INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DA PARAÍBA CAMPUS ESPERANÇA

CURSO SUPERIOR DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

FILIPE KEVYN GUEDES SILVA

JOSÉ THOMAZ DE ARAÚJO JÚNIOR

THALES LUIZ ARAÚJO CARVALHO

REGISTRO DE USO DE TECNOLOGIAS

ESPERANÇA/PB

2025

Este documento tem como objetivo detalhar a aplicação e a integração das tecnologias utilizadas ao longo do projeto, incluindo MongoDB, JDBC, PostgreSQL com extensão geográfica, MinIO e outros recursos de banco de dados. Nele, são apresentados os contextos em que cada ferramenta foi empregada, os motivos de sua escolha e a forma como contribuíram para o desenvolvimento e funcionamento do sistema. A intenção é fornecer uma visão clara e organizada do uso de cada tecnologia, facilitando o entendimento do projeto como um todo.

1. Banco de Dados Geográficos – PostGIS

O banco de dados geográficos escolhido foi o PostGIS, instalado diretamente no NeonDB. Ele foi utilizado para registrar e pontuar 21 localizações de casas lotéricas na região de Campina Grande, permitindo análises espaciais e consultas geográficas avançadas. Para representar os dados geográficos, foram utilizados os tipos Points (para pontos de localização) e Polygons (para delimitar áreas de interesse).

Inserção de pontos geográficos:

```
INSERT INTO tb_location (coordinates)
VALUES
  (ST_SetSRID(ST_MakePoint(-35.879924, -7.218432), 4326)),
  (ST_SetSRID(ST_MakePoint(-35.884795, -7.218454), 4326)),
  ...
  (ST_SetSRID(ST_MakePoint(-35.879924, -7.218432), 4326));
```

- Cada ponto representa a localização de uma lotérica.
- Os pontos podem ser visualizados individualmente ou em conjunto para criar mapas interativos.

Atualização com polígono:

```
UPDATE tb_location
SET shape = ST_SetSRID(
              ST_GeomFromText(
                'POLYGON((
                  -35.879924 -7.218432,
                  -35.884795 -7.218454,
                  ...
                  -35.879924 -7.218432
                ))'
              ), 4326
            )::geography;
```

- O polígono é construído a partir dos pontos, delimitando uma região da cidade.

- Pode ser utilizado para consultas espaciais, como identificar quais lotéricas estão dentro da área ou calcular distâncias.

1.1. Visualização

A visualização pode ser realizada exportando os dados para GeoJSON e utilizando plataformas online, como geojson.io, QGIS ou Kepler.gl, que permitem explorar os dados no formato GeoJSON de maneira interativa e intuitiva.

Para visualização dos Points:

```
SELECT json_build_object(
  'type', 'FeatureCollection',
  'features', json_agg(
    json_build_object(
      'type', 'Feature',
      'geometry', ST_AsGeoJSON(coordinates)::json,
      'properties', json_build_object('id', id)
    )
  )
) AS geojson
FROM tb_location
WHERE coordinates IS NOT NULL;
```

Para visualização dos Shapes:

```
SELECT ST_AsGeoJSON(shape) FROM tb_location;
```

