Java Technical Test

Scenario

This test presents a simplified micro service based architecture code implementation. The code solution contains unfinished functionality that saves blocks of arbitrary bank data to a persistence store. The persistence store is implemented as an in-memory embedded database. Candidates must complete the unfinished functionality through the 5 exercises below.

Guidelines

This test is to be done on the candidate's personal computer in their own time, it should take no more than two hours and candidates are allowed to use reference material.
You must not add any new build dependencies. Solutions must be written with best practice in mind, writing as little code as possible; reusing code and leveraging the Spring framework as appropriate.
This project should be treated the same as any production code you come across; code coverage must be high, test scenarios must be thorough, paying particular attention to the potential issues that the micro service / API architecture may introduce. Candidates can refactor existing code they feel is not production quality. Database transaction boundaries must be considered.
The final solution must compile, all tests must pass and the Spring boot application should start up. Your solution should be pushed to a personal GitHub repository, details of which will need to be emailed to us.

Exercises

Exercise 1

Update the client to push a block of data via an existing HTTP API on our test server. The end point to connect to is exposed in the ServerController.

Package	com.db.dataplatform.techtest.client.component.impl
Class	ClientImpl
Method	pushData
Package	com.db.dataplatform.techtest.server.api.controller
Class	ServerController

Exercise 2

Add functionality to the server to calculate and persist a MD5 checksum of the body of an incoming data block. The end point must return to the client whether the hash matches a client provided checksum. The system policy is that invalid data is wasted space.

Exercise 3

Expand server functionality to expose a new GET end point to obtain all persisted blocks that have a given block type. Update the client to call this new end point.

Exercise 4

Add a new end point to the server with an appropriate verb, to update an existing block's block type. The block to update can be identified uniquely in the persistence store by its block name. Add at least API input validation for the block name. Update the client to call the new update end point.

Exercise 5

In the existing server push data end point, in addition to persistence, add functionality to push data to the bank's Hadoop data lake. The URL for the data lake service end point is http://localhost:8090/hadoopserver/pushbigdata and it takes a payload string as the POST request body. The data lake has just gone live in the bank and we are their first customer; instability and long running calls can be expected.