# Introduction

## Coding Test

The following test is provided to you as a step through your application to Olympic Channel development department as part of the Quality Assurance Engineer role.

The exercise consists on the development of automated test suit of a small sample application backend as well as describing your approach for automated testing of a not yet coded piece of frontend application.

The test aims at providing an overview of your way of thinking, your automated QA skills and your general approach to software quality assurance and delivery as well as providing the basis for a conversation on the interview state.

# What you should have received

As part of this assignment you should have received the following files:

* OCS\_QA Test.pdf: This document
* Athletes-backend: A sample rest api server implemented in Node. Readme.md contains instructions for running the backend.

Note, the provided code is simple and far from perfect and just means to demonstrate how you would approach providing automated testing.

This test simulates a “half-cooked” software application in which the backend team has already provided a first version while the frontend hasn’t yet been developed. The initial specification for the problem has been described in the section “The problem” and is the spec used by the developers to develop the backend and the frontend.

For inspecting and working with the included database which serves as a backend, we recommend using SQLite Studio. We recommend you treat the included athletes.db database as if it was a “production database” and therefore don’t set your tests directly against it.

# What you are expected to produce and how long you’ll have

You’ll have a full week (7 days) to complete the assignment once you receive it.

The test should take from 2 to 6 hours to develop, the week is provided so that you can adapt the completion of the test to your schedule and timetable as you like. You’re of course free to dedicate as much extra time as you want to it.

The main purpose of the test is twofold. On one side you will need to automate testing of a simple REST API in a way that:

* Enables the testing of the provided functionality as part of a CI/CD pipeline.
* Enables modification and reusability of that test suit in case the requirements change (i.e. if new endpoints are defined or if the data coming back from such endpoints changes)
* Is repetable, that is, if there are changes on the environment database, it will not be affected by it.

On the other side, the frontend will need to be developed and automatically tested but it hasn’t yet been developed. In that way you are expected to provide a document named “Testing Strategy” (word, text or markdown) that:

* Describes how you would approach the frontend testing, specifically focusing on:
  + Tools you will use taking into account the frontend will be ReactJs based SPA consuming the REST API
  + Approach to testing, how would you get involved and in what point from beginning of the development to the end. Assuming the spec for the frontend has just been handled out to the team by the PO and it will be implemented in one sprint of two weeks, what would you involvement be, when and what specific tasks you will use.
  + Interaction with developers and developer code. How would your relation with the developers be and, specifically, what changes would you ask for in their code practices or structure?
* Describes any additional comments or remarks overall, including changes you would make (if any) to the REST API design or to the way it has been developed that would have enabled to automate the testing better as well as any remarks you feel prevented you from automating better.

The result of the test should consist on:

* A set of automated tests that hit the node server REST API and verify the requirements are correctly satisfied. The tests should be automated and runnable on a CI/CD pipeline. That means simple SOAPUI or Postman projects will not be accepted, your test needs to run from command line.
* Instructions on how to run the aforementioned tests. For the purpose of the exercise, you can assume the backend will always run at <http://localhost:3000> and hardcode it if necessary.
* The aforementioned document “Testing Strategy”.

Please zip all required files or else make it available as a private github / gitlab repo and give us access.

# The problem.

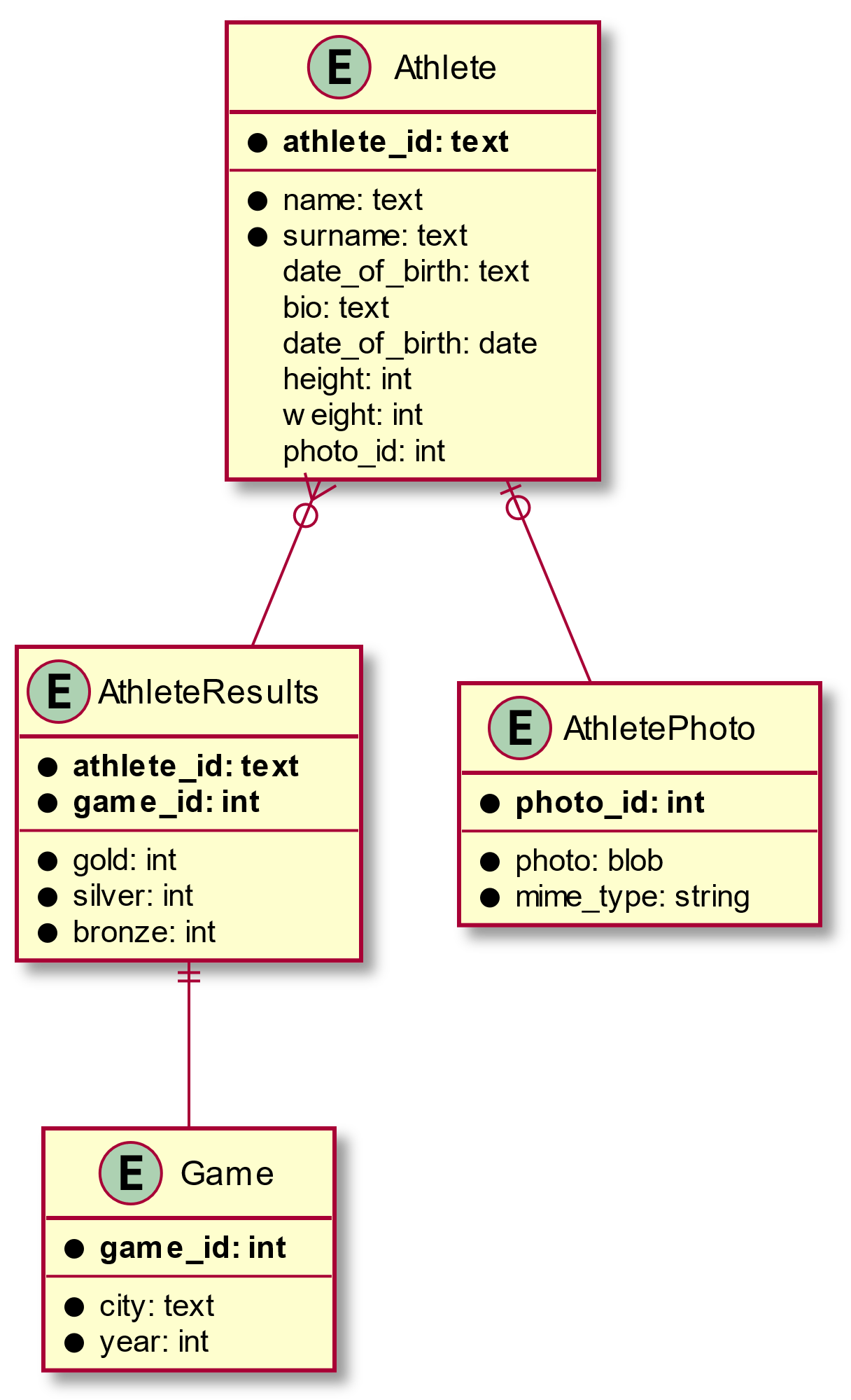
As interest grows in the Olympics, people are getting more and more interested on knowing everything there is to know about the athletes that compete on each of the games.

