POLITÉCNICO DO PORTO
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Tópicos Avançados de Bases de Dados

Urban Transporation System

José Santos nº 1232153

TABDD
Tópicos Avançados de Bases de Dados

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Instituto Superior de Engenharia do Porto

Outubro, 2025

Índice

1. Introdução

Este projeto visa a criação de uma aplicação dinâmica e inteligente para o sistema de autocarros urbanos de uma cidade. Capaz de monitorizar onde estão os veículos em tempo real, sugerir rotas mais rápidas, marcar linhas favoritas, guardar histórico de viagens, guardar paragens com a sua localização e nome e outras funcionalidades.

Neste primeiro relatório, proporei uma primeira versão da arquitetura das bases de dados que suportarão a aplicação final.

2. Databases & Schemas

A base de dados relacional vai ser baseada no OracleDB. A modelo relacional que apresento para a aplicação de transportes públicos é a seguinte:

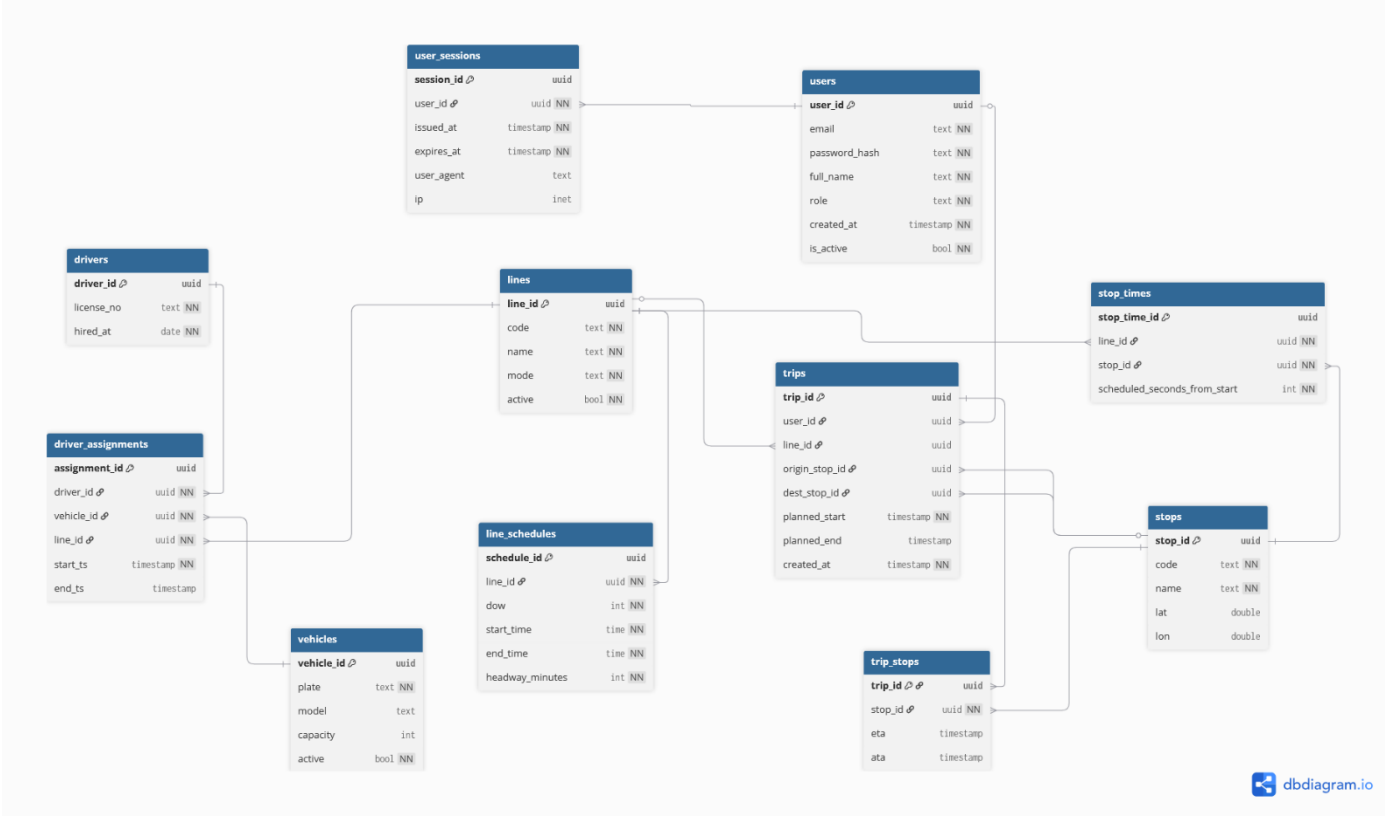

Figura 1 - Diagrama do modelo relacional

Este modelo utiliza a viagem como a tabela principal, que vai ligar linhas, paragens e utilizadores. Permite assim flexibilidade em guardar todas as viagens feitas, atribuir a cada utilizador que pode fazer viagens semelhantes a outros utilizadores, mas guardando a sua paragem inicial e final.