2. Armazenamento de Arquivos – MinIO

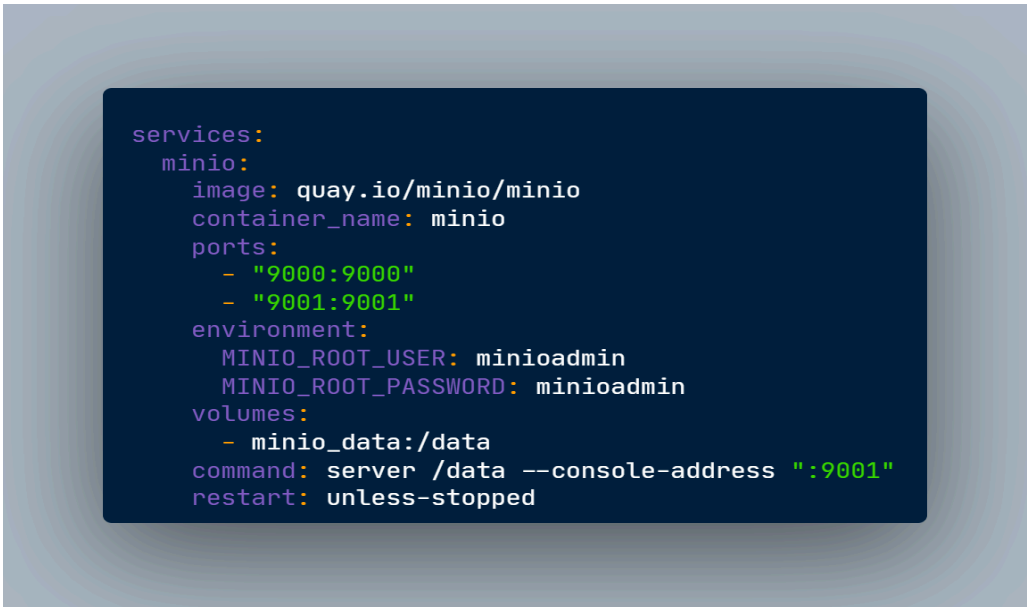
Para a implementação do armazenamento de arquivos no projeto, foi criada uma nova branch dedicada à integração do MinIO. Nesta branch, o código foi refatorado para incluir a lógica de envio e recuperação de arquivos diretamente no serviço, garantindo melhor organização e versionamento do desenvolvimento.

O MinIO foi utilizado para armazenar os comprovantes de apostas (bets) enviados pelos usuários. Esses comprovantes são enviados pelo sistema, armazenados em um bucket específico e ficam disponíveis para futuras consultas.

A ferramenta foi executada via Docker Compose, o que facilitou a configuração e a persistência de dados em ambiente local.

Observação**:** para a execução do MinIO, é necessário que o Docker esteja previamente instalado na máquina.

Configuração com Docker Compose:



```yaml
services:
  minio:
    image: quay.io/minio/minio
    container_name: minio
    ports:
      - "9000:9000"
      - "9001:9001"
    environment:
      MINIO_ROOT_USER: minioadmin
      MINIO_ROOT_PASSWORD: minioadmin
    volumes:
      - minio_data:/data
    command: server /data --console-address ":9001"
    restart: unless-stopped
```

3. Integração com Banco de Dados - JDBC e MongoDB

Para fins de experimentação e validação de outras tecnologias, foram criadas branches separadas dedicadas ao JDBC e ao MongoDB, permitindo que cada uma pudesse ser implementada, testada e refinada sem interferir na base principal do código, que utiliza JPA.

O JDBC nessas branches foi utilizado para gerenciar operações em bancos relacionais diretamente com SQL, garantindo controle detalhado das consultas e transações. Já o MongoDB foi incorporado como uma solução de banco de dados não relacional, baseada em documentos JSON, permitindo armazenar informações de forma mais flexível, sem a necessidade de uma estrutura rígida, e agilizando a manipulação de dados complexos.

A refatoração incluiu a criação de uma camada de repositório que abstrai as operações de persistência, de modo que a aplicação pode interagir com os dados independentemente do tipo de banco utilizado. Essa estratégia trouxe benefícios como maior modularidade, centralização da lógica de acesso a dados e facilidade de manutenção, além de permitir explorar simultaneamente as vantagens de bancos relacionais e não relacionais.

Insert com JDBC:

```
    @Override
    public Contest insert(Contest contest) {
        SimpleJdbcInsert jdbcInsert = new
SimpleJdbcInsert(jdbcTemplate).withTableName("tb_con
test");

        Map<String, Object> values = new HashMap<>
();
        values.put("id",contest.getId());
        values.put("pool_id",
contest.getPool().getId());

        String insertNumber = "INSERT INTO tb_number
(id, number, matched, contest_id, pool_id) VALUES
(?, ?, ?, ?, ?)";
        for (Integer num :
contest.getDrawnNumbers()){
            UUID numberId = UUID.randomUUID();
            jdbcTemplate.update(insertNumber,
numberId, num, false, contest.getId(),
contest.getPool().getId());
        }

        jdbcInsert.execute(values);

        return new Contest();
    }
```