As such, the Olympic Channel would like to provide a new application that allow the public to get to know them through their mobile devices or through the webpage.

All the information required is stored in an SqLite database which contains all the information for the athletes. For each athlete the database contains the following information:

* Id of the athlete. The unique id of the athlete
* Name. The athlete given name
* Surname: The athlete family name
* Date of Birth. The athlete date of birth
* Height. The athlete height in cm
* Weight. The athlete weight in kg.
* Photo. The athlete archive photo. This will always have a square size.
* Bio. A long piece of markdown formatted text with the bio of the athlete
* Results. The list of results for all the different games the athlete has participated on
  + Game: London, Tokyo, Rio, etc
  + Gold. Number of gold medals won
  + Silver. Number of silver medals won.
  + Bronce. Number of bronce medals won.

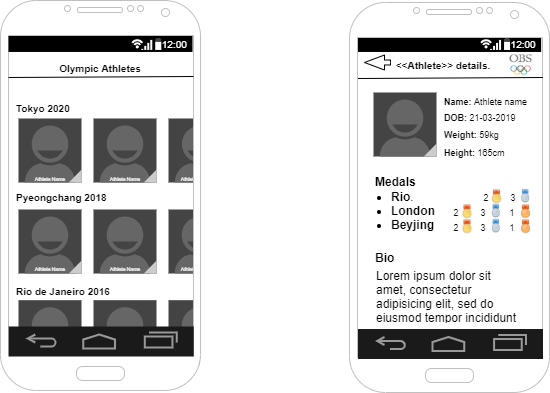
The following diagram illustrates the database model:



# Requirements

1. Upon opening the application, it shall display a list of all the athletes in the database (it can be paginated or infinite scroll) grouped by game from more recent games to older games.
2. Note an athlete may appear on several games in different orders depending on the result he/she obtained on the particular game.
3. Each “group” must be titled by the name of the game and the year of the game.
4. In each group there must be a horizontal scroll or carrousel with all the athletes ordered by global score in the current game with higher score first.
5. The global score of an athlete is calculated as follow:
   1. Gold Medal: 5 points
   2. Silver Medal: 3 points
   3. Bronze Medal: 1 point
6. The title of the application main screen shall be: “Olympic Athletes”
7. Each entry in the carrousel must contain
   1. A thumbnail of the athlete photo
   2. The athlete name and surname on the bottom part of the image
8. Upon clicking on the image of a given athlete the application will switch to a detail view of the athlete.
9. The title of the detail view shall be “<<Athlete name>> details” where athlete name corresponds to the name and surname of the athlete.
10. The detail view shall display the picture of the athlete.
11. The detail view shall display the information of the athlete (name, date of birth, weight and height) to the side of the photo.
12. The detail view shall display the list of all medals in all games the athelete has achieved.
13. The detail view shall display the markdown formatted bio of the athlete under the medals section.

The following is a sample mock up of the design for mobile web view or application.



# REST API Spec

/athletes

Returns a list of all athletes with no particular order. Each athlete on the array must have the following fields:

* athlete\_id: string; The id of the athlete
* name : string; The name of the athlete
* surname: string; The surname of the athlete
* dateOfBirth: string; The date of birth of the athlete
* bio: string; A mark down text with the bio of the athlete
* weight: number; The weight of the athlete
* height: number; The height of the athlete
* photo\_id: number; The photo id of the athlete

/athletes/:id

Returns a specific athlete given his id

/athletes/:id/photo

Returns a photo of a specific athlete in binary format with the appropriate content-type header and length

/games

Returns a list of all the games in no particular order

/games/:id/athletes

Returns a list of all the athletes that have participated in the game indicated by the id

/athletes/:id/results

Returns all the results for a given athlete. Each result entry must contain:

* city: number; The id of the game where the result was obtain.
* Year: number; The year the result was obtained
* Gold: number; The number of gold medals obtained
* Silver: number; The number of silver medals obtained
* Bronze: number; The number of bronze medals obtained
* Fourths: number; The number of fourth places obtained.

# How to deliver the results

In order to deliver the results please answer to the email with either:

* The solution attached, including instruction on how to execute. Please make sure you don’t include elements susceptible of being intercepted by the antivirus (i.e. executable binaries).
* A link to a private GitHub repository (or similar) with enough information in it to execute and test your solution (i.e. via de usual readme.md format on GitHub)