Clinic Management System

Research Document

INSTITUTE *of* TECHNOLOGY CARLOW

18th April 2018
BSc (Hons) Software Development

Name: Ryan Donoghue
Year: 4th year
Student ID: C00194829
Supervisor: Paul Barry

Table of Contents

Erlang

Overview

Erlang is an open source programming language designed for developing robust systems of programs that can be distributed among different computers in a network.The language was created by Ericsson Computer Sciences Lab to allow them to build software for its telecommunication products, and named after the Danish mathematician Agner Krarup Erlang. `Similarly to Java, Erlang uses a virtual machine that supports multithreading, however Java developers mostly use it for web applications, Erland is typically used to create extremely robust servers and embedded systems.

Erlang is described as a functional programming language, meaning that it emphasizes the evaluation of expressions rather than the execution of commands. The expressions use functions to derive basic values. Other well known programming models are procedural and object-oriented .)

Program threads can either be explicitly specified or invisible to the program. This means an application can easily be distributed and run on any point within a network
Erlang provides dynamic data types, allowing programmers to develop system components (such as message dispatchers) that do not care what type of data they are handling and others that strongly enforce data type restrictions or that decide how to act based on the type of data they receive.

Dynamic data types are possible with Erlang, which allows for variety in data restrictions. Components may be built to not care what kind of data they are handling, but others can be built to impose strong restrictions and even decide what action to take based on the data they receive.

Built in to Erlang are multiple design patterns/templates for client-side design (such as Model-View-Controller), thread supervision and event distribution as well as useful data conventions such as pattern matching. Pattern matching allows for extremely compact and clear programs. A pattern matches if it has the same "shape" as the term being matched, for example:

```
A = 1.
1 = 1.
{ok, A} = {ok, 42}.
[H|T] = [1, 2, 3].
```

[1]

Garbage collection is also supported by Erlang (similar to Java) and means a programmer need not be concerned with returning allocated memory space, as Erlang handles it silently.
Erlang's bytecode is identical on all platforms, and as such a network of Erlang nodes do not necessarily need to all be running the same platform. An additional benefit of Erlang is that it's relatively easy to learn relative to other programming languages such as C(#,++) and Java.**[2]**

Advantages of Erlang

- Lightweight threads. Erlang does an excellent job of handling threads to implement concurrency and distributed systems. It is easy to create parallel threads and allow them to communicate between each other.

- Fault tolerance/failure detection. Erlang VM's strategy is to let the portions of code that fail, fail but keep the remainder of the application continue to run. The VM gives you extended functionality such as(but not limited to):
    - Informing you when and why a process died
    - Allows you to force dependant processes to die together if one of them generates a fault/error
    - A logger that can automatically log all uncaught exceptions
    - Node monitoring can used to detect down/disconnected nodes
    - Failed processes or groups of failed processes can be restarted

- Mostly functional programming with matching constructs and dynamic typing. As mentioned previously, components can be designed to be data type fluid, allowing strong or loose restrictions depending on desired functionality.

- Reliability-oriented standard library. A good example of this is the AXD301 flagship project built by Ericsson, that contains over two million lines of Erlang

code. The project achieved a Nine Nine reliability (99.9999999%). To put this into context, this means in one year, the project was offline for 3.1536s in one year.

- Data structures are immutable, this means they are thread safe, can easily be shared (ie. by references) and makes debugging easier.

Disadvantages of Erlang

- Can be difficult to learn if prior knowledge of C-based language is held, as the conventions can be difficult to “unlearn”.
- Mentioned as an advantage, immutable data structures also have a disadvantage; objects are created on modification. There are scenarios where this property can potentially be costly if chaining/multi stepping the creation of a final object or modifying a large object.