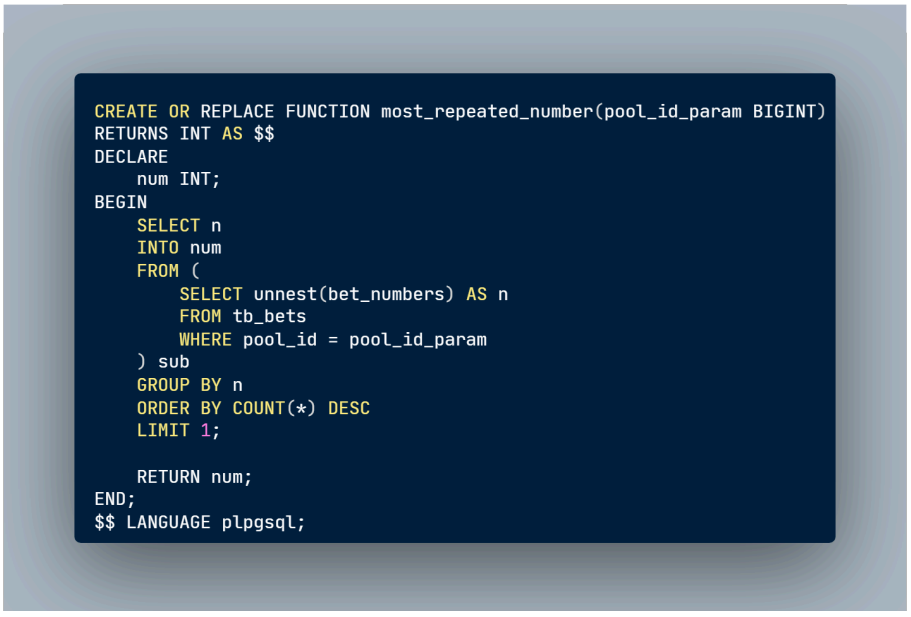
4. Função PL/pgSQL

A função most_repeated_number foi criada no banco de dados NeonDB para calcular o número mais repetido nas apostas de um determinado bolão. Esta função permite identificar rapidamente quais números são mais frequentes, auxiliando em análises estatísticas e relatórios sobre os bolões de apostas.

4.1. Implementação da função

A função foi implementada em PL/pgSQL e recebe como parâmetro o pool_id do tipo BIGINT. Ela percorre todos os números das apostas (bet_numbers) daquele bolão, agrupa-os, conta a frequência e retorna o número mais repetido.

Função:

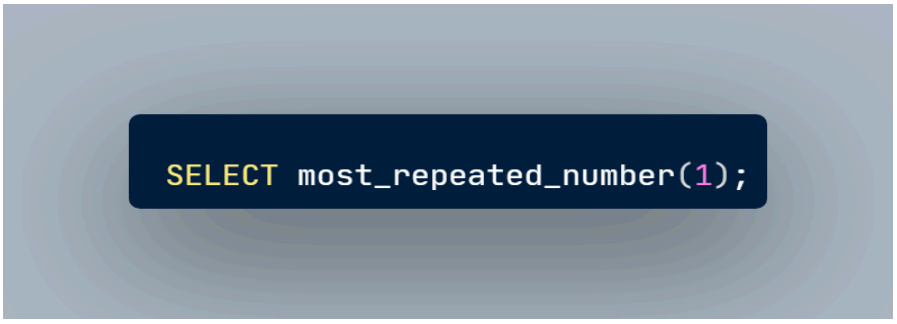


```
CREATE OR REPLACE FUNCTION most_repeated_number(pool_id_param BIGINT)
RETURNS INT AS $$
DECLARE
    num INT;
BEGIN
    SELECT n
    INTO num
    FROM (
        SELECT unnest(bet_numbers) AS n
        FROM tb_bets
        WHERE pool_id = pool_id_param
    ) sub
    GROUP BY n
    ORDER BY COUNT(*) DESC
    LIMIT 1;

    RETURN num;
END;
$$ LANGUAGE plpgsql;
```

- **Entrada:** pool_id_param – identifica o bolão cujas apostas serão analisadas.
- **Saída:** número inteiro (INT) que representa o número mais repetido no bolão.

A função pode ser testada diretamente no banco desta forma:



4.2. Integração com o backend

A função foi integrada ao backend utilizando Spring Boot e JPA, permitindo que o número mais repetido seja calculado e armazenado automaticamente sempre que uma aposta é adicionada.

Função integrada ao backend por meio do seguinte método no BetRepository: