UNIVERSIDADE D
COIMBRA

Final Architecture
PetPeople

Authors:
Edgar Duarte no. 2019216077
Sofia Neves, no. 2019220082
Tatiana Almeida, no. 2019219581

May 16^{th}, 2023

Contents

1 Project Context

PetPeople's main idea is to explore the world of pets by creating a platform, web and mobile, where people may post photos of their pet animals, as well as explore photos posted by other proud users. As with any other social network, you'll have the ability to interact with posts by giving them a 'like', replying, sharing, and more.

The app will also have a more informative side, where there are details about the veterinarians in each city, their contacts and specializations.

Furthermore, users may also navigate a marketplace where they may find multiple shops with the best deals for pet food and accessories. Pet shops will also have the ability to sell their items in PetPeople's platform. A small commission is then taken for each purchase made on the marketplace.

2 Design Patterns Decisions

In order to attend to the numerous quality attributes of the software, various design alternatives were considered for each scenario. In the following section, the thought process behind each of the scenarios will be presented, showing the chosen solution and all its alternatives.

2.1 Scenario 1 (QA: Correct emails)

The architecture has a Security Component whose job is to validate and sanitize incoming requests, as well as encrypt incoming sensitive data. For this scenario, after a Guest creates an account the Security component will be used to confirm that the email given is valid and that it doesn't already exist in the database. To perform the email validation, several methods were considered.

- **Simple Email Verification Link**: An email is sent to the provided email address which contains a unique token, that when clicked validates the account. Not only does this method increase overall security, by preventing fake/spam accounts, it is also a very simple method to implement. On the other hand, this methods relies on emails being delivered which might be hard to guarantee, hurting user experience. It also only verifies the legitimacy of the email after the it has already been sent, which causes unnecessary overhead and might block the system in case of a spam attack.
- **mailboxlayer**: An easy to use API for email validation that supports disposable email address detection. On the downside, it becomes costly for high volumes of email validation requests.
- **Express validator**: library for Node.js that provides validation functionalities. Some of its main advantages are a built-in email validation system with customizable rules and its integration with other Node.js libraries, especially with Express.js.

Seeing that the application is running on Express.js, the use of the Express validator seems the most logical. Additionally, an external email system is used to corroborate the legitimacy of the given email.

2.2 Scenario 3 (QA: Secure messaging)

Besides secure payments, in the case of a social networking platform, message privacy and confidentiality are properties that the system needs to guarantee. Not only should messages be encrypted when passing through the Internet, they should also not be readable when looked at in the database. For example, an admin should not be allowed to go to the database and look through the private messages of other users. This creates some constraints which the architecture has to attend.

The chosen solution was the implementation of a **Secure Communication Component (SCC)**, whose job is to guarantee an **end-to-end encryption** between the client and the server. End-to-end encryption is the standard for ensuring a secure and private communications. The data is encrypted on the message sender's device and is only decrypted on the message receiver's device. On top of using this method, the SCC will make use of SSL/TLS encryption protocol to safeguard all transmissions.

An **alternative** to the presented would be to use **Message-level encryption** . Used by VISA, it was made to store information and communicate with other parties while preventing undesired connections from understanding the stored information and the communication's content. In short, each message is individually encrypted using a several encryption certificates, making sure that only the receiver can decrypt it using a private key. Seeing that it was made having sensitive transaction data in mind (Personal Identification Information, for example) and has a considerable overhead, it seemed inadequate and overkill for a private messaging system.

2.3 Scenario 4 (QA: Easy maintenance)

A maintainable application is crucial to ensure a good software's quality, preventing undesired software instability and stoppages. In order to handle this quality attribute, a **Version Control Manager** (VCM) will be implemented.

The VCM is composed of a component and a container. Firstly, there will exist a **Version Control System (Git)** which will be responsible for storing different versions of the software. It will also be a vital tool in the organization of the project due to the ability of creating branches for the different production stages (testing, development, deployment, etc.). There are some alternatives to Git but none of them had any advantages over Git for our quality attributes, so the industry standard was chosen.

Finally, the VCM will contain a **Deployment Control Script (DCS)** that is responsible for choosing the next machine at which an update will occur by using a **rolling deployment**. In the architecture presented there are 2 virtual machines. The DCS chooses one of the machines to update while leaving the

other untouched and open to traffic. After the first has stopped updating, the DCS will now switch to the not updated virtual machine and starts updating it, leaving the updated virtual machine open to traffic. After updating both machines, they will both be open to traffic. This procedure leads to, ideally, no downtime. For our software, and seeing that the system only uses 2 virtual machines, this approach should suffice.