Table 1.1 Comparison of technologies used in two real-life web servers

Technical requirement	Server A	Server B
HTTP server	Nginx and Phusion Passenger	Erlang
Request processing	Ruby on Rails	Erlang
Long-running requests	Go	Erlang
Server-wide state	Redis	Erlang
Persistable data	Redis and MongoDB	Erlang
Background jobs	Cron, Bash scripts, and Ruby	Erlang
Service crash recovery	Upstart	Erlang

[3]

Elixir

Overview

"*Elixir is a dynamic, functional language designed for building scalable and maintainable applications.*
Elixir leverages the Erlang VM, known for running low-latency, distributed and fault-tolerant systems, while also being successfully used in web development and the embedded software domain."**[4]**

As mentioned above, Elixir runs on the battle tested Erlang VM, which is a great choice for its low-latency, distribution and fault tolerance. It also has the added benefit of being versatile enough to greatly simplify a project stack.**[5]**

Erlang and Elixir packages can easily be found via the HexDocs package manager, which contains documentation for all packages. There are many packages available through this site, most of them of high quality and maturity, with good documentation available for each. Again, the reliability of Erlang transfers to Elixir. In contrast to other popular programming languages, if a process fails in Erlang, it doesn't grind the rest of the application to a halt.

Elixir builds on this Erlang base in a multitude of ways. The first, is Erlang's relationship with strings. In Erlang, strings are represented as a list of characters, which can be inefficient. A more common and efficient way of representing strings is using binaries, however the syntax for this in Erlang is not as user friendly. Elixir uses its String type to create strings backed in Erlang binary format, but with easier use.

Code Library

Elixir's standard library is also superior to Erlang, for example, take the List data structure in Erlang. It doesn't have a lot of functions and as such can make comprehensions more difficult. Take the following code as an example, which iterates through a list and outputs the first value that is greater than 15:

```
case lists:dropwhile(fun(X) -> X =< 10 end, [1, 3, 8, 15, 7, 100]) of
  [] -> nil;
  [Y | _] -> Y
End
```

The same solution in Elixir can use the Enum module, which vastly simplifies the process:

```
Enum.find([1, 3, 8, 15, 7, 100], fn(x) -> x > 10 end)
```

The Elixir solution is notably easier to read and implement, as a result of using the Elixir.Enum module over the Erlang.stdlib.

The situation is similar with regard to Structs, with Erlang Structs in an “ad-hoc” manner with verbose syntax. A Person record would be implemented in Erlang using the following code:

```
-module(using_record).
-record(person, {fname, lname, phone, address}).

full_name(Person) ->
  Person#person.fname ++ " " ++ Person#person.lname.
```

The same code using Elixir:

```
defmodule Person do
  defstruct fname: nil, lname: nil, phone: nil, address: nil
end

defmodule UsingStruct do
  def full_name(person) do
```

```
    person.fname <> " " <> person.lname
  end
end
```

The code is much more readable and contains more better syntax.

Building on Erlang

Immutability is better handled with Elixir also. When data is changed in Erlang, a new object is created and returned. This means code can become very fragile, and careful consideration must be taken when naming variables during logical steps. This issue was overcome in Elixir by allowing variable rebinding, mutability can be achieved.

The pipe operator is also added in Elixir, which is not available in Erlang. This operator allows the result of one function call to be passed as an argument into the next function. This means code can be split into multiple calls, resulting in more readable code without the need for intermediate variables. This readability is also improved by Elixir's changed syntax when compared to Erlang.

Phoenix

"*Phoenix brings back the simplicity and joy in writing modern web applications by mixing tried and true technologies with a fresh breeze of functional ideas.Create rich, interactive experiences across browsers, native mobile apps, and embedded devices with our real-time streaming technology called Channels. Phoenix leverages the Erlang VM ability to handle millions of connections alongside Elixir's beautiful syntax and productive tooling for building fault-tolerant systems.*"**[6]**

Phoenix is a web framework that makes building API's and web applications easy. It is most similar to Python's "Django" framework or Ruby's Rails framework. Since it's built with Elixir and runs on Erlang's VM it's incredibly fast and has excellent support for handling a large number of simultaneous users.

