POLITECNICO DI TORINO

MASTER's Degree in COMPUTER ENGINEERING

MASTER's Degree Thesis

Development of Microservices-based web application

Supervisors

Prof. LUCA ARDITO

Candidate

HASSAN KRAYEM

MARCH 2020

Summary

The project consists of the development and maintenance of a micro-services based web application in collaboration with a company, offering banks an integrated system used for the appraisal of real estate properties. The system facilitates the entry of property and valuation data and ensures a secure and efficient communication between the bank and the real estate advisor and its agents, providing an objective and independent assessment of value. The main problem of such solutions is the lack of flexibility and portability, since banks generally use a private secure system, customized to satisfy their specific needs.

The goal is to provide a consistent solutions and features in the app that is simply integrated into any bank's internal process, and to make it maintainable mainly through logs and to work on its adaptability to requirement changes and modifications on any level.
The biggest challenge here lies in understanding the microservices-based architecture, as well as designing and working with the structure of the entire solution, from different communicating servers, web services, load balancers, databases, cloud analytics and reporting tools.
Generally a system of this scale should accommodate any client regardless of the internal system they're using or the way they store data. For this reason providing a unified solution means that all these difficulties are dealt with and with a well defined and consistent process the solution can be customized.
In order to do so, and in addition to the main web application, an "in the middle" server application, connected to its own database, is developed for every new client. Starting from a simplified integration application then performing modifications and adding features in accordance to the requirements.
The above process ensures an efficient and reliable communication between the clients, who are banks, and the real estate valuation company and the services they provide. All going through the customized integration applications that are essential not only for synchronizing data and the workflow, but for monitoring the entire process and serving as powerful debugging and maintenance tools.

Maintenance and debugging occurs on the internally developed admin portals. Again admin portals features depend heavily on the client and his specifications, as for some cases it might cover for a limitation in the internal app, like forcing a notification or communication with the client, or to speed up processes allowing bulk insert or modification of data. Of course all that is in addition to the essential purpose of the these portals, which is to monitor the entire operation through filterable error logs that are presented in a chronological order allowing fast detection, and solution, of any occurring problems.
These tools prove very efficient upon the introduction of a new feature, as they allow us to monitor closely the behaviour of the system, especially when used in conjunction with a ticketing system for the clients, notifying the team as soon as an unexpected behaviour occurs.

Since the system was adaptable, the modifications were possible but it was thoroughly discussed within the team, holding daily meetings and determining the most efficient approaches to proceed at every step. Important aspects that were taken into consideration when developing such as writing commented, reusable code and working with consistency and on delivering optimized solutions, facilitating later updates or code analysis.

Changes to the workflow were essential to certain clients, where they were in need of a simplified workflow, allowing to bypass certain actions or restriction on fields that would otherwise be required. This modification lies in working on both the main server application and front-end web application. Other than implementing the logic of the modification, very important aspects should be taken into consideration in development:

- **Security**: To respect and follow the security measures implemented. Especially if the modification allows otherwise unauthorized behavior, which is the case here when simplifying the workflow, as in certain cases the user can bypass important actions. The goal is to make sure these additional privileges are only provided to the desired user profile.
- **Testing**: As in testing the entire system while and after developing. Since all the parties (main server, customized integration application, bank internal application) are in constant communication, so a modification in any of the applications should not affect.

From the bank's side, changes would be already effected beforehand by their IT staff.

Working within a team on a project this scale would force us not only to use Version Control, but to follow a specific workflow while doing so such as the feature branch workflow. It also allows agility and clear communication and collaboration to present itself naturally, since many developers have been and are still working and maintaining this system, meaning each individual's code would affect directly and indirectly what the team is experiencing.

In addition to being individually organized, daily meetings were held in the company by the team for many reasons, such as deciding the most optimal approaches to follow, making sure the projects are meeting the deadlines, as well as brainstorming for upcoming functionalities.

Technically, working on this project required the use of many different development frameworks (for both the client and server side). From full off-the-shelf web frameworks like Spring Boot to persistence frameworks like MyBatis, all the way to internally developed customized frameworks such as the one used to perform queries in a simplified manner, which allowed to speed up the development process and prevented the repetition of code in many cases.

The realization and maintenance of this system was not a fast nor easy task, it required learning and working patiently and closely alongside an experienced team of developers and system architects, with a clear defined plan.
The results proved successful. Both the new client integration application and the new features introduced have been tested and pushed to production upon completion, and clients have started using them. Until now the system is daily maintained and supported, and new changes are still requested from clients and would be fulfilled following the same process for the recent update.

Acknowledgements

To my family and friends for motivating me during this journey, and for dealing with my problems as theirs. It would not have been possible without their help and constant support.
To my professor, and my company colleagues who have taught me a lot and still are.

To anyone who contributed, or tried to.

Table of Contents

List of Tables

List of Figures

Acronyms

WAS
Web Appraisal System

SDLC
Software Development Life Cycle

ORM
Object-Relation Mapping

POJO
Plain Old Java Object

EJB
Enterprise JavaBeans

URI
Uniform Resource Identifier

URL
Uniform Resource Locator

CI
Continuous Integration

SOAP
Simple Object Access Protocol

REST
REpresentational State Transfer

Chapter 1

Introduction

A system is under constant development during its life cycle. The advancement can be generally divided into two classes of changes that the system is confronting: first, it is gradually developed from general requirements to functional code in a few stages, (for example, structure, execution, and testing); and second, the useful code is altered, or kept up, as indicated by different needs from the clients of the framework, (for example, new prerequisites, bug fixes, or porting into new situations). Usage and the board of all these various and frequently unconstrained changes in a controlled way is one of the fundamental purposes behind the expense of growing huge and complex delicate product frameworks.

Working on a large scaled project such as a multi-banking system is not feasible without understanding the well defined architectural style, with its compatible frameworks and technologies. All in order to be able to contribute in the system especially upon clients' requests.

1.1 Scenario

Taking a higher look at the solution, there are three main parts that should be highlighted:

- **Core Application:** The core application is a spring boot project running as a server for all integrated parties. The server communicates with the main database, here lies the logic and workflow of the app.

- **Bank Integration Application:** Contains the services communicating with a specific bank. Here lies the interaction with the bank's API and implementing the necessary interfaces to enable a secure and reliable communication between the bank and all other parties.

- **Web Application:** An Angular web application, it is the principal mean for accessing the system. Mainly by the appraisal company users to perform different operations on bank submitted requests and a full set of other features.
- **Mobile Application:** Mobile application that has a limited set of features compared to the web application.

The idea here is to join the development of a system that manages real estate appraisals and data related to property value, storing it and tracking any underlying manipulation, thereby providing a secure and well defined property appraisal workflow.
Appraisals are normally completed by state-certified appraisers that estimate the value of real property (land or building).
Often, the appraisal is done in a document called appraisal report, which is compiled after conducting a thorough study of the appraised property, its geographical area, and economic trends.
In addition, the appraiser must have some knowledge of building construction to recognize the quality and condition of the subject property. Generally, the appraiser's client can be a buyer, seller, or as in our case, a bank or a corporation.

1.2 Objectives

Real estate appraisal management is not a new concept, nor the existence of a system that provide such functionality.
The real challenge lies in providing an integrable system that works side by side with an internal application of an enterprise while satisfying all their needs and following their guidelines, such as security and smooth reliable communication.

After the initial handover of a system, customers can review and test the functionalities and design of the delivered system and discuss the issues to be changed or added in the next release.

One of the recurring requested features was the ability to introduce a simplified workflow on project level: allowing users to create appraisal requests that follow a different, more straightforward workflow with the possibility to skip certain, otherwise required actions.
Another modification would be the introduction of a simplified assignment on request level: users should be able to specify the type of assignment while creating the appraisal request (standard or simplified). Changing this setting would influence the way users view the request and its valuations, and allows them to choose the latter's type upon creation, which can be a normal valuation or an generic activity.

Reporting-wise, clients' need of a downloadable document showing information about the appraisal and its property as well as images related to the latter.

1.3 Project

In order to create an appraisal request, users now are obliged to fill in all the required forms, containing information related to the request itself, issuing company, invoicing details, property details, as well as mandatory documents upload in order to effectively create the request and start the evaluation process.
This complete workflow is rather counterproductive when the creation of requests and valuations is done for different, more specific purposes. For a handful of reasons it would be beneficial to have the possibility to skip some or even most of these checks, controls and certain actions, allowing appraisers to move forward with the same request, but in a limited and faster workflow.
Appraisal reports are documents that are built dynamically on user demand. With the help of a java library The system starts with opening an excel template containing the requested structure, and then editing and compiling the missing information and images.

1.4 Results

After the introduction of the features, users could create "simplified" projects, by enabling the setting on project creation. Enabling this option on demand is simple and user-friendly.
When the request is linked to a project with a simplified workflow, both the request and valuation pages are slightly modified, hiding certain form sections and fields that are otherwise required. The change is most noticeable when viewing the list of available actions, as users are free to move the appraisal request forward without having to spend extra time uploading the required documents related to the request (such as the normally required Valuation Report, or the Inspection Report created and uploaded by a local specialist after taking charge of an appraisal).

In addition, clients will be able to generate a fully-detailed property appraisal report, which is a multi-sheet, dynamically created excel workbook printable as PDF and containing information about both the appraisal and the property (to be) valuated. The file contains as well a gallery of all images related to the property in question. This report can essential to the business as it is a very reliable way of exporting and communicating this amount of data.

Chapter 2

Software life-cycle

Software Development Life Cycle (SDLC) is a process of building or maintaining software systems. It includes various essential phases for developers from preliminary development analysis to post-development software testing and evaluation.
It also consists of the models and methodologies utilized by development teams to develop the software systems, which determines eventually the way the entire development process is planned and controlled.

2.1 Objective

In a software development cycle, it is not rare to see more than half the development time being dedicated to the verification of the software and its conformity with specifications. Formal methods offer new possibilities for the verification process on either the specification level, by constant model analysis/checking that allows the detection of problems early on, or on the implementation level using analysis techniques such as abstract interpretation that facilitate the verification steps. To further understand methodologies and their life-cycle we'll be conducting a brief comparison between heavyweight and agile methodologies. For **heavyweight**, several methods are available such as *Waterfall*, *Unified Process* and *Spiral*, while in **agile** approaches we'd find *Extreme Programming*, **Scrum**, *Dynamic System Development Method*, *Feature Driven Development* and *Adaptive Software Development.*
Since the focus is on the followed agile methodology, we will discuss its phases and the challenges associated with implementing agile processes in the software industry according to software practitioners and anecdotal evidence.

