BBC Technical Questions

by Steven Blowers

Question 1)

We're looking for people with a real passion for collaboratively creating great software. Please give an example of a software component you have designed and written from concept to deployment, outlining the steps you took.

Answer 1)

Whilst working on the Open Tree project at Findmypast there was a need to interact with an external API, FamilySearch.

It was decided that the functionality would best sit in a micro-service, as opposed to the front-end directly calling the API, providing the benefit of becoming a layer of abstraction between the two and also providing a basis to implement fallback measures if the API goes down.

Findmypast has an internal tool to scaffold micro-services based on templates, it sets up the deployment pipeline for the new service. Once we had used that tool, our micro-service was deployed in production, however at that point it didn’t provide any value.

What was left to do in designing this micro-service before it was ready for feature work was the structure. I felt less was more with a service of this size (micro), however there was some key concepts I wanted in place to ensure sanity as the code base grows. The main one being a separation of business logic and web router logic.

Question 2)

Using the example that you provided above, tell us about a significant decision you made to solve a technical challenge. Give details of technologies that you chose and why you chose them.

Answer 2)

Whilst implementing the micro-service, there was a decision to make on which language it should be written in, either Elixir or NodeJS.

Both are languages of choice for the company. Meaning there are internal tools to help facilitate developing in these languages, such as the scaffolding tool metioned in my answer to question 1.

Although both have templates the scaffolding tool uses, the NodeJS one had not been used by a feature team yet and was severly lacking in many areas. If it were to be used, it would mean extra working adding these in and pushing these improvements back into the template.

However, NodeJS has an SDK for the FamilySearch API, whereas Elixir does not. Working in Elixir would mean extra work implementing the functionality provided by the SDK.

The decision was made based on the value of the extra work to be done. Improvements to the template would benefit the entire company, whereas an SDK in Elixir would only benefit the team. Therefore, NodeJS was chosen.

Question 3)

Using the example that you provided above, tell us about how you ensured your software was fit for purpose and of high quality. What did you learn and what would you do differently next time to do a better job?

Answer 3)

It was confirmed that the new micro-service and template improvements were fit for purpose by working on a spike ticket to get the first piece of functionality done and review the functionality and developer experience.

Through using the SDK the development time was reduced and the functionality of the SDK was reliable.

The improvements to the template, such as adding git hooks, linting, code formatters and type checkers aided the developer experience. This was confirmed by positive feedback from other feature teams who used the improved template.

Through this experience I learned that when making decisions related to software sometimes situational benefits can outweigh technical benefits. Elixir is a functional language, that paradigm would have better suited the task at hand, however the fact that working in NodeJS provided a benefit to the whole company as a side-effect outweighed that.

Were I to make a similar decision in the future, I would focus more on situational benefits.