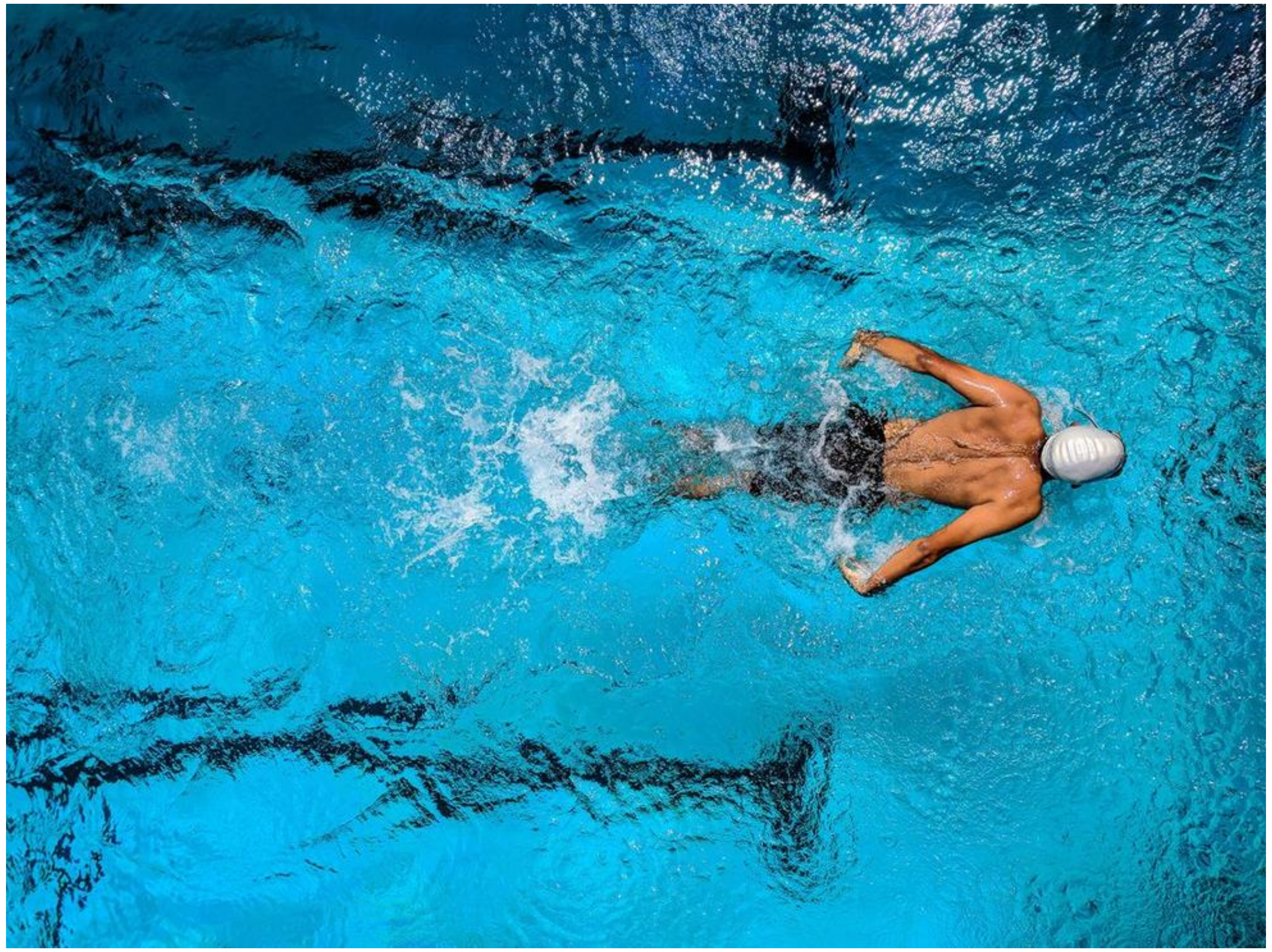
Pedro Martins nº 119916
Marta Cruz nº 119572



BD - APFT

SwiM

BASES DE DADOS 2024/2025

Introdução

- Desenvolvimento de um sistema de gestão de piscinas municipais;

1.

Marcações e cancelamentos de aulas e sessões

2.

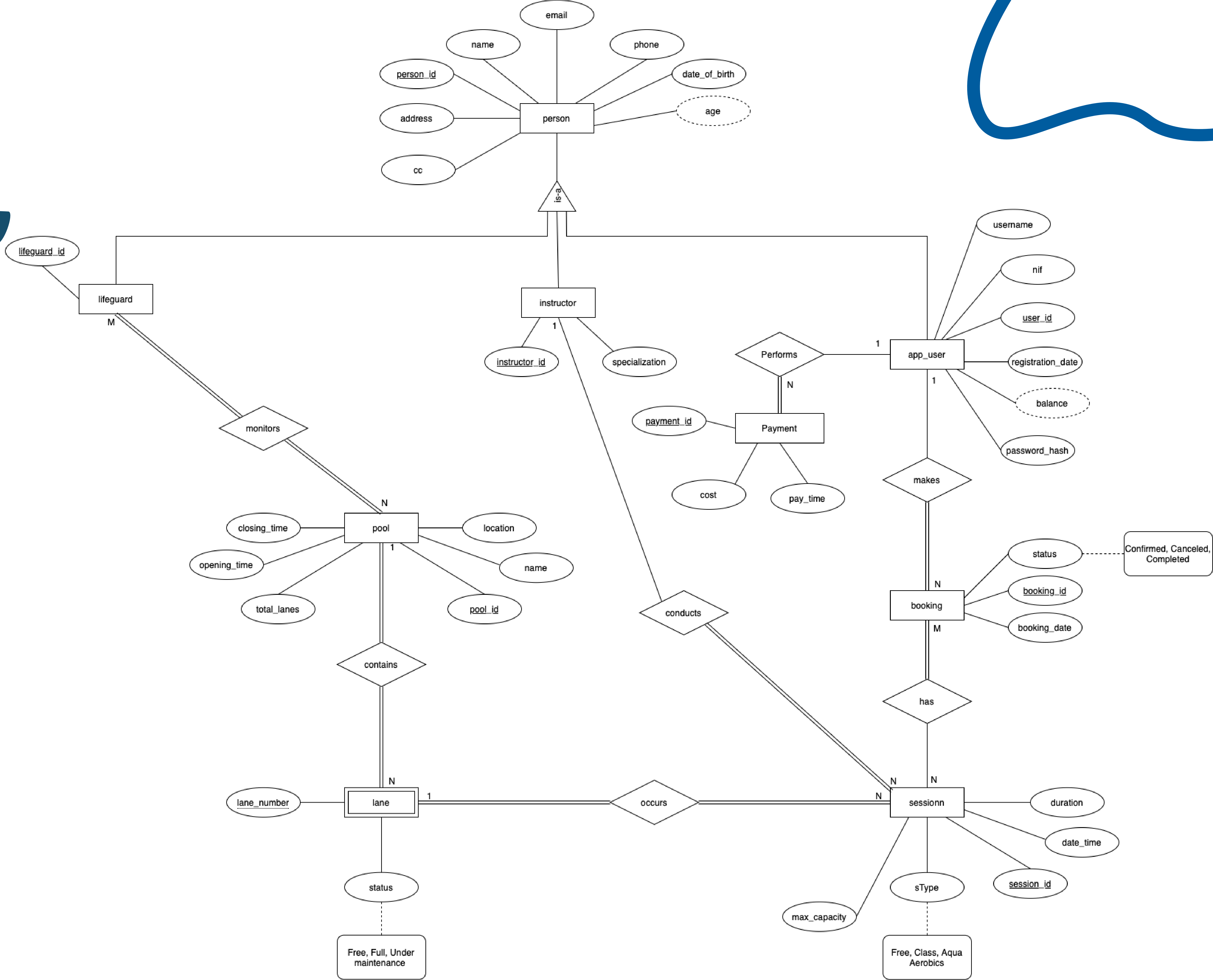
Gestão de saldo

3.