Agile Software Development Agile development is based on the idea of incremental and iterative development, in which the phases within a development life

cycle are revisited over and over again. It iteratively improves software by using customer feedback to converge on solutions. In agile development, rather than a single large process model that implemented in conventional SDLC, the development life cycle is divided into smaller parts, called "increments" or "iterations", in which each of these increments touches on each of the conventional phases of development.[1]
Our main focus has to be on **Scrum** which is briefly an agile process framework that serves the development of products and services through a Scrum Team, consisting of:

- **The Product Owner**: His duties consist of maximizing the value of the product resulting from work of the *Development Team*. As well as being the sole responsible for managing the *Product Backlog*, which is an ordered list of everything that is known to be needed in the product and the sole source of requirements, and ensuring that the team understands items and tasks there to the level needed.
- **The Scrum Master**: A servant-leader for the Scrum Team, responsible for advancing and supporting Scrum as characterized in the Scrum Guide. Scrum Masters do this by helping everybody comprehend Scrum theory, practices, rules, and qualities.
- **The Development Team**: Consists of professionals with the responsibility of delivering a potentially releasable Increment of "Done" product at the end of each Sprint. The effectiveness and overall efficiency is due to the fact they're organized and self-managing, which are values generally empowered by the organization.

This is all done through rapid deployment of functionalities previously collected and organized in the product backlog.
When dealing with a diverse list of tasks, **Version Control** comes as a necessity for handling dependent tasks and synchronizing work within the team, even though it is not strictly speaking an agile practice rather than a necessary technique widely used in the industry as a whole. Its mode of use will be further explained in the project section below.

2.2 Phases

Generally, a project is defined by the set of technical and managerial activities, necessary for meeting the requirements or terms agreed upon beforehand, including the scope, objectives, budget and timeline of said project (IEEE 1058-1998). Therefore, the phases followed during the project life-cycle differ massively depending on the methodology of work used. The agile methodology usually follows a defined process which can be roughly represented by:

- Requirements Analysis: also referred to as specification phase, consists of analyzing the user requirements defining how the system is supposed to function. The results of the analysis are saved in a document often called "requirements specification".
- Planning and Architecture: aims generally to decompose the project in more elementary activities and tasks that have minimal interrelations. Defining the tasks, the timetable, resources and their allocation for tasks, as well as the architecture and structure in which the system should follow.
- Implementation: The team codes, tests, and integrates the software. Also trying to significantly reduce the likelihood that security vulnerabilities will make their way into the final version of the software that is released.
- Testing and Maintenance: it includes software testing, verification and validation of the built system. Software testing can be implemented at any time in the development process. One of the most important purposes of testing is to detect software failures so that defects may be discovered and corrected.

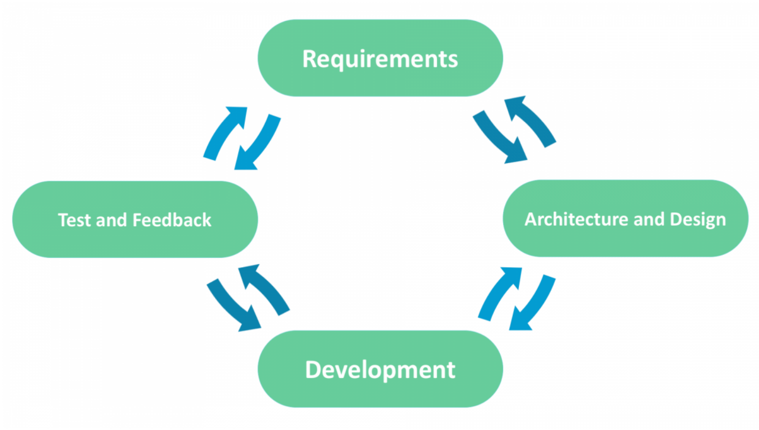


Figure 2.1: Agile Process

What gives this model its superiority are the following key values:

- The high ability to respond to the changing requirements of the project.
- Clarity and transparency between the development team and the customer, since there is continuous input and direct face-to-face communication between the two parties.[2]

2.3 Project

The agile methodology proved to be efficient for this project since it is suited for production with continuous delivery. The many scripts, independent modules and small projects integrated into the production application make it more susceptible to breakdown for testing and debugging purposes, as well as future addition of features.
Following the above mentioned process, the development team responsible of this project is normally given tasks on monthly or weekly basis, and it is up to them to organize their time and tasks on daily basis in order to deliver what's required on time. For this purpose, a project management application is typically utilized to facilitate teamwork, progress, visibility and coordination, by creating, assigning, and visually organizing the work.

2.3.1 Tools

Version Control

As previously mentioned, organizing each iteration is practically unattainable without a Version Control System. **Git** is a distributed revision control system available on all mainstream development platforms through a free software license.

One of the most used features of a versioning system is *branching*. A *branch* is a crucial go-to when launching a separate line of development inside a software product. It is a split from the source at a given state, permitting development to proceed in various directions at the same time and, possibly, producing different versions of the project.
Regularly, a branch is reconciled and merged with other branches to rejoin all participating efforts. The ability to have many branches makes this approach the go-to for most Git clients.

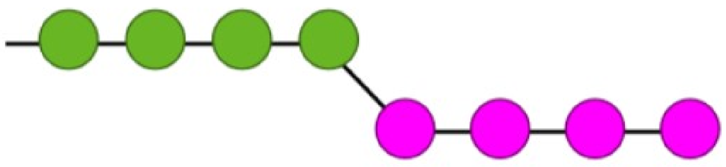


Figure 2.2: Working on a branch for a specific feature

Git offers a lot of flexibility in how users manage changes and given its focus on flexibility, there is no standardized process on how to interact with Git. There are several publicized Git workflows to follow, what's more important is to ensure the entire team is on the same page. Let us first take a quick look at some of the common workflows alongside the Git versioning system:

- **Centralized workflow:** In this scenario, you usually follow these simple steps:
  - Someone initializes the remote repository.
  - Other team members clone the original repository on their device and start working.
  - When the work is done, we push it to the remote to make it available to other colleagues.

  It is unlikely and discouraged that all developers are working simultaneously on the *master* branch.

- **Feature branch workflow:** In this approach every single developer works on their branch. Then the feature branch is merged with the master branch when the work is done.
  We might need to merge back from the master branch first in case someone merged a feature branch after starting our new branch.

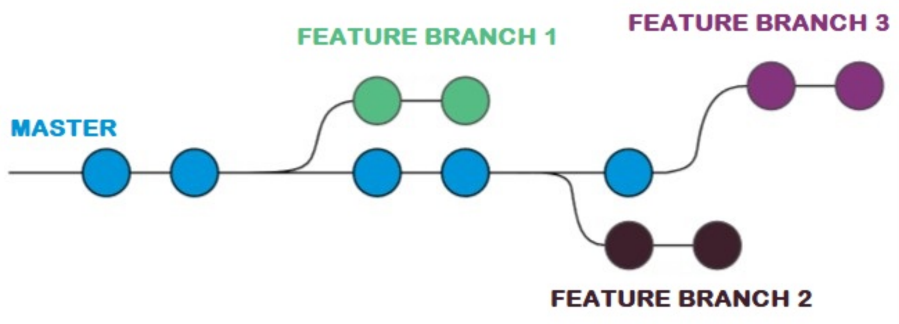


Figure 2.3: Representation of branches midst a feature branch workflow

- **Gitflow workflow:** Next to the main branches master and develop, this model uses a variety of supporting branches to aid parallel development between team members, ease tracking of features, prepare for production releases and to assist in quickly fixing live production problems. Unlike the main branches, these branches always have a limited life time, since they will be removed eventually. These branches could be feature, release or hot-fix branches. [3]

Testing

Every independent user-written function or method must have its equivalent test class especially in such systems where the project is continuously increasing in size and complexity. The goal is to be always certain our components remain functional after the implementation of a new feature or a requirement change, knowing that multiple members of the development team are working together. Worth noting that the developer who implements the feature is not necessarily the one writing its test classes.

Below is a code snippet of a **JUnit** class used to test and maintain a user-defined method *getWorkingInterval*:

```
 @RunWith(SpringJUnit4ClassRunner.class)
 @ContextConfiguration("classpath*:test-spring-configuration.xml")
 public class DatesUtilsTest {

     DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");

     @Test
     public void calcoloGiorniNettiTest() throws ParseException {

     String start = "29/09/2017";
     String end = "03/10/2017";
     Date startDate = dateFormat.parse(start);
     Date endDate = dateFormat.parse(end);

     long intervalNetto = DatesUtils.getWorkingInterval(startDate, endDate);

     assertEquals(2, intervalNetto);
 }

```

Continuous Integration

The term 'Continuous Integration' originated with the Extreme Programming development process, as one of its original twelve practices. [4].
In Continuous Integration (CI) after a code push or release, the software is built and tested immediately. In a large project with many developers, commits are made many times during a day. With each the code is built and tested. If the test is passed, build is tested for deployment. If deployment is a success, the code is pushed to production. This commit, build, test, and deploy is a continuous process and hence the name continuous integration/deployment.

Jenkins is a Java-based continuous integration server that supports the discovery of defects early in the software cycle. Thanks to over 400 plugins, Jenkins communicates with many types of systems, building and triggering a wide variety of tests.
The wide use of Jenkins is due to its advantages:

- Managed by a community that is very open. With public meetings held every month and taking inputs from the public for the development of Jenkins project.
- There is no fuss in installation, as installation is as simple as running only a single download file named jenkins.war.
- It has a simple configuration through a web-based GUI, which speeds up Job creation, improves consistency, and decreases the maintenance costs.
- Though highly supportive of Java, Jenkins also supports other languages.
- Deployable in cloud-based platforms since it supports cloud-based architecture as well. [5]

2.3.2 Scenario

Let us take a look at the process normally followed by a developer working on a certain task in this project.

The project manager starts the planning phase by decomposing the project or new features in hand into tasks, clearly differentiating depending on their priority, complexity and dependency.
Using a project management system or planner, the tasks are then distributed on the team, each according to their technical capacities.

The developer starts by analyzing and assessing the task at hand, this step is important since it is crucial in deciding the succeeding actions:

- If the task is a **hot-fix**, the developer may work on resolving it and directly pushing it to the origin staging branch.
- If it is a lengthy task or a new feature the developer creates or checks out a branch, normally named after the scope he's currently in (eg. creating a branch named simplifiedAssignment for completing tasks related to updating the assignment from standard to simplified).

Developers are encouraged to commit to their working branch and push it to origin after having finished a specified task or sub-task, in accordance with the requirements, while describing briefly their work in the message section.

After the task is implemented, the developer in charge is typically responsible for its reviewing and testing, to make sure it satisfies the specifications before pushing it to origin and merging with the parent branch after checking and resolving any existing conflicts.

Chapter 3

Technologies and Methodology

Choosing the right technologies and tools to use is essential when developing such complex system, we need to foresee future needs.
In this chapter, we will go through used technologies and explain the reasoning behind choosing them.

3.1 Programming Languages

3.1.1 Java

Overview

Java Enterprise Edition offers enterprise developers the possibility of building both local and online applications, in a simplified, component-based approach. It depicts the application configurations upheld by the J2EE platform and introduces practical rules and guidelines to follow while deciding the best design for specific needs. [6]

Version

Java 8 was used as it incorporates many useful new features such as lambda expressions, streams, improved garbage collection and better overall performance. This version is relatively new, it was first released in 2014 and is still consistently supported and maintained to date.

Notable libraries

- jjwt
- mssql-jdbc
- google-maps-services
- mybatis
- javax mail
- junit
- apache poi

3.1.2 HTML

Hyper**T**ext **M**arkup **L**anguage is used to create web pages. A simple data format that excels in creating hypertext documents that are portable from one platform to another. HTML documents are SGML (Standard Generalized Markup Language) documents with generic semantics that are appropriate for representing information from a wide range of domains.

HTML really shines when it's used in conjunction with an actual programming language, such as when a web framework is used in development. That way the developer can create dynamic web pages and database applications.

3.1.3 CSS

Cascading Style Sheets (CSS) is a style sheet language utilized for describing the presentation of a document written in a markup language like HTML. The introduction of CSS along with HTML 4.0 was a breakthrough in the field.

After it's launch, as developers we could separate presentation from content. As a result, styling could be removed from the HTML document and stored in a separate file, which would be included in the document with a reference.
The great advantage it has is that when changes are made to the design, not all the HTML files used have to be adapted, but only the corresponding CSS file. Content and design are clearly separated and the design is centrally managed in one or few files. The CSS files contain only CSS instructions.

3.1.4 JavaScript

JavaScript, is a lightweight, interpreted programming language with object-oriented capabilities and the most well received implementation of the language specification ECMAScript. This client-side script permits executable content to be included in web pages – it means that a web page is no longer compelled to be static HTML, but can include small programs that control the browser, interact with the user, and dynamically produce HTML content.[7]

The main reason for selecting JavaScript is its widespread use and availability. The most commonly used browsers Firefox, Internet Explorer and Chromium-based browsers support it, as do almost all of the less commonly used ones. It can be assumed that the majority of people browsing a web site will have a version of JavaScript installed, although it is possible to manually disable it through the browser's settings.
The most common uses of JavaScript are interacting with users, getting information from them, and validating their actions.

Despite its success, it remains a poor language for developing and maintaining large applications. **TypeScript** is an extension of JavaScript intended to address this deficiency. Syntactically, TypeScript is a superset of EcmaScript 5, so every JavaScript program is a TypeScript program. TypeScript enriches JavaScript with a module system, classes, interfaces, and a static type system. As it aims to provide lightweight assistance to programmers, the module system and the type system are flexible and easy to use. In particular, they support many common JavaScript programming practices.[8]

3.2 Database

Microsoft SQL Server was used in this project as a relational database management system. It was developed by Microsoft in the 1980's and became with time the go-to platform for large-scale enterprises due to its scalability and reliability.
Its main features include:

1. **High performance** - MS-SQL is very efficient in handling smaller-scope projects as it is for bigger, complex ones.

2. **Index-usage** - Indexes are used for performance optimization and data sorting.

3. **Keys** - The system make use of what is called primary and foreign key constraints to define tabular relationships.

T-SQL (Transact-SQL) was used heavily in our project, which is a set of programming extensions that add numerous features to the Structured Query Language (SQL), including transaction control, exception and error handling, row processing and declared variables. Most used features would be:

1. **Stored Procedures** - A compiled, stored T-SQL code that generate an execution plan on its first call. It executes any T-SQL code within its parameters. The purpose of usage is its faster execution and reduced network traffic, as well as it being a security mechanism and a structured way of working. Procedures can be modified independently of the program source code, the application doesn't have to be recompiled when the SQL is altered.

2. **User-defined functions** - Functions that accept parameters and return the results in the form of a table that can be queried and joined with other tables.

3. **Triggers** - A trigger is a special type of stored procedure that executes when a specified operation occurs.
   Most of the details of stored procedure programming apply equally well to triggers. In fact, since we can call a stored procedure from a trigger, we can effectively do anything in a trigger that a stored procedure can do. One thing that triggers do not normally do is return result sets. Most front ends have no way of handling trigger-generated result sets, so we just don't see it in production code. Note that SQL Server does not permit triggers to return result codes. [9]

3.3 Frameworks

3.3.1 Angular

Angular is an open source modern JavaScript framework used to build web, mobile web, native mobile and native desktop applications. It is also used in combination with any server-side web application framework, such as ASP.NET and Node.js.

Angular is the successor of AngularJS 1, one the best JavaScript frameworks for building client-side web applications[10]. Angular removed some of the concepts that were used in AngularJS auch as scope, controller, factory and others, while it also has a different syntax for building attributes and events.
With every release, new features and changes are introduced (adding *else async* to **ngIf* and **ngFor* respectively in Angular 4, Angular elements in version 6.0.0) and of course use of the latest versions of libraries (RxJS, Material, etc..).

Angular has built-in protections against common web-applications vulnerabilities and attacks, such as XSS, CSRF and XSSI. Certain application-level security, however, such as authentication and authorization, Angular leaves to the back-end. Even though Angular's HttpClient library also has support for the client-facing end for CSRF [11].

3.3.2 Spring

The Spring Framework is a lightweight solution and a potential one-stop-shop for building enterprise-ready applications. Designed to be non-intrusive, the domain logic code generally has no dependencies on the framework itself. In the integration layer (such as the data access layer), some dependencies on the data access technology and the Spring libraries will exist. However, it should be easy to isolate these dependencies from the rest of the code base.

Spring started as a lightweight alternative to Java Enterprise Edition (JEE, or J2EE as it was known at the time). Rather than develop components as heavyweight Enterprise JavaBeans (EJBs), Spring offered a simpler approach to enterprise Java development, utilizing dependency injection and aspect-oriented programming to achieve the capabilities of EJB with plain old Java objects (POJOs). [12]

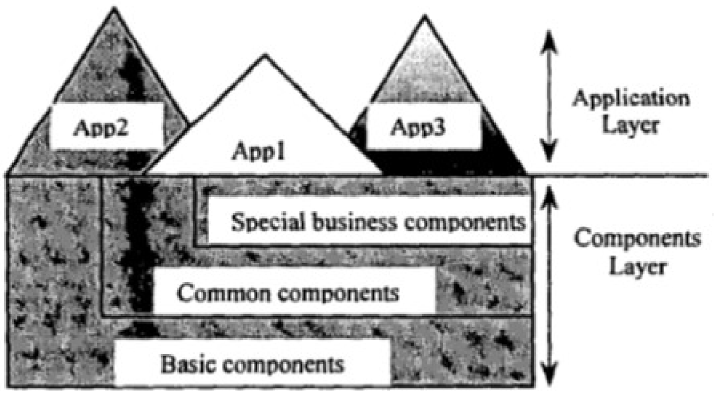


Figure 3.1: System architecture of component-based software systems

In previous versions, Spring relied on manually-written XML configuration files, used to declare and arrange the dispatcher servlet which is a controller that's main purpose is to receive all the requests dispatching them to other components. Spring framework improved and changed notably with time. Even so, there was no escape from configuration. Enabling certain Spring features such as transaction management and Spring MVC required explicit configuration, either in XML or Java. Enabling third-party library features such as Thymeleaf-based web views required explicit configuration. Configuring servlets and filters (such as Spring's DispatcherServlet) required explicit configuration in web.xml or in a servlet initializer. Component-scanning reduced configuration and Java configuration made it less awkward, but Spring still required a lot of configuration.

MVC Different developers may expend a great deal of development time and effort on solving problems from first principles each time they occur, and the solution that each produce may not be the most appropriate that could be achieved. Patterns provide a means for capturing knowledge about successful solutions in the software development. One of these patterns is Model-View-Controller (MVC) architecture, initially introduced for user interfaces in application implemented with the programming language Smalltalk. In this approach the system is divided in three components: *model* that express the domain knowledge, *view* that present the user interface, and *controller* that manages the updates to views.

3.3.3 MyBatis

When designing and developing applications, it is always the choice of using ORM (Object-Relation Mapping) or a framework that will serve to reflect the data from the user's database-objects. There are quite a few different ORMs, such as Hibernate, JPA, EclipseLink, etc.. Each of them has its own virtues and disadvantages. Most frameworks support the JPA specification.

MyBatis is a framework that displays SQL queries for interchange objects. This framework (as well as other ORM frameworks) eliminates the need to write a template code for previously unresolved tasks. Abstracting all these common tasks allows the developer to focus on the really important aspects, such as preparing the SQL statement that needs to be executed and passing the input data as Java objects.
At the beginning of 2010, project myBatis has split off from iBatis (fork). The development team of the Apache Software Foundation moved to the Google Code project platform. It is therefore necessary to use the myBatis framework for future developments because the iBatis development will be discontinued.
Interacting with POJOs, the mappers developed with this framework can be configured using XML, or with the help of java annotations.
In addition to this, MyBatis automates the process of setting the query parameters from the input Java object properties and populates the Java objects with the SQL query results as well.

3.3.4 FP

FP is a framework based on MyBatis, developed to have its flexibility when a complex query is needed, and at the same time to not having to write simple and repetitive queries. The framework makes use of two main classes that can help understand it:

- **BasePojo:** A java class extended by our POJO, containing the common attributes and an injected main MyBatis mapper called *baseQuery* that its methods are built dynamically to provide the query to perform on the POJO. In addition, the class contains basic methods (save(), select(), delete()) allowing us to perform flexible CRUD operations.

- **BaseList:** A java class that extends an ArrayList<E>, where the generic type E is specified upon class instantiation. The class is used when fetching more than one record from our database.

What also makes this framework handy is that the code written could be easily understood by any developer working on the project. An example of its functioning will be explained in the next chapter.

3.4 Web Services

3.4.1 Goal

The goal of web services is to allow normally incompatible applications to interoperate over the web regardless of language, platform, or operating system. Web services allow for business processes to be made available over the internet.
Web services allow systems to communicate with each other using standard Internet technologies. Systems that have to communicate with other systems use communication protocols and the data formats that both systems understand.
This platform independence is also evident on the World Wide Web itself. A Web site uses HTTP and HTML to pass data to a user's browser—this is the only requirement the site must support. A Web site may be developed using a large number of languages and platforms, but the platform is irrelevant as long as the data is ultimately provided to the browser using HTTP and HTML. These same principles apply to Web services.

3.4.2 Types

There are two types of web services based on **SOAP** (**S**imple **O**bject **A**ccess **P**rotocol) principle and REST principle. Various applications such as conferencing, web application can be developed using SOAP and RESTful web services. In SOAP based web services XML is used to define **SOAP**.
RESTful web services follows REST (**Re**presentational **S**tate **T**ransfer) principle for distributed hypermedia systems. REST design style is defined as network architectural style because RESTful web services depend on HTTP, HTML and other web technologies.

SOAP-based

The SOAP based web service architecture defines 3 entities: -service provider, service registry, and service requester.
The service provider is the service, the network addressable entity that accepts and executes requests from consumers.
The service consumer is an application, service or some other type of software module that requires a service.
A service registry is a network-based directory that contains available services. The service consumer finds the service description in the registry which is published by the service provider. [13]

REST-based

The term representational state transfer was introduced by Roy Fielding in 2000. REST style architecture is an architecture style that is often used in the development of web services. In which the client sends a request to the server that processes the request and returns responses. These requests and responses are built around the transfer of representations of resources. A resource is something that is identified by a URI.
REST does not require a message format like envelope and header which is required in SOAP messages.

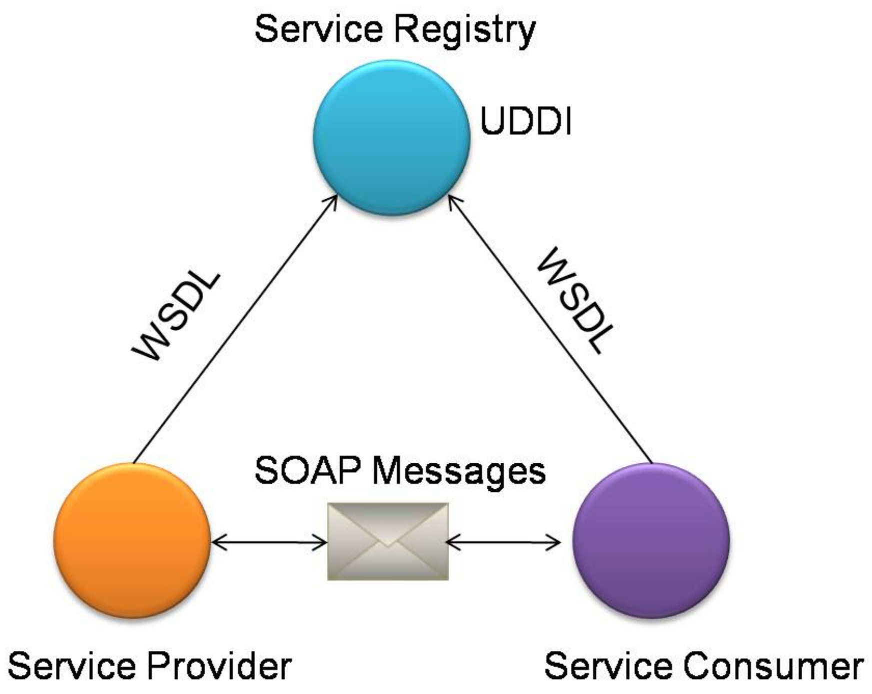


Figure 3.2: figure showing the three major roles within the web service architecture

3.5 Architecture

In order to further explain this, we must first differentiate between various common architectural styles:

- **Monolithic Architecture:** Consisting of developing an application as one unit. Even though it might have several services and components, it's still deployed as a unified solution. This approach is preferred for smaller-scale applications as it offers easier development and deployment, but not much scalability.

- **Service-Oriented Architecture (SOA):** An approach for the architectural conception that guides all aspects of the creation and use of services throughout their entire life-cycle, defining and producing the IT infrastructure that permits different applications to exchange data and participate in an enterprise process regardless the programming languages used and environments of which these applications use. As the name suggests, a service is a key concept of an **SOA**, which is technically defined as a fine-grained function that can be encapsulated in reusable components.

- **Microservice Architecture (MSA):** This paradigm is a relatively new approach that consists in developing an application distributed in model-based and autonomous components, called **microservices**. The architecture is used more and more in development especially after the emergence of *Cloud* and *Fog computing.*
  This concept is what was adapted in the project and will be further discussed in this section.

The term Microservice Architecture was coined by a group of software architects in 2012. With such an architecture approach, complex applications are not only divided into separate components, but these also remain independent during the life-cycle, running in a separate process and communicating with each other only via technology- and language-independent interfaces. The loose coupling of the components offers various advantages. This is why various large companies such as Amazon or Netflix have meanwhile adopted this architectural approach. The positive reception is generally due to this architecture's points of strength:

- **Extensibility:** The system's ability to have a new functionality extended, in which the system's internal structure and data flow are minimally or not affected (recompiling or changing the original source code is unnecessary).
- **Code organization** and **reusability** of microservices.
- **Scalability:** The architechture has to take into account future changes of the software according to business needs. If we can't anticipate these changes early on, the system must be flexible enough to make the modification possible.

The figure below gives us an idea of the structure:

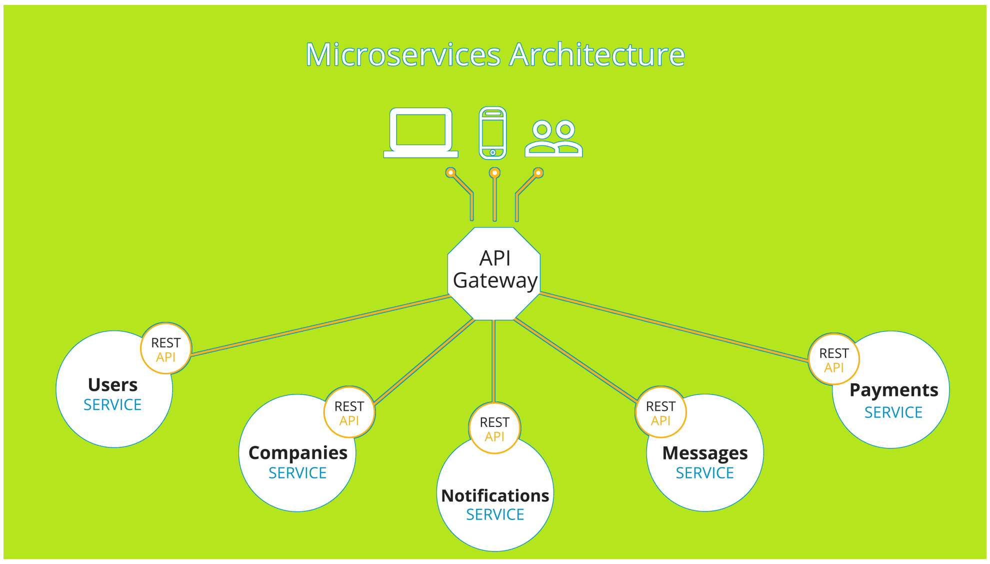


Figure 3.3: Figure illustrating the schema followed by a Miroservice Architechture

Guidelines

Enterprise Applications are often built in three main parts: a client-side user interface (consisting of HTML pages and JavaScript running in a browser on the user's machine) a database (consisting of many tables inserted into a common, and usually relational, database management system), and a server-side application. When working with a microservice architecture, we think of other internal development teams as external services that our microservice interacts with through APIs. The commonly understood "contract" between microservices is that their APIs are stable and forward compatible. Just as it's unacceptable for the Google Maps API to change without warning and in such a way that it breaks its users, custom built APIs can evolve but must remain compatible with previous versions. [14]

3.6 Security

One of the most important aspects of such applications is to ensure security and this to a sufficient degree. This point is often neglected when optimizing processes. Web applications are often launched without knowing if the security of the application has been sufficiently taken into account. Much more than in the past, business processes are mapped and executed on the Internet. This includes processes between business partners as well as processes between companies and their customers, ranging from very simple to very complex systems. Irrespective of this, sufficient attention should always be paid to security aspects. Often, due to lack of time or money, insufficient attention is paid to this aspect, so that security can only be guaranteed to a limited extent.

3.6.1 Vulnerabilities

XSS

Most security vulnerabilities on websites are based on the principle of cross-site scripting (XSS). It is an attempt to get a user's browser to execute a malicious JavaScript. Once an attacker has placed his JavaScript code, he can use it to perform any action he wants - including stealing the contents of the cookie with the session ID or performing threatening actions on the website. There are three types: reflected, persistent and DOM-based XSS.

SQL Injection

Regardless of the script or programming language used, SQL injections are a significant threat to the security of web applications. Whenever user input is used for a database query, it is essential to check for the existence of special characters such as quotation marks, apostrophe, semicolon or backslash or to mask them.

Considering that we're exclusively using **MyBatis** and **FP** for database communication, the app is practically protected against SQL Injections granted we take some extra measures while using it.
In **MyBatis**, string substitution using #{varName}, as seen in the snippet below, causes the framework to create a prepared statement which prevents SQL Injections as opposed to ${varName} which would inject unmodified string into SQL.

```
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;

@Mapper
public interface ProfileMapper {

    @SELECT("SELECT * FROM PROFILES WHERE IdProfile = #{idProfile}")
    Profile searchProfileById(@Param("idProfile") int idProfile);

    @SELECT("SELECT * FROM PROFILES WHERE idUser = #{idUser}")
```

```
      List<Profilo> searchProfiles(@Param("idUser") int
  idUser);

      @UPDATE("UPDATE PROFILES set idProfile = #{
  idProfile} WHERE idUser = #{idUser}")
      int (@Param("idUser") Integer idUser, @Param("
  idProfile") Integer idProfile);
   }

```

On the other hand, FP relies on MyBatis prepared statements by default and handles character escaping.

Non-structural vulnerabilities

Not all security vulnerabilities are structural or technical, so to speak. Business logic vulnerabilities are common in a relatively complex system.
The method consists of finding a specific feature in a system, and exploit it by using it in an overly exaggerated way in order to achieve various goals including accessing/blocking accounts, performing an illegal operation in the application like crashing it after bypassing validators and submitting a faulty input.
Since the permitted actions and redirections are dependent of the access group the user falls in, another important issue comes to mind which is the integrity of the logged user. To ensure the system remains invulnerable to this problem, the testing team performs periodic checks trying to reproduce the issue.

3.6.2 Authentication

JSON Web Token (JWT) is a JSON-based security token encoding that enables identity and security information to be shared across security domains. A security token is generally produced by an **identity provider** and consumed by a **relying party** that relies on its content to identify the token's subject for security-related purposes.[15]
The token itself typically looks like the following *xxxxx.yyyyy.zzzzz*
Breaking it down the different parts, it consists of:

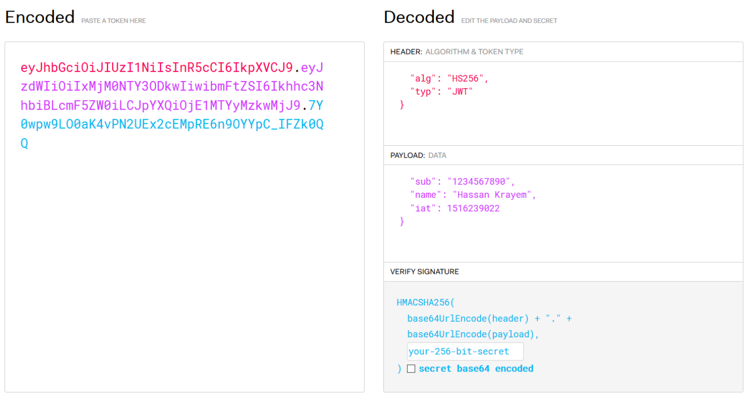
- **Header -** Usually consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as *HMAC*, *SHA256* or *RSA*.

- **Payload -** contains the claims. Claims are statements about an entity (typically, the user) and additional data. There are three types of claims:

  - **Registered claims:** These are a set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims. Some of them are: **iss** (issuer), **exp** (expiration time), **sub** (subject), **aud** (audience), as well as others.
  - **Public claims:** can be defined on demand by those using JWTs.
  - **Private claims:** These are the custom claims created to share information between parties that agree on using them and are neither registered nor public claims.

- **Signature -** To create the signature part we have to take the encoded header, the encoded payload, a secret and the algorithm specified in the header, then sign that. The signature is used to verify the message wasn't changed along the way, and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is the right person.

Putting it all together, the output is three Base64-URL strings (one for each part) separated by dots that can be easily passed in HTML and HTTP environments. The figure below from jwt.io explains further more:


Encoded PASTE A TOKEN HERE
Decoded EDIT THE PAYLOAD AND SECRET
HEADER: ALGORITHM & TOKEN TYPE
PAYLOAD: DATA
VERIFY SIGNATURE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
zdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikhhc3N
hbiBLcmF5ZW0iLCJpYXQiOjE1MTYyMzkwMjJ9.7Y
0wpw9LO0aK4vPN2UEx2cEMpRE6n9OYYpC_IFZk0Q
Q
```

```
  "alg": "HS256",
  "typ": "JWT"
}
```

```
  "sub": "1234567890",
  "name": "Hassan Krayem",
  "iat": 1516239022
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

In most solutions, the approach to follow would be the creation of the token by the back-end server (ex:Spring Boot) after a successful standard credential-based authentication – the e-mail provider matches a username with a known user in its database, and verifies that the password matches with what's on record – these credentials, alongside others such as the current date, are what the identity provider uses to generate a unique user token upon request.
The token is then sent to the user, and it's up to the client platform to save it, generally into local storage, and send it back to the server on each authenticated request.

After the server's reception, the token is decrypted and verified. Only then the server can decide either to authenticate the user, in case of a successful attempt, or deny him access in case of an unsuccessful one.

Chapter 4

Project

4.1 Overview

The application can be defined as a Real Estate Valuation system. In a way every bank user, valuation agency user and others who may access the app have a type of profile, acting as a user group with specific permitted actions depending on one's duties.

4.2 Scenario (As-is)

Before delving into the project's workflow and processes, it is necessary to differentiate between above mentioned user profiles. Only administrators are the ones allowed to navigate through all pages and execute all actions. Other profiles are:

Bank admins start by creating a project (BankA), filling out all required information and specifying settings such as if the requests and activities belonging to this project would later follow a standard or simplified workflow, or the ability of having a simplified assignment later on. These settings are non-modifiable as they represent the logic followed later by the app.
Bank business partners from clients to companies are also assigned to the project on or after creation, only assigned clients would later show up for the pick while creating the request.
Each project has predefined lists and templates automatically generated by the system on creation. The former are what is later used for populating selection menus in the request creation forms, while the latter are what determine which are the required fields in those forms.
These options were implemented in portable, modifiable way since they are susceptible for change, depending on the client's requirements.

Profile	Responsibilities
Local specialist	Managing the activities assigned to him by the company, after choosing to either accept or reject the assignment. Accepting means taking the activity under charge and pledging to carry out an inspection over said activities.
Technical reviewer	Revising the request assigned to him by the company, after choosing to either accept or reject the assignment. Accepting means verifying that the request is valid with all its required documents before sending it to the company. The feedback can be either negative or positive.
Project manager	A single contract agreed with a bank, generating dozens of valuations a day, is considered a project. The PM manages the site inspections and reviews scheduling, as well as the relationship with the bank.
Bank User	A profile that can insert and monitor requests of valuation for a bank. Bank users are not enabled for projects when an integration is provided, since they will manage the requests on their own portal.
System Administrator	A profile that can view and manage every request, user and project on the system.

Table 4.1: List of profiles and the permitted actions

After project creation, users of BankA are able to create a Request filling out related forms. Each request can have one activity or more assigned to it.

The system generally follows a standard workflow unless the project is explicitly set to follow a more simplified workflow, which derives from the standard but providing the possibility of skipping certain, otherwise required, actions and controls. During each phase the request is in a specific state, which determines the possible actions and fields at any point.
In this section we will briefly describe both workflows of the app, as well as different states of requests and appraisals, and other implemented processes.

4.2.1 Standard workflow

New Request

First, a request is created by the bank and assigned to a specific project, filling all the data required for its population (mainly the credit class, request class, survey

Request state	Code	Appraisal state
Draft	0	Draft
Sent to company	1	Sent to company
In Process	2	Picked up by company
In Process	3	Sent to local specialist
In Process	4	In process by local specialist
In Process	5	Inspection carried out
In Process	6	Documentation complete
In Process	7	Concluded by local specialist
In Process	8	Sent to technical reviewer
In Process	9	In process by technical reviewer
In Process	10	Under supervision
Complete	11	Inspection carried out

Table 4.2: Example of request states with their respective appraisal state and information

mode and type of assignment) and then adding one or more properties. The request is then assigned and sent to a local specialist of the bank's choosing.

Picked up

After having taken charge of the request, the local specialist has to download the specified list of documents, depending on the bank and its requirements and modify them. The files must be re-uploaded later and sent to the bank.

Suspension and reactivation

The appraisal can be suspended only by the company, following a simple action done by their user. It can be done either for the lack of documentation, for communication problems or after fixing a date for conducting an inspection on the property.
Reactivation can be done either by the bank itself or by the company.

In the first case, if the bank decides to reactivate the request, the latter rebecomes in processing by the local specialist who has then to replan an inspection. Documents that were not viewed/downloaded before sending the request would be downloaded.
In the second no documents would be downloaded and the request reverts back to the previous state: Processing.

Sending to bank

Performing the action *Send to bank* from the portal sends a request to the bank containing all the appraisal data with its attached files. It is mandatory to fill in all the required information and upload the necessary documents to proceed with this action.

Sending to subordinate bank

It's also possible to perform the action *Send to subordinate bank*, in this case the request is temporarily complete. As opposed to the previous action, it is mandatory to send solely the request's PDF.
The bank can then ask either for integration, or final submission.

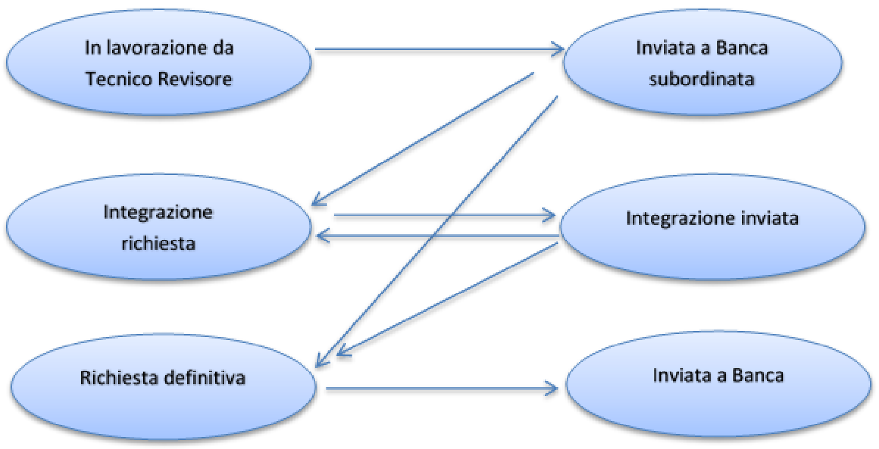


Figure 4.1: States of the request during the standard workflow

Closing and cancellation

The bank is able to perform a cancellation or closing of the request, blocking further actions.
Both these actions can be later revoked by the same party (revoking cancellation/re-opening request).

