Heart(h)stoneAssessment by Tom

Approach

I only had a few seconds to run through the specs and estimate when can I finish it. It's a complex task if attention is given to every detail.

I'm a big advocate of continuous feedback, so I rarely do this: Instead of asking questions from the Splendo staff members who are probably very busy, and we don't even know each other, I have rather made some decisions on a few things that I would normally ask. I hereby list a few of my decisions, these are mostly implementation / architectural decisions, and in most of these decisions I'm quite flexible to follow other standards / approaches with or without feeling the need to argue / reason.

Backend

Spring Boot

I have more than a decade of experience with Spring. In the last 2 years I've created 4 Spring Boot applications, 3 of them are in production. So it was a natural choice. I only shortly considered GAE endpoint API or JAX-RS without Spring Boot.

Google App Engine: Standard vs Flexible

As this was the preferred hosting, I gave it a try. It's different in the practice than what I thought by having watched several demos (Google Next, meetups, friends) since it's dawn. The first few deployments were Flex Instances, but it was bloody slow (7-9 minutes). Luckily I've managed to find a way to deploy it in a Standard Instance. Logging still doesn't work properly on GAE despite hours of trying different configs. I see either a good old classloader scrutinizing investigation or asking forums as possible solutions.

Gradle

For such small projects it may be a wiser choice to use Maven, as most examples are still in Maven. So why did I choose the harder way? Well, I've had my fair share of Maven hell on complex projects, and after reading the Gradle book ~2 years ago I was convinced that this is the way to go. The build file is definitely shorter and less error-prone just to mention a few benefits.

Design Patterns

Most of them are explicitly stated in the javadoc / comments. Some others: Singleton, MVC, Facade, Iterator (of course).

Runtime exception (wrap checked)

This topic has a long literature, and a long debate. I just follow the Spring style, because I like it. In most cases you can't really do anything with the caught exception and the best is to pass it forward seamlessly.

StringUtils not used

I've already felt some guilt of using Spring instead "poor Java" so I avoided adding more libraries that I usually use.

Data storage

Document DB / Object DB… No, a variable will do. I have experience with ElasticSearch, but I felt it a bit overkill here. MongoDB, PostgreSQL, Cloud Datastore are even bigger overkills for this project.

Object oriented filtering approach instead of string based / reflection based filtering

filter.setRarity(`"Legendary"`) instead of filter(`"rarity"`, `"Legendary"`)
This project is on the thin edge when both can be legit, I chose the OO version.

guard clause vs arrow code

https://blog.codinghorror.com/flattening-arrow-code/

non static

question of style… some methods could have been static… but what if they start to depend on instance variables

Jackson

Because it's stable, included in Spring and one of the fastest JSON parsers.

toString builder

I just used the Eclipse built-in, but there are great Builders out there… just another library

considered template method / visitor approach...

(registering and applying objects of the same interface) for these, but found it a bit overkill given the circumstances

- simpleFilterCondition(Card, CardFilter, Function<AbstractCard, String>)
- executeClassFilter(CardFilter, Card)
- executeMechanicsFilter(CardFilter, Card)

html header/footer copy-paste antipattern

I usually use thymeleaf (but I've worked with a lot of other template languages)

Language

initialize vs initialise. I've worked at British companies, so I'm not always consistent with the Spring log messages (US English).

Gradle wrapper committed

I generally discourage everyone from committing **binaries** into any **source** repository, but if the Google guys do this, https://github.com/GoogleCloudPlatform/app-gradle-plugin ... I thought I can also afford.

Cursor?

I argue with this functional requirement. Maintaining state on server-side should be avoided as much as possible. What would we do with the old cursors? Run a watchdog / eviction background job? Leave maintaining the scroll/page state handling rather to the client!

- It receives the *total* number of cards matching the filter
- It can count the returned items of the *list*

- It should know what was the *from* and *max*, when it sent the request
- So it can issue the next (paging) query if necessary
- No need for cursor on server side

TODOs?

end-to-end (frontend) testing, mocking
add all the nice coverage report / documentation stuff eg. spring-restdocs-mockmvc
paging in the dao instead of the controller is cleaner…
https://github.com/zkiss/gae-logging

User interface

Why have I chosen Vue.js?

There are plenty of JS libraries and frameworks to choose from, so it's a very valid question. Although I'm primarily a Java developer, I have years of experience with jQuery, but jQuery is the past, mainly because it's slow. When looking for alternatives I had found Vue.js to be a mature, solid, well documented, well-supported (not only on StackOverflow community) framework. To name a few of it's advantages:

- Performance
    - Blazing fast according to independent benchmarks
    - Low memory footprint according to independent benchmarks
    - Small bandwidth footprint
- All the traits of a modern (progressive) JS framework are presented out-of-the-box:
    - reactivity
    - components
    - two way data binding
    - templates

Some concepts in Angular and Vue are quite similar, but I personally find Vue slightly better.

Clusterize / lazy load

Too many data to be loaded at once, so I'm planning to use this technique that I had used many times as a former Flex developer. https://clusterize.js.org/#playground looks just to be the right tool. The vue-clusterize seems nice, but it has loads of dependencies and reported to be somewhat unstable.

Why is it my first project on GitHub?

Because I was lazy to export this small 10+ years old project of mine (downloaded by thousands, mentioned on the former Google Recaptcha contributors page, used by a few websites) https://code.google.com/archive/p/mailhide-tag/wikis/HowTo.wiki when Google Code was closed down :-(