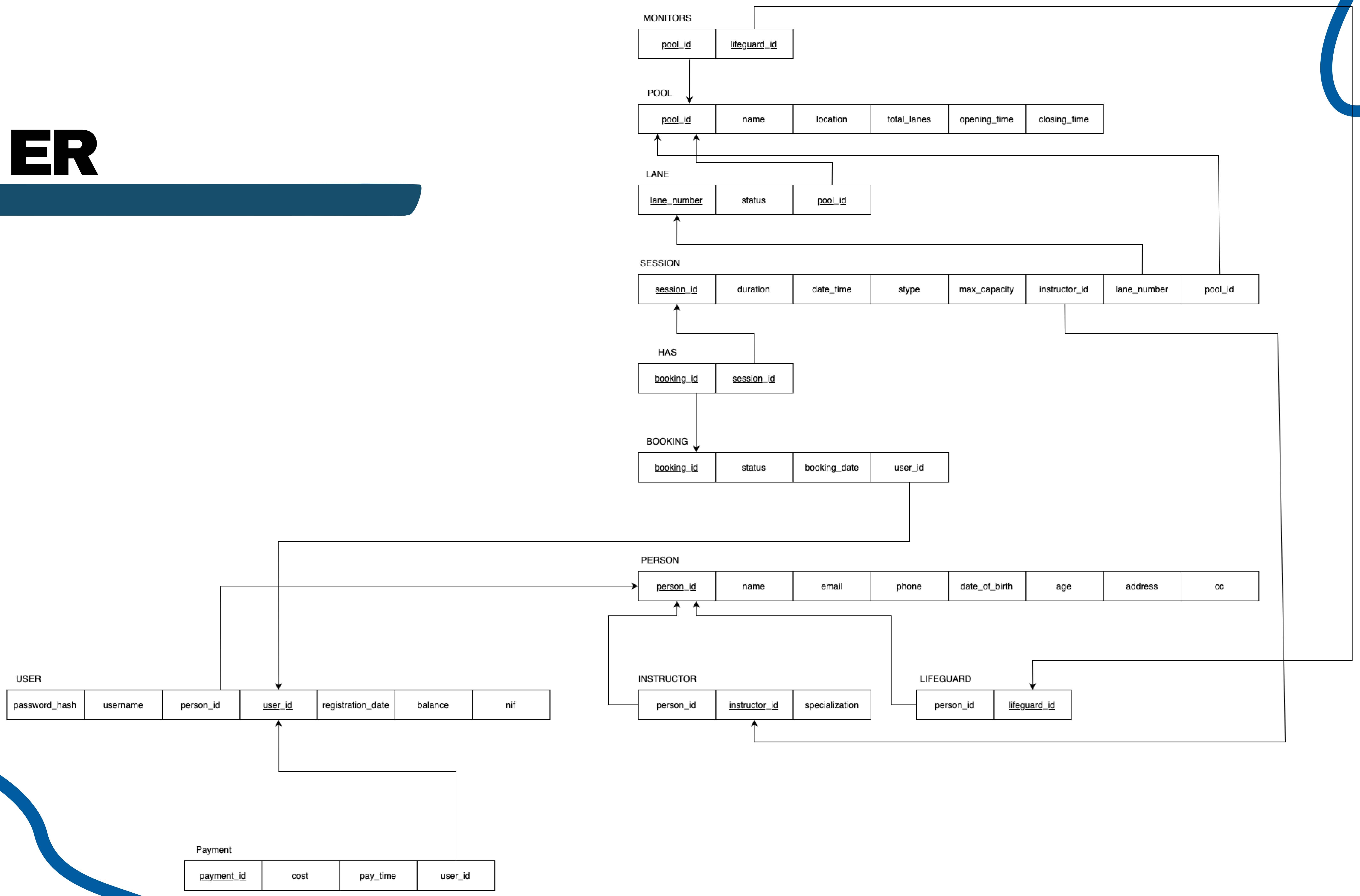
Verificação de ocupação em tempo real;

DER

- Person
- App_user
- Payment
- Instructor
- Lifeguard
- Pool
- Monitors
- Lane
- Session
- Booking
- Has



ER


MONITORS
pool_id
lifeguard_id
POOL
pool_id
name
location
total_lanes
opening_time
closing_time
LANE
lane_number
status
pool_id
SESSION
session_id
duration
date_time
stype
max_capacity
instructor_id
lane_number
pool_id
HAS
booking_id
session_id
BOOKING
booking_id
status
booking_date
user_id
PERSON
person_id
name
email
phone
date_of_birth
age
address
cc
USER
password_hash
username
person_id
user_id
registration_date
balance
nif
INSTRUCTOR
person_id
instructor_id
specialization
LIFEGUARD
person_id
lifeguard_id
Payment
payment_id
cost
pay_time
user_id

Elementos importantes

10.	**2.**	**3.**
SPs	Triggers	UDFs
1.	**1.**	
Index	View	

Stored Procedures (1)

