LOST MY PET

EVALUATION REPORT

Piotr Przechodzki
Graded Unit Project

TABLE OF CONTENTS

INTRODUCTION

This report is an attempt to document and summarize the work on the project called "Lost My Pet". I would like to describe each step of project development and my progress on the task by describing the analysis and design stage and publication of my working log describing what has worked and what was difficult or went wrong. I explained all my challenges and problems but also how I managed to find and implement the solution. Unfortunately, due to time constraints, the number of functions to implement and the level of project complexity, I was unable to implement the client version for Android devices. Apart from this one task, I managed to implement all the functions of the web client and create fully working endpoint service. Therefore, at the end of this report, I described my conclusions, insights and future advice for me and the people who will read this document.

PROJECT OVERVIEW

The City of Hillwood is a mid-size city based in the UK. For some time, the City Council was looking for a system that would allow residents to add announcements about lost or found pets. At this moment, there is no any solution for that and if anyone wants to report a pet then need to phone directly, email or contact in person to one of many animal shelters that operate in the city. Then shelters usually contact (by email or phone) with others pet rescues and share information about lost or found pets. This method is slow and not guarantee that all details of lost pet will be passed correctly.

The City of Hillwood is a mid-size city based in the UK. For some time, the City Council was looking for a system that would allow residents to add announcements about lost or found pets. At this moment, there is no any solution for that and if anyone wants to report a pet then need to phone directly, email or contact in person to one of many animal shelters that operate in the city. Then shelters usually contact (by email or phone) with others pet rescues and share information about lost or found pets. This method is slow and not guarantee that all details of lost pet will be passed correctly.

Work on the project began in mid-December 2017. From December to March, I worked on the investigation stage, interview questions, functional and non-functional requirements and UML diagrams to better understand the client's needs. In the meantime, I have developed several prototypes of the future application to better understand the technology in which I want to compose the project and to better estimate the time needed to complete tasks.

ANALYSIS

The goal of the analysis/planning stage in the development lifecycle is to define the scope of work, estimate time consumption, costs, time of completion and planning work on the project. I started my work by making Planning Report that would include a project description, customer requirements, user stories, and a description of the different development approaches and system implementation technology. At this stage of the work, I focused mainly on getting to know and analyse the needs and requirements of the client. For this purpose, I prepared a number of questions for the interview (Work Log, 20/12/2017) to get information about the most important features and system modules as well as user groups and attributes (roles). My knowledge about the questions that I should ask was taken from various sources. As I do not have experience in this, I mainly learn from academic projects and from websites (eg https://www.techrepublic.com/article/questions-you-should-ask-your-clients-before-you-take -on-a-project). I did not have any problems with creating questions. It seemed pretty obvious what I wanted to know, however not all information achieved this way at this stage of the work was exhaustive (level of details), but the information obtained gave the big picture of the whole project so I was able to create a description of functional and non-functional requirements. Thanks to the list of functional requirements, I could refer at all stages of my work to the requirements and services that the system should offer, how to respond to specific input data and how to behave in specific situations. On the other hand, non-functional requirements have allowed me to systematize knowledge about how the system should work, what interface should it contain, who it is intended for (roles) and what security requirements it should have.

Very often, as programmers, we forget that the project we write is to serve first of all the needs of the end user, and not the idea of writing the application. Sometimes the function in the code looks awesome from the programmers' point of view but not practical from the user point. That's why user stories turned out to be the most helpful in my case. The user's stories were not difficult to create. It seemed quite logical to put them together and formulate them. I obtained specific information about what the users expect and what they need from the system. The simple basic structure of the user story like: "As a <*specific user, role in the system*>, I want to <*need*>, so that <*problem to solve, a goal to achieve*> Acceptance Criteria/Conditions of Satisfactions", can be very helpful during the analysis stage. User stories helped me create functionalities. Guided by them, I knew exactly what the end user expects, thanks to which I could save time because I created only those components that met the user's requirements. I did not have to create a repetitive code that would not be useful later.

The greater part of the analysis took me preparation of the UML diagrams. Before creating diagrams for this project, I must admit that I did not understand the sense of building them before. It seemed for me as a waste of time, but now I see that with such large and complex projects it is better to spend a few moments more to create UML and analyse it rather than later to fix not working functionality or bad. For the analysis phase, I have created UML diagrams including:

- Initial Use Case Diagram
- Extended Use Case Diagram
- Activity Diagram

UML diagrams were particularly useful during the implementation phase of the project when I had to plan and think about each component. It saved me time and perhaps later frustration from a poor understanding of customer requirements. They were a clue as for example Activity Diagram, Sequence Diagram and Communication Diagram showed me step-by-step how system functions are to be implemented. The Use Case Diagram and Use Case Description allowed to trace not only the "happy path" but also think about designing all possible paths that can happen during the process.

