account_balance) klienta po dodaniu faktury *(tr_update_account_balance_after_add_invoice)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='TR'
AND NAME='tr_update_account_balance_after_add_invoice') DROP TRIGGER
tr_update_account_balance_after_add_invoice
GO

CREATE TRIGGER tr_update_account_balance_after_add_invoice
ON dbo.invoice
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON
    UPDATE dbo.[user]
    SET account_balance = account_balance - i.amount
    FROM dbo.[user] ud JOIN inserted i ON i.id_client = ud.id_user
END
GO
```

- *Aktualizacja bilansu (account_balance) klienta po dodaniu korekty faktury* *(tr_update_account_balance_after_add_invoice_correction)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='TR'
AND NAME='tr_update_account_balance_after_add_invoice_correction') DROP
TRIGGER tr_update_account_balance_after_add_invoice_correction
GO
```

```
CREATE TRIGGER tr_update_account_balance_after_add_invoice_correction
ON dbo.invoice_correction
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON
    DECLARE @id_client int = (SELECT inv.id_client FROM dbo.invoice inv
JOIN inserted i ON inv.id_invoice = i.id_invoice)

    UPDATE dbo.[user]
    SET account_balance = account_balance + (SELECT inv.amount FROM
dbo.invoice inv JOIN inserted i ON inv.id_invoice = i.id_invoice)
    FROM dbo.[user]
    WHERE id_user = @id_client

    UPDATE dbo.[user]
    set account_balance = account_balance - i.amount
    FROM dbo.[user] ud JOIN inserted i ON @id_client = ud.id_user
END
GO
```

- Aktualizacja bilansu (account_balance) klienta po aktualizacji statusu płatności na status 'completed' *(tr_update_account_balance_after_update_payment)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='TR'
AND NAME='tr_update_account_balance_after_update_payment') DROP TRIGGER
tr_update_account_balance_after_update_payment
GO

CREATE TRIGGER tr_update_account_balance_after_update_payment
ON dbo.payment
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON
    UPDATE dbo.[user]
    SET account_balance = account_balance + i.amount
    FROM dbo.[user] ud JOIN dbo.invoice inv ON ud.id_user = inv.id_client
JOIN inserted i ON i.id_invoice = inv.id_invoice
    WHERE i.status = 'completed'
END
GO
```

- Ustawienie daty zamknięcia zgłoszenia na aktualną datę po zmianie statusu zgłoszenia na zamknięty *(tr_update_ticket_status_set_end_date)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='TR'
AND NAME='tr_update_ticket_status_set_end_date') DROP TRIGGER
tr_update_ticket_status_set_end_date
GO

CREATE TRIGGER tr_update_ticket_status_set_end_date
ON dbo.ticket
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON
    IF ((SELECT s.name FROM status s JOIN inserted i ON s.id_status =
i.id_status) = 'closed')
        UPDATE dbo.ticket
```

```
        SET end_date = GETDATE()
        FROM dbo.ticket t JOIN status s ON t.id_status = s.id_status JOIN
inserted i ON i.id_ticket = t.id_ticket
    ELSE
        UPDATE dbo.ticket
        SET end_date = NULL
        FROM dbo.ticket t JOIN status s ON t.id_status = s.id_status JOIN
inserted i ON i.id_ticket = t.id_ticket
END
GO
```

- Podczas usuwania sprzętu, usuwane są powiązania danego sprzętu z umowami (agreement_hardware), następnie sprzęt przenoszony jest do archiwum (hardware_archive) i na koniec usuwany jest sprzęt ***(tr_delete_hardware)***

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='TR'
AND NAME='tr_delete_hardware') DROP TRIGGER tr_delete_hardware
GO

CREATE TRIGGER tr_delete_hardware
ON dbo.hardware
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON

    DECLARE to_delete_hardware CURSOR FOR SELECT * from deleted
    DECLARE @id_hardware int, @type VARCHAR(40),
            @manufacturer VARCHAR(40), @model VARCHAR(40),
            @serial_number VARCHAR(40),@fee money
    open to_delete_hardware
    fetch next from to_delete_hardware into @id_hardware, @type,
      @manufacturer, @model, @serial_number, @fee
    IF(@@FETCH_STATUS = -1)
    BEGIN
        RAISERROR('No hardware to remove with the given ID. id_hardware =
        ', 11, 1)
        PRINT @@ERROR
    END
    ELSE
    BEGIN
        WHILE(@@FETCH_STATUS=0)
        BEGIN
            INSERT INTO dbo.hardware_archive(id_hardware, type,
                        manufacturer, model, serial_number, fee)
            VALUES(@id_hardware, @type, @manufacturer, @model,
                   @serial_number, @fee)

            DELETE FROM dbo.agreement_hardware WHERE id_hardware =
                   @id_hardware

