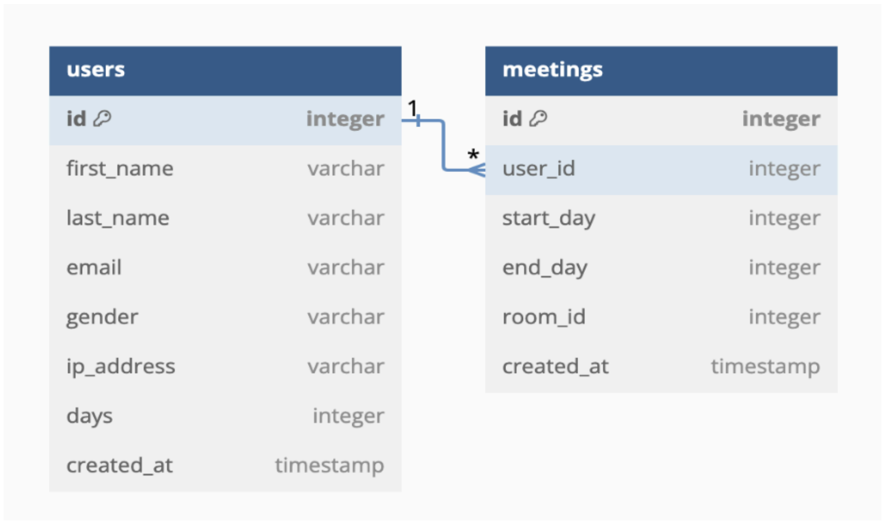
Take-home assignment (Fullstack Engineer)

Get & Show data working days

Database Options:

1. Use the two attached JSON files named 'users.json' and 'meetings.json'.
2. Create your own database following the format specified below.



users	
id	integer
first_name	varchar
last_name	varchar
email	varchar
gender	varchar
ip_address	varchar
days	integer
created_at	timestamp

meetings	
id	integer
user_id	integer
start_day	integer
end_day	integer
room_id	integer
created_at	timestamp

Database Attributes:

- Users table:
  - `id` (integer, primary key): Unique identifier assigned to each employee.
  - `first_name` (varchar): First name of the employee.
  - `last_name` (varchar): Last name of the employee.
  - `email` (varchar): Email address associated with the employee for communication and identification purposes.
  - `gender` (varchar): Gender of the employee, commonly represented as 'male', 'female', or 'other'.
  - `ip_address` (varchar): IP address attributed to the employee.
  - `days` (integer): Total number of days the employee is available for work, starting from day 1 of their availability.
  - `created_at` (timestamp): Timestamp indicating the date and time when the employee record was initially created in the system.
- Meetings table:
  - `id` (integer, primary key): Unique identifier for each attendance record.
  - `user_id` (integer): Identifier referencing the user who attended the meeting.
  - `room_id` (integer): Identifier referencing the room where the meeting took place.
  - `start_day` (integer): The day on which the meeting starts.
  - `end_day` (integer): The day on which the meeting ends.
  - `created_at` (timestamp): Timestamp indicating when the attendance record was created.

Constraints:

- 1 <= `users.days` <= 50
- 1 <= `meetings.start_day` <= `meetings.end_day` <= `users.days`

Requirements

Language Preferences (optional):

- Back-end: Node.js (NestJS)
- Front-end: React (Next.js)

Initial Data Fetch:

- Fetch the first 10 records when the page loads.
- Display these records on the webpage.

Infinite Scrolling:

- Detect when the user has scrolled to the bottom of the page.
- Fetch the next 10 records when the bottom is reached.
- Append the newly fetched records to the existing list on the webpage.

Front-end:

- Display data including the following fields:
    - `"id"`
    - `"first_name"`
    - `"last_name"`
    - `"email"`
    - `"gender"`
    - `"days"`
    - `"meeting_days"` (a list of meetings with start and end dates, represented by tags or other UI. For example `1 -> 2` `3 -> 4`)
    - `"days_without_meetings"` (number of days when the employee is free to work without any scheduled meetings)

Back-end API:

- Provide an endpoint to fetch working days (from your tables or read 2 file json).
- Support query parameters for pagination (offset and limit).

Error Handling:

- Handle and display errors that occur during the fetch operation.
- Inform the user if no more data is available.

Assignment Deployment

- Deploy your project to Vercel.
- Upload code to Github.

Assessment Criteria

- Typescript.
- ESLint - Babel - ES6/7 Syntax.
- Clean structure, components, style components, and API security.
- Code readability and maintainability.
- Performance and exception handling.