Markr - marking as a service

You work for Markr, the latest hot startup disrupting the exam-marking scene. Markr has just raised $800mil in its Series B off the back of a huge deal with every school system in Europe & North America, and wants to pump this cash into producing some sweet analytics for exam boards to analyse overall student performance. You have been made responsible for building out the data ingestion & processing microservice, and you've been asked to prepare a working prototype to demo at the next board meeting (the board is kind of concerned about how much progress is being made, you see).

Problem Description

Students in the school district take multiple-choice exams using those shaded-bubble forms. When the invigilators collect the papers at the end of a test, they feed them into a super-legacy machine which reads the tests, grades them, formats an XML document containing a set of results, and then sends them to the ingestion microservice via a HTTP POST.

The format of the document isn't super well defined, because the grading machines were build in the early 90's, never documented, and nobody really quite knows what's going on in there. They're also hooked up to a bunch of other critical systems for the school district, so you figure you're stuck with it. It looks something like this.

```
<?xml version="1.0" encoding="UTF-8" ?>
<mcq-test-results>
    <mcq-test-result scanned-on="2017-01-01T00:00:00Z">
        <first-name>Jimmmy</first-name>
        <last-name>Student</last-name>
        <student-number>99999999</student-number>
        <test-id>78763</test-id>
        <summary-marks available="10" obtained="2" />
        <answer question="1" marks-available="1" marks-awarded="1">A</answer>
        <answer question="2" marks-available="1" marks-awarded="0">B</answer>
        <answer question="4 marks-available="1" marks-awarded="1">AC</answer>
    </mcq-test-result>
    ...more mcq-test-result elements follow...
</mcq-test-results>
```

Thankfully, the designers of the original system put in a `<summary-marks>` element to add up all the answer data. You're not quite sure if you can trust it, but your boss told you to go with what's in there for now while you smash out this prototype - so you can safely ignore the `<answer>` elements. You also know you've seen some extra fields floating around in here under some circumstances, but they shouldn't concern you - probably some gunk the reporting team needs in there. Finally, you're tangentially aware that there are some other kinds of XML documents that the grading machines make, so you should try not to get your wires crossed with them.

"At least it's not SOAP", you quietly whisper.

(P.S. - one of your buddies works for the local school board and got you a "sample" of some real-world data. It's attached in `sample_results.xml`)

Requirements

At Markr, people don't like to muck around - the most important requirements are therefore at the top of the list.

- The grading machine will POST these documents to a HTTP endpoint at `/import`. The body of the request will be XML file content. They come with a content-type of `text/xml+markr`. You did suggest SSL, but your boss said something along the lines of "won't be here when it's hacked...", so you guess it's OK to ignore that.
- You need to ingest these documents into some kind of persistent storage - Markr has been having some problems paying the AWS bill lately so you need to be ready for your instances to turn off at any time. Nobody from the DevOps team responded to you on Slack when you asked what's used around here, so you should just be able to use whatever you want, right?

For example, the following HTTP POST request should cause a student's test results to be loaded into Markr's storage.

```
curl -X POST -H 'Content-Type: text/xml+markr' http://localhost:4567/import -d @- <<XML
    <mcq-test-results>
        <mcq-test-result scanned-on="2017-12-04T12:12:10+11:00">
            <first-name>Jane</first-name>
            <last-name>Austen</last-name>
            <student-number>521585128</student-number>
            <test-id>1234</test-id>
            <summary-marks available="20" obtained="13" />
        </mcq-test-result>
    </mcq-test-results>
XML
```

- The visualisation team have said they want to query your aggregate data at the endpoint `/results/:test-id/aggregate`, and receive a JSON document with the following fields
    - `mean` - the mean of the awarded marks
    - `count` - the number of students who took the test
    - `p25`, `p50`, `p75` - the 25th percentile, median, and 75th percentile scores

Note that the visualisation team require these numbers to be expressed as *percentages* (i.e. as a float from 0 to 100) of the available marks in the test.

For example, the following HTTP GET request should cause the average results for the previous example's test to be returned.

```
curl http://localhost:4567/results/1234/aggregate
{"mean":65.0,"stddev":0.0,"min":65.0,"max":65.0,"p25":65.0,"p50":65.0,"p75":65.0,"count":1}
```

- Sometimes, the paper gets a bit folded up when the invigilators put in the scanner, and some of the questions get covered up (and so the score appears lower than it should). When this happens, they usually just scan it again and move on with life. Therefore, you may see a particular student's submission twice, and it will have a different score - just pick the highest one. You'll also need to do the same with the *available* marks for the test.
- These duplicate documents may come in a single request or in multiple requests.
- Sometimes, the machines mess up and post you a document missing some important bits. When this happens, it's important that you reject the *entire* document with an appropriate HTTP error. This causes the

machine to print out the offending document (yes, print, as in, on paper) and some poor work experience kid then enters the whole thing manually. If you've already accepted part of the document, that'll cause some confusion which is *way* above their paygrade.
- Although your boss *said* this was just a working prototype for the board meeting, you have a sneaking suspicion it'll somehow find its way into production. Therefore, you figure you should at least take a swing at error handling and automated tests. Goodness knows you're not getting paid enough for 100% test coverage, but it's probably best if you at least put some tests around the basics.
- The current visualisation solution generates printed & (snail) mailed reports overnight, so the aggregate fetching doesn't need to be fast. However, you heard on the grape vine that part of the big fundraising round is going towards building real-time dashboards to be displayed at City Hall - so it's probably worth having a think about that & writing a few things down even if the prototype implementation you build is a bit slow.

How we work at Markr

You know Markr are big on documentation - so you figure should include a README file that specified

- Any key assumptions you made about the problem and solution spaces. Your boss *hates* questions, so you should feel free to make (and document!) whatever assumptions you need in order to get the prototype off the ground
- A short description of the approach you took
- Anything you'd like to draw particular attention to in your solution
- Instructions on how to build/run it (the DevOps team aren't the sharpest tools in the shed, so spell it out for them)

You also remember a few other things from your induction week -

- You must **provide a docker-compose file** that runs your service. (meta: this is by far the easiest way for us to run/test your code. Please ask us for help if you've never used Docker before! There's also lots of tutorials online, e.g. https://docs.docker.com/compose/gettingstarted/. If you Google search for " docker compose file" you should find some starter guides!)
- Markr is a forward looking startup - they use the right tool for the job all the time! You can therefore choose whatever language, technologies and frameworks you need to get this done.

- Version control is a *must* - Ops needs to roll stuff back all the time, so it's kind of important they have the history
- Payroll around here is a little strange - they pay you in Coles Myer gift cards instead of actual money for some reason (meta: that means we'll give you a $200 gift card for your time completing this exercise - it's generally easier than giving you actual money because you don't need to wait for payroll, provide us with an ABN/invoice, etc.)
- Finally, you've got a lot on at the moment (you're in charge of the GIF Slack channel this week!) so you're going to try and spend about 2-3 hours on this. They *definitely* don't pay you overtime at Markr.

So, in conclusion....

Good luck, and get hacking!