Werner Mostert
13019695

COS 731 Assignment 1 Retrospective

1. **Explain the process that you followed to add the feature to the existing application**

In order to add the additional functionality based on the reuirments received, it is important to ensure that the existing functionality within the system remains functional. An analysis was done as to what effect the next functionality can conceivably have on the existing in order to address the issue.

Additionally, leveraging existing functionality and frameworks within the system, a design was identified that would be both beneficial in terms of ease of development as well as structurally sound.

Lastly, component testing and regression testing was applied in order to ensure correct behaviour of the system.

2. **Describe the difficulties you faced when implementing the new features after the first phase was completed**

I actually found it much easier to implement new features given that there was already something to work with. One difficulty however was to define the data model. Originally after reading the specifications a rather strict data model was defined making use referrential integrity via database foreign keys. Only after implementing the data model was it realized that a Car could exist within the system without a User owning said car. Some difficulty was experienced in retrofitting the data model that was already defined. This is a typical example of not completely understanding the requirements before starting to code.

3. **Describe the aspects of the new features that you found easy to implement**

I was pleasantly surprised at how easy it was to do two things in particular. Firstly, at the ase of defining the new views on the Angular application. This is because it’s rather involved to set up a new Angular project and linking all the different components in a logical way at the start. Also, since the infrastructure for connecting to the database was already in place, all that was needed in order to accomplish further persistance of data was to write the domain objects and define the SQL queries that accompany them.

4. **Briefly describe the software architecture of your implemented system - if it has changed over time describe how it evolved.**

It was decided to go for a monolithics system. Monolithic in the sense that it is a single process application that has all the functional modules combined in one places where inter-module communication happens in-memory. A typical three-tiered monolithic system was implemented. The Presentation Tier (Front End) consists of an Angular 2 single page application (SPA) using popular web technologies such as HTML5, CSS, JS and TypeScript. The Logic Tier (Back-End) consists of a Spring Boot Java Application and the Persistence Tier (Database) consists of a PostgreSQL database.

The architecture of the system hasn't changed in the slightest since the inception of the project. This is due to the fact that the system hasn't sufficiently grown in complexity and size in order to validate a transition in architecture to something like an SOA or Microservices architecture.

Technology should be driven by business needs (requirements). Based on the requirements presented a monolithic architecture still addresses the functional and non-functional requirements.

5. **In what way was the architecture you have used in the first phase accommodating / hindering the implementation of the additional features? Give specific details**

The architecture was very accomodating towards adding new features in the second phase of the project. Due to the fact that the back-end application runs as a single process, it wasn't needed to go through the pain of setting up a new project. It was also accomodating in the fact that less integration work was needed. Since the data is persisted within the same datastore it makes it easier to establish relationships between the domain entities rigid way.

The layering approach that was followed by seperating the front-end, back-end and database was massively accomodating since each functional module of code was isolated and followed the single responsibility principle. This also allowed for components of the system to be inherintely decoupled since the network between the three layers acts as a barrier for couppling. A loosely coupled system in it's very nature makes it more maintainable and extensible. The value of this was releasized when adding the new features.

6. **Regardless of what architecture you have actually used, indicate what architecture you deem to be the most appropriate for the system you have implemeneted. Take the original specification of the system as well as the required changes into consideration in your answer. Give justification for your opinion.**

A monloithic architecture as was implemented is most definitely the best fit. Monoliths aren't always bad, even though it's given a really bad name in industry. That's exactly the point though. Monolithic architectures are greatly criticized due to the fact that they have a tendency to tighly coupled and slow. This behaviour only starts manifesting though for very large systems. Monoliths have the advantage of being quicker to develop. The amount of integration complexity due to netowrk decoupling between services on the same architectural layer is zero. Attaining modularity in a monolithic system is very possible, just hard to do when there multiple developers working on the same code base.

Let us contrast this to implementing a microservices architecture. Microservices simply aren't a good fit based on the requirements and environement. Microservices are brilliant when there are various teams where each takes ownership of a microservice. Microservices allow for massive fault-tolerance and scalability, both of which aren't requirements in this system. If a disaster recovery strategy were to be ccreated for the system, it would likely have a return time objective (RTO) of hours or days even. Microservices also have a direct impact on integration effort, which is often very high during the inception of a project and only once the metaphorical blood has been spilt does one see a return on investment (ROI) for a microservices architecture.