A helpful tool in the analysis stage has been uses of the Kanban board. For this purpose, I used the popular Trello application to create a simple table with three columns: *To-Do*, *Doing* and *Done*. In total, I created 3 boards for each application stage - documentation, creating REST API service and creating the web client (MVC). Each of the cards contained a name with a short description of the task and the date of start and end of the work. I have to admit that the moment when I move the card from the "Doing" column to the "Done" is very satisfying, motivates me to continue working and gives a feeling of progress in building the application. I used the Kanban board regularly in the implementation phase, but when it comes to documentation, I sometimes forgot to record something on the board because the documentation has fewer elements and is less complicated than dozens of modules and bugs that need to be written or correct, so I did not care too much about it. I did not treat the Kanban board as a schedule, as it was difficult for me to estimate the time of doing the task. Maybe it's not professional, but I knew that now I will be implementing this function and I have to finish it as soon as possible. If I did it faster than I estimated, I was taking the next task. I am simply not good or have no experience in estimating time. I treated Kanban more as a visualization of work and motivator (dropping cards from "Doing" to "Done" - priceless!).

In the matter of choosing a development approach, I had a choice to work in Scrum, TDD or Object-Oriented Analysis and Design. Unfortunately, I do not know much about the Scrum methodology and all that I know is mainly theoretical knowledge. That is why I did not choose this approach. Another option was to choose Test-driven development (TDD). Writing tests is one of my favourite activities, but sadly, the TDD approach is labour-intensive and time-consuming. It allows you to produce high-quality code, but at the cost of time. I was afraid that working in this approach I would have to sacrifice some functionality to complete the project. That's why my choice fell on Object-Oriented Analysis and Design. This approach is familiar to me and I have worked with it previously on another project. Also, at the College, I was prepared how to work in OOAD. This methodology is easy to use, cheap and best guaranteed the completion of the project.

The last thing I had to think about was choosing a work environment. My choice fell on Java because I feel confident working in this programming language. It is also one of my favourite technology. Java allows using tools such as Spring Framework to write from the beginning to the end the entire backend and frontend. Although from today's perspective, I would

wonder whether to choose Java only for backends, and for the frontend one of JavaScript frameworks such as Angular or React. The reason for this decision could be justified by the fact that JS is doing well as a frontend tool and is much more flexible than Java and could allow me to write better-implemented function.

The other technologies I considered were C# with ASP.NET and Python. I know C# well enough to write a project in it, but I want to develop my career in the direction of Java backend developer.

I have no knowledge of Python language to write this size and complexity project. I would have to learn it from the scratch which would not be the best idea because I could not finish this project.

DESIGN

The next step in the development of the system is its design, which is to determine the overall system architecture and requirements for its individual components. For this purpose, I produced 13 layouts (Work Log, 15/03/2018) using a whiteboard and markers. I did not use any special software because I wanted the idea for the layout to come directly from me, based on the user requirements, and from the imagination I have about the final design of the application. When I was making a sketch in my head, I saw more or less how I want it to look. Looking back at how the application looks now, and how I planned it in the layouts, I can say that practically most forms do not differ much, because thanks to the experience acquired in previous projects where I created mobile applications, where visualization is especially important. A good visualization of the project supports the organization of elements and helps to not forget about anything that must be implement. Screen layout gave me a good basis to understand how the application should work and what the graphical user interface should look like and what should be included in.

For the design phase, I have created UML diagrams including:

- Class Diagram
- ERD Diagram
- State Machine Diagram
- Sequence Diagram
- Communication Diagram

Class diagram proved to be the most useful in the work on the project, which helped me to understand the static relationships between elements (classes) and what names, attributes and operations should be included in the given class. Many times, during my work I had to create a class diagram from the beginning (Work Log, 18/04/2018) because during the implementation process it turned out that something does not work as expected or does not meet all requirements or was incorrectly designed. For example, I have redesigned Pet entity model and divided it into Dog and Cat subtypes which inherit from Pet. This seems to be a more proper way of design after adding the breed property. This will prevent from accidentally assign of dog breed to a cat object and cat breed to dog object (Work Log 29-40/03/2018).

As I mentioned previously in the Analysis description, if I had only spent a little more attention on the creation of diagram, I would be able to save much more time during the implementation phase. I must admit, however, that I became a fan of UML diagrams and began to appreciate their roles. My ignorance probably resulted from poor knowledge of this solution or/and a desire to start coding.

Grady Booch which one of the UML creators said:

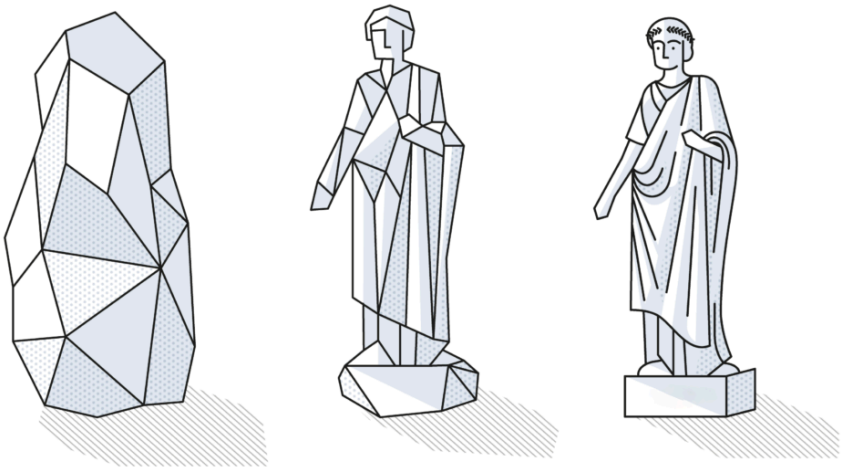
"Modeling is a central part of all the activities that lead up to the deployment of good software. We build models to communicate the desired structure and behavior of our system. We build models to visualize and control the system's architecture. We build models to better understand the system we are building, often exposing opportunities for simplification and reuse. And we build models to manage risk." (Booch, 2005)

Another useful mechanism is the normalization of the database. I was commissioned to normalize the example database of an online store model to better understand the process of eliminating any redundancy from the database.