A test was carried out by LittleLines, by which the Phoenix Framework was tested against Ruby's Rails framework, which found that:

"*Phoenix showed 10.63x more throughput, with a much more consistent standard deviation of latency*".**[7]**

To further emphasise this performance point, a blog by BigNerdRanch mentions:

"*What's more, Phoenix apps without caching drastically outperform Rails apps with caching. This is important because caching is notorious for being a source of complexity and bugs, and because caching can't be used for moment-by-moment, personalized content like that offered by Bleacher Report, which shows users news and tweets about the teams they're interested in and handles 250k concurrent users. According to a talk their developers gave, they went from over 100 AWS servers to 5, with CPU usage rarely going above 10%, and saw a 10x performance improvement over their Rails app.*" **[8]**

It is worth noting, Phoenix uses Model-View-Controller (MVC) in its generation of application components, along with Ecto for database objects.

Ecto

*“Elixir is a modern, dynamic, functional programming language used to build highly distributed and fault-tolerant applications. Ecto is its main library for working with databases, providing us with tools to interact with databases under a common API, version the database alongside our application, and handle data processing within our application.” **[9]***

Ecto is made up of 4 key components:

- **Ecto.Repo**
  - Defines repositories that are wrappers around a data store Defines repositories that are used to wrap around a data store such as PostgreSQL. This allows us to insert,create,delete and query a repository. Adapters and credentials are mandatory to communicate with databases.
- **Ecto.Schema**
  - Schemas are used to map any data source into an Elixir struct. Ecto uses schemas to map data sources to Elixir structs (similar to defining a SQL database layout)
- **Ecto.Changeset**
  - Changesets serve as a way to filter and validate parameters and validate changes before they are added to a database
- **Ecto.Query**
  - Ecto queries provides an SQL query (similar to query-DSL) for retrieving and inserting data to and from a repository. These queries have a benefit of being more secure than standard SQL queries, offering protection from common problems such as SQL injection natively

Model View Controller

Model view controller (MVC) is a popular design pattern. In most applications, 3 distinct parts can typically be found:

- Data (the model)
- An interface used to view and modify the data (the view)
- Operations can be carried out on the data (the controller)

The model is a representation of the structure of the data and nothing more. It has no dependency on the view or the controller.

The view displays data from the model and handles the sending of user actions (eg. button clicks, form submission) to the controller.

The controller can in some cases perform the same operations as the view, such as interpreting user actions. It highly depends on the model and the view.

The main advantage of MVC is the complexity (or lack thereof). Unnecessarily complex code is prone to bugs and can be expensive to maintain. This high complexity is usually achieved by placing too many dependencies in the code. A solution to this is to remove any unnecessary dependencies to create code with few bugs that is easier to maintain due to being reusable without modification. This is the primary advantage of MVC: it makes model and view classes reusable without modification.

The MVC design pattern places a middleman between the view and model, the controller. This minimises dependencies, and makes the model and view reusable, making new functionality and code maintenance simpler.

GDPR Legislation

Overview

One of the most prolific pieces of legislation is currently in effect, with companies scrambling to comply with it before the enforcement deadline in May 2018; the General Data Protection Regulation (GDPR). To quote the objectives of GDPR:

"1. This Regulation lays down rules relating to the protection of natural persons with regard to the processing of personal data and rules relating to the free movement of personal data.

2. This Regulation protects fundamental rights and freedoms of natural persons and in particular their right to the protection of personal data.

3. The free movement of personal data within the Union shall be neither restricted nor prohibited for reasons connected with the protection of natural persons with regard to the processing of personal data." ***[10]***

This regulation aims to protect data subjects data, their right to the protection of this data, and rules relating to the free movement of personal data. By law, any company who collects or processes EU citizen data, regardless of whether or not that company resides within the EU, must comply with the regulations set out within.

