Legal Services Platform

João Robalo
joao.t.robalo@gmail.com

Instituto Superior Técnico, Lisbon, Portugal

September 2020

Abstract

This thesis proposes to build a platform that will work on the Web - Actio, which will aim to facilitate the population's access to legal support. This platform had, as main requirements: the publication of a case by any individual who needs legal support; consultation of published cases by lawyers and their interaction with the individual in question; finally, the agreement with the Code of Ethics of the Bar Association. This thesis refers to the development of the back-end module. In order to achieve this objective, in a first phase, the market in which the platform is inserted was studied, meetings were held with several lawyers, with the Bar Association and a survey was conducted with 100 people. After this first phase, we moved on to the technological framing phase, studying various technologies from different fields necessary for development, then a solution proposal was elaborated, specifying the solution to be implemented. This was followed by a RESTful solution that is hosted on Amazon Web Services (AWS) using the Elastic Beanstalk service that runs the servers implemented in Node.js, using MongoDB Atlas, as the provider of the MongoDB database. Then, performance and scenario tests were performed, which were successful. In summary, part of this work was the identification of a problem (access to legal support), study of the market where the problem is inserted, projection of a solution, implementation of it, going through a test phase, the final product being a solution technology that aims to help the population.

Keywords: Actio; Web; Restfull; MongoDB; Node.js; AWS.

1. Introduction

The present work had as its final objective, the implementation of the back-end of a platform that reduces the barrier between people who need legal support and the people who provide this service. A platform that facilitates the search for the right lawyer, and helps lawyers to proliferate their practices, Actio. Thus, Actio is a platform focused on innovation and modernization of the legal sector, the goal is to publish a web service, accessible by a normal browser, a service supported by servers hosted in the cloud, with a database also hosted in the cloud, it is necessary that the servers are not exposed to the Internet, nor allow SSH connections and that the database is only accessible through the servers. Within the scope of the work is a market study, and consequent survey of requirements, the development of the servers, the database, the entire configuration of the infrastructure that has to be scalable, elastic, available and secure. It is an objective to use tools so that any change to the code triggers an automatic update action of the platform, without affecting its availability, in order to be able to manage its versions transparently to users. All the choices of technologies and tools to use, from programming language, to the cloud service provider, types of database and everything else involved in this development of software, was the product of a study that will be go deeper into the framing section and the resulting solution proposal. The implementation and its results will also be addressed with scenario and performance tests.

Thus, in a context of outlining objectives, it was intended:

- To study the market in which the platform is inserted;
- Analyze different languages and choose the most suitable for the implementation of the Actio server;
- Study different databases and choose the most appropriate for the implementation of the database;
• Evaluate different forms of communication and client-server architecture;

• Evaluate different services from different cloud providers, both for servers and for the database, choose and later configure a scalable, elastic, available and secure environment.

• Configure, if possible using the services of the chosen cloud provider, a code management environment and respective platform update without impacting its availability.

• Subsequent development of the platform’s back-end (servers and database) and tests (scenario and performance tests).

2. Related Work

2.1. Market Stakeholders and Problem

The market in which the platform is inserted is the legal one. In this sense, there are four fundamental clusters to name: the people who seek legal services, the professionals who provide them, the companies providing legal services and, finally, the entity that regulates the members of these service providers. It is essential to add that the market to which the platform is directed is wide. In this sense, in a perspective of problematization, before the aforementioned information, two questions are formulated: What do people do to find lawyers? And what do lawyers do to find new clients? In order to answer the first question, a survey was carried out with 100 people between the ages of 18 and 60, but with a greater focus between 20 and 39. To answer the second question, he met not only with several lawyers, as well as with the Bar Association.