Cada condutor também estará associado a um veículo e a uma linha. Guardando o início e fim das suas viagens com esse veículo nessa linha. Cada linha também tem o seu horário por dia da semana.

Este modelo relacional é flexível para ser implementado em várias cidades ao longo do mundo. Em termos de escalabilidade, ao guardar os dados no mesmo servidor, é recomendado a utilização de tabelas diferentes para cada cidade (e.g.: lines_<cityid>), que apesar de ocupar mais espaço em memória ajudará

com a segurança e estabilidade da aplicação, sendo difícil haver *data leaks*. Contudo, este tópico é algo a explorar no desenvolvimento da aplicação.

O atual esquema para a base de dados baseada em Oracle DB é a seguinte:

```
CREATE TABLE "users" (
  "user_id" uuid PRIMARY KEY,
  "email" text UNIQUE NOT NULL,
  "password_hash" text NOT NULL,
  "full_name" text NOT NULL,
  "role" text NOT NULL CHECK (role in ('passenger','admin')),
  "created_at" timestamp DEFAULT now() NOT NULL,
  "is_active" bool DEFAULT true NOT NULL
);

CREATE TABLE "user_sessions" (
  "session_id" uuid PRIMARY KEY,
  "user_id" uuid NOT NULL,
  "issued_at" timestamp DEFAULT now() NOT NULL,
  "expires_at" timestamp NOT NULL,
  "user_agent" text,
  "ip" inet
);

CREATE TABLE "drivers" (
  "driver_id" uuid PRIMARY KEY,
  "license_no" text NOT NULL,
  "hired_at" date NOT NULL
);

CREATE TABLE "vehicles" (
  "vehicle_id" uuid PRIMARY KEY,
  "plate" text UNIQUE NOT NULL,
  "model" text,
  "capacity" int CHECK (capacity > 0),
  "active" bool DEFAULT true NOT NULL
);

CREATE TABLE "lines" (
  "line_id" uuid PRIMARY KEY,
  "code" text UNIQUE NOT NULL,
  "name" text NOT NULL,
  "mode" text NOT NULL CHECK (mode in ('bus','tram','metro')),
  "active" bool DEFAULT true NOT NULL
);

CREATE TABLE "line_schedules" (
  "schedule_id" uuid PRIMARY KEY,
```

```
  "line_id" uuid NOT NULL,
  "dow" int NOT NULL CHECK (dow between 0 and 6), -- day of the week
  "start_time" time NOT NULL,
  "end_time" time NOT NULL,
  "headway_minutes" int NOT NULL CHECK (headway_minutes > 0)
);

CREATE TABLE "stops" (
  "stop_id" uuid PRIMARY KEY,
  "code" text UNIQUE NOT NULL,
  "name" text NOT NULL,
  "lat" double,
  "lon" double
);

CREATE TABLE "stop_times" (
  "stop_time_id" uuid PRIMARY KEY,
  "line_id" uuid NOT NULL,
  "stop_id" uuid NOT NULL,
  "scheduled_seconds_from_start" int NOT NULL
);

CREATE TABLE "driver_assignments" (
  "assignment_id" uuid PRIMARY KEY,
  "driver_id" uuid NOT NULL,
  "vehicle_id" uuid NOT NULL,
  "line_id" uuid NOT NULL,
  "start_ts" timestamp NOT NULL,
  "end_ts" timestamp,
  CHECK (end_ts is null or end_ts > start_ts)
);

CREATE TABLE "trips" (
  "trip_id" uuid PRIMARY KEY,
  "user_id" uuid,
  "line_id" uuid,
  "origin_stop_id" uuid,
  "dest_stop_id" uuid,
  "planned_start" timestamp NOT NULL,
  "planned_end" timestamp,
  "created_at" timestamp DEFAULT now() NOT NULL
);

CREATE TABLE "trip_stops" (
  "trip_id" uuid,
  "stop_id" uuid NOT NULL,
  "eta" timestamp,
  "ata" timestamp,
  PRIMARY KEY ("trip_id")
```

```
);

CREATE INDEX "idx_users_email" ON "users" ("email");

CREATE UNIQUE INDEX ON "stop_times" ("line_id", "stop_id");

CREATE INDEX "idx_driver_assignments_active" ON "driver_assignments"
("line_id", "start_ts");

ALTER TABLE "user_sessions" ADD FOREIGN KEY ("user_id") REFERENCES
"users" ("user_id") ON DELETE CASCADE;

ALTER TABLE "line_schedules" ADD FOREIGN KEY ("line_id") REFERENCES
"lines" ("line_id") ON DELETE CASCADE;

ALTER TABLE "stop_times" ADD FOREIGN KEY ("line_id") REFERENCES "lines"
("line_id") ON DELETE CASCADE;

ALTER TABLE "stop_times" ADD FOREIGN KEY ("stop_id") REFERENCES "stops"
("stop_id") ON DELETE RESTRICT;

ALTER TABLE "driver_assignments" ADD FOREIGN KEY ("driver_id") REFERENCES
"drivers" ("driver_id") ON DELETE CASCADE;

ALTER TABLE "driver_assignments" ADD FOREIGN KEY ("vehicle_id")
REFERENCES "vehicles" ("vehicle_id") ON DELETE RESTRICT;

ALTER TABLE "driver_assignments" ADD FOREIGN KEY ("line_id") REFERENCES
"lines" ("line_id") ON DELETE RESTRICT;

ALTER TABLE "trips" ADD FOREIGN KEY ("user_id") REFERENCES "users"
("user_id") ON DELETE SET NULL;

ALTER TABLE "trips" ADD FOREIGN KEY ("line_id") REFERENCES "lines"
("line_id");

ALTER TABLE "trips" ADD FOREIGN KEY ("origin_stop_id") REFERENCES "stops"
("stop_id");

ALTER TABLE "trips" ADD FOREIGN KEY ("dest_stop_id") REFERENCES "stops"
("stop_id");

ALTER TABLE "trip_stops" ADD FOREIGN KEY ("trip_id") REFERENCES "trips"
("trip_id") ON DELETE CASCADE;

ALTER TABLE "trip_stops" ADD FOREIGN KEY ("stop_id") REFERENCES "stops"
("stop_id");
```