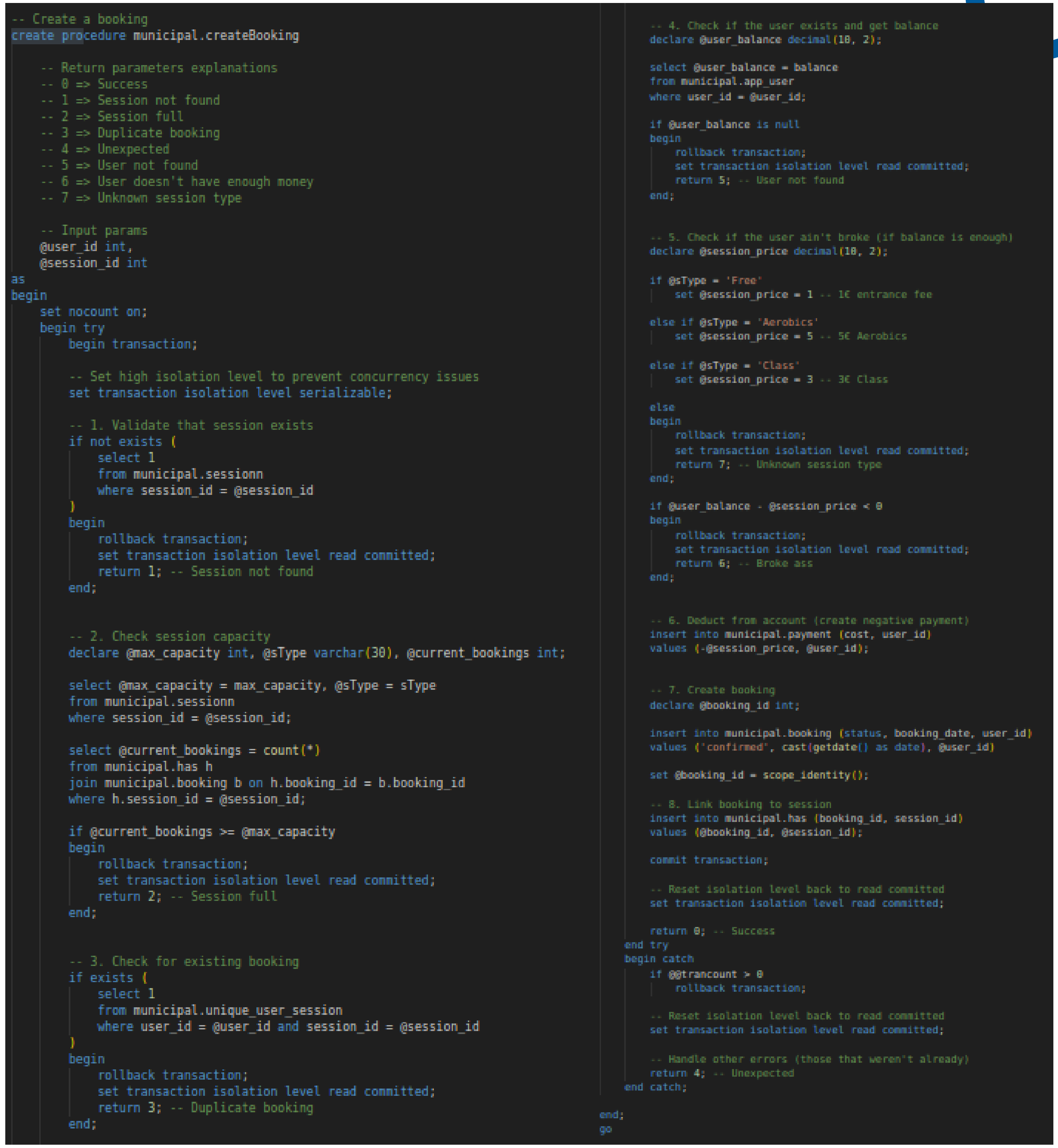
- CreateUser
- CreateInstructor
- CreateLifeguard
- MakePayment
- CreateMonitors
- CreateLane
- createSession
- createBooking
- deleteUser
- deleteBooking

Focados em:

- Criar
- Eliminar
- Backup
- Atualizar

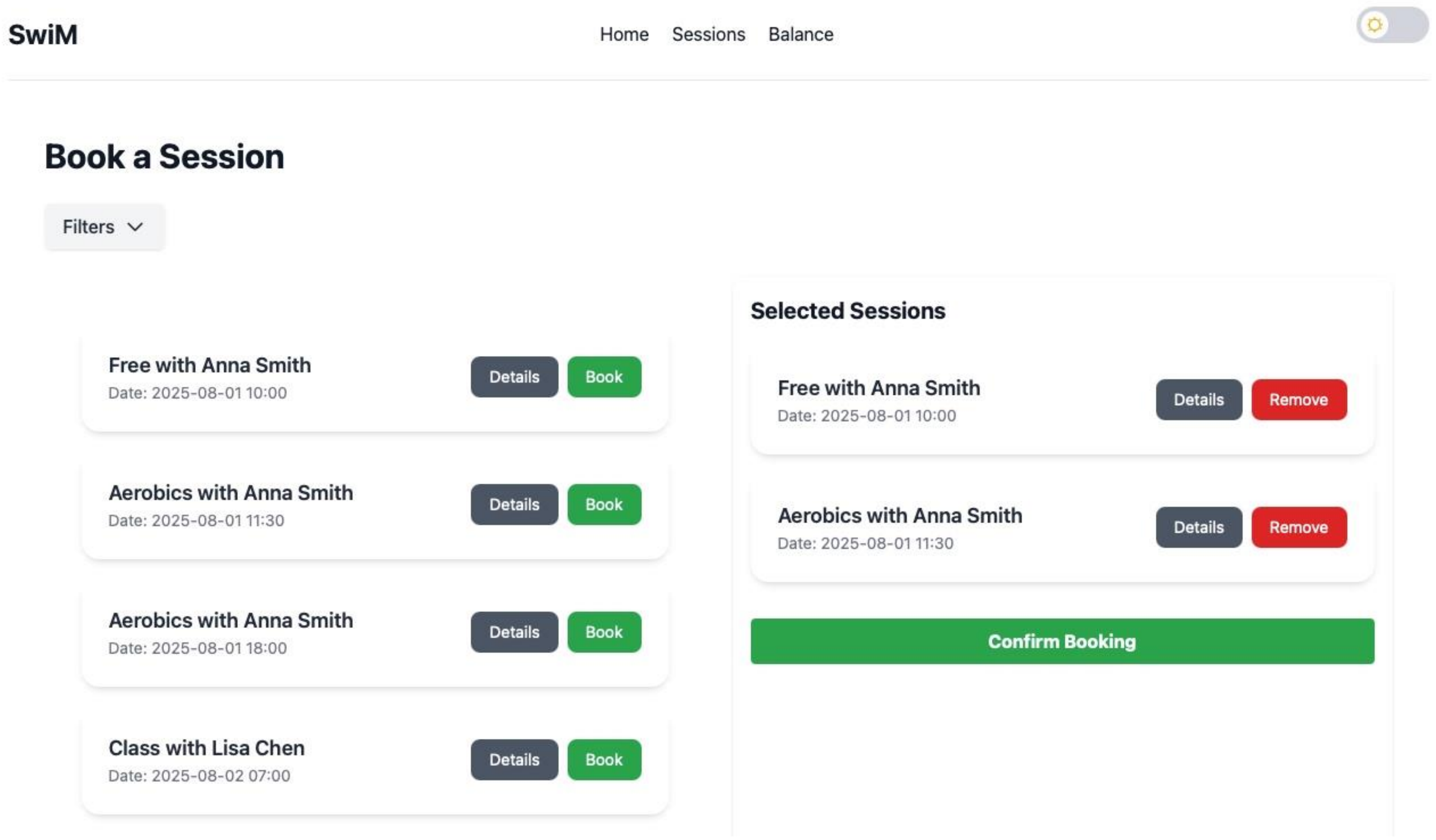
Stored Procedures (2)

- Valida se a sessão existe
- Verifica se a sessão não está cheia
- Verifica se não existe reserva
- Verifica se o utilizador existe
- Obtem o saldo do utilizador
- Verifica se tem saldo suficiente
- Cria um pagamento
- Cria reserva
- Conecta reserva com a sessão



```sql
-- Create a booking
create procedure municipal.createBooking

    -- Return parameters explanations
    -- 0 => Success
    -- 1 => Session not found
    -- 2 => Session full
    -- 3 => Duplicate booking
    -- 4 => Unexpected
    -- 5 => User not found
    -- 6 => User doesn't have enough money
    -- 7 => Unknown session type

    -- Input params
    @user_id int,
    @session_id int
as
begin
    set nocount on;
    begin try
        begin transaction;

        -- Set high isolation level to prevent concurrency issues
        set transaction isolation level serializable;

        -- 1. Validate that session exists
        if not exists (
            select 1
            from municipal.sessionn
            where session_id = @session_id
        )
        begin
            rollback transaction;
            set transaction isolation level read committed;
            return 1; -- Session not found
        end;


        -- 2. Check session capacity
        declare @max_capacity int, @sType varchar(30), @current_bookings int;

        select @max_capacity = max_capacity, @sType = sType
        from municipal.sessionn
        where session_id = @session_id;

        select @current_bookings = count(*)
        from municipal.has h
        join municipal.booking b on h.booking_id = b.booking_id
        where h.session_id = @session_id;

        if @current_bookings >= @max_capacity
        begin
            rollback transaction;
            set transaction isolation level read committed;
            return 2; -- Session full
        end;


        -- 3. Check for existing booking
        if exists (
            select 1
            from municipal.unique_user_session
            where user_id = @user_id and session_id = @session_id
        )
        begin
            rollback transaction;
            set transaction isolation level read committed;
            return 3; -- Duplicate booking
        end;

        -- 4. Check if the user exists and get balance
        declare @user_balance decimal(10, 2);

        select @user_balance = balance
        from municipal.app_user
        where user_id = @user_id;

        if @user_balance is null
        begin
            rollback transaction;
            set transaction isolation level read committed;
            return 5; -- User not found
        end;


        -- 5. Check if the user ain't broke (if balance is enough)
        declare @session_price decimal(10, 2);

        if @sType = 'Free'
            set @session_price = 1 -- 1€ entrance fee

        else if @sType = 'Aerobics'
            set @session_price = 5 -- 5€ Aerobics

        else if @sType = 'Class'
            set @session_price = 3 -- 3€ Class

        else
        begin
            rollback transaction;
            set transaction isolation level read committed;
            return 7; -- Unknown session type
        end;

        if @user_balance - @session_price < 0
        begin
            rollback transaction;
            set transaction isolation level read committed;
            return 6; -- Broke ass
        end;


        -- 6. Deduct from account (create negative payment)
        insert into municipal.payment (cost, user_id)
        values (-@session_price, @user_id);


        -- 7. Create booking
        declare @booking_id int;

        insert into municipal.booking (status, booking_date, user_id)
        values ('confirmed', cast(getdate() as date), @user_id)

        set @booking_id = scope_identity();

        -- 8. Link booking to session
        insert into municipal.has (booking_id, session_id)
        values (@booking_id, @session_id);

        commit transaction;

        -- Reset isolation level back to read committed
        set transaction isolation level read committed;

        return 0; -- Success
    end try
    begin catch
        if @@trancount > 0
            rollback transaction;

        -- Reset isolation level back to read committed
        set transaction isolation level read committed;

        -- Handle other errors (those that weren't already)
        return 4; -- Unexpected
    end catch;

end;
go
```

Stored Procedures (3)



SwiM

Home Sessions Balance

Book a Session

Filters

Free with Anna Smith
Date: 2025-08-01 10:00

Aerobics with Anna Smith
Date: 2025-08-01 11:30

Aerobics with Anna Smith
Date: 2025-08-01 18:00

Class with Lisa Chen
Date: 2025-08-02 07:00

Selected Sessions

Free with Anna Smith
Date: 2025-08-01 10:00

Aerobics with Anna Smith
Date: 2025-08-01 11:30

UDFs (1)

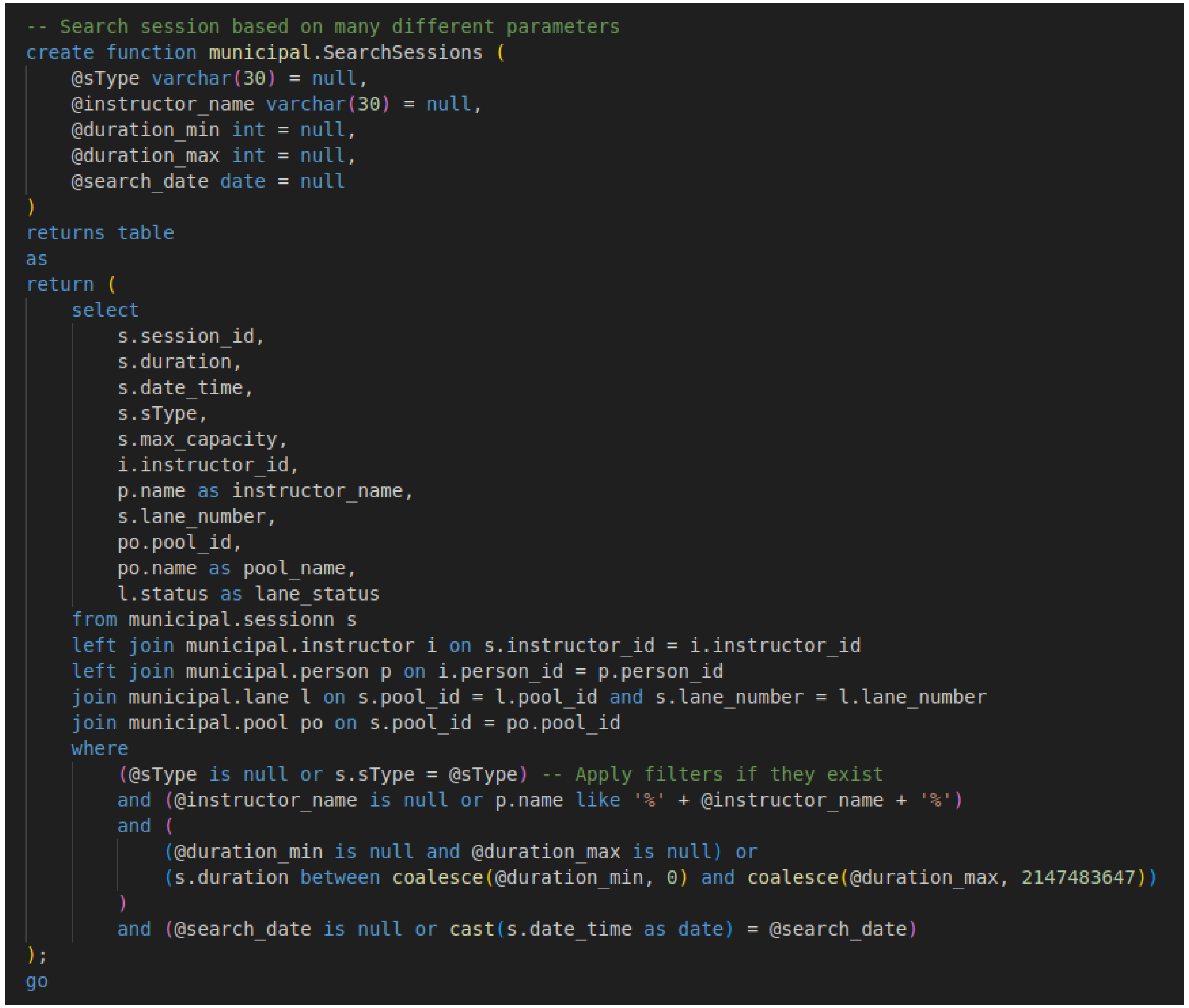
- SearchSessions
- PaymentHistory
- UserBookings

Focadas em:

- Filtrar
- Pesquisar

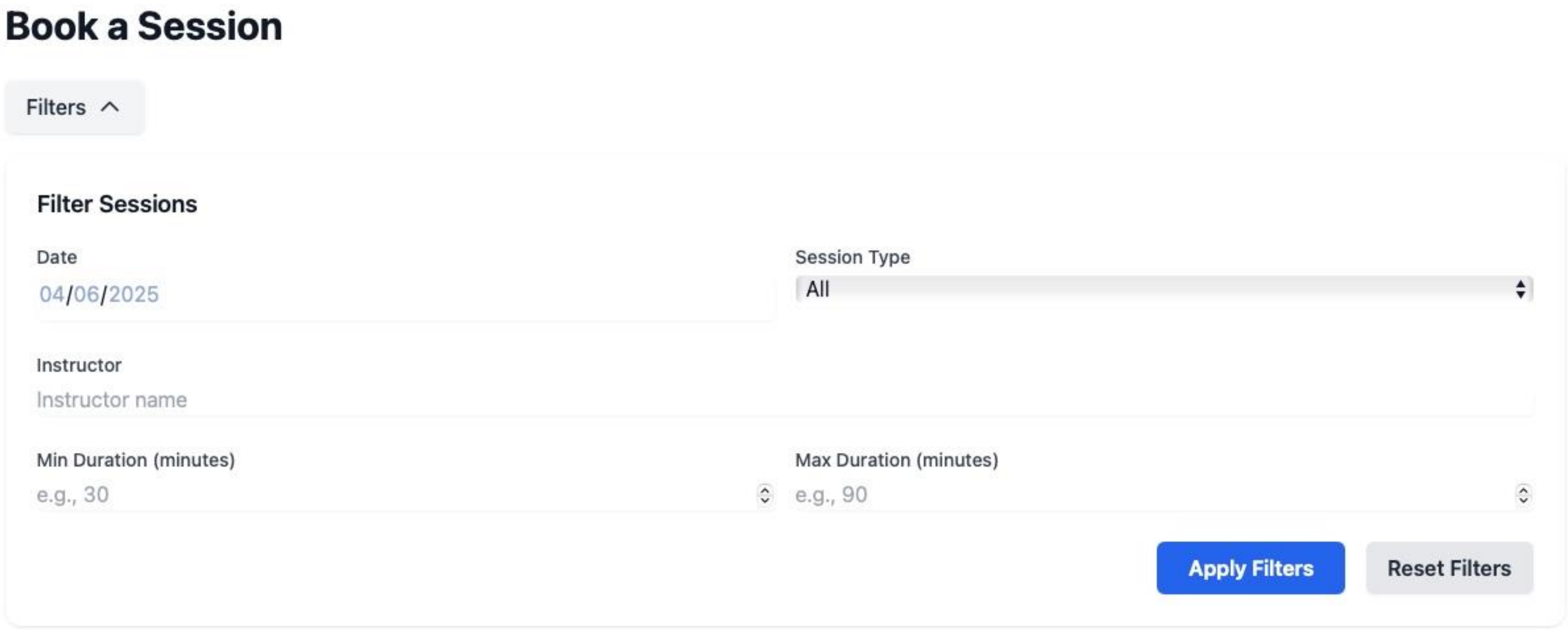
UDFs (2)

- Definição de parametros de entrada
- Definição da estrutura de retorno
- Seleção das colunas principais
- Junções entre tabelas
- Aplicação dos diferentes filtros
- Retorno de resultados



```sql
-- Search session based on many different parameters
create function municipal.SearchSessions (
    @sType varchar(30) = null,
    @instructor_name varchar(30) = null,
    @duration_min int = null,
    @duration_max int = null,
    @search_date date = null
)
returns table
as
return (
    select
        s.session_id,
        s.duration,
        s.date_time,
        s.sType,
        s.max_capacity,
        i.instructor_id,
        p.name as instructor_name,
        s.lane_number,
        po.pool_id,
        po.name as pool_name,
        l.status as lane_status
    from municipal.sessionn s
    left join municipal.instructor i on s.instructor_id = i.instructor_id
    left join municipal.person p on i.person_id = p.person_id
    join municipal.lane l on s.pool_id = l.pool_id and s.lane_number = l.lane_number
    join municipal.pool po on s.pool_id = po.pool_id
    where
        (@sType is null or s.sType = @sType) -- Apply filters if they exist
        and (@instructor_name is null or p.name like '%' + @instructor_name + '%')
        and (
            (@duration_min is null and @duration_max is null) or
            (s.duration between coalesce(@duration_min, 0) and coalesce(@duration_max, 2147483647))
        )
        and (@search_date is null or cast(s.date_time as date) = @search_date)
);
go
```

UDFs (3)



Book a Session

Filters

Filter Sessions

Date
04/06/2025

Session Type
All

Instructor
Instructor name

Min Duration (minutes)
e.g., 30

Max Duration (minutes)
e.g., 90

Apply Filters

Reset Filters

Triggers (1)

