Overview: A Case Study of Contract and Payment Service

This assignment serves as the final validation of core skills and competencies essential for the role of Payment Engineer. It is designed to be completed within 72 hours (3 days). Any candidate that gets **85%** - **100%** correctly will be eligible for the role. Key considerations include:

- Implementing transactions and locks
- Handling race conditions
- Ensuring concurrency at scale
- Adhering to code style (SOLID principles and software design patterns)
- Writing efficient and optimized database queries
- Writing tests

Stack to be used

The candidate is at liberty to make use of any of these tech stacks:

- Java Spring and PostgreSQL
- NestJS and PostgreSQL

Profile Model

Profile models in this system are categorized as either **client** or a **contractor**. Clients can create contracts with contractors. Contractors, in turn, perform jobs for clients and get payment. Each profile whether client or contractor has a **balance** property.

Contract Model

In the relationship between a **client** and a **contractor**, contracts play a central role. These contracts can exist in one of three statuses: **new**, **in_progress**, or **terminated**. Contracts are deemed active only when they are in the **in_progress** status. Additionally, contracts serve as containers grouping individual jobs within them.

Job

Contractors receive payment for the jobs they perform for clients under certain contracts.

API to Implement

- ***GET*** `/contracts/:id` - The endpoint should return the contract only if it is associated with the profile making the request.

- ***GET*** `/contracts` - The query should provide a list of contracts associated with a user (either a client or contractor). The list should exclusively include contracts that are currently active and not terminated.

- ***GET*** `/jobs/unpaid` - Get all unpaid jobs for a user (***either*** a client or contractor), for ***active contracts only***.

- ***POST*** `/jobs/:job_id/pay` - To pay for a job, a client is only eligible to proceed if their balance is greater than or equal to the specified payment amount. If the conditions are met, the designated amount should be transferred from the client's balance to the contractor's balance.

- ***POST*** `/balances/deposit/:userId` - Money can be deposited into the balance of a client. However, a client is restricted from depositing an amount that exceeds 25% of their total outstanding payments for jobs at the time of the deposit.

- ***GET*** `/admin/best-profession?start=<date>&end=<date>` - The endpoint should return the profession that earned the most money, calculated as the sum of payments for jobs, among all contractors who worked within the specified time range.

- ***GET*** `/admin/best-clients?start=<date>&end=<date>&limit=<integer>` - The endpoint should return the clients who paid the most for jobs within the specified period. The query should also include a limit parameter, with the default limit set to 2.

Authentication

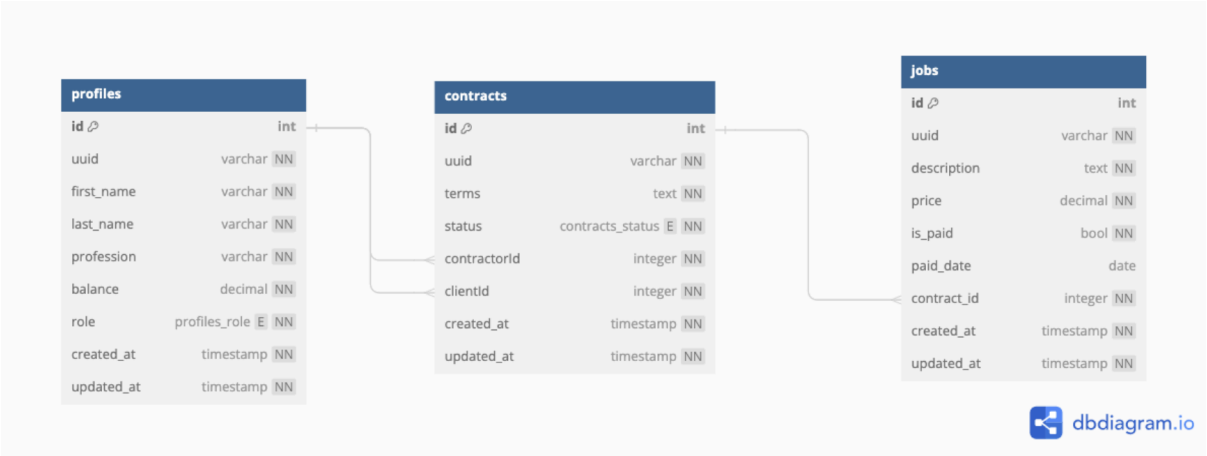
- To authenticate users create a middleware responsible for retrieving a user profile from the database based on a profile ID provided in the request headers.

- If no profile is found, the middleware immediately responds with a **401 Unauthorized** status and ends the response. This indicates that access is denied if the profile cannot be retrieved.

- If a profile is found, it is attached to the **req** object as **req.profile**, making it accessible to subsequent middleware functions and route handlers. Finally, the **next()** function is called to pass control to the next middleware function in the stack.