Survey of potential customers

The main objective of the survey was to understand and analyze clients - people who seek legal services - and how they proceed to seek legal support, when faced with a legal issue. The survey is as follows:

1. What is your age? 3% 18-20; 64% 21-29; 22% 30-39; 3% 40-49; 7% 50-59; 1% 60+ 2. Have you ever needed legal support? 72% no and 28% yes 3. If you needed legal support, what tool would you use to find a lawyer? 70% family and friends; 26% internet; 2% I don’t know; 1% Bar Association; 1% Law firm. 4. Have you ever used the internet to resolve legal issues? 50% yes and 50% no 5. How likely is it to use the internet to find a lawyer/firm? 14% Not likely; 25% Not likely; 21% Moderately Likely; 22% Likely; 18% Very likely 6. How many law firms do you know in Portugal? 8% 0; 70% 1-5; 14% 6-10; 8% 10+ 7. Would you ever use an online platform that facilitates contact with lawyers/firms? 75% yes and 25% no 8. Have you ever failed to solve a legal problem because you find the process of finding a legal solution complicated? 50% yes and 50% no.

Meetings with Lawyers

The meetings with the lawyers constituted the addition of a more practical perspective, since they involve the opinion of the legal service providers. The objectives of the meetings were, always, to understand the difficulties that the different levels of the market of legal service providers feel in their day-to-day, what are the problems in proliferating their practice and what is the area of the market where they are inserted, as well how to understand your interest in a platform with the potential of Actio. It was found that the market where platforms of this type attract a less valuable market per case, and thus excludes large societies, which, by accepting smaller cases, ends up spending effort and resources in cases with less return. However, the least valuable market exists and it has been confirmed that it is precisely at this point that there are more problems, regarding the barrier between clients and lawyers. Thus, the first conclusions we draw from the meetings are that the market that Actio will attract is that of the smallest cases and of the lowest monetary value (example: traffic tickets or the creation of a startup), but it is exactly in this field where there is more difficult to find a suitable lawyer, making it also a bigger problem and, consequently, a need to boost the growth of smaller companies and self-employed lawyers who are interested in these cases.

2.2. Databases

In order to implement a platform, such as the one proposed in this Dissertation Project, it is necessary to build a database that stores all data that reflect the state of all objects that intervene in the proposed solution. This section will cover relational and non-relational databases.

Relational Databases

In a relational database, the structural foundation will be the data model, which is defined as a set of conceptual tools that describe and relate data. It describes, both physically and logically, the architecture of a database. The data are organized and distributed in this static structure and previously defined. The relational database technology studied for this purpose was SQL based databases. The SQL language is commonly used in relational database systems. As it is a high-level language, it is considered an easy-to-interpret language. When manipulating the database, this language removes complexity from the process, since it is a
high-level language, where the manipulation effort goes through the declaration of what is intended to be accomplished, without having to specify how the process will solve the query. Contrary to lower level languages, which are faster to execute, but of greater complexity.

Non-relational Databases

In a relational database, its previous and static structure creates a limitation if there is a need to have a set of stored data that evolves from a structural point of view. Thus, non-relational databases were created in order to overcome this problem, which is caused by the dynamic evolution of the data. The most widely used non-relational database is MongoDB. In order to understand what non-relational databases are, we will then address some key points regarding their structure, association of records, storage organization and replication technology. The non-relational database studied for this purpose was MongoDB. MongoDB is a non-relational database with a variety of features that, in the open source NoSQL world, are difficult to overcome. In this database, JSON (JavaScript Object Notation) documents are handled natively. The names of properties of JSON documents, as in XML, can consist of very detailed text. However, MongoDB uses its own storage format, BSON (Binary JSON). This reduces, unlike JSON documents, the amount of space and processing required to store them. This binary format allows efficient serialization which is useful for both storage and transmission over the network. This operation of manipulating the documents is done by MongoDB and by the database user’s drivers, it is not necessary that the programmers have to take into account this characteristic of the implementation.

[1, 2]

2.3. REST

Roy Fielding gave the name REST (Representational State Transfer) to this style of architecture for distributed systems. This style, aims to define how the Web should work. This model does not operate directly on resources, but promotes the transfer of representations of resources. HTTP is very important in REST since the communication between client and server is done through HTTP, using its methods. This model uses stateless interactions, this means that when a server processes requests it does so taking into account, exclusively, the resource and the information inherent to the order itself, not depending on information from previous orders. That is, the server does not need to maintain state information between requests. In REST systems, resources are passed between client and server, and all of them are identified by the same mechanism, each resource is always identified by a URI (Uniform Resource Identifier), which in turn is defined as a unique identifier of a resource in the Internet. Thus, with HTTP methods, namely CRUD methods (GET, POST, DELETE and PUT) and a URI, a client makes requests about resources. The expected semantics for each of these CRUD methods must be taken into account when deciding which method is appropriate for a given action. Methods such as GET, PUT and DELETE, for example, must be idempotent. GET is a simple reading of a resource. The DELETE method is used for customers to delete a resource, whereas the PUT method is for a customer to replace a resource’s state with a new state. Although both (DELETE and PUT) obviously have effects on the state of objects, they are idempotent because calling them repeatedly has the same effect as calling them once. POST can be done to perform virtually any action, but on systems that work with REST - Restful, it is usually used to create or extend resources. In conclusion, REST introduces a series of standards that, when correctly applied in a distributed system, generate desirable attributes such as scalability, uniformity and performance. It is also added that this style of architecture for distributed systems is widely used on the Web, and therefore, there is a lot of supporting documentation for the development of Restful applications.

[3, 4]

2.4. Server Languages

Programming languages have evolved over time, either through updates and new versions, or through the appearance of new languages. Thus, Web development has seen several innovations over the years. In order to implement the server, which will make the features available to users of the platform, we will address some languages that are available to any programmer to be able to implement the server. Different languages have different specializations, as such, they respond to different requirements, so it is considered extremely important the approach that will be taken next.

PHP

PHP (Hypertext Preprocessor) is the technology that underlies millions of Web applications. PHP is a language that has a wide range of features, an accessible syntax and supports several operating systems, which makes it ideal for a fast and agile development of the Web. Its success comes from the fact that it is a scripting language for the Web and a means to process HTML, as well as creating web pages, being quite friendly. Other favorable points are the fact that PHP knows several Internet protocols, being able to talk to any number of
databases, simplifying the analysis of browser data and being able to carry out HTTP requests.

**Python**

Python is defined as a scripting language. Thus, this language is generally applied in scripts, in such a way that, often, the word script is applied, instead of a program when referencing a Python file. However, because it has different meanings, not all programmers apply the term “scripting language” to Python. The best association with the term scripting language is the reference to a simple language for quick coding. With Python, this appreciation is very real, since in most cases Python allows for faster development than with other compiled languages. But Python is not just for simple tasks, the fact is that, due to its flexibility and ease of use, it simplifies tasks. With Python programs are allowed to increase their sophistication as necessary, although it has a set of simple features. In conclusion, Python is used for both simple and fast tasks and long-term development.

**Node.js**

Development in Node.js is done in the Javascript language, which is a language most commonly used in front-end development. Asynchronous event oriented, Node.js is an open source platform created based on Chrome’s JavaScript runtime engine (V8) to easily create fast and scalable platforms on the Web. The Web platforms in Node.js are written in JavaScript. Node.js is supported by Microsoft Windows, Linux and OS X. It is also added that the use of Node.js for web development is in an increasing trend, and according to the 2018 annual survey by Stackoverflow, 50% of the surveyed developers responded that they work with Node.js.

3. Proposal

Taking into account the aspects observed, the contents identified, the causes appreciated and discussed, the elaboration of this dissertation project led to the production of the proposed solution described in this section.

3.1. Concept and Requirements

With the modernization of the law market, through its digitalisation, a greater flow of clients is being offered to lawyers, proliferating their practice, as well as making people a powerful tool for claiming their rights, which by sometimes go unnoticed by lack of information. In an era where internet use is daily, it is considered important to offer an alternative in this medium that certainly facilitates reciprocal access from one party to another. This technology does not presuppose an innovation in technical terms, however it aims at the same degree of innovation, when applied in the market in question. In this sense, innovation involves the application of a digital and modern model, in a market that needs it. This model will be in conformity with the doctrines of those who regulate this market, and this is a challenge in question, so this solution is presented, not as a way to deceive the deontological code of the legal profession, but as a way of bringing it to the digital world.

The global requirements consist of: 1. An online platform that allows people who need legal support to publish a case; 2. The existence of a group of lawyers registered on the platform, who consult the published cases and approach the client if they so wish; 3. Bring together clients and lawyers, in a way that continues to respect the doctrines of the Bar Association.

3.2. Technology

This architecture divides the process into three layers, in this case users of the application, server and database. Users of the application make
requests through a browser to the server, and the server, in turn, will process these same requests, whether or not (depending on the requested action) to manipulate the database to provide the service correctly, the database data is the layer that stores the state of the objects that are part of the application.

Frontend
This component will be developed in another project, which will perform HTTP requests, following the REST architecture, to the server and process the responses obtained. Since this work only addresses the issues related to the concept, and the backend (server and databases).

PaaS
The PaaS to use will be Amazon Web Services’ Elastic Beanstalk. All the services mentioned (Amazon Web Services - Elastic Beanstalk, Heroku and Google Cloud - AppEngine) have the features to respond to Actio’s needs. So, the factor that weighs in choosing the Elastic Beanstalk is the fact that we have visibility and control of the infrastructure layer (load balancer, auto scaling group of virtual machines and everything else).

Communication
Communication will be done through HTTP requests. For the implementation of the platform, it is imperative to safeguard the integrity of the information. Knowing that HTTP uses reliable data transmission protocols, thus ensuring that they do not suffer damage at the time of transmission, it is admitted that using HTTP, it will be possible to have a greater focus on the logical rules and details of Actio, since this protocol allows the programmer not to focus too much on the failures and weaknesses of Internet communication. Thus, one will resort to HTTP requests, using a TLS (Transport Layer Security) communication which means that the communication will be over HTTPS. Actio will be a RESTful application, that is, it will use the REST architecture. This architecture is considered a good choice, since it defines a set of standards that, at the outset, facilitate the development of Web platforms with good performance.

Language
The Actio server consists of the layer that processes HTTP requests from the front end. This server will be implemented in Node.js. This language was chosen because from the point of view of processing capacity it is sufficient to meet the requirements of the server, and from the point of view of popularity it is a technology that in the world of Web development is being used more and more and in an increasing trend. In addition, there is an interesting point in developing the platform in Node.js, the migration of code to the browser. That is, if for reasons of performance, it is concluded that there are certain processes that should occur in the browser and this did not happen in an initial development, the use or migration of the code already developed is easy if that code is already developed in a language that browser interprets as PHP or Javascript (as with Node.js), a fact that does not happen with Java and Python. At this point it is added that a test project was made, which consisted of a user authentication system, in PHP (Appendix C), in order to experiment with working with the language, it was concluded that, like Javascript, is a very accessible language with the necessary resources to respond to Actio’s needs. However, given the emerging and growing popularity of Node.js, it was decided to choose it for the server implementation.

Database
The Actio database is the layer that stores the status of all objects that make up the platform. In order to represent an object (for example a lawyer) in the Actio database, it is known from the outset the data that are necessary to do so and in this representation in no case will dynamic metadata be needed. However, Node.js has modules that speed up development with MongoDB. It is also possible, using these same modules, to implement a database in MongoDB where the objects obey a specific structure, given the Node.js approach and taking into account the ease that this language presents in handling MongoDB justified. the choice of this database for the platform.

4. Implementation
This chapter will address the stage of realization of the platform. As previously designed, the platform backend will consist of a MongoDB database and a server developed in Node.js hosted on Amazon Web Services Elastic Beanstalk.

4.1. Database
In this subsection the platform’s database will be addressed, it was developed in MongoDB. As already mentioned, the choice of this database was based on the fact that Node.js modules to manage MongoDB facilitate development and not on its non-relational database characteristics, so the first step was to define the various entities needed to meet the functional requirements of the
platform. The Node.js module used to manipulate the database was mongoose.