I also made a sitemap that served to me as a signpost or logical representation of where to position what functionality in order to be understandable and easily accessible to the end user. It gave me an idea of how the user would have to navigate the site and whether it would be easy and intuitive. In addition, it indicates to the user what is on the page relevant and where can find the information that needs. I must also mention how helpful it was to create a detailed requirements specification where a description of each page was found and the requirements for the minimum input and processing for the system. It was neither easy nor difficult to work on this task, it required the sacrifice of time and the consideration of the requirements of each of the components. However, this has made it easier for me to implement the desired functions and avoid situations where I work on some functionality, which later turns out to work quite differently. In this regard, I did not have any problems and the documentation was clear and understandable which allowed me to speed up the work on the building of the new functionalities.

The last thing I need to mention, and which was probably the most helpful, right after UML diagrams, was to create dozens of working prototypes of functions that I could later implement in the system. In prototyping, I adopted a strategy for creating small, specific and "dirty code" mechanisms of various functionalities such as logging, registration, publication of announcements, deletion, editing, messaging system, etc. The disadvantage of this may be the amount of the time that we must devote to build subsequent prototypes. The advantage of prototyping is the ability to test functions on a small scale, without the influence of other defective or malfunctioning system mechanisms which turns into a better quality code. This procedure is referred to as system prototyping and it is used in Scrum teams working in the Agile methodology, and it is nothing more than an incremental approach. The incremental development of software is based on a clear vision of the product and the implementation of which is carried out in repeatable steps. The effect of a single step is a small, working and finite product increase. The concept of building a small, test application is the implementation of the smallest possible step that will allow us to test the implementation of the system's function (modules). This approach is also called MVP - Minimum Viable Product. If during the analysis phase, the prototype turns out that it does not meet the system requirements, then we can reject it or if it meets the requirements, we can start the process of optimization and implementation of the code.

Iteration model on the example of creating a sculpture:



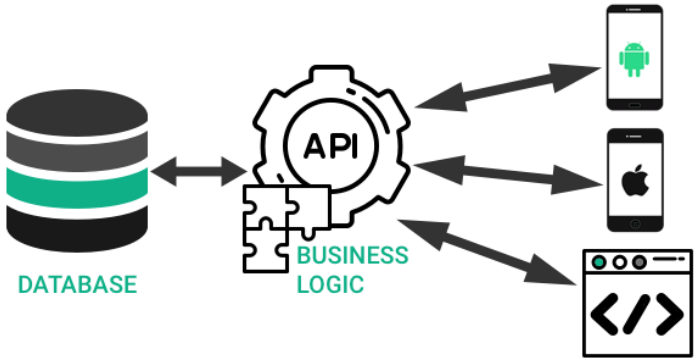
Thanks to prototyping I knew what to delete, what to change and what to add. I was not worried that by adding new functionality, the rest of the system would stop working. This experience has taught me the simple truth that programming is the only job where the ceiling in the lounge can collapse when we paint a kitchen wall.

IMPLEMENTATION

Implementation is one of the final stages of the software development process. Within it, an implementation of the system and code are created. I decided to implement the project using a very popular and mature programming language which is Java because I have 2 years of experience in this language, although it would not be a big problem to me to implement the same solution using C# and ASP.NET MVC, but I want to develop my career plans towards as a Java Developer that's why I chose this language and not any other one which I described in the Planning Report. As a programming environment, I chose to work with IntelliJ IDEA created by JetBrains (this company is responsible for the creation of Android Studio and the new but very popular Kotlin language). It is the most powerful and advanced IDE currently available on the market, used by many professional and companies. As we can read from HP case study: "The company gets optimum results by letting developers pick their own tools. When a top Java developer tried and loved IntelliJ IDEA, word spread fast. Now upwards of 70% of this geographically dispersed development team has made IntelliJ IDEA their Java coding tool of choice." (JetBrains s.r.o., 2010) Advantages such as increasing productivity, better work with keyboard shortcuts, managing settings or "you don't have to walk through wizards or fill huge forms to get things done" (Cheptsov, 2014), are described in more details by Andrey Cheptsov in the article - What Makes IntelliJ IDEA Different? (https://medium.com/@andrey_cheptsov/what-makes-intellij-idea-different-5d6333621ef2) So, IntelliJ IDEA is my choice because I'm already familiar with it and during my job, I'm using a lot of keyboard shortcuts that simplify and speed up my work.