            DELETE FROM dbo.hardware WHERE id_hardware = @id_hardware
            fetch next from to_delete_hardware into @id_hardware, @type,
                   @manufacturer, @model, @serial_number, @fee
        END
    END
    CLOSE to_delete_hardware
    deallocate to_delete_hardware
END
GO
```

- Przenoszenie powiązań umów ze sprzętem do tabeli (agreement_hardware_archive) w przypadku usunięcia powiązania. *(tr_move_agreement_hardware_to_archive)*

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE='TR'
AND NAME='tr_move_agreement_hardware_to_archive') DROP TRIGGER
tr_move_agreement_hardware_to_archive
GO

CREATE TRIGGER tr_move_agreement_hardware_to_archive
ON dbo.agreement_hardware
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON

    DECLARE to_delete_agreement_hardware CURSOR FOR SELECT * from deleted
    DECLARE @id_agreement_hardware int, @id_agreement int, @id_hardware
int
    open to_delete_agreement_hardware
    fetch next from to_delete_agreement_hardware into
@id_agreement_hardware, @id_agreement, @id_hardware
    IF(@@FETCH_STATUS = -1)
    BEGIN
        RAISERROR('No hardware_agreement to remove with the given ID.
id_hardware = ', 11, 1)
        PRINT @@ERROR
    END
    ELSE
    BEGIN
        WHILE(@@FETCH_STATUS=0)
        BEGIN
            INSERT INTO
dbo.agreement_hardware_archive(id_agreement_hardware, id_agreement,
id_hardware)
            VALUES(@id_agreement_hardware, @id_agreement, @id_hardware)

            DELETE FROM dbo.agreement_hardware WHERE id_agreement_hardware
= @id_agreement_hardware
            fetch next from to_delete_agreement_hardware into
@id_agreement_hardware, @id_agreement, @id_hardware
        END
    END
    CLOSE to_delete_agreement_hardware
    deallocate to_delete_agreement_hardware
END
GO
```

d) Widoki

- Widok faktur z wybranymi danymi klienta oraz informacją czy dana faktura została opłacona (invoices_with_client_and_payment_info_view)

```
IF OBJECT_ID(N'dbo.invoices_with_client_and_payment_info_view', N'V') IS
NOT NULL
DROP VIEW invoices_with_client_and_payment_info_view
GO

CREATE VIEW dbo.invoices_with_client_and_payment_info_view AS
```

```
    SELECT i.id_invoice, i.invoice_number, u.first_name + ' ' +
           u.last_name AS 'Client',
           ux.first_name + ' ' + ux.last_name AS 'Expositor',
           i.issue_date, i.payment_date, amount,
           dbo.fn_check_invoice_payments(i.id_invoice) AS 'Invoice Payment
            Status'
    FROM dbo.invoice i JOIN [user] u ON i.id_client =
         u.id_user JOIN [user] ux ON i.id_expositor = ux.id_user
GO

SELECT * from invoices_with_client_and_payment_info_view
```

- Widok sprzętów z informacją o kliencie, który aktualnie posiada dany sprzęt (hardware_with_client_info_view)

```
IF OBJECT_ID(N'dbo.hardware_with_client_info_view', N'V') IS NOT NULL
DROP VIEW hardware_with_client_info_view
GO

CREATE VIEW dbo.hardware_with_client_info_view AS
    SELECT  model, serial_number, u.first_name + ' ' + u.last_name AS
        'Client',  u.address + '; ' + u.zip_code + '; ' +  u.city AS
        'Address', u.phone_number
    FROM hardware h JOIN agreement_hardware ah ON h.id_hardware =
         ah.id_hardware JOIN agreement a ON ah.id_agreement =
         a.id_agreement JOIN [user] u ON a.id_user = u.id_user
    WHERE a.start_date <= GETDATE() AND a.end_date >= GETDATE()

GO

SELECT * FROM hardware_with_client_info_view
```

- Widok ticketów wraz z wszystkimi odpowiedziami (tickets_with_all_responses)

```
IF OBJECT_ID(N'dbo.tickets_with_all_responses', N'V') IS NOT NULL
DROP VIEW tickets_with_all_responses
GO

CREATE VIEW dbo.tickets_with_all_responses AS
    SELECT t.message AS 'ticket', t.created_date, rtt.message as
         'response', rtt.response_date
    FROM response_to_ticket rtt join ticket t ON rtt.id_ticket =
         t.id_ticket
GO

SELECT * FROM tickets_with_all_responses
```

- Widok klientów wraz z informacją odnośnie ich bilansu konta oraz sumie wydanych pieniędzy na zapłacenie faktur (clients_with_info_about_account_balance_and_total_paid_amount)

```
IF
OBJECT_ID(N'dbo.clients_with_info_about_account_balance_and_total_paid_amo
unt', N'V') IS NOT NULL
DROP VIEW clients_with_info_about_account_balance_and_total_paid_amount
GO

CREATE VIEW
dbo.clients_with_info_about_account_balance_and_total_paid_amount AS

    SELECT u.first_name + ' ' + u.last_name AS 'Client',u.account_balance,
    SUM(p.amount) AS 'In total paid'
```

```
   FROM [user] u JOIN invoice i ON u.id_user = i.id_client JOIN payment p
        ON i.id_invoice = p.id_invoice
   WHERE dbo.fn_check_user_role(u.id_user, 'klient') = 1 GROUP BY
        u.first_name, u.last_name, u.account_balance

GO

SELECT * FROM
dbo.clients_with_info_about_account_balance_and_total_paid_amount
```