1. Figure 1: Database UML

Using mongoose schemas and mongoose models it was possible to design a database that will always obey a certain structure. As can be seen in the UML presented, there are 5 collections in the database, lawyers, clients, cases, approaches and messages.

4.2. Access management

Like most web platforms, there is also user registration, authentication, authorization, cookies, and user credential management. At this point, we will discuss the workflow of the access to accounts (registration, login, recover password, life cycle of cookies). The validation of data for the routes will be mentioned several times, these validations are made using the Joi module, which allows checks of the length of strings, regular expressions (valid emails, prohibited characters, etc.) and even logical validations using custom logic implemented by the programmer. It should be noted that on all implemented routes, all fields that the server receives from the browser are validated in a first logical layer using Joi before any type of manipulation to the system, and if any parameter fails to validate a message from error is returned to the frontend, so that it deals with the error a way that allows the platform to function properly.

Cookies

Every time a user logs in or registers a cookie is created. This cookie is a Json Web Token whose data is the user id (whether lawyer or client), the cookie’s creation date and the cookie’s expiration date (seven days after its creation). This cookie is an HTTPS only cookie, which means that it is only sent to the server and helps to mitigate the risk of client side scripting to access the cookie. Cookies still have a life cycle, expire and are renewed accordingly. The diagrams in the following subsection (middleware) summarize the life cycle of cookies.

Middleware

Middleware is a logical layer that runs (only when called) before the route code itself. Each route has its purpose, context and consequently implementation. Thus, a layer was implemented that performs one of three purposes when called, one directed to clients, called when client routes are accessed, another, directed to lawyers, called when lawyer routes are accessed and the third purpose is the case when someone accesses the landing page of the application.

2. Figure 2: Example of a middleware workflow, this happens before any lawyer route

Register

There are 3 types of registration on the platform, 2 for clients and 1 for lawyers. The client can
associate his registration with the submission of a case, this route is present, because for the sake of UX UI, it was decided to allow the submission of a case directly from the landing page. For that, there is a form that allows you to fill in the data to directly submit a case and create an account simultaneously (Registration associated with a case submission). The customer can also make an isolated registration of his user without associating a case to his account right from the start. Lawyers have only one way to create an account on the platform. In any of the routes is necessary to save the customer’s or lawyer’s credentials, for this the bcrypt module was used where a random salt is generated and concatenated with the password provided, then a hash of that concatenation is made and saved in the database. This process allows, in the future, the module to be able to validate the saved hash and the original password using the compare method as mentioned later in the login.

Login

The action of starting a conventional session is the same for clients and lawyers. For UI UX reasons, it was decided that any user logs on to the same site and in the same way. Thus, there is only one route (used for both clients and lawyers), for this purpose and its endpoint is / api / login. Firstly, as with all other routes, the data on the login form is validated. Then, if the fields are well constructed and according to expectations, the user in question can be found via email. Then, it is necessary to validate the credentials, as mentioned in the user registration routes (client or lawyer) the bcrypt module was used to deal with this point. Thus, we use the compare method present in the bcrypt module between the hash of the password concatenation with the random 10-character string (salt) that we have stored in the database and the password that is on the customer form. Finally, and depending on the validation of the password, it is verified whether the user is a lawyer or a client, we generate a lawyer or client cookie respectively and redirect the user to their Homepage. It should also be noted that the process of logging out, consists of a specific route (/api/logout), shared between clients and lawyers, which clears the browser cookie, after which the user is redirected to the platform’s landing page.

4.3. User Functionalities

Here, some key features available to users of the platform will be addressed. The validation of data for routes will be mentioned several times, these validations are made using the Joi module. Here, it is also important to mention that in all implemented routes, all fields that the server receives from the browser are validated in a first logical layer using Joi before any type of manipulation to the system, and if any parameter fails validating an error message is returned to the frontend, so that it deals with the error in question in whatever way is necessary for the platform to function properly.