To perform the task entrusted to me, I decided to create a system based on REST service. Because the initial plans of the project assumed that there will be two separate applications (clients - web and mobile app) and they will use the same data source - endpoint. It seemed natural to separate the entire business logic and create a separate service that could handle the queries sent by clients and communicate with the database. In a nutshell REST (**RE**presentational **S**tate **T**ransfer) is not a protocol but an architectural style. It describes how a system (server) can communicate with a client using HTTP methods to exchange data in JSON, XML, text or any other format. There are no strict rules describing how to build and how the REST Controller should look like but there are a few characteristics and guidelines that we need to keep in mind. This way I standardize all operations and save the time because I don't have to code twice database communication layer and business logic. This strategy allows to build on the top of same basic conditions, app on various platform. According to Nordic Apis blog these are tree main benefits of building the solutions using backend as a service (Riggins, 2015):

1. Eliminates redundant stack setup for each app.
2. Eliminates boilerplate code.
3. All within one model.



Unfortunately, due to time constraints, the mobile application could not be completed. However, the system is fully functional and the REST API documentation written by me using the JSONDoc tool provides all the information about exposed HTTP methods (GET, POST, PUT and DELETE) needed to connect with any other application.

Another step of creating the project was to choose the platform on which I will build the whole system. The most popular is Spring Framework. Spring is a platform composed of many projects, which is dedicated to creating applications in Java. Its key element is the dependency injection container, but over the years Spring has gained support for many technologies and is today one of the key elements of the entire Java ecosystem. I decided to use Spring Boot which is a convention-over-configuration solution based on Spring Framework. The main benefits that Spring Boot provides are:

- Easy to start - the Spring Boot package has a built-in server (Apache Tomcat) and other necessary components that are needed to run the application.
- Automatic configuration - no additional configuration is required to run the main application.
- Speed - creating applications using Spring Boot is simplified which translates into a faster and easier development process.

Besides, I've already used in a few previously project Spring Boot and it proved to be a powerful and irreplaceable framework. Thanks to this project, I have established my knowledge about the Spring Framework environment. I have learned new modules like Spring Email, which I have never used before and I gain more experience with using entity mapping in the Hibernate.

Another step was to choose how to communicate with the database. During the prototyping I decided to stick with Hibernate ORM (Work Log, 10/03/2018). Hibernate is the most popular library for object-relational mapping in Java which uses JPA (Java Persistence API) - a Java EE specification. Hibernate provides 4 basic methods to perform simple CRUD queries on entity objects and special language HQL (Hibernate Query Language) or its standardized version of JPQL (Java Persistence Query Language). These languages are very similar to SQL, but the difference is that they do not operate on tables, but instead we use object notation. The overall reason for choosing the Hibernate was my personal preference - I had always used Hibernate with Spring and all tutorials and official documentation are based on the

Hibernate. Besides this ORM tool is an industry standard in Java world and is same as important as Entity Framework for C# (ADO.NET). Actually, there is no any significant competition for this tool. For the purposes of the project, this solution was more than sufficient, and work in this environment did not bring any major problems. Most bugs were solved with the help of the huge community (various web pages and StackOverflow) that is around the Hibernate and JPA. Most of my problems were that I'm not so familiar with these tools. The problems I came across mainly concerned the creation of associations between tables (Work Log, 19/04/2018), unsaved transient instance (Work Log, 20/04/2018), the problem with mapping polymorphic object models – could not get the type of the sub-type (Work Log, 17/05 / 2018) or the well-known N+1 SELECT which is the issue that mainly occurring in the Hibernate.

How did I resolve these difficulties?

1. The problem with an association between tables consisted in a poor mapping of the many-to-many relationship (after the preceding change of the Notification model from the one-to-many relationship with User). I found the solution on this page: http://www.codejava.net/frameworks/hibernate/hibernate-many-to-many-association-with-extra-columns-in-join-table-example. I had to use many-to-many mapping with a separate primary key for the join table, instead of using the two foreign keys as a composite primary key. But at the end, I rebuilt the whole Notification model and I don't need this solution any more, but I used the same technique to solve mapping of a relationship between User and Role - I created a join column UserRole with an additional field.
2. Again I had a problem with Notification entity. I've received "unsaved transient instance" error, which means that Hibernate is trying to save the object that is associated with another object which is not yet saved in the database. This occurred when I tried to add user notification about a new announcement at the moment when I was saving the announcement but still it was not saved. I could solve this problem using two options:
    a. Write an additional function which will perform in the first line save the announcement, get the respond to receive the new announcement ID and then check the database according to the logic to which user should get the notification and finally insert notification associated with the announcement by ID. But what would happen if I had to do many such operations, maybe even more complex on various functions? Would I have to create additional extra methods? That's why I chose another solution which is AOP ...
    b. The second solution was to use the Spring built-in module for aspect-oriented programming (AOP). This programming paradigm allows running any code before, during or after performing some other function. The methods after which the code will be run can be many and to describe them we use Spring @Pointcut annotation. This solution is more elegant and readable, it also allows to expand the system in the future and helps with maintenance. AOP

was already known to me so the choice was not accidental, and by the way, I gained more experience in this field.