The regulation is currently active, however will only be enforced after May 2018 whereby sanctions exists. Any company discovered to not be in compliance with the regulations is subject to fines of up to 4% of their revenue (not profit) or up to €20 million.

As the clinic management system will store patient data, the regulations of GDPR apply to any data collection or processing carried out by the system. Some of the relevant articles from the legislation are outlined below

Data protection by design and by default

Taken directly from the GDPR:

"*1. Taking into account the state of the art, the cost of implementation and the nature, scope, context and purposes of processing as well as the risks of varying likelihood and severity for rights and freedoms of natural persons posed by the processing, the controller shall, both at the time of the determination of the means for processing and at the time of the processing itself, implement appropriate technical and organisational measures, such as pseudonymisation, which are designed to implement data-protection principles, such as data minimisation, in an effective manner and to integrate the necessary safeguards into the processing in order to meet the requirements of this Regulation and protect the rights of data subjects.*

*2. The controller shall implement appropriate technical and organisational measures for ensuring that, by default, only personal data which are necessary for each specific purpose of the processing are processed. That obligation applies to the amount of personal data collected, the extent of their processing, the period of their storage and their accessibility. In particular, such measures shall ensure that by default personal data are not made accessible without the individual's intervention to an indefinite number of natural persons."****[11]***

Paraphrasing Article 2 Paragraph 1, appropriate technical and organisational measures must be set in place to implement data protection principles to protect the rights of the data subject. This statement is intentionally broad, as the circumstances can vary between data controllers. However in this case, there is a number of safeguards that must be put in place. Because Personally Identifiable Information (PII) is required for the clinic to function as such, pseudonymisation is not appropriate. This can be mitigated by encrypting the data to appropriate medical standards. To quote

"*Encryption In Healthcare*" by Apricorn, who sell encrypted external storage products,

*"Of all the encryption methods, AES (Advanced Encryption Standard) receives the lion's share of attention, and for good reason. The NSA uses AES to encrypt data, which ought to be proof enough of its security. AES can use 128-bit, 192-bit, or 256-bit keys and thus far has been extremely resistant to attempts at exploiting potential weaknesses. Several cryptographers have tried to break AES, but none have succeeded."****[12]***

It appears AES encryption is one of the best solutions for encryption and will require further research in order to potentially incorporate it into the system.

Paragraph 2 is concerned with what data is collected, and their purposes. Under GDPR legislation, only data directly necessary for each specific purpose should be collected, and should only be maintained for as long is necessary for processing. As the clinic will require a variety of data regarding the patient, careful attention should be given to what data is collected. A basic rule of thumb, is to ask the question "Is this data necessary for the well-being of the patient". If the answer is yes to this question, store the detail in question, if no do not, so as not to infringe on the GDPR legislation.

Processing of special categories of personal data

Article 9 refers to special categories of data, grounds for processing and legislation relating to this processing.

"Processing of personal data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs, or trade union membership, and the processing of genetic data, biometric data for the purpose of uniquely identifying a natural person, data concerning

health or data concerning a natural person's sex life or sexual orientation shall be prohibited." ***[13]***

"*Processing is necessary for the purposes of preventive or occupational medicine, for the assessment of the working capacity of the employee, medical diagnosis, the provision of health or social care or treatment or the management of health or social care systems and services on the basis of Union or Member State law or pursuant to contract with a health professional and subject to the conditions and safeguards referred to in paragraph 3;*" **[13]**

As outlined in Article 9(1), processing is prohibited with regards to certain types of data, which includes data concerning health. This rule applies to the clinic management system being researched, as such such Article 9(1) prohibits the processing of this data.

However, Article 9(2) outlines some of exceptions to this rule, therefore Article 9(1) does not apply. Article 9(2)(h) outlines one such exception, which is for "the purpose of preventive or occupational medicine, … medical diagnosis … or treatment or the management of health or social care systems".**[13]** This makes a clinic exempt from this prohibition.

Encryption

The data held by the clinic must comply with Health Service Executive (HSE) encryption policy, a strict set of rules that apply to medical data and devices, as the clinic the system is aimed at is owned by the HSE. The HSE has publically released the policy for encryption, and among the rules lie the “Approved Encryption Algorithms and Protocols” . These are as following:

“*6.0 Approved Encryption Algorithms and Protocols*

6.1 Symmetric Key Encryption Algorithms

- *Triple Data Encryption Standard (3DES) (Minimum encryption key length of 168 bits)*
- *Advanced Encryption Standard (AES) (Minimum encryption key length of 256 bits)*
- *Blowfish (Minimum encryption key length of 256 bits)*

6.2 Asymmetric Key Encryption Algorithms

- *Digital Signature Standard (DSS)*
- *Rivest, Shamir & Adelman (RSA)*
- *Elliptic Curve Digital Signature Algorithm (ECDSA)*“

[14]

Mentioned in 6.1 are some symmetric key encryption algorithms, which I will take more concern with than the asymmetric key encryption algorithms used in 6.2, as they are generally computationally lighter than symmetric key encryption algorithms. Also, as the data is not intended to be shared with untrusted parties, asymmetric is relatively unnecessary.

This leaves us with 3 distinct possibilities for encrypting the data:

- Triple Data Encryption Standard (3DES)
- Advanced Encryption Standard(AES)
- Blowfish

Triple Data Encryption Standard (3DES)

3DES is based on the “old” data encryption standard DES, whose key size was too short for proper security and could easily be brute forced as early as 1998 via the EFF DES cracker **[15]**. 3DES uses cascading instances of DES (3 to be exact, hence the name) and offers relatively unbreakable (with modern day hardware) security.

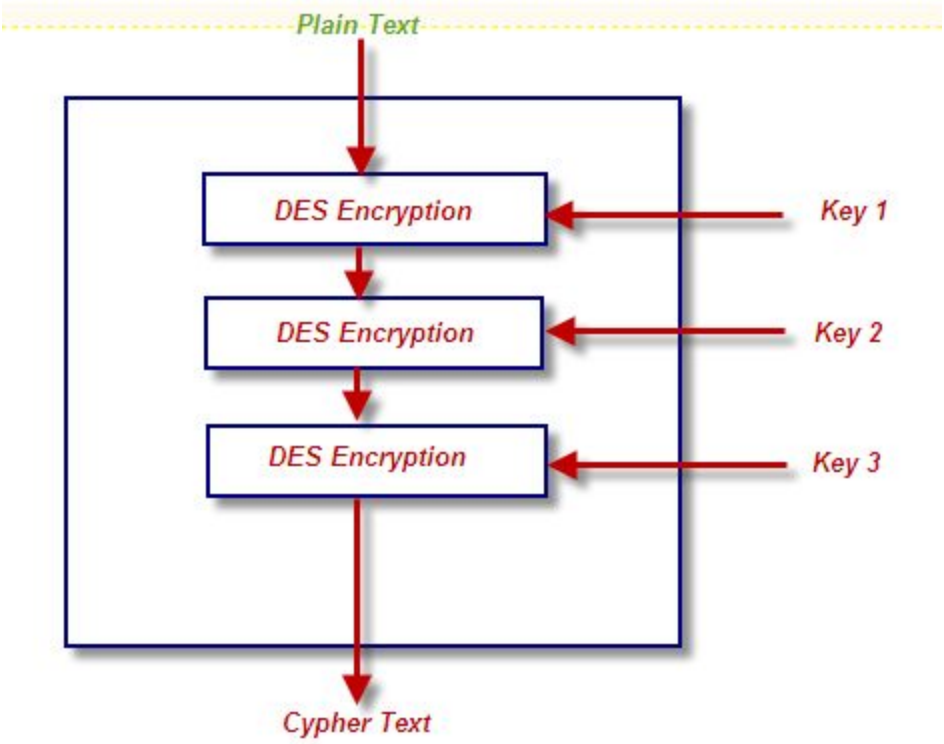


Figure: Working Of Triple DES Algorithm

www.sanjaal.com

[16]

