Summary

"A computer lets you make more mistakes faster than any other invention with the possible exceptions of handguns and Tequila" ~ Mitch Ratcliffe

Project Context

We were required as the development team to develop a smart phone application which visualised house price data in the local area. This built on work from previous weeks that required server set up and development of GPS smart phone applications. Each layer of the application was built up week by week, directed initially by the written customer brief and subsequently by recurrent meetings with the customer. These customer meetings allowed us to demonstrate our current progress and interpretation of the initial customer request. We could then adjust and steer further development to a completed product that was more in line with what the customer required.

To ensure we had a good foundation for a successful project we gave adequate time to scheduling and pre-build research. We used the weekly meetings to make group decisions on the architecture, the design process of the software and the testing strategy. We also utilised live streamed video-chat to perform live code demonstrations.

Our final software architecture is as follows. We chose AWS with a lambda serverless approach to limit resource use. The application front-end was built in React Native. A python Flask framework was used for the back-end and was used to take current longitude and latitude information, and use an API to extract and return all postcodes within a two kilometre square, centered around the provided longitude and latitude point.

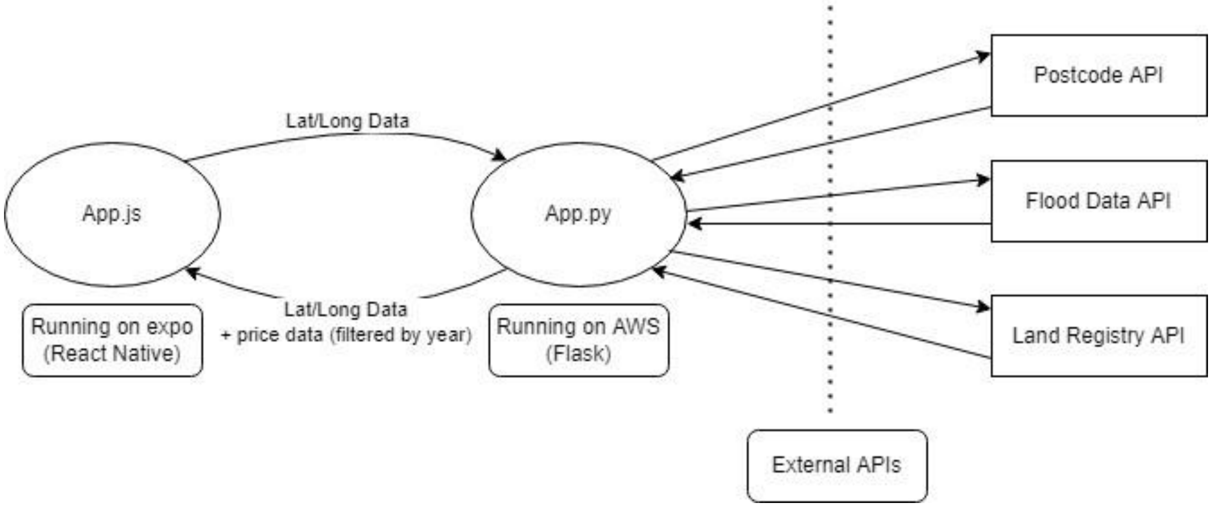

Figure 1. Diagram of software architecture

SPARQL queries were made to the National Land Registry database to obtain all properties under those postcodes and the prices attached to them. These prices were processed (processing was done in the AWS server) and applied as coloured circles of a fixed radius to the postcode locations on the map. Note that each postcode (which could apply to one of many houses with identical postcodes) is linked to a single longitude and latitude point on the map.
These circles formed the basis of the heat map.

To ensure the business logic on the backend is working as expected, we created a testing suite using the pytest framework. The testing suite consists of multiple unit tests for each of the dataprocessing functions. As our back-end relies on external API calls to retrieve our data, we mocked the API return values in our tests to eliminate any randomness. Lastly, we partitioned the input space such that all input possibilities are tested at least once.

Team Structure and Training Mentality

One of the strongest features of our software development process was our team structure and the prioritisation of training. Early on in the development process we identified two individuals with advanced web application development skills, and two individuals with less advanced skills but with a drive to learn.

Before starting any work on the deliverables, a code along session was led by a team member who was familiar with Reactive Native. This allowed the whole team to hit the ground running with our development task, as this provided the whole team with an equal footing with our chosen framework.

During the initial programming task two teams were formed, each consisting of one team member with advanced skills, and one team member with less advanced skills. The team members were matched on prior experience and on interest levels in various components of the development task (e.g. front-end, back-end, data processing).

Each week appropriate tasks were assigned to the separate teams depending on current project requirements and previous experience. This was done during a weekly team meeting where everybody was present either in person or via video-call. Each team had a “mentor” (the more experienced member) and a “mentee” (the less experience team member). The teams remained fixed throughout the project to allow the individuals to build a working and trainertrainee relationship. The variety in the nature of tasks was enough to ensure the “mentee” still received exposure and training on a wide range of components involved in building a web application.