- The idea is to simulate an experience that makes it useful for ensuring that a valid profile is associated with incoming requests, enabling subsequent handlers to access user-specific data securely.

Database schema Image



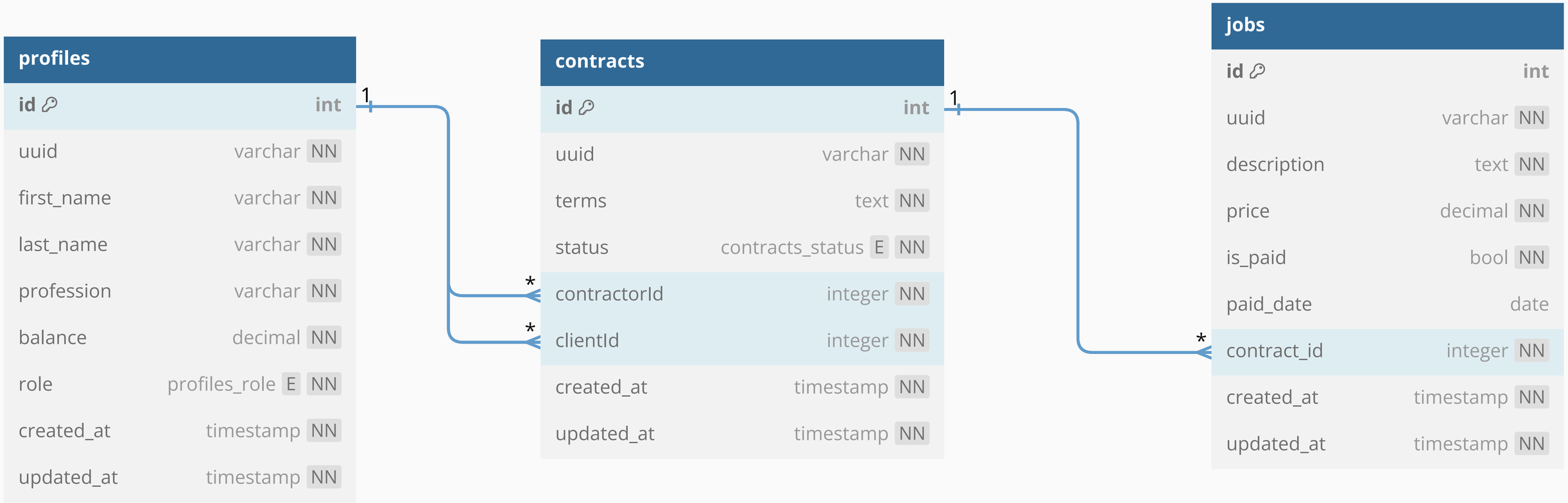
PDF Schema Version

The PDF Schema Version is attached to the end of this document.

SQL Commands

```
CREATE TYPE "profiles_role" AS ENUM (
  'client',
  'contractor'
);

CREATE TYPE "contracts_status" AS ENUM (
  'new',
  'in_progress',
  'terminated'
);

CREATE TABLE "profiles" (
  "id" int PRIMARY KEY,
  "uuid" varchar UNIQUE NOT NULL,
  "first_name" varchar NOT NULL,
  "last_name" varchar NOT NULL,
  "profession" varchar NOT NULL,
  "balance" decimal NOT NULL,
  "role" profiles_role NOT NULL,
  "created_at" timestamp NOT NULL,
  "updated_at" timestamp NOT NULL
);

CREATE TABLE "contracts" (
```

```
  "id" int PRIMARY KEY,
  "uuid" varchar UNIQUE NOT NULL,
  "terms" text NOT NULL,
  "status" contracts_status NOT NULL,
  "contractorId" integer NOT NULL,
  "clientId" integer NOT NULL,
  "created_at" timestamp NOT NULL,
  "updated_at" timestamp NOT NULL
);

CREATE TABLE "jobs" (
  "id" int PRIMARY KEY,
  "uuid" varchar UNIQUE NOT NULL,
  "description" text NOT NULL,
  "price" decimal NOT NULL,
  "is_paid" bool NOT NULL,
  "paid_date" date,
  "contract_id" integer NOT NULL,
  "created_at" timestamp NOT NULL,
  "updated_at" timestamp NOT NULL
);

COMMENT ON COLUMN "contracts"."terms" IS 'Content of the contract';

COMMENT ON COLUMN "jobs"."description" IS 'Details of the contract job';

ALTER TABLE "contracts" ADD FOREIGN KEY ("contractorId") REFERENCES "profiles" ("id");

ALTER TABLE "contracts" ADD FOREIGN KEY ("clientId") REFERENCES "profiles" ("id");

ALTER TABLE "jobs" ADD FOREIGN KEY ("contract_id") REFERENCES "contracts" ("id");
```



```
Enum profiles_role {
    client
    contractor
}


Enum contracts_status {
    new
    in_progress
    terminated
}
```