Lawyer

- **Homepage**: This route returns the homepage to lawyers;
- **Open Approached Cases**: This page contains the same information as the lawyers’ Home page, with the difference that instead of containing all cases that have either the same region or legal area of the lawyers’ definitions, here only the cases that the lawyer has already addressed;
- **Approach Case**: This route allows lawyers to address a case;
- **Edit Profile**: Here it is possible to access and edit the profile settings of the lawyer;
- **Delete Account**: The purpose of this route is to delete the lawyer’s account and all its dependencies;

Client

- **Homepage**: This route returns the homepage to clients;
- **Submit Case**: This route allows clients to submit new cases;
- **Edit Case**: This route allows clients to update previously submitted cases;
- **Delete Case**: This route allows clients to delete previously submitted cases;
- **Open Messages**: When a client submits a new case, it has no approaches. One approach contains a mailbox, that is, a list of messages. This route allows clients to see those;
- **Edit Profile**: Here it is possible to access and edit the profile settings of the client;
- **SMS Notifications**: This feature is activated when a client wishes to receive an SMS notification that a lawyer has addressed their case;
- **Delete Account**: The purpose of this route is to delete the client’s account and all its dependencies;
4.4. Infrastructure

As mentioned in the solution proposal, the approach taken for hosting the platform is the Amazon Web Services AWS Elastic Beanstalk service. So the following configuration was made: A Virtual Private Cloud (VPC) was created. A VPC is also associated with an AWS region. Each region has several availability zones. Availability zones are data centers completely separate from each other, which in the event that an entire data center fails, there is another one to support redundancy in solutions. This redundancy is not achieved by default, and certain configurations are accurate. In this case, VPC was configured, namely the subnets and later Elastic Beanstalk to support an infrastructure with redundancy between two to three availability zones, it should be noted that a solution with two availability zones has an SLA of 99.999%. In this case, six subnets were created, three public and three private. Three have an internet gateway (public), and another three have a NAT gateway (private) and a public IP has been allocated to the NAT Gateway. Regarding the architecture chosen in this service, a load balancer was configured, which is launched in the three public subnets in order to be exposed to the Internet and always working even if two availability zones fail, which means that the load balancer has a higher availability to 99.999%. It should be noted that this is the only resource of the elastic beanstalk exposed to the Internet. This load balancer also has two listeners, one listening on port 443 over HTTPS and the other listening on port 80 over HTTP that redirects traffic to HTTPS. For the load balancer to support HTTPS, a certificate was generated and saved in the Amazon Certificate Manager service, so that the load balancer can use it later. Regarding the architecture chosen in this service, a load balancer was configured, which is launched in the three public subnets in order to be exposed to the Internet and always working even if two availability zones fail, which means that the load balancer has a higher availability to 99.999%. It should be noted that this is the only resource of the elastic beanstalk exposed to the Internet. This load balancer also has two listeners, one listening on port 443 over HTTPS and the other listening on port 80 over HTTP that redirects traffic to HTTPS. For the load balancer to support HTTPS, a certificate was generated and saved in the Amazon Certificate Manager service, so that the load balancer can use it later. In this case, two security groups were set up, the first for the load balancer, which, as inbound, blocks everything other than HTTPS traffic on port 443, and HTTP traffic on port 80 (HTTP traffic will be redirected to HTTPS by the listener load balancer), as outbound the load balancer is allowed to forward traffic to the group of machines. In the case of the security group of the machine group, as inbound, only traffic coming from the load balancer is accepted, everything else is blocked. Which means that in addition to the machines not being exposed to the Internet because they do not have a public IP, they only accept traffic that comes from the load balancer which in turn only accepts HTTPS traffic, this makes the machine only exposed to HTTPS requests that they pass through the load balancer, making it impossible, for example, to make direct SSH connections to them. In the case of machines, the outbound is open everywhere (0.0.0.0/0), since in addition to these needing to access public services such as the Nexus to support SMS notifications, the outbound does not pose any danger to the machines. It should be noted that no databases have been launched in the elastic beanstalk service. There is a service from MongoDB itself (MongoDB Atlas) that works as a SaaS for MongoDB, so managing the database becomes much easier. This service, in addition to the credentials to access the database, allows to make a whitelist of IP addresses, with the public IP address of NAT Gateway being added and thus only the resources that are running in the private subnets (those served -res), so those using the NAT Gateway as a public IP address are able to access the database. An automation for updating the platform in case the production branch is changed has been developed. The way the update is carried out has been configured so as not to affect the application's SLA, in other words the pipeline doubles the machine fleet, updates the new fleet and only after these new machines are updated does the new fleet register on the load balancer and deletes the group of machines running the previous version, so while the application is being updated everything continues to work and when there is a fleet of machines ready with the new version, the servers are replaced.