3. The problem with mapping polymorphic object occurs when I tried to convert fetched from the API, single Announcement object, into one of the child objects - Lost or Found. The problem did not occur when I was trying to convert an object in the arrays of the Announcements. The java instance of operator did not recognize the class, and the casting operation threw an exception. The problem was that the information about object class type was not transferred (between the client and the backend) in the JSON document. I could not find any solution, but I found a possible reason on StackOverflow (https://stackoverflow.com/a/39479844/6938943), where one of the users wrote that @JsonSubTypes wants an array of sub-types, not a single value - so that's why I was not able to get instance of a single object. To fix this problem I added a hardcoded getter for a post type field into REST API. Lost announcement is returning the string 'lost' and Found class returning 'found'. I don't know is this is a right way to solve this problem but it works fine and I couldn't find any working solution. So, I figured out my own.
4. With the Hibernate N1+SELECT problem, I've faced many times before so I already known the solutions. The easiest is to simply add to the @JsonIgnore annotation on a property in the entity class, which caused the problem. Alternatively, @JsonProperty(access = Access.WRITE_ONLY) can be used as well. Both annotations are used to exclude a property only from JSON deserialization.

At the beginning of work on the project, I decided to do a web application based on the MVC pattern. Model-View-Controller is probably the most-used design pattern at the moment. Almost every application, especially web applications, uses it. The most important advantage of the MVC standard is the hermetic model. Business logic is isolated from any View technologies or protocols for handling requests sent by users. For this purpose, I used the Spring MVC component. In the test project it was a good idea (Work Log, 23/01/2017), but if I could do it from the beginning now, I would probably choose something lighter, like Angular or React. Spring MVC is a large module, and not all of the elements I wanted to implement have been implemented due to technological limitations, for example, I wanted the number of messages received to refresh in the navigation bar every time a user receives a message. I could not do this and I didn't find any solution on the websites such as StackOverflow, GitHub Community Forum and by searching on Google. I do not have any experience in JavaScript but I am able to read and understand the code written in this language. In addition, I wrote a few simple functions in JS without which I could not keep the UI of the web site as I planned (for example buttons to choose the type of pet, breed or gender).

For the layout I used HTML5, CSS and Bootstrap Framework. In previous projects, I already had contact with these technologies. Bootstrap is a framework used to create responsively user interfaces on mobile devices and computer desktops. When designing a page with Bootstrap, I could choose which element to use, what grid layout and what CSS class (Work Log, 05-09/05/2018). More importantly, I was sure that the selected elements would not

interfere with each other. It's like laying out a puzzle, except that in this case, each puzzle suits the other, no matter what you choose. I really enjoyed working with Bootstrap and for building the grid layout I've used buildbootstrap.com tool which can save time and better plan the UI grid. I think that using Bootstrap saved me a lot of time and frustration, as I feel more comfortable with backend technology than the frontend. Thanks to Bootstrap, I did not have to focus on the frontend to much and I could improve the endpoint service.

From the system security side, according to the requirements and good practice, all user's passwords have been encrypted with salt using the BCrypt function, which is a part of the Spring Security module. Other functionalities that were not included in the requirements and which I wanted to implement were the use of SSL (Secure Sockets Layer) between the database and the REST API service and the use of the JWT (JSON Web Tokens). As the name suggests, this is the type of token stored on the client side and allows authentication of the JSON documents (which I'm using to communicate between client and REST service) using the key on the server side. Unfortunately, this has not been implemented. I preferred to focus my time on other problems as SSL and JWT were not part of the client's requirements, but only my need to learn something new and carry out tasks in accordance with good practices.

One additional functionality that I planned to complete and I couldn't finish (besides creating a client for Android phones) was to create a real-time message system, where users could be able to send conversations between other members like in online chat. I've completed the same functionality in the other project in the first year at the College, but this time I could not finish it because of the time constraint (Work Log, 19/05/2018).

For the bug tracking and management of the implementation phase, I used the same Trello storyboard that I mentioned in the Analysis part of this report. I gave a proper description for every bug that I have found. If the bug prevented me from further work on the project then I tried to immediately fix it. If not then I put the bug on the card (to not lose the focus on what I'm doing now) under the To-Do column and add a label with colour which gave me a simple information how complex is this problem. From green - which means easy difficulty to red - its means a hard or serious problem. This "colours system" is my idea and I had already been using it in previous projects. It was working well previously and now allowed me to better organize tasks and time.

The last thing that is worth to mention is the use of the version control system when working on the project. There are many services that offers hosting for version control. The most popular are GitHub, Bitbucket and SVN (Apache Subversion). I already have an account and using for last two years GitHub, so I decided to create a repository for this project. At this moment the repository has more than 90 commits. I never had a situation where I accidentally deleted an entire project or some part of it, so I did not need to recover the data, but it happened to me that some functionality worked a few weeks back, and now it is not working. Thanks for using the version control I was able to find the problem and browse for the working code in one of the previous commits. Besides, I think that using version control gives a peace of mind in the event of a crash. It also allows us to experiment with the code and even if we break something, we can always download the previous working version. Also,

for the backup, once in a while, I copied all project files and all documentation to an external drive and on my Google Drive account.

TECHNOLOGY STACK

The whole project was written using Java 1.8 and Spring Boot Framework 2.0.0 version.