For the initial set of tasks split between the two groups, the mentee worked directly with the group leader to learn the basics of web development, and the style in which their team leaders worked. This included elements such as language and framework preferences and personal coding styles and standards. The team leaders took their respective trainees through the tasks at an appropriate pace ensuring that the process was clear and that all questions were answered.

During the next team meeting, prior to the next set of tasks being allocated, a period of time was allocated for reflection on the initial mentor-mentee experience. We discussed the relative successes and failures and areas in which the experience could be improved in the following weeks.

For the next round of development, the tasks were subdivided by the team leaders after allocation. They delegated the less complicated elements to the mentees and supervised the execution of these tasks to ensure completion in a timely fashion. This was the first episode of supervised mentoring (in comparison to the previous week which had primarily been geared towards teaching alone). During this week the mentees were encouraged to lead their allocated sub-tasks but to work closely with their team leaders at all stages. Again, during the team meeting of the following week, the process was discussed and reflected upon, and suggested made to improve the process were implemented. These included creating a specific time and platform for an official mid-week meeting to ensure adequate progress was being made and creating official channels for communication utilising the Discord platform.

During the following rounds of development, the level of supervision was gradually reduced week by week, giving the mentees significantly more autonomy in the structure and implementation of their tasks (although supervision was of course never withdrawn completely). The weekly reflection process was used as a way to ensure no major issues were arising and that the level of autonomy was appropriate for the tasks in hand.

Not only did this process maximise the training opportunities for less experienced programmers, but it gave the more experienced programmers the opportunity to work on their teaching and mentorship skills, leaderships skills and time management skills that might be required in larger software projects in the future. From a practical point of view, we doubled the number of capable web developers within our team in a matter of weeks, giving us the opportunity to develop the more complicated components of our software in a timely fashion.

Code Review Strategy

On reflection one element of our software engineering process which could have been improved was our code review strategy. We decided in our initial meeting that code reviews would be every fortnight, and that it would be done in person, rather than using semi-automated or online resources. This decision was made because we believed the size of the code base would be manageable, meaning that more frequent reviews or excessive automation would not be required. We agreed on a schedule for the code reviews. We opted for this over any other strategy as we did not have the programming experience within the team to adequately implement "pair programming". We agreed that feedback would be given immediately post code review (conducted offline) and we did not enforce rules regarding committing before review. That is to say newly written code could be pushed to the main repository prior to formal review.

As the project advanced there was not sufficient time to work through the large quantities of code being produced whilst maintaining the schedule that we had originally designed. It was clear that too much code was being written for relatively complex tasks, and that fortnightly reviews were insufficient for this. It was also clear that the schedule was light on man-power, and even with increased frequency in reviews, the code would need to be reviewed in a modular fashion by more than one individual.

The plan to conduct the feedback process offline worked successfully, but it meant the team, in particular the less experienced developers, missed out on good learning opportunities. The act of doing it in person or as a live stream is reflective and educational and perhaps could have lead to reduced errors in the future code.

Our pre-determined coding standards were not specific enough for some of the frameworks we used in the final implementation of the project, and we had not accounted for the wide variety of coding styles within the group when creating these standards. This was confounded by using frameworks which the majority of the team had not used before.

The result of these issues meant that we ultimately put more stress on the testing process rather than locating and fixing the errors at the time of writing. In retrospect we would have made a number of changes to this process.

Firstly we would have addressed the imbalance between the amount of code that required reviewing and the time and man-power required to do it. These reviews could have been conducted on a weekly basis, to create smaller, more manageable segments of code, and a checklist of common areas for mistakes could have been used by each reviewer to more effectively target code snippets that were more prone to error (Pareto Principle). This process would have been reviewed weekly to ensure this was adequate.

Secondly, to optimise on the learning potential from these code reviews, they could be conducted online, and utilise a more formal feedback process. In this way constructive changes can be made directly to the code, but the feedback process can also be used to help adjust the programmer's style and prevent future mistake or coding style clashes.

Finally, a more structured set of coding standards could be drawn up to account for the different frameworks being used. This would also save time during the code reviews and provide everyone with a more rigid platform on which to review the code.

Despite these mistakes and the suggested alterations, the code reviews were still partially effective. We still do not believe automated reviews were required at this stage however with the growing code base this decision would be regularly reviewed.

Conclusion

Our overall software development strategy enabled us to deploy a successful mobile application in line with the specifications. We found using a teaching and training mentality very useful in developing our team and broadening our skill set. This allowed less experienced developers to build their skills in a safe and supportive environment and allowed our more experienced developers to enhance their teaching skills and their ability to manage and supervise more junior team members. Our code review strategy was initially acceptable, but as the code base grew and the project became more complex, we found that it was not sufficient and ultimately overburdened the testing process.

We found the underlying theme for a successful software development strategy to be regular reflection on the current strategy with small adjustments made weekly in accordance to the evolving needs of the project. In essence this is directly relatable to an "agile" working pattern. An agile methodology in the context of software development refers to constantly adapting the software production plan in accordance to evolving customer needs. This could also be extended to team management and the wider methodologies behind coordinating the project. Thus, elements of the process not directly related to software production are reviewed at regular intervals and restructured in order to maintain growth of the project in an efficient and meaningful way.