5. Results

The evaluation of the work will be done through tests, the purpose of which is to demonstrate that the program behaves the way it is actually said to behave; and to detect errors before the product enters a production phase. Tests are considered fundamental, as they allow us to validate and verify the system.

5.1. Performance tests

Performance tests are used to measure the efficiency and reliability of the product for different situations, whether stressed or not. In this case, the performance tests were divided into 3 routes that alone manage to populate the database,
namely, registration of a client associated with a case sub-mission, registration of a lawyer and approach of a lawyer to a case. It should be noted that all these tests were performed with only 1 server (which can always automatically scale), which is the second weakest virtual machine from Amazon Web Services (machine that is even covered by the free tier), that is, it is an instance of the type t2.micro with 1 vCPU and 1 Gb of RAM, and with the database also in the free tier regime. In other words, the system was tested with weak infrastructure, and with a small upgrade in the database, this component can be improved without any impact on the system.

**Registration of clients associated with a case submission**

In this case, 5000 clients and 5000 cases were created at a rate of 3 clients and 3 cases per second. The average response time for these requests was 298.73 ms. The average CPU of the system was 17.2% and the server did not need to scale and there were no spikes of CPU and/or response time, which means that the test was all supported by only one t2.micro machine that had an average CPU of 17.2% throughout the test. The linear regression of response times throughout the test had a slope of 0.0008. This shows the stability of this metric throughout the test.

**Registration of lawyers**

In this case, 10000 lawyers were created at a rate of 3 lawyers per second. The average response time for these requests was 447.25 ms. The average CPU of the system was 30.3% and the server did not need to scale and there were no spikes of CPU and/or response time, which means that the test was all supported by only one t2.micro machine that had an average CPU of 30.3% throughout the test. The linear regression of response times throughout the test had a slope of 0.0041. This shows the stability of this metric throughout the test.

**Address cases**

In this case, 50000 approaches were created at a rate of 6 approaches per second. The average response time for these requests was 172.73 ms. The average CPU of the system was 19.6% and the server did not need to scale and there were no spikes of CPU and/or response time, which means that the test was all supported by only one t2.micro machine that had an average CPU of 19.6% throughout the test. The linear regression of response times throughout the test had a slope of 0.00008. This shows the stability of this metric throughout the test.

**Performance Overview**

Throughout any of the performance tests, the system had no problem, in all 65,000 requests were made and the percentage of successful responses from that sample was 100%, the average percentage of the system from any of the three tests was perfectly bearable, the group of servers did not need to scale even though it was composed of a relatively weak machine, it is important to emphasize that if it needed to scale it would not be considered a problem since it is one of the features of Elastic Beanstalk, however the fact of the system not having climbed with this ability demonstrates its ability. Regarding the database, even after the creation of 70000 new objects (5000 clients, 5000 cases, 10,000 lawyers and 50,000 approaches), it was only 45 Mb and has a performance that has not been shown to have negative impacts on response times even though in a free tier. However, in a future and more demanding scenario, it is possible to scale vertically without causing any problem in the system if the database performance is not sufficient, the same is true for the type of individual machines in the server group.