4.2.2 Integration

An integration application is a server application that communicates with its own database, referred to generally as migration database.

The app should receive requests through its deployed web services from the bank's internal application, specifying an action to be performed (e.g. Create request, etc..).

The common models the apps make use off are part of a separate project that is added to the integration project as a dependency.

The integration application processes the request, storing certain information in the integration database then communicating with the core application, in order to effectively inform the latter of this action. This communication works both directions, when the company users perform actions from their portal, the core application communicates this action with the integration application, that communicates with the bank through its provided APIs.

The current solution might satisfy all upcoming needs, which is one application that both the core application and the bank communicate with. But since the work process and specifications heavily varied from one client to an other, the need of a separate application or portal that satisfies those specifications emerged previously, some older clients are still performing some work through a separate system that communicates with the core application, while the unified integration application is fully functional.

For the team responsible for the project the goal is to limit all these independent applications and make the software truly unified.

4.3 Scenario (To-be)

4.3.1 Simplified workflow

After consulting with clients and analyzing their requirements it proved necessary, in certain cases, for them to have the possibility of starting a simplified workflow, introduced on project-level. This feature consists in specifying the type of workflow followed while creating a project. All requests belonging to this project should follow this type.
A request belonging to a project following a simplified workflow allows special actions to be performed, such as skipping fields and file requirements. An example would be the ability to send the appraisal request directly to the company upon creation, without having to upload a descriptive file for each property to be valuated.

4.3.2 Simplified assignment

After the introduction of a simplified workflow on *project level*, clients realized their need for a similar indicator on *request level*, that specifies the request's type of assignment. The assignment could be:

- Standard: The assignment traditionally followed by the app. This type of assignment is what most clients use and it consists of creating an appraisal request and assigning one or more activities called valuations.
- Simplified: Similar to the standard assignment, after users create requests that have this type, they'll have the possibility to create one or more activities, the difference is the ability to choose the type of said activities, which can be generic or valuation.

It is worth noting that once the project is created, neither the type of workflow nor the possibility to enable simplified assignment can be changed afterwards.

4.3.3 Appraisal report

Another feature that was requested by certain clients and discussed internally was to update the generation of the appraisal report, aiming to facilitate the representation and exchange of data. The additional features were the inclusion of images, of all sizes, in the Excel workbook generated by the appraisal agents.

In order to perform this modification an external java library "Apache Poi" was used for reading and writing files in Microsoft Office formats like Excel. Other required operations were challenging such as manipulating the images dimensions to present them smoothly and correctly regardless of the original dimensions.

These changes were done on the main server application, as it was essential for all clients, and not specific for a certain client or project.

4.3.4 Integration

As mentioned previously, an older client (e.g. ABC) was working on a separate legacy integration application that needed to be merged into the main system. The strength of the current integration app is that it is designed for specifically this purpose, to adapt with such change without affecting its performance.

After the integration of ABC Bank, nothing changes from the perspective of the user, their internal application will still functions the same of course. The change was more internal and structural, the deployed web services the bank communicates with are are merged in the same system making testing, maintenance, requirement changes and general feature updates unified for all clients and optimizing the solution as a whole.

Below is a set of important implementations while integrating:

- Object mappers: Objects used by the system might vary from one project to the other, mappers are created to safely convert models to POJOs and vice-versa. Implementing the mappers is essential for ensuring communication between applications and databases.

- Exposed API: A common core interface implemented in the bank's integration application, with the main purpose of communicating with WAS.

- Invoked API: Another common core interface also implemented in the bank's integration application, used for communicating with or notifying the bank.

4.4 Team

The team in charge of this project is composed of:

- Project Manager: assigns tasks to the team members, schedules the activities and sets the deadlines, agrees the priorities with the customer and manages the project's costs.

- Development Team Leader: responsible for defining the technical solutions, as well as coordinating the developers and monitoring the code's quality, ensuring it's consistency and correctness. In addition, such individual would manage the deployments.

- Developers: responsible for implementing new features, bug-fixes, and hot-fixes. In this project we have both expert developers (Senior) and less experienced developers (Junior).

After going through the general technical training, it became possible to join the project's development team.
Upon the introduction of a project or a main feature, the project manager assembles the team, possibly after discussing the points with the Team Leader, for setting the project's timeline and resources for the upcoming period that can vary depending on the complexity and size of what's required.

4.5 Development/Implementation

In this chapter, we will first discuss the implementation process of this work from a high level, then we will explain the interesting parts of it in detail.

The development team responsible of this project works generally on a full stack developing both the server and client side of the system, we'll be talking about each of them separately:

4.5.1 Server-side

Using a structured Java Spring as mentioned above, developing a new feature would require us to follow our architecture's guidelines, meaning we have to create various components, including but not limited to models, services, controllers and the data access objects, with their equivalent in the database if necessary.

Models

In this Spring architecture, a Model is a serializable class used as a communication contract between the controller and the client interacting with it, usually using a different platform.

Models in Spring generally don't contain any application logic, unlike other approaches where the model acts as both the data contract and data access object, meaning it's closer to the data layer having direct access to it.
They might of course contain basic methods overriding (equals, contains, etc..).
Below is a code example of a model: Below is an example of a simple model used in this project:

```
    package it.project.model;
    public class ProgettoM {

        private int id;
        private String codice;
        private String descrizione;
        private Boolean flagVal;
        private Boolean flagClient;
        private Integer idClient;
        private boolean flagVisible;
        private boolean flagAbilitaIncaricoSemplificato;
        public int getId() {
            return id;
        }
        public void setId(int id) {
            this.id = id;
        }
        // other getters and setters

        // methods override
    }
```

POJOs & FP

POJOs are generally the objects passed to a query to be inserted into the database. While using FP as mentioned, POJOs extend a base class allowing several operations to be performed. The corresponding table name can also be specified within the class if it's different than the class name in lower-case.

```java
public void getPratica(int id) throws Exception {
    try {
        //FETCHING ONE RECORD
        Pratica pratica = new Pratica();
        if (pratica.select("id", id)) {
            ..
            //pratica now contains the record with the id
  provided
        }
        //FETCHING MULTIPLE RECORD
        BaseList<Pratica> praticaList = new BaseList<
  Pratica>(Pratica.class);
        praticaList.select();
        //praticaList should contain a list of Pratica
    }
}
```

A dependency called **Jackson** is used by Spring to serialize and to deserialize objects. For generic shallow conversion no additional annotations are required in the Model, while for specific cases, annotations on either field or class level can be added to satisfy our requirements. An example of these annotations would be the *@JsonIgnoreProperties("field1", "field2")* annotation that is added on class level and allows, as the name suggest, to ignore specified fields during the serialization and deserialization of the class. Models in this project follow a clear naming convention, which matches the format ClassName**M**.

Controllers

A user created component annotated with @Controller having the main purpose of handling requests. It contains a set of methods generally called actions. It also encapsulates the navigation logic and delegates the services for the service object.

Controllers and their actions are identified by at least the name, parameters, path, as well as the request method in order to indicate how this specific action could be reached, these last two can be specified using the annotation *@RequestMapping(method = RequestMethod.GET, value = "/path/to/users")*, providing the request method (typically GET/POST) and path (value) parting from the controller's main path.
Parameters type can vary depending on our diligence using the following two annotations:

- **@PathVariable:** used to extract any value which is embedded in the URL itself.
- **@RequestParam:** used to extract query parameters.

The latter being more useful on a traditional web application where data is mostly passed in the query while @PathVariable is more suitable for RESTful web services where the URL contains values (e.g. *https://{domain}/valutazione/100023*, here data, which is the appraisal's id is part of the URI.

On each request, Spring makes use of the same Jackson dependency that handles the received data and converts it implicitly into its equivalent POJO. This communication goes in both directions since the same method is used for converting data before returning it, as part of a REST controller for example.
In order to achieve this it is sufficient to add the annotations *@ResponseBody* and *@RequestBody* before our response and parameter data types respectively. The former indicates that a method return value should be bound to the web response body, while the latter indicates that a method parameter should be bound to the body of the web request.
In later versions (as of v4.0) these annotations could also be added on the class level in which case it is inherited and does not need to be added on the method level.
Below is a code snippet from an implemented controller, handling requests related to zones and municipalities (comuni):

```java
package it.project.controller;

@Controller
@RequestMapping("/comuni")
public class ComuneController {

    @Autowired
    ILogService logService;

    @Autowired
    ComuniMapper comuniMapper;

    @RequestMapping(method = RequestMethod.GET, value = "/")
    public @ResponseBody List<Comune> getListComuni(@RequestParam(value = "q", required = false) String q) throws Exception {
        try {
            if (!StringUtils.isEmpty(q)) {
                q = q.concat("%");
            }
            List<Comune> comuniList = comuniMapper.getComuni(q);
            return comuniList;
        } catch (Exception e) {
            logService.log("Error while fetching comune: " + e.getMessage() + LogUtils.getStack(e), LogLevels.ERROR);
            throw new Exception("Error while loading data");
        }
    }

    @RequestMapping(method = RequestMethod.GET, value = "/{id}")
    public @ResponseBody Comune getComuneByCodCat(@PathVariable("id") Integer id) throws ItemNotFoundException {
        try {
            Comune comune = comuniMapper.getComuneById(id);
            if (comune == null) {
                comune = new Comune();
            }
            return comune;
        } catch (Exception e) {
            logService.log("Operation failed " + id + ": " + LogUtils.getStack(e), LogLevels.ERROR);
            throw new ItemNotFoundException("Unable to fetch list " + id);
        }
    }
}
```

As seen above custom exceptions come in handy while implementing controllers to help troubleshoot problems.