Em termos das bases de dados NoSQL, vai ser utilizado o Redis para fazer o login e manter sessões de utilizadores, Neo4j para lógica orientada em grafos, ou seja, dar sugestões de caminhos mais rápidos, conexões paragem a paragem das linhas, etc.. e para a escrita de dados como favoritos dos utilizadores, itinerário das linhas, começo e fim de viagens, horários em tempo real das paragens, etc. é utilizado MongoDB. No seguinte parágrafo apresento o esquema atual para estas bases de dados:

```
// ######################
// MongoDB (Document DB)
// ######################

db.createCollection("lines", {
 validator: {
  $jsonSchema: {
   bsonType: "object",
   required: ["_id", "code", "name", "mode", "itinerary"],
   properties: {
    _id: { bsonType: "string" },
    code: { bsonType: "string" },
    name: { bsonType: "string" },
    mode: { enum: ["bus", "tram", "metro"] },
    itinerary: {
     bsonType: "array",
     items: {
      bsonType: "object",
      required: ["stop_id", "seq", "avgStopSec"],
      properties: {
       stop_id: { bsonType: "string" },
       seq: { bsonType: "int" },
       avgStopSec: { bsonType: "int" }
```

```
        }
       }
      },
      alerts: {
       bsonType: "array",
       items: {
        bsonType: "object",
        required: ["msg", "from"],
        properties: {
         msg: { bsonType: "string" },
         from: { bsonType: "date" },
         to: { bsonType: ["date", "null"] }
        }
       }
      }
     }
    }
  }
});

db.createCollection("stops", {
 validator: {
  $jsonSchema: {
   bsonType: "object",
   required: ["_id", "code", "name", "location"],
   properties: {
    _id: { bsonType: "string" },
    code: { bsonType: "string" },
    name: { bsonType: "string" },
    location: {
     bsonType: "object",
     required: ["type", "coordinates"],
     properties: {
      type: { enum: ["Point"] },
```

```
          coordinates: {
            bsonType: "array",
            items: { bsonType: "double" },
            minItems: 2,
            maxItems: 2
          }
        }
      },
      amenities: {
        bsonType: "array",
        items: { bsonType: "string" }
      }
    }
  }
});

db.createCollection("vehicles", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["_id", "plate", "capacity", "line"],
      properties: {
        _id: { bsonType: "string" },
        plate: { bsonType: "string" },
        model: { bsonType: "string" },
        capacity: { bsonType: "int" },
        features: {
          bsonType: "object",
          properties: {
            lowFloor: { bsonType: "bool" },
            aircon: { bsonType: "bool" }
          }
        },
```

```
      line: { bsonType: "string" },
      lastKnown: {
       bsonType: "object",
       required: ["ts", "loc"],
       properties: {
        ts: { bsonType: "date" },
        loc: {
         bsonType: "object",
         required: ["type", "coordinates"],
         properties: {
          type: { enum: ["Point"] },
          coordinates: {
           bsonType: "array",
           items: { bsonType: "double" },
           minItems: 2,
           maxItems: 2
          }
         }
        }
       }
      }
     }
    }
   }
  }
});

db.createCollection("user_profiles", {
 validator: {
  $jsonSchema: {
   bsonType: "object",
   required: ["_id", "favorites", "prefs"],
   properties: {
    _id: { bsonType: "string" },
    favorites: {
```

```
        bsonType: "object",
        required: ["lines", "stops"],
        properties: {
          lines: { bsonType: "array", items: { bsonType: "string" } },
          stops: { bsonType: "array", items: { bsonType: "string" } }
        }
      },
      prefs: {
        bsonType: "object",
        properties: {