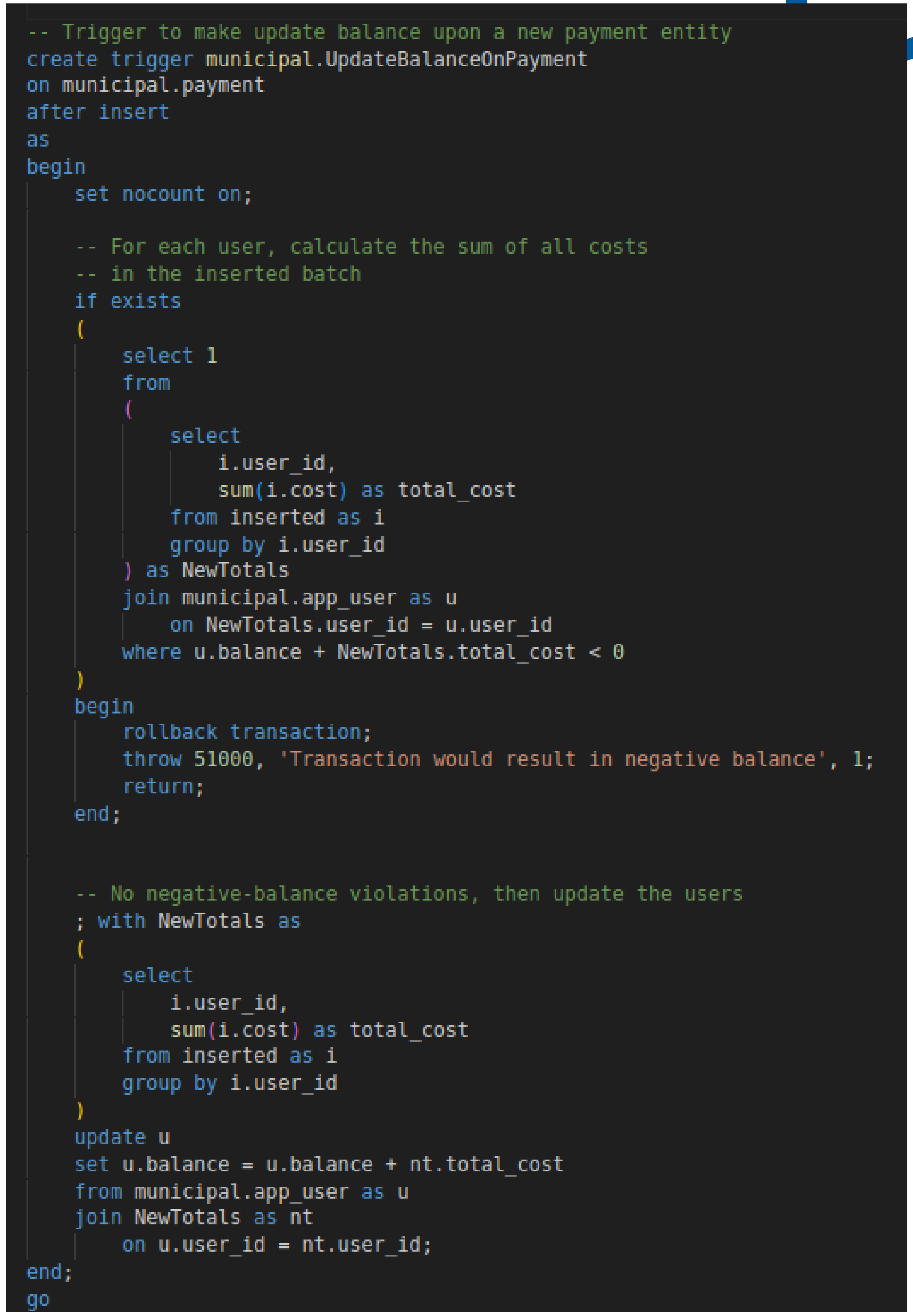
- trg_prevent_duplicate_booking
- UpdateBalanceOnPayment

Focados em:

- Lógica
- Impedir comportamento indevido
- Atualizar

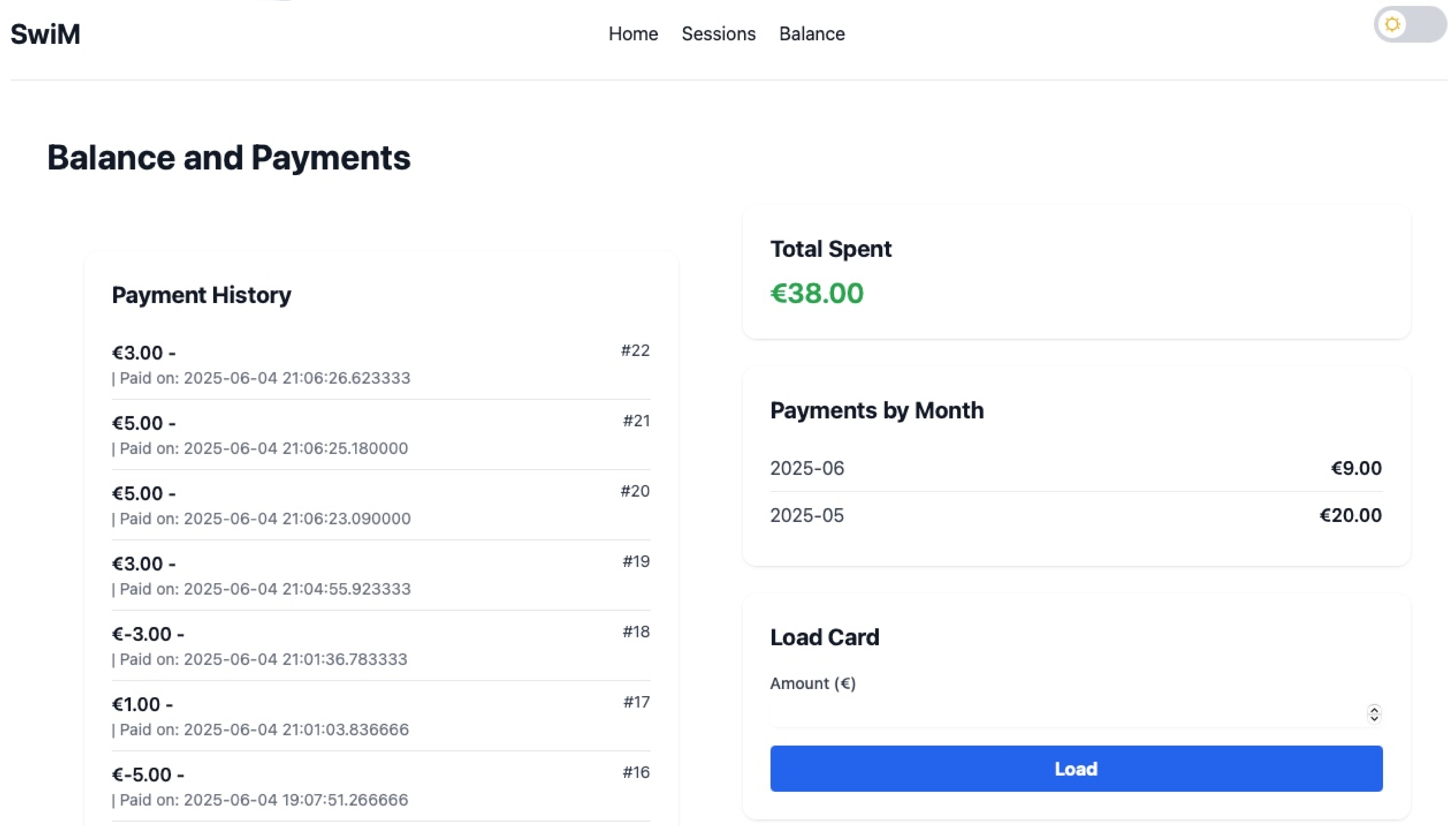
Triggers (2)

- Ativado após a criação duma entidade pagamento
- Verifica se o saldo ficou negativo com o novo pagamento
- Se sim, levanta um erro e retrocede as ações
- Se não, atualiza o saldo do cliente



```sql
-- Trigger to make update balance upon a new payment entity
create trigger municipal.UpdateBalanceOnPayment
on municipal.payment
after insert
as
begin
    set nocount on;

    -- For each user, calculate the sum of all costs
    -- in the inserted batch
    if exists
    (
        select 1
        from
        (
            select
                i.user_id,
                sum(i.cost) as total_cost
            from inserted as i
            group by i.user_id
        ) as NewTotals
        join municipal.app_user as u
            on NewTotals.user_id = u.user_id
        where u.balance + NewTotals.total_cost < 0
    )
    begin
        rollback transaction;
        throw 51000, 'Transaction would result in negative balance', 1;
        return;
    end;


    -- No negative-balance violations, then update the users
    ; with NewTotals as
    (
        select
            i.user_id,
            sum(i.cost) as total_cost
        from inserted as i
        group by i.user_id
    )
    update u
    set u.balance = u.balance + nt.total_cost
    from municipal.app_user as u
    join NewTotals as nt
        on u.user_id = nt.user_id;
end;
go
```