An **alternative** to this approach that was considered was the **Blue-Green Deployment**. In this approach, there is an environment in isolation that is firstly updated. The traffic is then redirected to that environment while the main system gets updated. When the main system is updated, the traffic gets redirected there. This approach would require an additional container, the isolated environment container. The downsides to this approach is that it is generally slower than the rolling deployment and is more susceptible to downtime (seeing that we have some quality attributes that require high availability, this is critical, see QA-10).

2.4 Scenario 10 (QA: Up time)

The availability of a system is the percentage of time the system is operational. In the case of PetPeople, an availability of 99% per year, is desired.

Load Balancer

The **AWS Load Balancer** seemed to be the best option, seeing that it is a service that distributes traffic to multiple instances. It provides health checks and auto scaling features to help monitor and adjust to which machines to send traffic. It is also integrated in the AWS global network, which has a 99.99% to 99.999% availability per year, a value superior to what we are aiming.

The main alternative we looked at was **NGINX**. Ngingx is a very popular reverse proxy that can be used as a load balancer. Just like the AWS Load Balancer, it was designed to handle large amounts of traffic, delivering web content to users. It is often used with other tools such as Docker and Kubernetes. Some advantages brought by the use of this technology is the lightweight footprint and its ability to easily scale. Seeing that PetPeople will be deployed on AWS, using a AWS load balancer is much simpler and cost effective than configuring Nginx. On top of that, the cost of an AWS load balancer, although higher than the Nginx's cost, comes with a variety of features not supported by Nginx such as automatic scaling. In short, both technologies impact on availability is similar, but due to the application being deployed on AWS, it makes a lot more sense to use a AWS load balancer, seeing that it would require some additional architecture components and configurations to use Nginx, which could delay the application's release date **impacting BC2-Time to market**, which imposes a deadline of 1 year.

Availability Model

The rest of the components that affect the availability of the system are the 2 Virtual Machines, the Version Control System (Git repository) and the MongoDB Replicas. If one of these components or the load balancer goes down, the system will become unavailable. The desired availability of the application is 99%. As such, we need to verify the attainability of this goal, using an availability model:

The advertised uptime of the each of the services, according to documentation, is as follows:

- **AWS:** 99.99% to 99.999%
- **Git:** 99.9%
- **MongoDB Atlas:** 99.995%

Due to AWS having a upper and lower bound of possible values, we will calculate the best and worst possible availability values for our system. There is a probability that the hardware fails and probability software fails. In case of the hardware, the probability of it occurring is equal to the probability that AWS goes down. For the case of the software the probability is calculated as follows:

$P((\text{VM1 Down} \cap \text{VM2 Down}) \cup \text{Load Balancer Down} \cup \text{Git Down} \cup \text{MongoDB Down})$

The results of the calculations are as follows:

	Worse expected downtime	**Best expected downtime**
Hardware	0.01%	0.001%
Software	0.1342985%	0.10500095%
Availability	99.8557%	99.89399%

As can be seen, even in the worst case the theoretical availability suffices the 99% requirement, almost reaching the 99.9% step. As such, in what concerns availability, we are happy with the architectural decisions chosen.

2.5 Scenario 12 (QA: Video loading time)

It is very important for our application that videos load fast so that we retain user attention. We desire to load videos to users in less than 2.5 seconds. Seeing that there is virtually no way to control a user's bandwidth, a Bandwidth Controller will have the responsibility of adapting a video's quality. If it detects that a user has a low bandwidth, it will lower the video quality in order to ensure a quicker load time. The videos will also be compressed, which will hurt their quality but in return should lead to a quicker load time.

Another component that could be added to the architecture to try and help this quality attribute could be **caching videos** in memory could help improve

the response time for frequently viewed videos. Seeing that it is a social media platform where a user is continuously scrolling through posts, we did not find this to be a justifiable addition, as it does add some overhead for users.

2.6 Scenario 13 (QA: Transaction time)

Seeing that there are transactions, one of the main metrics that we need to keep in mind is transaction time. PetPeople wants to guarantee the user that the transaction is completed in, at most, 30 seconds. To achieve this, the Payments Facade will abort any transaction that takes longer than 30 seconds to execute and will inform the user, through the Communication Controller, if the transaction did or not succeed. The choice of a Payments Facade built in Express.js was derived from the fact that Node.js is a fast and efficient server-side environment, which allows it to quickly give feedback to the user.

2.7 Scenario 14 (QA: Backup of data)

In the event of a database corruption, it is essential to ensure that there is a backup available in order to protect against the loss of any stored information. To address this, a **Replica Controller** is used to guarantee that the user can always access data, even if one of the databases becomes unavailable or corrupted. It is to note that we have two different databases, each with one replica, meaning we have four databases in total. The two different databases will store different information. The first will store media content, such as accounts, posts, videos, comments and news, while the second will store more sensitive data like transactions and private messages.

MongoDB replica set are servers that maintain identical copies of the data in the databases, providing redundancy and high availability (which is beneficial for **QA-10: Up time**). The replica controller coordinates the behaviour of each replica set and is responsible for electing the current primary database in case of failure of the previous one. It is also responsible for ensuring that the data is synchronized across all the replicas, guaranteeing that any read operation will have the same results, independent of which replica we are currently reading from. Finally, it monitors the health of the replicas and automatically starts a failover if the primary replica fails. Due to all of these features, it seemed like an adequate architecture to use in order to have back up of data and have high availability.

Another **alternative** that seemed very appealing was the use of a **cloud backup provider** such as AWS. Seeing that the application will be deployed in the AWS environment, it would make sense to use Amazon DocumentDB for database replication. The main fears an risks we saw with this approach was that AWS DocumentDB does not support all features in the new versions of MongoDB, it allows for less replicas, only allowing for one replication zone and does not allow sharding. Due to these reasons, it seemed preferable to have a slightly higher costs in order to guarantee that the scalability of the databases is not impacted (**QA:17** and **QA:22**).

2.8 Scenario 15 (QA: Censoring of adult content)

In a social media where the presence of users of age 13+, it is crucial to have a filtering system for NSFW content. PetPeople's policy is to have zero tolerance for this type of content, needing it to be identified and quickly removed from the platform. As such, the **Censor Controller** is responsible for detecting and banning any inadequate technology. It will do so using **AWS Rekognition**, a video and image analysis service provided by AWS that automatically detects inappropriate content.

Validation

The main issue with the use of this service might be its price, seeing that **BC-7** says that the system should have a return on investment of 200% in 3 years. Next we shall calculate the costs of using AWS Rekognition in order to see if its use is feasible.

The price of AWS Rekognition for inappropriate content detection is as follows:

- First million images: $0.001 per image
- All following images: $0.00075 per image
- $0.10 for each minute of video

Assuming that **BC-8** is achieved, then the application will have a user base of 1 million users. Looking at other social media platform statistics, the average number of posts per user, per day is around 0.19, from which 31% is video content (which averages 30 seconds each) and the rest is photo content. Assuming that the average video time is around 30 seconds, the costs of using Rekognition is as follows:

1. Content per million users = 1,000,000 * 0.19 = 190,000
2. Amount images = 190,000 * 0.69 = 131,100
3. Images cost before million day = 131,100 * 0.001 = 131.1$
4. Images cost after million day = 131,100 * 0.00075 = 98.325$
5. Amount video = 190,000 * 0.31 = 58,900
6. Video cost day = 58,900 * 0.1 * 0.5 = 2,945$

After around seven days, we hit the one million threshold of images. As such the cost per year (365 days) is as follows:

1. Cost images per year $= 7 * 131.1 + 358 * 98.325 = 36,118.05\$$
2. Cost videos per year $= 365 * 2,945 = 1,074,925\$$

3. Total Rekognition cost year $= 1,074,925 + 36,118.05 = 1,111,043.05\$$

As was calculated, a whooping 2 million dollars per year would be spent using Rekognition. To see if this is acceptable, we need to calculate the expected income of the application. Assuming 1 million users and using publicly available financial data:

- The **revenue per thousand ad impression**s is between 8$ and 20$. Assuming the worse case scenario, we shall assume that 1000 ad impressions equal to 8$. Seeing that this is a social media platform, an user will use the application, most likely, a few times per week (for example once a day), meaning 365 visits per year. Lets assume that a quarter of the user base spends at least 5 minutes in PetPeople a day. As the app doesn't want to be too intrusive to users, the amount of ads will be kept to a minimum, but it is safe to assume that in this time the user will be subject to at least 10 ad impressions. With this, we can calculate the total expected revenue from ads in a year:

  1. Total number of ads year $= 250,000 * 365 * 5 = 912,500,000\$$
  2. Revenue from ads $= (912,500,000/1000) * 8 = 7,300,000\$$

  So, in the worse case scenario, ads will generate an approximate revenue of **7.3 million dollars per year**.