          notifyDisruptions: { bsonType: "bool" },
          units: { enum: ["metric", "imperial"] }
        }
      },
      recentTrips: {
        bsonType: "array",
        items: {
          bsonType: "object",
          required: ["trip_id", "at", "origin", "dest"],
          properties: {
            trip_id: { bsonType: "string" },
            at: { bsonType: "date" },
            origin: { bsonType: "string" },
            dest: { bsonType: "string" }
          }
        }
      }
    }
  }
});

db.createCollection("trips", {
  validator: {
```

```
   $jsonSchema: {
    bsonType: "object",
    required: ["_id", "user_id", "line_id", "planned_start"],
    properties: {
     _id: { bsonType: "string" },
     user_id: { bsonType: "string" },
     line_id: { bsonType: "string" },
     planned_start: { bsonType: "date" },
     planned_end: { bsonType: ["date", "null"] },
     origin_stop: { bsonType: "string" },
     dest_stop: { bsonType: "string" },
     created_at: { bsonType: "date" }
    }
   }
  }
});

db.createCollection("trip_stops", {
 validator: {
   $jsonSchema: {
    bsonType: "object",
    required: ["trip_id", "stop_id", "seq"],
    properties: {
     trip_id: { bsonType: "string" },
     stop_id: { bsonType: "string" },
     seq: { bsonType: "int" },
     eta: { bsonType: "date" },
     ata: { bsonType: ["date", "null"] }
    }
   }
  }
});

db.createCollection("line_schedules", {
```

```
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["_id", "line_id", "dow", "start_time", "end_time", "headway_minutes"],
      properties: {
        _id: { bsonType: "string" },
        line_id: { bsonType: "string" },
        dow: { bsonType: "int", minimum: 0, maximum: 6 },
        start_time: { bsonType: "string" },
        end_time: { bsonType: "string" },
        headway_minutes: { bsonType: "int", minimum: 1 }
      }
    }
  }
});

// ##################
// Neo4j (Graph DB)
// ##################

Schema Constraints
CREATE CONSTRAINT stop_id_unique IF NOT EXISTS FOR (s:Stop) REQUIRE s.stop_id IS UNIQUE;
CREATE CONSTRAINT line_id_unique IF NOT EXISTS FOR (l:Line) REQUIRE l.line_id IS UNIQUE;
CREATE INDEX stop_code IF NOT EXISTS FOR (s:Stop) ON (s.code);

// Sample Nodes
CREATE (:Stop { stop_id: 'STOP001', code: 'ST01', name: 'Example Station', location: point({ longitude: -3.14, latitude: 42.42 }) });
CREATE (:Stop { stop_id: 'STOP002', code: 'ST02', name: 'Example2 Station', location: point({ longitude: -3.14, latitude: 42.42 }) });
CREATE (:Stop { stop_id: 'STOP003', code: 'ST03', name: 'Example3 Station', location: point({ longitude: -3.14, latitude: 42.42 }) });

CREATE (:Line { line_id: 'LINE001', code: 'L1', name: 'Colored Line' });
CREATE (:Line { line_id: 'LINE002', code: 'L2', name: 'Colored2 Line' });
```

```
// Relationships: Connectivity
MATCH (a:Stop { stop_id: 'STOP001' }), (b:Stop { stop_id: 'STOP002' })
CREATE (a)-[:NEXT { line_id: 'LINE001', seq: 1, avg_travel_s: 120 }]->(b);

MATCH (b:Stop { stop_id: 'STOP002' }), (c:Stop { stop_id: 'STOP003' })
CREATE (b)-[:NEXT { line_id: 'LINE001', seq: 2, avg_travel_s: 180 }]->(c);

// Transfer path between lines
MATCH (a:Stop { stop_id: 'STOP001' }), (c:Stop { stop_id: 'STOP003' })
CREATE (a)-[:TRANSFER { walk_s: 300 }]->(c);

// Stop-Line service mapping
MATCH (s:Stop { stop_id: 'STOP001' }), (l:Line { line_id: 'LINE001' })
CREATE (s)-[:SERVES]->(l);

MATCH (s:Stop { stop_id: 'STOP002' }), (l:Line { line_id: 'LINE001' })
CREATE (s)-[:SERVES]->(l);

MATCH (s:Stop { stop_id: 'STOP003' }), (l:Line { line_id: 'LINE002' })
CREATE (s)-[:SERVES]->(l);

// ######################
// Redis (Key–Value DB)
// ######################

// Format: access_token:<session_id> → user_id
SET access_token:abdc1234 user_1234 EX 3600
```