Triggers (3)



SwiM

Home Sessions Balance

Balance and Payments

Payment History

€3.00 - | Paid on: 2025-06-04 21:06:26.623333	#22
€5.00 - | Paid on: 2025-06-04 21:06:25.180000	#21
€5.00 - | Paid on: 2025-06-04 21:06:23.090000	#20
€3.00 - | Paid on: 2025-06-04 21:04:55.923333	#19
€-3.00 - | Paid on: 2025-06-04 21:01:36.783333	#18
€1.00 - | Paid on: 2025-06-04 21:01:03.836666	#17
€-5.00 - | Paid on: 2025-06-04 19:07:51.266666	#16

Total Spent

€38.00

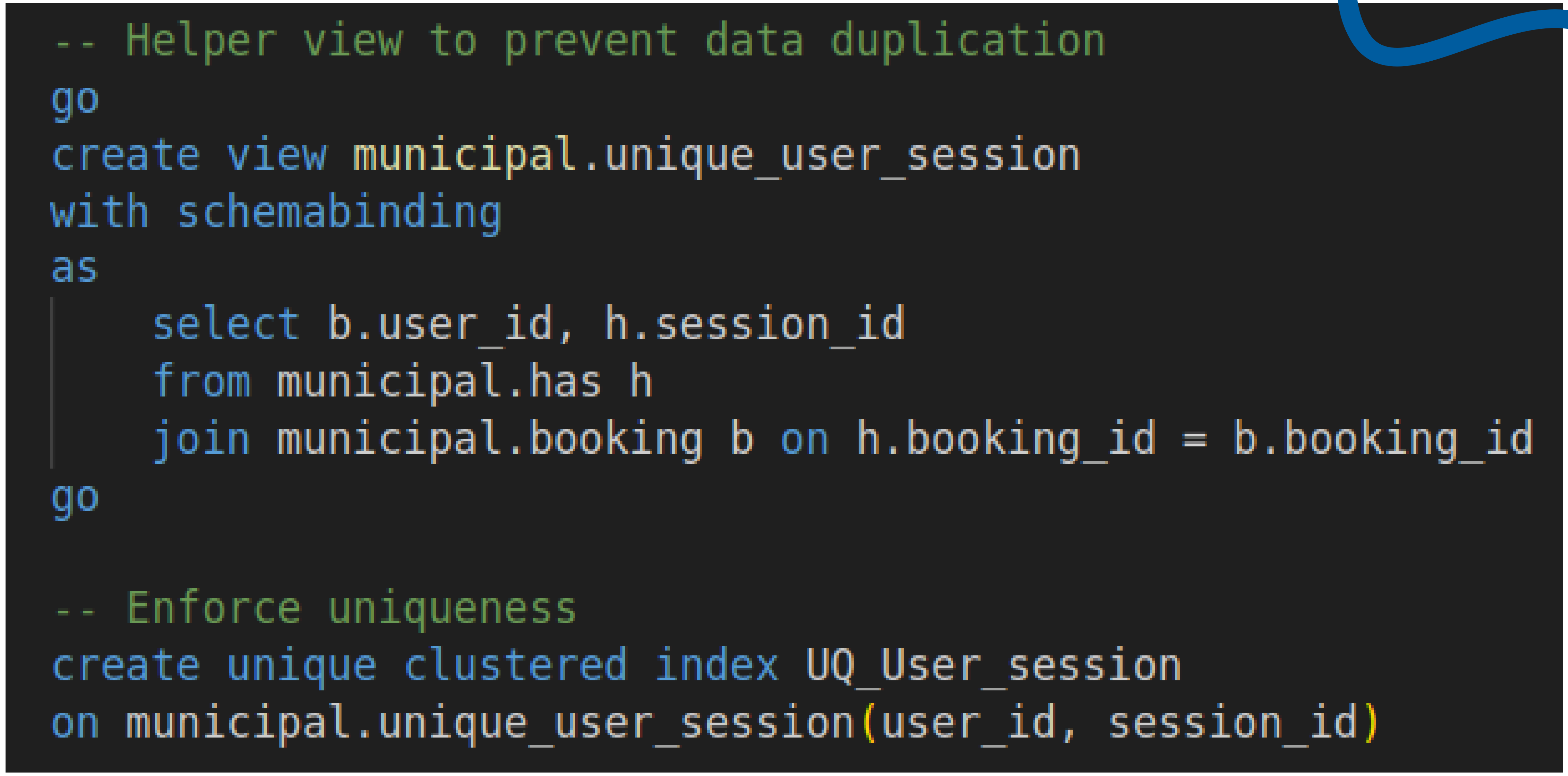
Payments by Month

2025-06	€9.00
2025-05	€20.00

Load Card

Amount (€)

View e index



```sql
-- Helper view to prevent data duplication
go
create view municipal.unique_user_session
with schemabinding
as
    select b.user_id, h.session_id
    from municipal.has h
    join municipal.booking b on h.booking_id = b.booking_id
go

-- Enforce uniqueness
create unique clustered index UQ_User_session
on municipal.unique_user_session(user_id, session_id)
```

Focados em:

- Impedir duplicação de reservas pelo mesmo cliente

Tecnologias usadas

HTML

tailwindcss

</>htmx

Flask
web development,
one drop at a time

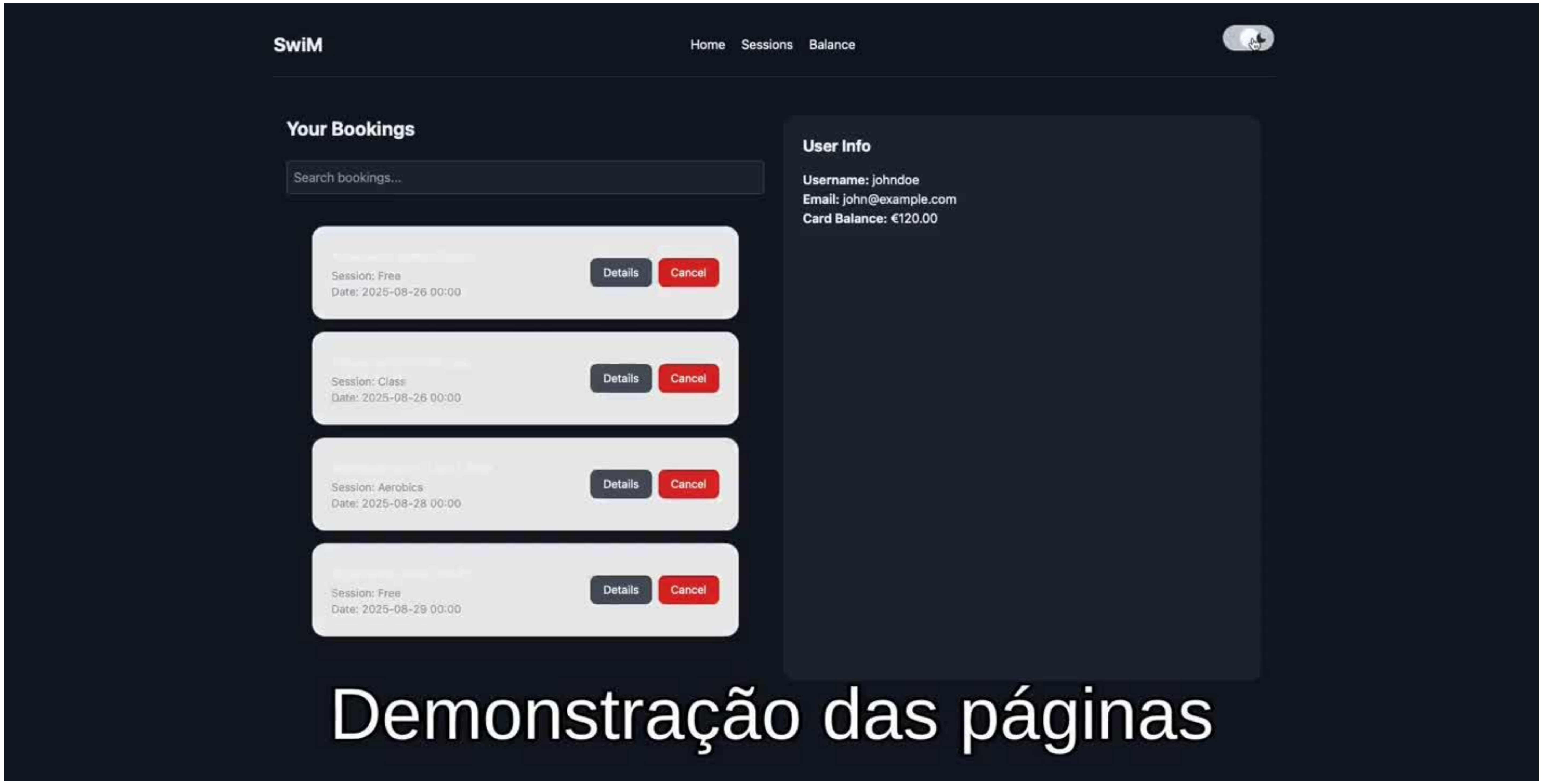
Poetry

django

Tecnologias usadas

Microsoft® SQL Server®™

Video Demo


SwiM

Home Sessions Balance

Your Bookings

Search bookings...

Session: Free
Date: 2025-08-26 00:00

Details Cancel

Session: Class
Date: 2025-08-26 00:00

Details Cancel

Session: Aerobics
Date: 2025-08-28 00:00

Details Cancel

Session: Free
Date: 2025-08-29 00:00

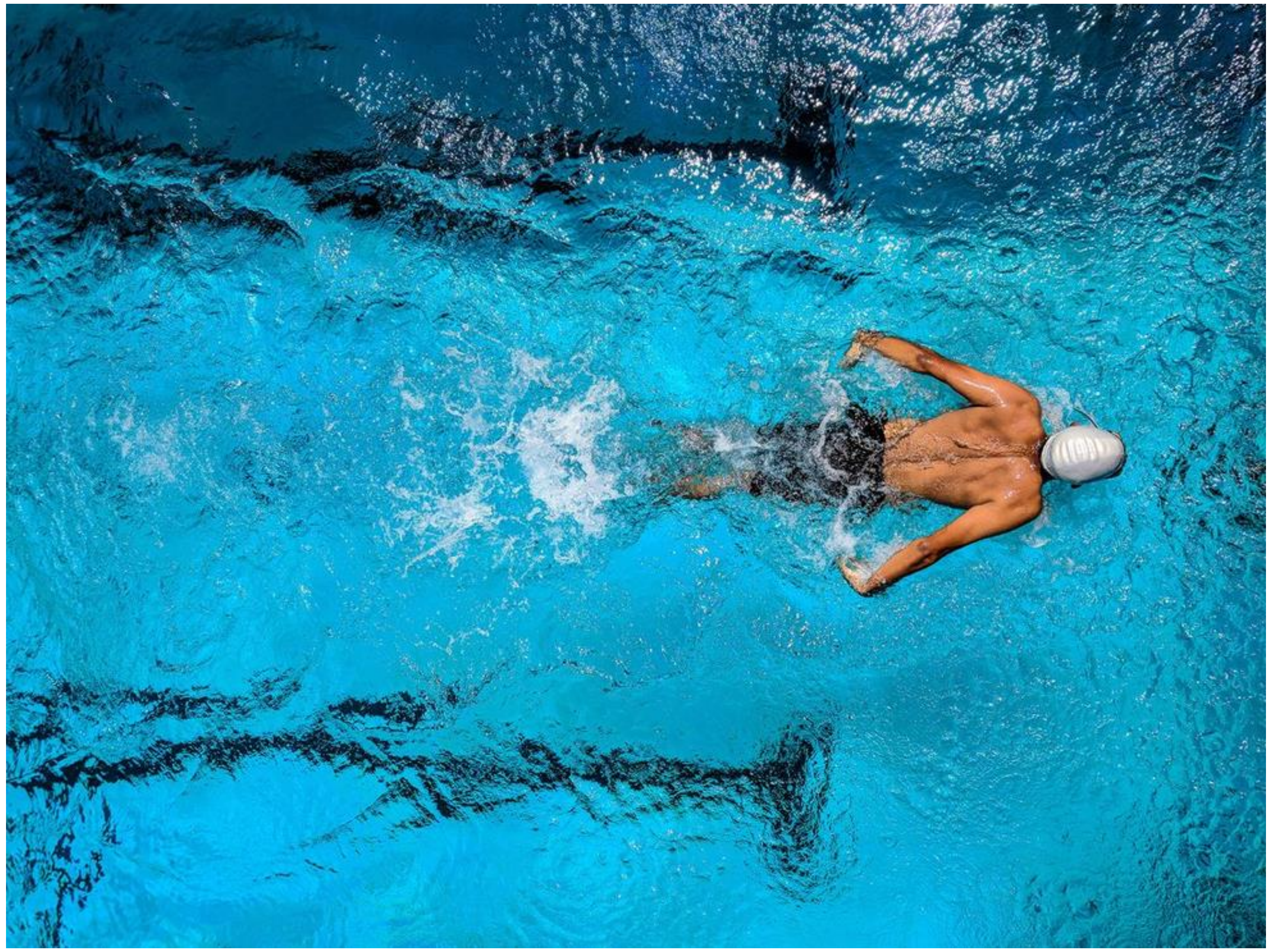
Details Cancel

User Info

Username: johndoe
Email: john@example.com
Card Balance: €120.00

Demonstração das páginas

Pedro Martins nº 119916
Marta Cruz nº 119572



BD - APFT

Questões?

SwiM
BASES DE DADOS 2024/2025