- PetPeople's **selling fee** will be around 20%. The average price of pet items heavily varies from pet to pet, but a fair estimate is around 10$ per toy. With this in mind, looking at already existing and functioning marketplaces, a rough average of the percentage of users that make a purchase in a day is 0.1%. As such, we can now estimate how much money we will receive through selling fees:

  1. Income from items daily $= (1,000,000 * 0.001) * 10 * 0.2 = 2000\$$
  2. Total income yearly $= 2000 * 365 = 730,000\$$

  As can be seen, from selling pet items the application will **profit around 730,000 dollars**.

- The application will also make revenue from **selling data**. The price at which the user data could be sold at is very uncertain. Our estimates point that with good data collection practices that allow buyers to get some deep insight about user preferences, the data will be worth at least 1$ per user, meaning an extra **1 million dollars in revenue** per year.

In total, the **expected revenue** per year for 1 million users, is 7,300,000 + 730,000 + 1,000,000 = 9,030,000$. It is to note that this value is the worse scenario. Taking this into account, we can see that the use of Rekognition for all data would be too expensive. It would be a whooping 12.3% of the total revenue. The **alternative** would be for PetPeople to create their own censoring

system. Now this is also be very expensive and will consume a lot of resources and time. Even so, in the long run it should prove to be cheaper. Unfortunately the initial team does not possess the abilities to implement such a big feature. As such, Rekognition will be used in an initial phase, until the PetPeople's developer team can successfully create their own alternative.

2.9 Scenario 16 (QA: Update rollback)

The **Version Control System** Container is responsible for guaranteeing that all versions of the software are correctly stored on Git. Git also ensures that rollbacks can be made in order to restore a previous safe state of the software if need be.

One **alternative** to Git could have been Mercurial which also has a distributed version control system, good performance and, branching and merging capabilities. However, Git outperforms in terms of speed and performance, it has a larger community base and has more tools.

As for the update rollback, one of the other options was snapshots. It is possible to do snapshots of the entire directory structure and files associated with the software which makes rollbacks possible. As opposed to Git rollbacks, snapshots would encompass all of the files and not only the different files creating in this way an unnecessary overhead which will occupy much more space than Git does. Therefore, Git rollbacks were the preferred choice.

2.10 Scenario 20 (QA: Number of transactions)

The **Payments Facade** Component aims to handle 1000 transactions per second. The choice befell between a Facade and an Adapter. Since the purpose was to only have a simplified interface to a complex subsystem instead of an adaptation to a library, a facade pattern was chosen.

Validation

In order to ensure that 1000 transaction per minute can be made with a Payment Request Orchestrator. This orchestration can be made through:

- AWS Step Functions: A fully-managed service that allows you to coordinate distributed applications and microservices using visual workflows. Complex workflows can be created that trigger the execution of multiple AWS services.
- AWS CloudFormation: A service that allows you to automate the deployment and management of AWS resources and applications using code.
- AWS Elastic Beanstalk: A service that allows you to deploy and manage web applications in the AWS Cloud without worrying about the underlying infrastructure. You can upload the application code and the service automatically handles the deployment, load balancing and scaling of the application.

The final decision was to use AWS Step Functions.

2.11 Scenario 22 (QA: Sustain an annual content growth of at least 200% in content)

It it expected that in a social media application there is constant growth in data. AS such, the databases must be ready to accommodate large amounts of data, without compromising availability and performance. For this, a **Replica Controller** was implemented. The reasoning behind the choice of this component was already explained in scenario 14. The fact that MongoDB Atlas allows for up to 50 replicas, in multiple replication zones, allows it to scale much more efficiently.

Validation

To validate this quality attribute, we resorted to MongoDB's documentation. As will be calculated in section 5, the amount of data produced in the first year is estimated at 643 terabytes. With this in mind, the use of sharding is essential in order to horizontally scale, making it possible to handle large amounts of data. According to some documentation, each shard can hold up to 64 terabytes of information. We shall use the Amazon Elastic Block Store (EBS) to store the databases,whose maximum storage suffices. The recommended maximum size for a shard is 50 terabytes. For our MongoDB cluster, we only need to create new shards when required, having virtually no limit. As such, scalability is assured in our architecture.

2.12 Scenario 23 (QA: Number of concurrent requests)

To ensure that our system can handle 800 requests per second, we first began by analyzing the technology to be used. The traceability matrix provides a detailed explanation of the several options we considered and the reason for our choice. Given that Express is one of the best technologies when it comes to performance, we believe it is an excellent starting point to guarantee this quality attribute.