The backend (RESTful API) was created using:

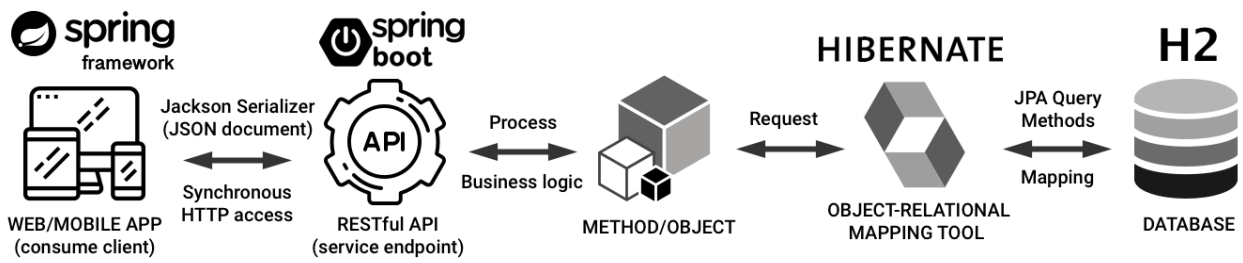
- Spring Data JPA
- Spring Email
- Google Guava (helper library for java.lang.String API)
- Hibernate ORM
- H2 Java SQL RDBMS
- JSONDoc (document and expose REST API)
- Testing tools:
    - JUnit 4
    - Mockito (mocking framework for unit tests)
    - Hamcrest (flexible matchers expressions of intent)

And the web client with:

- Spring MVC
- Spring Security (with BCrypt password encoding)
- Thymeleaf (Java template engine)
- Bootstrap 4
- Apache HttpComponents (creating and maintaining HTTP protocols)
- Apache Common Langs (helper utilities for the java.lang API)
- Apache POI (API for Microsoft Documents to generate Excel reports)
- iText PDF (PDF generation and manipulation tool)
- Ocp PrettyTime (java.util.Date API helper library)

In both projects, I've used Apache Maven tool for build automation and Apache Tomcat Server.

High Level Process View



Most of the technologies used by me in the project and listed on this list are known to me at least at the basic level, so using them in the project allowed me not only to quickly start writing code but also to increase the knowledge and gain new experience. The use of such libraries as Apache Common Langs and Google Guava made it easier for me to work with Strings variables by completing the missing functions in Java, such as comparing variables or detecting empty String.

Spring Email is the component I used for the first time. It was not difficult, and at the same time, I learned how to use the SMTP protocol to send emails.

Apache POI and iText PDF both are responsible for generate a report view. They are well known to me thanks to which I could quickly (in a few days) implement the report generation functions.

Other tools such as Hibernate, Thymeleaf or Bootstrap have also been used before, which allowed me to write better code in less time. Thanks to using all these tools, I learned new techniques and I'm now more confident in their practice, but it was not a work from scratch, where I would have to learn all the basics.

TESTING

Testing is the last stage of the development lifecycle and it is the process of verifying the correctness of the software operation and providing information about it. I divided the tests into two phases. The first one is testing the REST API service, and the second is testing the client's application. Because the entire API logic can be tested by testing the web application, because all CRUD method queries we use will have to be sent to the backend anyway, so I decided not to duplicate the test documentation and create only test documents based on the web client. However, the API service has been tested additionally using unit tests.

Unit testing is a code (method) that executes a different part of the code under controlled conditions. Its task is to verify that the code being tested works correctly. To do so, I had to provide a sample input data and the test performs some instructions. Then it checks whether the result of the action is consistent with expectations. I used the most popular Java testing framework, which is JUnit. This is not the first time that I used this testing framework. Previously, I made a prototype of a very simple REST service, where I tested the capabilities of JUnit on all CRUD operations (Work Log, 24/03/2018). The results were satisfactory, so I decided to test the backend with this method. I started to perform the test from the service layer which is responsible for the business logic abstractor and depends on Repository and DAO (Data Access Object) module. It is also injected in REST controllers. To test this application layer, I needed to create mock (fake) objects and simulates classes without integrating into the database structure. And again, I had used the tool that I already familiar with - Mockito. Using Mockito was easy for me and test are made very quickly. Almost always after adding a new service class, I wrote the test for it. And so, I tested a total of 11 service classes. I made 47 unit tests on them, testing all possible options. In this way, I received a result of 96% coverage of methods and 95% coverage of the code line. All tests pass.

Next, I've tested REST controller classes. Same this time I tried to write the unit tests after each time I created a new class. The REST controller class is responsible for handling request and responses. So, the best way to test it is to use integration tests. Integration tests are designed to test all or several components working together. For example, we can test implementation of databases, infrastructure, interfaces, system configuration and configuration data. Comparing to mock test, this time it was more tricky. I had to create test objects in the database to perform the tests. To do so, I used the H2 database in-memory configuration mode. Whenever I started the REST API service through specially configured files, I automatically added (inserted) the same test objects, so I could many times perform the test using same data. I tested all of the 8 controller classes and received a result of 100% coverage of methods and 99% coverage the code line. I wrote 90 integration tests. All pass.