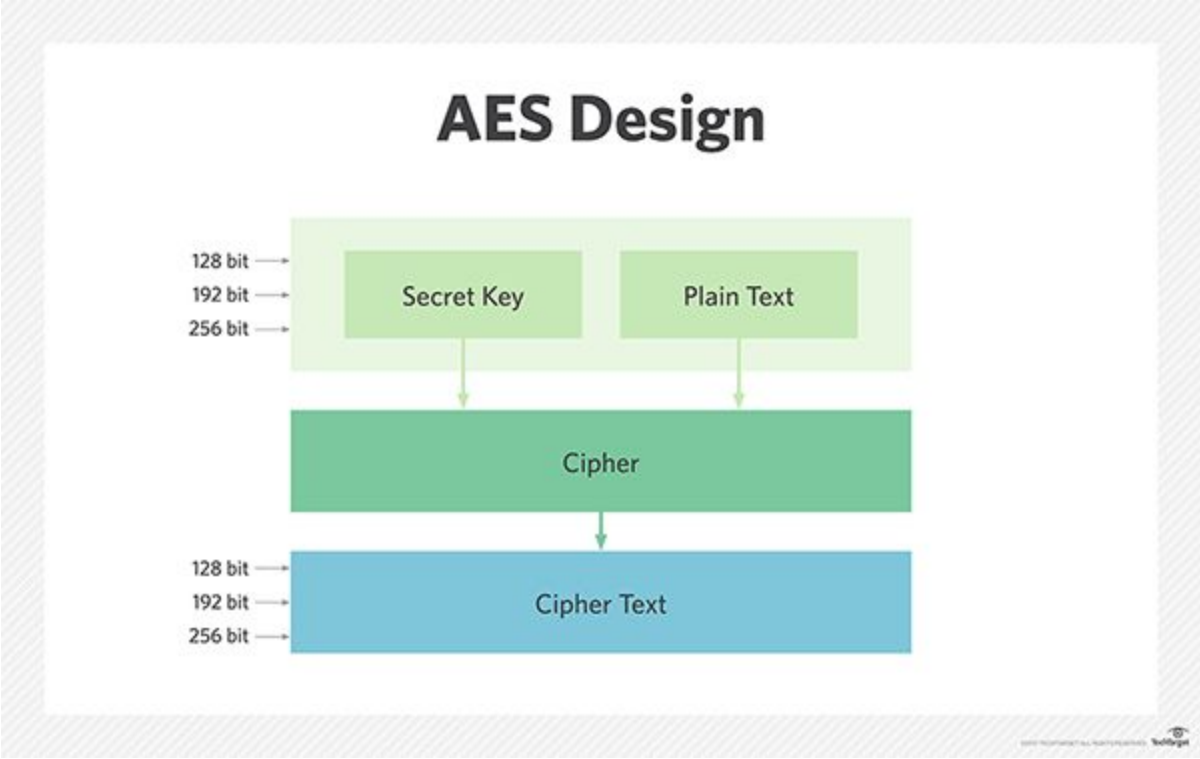
An issue with 3DES however, is that DES encryption was originally designed to encrypt hardware efficiently, however it tends to perform slowly and poorly in software implementations. Additionally, DES uses 64-bit blocks, which can cause some issues when encrypting multiple gigabytes of data.

Blowfish

Blowfish is a block cipher that is deployed in software. It uses huge keys and is accepted as secure, however much like DES and 3DES, it's block size is only 64-bits and may cause some issues. Blowfish is efficient in software, however can be platform dependent, as performance can be affected by memory handling and caching. Much like 3DES, it is considered unbreakable by modern computing standards.