**5.2. Real scenarios tests**

Scenario tests simulate the development of realistic events. A scenario turns out to be a story that defines a use case likely to happen when the product is in the hands of users. These are intended to demonstrate the functionality of the product in a realistic setting. It is important to mention that these scenario tests were carried out after the performance tests, that is, with a population database of 5000 clients, 5000 cases, 10,000 lawyers and 50,000 approaches. These tests also aim to test the integration of the front end with the backend having been done manually, through the browser, exactly the way a real user would interact with the platform. Several scenarios were tested and all performed as expected, with no errors.

[6, 7]

**6. Conclusions**

Having identified the existence of a problem in the legal market, we worked and carried out this dissertation in order to reach a solution. After analyzing this market and the respective identified problem, it was concluded that the problem exists and was correctly assessed, a fact that was even recognized by the Bar Association and by the lawyers addressed, and still observable in the results of the investigation. Then, in the technological framing phase, different types of databases were studied, comparing relational and non-relational databases...
and, finally, relating the technologies available in both with the implementation context, namely the server language, in the end, this was the decisive factor in choosing the database. Communication protocols of the client-server architecture and languages to implement the server were also studied, as well as different providers and services in the cloud. With this theme it was intended to achieve not only a hypothetical projection of a solution, but also a materialization of it, therefore, after the problem identification phase, requirements gathering and solution projection. This was followed by the implementation of a solution published on the Web, which obeys the RESTful architecture and which is hosted on Amazon Web Services (AWS), using the Elastic Beanstalk service, a PaaS service that provides all the infrastructure necessary for the platform to function properly, followed by a redundant solution among several availability zones, thus offering a 99.99 % SLA on servers. It should be noted that it was an objective, from the beginning, not to expose servers directly to the Internet, nor to allow SSH connections to them. This objective was achieved using the configuration of public subnets for the load balancer and private for the servers, it was also a requirement for the database to be accessible only through the servers, which is exactly what happens. It can be said that the infrastructure is solid, a fact that was even demonstrated in the tests that we will describe later. It is also added that attention was paid to the security and type of traffic, and its origin to which all infrastructure resources are exposed (for example, servers not only do not have a public IP address, they only accept HTTP traffic and that it comes from do load balancer). Regarding the servers, these are implemented in Node.js, the moongoose module was used to interface with the MongoDB database, which in turn is hosted in MongoDB Atlas, and that only accepts traffic (queries) from the servers. After the implementation phase, performance and scenario tests were carried out. Throughout any of the performance tests, the system had no problem, in all 65,000 requests were made and the percentage of successful responses from that sample was 100 %, the average percentage of the system from any of the three tests it was perfectly bearable, and the group of servers did not need to scale, even though it was composed of only one relatively weak machine. Regarding the scenario tests, all the tested scenarios had positive results, having documented two very comprehensive scenarios with regard to the platform’s functionalities. Regarding the database, even after the creation of 70000 new objects (5000 clients, 5000 cases, 10,000 lawyers and 50,000 approaches) it was only 45 Mb and has a performance that has not shown to have negative impacts on response times, even though num free tier. It is also important to mention that any of the components (servers and / or database) can be scaled vertically, without any problem in the availability of the platform, the horizontal elasticity of the servers is automatic, however, despite not affecting the availability of the platform, vertical scaling requires manual intervention. It should also be added that an automatic platform code update process without affecting the platform SLA was successfully developed, in other words the automation developed doubles the machine fleet, updates the new fleet and only after these new machines are updated does it register the new fleet on load balancer and deletes the machine group that runs the previous version, so while the application is being updated everything continues to work and when there is a fleet of machines ready with the new version, the servers are replaced, however, there may be a period of time, in the order of a few seconds, when the load balancer already has the new fleet of registered machines and still has the old ones registered. In short, part of this work was the identification of a problem (access to legal support), study of the market where the problem is inserted, projection of a solution, implementation of it, going through a test phase, the final product being a solution technology that aims to help the population.  

References


Statement: This thesis is not affiliated with the Bar Association nor with any other institution or organization.