When it comes to testing the MVC application, I decided to run the functional test (black box test) as I wanted to make sure that I have implemented all required functions. For this purpose, I created the test plan, test log and test cases documents (Work Log, 15/03/2018) that was provided to the City Council. I made the test cases based on the user stories and functional requirements. I tried to analyse and test all possible options - normal and boundary values. I've started the tests on the 24 of May (Work Log, 24/05/2018) and I was

terrified how many functions are not working but previously had worked well. This was due to the fact that I did not have time to keep testing of all the functionalities, because I wanted to implement as many functions as I could. I should every time I implement a new function, perform the test on the others, but because of lack of time, it was not possible to do. Unfortunately, I also do not have enough knowledge about UI automated tests and the possibility to learn it now. That's why I noted all bugs on the Kanban board and at the right moment, I tried to fix the errors. To fix the errors I usually used the debugger tools which is a part of the IDE that I've used.

The system provides the functionality that is sending the confirmation email with the token to newly registered account user. It's not possible (or it will be silly) to test in on the real email addresses. So, I have found a perfect solution that has provided me with a fake SMTP server. It's mailtrap.io. I'm using for the Spring Email configuration provided from mailtrap SMTP address, username and password and every time when I'm creating a new user account I received an email message on my mailtrap account where I can read the message and copy the token to activate the members account.

In the end, I did tests for the correctness of displaying content, scripts and CSS in various popular web browsers (Work Log, May 26, 2018). I tested the application on Google Chrome, Mozilla Firefox and Safari. I did not notice any glaring differences except that Chrome incorrectly displayed HTML5 date-picker input. I couldn't figure out why this happens, but I found the solution on the GitHub community forum, where someone wrote to change the input type from "date" to text".

Testing the application code is one of my favourite activities and I really enjoy it. Thanks to this project I could expand my knowledge about unit tests and understand how important it is to use them in such large projects, where it is easy to get lost in your own code. With tests, it is easier to identify the problem if the tests are written in a good way, you can test the entire application.

MAINTENANCE

Writing software in such a way that it is easy to maintain is not a simple task and can be a challenge. Typically developers train this skill for years, gaining experience in continuous code writing. Many times during my working on this project I asked myself "Can this function be written shorter? ", "Is this 'IF' needed here? ", " Here I could reorganize the code, and the repetitive fragment can be thrown into a separate function". I admit that if I was not constantly asking myself these questions I would probably finish the project earlier. However, I kept trying to find better solutions and polish my coding skills. I was looking for answers to issues on StackOverflow, where I put a few questions about how to better write a given code (Work Log, 25/03/2018, 28/03/2018, 15/05/2018). It is worth to mention that the programmers' job is not only about writing the code but also it is reading and understanding what someone has written before. A good code should be readable. I realized that when I write, even the smallest program or function, it has a later impact on the whole project. It's best practice to write a clean code from the beginning and form some good habits.

So how and what I did to try to ensure that my code is as much readable as I can afford with my skills? First, the names of variables, constants, functions and classes. Each name should be meaningful unless it is marginal. One-letter names may be suitable for temporary variables such as loop variables. The other names should reflect the purpose. A separate issue is what language we use in variable names. A good practice and a sign of professionalism are using only English names even if English is not our first language. Unfortunately, I've seen in many cases, where someone placed a code with variables and functions in another language and asked for help on an international forum. There are many websites on the Internet that describe the naming guidelines for variables and functions. I was using the guidelines found on the Medium website (https://medium.com/modernnerd-code/java-for-humans-naming-conventions-6353a1cd21a1).

Another point is testing the code. No one needs a nicely written program that does not work. Each time I tried to think about extreme cases, such as "what will happen if instead of a number the user enters letters?". To this end, I tried to secure the program as well as to apply defences programming rules through the use of validation (Spring Validation Annotations), regular expressions, and where I needed wrote JavaScript code to make it impossible to break the app. For example, the user cannot enter in the name input field numbers, and where the age of the pet is given, only numbers can be entered. I learned that in the history there was no any home pet who lived more than 30 years, so I restricted age field input to accept only up to 30. Also, users can only pick the past dates for when the pet was lost or found. Another small, but significant proof that I tried to take care of the details, is that after generating the report (Excel or PDF), the user could press the button several times and download the file the same number of times. So I wrote a script that, after generating the file, blocks the button for a few seconds. Another example of defence programming is that I completely remodelled the Pet entity so that user cannot accidentally assign the dog breed to a cat object and cat breed to the dog object (Work Log, 29-30/03/2018). Every time, when a user wants to add an announcement, a new pet or register a new account, all fields are

validated on the backend server side, in terms of whether the location or breed exists in the database or whether the user with the same email already has an account, etc. It is also a good practice to write your own unit tests that will allow you to catch a lot of errors.

What I would like to improve is the use of design patterns. It is still a topic I'm learning about, which is why I must admit that I did not succeed in implementing everything beyond basic principles. However, I tried to apply generally known object-oriented programming principles such as SOLID, KISS and DRY.

The principle "Do not Repeat Yourself" (DRY) or "Duplication is Evil" (DIE) has been implemented by throwing out repeating code to separate functions (eg validateFoundPostId(), validateLostPostId() in the REST service controller class). Also, for this purpose, I created separate classes such as AnnouncementValidation.java and UserValidation.java, where I put validation methods that are constantly repeated and used.

Another principle is KISS "Keep it simple, Stupid!" it has been used in the creation of methods and various functions. For example, the User and Announcement Validation classes (backend service) are as simple as possible and divide into small methods that do different tasks, so I can use them as and when I wish and it's easy to expand them and add new validation methods. I tried to build as small and short methods as possible. This way I make it easy to understand and read and they do not contain too many loops (like a loop in the loop and the next loop is not the best idea).