Advanced Encryption Standard (AES)

AES is the successor of DES, and replaced it as the standard symmetric encryption algorithm used for US Federal organisations. It accepts keys of 128-bit, 196-bit and 256-bit with 256-bit being the safest, ie. the most unbreakable (however 128-bit is already unbreakable). It uses 128-bit blocks, a vast improvement over the 64-bit blocks of Blowfish and 3DES and is efficient in both hardware and software.



[17]

Choice of encryption standard

After carrying out research on the encryption standards approved by the HSE, it seems that AES encryption would be the most appropriate for the purposes of the clinic. The software will be required to run on a potentially varied range of hardware. Also, the system will need to be as fast as possible, as slowdowns in the system as a result of encryption will have a negative effect on the the experience of the user. The speed of the system will be heavily influenced by the speed of the encryption used, as the data will need to be encrypted on insert to the system and decrypted on read.

3DES more than meets the requirements for complexity and how difficult it is to break. However it is known to be slow in software implementations, and as the encryption must be software implemented this means it is not suitable. Also, the block size of 3DES can cause issues when encrypting large quantities of data.

Blowfish, similarly to 3DES, can handle strong encryption with ease. However it also bears another similarity to 3DES; the block size. The 64-bit block size of Blowfish can be a hindrance to the system when it comes to large quantities of data which again, would influence the performance and therefore UX of the system.

AES, similar to the aforementioned standards, is cable of “unbreakable” encryption by modern day standards, even at its minimum key length of 128-bit (256-bit is required by the HSE policies, meaning even more unbreakable). The block sized is more capable of large ata quantities also, making it much more suited than Blowfish or 3DES.

As a result of this research, the chosen encryption method is AES (more specifically AES-256).

Elixir Encryption

With the encryption method decided, research must be carried out on how to implement the encryption. There are 2 main approaches to this: Self-written code and Pre-written code. These approaches have their merits and their drawbacks, which can have ramifications on the efficiency of the system.

If the code is written entirely from scratch by me, it will take longer than pre-written code which given the timeframe of this project may be unideal. The code may not work as intended if there is not a firm understanding of the methodology behind AES encryption, and if something goes wrong, it is more difficult to troubleshoot.

Pre-written code has the benefit of working out of the box with less development time cost, and if there is a community behind it, troubleshooting issues may be easier. An additional benefit of being community supported, is that the code is more likely to be optimised and more efficient, as continuous feedback can be integrated into the code module.

From these observations, I chose the pre-written code option. I began researching potential modules, and there was one in particular that was maintained and commented

well, has plenty of community activity and has been in active development for over 2 years. It was previously mentioned by Paul Barry as a potential encryption route also: Cloak.

Cloak

Module Name	Cloak
Developer	Daniel Berkompas
Hexdocs	https://hexdocs.pm/cloak/Cloak.html
Version	0.6.2
Github	https://github.com/danielberkompas/cloak

Cloak is an Elixir module developed with a sole purpose: encrypting data. It uses AES-256 out of the box. Installation is straightforward, and offers some additional features such as transparently encrypting/decrypting data as it is read/written to/from the database. The developer actively takes suggestions, issues and bug-fixes onboard and implements them into each version of the module.

This module would fit the purpose of this system very well, and the large community associated with it may potentially come in useful in case the installation/implementation goes wrong.

References

[1]. Erlang Programming/Pattern Matching - Wikibooks, open books for an open world. 2018. *Erlang Programming/Pattern Matching - Wikibooks, open books for an open world*. [ONLINE] Available at: https://en.wikibooks.org/wiki/Erlang_Programming/Pattern_Matching. [Accessed 10 April 2018].