Services

Services represent the service layer, where the business logic of the application usually resides. Services are annotated by **@Service** indicated that a class belongs to that layer. This annotation serves as a specialization of *@Component*, allowing for implementation classes to be auto-detected through classpath scanning. Normally the naming convention followed by service classes is ClassName**I** for the interfaces and ClassName for its implementation.
Below we can see the service interface and its implementation. The service contains one method searchConflictOfInterest(int) fetching data using our custom framework.
IConflittiInteresseService.java:

```
package it.example.service;

//imports

public interface IValutazioneService {

    public List<Valutazione> searchValutazione(ExcelProgetto excelProgetto);

}
```

ConflittiInteresseService.java:

```
package it.project.service;

// other imports
@Service
public class ValutazioneService implements
 IValutazioneService {

  @Autowired
  ILogService logService;

  @Value("${rank.indirizzo}")
  private int rankIndirizzo;

  @Autowired
  private ValutazioneMapper valutazioneMapper;

  @Value("${rank.header}")
  private int rankHeader;

  private List<Valutazione> searchValutazioneGeneric(
 Integer id, String nome, String cognome, String codice,
  String IVA, String indirizzo, BigDecimal lat,
 BigDecimal lng, Integer comune, String civico, Integer
 idProgetto) {

    // List<Valutazione> listValutazione = new
 ArrayList<>();

    Calendar data = Calendar.getInstance();
    data.set(Calendar.YEAR, data.get(Calendar.YEAR) -
 1);
    return valutazioneMapper.searchValutazione(id,
 nome, cognome, IVA, indirizzo, comune, civico, data,
 idProgetto).parallelStream()
        .filter(vcExt -> {
          if (vcExt.getIdValutazioneOrig() !=
 vcExt.getIdValutazione()) {
```

```
            if ((lat != null) && (lng != null) && (vcExt.getLat() != null) && (vcExt.getLng() != null))
              return true;
            else
              return comune.equals(vcExt.getComune());
          }
          return false;
        }).map(vcExt -> vcExt.toConvertIntoSimple()).collect(Collectors.toList());
  }

  @Override
  public List<Valutazione> searchValutazione(ExcelProgetto excelProgetto) {
    String nome = null;
    if (excelProgetto.getPersona() != null) {
        nome = excelProgetto.getNome();
    }
    return searchValutazioneGeneric(excelProgetto.getExcelRow(), nome, null, null, null, excelProgetto.getIndirizzo(), excelProgetto.getLat(), excelProgetto.getLng(), excelProgetto.getComune(), excelProgetto.getCivico());

  }
}
```

Dependency Injection One thing we failed to mention describing Spring is dependency injection, a pattern that allows us as programmers to inject objects into a class by using a container that is externally configured (often by an XML file), instead of letting the class directly instantiate the object.[16]
In the service above, the component *LogService* is injected simply using the annotation **@Autowired**, this way Spring creates a ready-to-use instance of the injected service. Similarly, Spring also allows us to work with values from a properties file with the **@Value** annotation (see *rank* above).

4.5.2 Client-side

As previously mentioned, Angular is a framework that offers the possibility to create web pages and mobile applications in a simplified manner making use of model declarations, dependency injection and data binding.

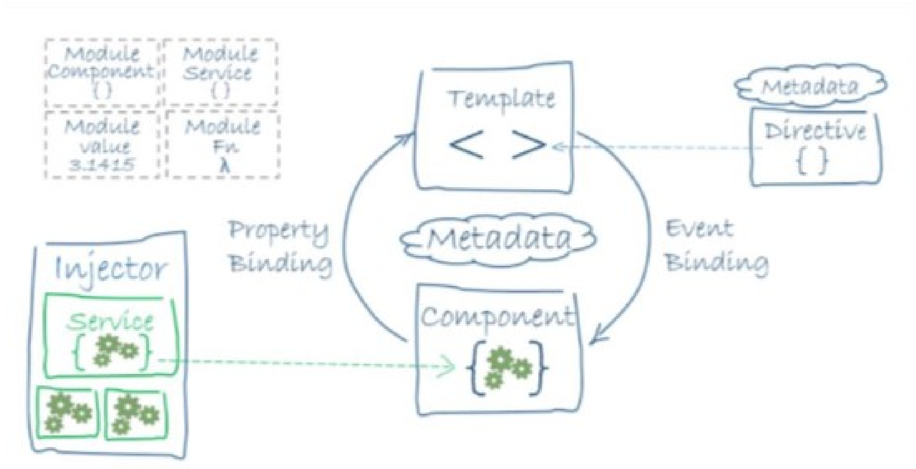


Figure 4.2: Component-based Angular

In the figure above we can observe how Angular functions: Each logical object found in the app (page, list, personalized button, etc..) would be registered as a "Component". We define a component's application logic—what it does to support the view—inside a class. The class interacts with the view through an API of properties and methods [10]. In this section we will go through the most notable angular features implemented in this project:

Services

Angular distinguishes components from services to increase modularity and reusability. By separating a component's view-related functionality from other kinds of processing, we make our component classes lean and efficient.

In order to make HTTP requests, a service makes use of HttpClient, an available injectable class with methods to perform specifically that in different request methods, most notably **get** and **post**. Below is a code snippet from an implemented service containing two methods and injecting and using the previously mentioned HttpClient:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable()
export class AziendaService {

  url: string;

  constructor (private http: HttpClient) {
    this.url = environment.apiUrl + 'aziende/';
  }

  getAziende(): Promise<ListResult> {
    return this.searchAdvanceAziende(null, null, null);
  }

  searchAdvanceAziende(filtri: any, page: number, rowsPerPage: number): Promise<ListResult> {
    let url = this.url;
    let q = '';
    if (filtri) q += this.toQueryString(filtri);

    q += (q !== '' ? '&page=' : 'page=') + page;

    return this.http.get<ListResult>(url).toPromise();
  }

```

The **get** method can return an observable of a type, which can be either directly returned by the service for subscription or converted into a Promise as can be seen above, which is a placeholder for a future value that serves the same function as callbacks but has a nicer syntax and makes it easier to handle errors.

Dependency Injection is wired into the Angular framework and used everywhere to provide new components with the services or other things they need. Components consume services; that is, we can inject a service into a component, giving the component access to that service class. The service in the example above shows also how we can make a component injectable, using the annotation *@Injectable()*
At least one provider of any service must be registered before we are able to use it. The provider can be part of the service's own metadata, making that service available everywhere, or we can register providers with specific modules or components. we can register providers on:

- **Service level:** in the metadata of the service, inside the @Injectable() decorator:

```
import { Injectable } from '@angular/core';
import { MyModule } from './my.module';

@Injectable({
    providedIn: MyModule
})
export class MyService {
}

```

- **Component level:** in the metadata of the component, inside the @Component() metadata:

```
@Component({
    selector: 'my-component',
    templateUrl: './my.component.html',
    providers: [MyService]
})

```

- **Module level:** in an App Module, more specifically the providers array of @NgModule():

```
import { AppComponent } from './app.component';
import { MyService } from './my.service';
```

```

    @NgModule({
      declarations: [
        AppComponent
      ],
      imports: [
        //
      ],
      providers: [
        MyService,
        //other providers
      ],
      bootstrap: [AppComponent]
    })
    export class AppModule {}

```

When we provide the service at the root level, Angular creates a single, shared instance of the service and injects it into any class that asks for it.

Route Guards

As then name implies, route guards provide a way to prevent navigation to a route. Angular offers various route guards, including *CanActivate*, *CanActivateChild*, *CanDeactivate*, *CanLoad* and *Resolve*. The most common of these hooks is CanActivate which is the one implemented in the example below:

```
  import { Router, CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';
  //imports

  @Injectable()
  export class AuthenticationGuard implements CanActivate {
    constructor(private router: Router) { }
    canActivate(
      next: ActivatedRouteSnapshot,
      state: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {
      if (localStorage.getItem('currentUser') || sessionStorage.getItem('currentUser')) {
          // logged in so return true
          return true;
        }
        // not logged in so redirect to login page with
 the return url
        this.router.navigate(['/login'], { queryParams: {
 returnUrl: state.url }});
        return false;
      }
    }

```

This guard is checking our browser's *localStorage*, which is a read-only property that allows us to access a storage object for the document's origin; the stored data is saved across browser sessions.

Several ways exist for configuring route definitions in Angular, which is where we associate our route guard with a component page. The most trivial way of doing so would be creating route definitions within the *AppModule* file. All we need to do is import the *RouterModule* library, then build our routes according to our needs such as the following:

```
        RouterModule.forRoot([
        {
            path: '', component: MainpageComponent,
 canActivate: [AuthenticationGuard],
            children: [
              { path: '', component: HomeComponent,
 canActivate: [AuthenticationGuard] },
              { path: 'valutazioni', component:
 ValutazioniListComponent, canActivate: [
 AuthenticationGuard] }
        }

```

In this case all above routes are inaccessible to an unauthenticated user, while it is sufficient to remove the *canActivate* method to make the page publicly available.

HttpInterceptors

An HttpInterceptor is an interface that can be implemented by a class and it has only one method called *intercept* that intercepts an outgoing HttpRequest and optionally modifies it or the request. Typically an interceptor will transform the outgoing request before calling *next.handle(request)* passing the request to the next interceptor in the chain.
In this project an interceptor was used to attach a user authentication token (JWT) to the request's header before sending it to the server allowing it, after having received the request, to extract the token and verify it granting or forbidding the client access depending on its validity. Below is an example of our implemented Interceptor:

```
        // imports
        @Injectable()
        export class ApiInterceptor implements
   HttpInterceptor {

          tokenSubject: BehaviorSubject<string> = new
   BehaviorSubject<string>("");

          constructor(
            private authenticationSrv:
   AuthenticationService,
            private router: Router
          ) { }

              addToken(req: HttpRequest<any>, token:
   string): HttpRequest<any> {
                return req.clone({ setHeaders: {
   Authorization: 'Bearer ' + token } });
              }
              intercept(req: HttpRequest<any>, next:
   HttpHandler): Observable<HttpEvent<any>> {
                //Aggiungo l'Authorization header JWT solo
    se sto chiamando le REST API
                if (req.url.startsWith(environment.apiUrl)
   ) {
                  const token = this.authenticationSrv.
   getAccessToken();
```