Next, we conducted an analysis of the available load balancer options. This analysis mirrored our findings from Scenario 10. After careful consideration, we determined that the AWS Load Balancer was the best choice to maintain this quality attribute.

We carefully considered multiple hosting providers and concluded that Amazon Web Services (AWS) was the best choice to meet the needs of our application. AWS stands out from its competitors, Google Cloud Platform (GCP), Microsoft Azure, and DigitalOcean, in that it offers a comprehensive suite of services for optimizing application performance and ensuring user availability. AWS's established history of delivering high-performance and reliability makes it more reliable than similar services provided by GCP and Azure.

Another advantage of AWS is its strong focus on security. It provides a variety of tools and features that allow developers to construct secure applications, such as AWS Identity and Access Management (IAM), Amazon Virtual Private Cloud (VPC), and AWS Security Hub. While other providers offer similar security services, we feel that AWS has a more comprehensive approach to data protection.

In addition, AWS stands out when it comes to scalability. Using AWS, we can access an array of services which enable us to adjust our infrastructure up or down in line with fluctuations in traffic or demand. It offers an extensive selection of instance types from specialized instances for compute, memory, and storage capacities that make it easier to customize our system according to the needs of the project..

Finally, AWS provides a high level of availability and uptime. With its multiple regions and availability zones, our application can remain operational in the event of an outage in one zone. Additionally, AWS offers comprehensive and user-friendly tools and services through Amazon CloudWatch to help us monitor and manage the performance of our application. Our decision to go with AWS Cloud Services was further driven by the developers' familiarity with this cloud computing platform.

Validation

To demonstrate compliance with this quality attribute, we decided to gain a comprehensive understanding of the technology we were going to use to ensure it could handle 800 requests per second, as required by the quality attribute. We opted to utilize an Amazon EC2 server and carefully chose c6g instances to guarantee that they possess enough power for CPU-intensive workloads. The compute optimized instances provide an advantageous combination of compute strength and memory, making them perfect for our purpose.

Additionally, we have chosen to utilize an Amazon Elastic Load Balancer from AWS to assist with load distribution. This load balancer distributes incoming traffic evenly across multiple instances, allowing each instance to handle a portion of the requests and prevent any single instance from becoming overloaded by an abundance of requests. This helps improve overall performance and availability.

To guarantee the most efficient data storage, we opted to link an Amazon EC2 server with Amazon Elastic Block Store (EBS). Specifically, we selected IO2 (Provisioned IOPS SSD), which is tailored towards I/O-intensive workloads like databases while providing consistent low-latency performance. Through this technical solution and the choices made, we can confidently ensure that the 800 request per second benchmark is met and validated.

3 Costs

An important step for any business is to estimate the costs of operation. As such, in the following section, an estimate for the costs of the server that was chosen. As such, an estimate of the amount of data that will be employed by the server needs to be calculated:

- average image size - 160 KB - 0.0001525879 GB
- average video size - 30 MB - 0.029296875 GB
- average test based content - 3 kb - 0.000002861 GB
- number of videos uploaded per day - 58900
- number of photos uploaded per day- 131000
- text based content (message/comment) uploaded per day- 5000000

Storage : - 58900*0.029296875 + 131000 * 0.0001525879 + 5000000 * 0.000002861 = 1759.87 GB per day
1759.87 * 365 = 642352.55 GB (storage needed per year)

$$\approx 643TB$$

Data transfer:
Data transfer per month = (average daily visitors) x (average page views per visitor) x (average page size) x (30)

- average daily visitors: 250000
- average page views per visitor: 10 (assuming 5 and 10 minutes in the platform)
- average page size: 4 MB

2500000 * 10 * 4 * 30= 300000 GB per month
Instances:
Since we want to handle 800 requests per second, we need to use larger instances to accommodate the workload. We decided for c6g.4xlarge instances, so the total cost for 4 instances per hour is:

- 4 x 0.676$ = $2.704 per hour

The cost for running 4 instances for an entire year (assuming 365 days, 24 hours per day) is:

- $2.704 x 24 x 365 = $23,706.24

EBS Storage Cost:

Assuming that we need 643 TB of storage, we can use io2 volumes which cost $0.125 per GB per month. Since we need to store the data for an entire year, the total cost for 643 TB storage is:

- 643 x 1024 x 0.125 x 12 = $993,280.00

Data Transfer Cost:

Assuming 300,000 GB of data transfer per month, the cost for data transfer is as follows:

- Up to 10 TB: $0.085 per GB
- Next 40 TB: $0.076 per GB
- Next 100 TB: $0.047 per GB
- Next 350 TB: $0.039 per GB
- Over 500 TB: $0.030 per GB

Since we need to transfer 300,000 GB per month, the cost for data transfer for an entire year is:

- Up to 10 TB: 10 x 1024 x $0.085 x 12 = $103,219.20
- Next 40 TB: 40 x 1024 x $0.076 x 12 = $385,024.80
- Next 100 TB: 100 x 1024 x $0.047 x 12 = $610,355.20
- Next 150 TB: 150 x 1024 x $0.039 x 12 = $746,496.00
- Total cost for 300,000 GB/month data transfer: $1,845,095.20

Elastic Load Balancer Cost:

Assuming we use an Application Load Balancer, the cost per hour is $0.008 per hour plus $0.008 per LCU-hour. Assuming we need 100 LCUs to handle 800 requests per second, the cost per hour for the load balancer is:

- $0.008 + 100 x $0.008 = $0.816 per hour

The cost for running the load balancer for an entire year is:

- $0.816 x 24 x 365 = $7,158.72

Total Cost:

Adding up the costs from each component, the total cost for hosting the application for one year is:

- EC2 instance cost: $23,706.24
- EBS storage cost: $993,280.00
- Data transfer cost: $1,845,095.20
- Elastic Load Balancer cost: $7,158.72
- **Total: $2,869,240.16**

4 Security Checklist

In order to validate the security quality attributes, a checklist of attributes needed to verify was created:

	Security Checklist	Validation
User Authentication	1. Verify that users are authenticated before accessing the messaging system	✓
	2. Verify that strong passwords are enforced for user accounts	✓
	3. Verify that users are logged out after a period of inactivity	✓
Encryption	4. Verify that messages are encrypted using a secure encryption algorithm	✓
	5. Verify that encrypted messages can only be decrypted by authorized users	✓
	6. Verify that encryption keys are securely stored	✓
	7. Verify that encryption keys are changed regularly	✓
Message Integrity	8. Verify that message checksums are used to ensure message integrity	✓
	9. Verify that the messaging system can detect and report any tampering attempts	✓
Access Control	10. Verify that users can only access messages that they are authorized to see	✓
Error Handling	11. Verify that error messages do not leak sensitive information	✓
	12. Verify that error messages are logged for debugging purposes	
	13. Verify that error messages do not cause the system to crash or become unstable	✓
Audit Logging	14. Verify that all user actions are logged for auditing purposes	
	15. Verify that audit logs are kept secure and can only be accessed by authorized users	
	16. Verify that audit logs cannot be tampered with or deleted by unauthorized users	

For the number 1, 8 and 10 in the checklist, the Message Controller is the component who will verify if the users are indeed authenticated, the message checksums and that the users can only access messages they are authorized to see. By trying to access any endpoint related to the messages the user will be redirected to the log in controller if they are not authenticated. The message checksums will be calculated at the client's side and then sent to the server who will then also calculate the checksum and verify if both are equal.

The number 2. will be covered by the Register Controller as well as the Account Recovery Controller who will make sure that a strong password is used for each user because upon registration it will only allow strong passwords that meet a certain criteria.

Number 3. will be ensured by the Sign In Controller who will make sure that users are logged out after a period of inactivity. This component is also responsible for the authentication of each user and the attribution of a sign-in token that will be used by the user for subsequent requests made to the server. This token will expire after a certain period of time and its the sign in controller's responsibility to then check wether the user is or isn't logged in.

As for the numbers 4, 5, 6, 7 and 9 are going to be verified by the Secure Communication Component inside the Message Controller. This component will make sure that the messages are encrypted and decrypted by the correct user, it is also going to verify the correct storage of the encryption keys and that they are changed regularly so as to better protect users from eavesdroppers. It will also detect and report any attempt of a third party to tamper with the messages of the user.

Numbers 11 and 13 will be verified by all Petpeople's app components who will make sure that no error message leaks any sensitive information and that

neither any message causes the system to crash or become unstable. When it comes to the numbers 12, 14, 15 and 16 it is visible from the table that no component in the architecture is ensuring that logs are made.

5 Conclusion

With this document, we validated the presented architecture according to the various quality attributes that were imposed. Not only did we justify our decision, but also looked at alternative implementations, their benefits and disadvantages. In junction with the data in the traceability matrix, we can confidently say that the found architecture meets the requirements of the client.