[2]. Medium. 2018. *Understanding Ecto Through a Phoenix Request / Response Cycle*. [ONLINE] Available at: https://medium.com/@StevenLeiva1/understanding-ecto-through-a-phoenix-request-response-cycle-65b5c10e46cb. [Accessed 10 April 2018].

[3]. Origins of Elixir programming language. 2018. *Origins of Elixir programming language*. [ONLINE] Available at: https://www.slideshare.net/pivorak/origins-of-elixir-programming-language. [Accessed 10 April 2018].

[4]. elixir-lang.github.com. 2018. *Elixir*. [ONLINE] Available at: https://elixir-lang.org/. [Accessed 10 April 2018].

[5] Jurić, S., 2015. *Elixir in Action*. Manning Publications.

[6] Phoenix. 2018. *Phoenix*. [ONLINE] Available at: http://phoenixframework.org/. [Accessed 10 April 2018].

[7] Elixir vs Ruby Showdown - Phoenix vs Rails | Littlelines. 2018. *Elixir vs Ruby Showdown - Phoenix vs Rails | Littlelines*. [ONLINE] Available at: https://littlelines.com/blog/2014/07/08/elixir-vs-ruby-showdown-phoenix-vs-rails. [Accessed 10 April 2018].

[8] GitHub. 2018. *GitHub - bignerdranch/why_elixir: Selling points for Elixir / Phoenix*. [ONLINE] Available at: https://github.com/bignerdranch/why_elixir#simplicity. [Accessed 10 April 2018].

[9] SitePoint. 2018. *An Introduction to Elixir's Ecto Library — SitePoint*. [ONLINE] Available at: https://www.sitepoint.com/introduction-to-elixirs-ecto-library/. [Accessed 10 April 2018].

[10] General Data Protection Regulation (GDPR). 2018. *Art. 1 GDPR – Subject-matter and objectives | General Data Protection Regulation (GDPR)*. [ONLINE] Available at: https://gdpr-info.eu/art-1-gdpr/. [Accessed 10 April 2018].

[11] General Data Protection Regulation (GDPR). 2018. *Art. 25 GDPR – Data protection by design and by default | General Data Protection Regulation (GDPR)*. [ONLINE] Available at: https://gdpr-info.eu/art-25-gdpr/. [Accessed 10 April 2018].

[12] Encryption in Healthcare The Right Prescription for Maintaining Compliance with Patient Security Regulations. (2018). [ebook] Apricorn. Available at: https://www.apricorn.com/media/pressreleases/file/h/e/healthcare_data_encryption_whitepaper.pdf [Accessed 10 Apr. 2018].

[13] General Data Protection Regulation (GDPR). 2018. *Art. 9 GDPR – Processing of special categories of personal data | General Data Protection Regulation (GDPR)*. [ONLINE] Available at: https://gdpr-info.eu/art-9-gdpr/. [Accessed 10 April 2018].

[14] Information Security Project Board (ISPB) on behalf of the HSE., ., 2018. .. HSE Encryption Policy,[Online].-,-.Available at:https://www.hse.ie/eng/services/publications/pp/ict/encryption-policy.pdf [Accessed 16 April 2018].

[15] EFF DES cracker - Wikipedia. 2018. *EFF DES cracker - Wikipedia*. [ONLINE] Available at: https://en.wikipedia.org/wiki/EFF_DES_cracker. [Accessed 10 April 2018].

[16] Java Tutorial Triple DES Encryption – Java and Android Programming Blog @ iCodeJava.com. 2018. *Java Tutorial Triple DES Encryption – Java and Android Programming Blog @ iCodeJava.com*. [ONLINE] Available at: http://blog.icodejava.com/tag/java-tutorial-triple-des-encryption/. [Accessed 10 April 2018].

[17] TheBestVPN.com. 2018. *What is Advanced Encryption Standard (AES): Beginner's Guide*. [ONLINE] Available at: https://thebestvpn.com/advanced-encryption-standard-aes/. [Accessed 10 April 2018].