3. Arquitetura

A arquitetura do projeto numa fase inicial encontrar-se há do seguinte modo:

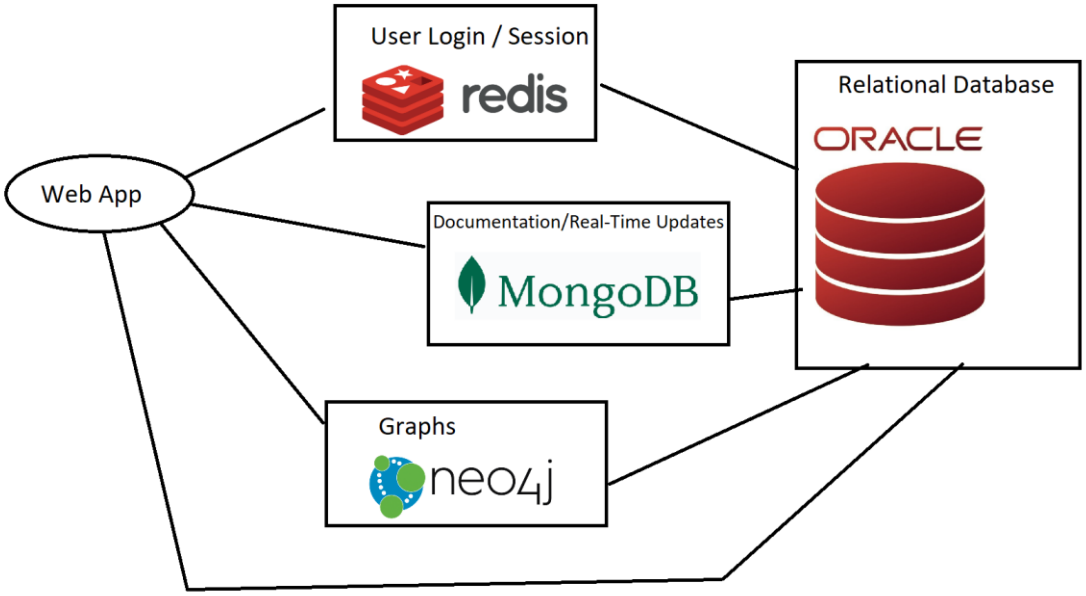

Figura 2 - Arquitetura da Aplicação

O login do utilizador e a sua sessão será mantida pelo Redis (*User Authentication System*), qualquer atualização passará pelo MongoDB (*Real-time vehicle tracking, User Interaction Options, Public Route Transport Information, Travel History*) e os cálculos de linhas e rotas mais rápidas serão mantidas pelo Neo4j (*Automatic Suggestion of optimized routes, Public Transport route information*). O OracleDB guardará os dados e apresenta-os em queries pedidas pelo utilizador.

Penso que esta arquitetura vai de encontro com os objetivos do projeto e é uma arquitetura simples que permite ser escalável e flexível para várias cidades e vários tipos diferentes de transportes urbanos ao longo do mundo.

Quanto ao UI, será feito com Python com recurso ao FastAPI. Em fases mais avançadas do projeto, seria possível trocar para uma interface mais recente e polida como Typescript e bibliotecas React, para tornar o webapp mais atual e recente. Também seria útil o desenvolvimento de uma aplicação móvel. Contudo, numa fase inicial será mantido uma WebAPP simples e funcional.