The last principle is SOLID. This is the five basic rules on how to write a good code:

- Single responsibility principle - that is, every class in my code is responsible only for one thing. For example, in the class responsible for generating Excel reports (ExcelReport.java - MCV app) I did not put any other code that is not related to the Excel format report (for example, it would be a bad practice to create one Report.java file and place the code for generating Excel and PDF files).
- Open/close principle - each class should be open for development, but closed for modifications. The interfaces used by me in the REST API service layer allow other programmers to extend the class with additional functionalities without interfering with the existing code.
- Liskov substitution principle - any derived class can be used in place of the base class (compatibility of all methods). This rule is usually broken in cases of poorly planned inheritance or when the polymorphic interface is too general. In my project, I tried to plan well the classes that inherit such as Pet, Dog, Cat and Announcement, Lost, Found. I did not combine responsibility in these classes, and the interfaces contain only the necessary methods.
- Interface segregation principle - all the interfaces used in my code are concise and specific so that the classes do not implement methods that they do not need.
- Dependency inversion principle - all dependencies should depend as much as possible on abstraction and not on a particular type. A good solution is not to depend on a single method for a specific type but only from an interface that can implement large

groups of subtypes (for example, LocationService.java is an interface that can be implemented in any number of controllers).

In my work, I have mainly supported myself with the principles of writing a good, maintenance code by reading one of the best books about good programming practices: "Clean Code: A Handbook of Agile Software Craftsmanship" (Martin, 2008)

From the reading of the book, I learned about the existence of "TODO" comments (chapter 4, page 58), which help to remind about the removal of some old function or the implementation of a new solution. The IDE in which I working supports "TODO" comments and provides special locating functions. The whole chapter on how to write good comments was helpful, however, it is admitted that due to lack of time, I could write better comments. Although on the other hand, there is probably no way that everything is perfect. You can always improve something.

Another example of using a good programming practices thanks to the book is the formatting of the code. Chapter 5 of the book allowed me to understand how important is vertical and horizontal formatting and how indentations can on affect the readability of the code. All my code has been formatted in accordance with the instructions provided by the author of the book.

In chapter 7 of the book, the author raised the subject of error handling. He writes in the book:

„Clean code is readable, but it must also be robust. These are not conflicting goals. We can write robust clean code if we see error handling as a separate concern, something that is viewable independently of our main logic. To the degree that we are able to do that, we can reason about it independently, and we can make great strides in the maintainability of our code." (chapter 7, page 112)

This sentence was like a revelation to me. I have never thought about error handling this way before, so after reading this chapter I decided to write error handling for all HTTP methods on the REST API. So, I created two classes to handle exceptions – CustomException.java for custom exception message when initialized by a call to Throwable class, and RestExceptionHandler.java which is handling all logic and validation errors.

The last aspect of the project's maintenance is documentation. In my work, I used the comments corresponding to the rules of generating automatic documentation using the JavaDoc tool. It required a lot of work and rethinking each line of code to describe it in a brief and simple way. In addition to documenting the Spring RESTful API, I used JSONDoc tool where I have documented all exposed HTTP methods.

CONCLUSION

Working on the "Lost My Pet" project was a satisfying challenge for me. I admit that at first, I did not understand the scale of the project and the work I need to put in to complete the system of this complexity level. Unfortunately, I was unable to create an additional mobile application client and I fail in the attempt to implement the real-time message system (which was not the function requirement but something extra that I wanted to add). All these functionalities could be completed in the next 2-3 months, but unfortunately, this time is gone. If I could go back in time, I would a little better arrange all the tasks. But overall I am happy with the project and I think that I have done a good job trying not to create an "academic project" but to design the real world solution (database and backend separately from the client apps). In the work on the project, I learned how to use SDLC, how to create UML diagrams to help me in my tasks. Also, the Kanban board proved to be helpful especially when I added to a card any error that I found. Thanks to that I did not worry about forgetting something. I'm also glad that I could create unit tests so that I learned how to write them better. I believe that the system meets all the criteria set by City Council, and the proof is the creation of all functionalities in the MVC web application and the design of an easy-to-use RESTful API service.

BIBLIOGRAPHY

Booch, G., 2005. In: *The Unified Modeling Language User Guide.* 2 edition ed. s.l.:Addison-Wesley Professional, pp. 5-1.

Cheptsov, A., 2014. *What Makes IntelliJ IDEA Different.* [Online]
Available at: https://medium.com/@andrey_cheptsov/what-makes-intellij-idea-different-5d6333621ef2
[Accessed 4 June 2018].

JetBrains s.r.o., 2010. *HP Case Study.* [Online]
Available at: https://www.jetbrains.com/company/customers/JetBrains_HPCaseStudy_5_14.pdf
[Accessed 3 June 2018].

Martin, R., 2008. *Clean Code: A Handbook of Agile Software Craftsmanship.* First ed. s.l.:Prentice Hall.

Riggins, J., 2015. *Why You Should Build Apps With An API Backend – BaaS.* [Online]
Available at: https://nordicapis.com/why-you-should-build-apps-with-an-api-backend-baas/)
[Accessed 03 June 2018].