Test Assignment for Kotlin Developer

Introduction

In any web-application the majority of HTTP requests are handled by code that queries a database and returns data back
with some data transformations.
All HTTP requests are basically divided into two groups, Write and Read.
A write request is something that changes the internal state of the application and the application must ensure that its state is valid.
A read request is just a single or multiple requests to a database with certain criteria, with further data mapping & transformation back to end-user. In many cases read a request breaks down into multiple queries to database. For example limited listing of records with the total number of all records from the table.

Task

You’ve been asked to create a simple but not trivial web-application written in Kotlin
that can do following things:

Data Write

1. On following HTTP request

```
POST /users
Content-Type: application/json

{ "name": "John Doe", "email": "test@users.com", "password
": "any-random-string" }
```

1. It should insert a row in following postgres table

```
CREATE TABLE users (
    email text NOT NULL PRIMARY KEY ,
    name text NOT NULL,
    password_hash text NOT NULL
)
```

Password must be hashed with BCrypt version 2a and 10 rounds of hashing.

1. And respond back with status either `202 Accepted` with empty body,
   or `400 Bad request` with following body:

```
{ "error": "Email duplicate: <entered_email>" }
```

Data Read

1. On following HTTP Request:

```
GET /users/name-lookup?q=User&n=10
```

where query parameters: `q` – is searching name of user and `n` is total records to be displayed

1. It should query database with following sql:

```
SELECT * FROM users WHERE name ~~* :name LIMIT :limit
```

where `:name` is concatenation of parameter `q` and character `%` (ex: `John%`)
and `:limit` is value from `n`

1. And then respond back all found rows and total matched records:

```
{
  "users": [
    {"email": "test@users.co",  "name": "John Doe"},
    {"email": "test1@users.co",  "name": "John Senna"}
  ],
  "total": 42
}
```

Requirements

You are free to choose any libraries you want for implementing that application.
There is a list of recommended libraries:

- Kotlin coroutines
  Library that unlocks asynchronous programming using coroutines (a.k.a. “light-weight threads”)
- Ktor Kotlin & coroutines based web-server
- Jasync
  Postgres/Mysql Asyncronous DB driver written in Kotlin, supports coroutines

Any form of automated tests are more than welcome (unit/integration/ …) and can score you some extra points!

Benchmarking

Once application is ready and perform all those operations correctly a benchmark process can be started.
Suggested way of benchmarking would be to use wrk tool
which allows to have a random test-set from some seed
Example scripts and benchmark commands provided below:

Write Benchmark

Command: `wrk -t17 -c400 -d30s -s write.wrk.lua --latency http://127.0.0.1:8080`

```lua
-- write.wrk.lua
local counter = 1

setup = function(thread)
    thread:set("seed", 42 + counter)
    counter = counter + 42
end

init = function(args)
    math.randomseed(seed)
end

request = function()
    local name = "User_" .. math.random(1, 1000000)
    local email = name .. "@test.email"
    local password = "Password_" .. math.random()
    local body = '{"name": "' .. name .. '", "email": "' .
. email .. '", "password": "' .. password .. '"}'
    local headers = {}
    headers["Content-Type"] = "application/json"

    return wrk.format("POST", "/users", headers, body)
end
```

Read Benchmark

Command: `wrk -t17 -c400 -d30s -s read.wrk.lua --latency`

`http://127.0.0.1:8080`

```
-- read.wrk.lua
local counter = 1

setup = function(thread)
    thread:set("seed", 42 + counter)
    counter = counter + 42
end

init = function(args)
    math.randomseed(seed)
end

request = function()
    local name = "User_" .. math.random(1, 1000)
    return wrk.format("GET", "/users/name-lookup?q=".. nam
e ."&n=10")
end
```

Results

Benchmark results should be written with full information about infrastructure where tests were performed

CPU, RAM, Disk, OS, Software including their versions.

Comparision

It would be nice to have different tools to be compared with each other. You can compare different languages, libraries and find pros & cons of each approach. This is not mandatory and in essence not the goal of the assignment. It’s however a way to score some bonus points :-)