```
                  if (token != null) {
                    // Clone the request to add the new header and pass the cloned request instead of the original request to the next handle
                    return next.handle(this.addToken(req, token)).catch(error => {
                      if (error instanceof HttpErrorResponse) {
                        switch ((<HttpErrorResponse>error).status) {
                          case 400:
                            return Observable.throw(error);
                         //error handling
                        }
                    });was
                  }
                }
                return next.handle(req);
              }

    logoutUser() {
      this.router.navigate(['/login']);
      return Observable.throw("Cannot refresh token, route to the login page");
    }
}
```

4.6 Infrastructure

The system as a whole is technically composed of different projects or applications. Mainly the decomposition can be seen in the figure below:

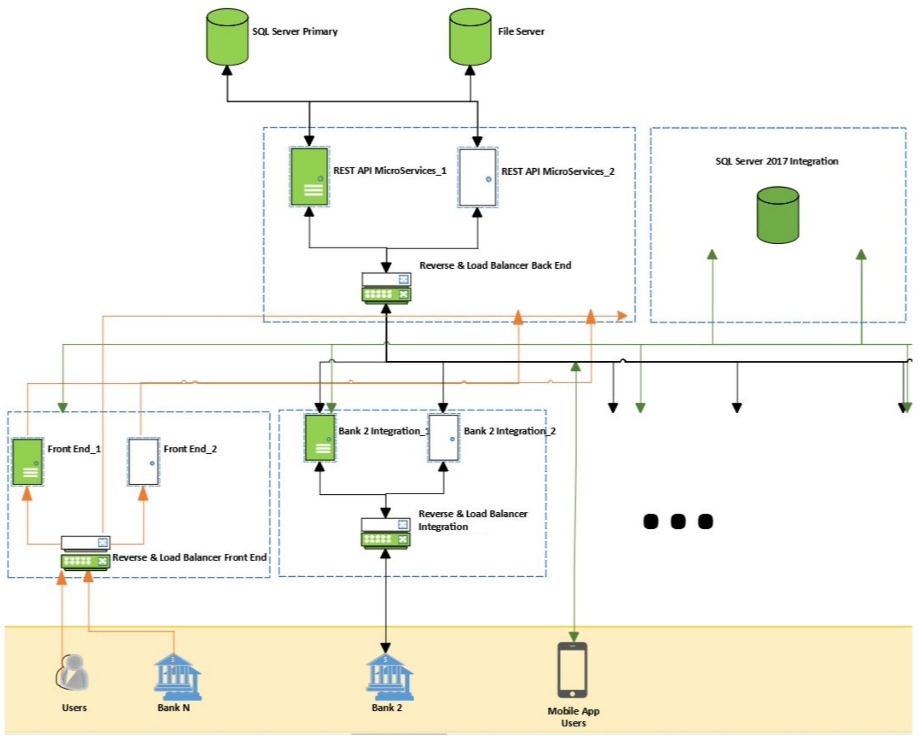


Figure 4.3: Figure showing the infrastructure of the system

Load balancing is efficiently distributing incoming network traffic across a group of back-end servers, As can be seen above we can find a balancer in each of the Angular front-end, WAS server and each bank integration app. IT teams are increasingly relying on server load balancers. To increase efficiency of application delivery to end users for a reliable application experience. More specifically for:

- Increase Scalability: load balancers are able to spin up or down server resources based on spikes in traffic to the pool of servers that are best suited to handle these increases in traffic and keep applications performance optimized.
- Redundancy: Using multiple web servers to deliver applications or websites provides a safeguard against the inevitable hardware failure and application downtime. When server load balancers are in place they can automatically transfer traffic to working servers from servers that go down with little to no impact on the end user.

- Maintenance and Performance: Business with web servers distributed across multiple locations and a variety of cloud environments can schedule maintenance at any time to improve performance with minimal impact on application uptime as server load balancers can redirect traffic to resources that are not undergoing maintenance.

One of the common obstacles faced when working with a load balancer is managing user session. By default, a Classic Load Balancer routes each request independently to the registered instance with the smallest load. However, the use of the sticky session feature (also known as session affinity), enables the load balancer to bind a user's session to a specific instance. This ensures that all requests from the user during the session are sent to the same instance [17].

4.6.1 Databases

We differentiate between databases which WAS communicates with:

- - Primary core database: The main data store in this system. Contains all data needed by the web application.
- - Bank integration databases: Specific for each bank, with a particular structure that differs from one client to the other depending on their specifications and internal system's structure and logic.
- - Mobile application database: Data specifically related to the mobile application.

History

Upon every action modifying pre-existing data, such as update, deletion and state change, it is crucial to keep a copy of modified data.
Archiving was used creating additional tables for every state-dependent entity and a trigger that runs upon a table's modification, to avoid confusion such tables follow the convention T_EntityName for the table and T_EntityName_**ST** (ST is short for Storico, meaning Historical in english).

Domains

Since the options in predefined lists (domains) in the web application might differ from a project to another, it was seen beneficial to store its data on the main database. The options are customizable on project creation, providing a default template of domains, considered minimal for running the system properly.

4.6.2 Integration

The system integration is technically composed of a Java module communicating with both WAS using REST calls, and with the specified bank's internal application.

Each client or bank would have its own integration application, the communication between the latter and the bank itself is done through SOAP, with the request WSDL issued by the bank being identical in both directions.

The communication between the bank, the integration application and the bank's internal application is shown in the figure below:

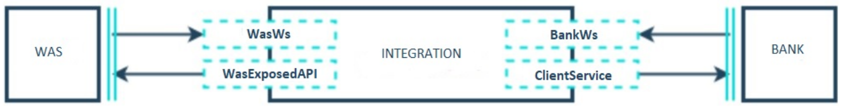


Figure 4.4: Figure showing the communication between different parties in the system

The web service *WasWs* which is exposed to **WAS** relies on Apache CXF and implements the interface InvokedI: this web service handles calls from the core application.
The implemented interface has basic methods that seem essential to every integration application, most notably the method *notify* responsible of notifying the bank of any occurring action.
This communication with the bank is done through SOAP services called from the server and sending messages directly to the bank.

Chapter 5

Conclusion

This project started with studying different technologies and business processes, then analyzing clients requirements and consulting with a full team of developers to be able to achieve the results we were working towards.

The study and development of a system of such complexity was a long, valuable process due to the various technologies utilized from architecture to languages and server infrastructure, which allowed us as developers working on it to expand our base of knowledge, both technically and business-wise.
The in-depth look into the microservices architecture and working with a practical demonstration of it perfectly portrayed its points of strengths and capabilities, and how different approaches are taken depending on the architecture and methodology followed.
As was long-established in previous studies, it proved necessary to divide the project into different components and communicating parties, then working on the development and integration of each of these parts separately ensuring a smooth and efficient development process.
Another important remark to be highlighted is the necessity of working in a well documented and ordered manner. While this point is always stressed-on in software development teams, it is often ignored in practice or inadequately applied throughout the project life-cycle, causing it sometimes to be counter-productive.

Bibliography

[1] Victor Szalvay. «An introduction to agile software development». In: (2004). URL: http://www.danube.com/docs/Intro_to_Agile.pdf (cit. on p. 5).

[2] S Balaji and M Sundararajan Murugaiyan. «Waterfall vs. V-Model vs. Agile: A comparative study on SDLC». In: *International Journal of Information Technology and Business Management* 2.1 (2012), pp. 26–30 (cit. on p. 7).

[3] Atlassian. *Comparing Workflows.* URL: https://www.atlassian.com/git/tutorials/comparing-workflows (cit. on p. 10).

[4] Martin Fowler and Matthew Foemmel. *Continuous integration.* 2006 (cit. on p. 12).

[5] Alan Berg. *Jenkins Continuous Integration Cookbook.* Packt Publishing Ltd, 2012 (cit. on p. 12).

[6] Enterprise Team Nicholas Kassem. «Designing Enterprise Applications with the Java(TM) 2 Platform (Enterprise Edition)». In: (Oct. 2000), pp. 10–12 (cit. on p. 14).

[7] David Flanagan. «JavaScript: The Definitive Guide». In: (Nov. 2001), p. 7 (cit. on p. 16).

[8] Gavin Bierman, Martín Abadi, and Mads Torgersen. *Understanding TypeScript.* Vol. 8586. July 2014, pp. 1–2 (cit. on p. 16).

[9] Ken Henderson and Ron Soukup. *The Guru's Guide to SQL Server Stored Procedures, Xml, and HTML with Cdrom.* USA: Addison-Wesley Longman Publishing Co., Inc., 2001, p. 281. ISBN: 0201700468 (cit. on p. 17).

[10] Microsoft. *Introduction to components.* 2020. URL: https://angular.io/guide/architecture-components (cit. on pp. 17, 44).

[11] S.K. Kasagoni. *Building Modern Web Applications Using Angular.* Packt Publishing, 2017. ISBN: 9781785880032. URL: https://books.google.it/books?id=qnc5DwAAQBAJ (cit. on p. 18).

[12] Craig Walls. *Spring Boot in action.* Manning Publications, 2016 (cit. on p. 18).

[13] Snehal Mumbaikar, Puja Padiya, et al. *Web services based on soap and rest principles*. Vol. 3. 5. 2013, pp. 1–4 (cit. on p. 21).

[14] J Lewis and M Fowler. «Microservices Guide». In: *martinfowler.com* (2016). URL: `martinfowler.com` (cit. on p. 24).

[15] Microsoft M. Jones. «JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants». In: (May 2015). ISSN: 2070-1721. URL: `https://tools.ietf.org/html/rfc7523` (cit. on p. 26).

[16] Ekaterina Razina and David S Janzen. *Proceedings of the 11th IASTED International Conference on Software Engineering and Applications: Cambridge, MA*. 2007, p. 7 (cit. on p. 43).

[17] Amazon. *Elastic Load Balancing*. URL: `https://docs.aws.amazon.com/elasticloadbalancing` (cit. on p. 52).